

Final Design Report

μLAREN — Small-Scale Intelligent Vehicle Design Platform

Jay Miley		jcmiley@calpoly.edu
Evan Phillips		evphilli@calpoly.edu
Chris Grant		chgrant@calpoly.edu

Dr. Charles Birdsong, Sponsor

Mechanical Engineering Department Senior Project
California Polytechnic State University, San Luis Obispo
2016 - 2017

Statement of Disclaimer

Since this project is a result of a class assignment, it has been graded and accepted as fulfillment of the course requirements. Acceptance does not imply technical accuracy or reliability. Any use of information in this report is done at the risk of the user. These risks may include catastrophic failure of the device or infringement of patent or copyright laws. California Polytechnic State University at San Luis Obispo and its staff cannot be held liable for any use or misuse of the project.

Abstract

Intelligent Vehicle Design is a growing field with the potential to save many lives by actively minimizing the impacts of human error. Though there are many ways to research intelligent vehicle control, full-scale implementations are expensive and dangerous and computer simulations have extremely steep learning curves. Researchers and students need an accessible, adaptable, and robust development platform to rapidly create and test autonomous control algorithms. While small-scale platforms are often designed from the ground up for specific projects, this requires analysis, design, and manufacture. The goal of this project is to develop a small-scale intelligent vehicle that can be configured with physical sensors and programmed with control algorithms designed in Simulink. We will strive to make our design adaptable and reproducible through intentional design and documentation. We have completed the design to adapt a 1/7th scale remote control vehicle with a custom chassis, independently driven wheels, and a Raspberry Pi based control package. An inertial measurement unit, an ultrasonic rangefinder, and a camera will give the system real-time data about itself and its surroundings. This well-documented research platform will enable more students to get hands on experience in developing and testing intelligent vehicle systems. These students will become the next generation of vehicle safety engineers, developing the life-saving intelligent vehicle systems of the future.

Acknowledgements

We would like to thank our sponsors:

The Maxon Motor logo consists of the words "maxon motor" in a white, lowercase, sans-serif font, centered within a solid red rectangular background.

driven by precision

Mechanical Engineering Student Fee Allocation Committee (MESFAC) 2016-2017

CPCConnect 2016-2017

We would also like to thank our Project Advisor, Technical Advisor, and CNC support:

Dr. Charles Birdsong, Project Advisor

Charlie Refvem, Mechanical Engineering Graduate Student and Technical Advisor

Max Selna, CNC Support

1. Introduction.....	1
2. Background	1
2.1 ESV Competition.....	2
2.2 Existing Technologies.....	3
2.3 Similar Products.....	6
2.4 RC Platforms.....	8
2.5 Microcontroller Platforms.....	8
2.6 Technical Challenges	9
3. Design Requirements and Specifications	10
3.1 Customer Requirements.....	10
3.2 Specifications	10
3.3 Benchmarking Results	12
3.4 Scope.....	12
3.5 Boundary Sketch	13
4. Design Development.....	13
4.1 The Design Process.....	14
4.2 Concept Development.....	15
4.3 Comparison of concepts.....	17
4.3.1 Initial RC Car Comparison.....	17
4.3.2 Microcontroller Selection.....	17
4.3.3 Baseline Algorithms.....	18
4.3.4 Braking Method	18
4.3.5 Protective Housing.....	19
4.3.6 Center of Gravity Modification.....	19
4.3.7 Electronics Layout	19
4.4 Selected Concept.....	19
4.5 Preliminary Design Analysis	22
4.5.1 Dimensional Similitude.....	22
4.5.2 Braking Forces	24
4.5.3 Motor Selection.....	24
4.6 Concept refinement.....	24
4.6.1 Design Planning	25
5. Final Design	27
5.1 Mechanical Architecture.....	28
5.1.1 Structural Design.....	28

5.1.2 Dynamic and Similitude Properties	29
5.2 Controls Architecture.....	30
5.2.1 Steering and Drive Motors	31
5.2.2 Radio Control.....	32
5.2.3 Sensors	32
5.2.4 Motherboard.....	33
5.2.5 Communication Scheme	34
5.3 Firmware Architecture	34
5.3.1 Raspberry Pi	34
5.3.2 Teensy	35
5.3.3 Fault States	35
5.4 Safety Hazards	35
5.5 Maintenance & Repair	36
6. Detailed Design & Supporting Analysis.....	36
6.1 Material Selection	36
6.2 Basics of Vehicle Dynamics	36
6.2.1 Equations of Motion.....	37
6.2.2 Lateral Forces.....	37
6.2.3 Longitudinal Forces	38
6.2.4 The Understeer Gradient	39
6.2.5 Preliminary Vehicle Simulation.....	39
6.2.6 Turning Radius.....	41
6.3 Chassis Design	41
6.3.1 Similitude	41
6.3.2 Design for Stiffness.....	41
6.3.3 Chassis Mount Design	43
6.4 Shaft Coupler Design	44
6.4.1 Concept Generation.....	44
6.4.2 Failure Analysis	45
6.5 Design Analysis of load bearing members	46
6.6 Maxon Motor Curves and heat considerations	48
6.7 Understanding The Existing System FOr Future Design	49
6.8 Creating The New System	51
6.9 Software Design.....	53
7. Manufacturing.....	55
7.1 ESV Systems Demo.....	56
7.2 Sourcing	56

7.3 Budget and Bill of materials	56
7.4 Manufacturing.....	57
7.4.1 Motor Housing	58
7.4.2 Shaft Couplers.....	59
7.4.3 Steering Posts.....	61
7.4.4 Suspension System Mounts	61
7.4.5 Chassis	62
7.4.6 Chassis Mounts	63
7.4.7 Sensor Mounts.....	63
7.4.8 Motherboard.....	64
7.4.9 Wiring	66
7.5 Assembly.....	68
7.6 Hardware Safety considerations	69
7.7 Software breakdown	70
7.7.1 Teensy Software File Layout	70
7.7.2 Teensy Main Loop Brief.....	71
7.7.3 Teensy's Pertinent Variables.....	72
7.7.4 CAN User Guide.....	73
7.8 Recommendations for Continued software development.....	82
7.8.1 Simulink.....	82
7.8.2 Adding a Sensor	83
7.8.3 Teensy Loop.....	86
8. Design Verification.....	86
8.1 Testing Plan	86
8.2 Build Quality Evaluation	86
8.2.1 Modifications	86
8.3 Quantitative Testing.....	88
8.3.1 Center of Gravity Position	88
8.3.2 Steering Model Verification & Repeatability	89
8.4 Possible Improvements	93
8.4.1 Geometry & Structure.....	93
8.4.2 Circuit Design	94
8.4.3 Sensors & System Architecture	94
9. Reproduction	95
9.1 Motherboard.....	95
9.2 Mechanical System	96
9.3 Overall System Cost	96

10. References.....	97
----------------------------	-----------

Figures

Figure 1: Schematic of adaptive cruise control.....	4
Figure 2: Braking procedures for oversteer and understeer corrections	6
Figure 3: CSU Northridge Autonomous vehicle and test course. Students develop and program the vehicle to navigate through a course. The product competes at the International Ground Vehicle Competition every year.....	6
Figure 4: Cal Poly ME senior project flowchart.....	14
Figure 5: Braking Subsystem Example.....	16
Figure 6: Brainstorming and prototyping examples for (a) brain sketching activity and (b) braking method test stand.....	17
Figure 7: Design concept sketches for (a) 2 motor power system and (b) rack with weights mounted on wire cage.	19
Figure 8: Final Concept Sketch of Modified RC Car. Not Shown: Wire cage for component protection and weight rack for CG modification	21
Figure 9: Design challenges with the specific subsystems that they stem from.....	22
Figure 10: Early planning timeline of milestones through ESV International Notification	25
Figure 11: Solid models of (a) Stock Traxxas Slash (2WD) chassis and (b) early mockup of custom replacement chassis.....	25
Figure 12: Stock front drivetrain of Traxxas Slash – tan components will be replaced with custom parts that accommodate the Maxon motors.	26
Figure 13: Comparison of stock (top) and custom rear drivetrain mounts. Note that the rear suspension uprights are not pictured on the custom mount.....	27
Figure 14: Isometric view of final design with approximate shapes for electronic components and cabling omitted.	28
Figure 15: Existing Traxxas steering and suspension system integration with designed components.....	29
Figure 16: Hardware abstraction design of the SSIVD platform.....	31
Figure 17: Maxon EPOS4 Compact 50/5 Can Positioning controller with manufacturer supplied connector board.	31
Figure 18: Maxbotix URF Detection characteristics for various object types.....	32
Figure 19: Teensy 3.6 microcontroller pinout and pin capabilities. The Teensy 3.5 is physically similar, and the Teensy 3.2 is truncated to pins 0 through 23.	33
Figure 20: Oscilloscope Capture of the 3.3V and 5V Compatible CAN Bus	34
Figure 21: Schematic of Bicycle Model	37
Figure 22: Lateral tire force as a function of slip angle for Traxxas slash tires. The curve is generated using Pajacka's magic formula and data referenced from Thomas Fitzgerald.	38

Figure 23: Steering response for vehicle simulation. Top plot shows the input steering angle and the bottom plot shows the lateral acceleration.	40
Figure 24: Vehicle stick model used to predict overall vehicle compliance	42
Figure 25: Normalized torsional stiffness of the entire vehicle as a function of chassis thickness	43
Figure 26: Exploded view of shaft coupler converging on the u-joint ball.	45
Figure 27: Shaft coupler FEA results for unmodified (left) and modified coupler designs (right).	46
Figure 28: Hole tear-out path and critical dimensions	47
Figure 29: Maxon performance curves, steady state and acceleration system curves	49
Figure 30: Existing Traxxas Slash Flow of Control	50
Figure 31: Signal from receiver to ESC at neutral throttle (a), 15% duty cycle, reverse (b), 10% duty cycle, and forward (c), 20% duty cycle	51
Figure 32: Completed Component Selection and electrical layout, with voltages and component information	52
Figure 33: Design of the new system from a communication standpoint	53
Figure 34: Teensy Task Decomposition	54
Figure 35: Raspberry Pi Software	55
Figure 36: Planning timeline from CDR, including critical dates, milestones and deliverables	56
Figure 37: Current budget compared to PDR expectations and total available budget	57
Figure 38: Heat setting threaded inserts into the motor housings using the soldering iron tip (a). A motor housing with most of the inserts in place (b).	58
Figure 39: Finished soft-jaws for manufacturing the custom shaft couplers	59
Figure 40: Shaft coupler half after the first operation (a). Both halves in the soft-jaws after the final operation (b)	60
Figure 41: Test fits of the first shaft coupler, completed before manufacturing the rest	60
Figure 42: Suspension mounts manufactures on the Bridgeport mill (top left) and cut with the water jet (right). Manufacturing of front suspension mounts on Bridgeport mill (bottom).	62
Figure 43: Chassis template made with the laser cutter and mounted to the chassis for hole locating (left) and 2D profiles cut on water jet for chassis and chassis mounts (right)	63
Figure 44: Wire modifications on top (a) and bottom (b) faces made to the first revision of the motherboard.	65
Figure 45: Probing the second iteration of the motherboard to ensure it is safe for integration of components and microcontrollers.	65
Figure 46: Final implementation of the motherboard, with Raspberry Pi and Teensy	66
Figure 47: Connectors and crimp pins for the motor to driver cable harness	67
Figure 48: Fully assembled motor to driver cable harness.	67

Figure 49: Open ended CAN connector, with twisted wire pair for CAN high and low.....	68
Figure 50: Motor integration on the front drivetrain.	69
Figure 51: Fully assembled vehicle	69
Figure 52: A Breakdown of a Complete CAN Frame	74
Figure 53: COB-ID Values Broken Up by Function Codes	76
Figure 54: Notable CANopen Fields	76
Figure 55: Command Codes for a Write Request Frame.....	77
Figure 56: Example of a Read Request SDO.....	78
Figure 57: Example of a Write Request SDO.....	78
Figure 58: Example of a Configured PDO	79
Figure 59: Start CAN Network Example.....	79
Figure 60: Controlword Write of Shutdown State.....	80
Figure 61: Profile Velocity Mode Selection	80
Figure 62: Controword Write of Switched-On State	80
Figure 63: Controlword Write of Operation Enabled State	81
Figure 64: Writing to Index 0x60FF (Target Velocity).....	81
Figure 65: Controlword Write with Target Velocity Enabled	82
Figure 66: Insert Initialize Function in the INITIALIZE_PERIPHERALS Case	84
Figure 67: Where to Find the SIMULINK Variable ('1' means on, '0' means off).....	84
Figure 68: Two Examples of Sending a Sensor Variable to Simulink	85
Figure 69: Example of Sending Serial Opcode and Receiving the Sensor Data	85
Figure 70. Steering linkage modification to mitigate bump steer.....	87
Figure 71. Grooves filed in servo slot to incorporate the structural ribs on the servo.....	87
Figure 72. Lifted rear end for measuring CG height(left) and static weight distribution (right).....	88
Figure 73: Inside steering angle versus the input value to our servo function.	90
Figure 74: Steering angle profile for the repeated steering profile test.	90
Figure 75: Starting point of the repeated steering profile test (a) with Chris standing at the endpoint. Marked end locations of the repeated steering profile test (b) demonstrating high repeatability across large distances.	91
Figure 76: Comparison of steering profile yaw from the bicycle model, the platform response, and filtered platform response.....	91
Figure 77: Comparison of steering profile lateral acceleration from the bicycle model, the platform response, and filtered platform response.	92
Figure 78: FFT of the raw IMU lateral acceleration data.	93

Tables

Table 1: RC Car common feature options	8
Table 2: Microcontrollers and critical specifications.....	9
Table 3: List of engineering specifications and tolerances	11
Table 4: System level concept descriptions.	20
Table 5: Results of dimensional similitude analysis, with RC car for comparison	23
Table 6: Comparison between ideal dimensionally scaled model and Traxxas Slash.....	23
Table 7. Estimated final design vehicle parameters.....	30
Table 8. Pajecka's Magic formula coefficients for Traxxas RC car tires.	38
Table 9. Results from hole tear out calculations for custom components.	48
Table 10: Tooling required to manufacture the custom components.	58
Table 11: Serial Connection Sequences.....	Error! Bookmark not defined.
Table 12. Position of center of gravity with propagated resolution uncertainty.....	89
Table 13. Custom parts and suggested manufacturing methods.....	96
Table 14. Reproduction costs for future SSIVD platforms.....	97

1. INTRODUCTION

Intelligent Vehicle Design is a growing field with the potential to save many lives. It enables vehicles to adapt and react to situations that the driver may not notice, resulting in a safer, more efficient traffic flow. Researchers and students need an accessible, adaptable, and robust development platform, which does not currently exist in an accessible format. Presently, research is completed in several ways. Modified full scale vehicles are used but are expensive and dangerous. Simulation can be used for preliminary research, but requires both mechanical and extensive software knowledge. While small scale platforms are often used, they require significant start-up design. A well-documented, programmable and modular computer driven small-scale vehicle will enable rapid potentially life-saving advancements.

The goal of this project is to develop a small scale intelligent vehicle that can be configured with physical sensors and programmed with control algorithms designed in Simulink. We will enter the final design to compete internationally in the International Technical Conference on the Enhanced Safety of Vehicles (ESV) Student Safety Technology Design Competition (SSTDC), hopefully bringing prestige to the university and department. Additionally, the design, build, and testing of this vehicle will support creation of kits for an intelligent vehicle controls course being designed by Dr. Charles Birdsong for the Mechanical Engineering Department at Cal Poly. The new course will allow students with limited programming knowledge to gain practical experience with developing control algorithms for autonomous vehicle functions.

Cal Poly will provide an open system that can be used by other research and educational institutions to easily participate in intelligent vehicle design research.

2. BACKGROUND

To develop this project well, we must understand the implications and impact of our project, similar technologies, and context of our industry. Having this context will allow us to create a more useful, pertinent, and thoughtful product.

With the increasing presence of autonomous technologies in the automotive industry, there is an inherent need to develop testing methods for the control systems that will improve vehicle safety. New systems are being generated at an impressive pace, which means that test methods need to be established. In Ann Arbor Michigan, the UMTRI Safety Pilot Program has invested \$20 million into creating a test track for vehicle to vehicle connection experiments [1]. Projects such as this are extremely valuable in pushing the envelope regarding autonomous vehicle technologies, but are inaccessible to smaller institutions. Universities across the United States contain the country's next generation of intelligent vehicle researcher's and engineers. A cheaper and more practical alternative for autonomous vehicle research is through small scale models. Although small scale vehicles are largely thought as inferior in terms of similitude, a prototyping platform

would provide valuable experience to engineering students who can bring knowledge and new ideas to the automotive industry.

2.1 ESV COMPETITION

The SSTDC is a competition that challenges students to conceive, design, and test cutting edge vehicle safety technologies. Competitors converge at the ESV conference to present their findings to the automotive industry. The ESV conference is a technical conference that is intended to provide a collaborative space for the leading edge of vehicle safety research. It is a collaboration between the United States' Department of Transportation & National Highway Traffic Safety Administration and 14 participating ESV member countries and governmental organizations. The 2017 conference will take place from June 5-8, 2017 in Detroit, Michigan. The first stage of the competition is a call for 300 word abstracts, due November 11, 2016. Judges will select which teams participate in each of the three regional competitions (North America, Europe, Asia-Pacific) based on those abstracts. In March 2017, prototypes / progress will be evaluated and three finalists will be selected from each regional competition to proceed to the ESV conference & international competition. At the conference, teams will set up displays and offer demonstrations to conference attendees. Each team will give both a 15-minute technical oral presentation and a 10-minute functional model demonstration to judges and other teams. After these sessions, judges will deliberate and select the winner and runner up.

Abstract submissions will be scored out of 100 as follows:

- Potential impact on safety problem being addressed (30 points)
- Originality (25 points)
- Practicability of creating a functional scale model (25 points)
- Supporting details, quality, technical depth (20 points)

Abstract submissions should specify which 'safety category' the project falls under. One such category is 'Autonomous Vehicle Issues,' which this project falls under.

Both regional and international competitions consist of a six-page report and a functional prototype demonstration. They will be scored out of 100 as follows:

- Potential impact on safety problem being addressed (40 points)
 - Did the team address a safety problem?
 - How did the team test and evaluate its system?
 - What metrics did the team use?
 - What are the results of the testing?
 - Are conclusions presented clearly?
 - What potential or expected effects will the system have on traffic safety?
- Originality (20 points)
- Functional scale model, physical presentation (20 points)
- Oral presentation (10 points)
- Supporting details, quality, thoroughness, technical depth (10 points)

Additionally, students are encouraged to include in their report: estimated safety benefits in terms of lives saved or crashes prevented and percentage of the fleet covered.

2.2 EXISTING TECHNOLOGIES

There are a wide variety of autonomous and semi-autonomous technologies that are being developed as well as ones that are already present in the automotive industry today. These technologies include, but are not limited to: pedestrian detection, terrain sensing, adaptive cruise control, collision detection, motion prediction, lane assist, and a wide variety of stability control systems. All of these and more aim to improve the driving experience and decrease the inherent risk that comes with traveling the roadways.

Pedestrians account for roughly 14% of the total fatalities per year in US traffic accidents [2]. In order to avoid pedestrian fatalities, extensive research is being devoted to human detection software that enables collision avoidance. This software requires in depth algorithms to assess and analyze very complicated and sometimes noisy data. One method of pedestrian detection is through computer vision. Lie Guo et al. uses a complicated camshaft algorithm to detect color probabilities at the users' torso, and processes data with a Kalman filter [3]. Another method of pedestrian detection is by using recognized shapes to analyze the contours through a camera. These shapes can allow a controller to accurately predict future movements that are common among pedestrians [4]. There are a significant number of already discovered methods for detecting pedestrians, predicting movement, and avoiding collisions, as well as many that are yet to be developed. A small scale vehicle platform could provide a means for developing pedestrian detection and avoidance algorithms.

Although terrain sensing abilities is not necessary for urban vehicles that travel on paved roadways, development in this area could lead to more adaptable and safe off-road vehicle. Typically, terrain sensing is an area of research for military or government operations, but could generate a significant amount of interest among students. Furthermore, vehicles that can accurately predict and adjust to a changing terrain would decrease the likelihood of dangerous rollover or loss of traction on dirt roads. An adapted off-road small scale vehicle could implement traction control systems that allow vehicle to better adapt to changing terrain. The fundamentals of traction control systems are discussed in more detail below.

A more applicable area for roadway safety research is adaptive cruise control. Adaptive cruise control is a highway vehicle feature that adapts an automobile's speed based upon the traffic environment. Typical systems use radar to determine the distance between two vehicles, and then changes the following vehicle velocity based upon a relative safe distance. Figure 1 depicts the basic principle behind adaptive cruise control systems [5]. These systems are an example of semi-autonomous vehicle technology that could drastically reduce the amount of accidents due to distracted driving or carelessness.

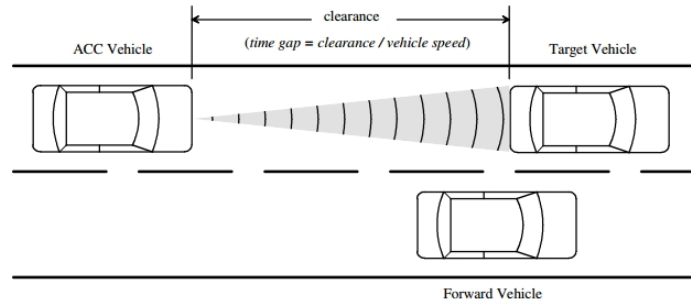


Figure 1: Schematic of adaptive cruise control

Collision detection and avoidance is one of the most heavily researched autonomous vehicle technology, but it is also very difficult to develop algorithms than can safely avoid obstacles. The safest option is to simply brake to avoid a collision, but more research is being done for alternatives. When traveling at high speeds, swerving to avoid an object can potentially be very dangerous, as this could cause rollover or potentially even collision with other undetected objects. As a result, a system is needed to detect dangerous obstacle in the path of the vehicle and monitor vehicle angular and linear accelerations. The system also needs to be careful to not interfere to much with a driver's natural abilities. A system too invasive will take away a driver's confidence and potentially have adverse effects, including the car seizing control from the driver at an inopportune time [6].

A master's thesis at Cal Poly by Thomas Stevens investigated the effects that a collision avoidance system has on a driver. The key to developing an effective collision avoidance system is that the software should not be too overpowering as to make the driver feel like they are no longer in control. It should be a seamless integration that only interferes when absolutely necessary. Drivers need to be willing to embrace an active safety system. Fitzgerald's vehicle platform was designed to assess how to integrate a semi-autonomous driving system, and found that there needs to be a balance between lightly tuned interferences and more sensitive ones [7]. This balance largely depends on the driver and their tendencies. Developing a research platform can drastically increase the rate at which useful information can be gathered from the driving population.

When considering vehicle to vehicle collisions, extensive research must go into motion prediction models that govern the control algorithms. There are a variety of physics, maneuver, and interaction aware models [8]. The numerous research possibilities presented by the differing motion prediction models alone is enough to suggest the need of a small scale platform. Developing a model that accurately assesses risk and predicts future vehicle trajectories is the framework to a successful collision avoidance system.

Electronic Stability Control (ESC) is an intelligent vehicle feature that works to reduce the loss of traction when cornering or swerving. The Society of Automotive Engineers (SAE) defines an ESC system as a car that does the following: [9]

- Is computer controlled and the computer contains a closed-loop algorithm designed to limit understeer and oversteer of the vehicle
- Has a means to determine vehicle yaw velocity and side slip
- Has a means to monitor driver steering input

- Has a means of applying and adjusting the vehicle brakes to induce correcting yaw torques to the vehicle
- Is operational over the full speed range of the vehicle (except below a low-speed threshold where loss of control is unlikely)

The system works by engaging specific brakes to help direct the vehicle in the direction intended. There are a wide variety of active stability and traction systems operating in modern vehicles. For example, BMW developed a Dynamic Traction Control (DTC) system that actually allows slip in certain situations. DTC claims to allow a more ‘sporty’ drive by permitting some small levels of slip at the tires that increase the vehicles ability to corner at a faster rate. Although this is not necessarily a safety system, it works in conjunction with the Dynamic Stability Control (DSC) system to keep the vehicle on the roadway [10]. Electronic systems such as this all monitor several kinematic and kinetic aspects. The angular accelerations of the vehicle—pitch, roll and yaw are all determined by an IMU that sends signals back to the car’s computer. The computer determines what the driver is intending to do, and adjusts the torque through the drivetrain. A stability control system must be able to work in tandem with collision avoidance systems in order to prevent potentially dangerous maneuvers that may send a vehicle off the roadway.

In order to determine the necessary requirements of a small scale vehicle platform used for testing a traction or stability control system, it is important to understand how these systems work at a fundamental level. Traction systems simply work by limiting the slippage at each tire/ground interface: Tire slippage, λ , can be defined as the ratio of the relative velocity between the tire road interface to the absolute velocity of the vehicle:

$$\lambda = \frac{(\omega_w * r_w - V_v)}{V_v}, \quad (1)$$

where ω_w is the angular velocity of the wheel, r_w is the radius of the wheel, and V_v is the absolute velocity of the wheel [11]. On a scale vehicle platform, the angular velocity can be measured by a tachometer mounted at the inside of each wheel, and the absolute velocity can be obtained by integrating the IMU longitudinal acceleration data. A control system would work to minimize the slip ratio by adjusting the output torque from the motor shaft and the braking force applied to each wheel. Therefore, a more effective vehicle platform would allow for independent torque adjustment at each wheel.

Furthermore, torque adjustments for electronic stability control are dependent upon the type of motion the vehicle intends to make. Figure 2, from the Insurance Institute for Highway Safety [12] shows how individual braking systems can be used to correct oversteer and understeer in a vehicle. The basic principle is to control the yaw rate by applying a braking torque to one of the wheels. This load is transferred through the tire and into the ground to create a restoring moment about the vehicle’s central axis.

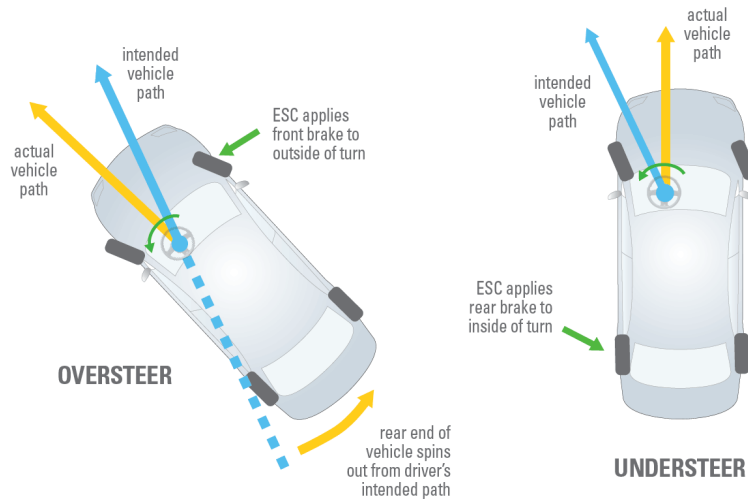


Figure 2: Braking procedures for oversteer and understeer corrections

2.3 SIMILAR PRODUCTS

The typical benefactors of a small scale vehicle platform are engineering instructors and students who are attempting to develop and test control algorithms associated with autonomous vehicle technologies. Very few courses at universities in the United States are offered that present students with the chance to experience autonomous vehicle technologies in a "hands on" environment. CSU Northridge currently offers a two semester, capstone style course for intelligent ground vehicles [13]. The course specifically focuses on the navigation of autonomous vehicles, where students design the vehicle from start to finish. Figure 3 shows students at CSU Northridge testing their vehicle's navigation skills.



Figure 3: CSU Northridge Autonomous vehicle and test course. Students develop and program the vehicle to navigate through a course. The product competes at the International Ground Vehicle Competition every year.

Testing control systems on full-scale models would be expensive and time consuming to implement. A smaller test platform would allow for rapid testing, as well as utilize the fresh and creative minds of undergraduate and graduate students.

Turkish researchers developed a small scale prototyping platform for vehicle dynamics. They modified a LOSI brand electric RC car by adapting it with an I/O board and IMU. The I/O board relays data from the IMU via Bluetooth to MATLAB-Simulink, where the computer operates as the controller. The researchers then tested and developed an anti-lock brake safety system, as well as a roll over prevention system. The ABS system on the model used a pulse brake operating at 10 Hz to simulate an actual ABS system. A weight was also placed on top of the vehicle to raise the center of gravity [14].

Advancements in technology suggest that fully automated vehicles are a very realistic possibility in the future. Research is already being done to model computer governed highways. Manh la et al. developed a small scale research platform to investigate potential fully autonomous transportation systems [15]. The platform consists of a driving arena, an indoor localization system, automated RC cars, and roadside monitoring facilities. The small scale remote controlled vehicles used in the model were programmed to follow a pre-determined trajectory, where they were tracked by the localization system. The algorithm used to control the vehicles trajectory was the primary parameter under investigation. Future research problems were also presented that could be investigated by this kind of research platform. The platform was found to be very effective at investigating potential intelligent transportation algorithms. Small scale research can be very useful when investigating and developing intelligent vehicles.

With the growing presence of autonomous technology in the automotive industry, more vehicles are becoming equipped with advanced safety features. These systems are governed by complex algorithms that cannot be verified until a test or simulation is conducted. The testing process can be very tedious, which slows down the development of potentially innovative control algorithms associated with autonomous vehicle safety. Small scale research platforms naturally increase the rate at which programs can be evaluated in a realistic environment.

An article on Hackaday.com [16] describes a project for a master's dissertation. The student took a 1/10th scale RC car and completely removed the drivetrain, replacing it with a custom implementation of a brushless out-runner motor on each wheel. Combining the unique drivetrain with hall effect sensors for speed feedback, an IMU, and a Simulink programmable microcontroller, the student produced a rapidly adjustable controls system. At the time the article was written, the student had implemented front and rear power offsets as well as virtual differentials that could transition between locked, open, and overdriven. The student commented that they planned to implement torque vectoring – a technology useful in electronic stability control. Between the Simulink compatibility, IMU, and independently drive wheels, this project's capabilities are probably the closest to our needs, though it lacks open documentation and thus does not fulfill the project need.

In addition to physical prototyping platforms, the use of computer simulation for vehicular dynamics and control systems is widespread [7]. These simulation systems can be extremely robust, with the ability to monitor and export up to 800 variables. Unfortunately, the available systems are so versatile and full-featured at the expense of usability. The resultant learning curve is very steep, preventing rapid development of multiple algorithms. Computer simulation is often used as a verification method for potential algorithms before bringing the control system to a full scale vehicle test platform. Despite the accuracy and importance of simulation and full scale vehicle testing, small-scale testing remains extremely relevant as an avenue for rapid, accessible innovation.

2.4 RC PLATFORMS

For the purposes of this project, RC cars can be divided into two classes – hobby grade and toy grade. Hobby grade RC cars tend to be more fully featured – including powerful motors, suspension, spare parts, and more. Toy grade tend to be less expensive but less robust. Table 1 below contains various feature ranges that RC cars may have.

Table 1: RC Car common feature options

Vehicle Scale	1/18 th to 1/7 th
Power Source	Electric / Gas / Nitro
Motor Type	Brushed Electric / Brushless Electric / Gas Piston
Driven Wheels	4WD / 2WD
Drivetrain	Single shaft driven / Differential driven
Suspension	None / Full independent suspension / Independent front
Chassis Materials	Plastic / Metal / Mixed
Drivetrain Materials	Plastic / Metal / Mixed
Reparability	Kit style car / Spare parts available / No sourceable parts

Several things will narrow the scope of cars that we will look at. To maximize dynamic similitude between our platform and a real car, suspension and a differential are strongly desired. Selecting an electric car will serve the dual purpose of improving safety and providing a power source to our electronic system. Since a secondary goal of this platform is to be used in a course (for several years), the kit should be repairable with easily source-able parts. In the preliminary design phase, a decision matrix will be used to compare several different RC car options and support the final decision.

2.5 MICROCONTROLLER PLATFORMS

This project has several requirements that affect the selection of a microcontroller. A critical requirement is the usage of Simulink for programming to provide mechanical engineers with a familiar environment to explore intelligent vehicle control. On a deeper level, selection of a microcontroller with extensive documentation and user base will enable the possibility of more advanced projects. Since the goal is to integrate sensors and in turn control the car, the microcontroller must have reasonable input / output support. Finally, the microcontroller must have reasonable storage and processing power for extensive control algorithms. Table 2 below contains several popular microcontrollers and project critical specifications.

Table 2: Microcontrollers and critical specifications

Arduino Uno [17]	Simulink Compatibility	Yes
	I/O Ports	14 Digital (6 PWM), 6 Analog
	Clock Speed / Program Memory / RAM	16Mhz / 32KB / 2KB
	Operating System / Firmware	Arduino Bootloader, C variant
	Cost	\$25
Arduino Mega [18]	Simulink Compatibility	Yes
	I/O Ports	54 Digital (15 PWM), 16 Analog
	Clock Speed / Program Memory / RAM	16Mhz / 256KB / 8KB
	Operating System / Firmware	Arduino Bootloader, C variant
	Cost	\$46
Raspberry Pi 3B [19]	Simulink Compatibility	Yes, incl. computer vision
	I/O Ports	26 GPIO / 4 USB / Camera
	Clock Speed / Program Memory / RAM	1.2GHz / SD card / 1GB
	Operating System / Firmware	Linux Variants
	Cost	\$35
Raspberry Pi 1 A+ [20]	Simulink Compatibility	Yes, incl. computer vision
	I/O Ports	26 GPIO / 1 USB / Camera
	Clock Speed / Program Memory / RAM	700MHz / SD card / 512MB
	Operating System / Firmware	Linux Variants
	Cost	\$25
BeagleBone Black [21]	Simulink Compatibility	Yes, thru embedded coder, incl. computer vision
	I/O Ports	69 GPIO / 1 USB
	Clock Speed / Program Memory / RAM	1GHz / 4GB / 512MB
	Operating System / Firmware	Linux Variants
	Cost	\$55

Technical specifications were retrieved from the referenced webpages, save for Simulink compatibility which was retrieved directly from MathWorks by searching the Hardware Support webpage [22].

2.6 TECHNICAL CHALLENGES

Although it is near impossible to plan for every problem a project will have, thinking through some possible challenges in the project provides a great start. The first challenges involve the implementation of the technology. This includes the compatibility of Simulink, ease of extra module integration, and response time of the vehicle to user and environmental inputs.

Compatibility with Simulink is essential to this project. Mechanical engineers at Cal Poly are exposed to Simulink in their controls course, and it is used across the industry for controls and automotive applications. To use this to our advantage we must know in advance that the microcontroller we choose can load code exported from Simulink. The next challenge is to allow the user to have confidence that any additional sensors will function properly on the vehicle platform. The platform must be powerful enough to run the designed algorithms, and it must be fast enough to respond to new inputs. A product that works perfectly but takes ten times longer to use will not be seen as an effective prototyping platform, so it must be powerful and adaptable.

The next area of challenges deals with the physical properties of the platform. There are many details that must be incorporated to create a small-scale vehicle with reasonable similitude to full scale vehicles. In our design for similitude we will consider factors including weight, center of gravity, friction, acceleration, braking, turning radius, and many more. Some factors that will challenge the physical properties of the system include the weight distribution of the vehicle and keeping the system rugged and resilient. Though similitude will be considered throughout the design process, it is not the primary goal – the critical path is creating an accessible, adaptable development platform.

3. DESIGN REQUIREMENTS AND SPECIFICATIONS

The primary goal of this project is to develop a 10th scale research platform that will allow students to develop new and innovating intelligent vehicle safety systems. To accomplish this, the model must be adaptable, robust, easily maintained, and easily reproduced.

3.1 CUSTOMER REQUIREMENTS

Professor Birdsong is in the process of developing a controls course that will be focused around a platform similar to ours. We will focus on developing a project for the ESV competition but design it in such a way that it can be adapted for the course. After meeting with him and discussing the project goals and scope, the following requirements were developed. The vehicle must be:

- Physically robust so that it can absorb repeated impacts
- Easy to program for undergraduate and graduate level mechanical engineering students
- Adaptable to new sensor modules
- Compact for transportation and storage
- Low cost so that a number of models can be reproduced
- Display as close to realistic vehicle dynamics as possible
- Operate via autonomous and semi-autonomous control
- Easily maintained and reproduced

3.2 SPECIFICATIONS

Through Dr. Birdsong's input and researching the ESV design competition, a set of engineering specifications were developed. Though low-cost is a customer requirement, we did not include it as an engineering specification because while it is a consideration, we are making this model specifically for the ESV competition and hope to make an impressive 'floor model.' The quality function deployment model helped to produce a set of engineering specifications related to these requirements. The QFD, found in Appendix A, also helped to assess where we need to fall relative to other similar products.

Table 3: List of engineering specifications and tolerances

Spec. #	Parameter Description	Requirement or Target (units)	Tolerance	Risk	Compliance
1	Withstands impacts	---	---	H	T
2	Latency	50 ms	Max	M	I,A
3	Vehicle Size	1/10th Scale	---	L	A,I
4	Vehicle Acceleration	2.5 ft/s ²	Max	M	T,A
5	Turning Radius	3.8 ft	+/- 0.5 ft	L	T
6	Vehicle Speed	15 ft/s	Max	L	T
7	CG Height	2 inches	Min	L	A
8	Works with Simulink	Yes	---	H	I
9	Suspension	Yes	---	M	I
10	Digital I/O Ports	3 Modules	Min	M	I
11	Independently powered wheels	2	Min	M	I
12	Tetherless	Yes	---	M	I
13	Autonomous / Hybrid Control	Yes	---	H	I
15	Battery Life	3 hours	Min	L	A,T
16	Protected Electronics	Yes	---	M	I,T

Specific requirements were developed through brief calculations to maintain similitude between our tenth scale model and a full scale model. Latency was based upon an average human reaction time to visual stimuli of 200 ms [23]. We decided that our system should perform at least 4x faster, reacting to stimuli faster and thus being able to catch things that a human may not. Maximum acceleration was based upon a full scale zero to sixty MPH time of four seconds. The turning radius was based upon a standard vehicle turning radius. Top vehicle speed assumes a full scale speed of 100 MPH. The minimum height of the center of gravity assumed a 6-inch scale wheel base (reasonable for RC cars), where rollover occurred when the normal force on the outside tire was reduced to zero and the vehicle was turning its minimum radius at maximum speed (note: this is a bare minimum case). The battery must be able to last the duration of a 3-hour lab period, with moderate use. Risk assessment was defined as how critical meeting each specification is to the overall success of the project. Specifications that were assigned a high risk (Withstands impacts, Works with Simulink, Digital I/O ports, User controller) are deemed to be absolutely necessary to make the vehicle platform perform its intended function and adapt to future needs. Parameters pertaining to similitude (Size, acceleration, speed, independently powered wheels) were medium risk. These parameters would not strictly determine the effectiveness of the platform, but still need to be met as best as possible. Furthermore, low risk assessment was given to specifications that were simply desirable, but not critical.

Compliance methods primarily fell under inspection when the specification was simply a Yes/No answer. However, similitude could be analytically determined with the Buckingham-Pi theorem. The vehicle's durability could be simply tested by subjecting it to extreme cases (i.e. top speed into a cinderblock), and the exposure of fragile electronics during roll-over could be assessed through a similar test. Most other specifications depend on the type of vehicle platform that is purchased, so an extensive amount of research will need to be dedicated to choosing the correct RC car.

3.3 BENCHMARKING RESULTS

Benchmarking with similar products is a great way to analyze the field of competition for their weaknesses and strengths. To help us find what we need to focus on, we hypothesized a ‘current product’ that would meet all the specifications outlined in the QFD. Similar products were not designed to explicitly meet our expectations; therefore, it is expected that our vehicle will outperform the competition in this form of benchmarking. From our benchmarking section of the QFD, we realized that the biggest challenges were in the low cost, realistic, and manufacturability customer requirements. Interestingly enough, these three columns have a deep relation with each other. Due to the lack of acceptable vehicle platforms, a user must create his/her own vehicle to live up to their own specifications. If a user aims to have a quality and realistic machine, many modifications and enhancements must be made to bring the product up to par, leaving the user spending more money and time than they would like. In order to beat the competition, our project will need to be especially careful with these three customer requirements so it can maximize the score in these areas.

Customer requirements where we were strong included being portable, and having hybrid autonomous control capabilities. As far as being portable goes, this requirement is the easiest for an RC car to achieve given its scale and function. The hybrid autonomous control is a bit more of an interesting design challenge – the car needs to be able to adjust user input for minor corrections and fully override user inputs in critical situations. With a microcontroller to take input from the RC receiver, it is a manageable challenge to adjust or ignore this input.

Most of the benchmarked products seemed to rate in the middle ground for each category. In fact, only the Independent Wheel Drive project seemed to reach the maximum, as well as minimum, allotted point total in any category. The fact that most projects are found in the middle-ground seem to represent that those projects have different goals to meet. These findings reinforce the fact that there is no product that fills the role of our proposed product. Some projects barely cover all the requirements, and others find themselves in all-or-nothing scenarios, but our product should do well in all required areas.

Specifications developed in the QFD include: withstand impacts without damage, visible latency, vehicle size, top vehicle speed, works with Simulink, suspension, digital I/O ports, tetherless, independently controlled wheels, and user controller. A vehicle that can withstand impacts is considered to be able to survive crashes and rollovers at top vehicle speed. Visible latency is dependent upon the capabilities of the microcontroller and its ability to run basic algorithms without lag. The top vehicle speed is a parameter to maintain similitude and is calculated as a full scale vehicle traveling at 100 MPH. Based upon our sponsor discussions, it is imperative that the microcontroller is compatible with Simulink and the algorithms can be developed in this program. Digital I/O ports are necessary to enable adaptations to the platform. A tetherless vehicle is also necessary for ease of use and more portability. A vehicle platform capable of testing electronic stability and traction control must have at least two independently controlled wheels for braking and accelerating. Finally, a user controller is necessary for the hybrid manual/autonomous interface. The list of specifications was assessed, modified and expanded to create our final specifications table.

3.4 SCOPE

The main scope of this project is simple and straightforward: to create an intelligent vehicle platform to participate in and win the 2017 ESV competition. Most projects entered in this contest are solving a

straightforward and tangible safety problem that can be applied to bigger projects in a reasonable amount of time. Our finished product will instead build upon the promise of what is to come. This will require convincing the judges that the platform that we made is more important for future of human safety than the solution another team made to immediately help save lives.

The other goal we hope to reach is to inform the design and construction of eight more platforms for Dr. Birdsong's new course. We initially believed this goal to be the main focus but soon realized a platform for the ESV competition could easily be modified for this course.

3.5 BOUNDARY SKETCH

Located in Appendix B, the Boundary Sketch gives a visual image of what the project will be held responsible for. We included the purpose of our project in the sketch as well to clarify our objectives even further. Creating an exact model for the upcoming design class is not within our scope; however, we make a design from which a class model can be established while maintaining our focus on competing in the ESV competition. The aim of this project is not to design new, complicated control algorithms. Instead, we will only pick a few basic algorithms to show off the ability and the platform foundation for which the former idea is possible.

The scope also includes advanced features such as manual/remote capabilities, an adjustable center of gravity, and realistic vehicle dynamics. Dynamic components such as the manual/remote capability will allow future algorithms the opportunity to implement computer driven adjustments to each motor. The platform as a whole will be powered by a rechargeable battery. An adjustable center of gravity will allow for flexible vehicle dynamics when interested students need to experiment with how a particular algorithm interacts with the center of gravity. We expect to strive towards the best response time possible for the system by limiting the latency wherever possible for the components used in the platform.

Other key scope features include durability, accessibility, manufacturability, and the connection between the platform and a Simulink-capable computer. Durability corresponds to our need to make our platform robust while accessibility refers to the need to make components on the platform convenient for modifications. Manufacturability represents an extension of the purpose and refers to the idea of creating a plan for the class model that can be easily built. The connection to download Simulink code to the platform needs to be robust, fast, and as convenient as we can make it.

4. DESIGN DEVELOPMENT

Following a structured design process, we have identified customer requirements and specifications, developed various concepts, and completed initial analysis and design for the selected concept.

4.1 THE DESIGN PROCESS

To ensure an excellent final product, a structured design process will be used. Figure 4 shows the Cal Poly Mechanical Engineering senior project process, as outlined in the *Student Success Guide* [24]. This flowchart indicates required processes and deliverables that make up the senior design project.

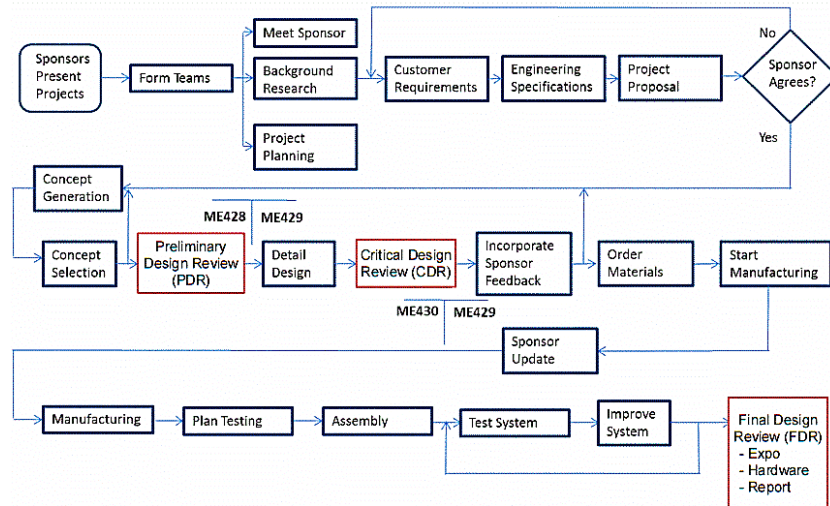


Figure 4: Cal Poly ME senior project flowchart.

We first met with the sponsor to discuss the details of the project. After that we began background research on both existing systems and potential components of our solution. We concurrently worked to identify customer requirements and expand those into engineering specifications. After bringing our research, ideas, and specifications to our sponsor, we entered the brainstorming and preliminary design phase. From there, we built upon our work with detail part design and systems architecture. We have completed the mechanical and systems design, and are ready to begin construction and implementation of our design.

Going forward, we have to construct a prototype and test it against our engineering specifications and customer requirements. We will continue to document our design philosophies, lessons learned, and retrospective changes we would have made to ensure that this project is useful for those who would adapt it. If time permits, we will modify our prototype design based upon our testing and knowledge gained throughout the course of the project. Since the ESV competition is a major interest of this project, key deliverables and milestones for the ESV competition are laid out in the Management Plan section. The regional evaluation is in the beginning of March; while we will not have a functional prototype at this point, we will have systems demonstrations and sensor interactions prepared to demonstrate. Fortunately, the international conference is not until June, providing us the time to continue developing and refining the prototype.

We have encountered and processed several design challenges, and we expect to encounter more throughout the course of this project. The first method that will be utilized on a regular basis is to confer with other group members. Each member has unique skills and can offer alternate perspectives. However, problem-solving is never this simple, and more extensive measures will likely need to be taken. Problem solving is a lot like the design process, just scaled down. The problem should be defined and causes determined. Next,

brainstorming solutions, preferably with a group, can generate a wide variety of ideas. Assessing each idea and choosing the best solution requires a bit more thinking and is a very important step in the problem-solving process. Once all of this has been completed, the solution should be implemented, and modified if the need arises.

4.2 CONCEPT DEVELOPMENT

In developing an idea of what we needed our prototype to be we needed to consider a few unique ideas. The first of which is that we are not developing this for ourselves but for other students and researchers to use. That means it needs to have a high level of usability and accessibility in the design. Along with building it for usability, we also needed to construct a very robust and enduring model. Since our group won't be there to fix any parts, we needed to make a model and will last through 2+ years of student use. This is no small feat if you've seen how students treat lab equipment.

Another consideration would be the true performance of the vehicle and how many detailed features we can install for in-depth development. This third performance aspect creates somewhat of a Venn-diagram triangle with usability and durability. We want all three but we won't be able to significantly sacrifice one aspect for the sake of another.

A structured concept development process ensures a variety of potential solutions can be generated, assessed, and validated. The first step to generating good design concepts was to brainstorm. The overall concept of our project was already defined as a small scale vehicle platform. This meant that the brainstorming was primarily focused on specific subsystems and functions that the vehicle needed to perform. The first brainstorming session focused on developing concept ideas for the different subsystems. The subsystems/functions were identified as follows:

- Protective System
- Electronics Layout
- Vehicle Protection
- Types of Sensors
- Center of Gravity Adjustment
- Algorithms
- Braking Systems

To generate the new concept ideas, we each individually came up with concept ideas that fell within each of the subsystems listed above. We transcribed our ideas on sticky notes and placed them on an empty wall in no particular order. Three sessions were conducted that lasted for two minutes each. The results from this brainstorming session can be seen as a morph chart in Appendix C**Error! Reference source not found.**

Another brainstorming session focused on a functional decomposition of what makes a vehicle. This is what helped us to explicitly define the subsystems of our concept. To be a complete scale model, our vehicle would need a drivetrain, braking system, electrical system layout, autonomous/manual control system, sensor integration, chassis, protective frame, wireless remote control, and a Simulink template.

We then conducted a brain sketching activity to help to visually generate concepts of the subsystems in a variety of different ways. We each started off with a specific system and sketched a concept. The concepts

were rotated between the three of us until we had developed a concept for each subsystem. Our drawings were not professional by any means but this helped us to flush out ideas at a high pace. An example of a drawing can be seen in Figure 5 and the rest of the production can be seen in Appendix C.

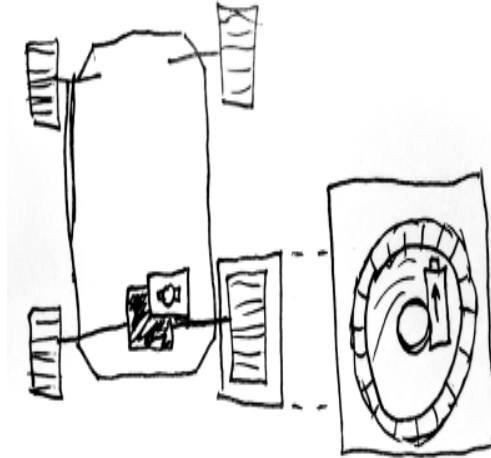
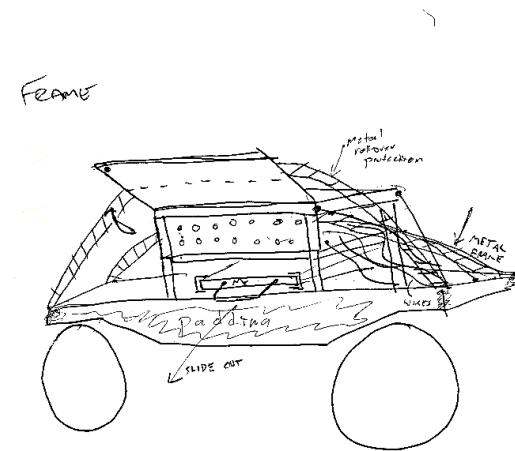


Figure 5: Braking Subsystem Example

Eventually when we had burnt ourselves out of subsystem drawings we decided to try to visualize different subsystems working together. We used the same brain sketching model as described above to try to draw a full system. Depicted in Figure 6a below is an example brainsketch of a full system design.



(a)



(b)

Figure 6: Brainstorming and prototyping examples for (a) brain sketching activity and (b) braking method test stand.

After generating concept ideas and sketching them, we next made physical prototypes. Our prototypes consisted of a braking method test stand (Figure 6b), an electrical layout, and several protective housing systems. The braking stand had a mounted disk that could be braked with an actuated rod that modeled the friction solenoid, a permanent magnet, and an electromagnet. Results from the braking test led us to believe that any magnetic braking system would not be feasible. The electrical layout was a piece of foam board cut in the shape of the vehicle chassis, where we could move components around and place them with pins. The protective housings consisted of a wire cage, a hinged box, and a drop out electronics board. Images from the prototyping session can be seen in Appendix C.

4.3 COMPARISON OF CONCEPTS

After the brainstorming sessions were complete, we began to eliminate ideas with Pugh matrices for the RC car selection, microcontroller selection, potential algorithms, braking methods, protective housing, CG modifications, and electrical layout. The complete set of matrices can be seen in 0, while a discussion of each matrix is found below.

4.3.1 Initial RC Car Comparison

Six initial RC cars were identified from leading manufactures. The main criteria for the cars that were selected was that they have suspension and a source for purchasing individual replacement parts. The six selected vehicle were then compared against each other, with a 1/16th scale Traxxas E-Revo as the baseline. Despite being suggested to build a 1/10th scale prototype, a 1/16th scale car would be a cheaper alternative and would allow us to easily outfit an independent motor system with a custom chassis. A 1/16th scale 4WD vehicle would be cheaper than its 1/10th scale 2WD counterpart and the 4WD would already include constant velocity shafts for the front two wheels. This could save us a considerable amount of work integrating a custom drivetrain. Compared to HPI racing and Axial, Traxxas vehicles have a much wider array of replacement parts, which is a very important parameter in choosing which RC car to purchase. Results from the Pugh Matrix allowed us to eliminate most of the vehicles, where we narrowed it down to the 1/10th scale Traxxas Slash and 1/16th scale 4WD Traxxas slash. These vehicles were included in the final decision matrix as different system level concepts. We also decided to entertain the idea of a custom chassis. This would allow us to design a shape for the chassis that can explicitly fulfill our space requirements. If designed correctly, a custom built chassis would also result in a sleeker, more professional and high quality build. It would also allow a more precise and consistent result, as we would not have to drill mounting holes into the stock chassis by hand.

4.3.2 Microcontroller Selection

In our microcontroller analysis, we compared the Arduino Mega 2560, BeagleBone Black, and the Raspberry Pi 3B. We quickly determined that the Arduino Mega 2560 did not have the clock speed or RAM quantity that we were looking for. It also lacked compatibility with the Simulink computer vision toolbox. While computer vision is not directly in the scope of our project, it is a system that many find interesting and our platform may see computer vision research in the future. This left the BeagleBone and the Raspberry Pi. These two systems are very comparable and both had some beneficial attributes that the other

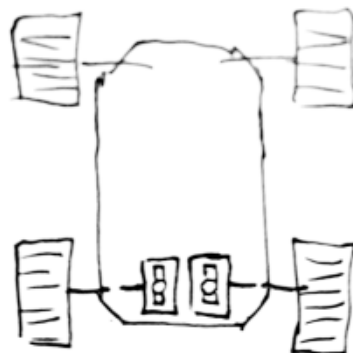
didn't. While the Raspberry Pi is slightly less expensive than the BeagleBone, the difference is a small portion of our overall project; we decided to focus on assessing which would work better. The BeagleBone had nearly twice as many ports for future sensor connections than the Raspberry Pi. However, The Raspberry Pi had a faster CPU, more RAM available, and had multi-processing functionality. The multi-processing feature for the Raspberry Pi is what ultimately tipped the scales. In a complicated and intense control algorithm, many tasks will need to be computed as fast as possible and the multi-processing feature means that we can do more than one task at a time. This attribute could greatly speed up an intense control algorithm and had a substantial influence in our Microcontroller selection.

4.3.3 Baseline Algorithms

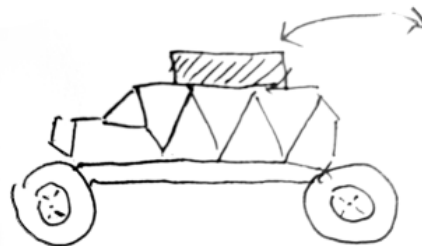
The Pugh Matrix for potential algorithms helped us to determine what we wanted our platform to be able to run without any adjustments or add-ons. Essentially, these are the simplest, most practical, and interesting systems that we want to be able to test. These systems were determined by the matrix to be adaptive cruise control, electronic stability control, multi-vehicle management, and collision detection. Reassessing these results led us to believe that a lane following system would be useful on the baseline model. Multi-vehicle management and collision detection systems would require more complex systems that would be interesting to incorporate by adding different modules, but is not something that should be required by the simplest model. As a result, our baseline model will simply have adaptive cruise control capabilities through a 1-D distance sensor, lane following through a light sensing module, and electronic stability control through the IMU and independently powered wheel systems.

4.3.4 Braking Method

The braking method was seen as one of the most important brainstorming topics. It is imperative the vehicle can brake at least two wheels independently in order to test ESC algorithms. Our prototyping session essentially allowed us to discard of any magnetic system, however, we still included these in the initial Pugh matrix. As expected, the results from the matrix helped us to narrow down braking concepts to a friction solenoid and regenerative braking through independently powered motors. A two motor system (Figure 7a) would be much simpler than a four motor system, but would not provide as much control when compared to a four motor system. These concepts were next incorporated into our final set of concepts, which was analyzed in a decision matrix.



(a)



(b)

Figure 7: Design concept sketches for (a) 2 motor power system and (b) rack with weights mounted on wire cage.

4.3.5 Protective Housing

Electronic components need to be protected in order to prevent irreparable damage or unnecessary replacements. During brainstorming we came up with several methods for protecting the MCU and associated central components. A wire cage, hinged shell, drop-out assembly, and cushioned bumpers were each considered in the Pugh matrix. Results determined that a drop-out assembly and boxed hinge system were potentially the least effective and most impractical options. We predict the wire cage to be easy to implement and adjust, as roll cages are available for many RC cars from third-party manufacturers and a demo we recently looked at enhanced these suspicions. A bumper system also usually comes equipped on off the shelf products. These two systems may be used in conjunction with each other and not require a lot of modification, which would be ideal.

4.3.6 Center of Gravity Modification

Raising the center of gravity is necessary to allow the vehicle to rollover and mimic standard vehicle dynamics. We considered using an adjustable center of gravity through some sort of an angled boom, but this idea was unnecessarily complex. Simply using a rack mounted to the wire cage (Figure 7b) would be a very simple, and even adjustable method of approach. If a custom chassis is used, we may also implement an adjustable chassis height through spacers. The size of the spacers could be left up to the user and would provide valuable customization through the height of the chassis.

4.3.7 Electronics Layout

Electronics layout is an interesting design challenge – we must interface with sensors across the car, motor(s) for the drivetrain, and the central processing unit, all while being adaptable to other sensors and components. For adaptability, connecting the microcontroller to a breadboard may be ideal as it provides the ability to plug in and remove components and wires at the user’s discretion. Unfortunately, this leads to inconsistent connections, loose wires that complicate cable management, and an inconsistent user experience. Another option is to create a ‘stack’ of plates and attach individual components to that. While this enables better cable management, it is space intensive and not as adaptable. Instead of a stack, the components could be affixed to the chassis and permanently interconnected, but this again loses adaptability and complicates manufacturing. The most attractive choice for us was to use a motherboard / daughterboard system. The microcontroller will act as a daughterboard, and a motherboard will be made of single PCB or perfboard will be attached to the microcontroller’s pins. We will attach the IMU and other sensors to motherboard directly, and break out multi-pin connectors for connecting motors, baseline sensors, and user-selected sensors. This layout gives us the most compact system and provides excellent durability and consistency while maintaining adaptability. The use of multi-pin connectors also supports effective cable management.

4.4 SELECTED CONCEPT

After narrowing down our subsystem concepts and deciding on a microcontroller, compatible algorithms, protective housing, and an electrical layout, a collection of system concepts was generated. The next task was to decide the final RC car and chassis type, braking method, and type of CG modification. Table 4

defines the eight distinct concepts that were compared in our decision matrix, which can be found in Appendix D. It should be noted that the weight factors generated from the QFD were redistributed for our version of the decision matrix. This is because at this point in the decision process, subsystem designs were already determined, meaning that all concepts would have the same score in certain sections.

Table 4: System level concept descriptions.

System	Function						
	RC Platform	Microcontroller	Compatible Algorithms with Baseline Design	Braking Method	Protective Housing	CG Modification	Electrical Layout
1	Traxxas Slash	Raspberry Pi 3	ACC, ESC, Lane Following	Friction Solenoid	Wire Cage	Rack with weights	Mother/Daughterboard
2	Traxxas Slash w/ custom Chassis	Raspberry Pi 3	ACC, ESC, Lane Following	Regenerative (2 x Motor)	Wire Cage	Rack with weights	Mother/Daughterboard
3	1/16th Slash w/ custom chassis	Raspberry Pi 3	ACC, ESC, Lane Following	Regenerative (4x Motor)	Wire Cage	Rack with weights	Mother/Daughterboard
4	Traxxas Slash w/ custom Chassis	Raspberry Pi 3	ACC, ESC, Lane Following	Regenerative (4x Motor)	Wire Cage	Adjustable Chassis	Mother/Daughterboard
5	1/16th Slash w/ custom chassis	Raspberry Pi 3	ACC, ESC, Lane Following	Friction Solenoid	Wire Cage	Adjustable Chassis	Mother/Daughterboard
6	Traxxas Slash	Raspberry Pi 3	ACC, ESC, Lane Following	Regenerative (2 x Motor)	Wire Cage	Rack with weights	Mother/Daughterboard
7	Traxxas Slash w/ custom Chassis	Raspberry Pi 3	ACC, ESC, Lane Following	Regenerative (4x Motor)	Wire Cage	Rack with weights	Mother/Daughterboard
8	1/16th Slash w/ custom chassis	Raspberry Pi 3	ACC, ESC, Lane Following	Friction Solenoid	Wire Cage	Rack with weights	Mother/Daughterboard

The results from the decision matrix indicated that concept 2 was the best system level design for meeting our specifications. Upon inspection, this seems like a very reasonable result. We would expect a 1/10th scale vehicle with a custom chassis to be physically robust, and easily adaptable. Incorporating a two motor system would also be much easier than a four motor system and even the friction solenoid method. However, we have decided that a four motor system would be much more beneficial in the long run, allowing more in depth development of stability and traction control algorithms. A sketch of the selected concept can be seen in Figure 8.

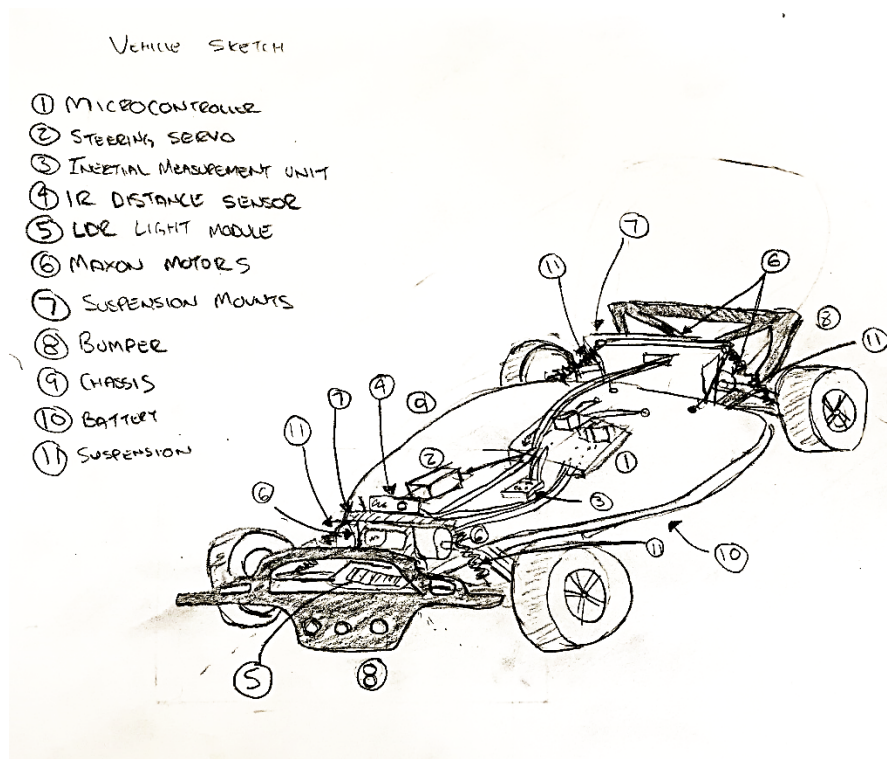


Figure 8: Final Concept Sketch of Modified RC Car. Not Shown: Wire cage for component protection and weight rack for CG modification

In our selected design, a custom chassis would be easily repairable, and if designed correctly easily manufactured. When considering cost concept 2 is relatively average, but cost is not a driving factor. In conclusion, our final design will be adapted from a 1/10th scale Traxxas Slash with a custom chassis and four independent motors. Though our initial thought was to use the 2WD Traxxas Slash, the inclusion of four independent motors makes the 4WD Slash a more attractive chassis. While more expensive, it contains all of the drive-shafts and steering geometry required. The microcontroller will be a Raspberry Pi 3 and be able to link with sensors for ACC, ESC, and lane following control systems. The microcontroller will easily be able to intermediate control algorithms without any visible latency. ACC systems will be made possible using an ultrasonic linear distance sensor. Yaw rate and axle angular velocities necessary for stability control will be determined by the IMU and halls sensors, respectively. A wire cage will protect the fragile components with a rack on top for a weighted CG modification. A mother/daughterboard layout will also be utilized. A secondary microcontroller will be present on the motherboard, allowing for additional input / output pins and handling of time sensitive tasks. Parts will be easily sourced for any repairs from the Traxxas website.

We will work to compartmentalize our design by having our drivetrain, mechanical system, and electrical system all work on their own. While they will work together in our final design, this compartmentalization will allow easier adaptation of our design work for future researchers. While our design does well to meet our customer requirements and specifications, it does come with challenges, including space constraints, custom PCBs, manufacturability concerns, and more. Figure 9 presents the challenges of specific subsystems and how they play in to our compartmentalized design.

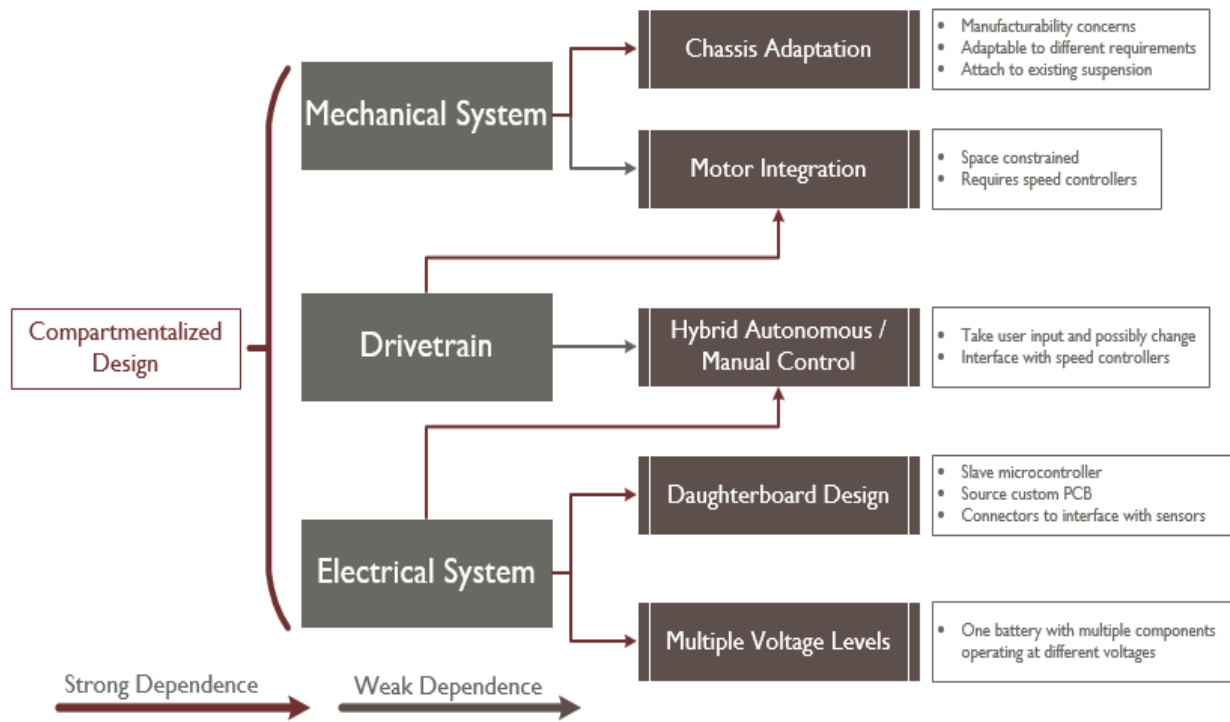


Figure 9: Design challenges with the specific subsystems that they stem from

4.5 PRELIMINARY DESIGN ANALYSIS

Now that we have selected a concept, analysis must be completed to ensure that the design meets our requirements and is feasible to manufacture within the time and budget constraints. In addition to identifying if our selected components will work, our preliminary analysis will include similitude calculations for dimensions and forces to show the strengths and weaknesses of our design as an advanced rather than preliminary research platform.

4.5.1 Dimensional Similitude

When designing a scale model, it is very important to maintain relative similitude between the full scale model under investigation and the small-scale model used to gather information. As a result, a similitude analysis should be completed. For this specific case, similitude is not necessarily a critical issue, as the main purpose of the platform is for experience and testing the feasibility of vehicle algorithms, not for exact modeling of a full scale vehicle. Nonetheless, we want to design a platform as realistic to a full scale vehicle as possible. When creating a scale model, it is possible to scale different base quantities, primarily being mass and length. Scaling by length puts the mass at a different scale, and vice versa. To be thorough, we analyzed the dimensional requirements of a vehicle scaled on both mass and length, just mass, and just length. These values were compared to that of a standard vehicle, for our purposes, a Ford Edge. The analysis can be found in Appendix E, and the results are tabulated below:

Table 5: Results of dimensional similitude analysis, with RC car for comparison

Parameter	Wheel Diameter (in)	Wheelbase (in)	Track Width (in)	CG height (in)	Power (hp)	Weight/Power (lb/hp)
Typical Vehicle (Ford Edge)	20	112	65	36	250	16
All Parameters Scaled	2.0	11.2	6.5	3.6	25	16
Length Scaled	2.0	11.2	6.5	4	0.25	16
Weight Scaled	11.1	88.2	33.4	18.6	25.0	16
RC Car (Slash 2WD) [25]	2.20	13.20	11.65	-	0.60	1.97

Following the similitude analysis, it becomes immediately apparent that scaling a vehicle by weight would result in large dimensions that are impractical for the purpose of this project. The most reasonable method is to scale the vehicle by its dimensions, and simply purchase and off the shelf 1/10th scale RC car. When scaling by length, mass scales in a cubic fashion – our 1/10th length scale vehicle will have 1/1000th mass scale. The smaller mass scale has implications on both dynamic similitude and power. The Traxxas vehicle is significantly more powerful per pound than a full scale car, which is something to be aware of when attempting to address similitude. We can maintain a more similar power by using the lower portion of the motor’s performance curve and not letting the motor reach its full potential. Operating the motor at this point reduces the motor efficiency, but also reduces the overall power consumption. Since efficiency is not a primary concern for our project, artificially limiting the motor power will not detract from our final design.

Table 6: Comparison between ideal dimensionally scaled model and Traxxas Slash

Parameter	Traxxas Slash	Dimensions Scaled	Percent Difference
Weight (lb)	4.76	4	19
Wheel Diameter (in)	2.2	2.0	10
Wheelbase (in)	13.2	11.2	19
Track (in)	11.65	6.5	80
CG Height (in)	-	4	-
Power (hp)	0.6	0.25	140
W/P (lb/hp)	1.97	16	88

Table 6 provides insight to what parameters are going to be important to pay attention to when modifying an off the shelf RC car. It becomes obvious that the vehicle will be overpowered, but another important aspect will be the center of gravity. The ratio of the track length to the wheelbase is much greater for the Traxxas vehicle than a scaled model. This means the Slash will be more difficult to roll-over than a standard vehicle. Normally this would not be an issue. In fact, the Slash is probably designed not to roll-over. However, for our experimental purposes, we want a car that can flip and rollover in ways similar to many standard cars on the market. This can be overcome by simply modifying the center of gravity of our 1/10th scale vehicle.

4.5.2 Braking Forces

One of the main requirements of the vehicle platform is that it has independently controlled wheels. Standard RC cars do not come equipped with disc brakes, or independently powered wheels, meaning that an off the shelf vehicle would have to be equipped with some sort of a mechanism to brake at least two wheels. During brainstorming, we developed several methods for braking an RC car, one of which was a friction solenoid. In this concept a solenoid equipped with a rubber tip would actuate toward a disk mounted on the wheel shaft, causing a reversing torque that would be transmitted to the ground as a force. We were interested in the feasibility of using a solenoid as a brake, so brief calculations were conducted and can be seen in Appendix E. Through these calculations, it was found that a solenoid would need to generate about 6 lbf to successfully stop an RC car traveling at 15 ft/s. This is a reasonable force for a solenoid to generate, meaning we could continue pursuing it as a possible braking solution.

4.5.3 Motor Selection

Collaborating with Charlie Refvem, a prospective graduate student interested in completing work consistent with our goals, we have selected motors to use on each wheel. We plan to use Maxon flat brushless motors, Part Number 397172 [26]. These are 24V, 70W brushless motors that have built in hall effect sensors (for low resolution position + speed) and the option for a built in high-resolution encoder. These specific motors were selected due to their unique design for high torque, relatively low speed, profile, and low starting threshold. Standard brushless motors have extremely low starting torques and extremely high nominal speeds, reducing their applicability for ground vehicle design. While a gear reduction would often be used in an RC car to either increase torque or speed, these motors were selected specifically to not require a gearbox. A gearbox at each wheel would add complexity to the drivetrain, reduce the amount of space we have to work with, and the high torque of the motor may wear down the gears, creating another potential point of failure. Additionally, gearboxes would add backlash and inertia – two things that make electronic stability control more difficult. We confirmed that these motors would be appropriate for our platform by analyzing their torque and speed outputs, calculations for which are found in Appendix E. We found that we could produce a nominal speed of 43.3 mph and an acceleration of 9.5 ft/s^2 . While this exceeds our specifications, we can control the motors to a lower speed and torque.

4.6 CONCEPT REFINEMENT

As we exited the preliminary design phase, we began refining the specifics of the motherboard, drivetrain system, and more. At this stage, we were beginning to think about manufacturing and documentation. Figure 10 presents our initial plan for work from PDR through April with Critical Design Review and ESV Prototype Evaluation noted as important deliverables.

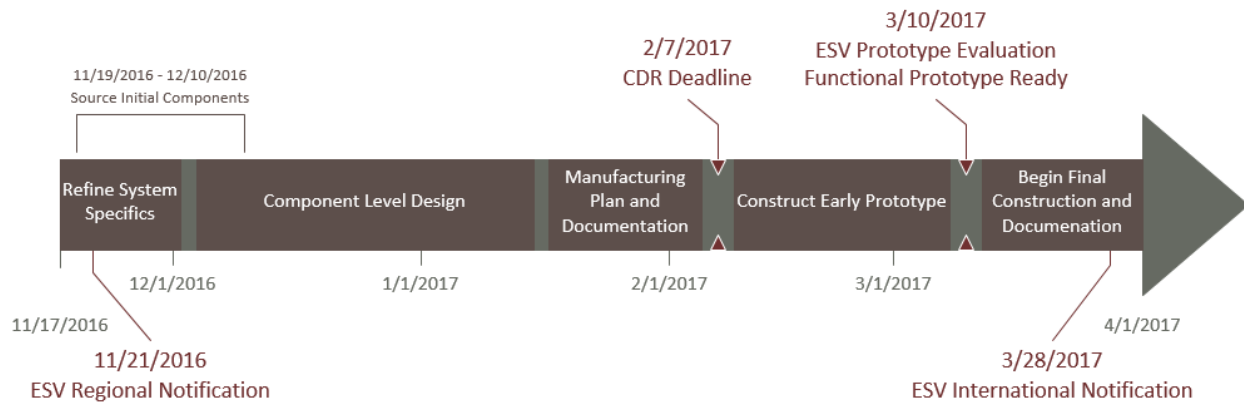


Figure 10: Early planning timeline of milestones through ESV International Notification

4.6.1 Design Planning

A preliminary solid model for our custom chassis has been created based upon the dimensions of the Slash and can be seen below in Figure 11. More detail will be added once the Traxxas Slash is purchased and inspected. We will create geometry and attachment points that align to our specific system's needs. This will support adaptation and recreation of the design, support consistent wire management, and enable us to create space for sensors that the user may add.

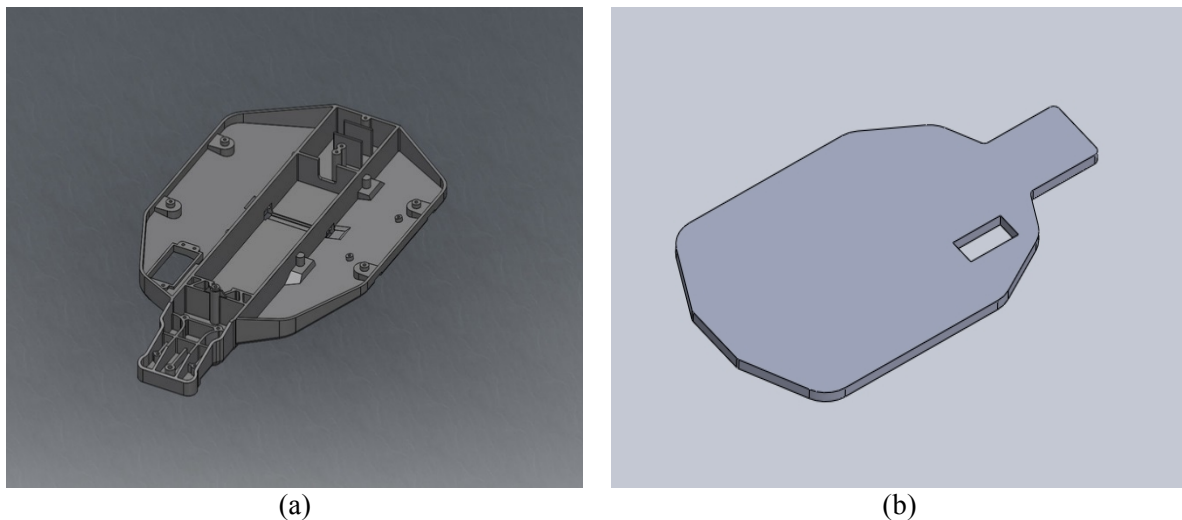


Figure 11: Solid models of (a) Stock Traxxas Slash (2WD) chassis and (b) early mockup of custom replacement chassis.

In addition to the custom chassis, we must design motor and component mounts, shaft collars for the motors, and our motherboard.

For the motor mounts, we need to integrate the mounting locations to incorporate the Traxxas suspension. Our motor selection does result in us widening the drivetrain due to the possible locations for a joint / shaft

couplers. To complete this in a way that fully supports the stock Traxxas suspension / drive system, we will review the geometry of the Traxxas Slash and create a full drivetrain solid model. Splitting the geometry down the middle, we will expand the hole layout and u-joint position to accommodate the additional space requirements. Using calipers, we reviewed, documented, and recreated the Slash's front and rear drivetrain geometry in Solidworks. The solid model of the Slash's front drivetrain is presented below in Figure 12.

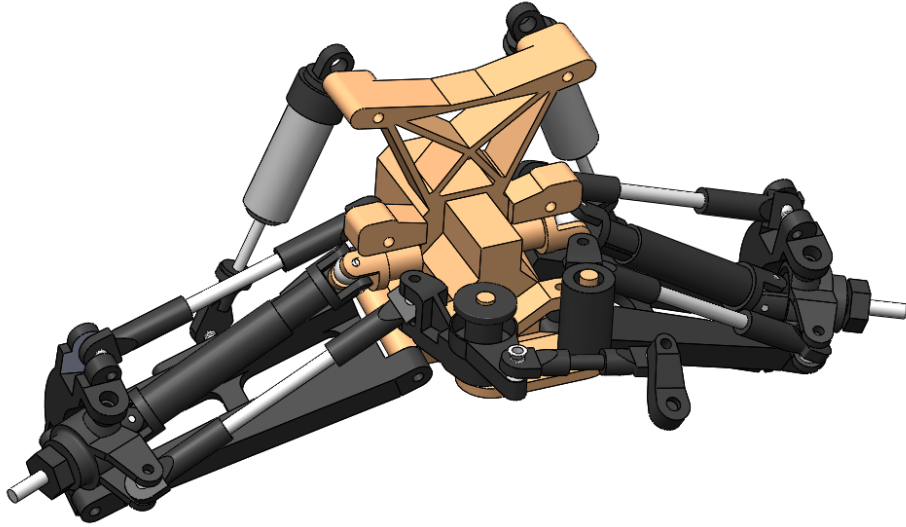


Figure 12: Stock front drivetrain of Traxxas Slash – tan components will be replaced with custom parts that accommodate the Maxon motors.

Our adaptation maintains the relative location of each side's mounting points to other mounting points on the same side, but increases the dimension from the mounting points to the centerline. This provides more space to integrate the motors.

As we developed the geometry to integrate the Maxon motors in to the drivetrain, we found that we were having to increase the track width more than anticipated. Figure 13 compares the stock and custom drivetrain mounts and shows the increase in track width.

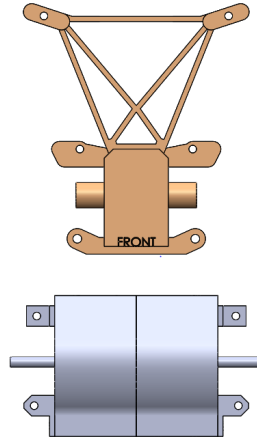


Figure 13: Comparison of stock (top) and custom rear drivetrain mounts. Note that the rear suspension uprights are not pictured on the custom mount.

Since the Traxxas Slash already had an aspect ratio more square than real cars, we elected to also increase the wheelbase to refine the dynamic properties that stem from aspect ratio. We found that our new geometry reflected a $1/7^{\text{th}}$ scale vehicle. RC cars often have oversized suspension and wheels, so the increased size improves the drivetrain similitude.

As a result of the increase in vehicle size, we are no longer able to easily implement an aftermarket protection system for the Traxxas Slash. Through the prototyping process we will continue to search for and develop ideas for an effective protection system, but we have withdrawn the protection system from our critical deliverables. This is appropriate because our system is intended to demonstrate the capabilities of a small scale vehicle platform and not to be a final design. While we continue to search for an effective protective system, we will complete thorough stationary testing on the controls system and low speed dynamic tests. This methodology will also us to test and refine the system without putting it in jeopardy.

5. FINAL DESIGN

Our final design implements a custom drivetrain and control scheme while retaining the Traxxas suspension, drive-shafts, steering configuration, and radio receiver. We will provide a Simulink template that includes custom blocks receive data from the sensors and output control data to the steering servo and motor on each wheel. The user can upload designed Simulink algorithms to the on-board Raspberry Pi, which interprets sensor data to produce the programmed response. Figure 14 below pictures the car and component placement – cabling and sensors are omitted.

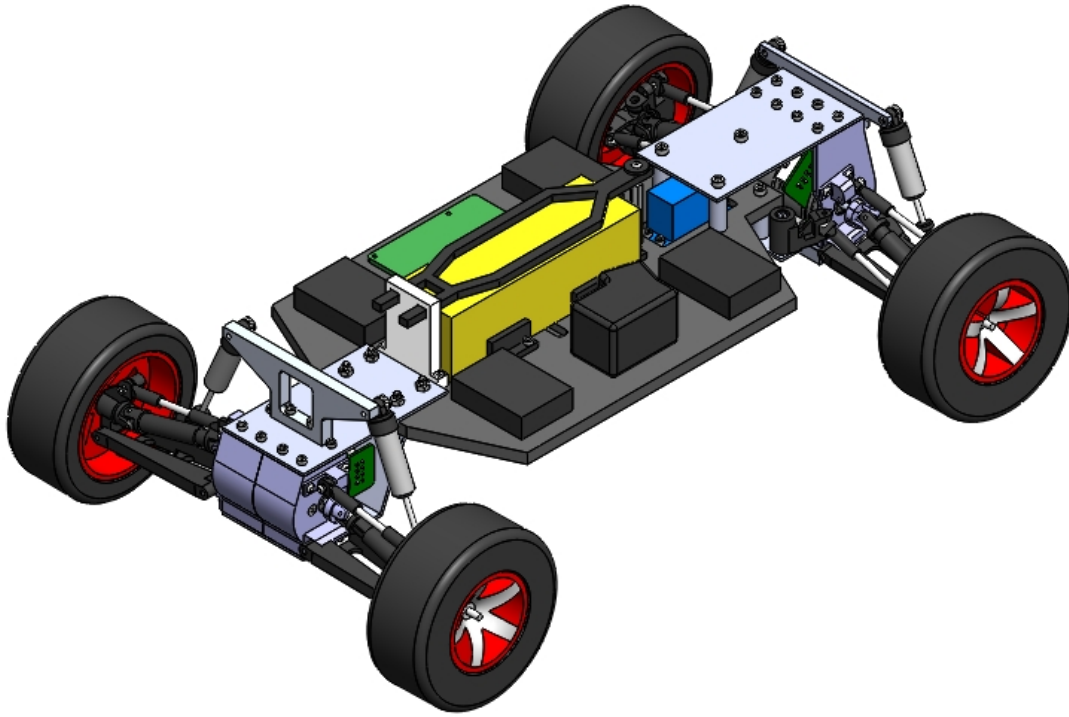


Figure 14: Isometric view of final design with approximate shapes for electronic components and cabling omitted.

This design accomplishes our primary objective – provide a simple accessible platform for developing intelligent vehicle control. It provides an Inertial Measurement Unit (IMU), Ultrasonic Rangefinder (URF), camera, and motor encoders to provide the spatial data required to implement many intelligent vehicle control regimens. Through the designed control algorithms, the user will be able to adjust the remote control inputs and control the steering angle and manipulate the position / speed of each wheel. Our custom motherboard attaches directly to the Raspberry pi and offers connections to the battery, motor drivers, and sensors. Additionally, it can offer data-logging support and has an RGB indicator led to visually notify the user of fault states. For additional safety, we have included a remote cutoff switch that will disable the motors.

5.1 MECHANICAL ARCHITECTURE

5.1.1 Structural Design

The new components that we designed were typically static members, which means they would need to be able to support impulses and static forces but would not be required to translate or rotate. The two exceptions are the shaft couplers and the new steering linkage. We tended to take a conservative approach with our design, with similar geometry to the Traxxas parts we were replacing. The majority of the parts will be manufactured from aluminum, while the chassis and non-critical components will be made out of Nylon and PLA, respectively.

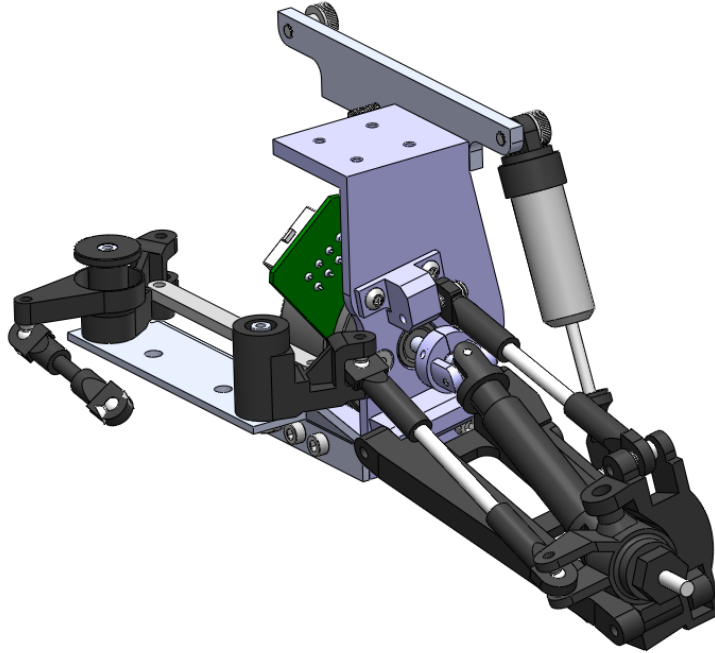


Figure 15: Existing Traxxas steering and suspension system integration with designed components.

The mechanical layout is simple, and will maintain the vehicle like properties of the Traxxas Slash. The front and rear motor blocks consist of the original Traxxas suspension linkages, where the front is integrated with the original steering system. Widening the track width requires that a new steering linkage be designed that connects the left and right steering bell cranks. Furthermore, using the Maxon motors requires a coupler that is compatible with the existing Traxxas constant velocity shaft.

5.1.2 Dynamic and Similitude Properties

Developing a realistic model in Solidworks is key to estimating the mass and geometric properties of our design. The estimated parameters, along with their sources, can be seen in Table 7. A significant portion of the design phase was spent creating solid models of the existing Traxxas Slash parts so that we could accurately incorporate them into our final design. This allowed us to design our new parts so that the steering and suspension geometry would not change. However, incorporating the Maxon motors into the design meant that we would need to increase the track width.

Table 7. Estimated final design vehicle parameters

Symbol	Description	Value	Units	Source
m	Vehicle mass	3.8	kg	Measured
a	Distance from CG to rear axle	204	mm	Measured
b	Distance from CG to front axle	197	mm	Measured
H	Height of CG	75	mm	Measured
I_{zz}	Moment of inertia about vertical axis	0.136	kg-m ²	Solidworks Model
c_α	Tire cornering stiffness	63.2	N/rad	T. Stevens
L	Wheelbase	402	mm	Measured
T	Track	275	mm	Measured
D_w	Wheel diameter	112	mm	Provided
AR	Aspect ratio (L/T)	1.45	-	Measured

Our scale model is sized to be similar to a standard SUV, such as the Ford Edge. A side by side comparison of the two can be seen below:

Parameter	1/7 th Scale Ford Edge [27]	Our Vehicle
Wheelbase (mm)	406	402
Track (mm)	236	275
Aspect Ratio	1.72	1.45
Weight (kg)	5.03	3.8
Wheel Diameter (mm)	90	112
Turning Radius (mm)	1600	1300

We can see from this comparison that widening the track width has made the vehicle more of a 1/7th scale vehicle, but dimensionally it is very similar. The tires appear to be oversized, but this is a factor out of our control. Another interesting parameter is the weight. As mentioned earlier in the report, scaling 1/10th by dimension scales the weight by 1/1000th. As a 1/7th scale dimensionally, the weight should scale by 1/343rd. This makes a 1/7th scale model of the Ford Edge slightly heavier than our model. However, our model is not complete, or completely accurate and we anticipate the final weight to rise to 4.5 to 5 kg.

5.2 CONTROLS ARCHITECTURE

The mechatronics systems were designed to maximize functionality while retaining a high level of usability and accessibility. Upon successful firmware implementation, the user will be able to access the full functionality of the sensors and motor drivers from Simulink. The first implementation will have hardware that allows for the addition of new sensors and outputs. While the user will have to modify the firmware depending on the functionality they hope to add, we will provide an example of what a user would have to do to add a separate module. Figure 16 shows the designed modularity, with native Raspberry Pi code (Simulink, C++, Python, or otherwise) talking through the motherboard in a standardized way. The motherboard holds the Teensy, which will have firmware specifically adapted to talk to the hardware

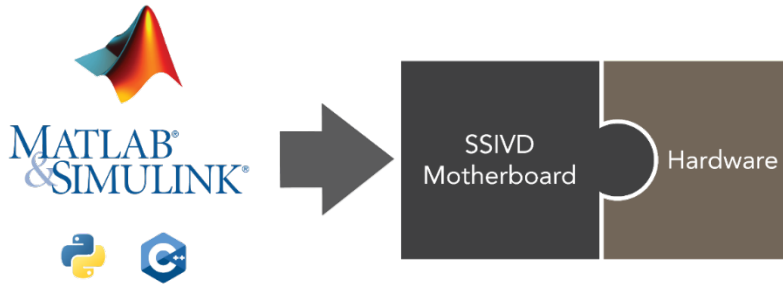


Figure 16: Hardware abstraction design of the SSIVD platform.

5.2.1 Steering and Drive Motors

The car is moved by the drive motors and the steering servo. For steering, we are still using the servo from the Traxxas Slash for compatibility with the steering hardware that is part of the original suspension and drive system. On the Slash, this servo is driven at 6V and the angular position controller takes a pulse-width modulation (PWM) input. The drive motors are Maxon EC 45 Flat 70W brushless motors with a 1024 count per revolution encoder. Additionally, they have a 3 count hall sensor that is used to assist the motor driver in properly timing signal at lower speeds. We selected the 24V configuration of this motor, and will be driving them using the Maxon EPOS4 Compact 50/5 positioning controller, pictured in Figure 17.

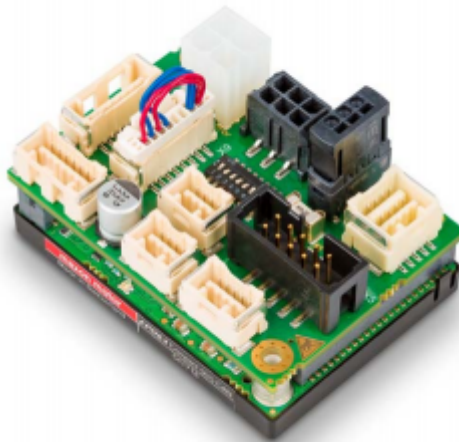


Figure 17: Maxon EPOS4 Compact 50/5 Can Positioning controller with manufacturer supplied connector board.

Each controller can drive up to 50V and 5A, and have a voltage drop of approximately 10% the supplied voltage. A 7s LiPo battery will supply the controllers with 25.9V, which will output just under the desired 24V. The drivers receive commands and transmit position and speed data on a Communication Area

Network (CAN) line compliant with the CiA 301 V4.2 Communication Protocol for Industrial Systems. The drivers run the CAN transceiver at 5V [28].

5.2.2 Radio Control

Using the Traxxas radio receiver, we can take the throttle and steering input from the handheld controller. It comes in as a 5V 100Hz PWM signal, which can be read, processed by the Raspberry Pi, and re-output with modifications defined by the control algorithm. Additionally, we are using a 315Mhz single button radio toggle as a remote kill switch.

5.2.3 Sensors

An intelligent vehicle requires awareness of itself and awareness of its surroundings. For awareness about itself, we are using a BNO055 IMU to provide orientation and acceleration data at 100Hz. The BNO055 module has a built-in processing unit that can output three-dimensional vectors for absolute orientation, angular velocity, acceleration, magnetic field strength, gravity, and temperature. It can be operated from 3.3-5V and communicates via the I2C protocol. To give a powerful but accessible awareness of the surroundings, we are providing a URF for distance measurements and a camera intended for computer vision applications. The Maxbotix HRLV-EZ0 URF operates at 2.7-5V, detecting objects from 1mm to 5m and returning a range for objects from 30cm to 5m with up to 1mm accuracy. The specific detection characteristics for varying shaped objects is shown below in Figure 18.

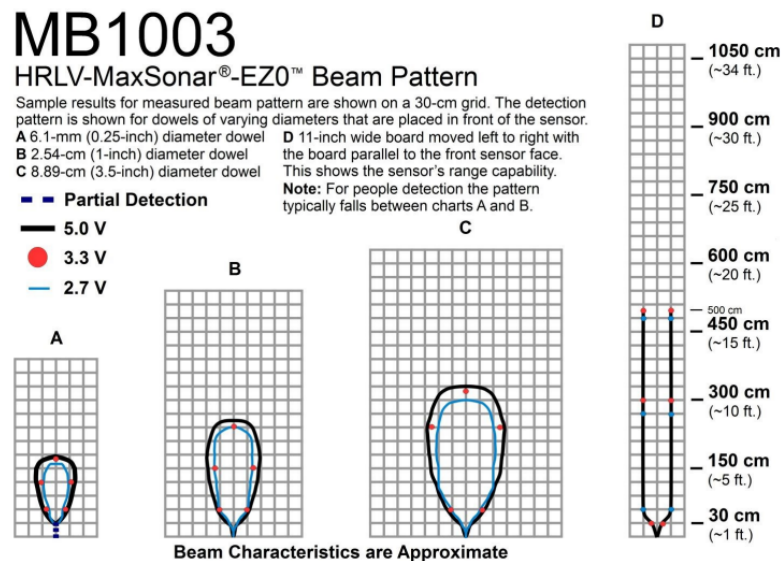
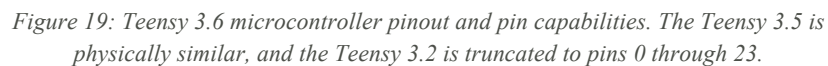


Figure 18: Maxbotix URF Detection characteristics for various object types.

The included Raspberry Pi Camera Board v2 is an 8 megapixel still and video camera than connects directly to the Raspberry Pi. It can be used with the Simulink computer vision toolbox for intelligent vehicle control algorithms or it can record and be used to review the performance of algorithms.

Since the sensor position configuration will depend on the user's application, we have not created fixed mounting locations. For our application, we 3D printed brackets for the IMU at the center of gravity and for the other sensors at the front of the car.

The motherboard will have an attached wire that connects directly to the LiPo battery. A commercial module will isolate and regulate 5V to power the Raspberry Pi, which will then output 3.3V. Our custom motherboard is designed to plug directly in to the top of the on-board Simulink programmable Raspberry Pi. We use a secondary microcontroller to interface with the peripheral sensors and the motors. On the motherboard is an attachment for any of the pin compatible Teensy 3.2, 3.5 or 3.6 microcontrollers. Any of those three microcontrollers work with the default sensors and configuration we have created. For maximum processing power and pin capability, we will be using the Teensy 3.6 microcontroller, the pinout for which is shown below in Figure 19.



33

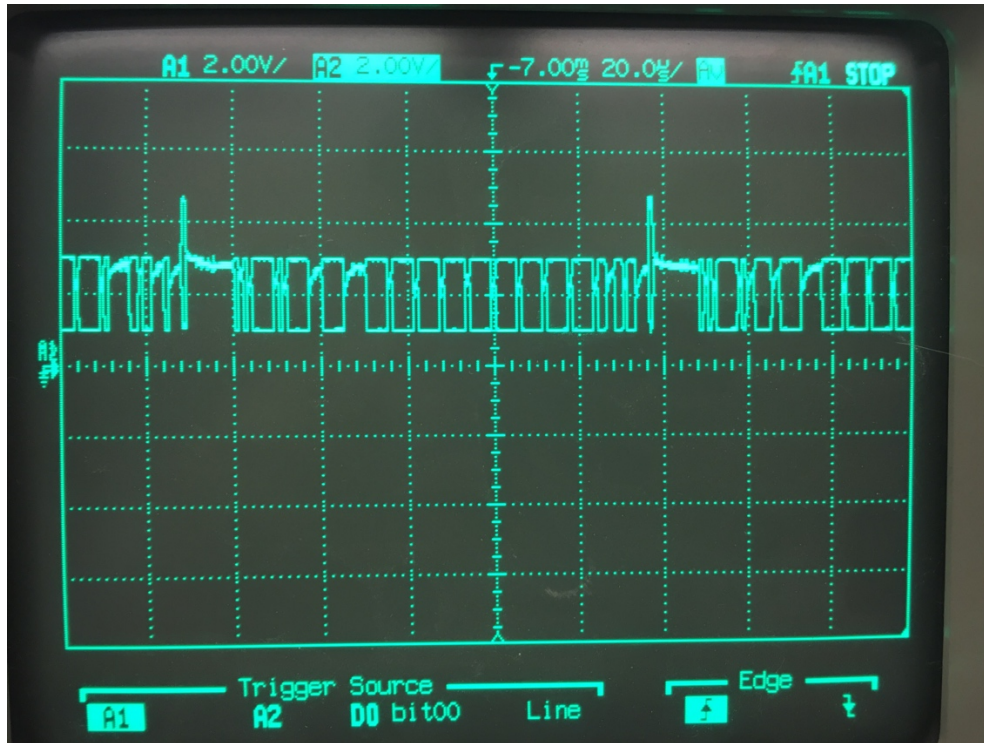


Figure 20: Oscilloscope Capture of the 3.3V and 5V Compatible CAN Bus

5.2.5 Communication Scheme

The Raspberry Pi receives data from and instructs the Teensy microcontroller via a serial connection. We had originally tried to use SPI as a connection protocol but due to trouble implementing this we switched over to strict serial communication. A CAN line is produced by a CAN transceiver chip connected to the Teensy's onboard CAN port which enables communication to the Maxon 50/5 Motor Controllers. PWM modules are used to communicate from the radio receiver to the Teensy and from the Teensy to the servo. I2C is used to communicate with the IMU and Laser Distance Sensor. Communication with the URF Distance Sensor uses analog input. Further discussion on the implementation of Pi / Teensy communication and sensor interaction can be found in the Firmware Architecture section below.

5.3 FIRMWARE ARCHITECTURE

5.3.1 Raspberry Pi

The Raspberry Pi will be programmed through Simulink and its control modules. The main functionality of the Raspberry Pi will be a loop format, continuously getting sensor and receiver data and outputting the calculated values to the Teensy microcontroller. Due to the limitations of the Raspberry Pi, it will need to be setup as the master in the SPI protocol when communicating with the Teensy. This isn't much of a problem though as the communication procedures will be planned to give enough clock cycles for the Teensy to respond back with all the sensor and receiver data. The Simulink code provided will then act as

the control algorithm and repeat the SPI communication. This loop will repeat for as long as the vehicle is turned on.

5.3.2 Teensy

The Teensy will be pre-programmed in C and will act as a fixed firmware. The Teensy, like the Raspberry Pi, will be also be programmed in a loop format. We had a goal of using hardware capture timers but instead decided to use interrupts due to time constraints.

With the steering servo that uses a PWM signal, the Teensy will be using a hardware timer that turns the pin on or off. The timer registers will be updated according, and will be discussed further shortly. The motor controller uses the CAN protocol which will be sent out as messages as soon as we receive the info. This leads us to talk about the SPI protocol with the Raspberry Pi. An interrupt will be programmed to read the input from the SPI signal and update the hardware timer for the servo and send a CAN message every time the SPI transfer is initiated.

The remaining modules are the URF sensor, receiver, and the IMU sensor. The IMU will be the only current module that will need to be polled for data in the main loop. The I2C protocol will need to query the sensor for its info periodically, which will be implemented in the main loop. The URF sensor will be read from a hardware analog module that updates a register with the new information. The register data will then be copied to a known variable in the main loop. The last module, the receiver, will be read using the hardware input capture which uses hardware to read the incoming PWM signal. The data received will then be copied to a known variable in our loop.

In all, we will have one interrupt from the raspberry pi driving the motor controller and steering servo's data. We will be capturing data using two different hardware configurations and one polling mechanism.

5.3.3 Fault States

Within both the Raspberry Pi and Teensy's firmware we will be fixing a series of fault states. These states will be more fully defined as a part of the firmware programming, but will prevent motion and illuminate the indicator led in a manner specific to the fault state. We anticipate designing fault states for user errors, communication failures, loss of radio control, and remote cutoff being triggered. As we encounter faults in the electrical prototyping phase we will continue to define and document fault states.

5.4 SAFETY HAZARDS

Using the Design Hazard Checklist from student success guide, we have proactively identified the following safety hazards in our device and have created mitigation plans for each.

- Design contains revolving action at wheels and potential pinch / shear points in structure
- Design can undergo high acceleration due to powerful motors
- Design will be able to go very fast presenting impact hazard
- There will be a large Lithium Polymer battery in the system, running at 25.9V
- It will be possible to use the system in an unsafe manner by driving into people or walls

The full design hazard checklist including mitigation plans is presented in Appendix E.

5.5 MAINTENANCE & REPAIR

The driving factor behind our decision to use Traxxas parts is that they are easily sourced and can be replaced if need be. Therefore, any Traxxas component that fails can simply be purchased from their website. All other designed components are intended to last for the entirety of the vehicle's life. However, some parts may need to be replaced simply due to wear and tear. This may include any electrical connections that become frayed or disconnected, bolts that strip from being over-torqued, or some of the 3-D printed parts. In each case, we anticipate that repair will be simple and not time-consuming or expensive. Overtime bearings used in the steering bell cranks may need to be lubricated, as well as the motor shafts.

6. DETAILED DESIGN & SUPPORTING ANALYSIS

6.1 MATERIAL SELECTION

We have been working with a materials engineering consultant group to help determine the best material to use for our purposes. The first design we discussed with them was the chassis. Our primary objectives with the chassis was that it would be lightweight, durable, and stiff. They generated a set of Ashby charts that helped to compare different material classes against each other. These charts can be seen in Appendix E – Materials Selection. The results showed that compared to other plastics, polyamides, such as Nylon, would maximize fracture toughness and fatigue strength, while still remaining stiff and light. Nylon is also easily machined and will be easy to tap for mounting purposes. Regarding other parts, we have sent the materials consultants all relevant information pertaining to the rest of our design. We are waiting for verification that aluminum is a strong choice for use in the motor housing and suspension mounting components.

6.2 BASICS OF VEHICLE DYNAMICS

Developing control algorithms for a small scale vehicle implies the inherent need to understand how the vehicle will move. A simple way of modeling a vehicle is through a single-track, bicycle model. This method combines the two tracks into one, eliminating a great degree of complexity while still maintaining an accurate representation of the vehicle dynamics. The basics of vehicle dynamics are outlined effectively in Rajesh Rajamani's book "Vehicle Dynamics and Control". It should be noted that the majority of the governing equations utilized in this section are referenced from Rajamani's derivations.

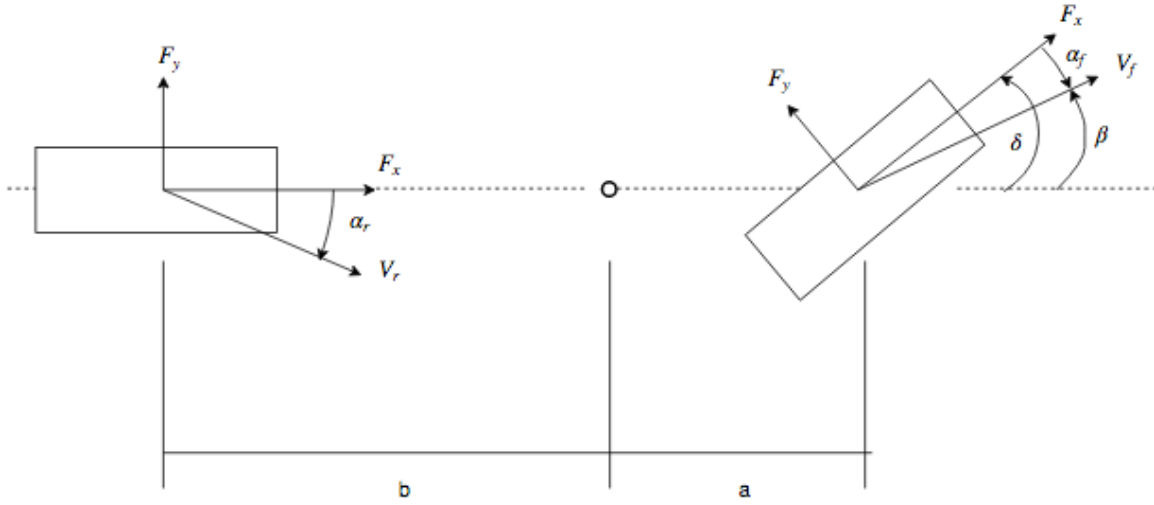


Figure 21: Schematic of Bicycle Model

6.2.1 Equations of Motion

The governing equations for the bicycle model can be derived using Newton's Second Law, resulting in a system of coupled, first order differential equations. In order to fully describe the motion of the vehicle, it is necessary to know the details of the forces that are acting at each tire-road interface. This is a widely studied area in vehicle dynamics, so a wide variety of literature is available on the subject. Note that the equations below omit the influence of rolling resistance and drag. These forces can be accounted for by simply including them in the longitudinal component.

$$\begin{aligned}\ddot{x} &= \frac{F_{xf}}{m} \cos \delta + \frac{F_{xr}}{m} - \frac{F_{yf}}{m} \sin \delta + \dot{\phi} \dot{y} \\ \ddot{y} &= \frac{F_{yf}}{m} \cos \delta + \frac{F_{yr}}{m} + \frac{F_{xf}}{m} \sin \delta - \dot{\phi} \dot{x} \\ \ddot{\phi} &= \frac{a * F_{yf}}{I_z} \cos \delta + \frac{a * F_{xf}}{I_z} \sin \delta - \frac{b}{I_z} F_{yr}\end{aligned}$$

6.2.2 Lateral Forces

Understanding tire mechanics is fundamental to predicting how a vehicle will react to a steering angle change or throttle input. Most tire models rely on empirical data gathered in a controlled test set up. One such model is Pajacka's magic formula, where the lateral tire force is calculated as a function of the slip angle:

$$F_{yf} = D \sin \{ C \tan^{-1} [B \alpha_f - E (B \alpha_f - \tan^{-1} (B \alpha_f))] \}.$$

Fortunately, Thomas Stevens, who worked on a similar project involving a Traxxas Slash, developed a tire test stand specifically for calculating the coefficients for the magic formula. These coefficients were slightly modified, and are listed below in Table 8.

Table 8. Pajecka's Magic formula coefficients for Traxxas RC car tires.

B	Pajecka stiffness factor	10.0	-
C	Pajecka shape factor	0.09	-
D	Pajecka peak factor	70.0	N
E	Pajecka curvature factor	0.65	-

Figure 22 shows the lateral force plotted as a function of slip angle using Pajecka's equation and the parameters collected by Stevens. Investigation of this plot shows that at small slip angles (about -0.1 to 0.1), the lateral force developed by the tire is linearly proportional to the slip angle. This is described by the cornering coefficient, c_α , which is approximately 63.2 N/rad. This value will potentially be very useful in the development of vehicle control algorithms that need to manage the motion of the vehicle.

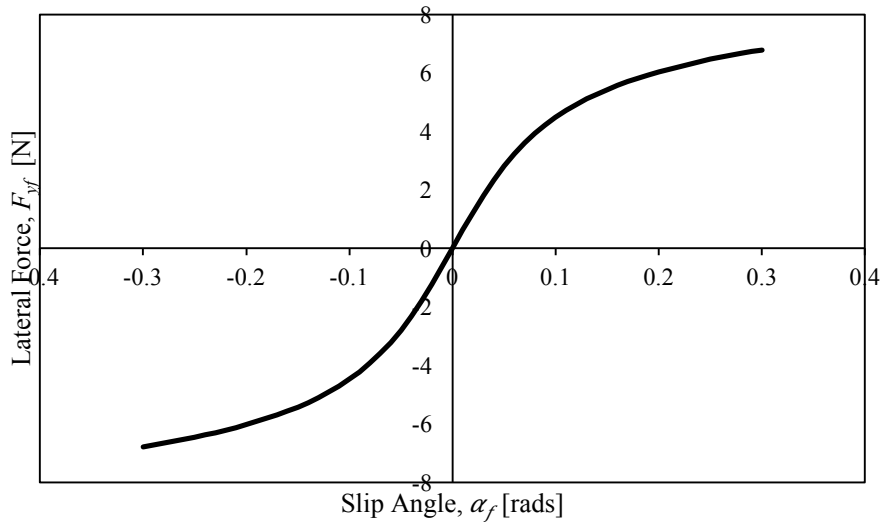


Figure 22: Lateral tire force as a function of slip angle for Traxxas slash tires. The curve is generated using Pajecka's magic formula and data referenced from Thomas Fitzgerald.

6.2.3 Longitudinal Forces

The longitudinal forces at each of the tires is what drives a vehicle forward. The drivetrain is responsible for transferring the mechanical energy generated by the motor to the wheels, which in turn is transmitted to the ground. Typically, vehicles will have some sort of gear reduction from the output driveshaft, that reduces the speed and increases the torque at each wheel. Our vehicle platform will not have a gear reduction from the motors, which means that the output speed of the motor shaft is equal to the speed of the wheel it is driving. Used in conjunction with data processed from the IMU, a slip ratio can be calculated with the following two equations:

$$\sigma_x = \frac{r_{tire}\omega_{wheel} - V_x}{r_{tire}\omega_{wheel}} \quad (\text{Acceleration})$$

$$\sigma_x = \frac{r_{tire}\omega_{wheel} - V_x}{V_x} \quad (\text{Braking})$$

Similar to the lateral tire mechanics, a force is generated at small slip angles that is linearly proportional to the slip ratio. If the longitudinal stiffness, c_σ , is known, the longitudinal tire force can be calculated.

$$F_x = c_\sigma \sigma_x$$

Typically, the only provided data is the speed of the output shaft of the motor, rather than the torque. This means that the generated tire force cannot be directly calculated and this relationship will need to be utilized. The slip ratio would be a readily calculated variable on our vehicle platform, as the IMU will be able to measure the absolute velocity of the vehicle and the tachometers in the motors can quantify the wheel speeds. However, the longitudinal stiffness still needs to be quantified in order for this relationship to be useful. Estimating this value for the Traxxas Slash would require extensive testing and falls outside the scope of our project.

6.2.4 The Understeer Gradient

At low speeds, a vehicle travels in the direction of the steering angle. At high speeds, a slip angle is generated that produces a force proportional to the cornering stiffness of the tire. An important parameter in vehicle design is the understeer gradient. This value helps to define the type of handling a vehicle will have. The understeer gradient is defined as:

$$K_V = \left(\frac{m_f}{2c_{\alpha f}} - \frac{m_r}{2c_{\alpha r}} \right)$$

where m_f and m_r are the mass transfers to the front and rear axle, respectively. Typical consumer vehicles have a positive understeer gradient, meaning that the driver has to turn the wheel more in order to travel the intended path. This makes complete sense from a safety standpoint, because it would minimize the risk of sharp, accidental turns that could potentially be dangerous. This is also something to take into account when modifying the center of gravity on the Traxxas Slash. Since all four tires have the same cornering coefficient, the only way to ensure the understeer gradient is negative is by positioning the weight further back from the center of the vehicle. This shifts the weight to the rear axle, resulting in an understeer governed vehicle. Our vehicle would be classified as having a positive understeer gradient, as the CG is positioned slightly forward of center.

6.2.5 Preliminary Vehicle Simulation

The equations of motion are very useful for developing vehicle models that can predict the motion of the vehicle. Even though the bicycle model greatly simplifies the overall system equations, further simplifications can be made develop a more basic simulation. Assuming a constant longitudinal velocity removes the longitudinal parameters, resulting in two first order differential equations. Substituting the lateral tire forces for functions of the slip angles and rearranging the equations to solve for the highest order derivative (See Appendix E for details) allows us to develop a state-space model [30]:

$$\frac{d}{dt} \begin{Bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2(C_{\alpha f} + C_{\alpha r})}{mV_x} & 0 & -V_x - \frac{2(C_{\alpha f}l_f - C_{\alpha r}l_r)}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2(C_{\alpha f}l_f - C_{\alpha r}l_r)}{I_z V_x} & 0 & -\frac{2(C_{\alpha f}l_f^2 + C_{\alpha r}l_r^2)}{I_z V_x} \end{bmatrix} \begin{Bmatrix} y \\ \dot{y} \\ \psi \\ \dot{\psi} \end{Bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2l_f C_{\alpha f}}{I_z} \end{bmatrix} \delta$$

The output from the state-space model gives positions and velocities in the vehicles coordinate system. These outputs are similar to values we would see coming directly from the IMU, as it measures the vehicle's acceleration and can be integrated to get the position and velocity. Figure 23 shows the vehicle response to a given steering input using the state space model. For more details and other results from the simulation, see Appendix E. Results from this simulation can be compared to tests with our completed vehicle to tune the vehicle parameters.

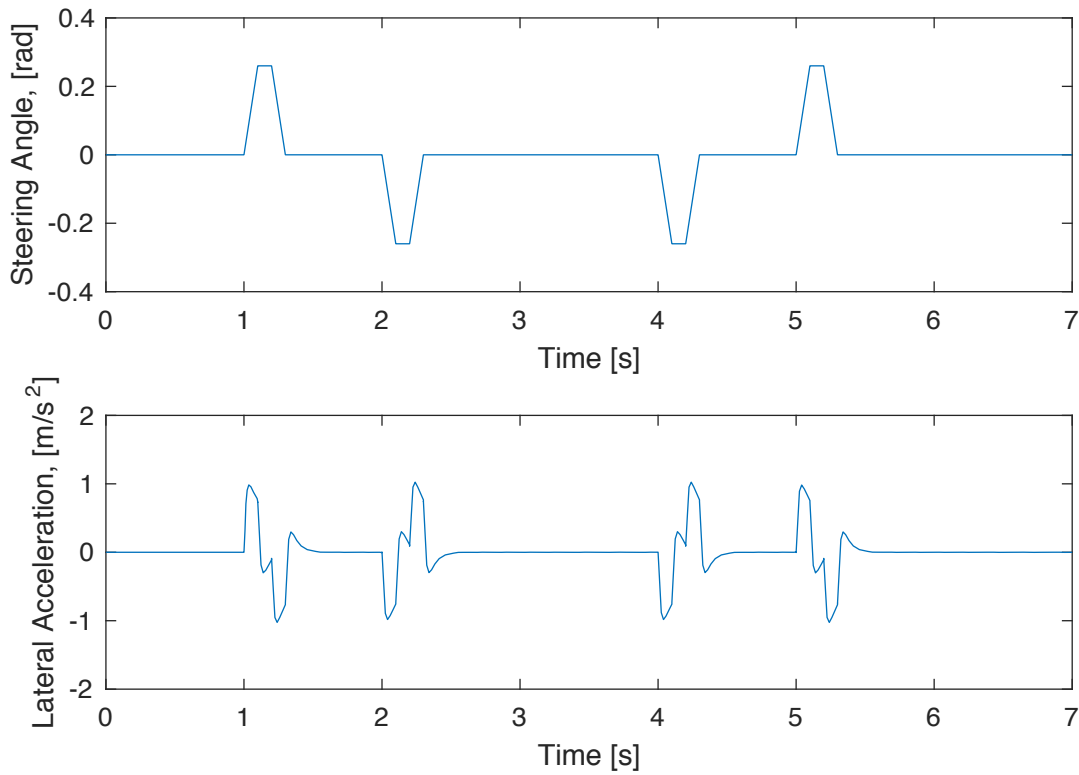


Figure 23: Steering response for vehicle simulation. Top plot shows the input steering angle and the bottom plot shows the lateral acceleration.

If path planning is important, a coordinate transformation can be used to change to the global coordinate system:

$$X = x\cos(-\varphi) + y\sin(-\varphi)$$

$$Y = -x\sin(-\varphi) + y\cos(-\varphi)$$

The coordinate transformation is useful for visualizing the anticipated path based upon the applied steering angle. For the plotted results from a sample vehicle simulation, refer to Appendix E.

6.2.6 Turning Radius

Another important vehicle parameter to estimate based upon our design is the turning radius. Increasing the wheelbase and track naturally increases the turning radius, which is determined by:

$$R = \frac{T}{2} + \frac{L}{\sin(\delta)}$$

Our final design limited the maximum steering angle due to the inclusion of the motors. We estimated that the maximum angle decreased from 25 degrees to 20 degrees. Furthermore, our overall vehicle dimensions increased. As a result, the turning radius increased from about 900 mm to 1300 mm. This seems like a substantial change, but our vehicle is also a 1/7th scale dimensionally rather than 1/10th. The Ford Edge has a curb to curb turning radius of 38.6 ft [31] – or in 1/7th scale 1675 mm. This means that our model actually has better turning capabilities than necessary, even despite the steering angle reduction.

6.3 CHASSIS DESIGN

6.3.1 Similitude

The first design decision with the chassis was deciding upon a wheelbase that would be realistic compared to our new track width. Overall, our track width increased 33.6mm compared to the original Traxxas Slash. This required a brief similitude analysis. A standard vehicle SUV has a wheelbase to track width ratio of between 1.7 and 2.0. Increasing the track width meant we would need to lengthen the wheelbase to maintain similitude. Our updated track width is approximately 250 mm, which corresponds to a wheelbase of at least 425 mm. Furthermore, typical nylon sheets come in lengths of 12 in (~305mm), which would result in a wheelbase of 420mm with the chassis mounts. For simplicity, a 300 mm long chassis would suffice. This would result in an aspect ratio of 1.7 for a 1/7th scaled vehicle to an SUV, such as the Ford Edge.

6.3.2 Design for Stiffness

The overall stiffness of a vehicle is dependent upon the tires, suspension and suspension members, and the chassis. The interplay between these elements can be simplified into a stick model as shown in Figure 24 [32]. The central chassis is modeled as a torsional spring and the suspension members are simple longitudinal springs. Treating the springs as acting in series, meaning that the total deflection of the chassis is the sum of the deflection of each element, the following equation is used to determine the total torsional stiffness of the vehicle:

$$\frac{1}{K_{Total}} = \frac{1}{K_{Chassis}} + \frac{Track^2}{K_{Suspension}} + \frac{Track^2}{K_{Tire}}$$

Note that this equation converts the longitudinal stiffness of the springs to a torsional stiffness in order to maintain consistency between the spring constants. Furthermore, including the tires as another term will account for their contribution to the overall stiffness of the vehicle. This can also be done for the suspension members, however, it could be difficult to approximate values for these parameters, so they are assumed to be rigid.

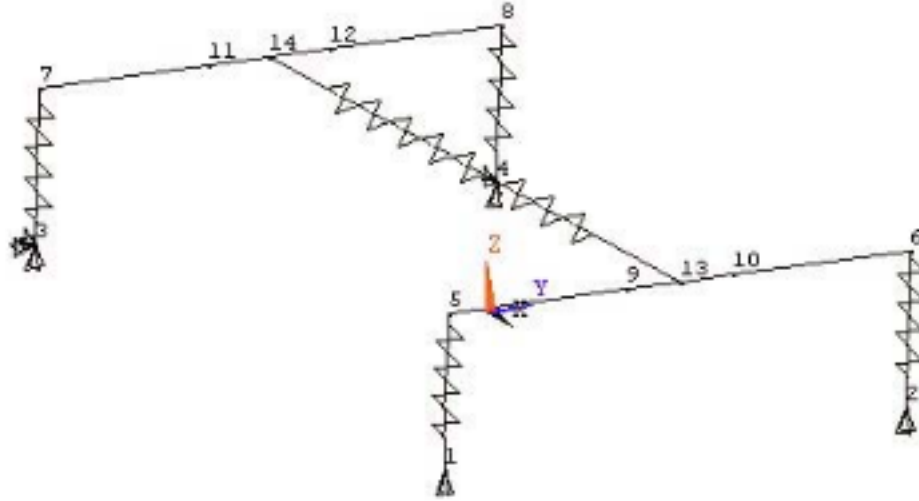


Figure 24: Vehicle stick model used to predict overall vehicle compliance

Ideally, the chassis is designed to be as stiff as possible in order to allow the suspension to be the primary contributor to shock absorption. However, weight considerations typically mean that compromises are made that increase the overall compliance of the vehicle. Weight is not a primary concern for the purpose of our project. In fact, a heavier vehicle would represent an actual vehicle more accurately. The only concern regarding weight is that if the chassis is designed too thick and additional components add a sufficient amount of weight, the suspension would bottom out too easily. Once again, keeping in mind the end goal for this project, the vehicle will not likely be used in scenarios where the full travel from the suspension is necessary.

An important simplifying assumption for the chassis design is that it is a flat plate in the shape of a rectangle. A complex geometry would mean that a computer software running a finite element code would need to be utilized. In reality, chassis design is not a rectangular shape, but for the purposes of this design, we can assume it is a rectangle. Torsional rigidity is a shape's ability to resist torsion. It is similar to the moment of inertia, in fact, it even has the same units. Unfortunately, there is not a closed form solution to solve for the torsional rigidity of a rectangle. The exact solution for torsional rigidity of a rectangle is given as [33]:

$$K_{Rect} = \frac{a^3 b}{3} \left[1 - \frac{192 a}{\pi^5 b} \sum_{n=1}^{\infty} \frac{1}{(2n-1)^5} \tanh\left(\frac{\pi(2n-1)b}{2a}\right) \right]$$

where we assume a is the average width of the chassis and b is the thickness, which we would like to solve for. This equation can be used to predict the effective torsional stiffness of the chassis through the relationship:

$$K_{Torsion} = \frac{K_{rect}G}{L}$$

where G is the shear modulus of the material used and L is the total length of the chassis.

The measured spring constant for the suspension is 0.821 N/mm. This was calculated by measuring the unsprung length and then placing a mass (1.675 kg) on the spring and measuring the deflection. This methodology assumes the relationship between deflection and force is linear. Calculations for this section of the report were completed with a 305 mm track width and 250 mm chassis width, which is not the same as the final design parameters, but provides an accurate estimate for design purposes.

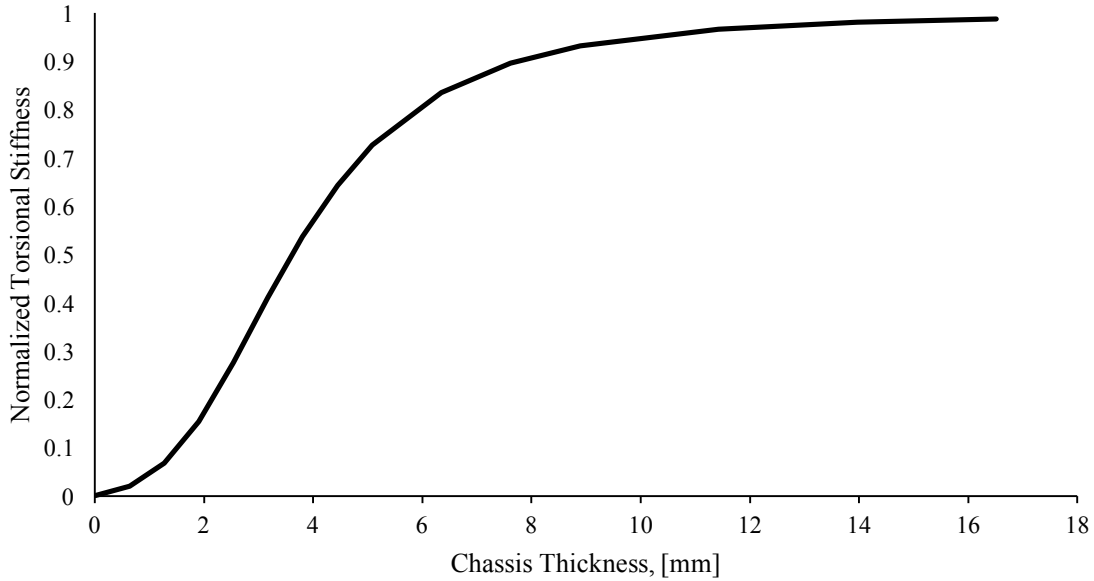


Figure 25: Normalized torsional stiffness of the entire vehicle as a function of chassis thickness.

For our purposes, the spring constants are a set value that need to be designed around. This means that any additional compliance added to the system by the chassis should not significantly affect the overall compliance of the vehicle. A simple way to ensure this was to plot the total torsional stiffness of the vehicle as a function of the chassis stiffness—which was determined by the thickness. This can be seen in Figure 25. It should be noted that the stiffness was normalized to where “1” represents a completely rigid chassis. A completely rigid chassis would allow for the suspension, tires and suspension members to dictate how the vehicle responds to ground disturbances, but may not be feasible from a design standpoint. A normalized stiffness value of about 0.85 is a more realistic value, as it results in a chassis that is about 8 mm thick.

6.3.3 Chassis Mount Design

Mounting the chassis directly to the top of the motors would have put the chassis height at approximately 125 mm. This would have been a very high position to mount the chassis and would have lifted the steering servo above the driveline plane. As a result, we chose to step the chassis down using aluminum sheet metal. However, we needed to ensure that the gage of sheet metal would be thick enough as to not compromise

the structural integrity of the chassis itself. From the previous section, we concluded that 8mm thick nylon would only reduce the overall stiffness of the vehicle by 15%. Aluminum is much stiffer than nylon, however we need to use a reasonable gage of sheet metal so that it can still be bent with a sheet metal bending brake.

Comparing the flexural rigidity of the aluminum sheet metal cross section with the nylon chassis cross section would give us a good idea of how the two components would bend relative to each other. If the two rigidities are similar, we would expect the two materials to behave similar to each other. That is, the overall compliance of the chassis would not be reduced by adding the sheet metal mounting brackets. The flexural rigidity of a rectangular cross section is defined as:

$$\text{Flexural Rigidity} = \frac{1}{12}bh^3 * E_f$$

where E_f is the flexural modulus (tensile modulus of isotropic materials). The flexural rigidity of the 8mm x 330 mm nylon cross section is about 56 N-m². The necessary thickness of an aluminum cross section 64mm wide to match this value is about 5 mm thick. We chose to use two sheets of 2mm thick sheet aluminum to clamp the chassis. Although this is would essentially be just 4mm the difference is still small enough as to not play a large role in decreasing the overall stiffness of the chassis. Furthermore, the geometry of the bent sheet metal and the clamping structure only furthers the bending stiffness, as more material is distributed further away from the neutral axis, effectively increasing the moment of inertia.

6.4 SHAFT COUPLER DESIGN

6.4.1 Concept Generation

We have two primary requirements for the shaft couplers – they must act as a u-joint compatible with the Traxxas drive-shafts and they should occupy as little depth along the motor shaft as possible. This allows us to minimize further widening of the track width. We initially looked at set screw couplers and split couplers, but found that those would not be feasible. Since there must be a way to get the u-joint on to the fixed ball of the Traxxas shaft, we had to opt for a split design as shown in Figure 26.

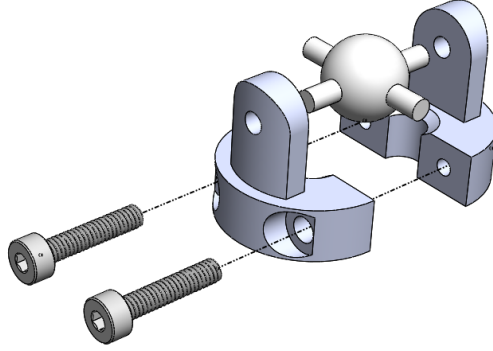


Figure 26: Exploded view of shaft coupler converging on the u-joint ball.

The clamping force generated by the tightening the screws will be sufficient for the transfer of torque between the shaft and coupler. This will nullify the need to modify the motor shafts, saving significant manufacturing time as well as eliminating the possibility of damaging any of the expensive motors.

6.4.2 Failure Analysis

The driveline yokes that the original Traxxas Slash came assembled with were made out of a heavy duty molded plastic. The shaft couplers that we are implementing will be made out of aluminum. Using aluminum will simplify both the analysis and manufacturing process for the shaft couplers. Aluminum is a stronger and stiffer material than the plastic used by Traxxas, which means we can assume the coupler design will not fail under normal, or even more extreme operating conditions. We anticipate that the most failure prone piece of the assembly will be the clamping screws. These will need to be able to support the torque in the shaft, while still be pre-stressed to effectively transfer torque between the motor shaft and yoke.

The chosen fastener to generate the clamping force on the coupler is an M2x0.5 10 mm screw. We chose this screw because it fits the compact profile of the coupler and it is the appropriate length to maximize the thread surface area that will be clamping the two coupler pieces together. However, calculations were needed to ensure that the resultant stress in each screw would not exceed the proof stress at maximum torque. The first assumption to make for the analysis was that the clamping screws would be tightened to 75% of their proof strength. This is a common loading condition for non-permanent fasteners [34]. This generates a clamping force of 1000 N in each screw, plenty sufficient for effective power transmission between the coupled shafts.

The clamped material also carries a portion of the load. The portion of the load carried by the fastener is:

$$C = \frac{K_b}{K_b + K_m}$$

where K_b and K_m are the bolt and member stiffnesses, respectively. The applied load, P (different from the preload), was based upon the maximum torque output from the Maxon motor. The resultant stress in the motor with this loading condition was found with the following equation, which sums the preload and applied external load from the motor:

$$\sigma_b = \frac{CP + F_i}{A_t}$$

The proof stress of the M2 screw is about 650 MPa. The loading condition above approximated the load to be 495 MPa, where the majority of the load comes from the clamping force. This fact means that a fatigue analysis was not necessary, as the compressive resultant on one side of the rotation would not make a significant difference in the loading conditions.

A supplementary analysis was conducted to determine the feasibility of creating shaft couplers out of a 3D printed plastic. Figure 27 shows the deformed finite element results using ABAQUS to determine the maximum normal stress that may occur in the coupler arm. The PolyJet printer uses FullCure 720 plastic with a yield strength of approximately 60.5 MPa. Even in the unmodified design, the highest normal stress is about 46 MPa, which is reduced to 38 MPa for the improved design. The improved design increases the width of the coupler arm and fillets the corners to reduce the severity of the stress concentration. It should also be noted that the couplers will be in a torsional stress state. The reported von Mises stress for the unmodified coupler is about 58 MPa, which is still below the yield stress of the material.

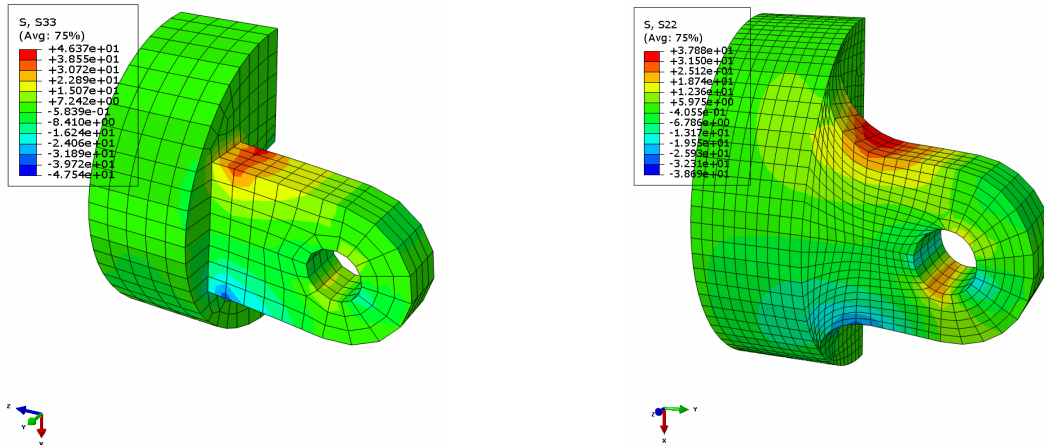


Figure 27. Shaft coupler FEA results for unmodified (left) and modified coupler designs (right).

6.5 DESIGN ANALYSIS OF LOAD BEARING MEMBERS

Modifying the mounting components for the suspension, motor, and steering members requires an analysis that takes into account high impact loads. Large cycles are not expected for members other than the shaft coupler, meaning we could forego a fatigue analysis and simply ensure that the new components could support non-cyclic forces. The basis for our design was built around the motor housing blocks, where brackets for mounting the suspension turnbuckles, a-arm, and ultra-shock are attached. Working in a very small space meant that the brackets would be small and there is a limited amount of material to prevent the mounting bolts from tearing out. As a result, our main form of analysis was hole tear out in the direction of minimum material thickness.

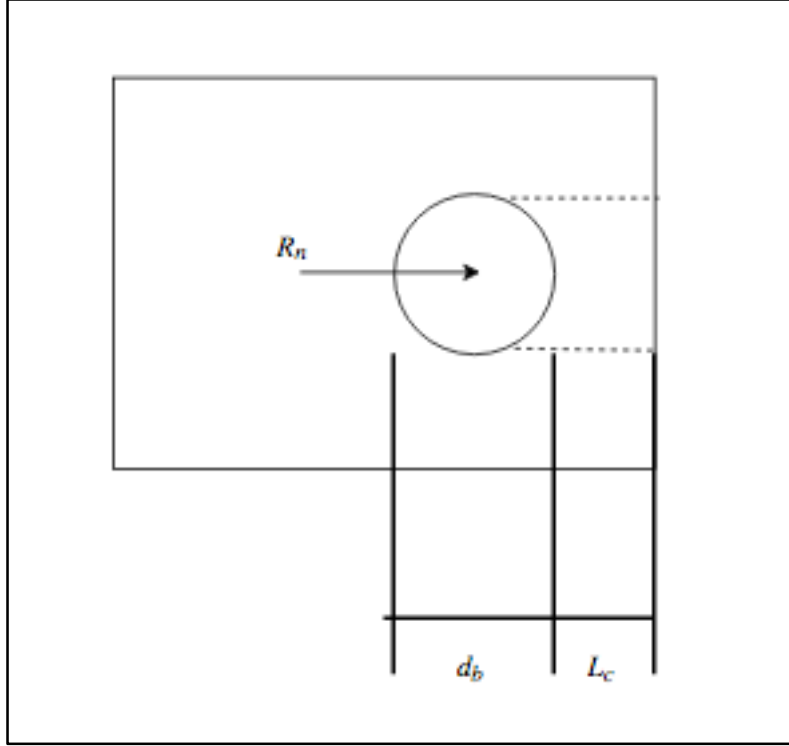


Figure 28: Hole tear-out path and critical dimensions.

A typical rule to follow when designing joints that are exposed to a shear force is $L_c = 1.5d_b$. If this rule is followed the primary mode of failure is the bolt shearing. As mentioned earlier, we were not able to design to this specification, meaning that we need to ensure the joints will not fail through shear of the aluminum mounts. R_n , the maximum resistive force before the joint failed, is applied in the direction of minimum material between the hole and parts edge. The equation of the limiting state for bolt bearing where the hole deformation is not a concern is provided by Bartlett Quimby [35]:

$$R_N = 1.5L_c t F_u$$

where t is the material thickness and F_u is the ultimate shear strength of the member. Results for the maximum force before hole tear out in each member can be seen Table 9. Also listed are the estimated loads and a resulting factor of safety. The loads were estimated based upon a large vertical deceleration and the vehicle mass. It is impossible to know exactly what magnitude of forces will actually occur in the joints, so seeing such large factors of safety is good. This will result in the robust and sturdy design we are looking for.

Table 9. Results from hole tear out calculations for custom components.

	Member	L_C	t	F_{Max}	F_{approx}	FS
		[mm]	[mm]	[N]	[N]	[-]
Rear	Turnbuckle Mount - A	2.3	2	1100	87	12.6
	Turnbuckle Mount - B	1.0	9.0	2150	174	12.3
	Ultra-shock Mount	3.4	8.0	6400	150	42.8
	A-Arm Mount	2.3	1.0	549	87	6.3
Front	Suspension Turnbuckle	2.1	2.0	980	87	11.3
	Ultra-shock Mount - A	2.5	4.0	2390	146	16.4
	Ultra-shock Mount - B	2.3	4.0	2200	146	15.1
	A-Arm Mount	2.3	1.0	550	87	6.3

After this analysis it becomes clear that majority of the members would not fail despite the relatively low hole spacing from the edges. We could increase the safety factors by simply increasing the thickness of the mounting surface or lengthening the distance of the edge from the hole. Nonetheless, the lowest F.S. is 6.3, which is still a very large safety factor even based off the ultimate strength.

6.6 MAXON MOTOR CURVES AND HEAT CONSIDERATIONS

Maxon provides data for the motor specifications on their website. This data needs to be compared to the vehicle system and parameters to ensure that the motors can produce the required amount of power for our purposes. Charlie Refvem generated a set of system curves for the Traxxas slash, as well as the Maxon performance curves based upon the provided data from the specifications sheet. These curves can be seen plotted in Figure 29. The vehicle system curves account for losses due to kinetic friction, drag, an incline angle of degrees, driveline inefficiencies and electrical inefficiencies. The assumptions made in this model are conservative, meaning we would expect to require less power than what is calculated. It should be noted that our desired maximum vehicle speed of 15 MPH corresponds to a motor speed of approximately 2000 RPM. The motors will allow us to continually operate at this speed without overheating the windings in the motor.

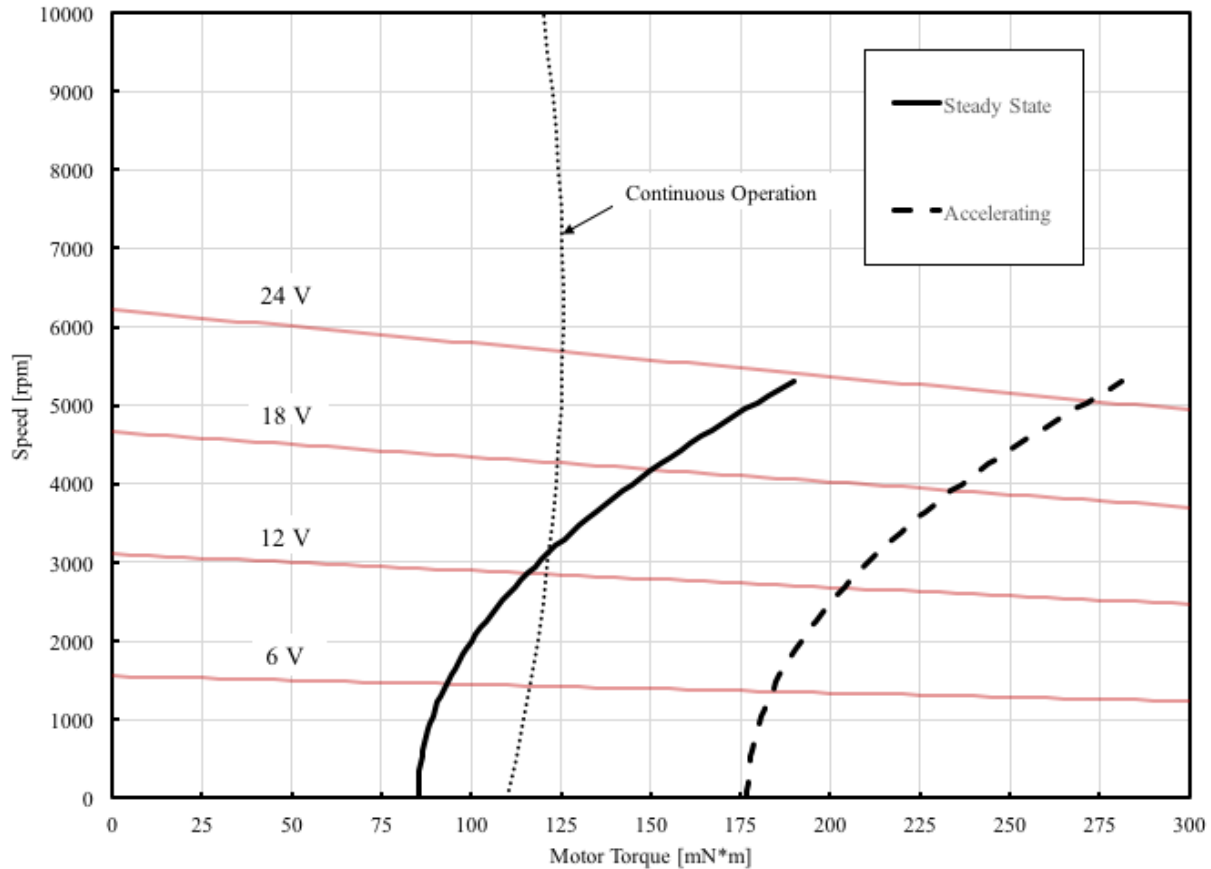


Figure 29: Maxon performance curves, steady state and acceleration system curves.

Our motor housing design mounts the motors back-to-back, with essentially eliminates one of the surfaces for heat transfer to the surrounding environment. The curves above do not account for this for continuous operation. A more accurate model would push the continuous operation curve to the left, limiting the power output that the motors can operate at before overheating. This fact alone may warrant the need to include some sort of a heat sink between the motors to better disperse the heat. We noted earlier that the assumptions made to develop the system curves were conservative, which would also mean that a more accurate model would be pushed further to left. For this reason and the fact that a heat sink would widen the track width even more, we do not believe heat dissipation will be much cause for concern. The Maxon motors are being operated well below their limits, which means overheating will be unlikely.

6.7 UNDERSTANDING THE EXISTING SYSTEM FOR FUTURE DESIGN

Everything up until this point has been focused on what functionality the electrical system will have and specifying the general requirements needed to provide that functionality. This is the part of the project where we can get into detailed selection and design of the components, like wiring diagrams and types of digital communication, and actually specify every connection and component in the system.

Before we can get detailed, we need a scope and a big picture of the project to work with. While we were on break we took a lot of time thinking of the system as a whole and constructing a mental image of the

components we might need. After creating this mental picture, we researched the system of the Traxxas Slash vehicle we had chosen to work with. The Traxxas control system is as shown in Figure 30.

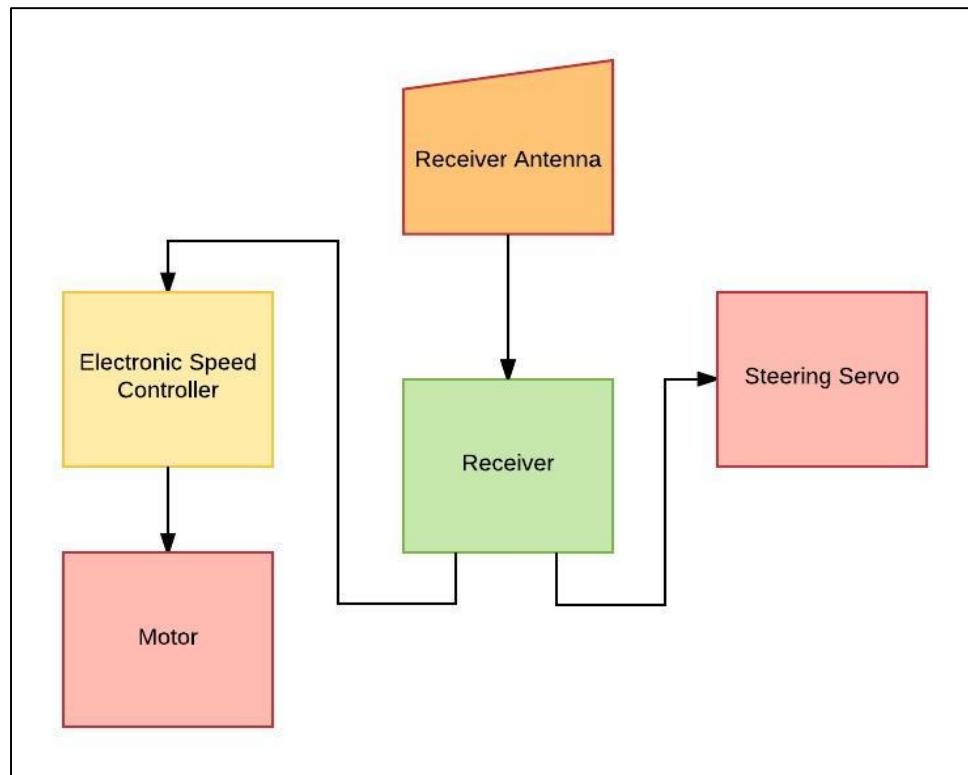


Figure 30: Existing Traxxas Slash Flow of Control

Luckily, not much time ended up going by until we could get our hands on the physical vehicle we ordered and we could spend time looking into the actual signals being sent. The first logical part to look at is the receiver as this is the component that starts all internal signals. We figured out which wires were which and hooked the output signal and ground signal up to an oscilloscope. As expected, we saw a Pulse Width Modulation (PWM) signal in the control wires that were associated with the motor/speed controller and the steering servo. The oscilloscope outputs from the signals are presented below in Figure 31.

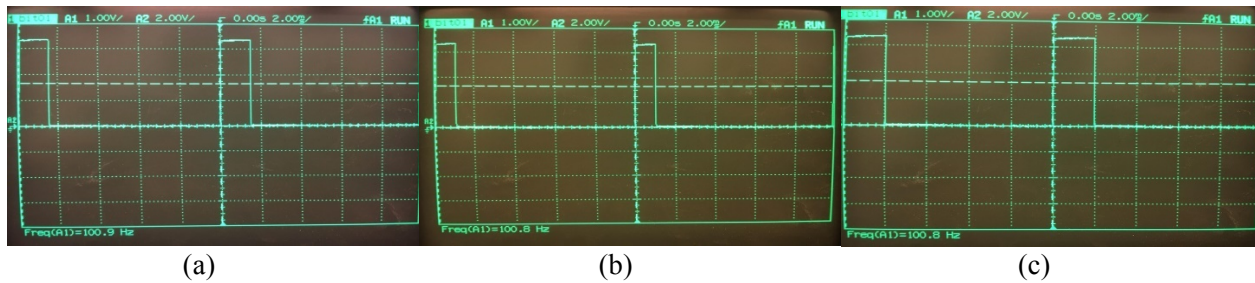


Figure 31: Signal from receiver to ESC at neutral throttle (a), 15% duty cycle, reverse (b), 10% duty cycle, and forward (c), 20% duty cycle.

As indicated, these signals operate on a 10ms period ranging from a 10% duty cycle to a 20% duty cycle. These signals are also the same ones sent to the steering servo with the 10% duty cycle indicting a hard left turn and a 20% duty cycle indicating a hard right turn. After analyzing the signals from the Traxxas Slash, we can interpret this data for our own control regimen.

6.8 CREATING THE NEW SYSTEM

When reinventing a system on a machine, we’ve learned through previous endeavors it is best to use as much of the old system as possible. For our project, we’ve already listed the necessity of using new motors and their motor controllers to power the vehicle. Other items already required in our new system is the Raspberry Pi 3, a motherboard, a camera sensor, IMU, and ultrasonic range finder. So all in all we have new motors selected, a new “brain trust” of the Raspberry Pi and its motherboard, and new sensors. Other criteria crucial to the system but haven’t had a need to be replaced yet include the steering servo, the receiver, and its antenna. Since we’ve seen no reason to replace these parts and we have verified they will work with the new system, we will go ahead and plan to include them in the new system. If you’re reading this and wondering why he haven’t chosen the motherboard yet, this is because we are waiting to plan out all the communication protocols and the specific pin arrangements we will need the motherboard to work with before we choose one that may or may not have the characteristics we are looking for. We have included Figure 26 to depict the power and data connections in our system.

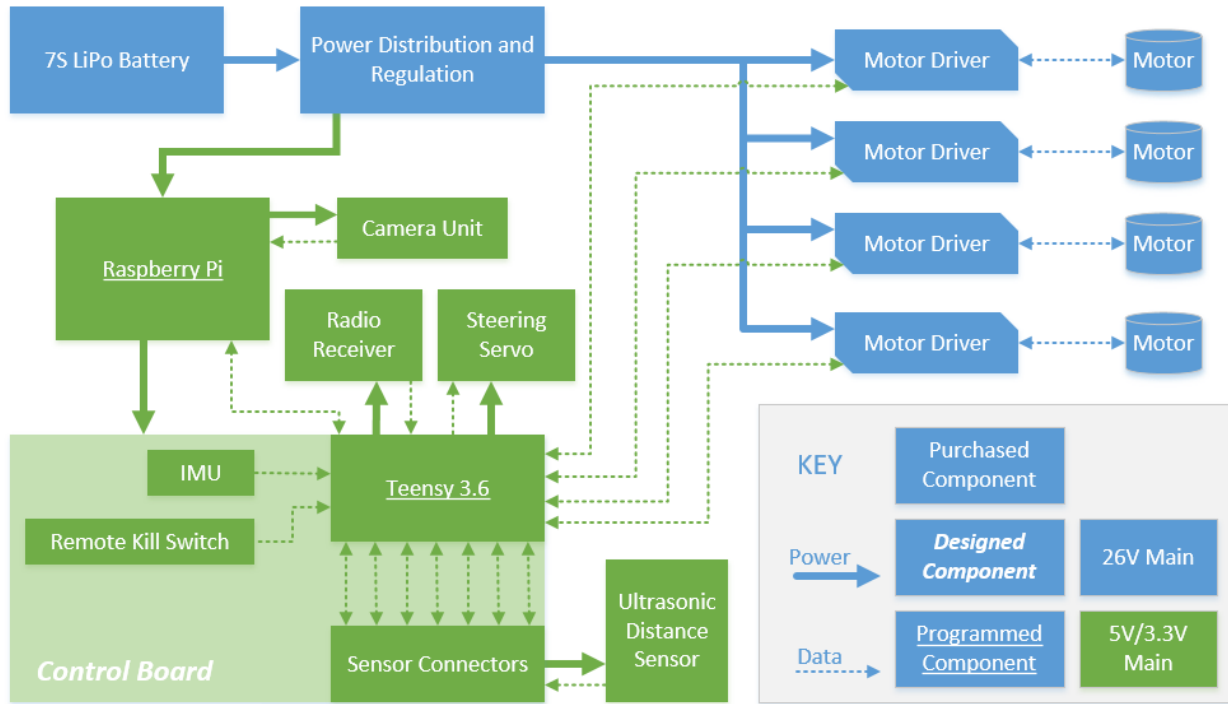


Figure 32: Completed Component Selection and electrical layout, with voltages and component information.

Now that we have chosen all the parts we will need we can proceed with planning the communication system between the electrical components. The existing servo will need to use PWM protocols as we discovered in the previous section. The receiver will obviously output a PWM signal for both the servo and motor that will need to be read by the motherboard. The IMU we have selected requires the use of the I²C communication protocol. The ultrasonic range sensor we picked out uses Analog protocols to communicate with the motherboard. The camera will attach directly to the Raspberry Pi using a specialized Raspberry Pi connector. The last communication we need to scope out before we pick which motherboard we need is the motor controllers to the motherboard. We talked about using SPI, PWM, or Analog (in a worst case scenario) for the motor controller. However, after a meeting with Maxon who will be sponsoring the motors and motor controller, we were able to upgrade motor controllers to one that is able to use CAN protocols. CAN is widely used in the automotive industry when they construct their full-vehicle communication systems and we thought that using the CAN protocol in our system would improve our platform for both learning purposes and for making a more reliable and flexible communication system.

Our motherboard microcontroller will now need to be able to use CAN, SPI, Analog, I²C, and PWM protocols. It will also need to accommodate our requirement of 15 pins available for data usage. We identified a few microcontrollers which seemed appropriate for our project including the Arduino Mega and the Teensy 3.2. We decided to go with the Teensy. We found that the Teensy had good documentation, ran on 3.3V (which matches the voltage level of the Raspberry Pi for a simpler circuit), is a very small part, and ran at a decently high clock rate to go along with the fact that it had enough pins for our system and can run all the communication protocols we listed above. We also decided to upgrade to the Teensy 3.6 device for the higher clock speed and having more pins for future sensor connections. The Teensy 3.6 and 3.2 are very similar in design and the Teensy 3.6 is almost an exact extension of the 3.2 version board.

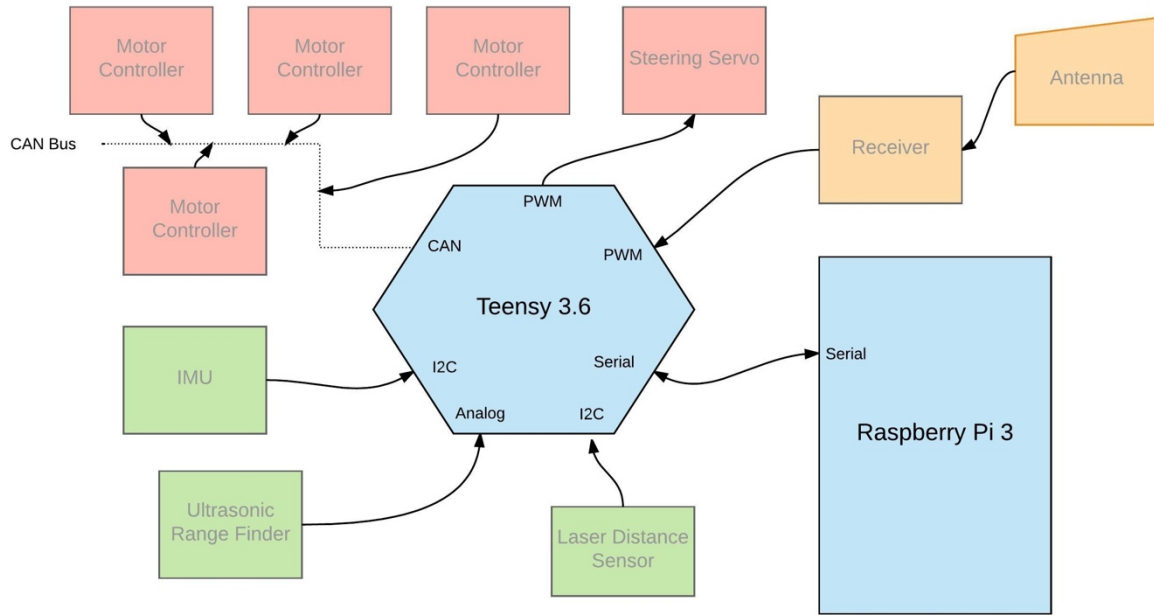


Figure 33: Design of the new system from a communication standpoint

6.9 SOFTWARE DESIGN

Software planning is the next logical step in the design process. We will have two main components to program in our project: the Teensy and the Raspberry Pi. The Teensy will mostly act as a signal handler for the Raspberry Pi but has some important tasks. The Raspberry Pi on the other hand will take the data available to it (through its connection with the PiCam and the sensor data that is passed along by the Teensy), calculate the appropriate actions to take, if any need to be taken, and output the result back to the Teensy for redistribution. The Teensy will then give the steering signal to the steering servo through PWM communication and the motor signal to the motor controllers through the CAN Bus protocols. Through looking at the specific needs of each input/output signal we developed a list of tasks the Teensy will need to do.

There are a couple options of how to implement the software in the Teensy. The first is a basic while loop running in main that continuously performs our tasks by calling functions and updating its data as is appropriate. The second idea is to implement a real time operating system in the Teensy and incorporate a scheduler that splits the different functionality needed into tasks which each gets their own amount of time to run. To implement a real-time operating system and scheduler is no easy task. It also didn't seem very appropriate for the Teensy's tasks.

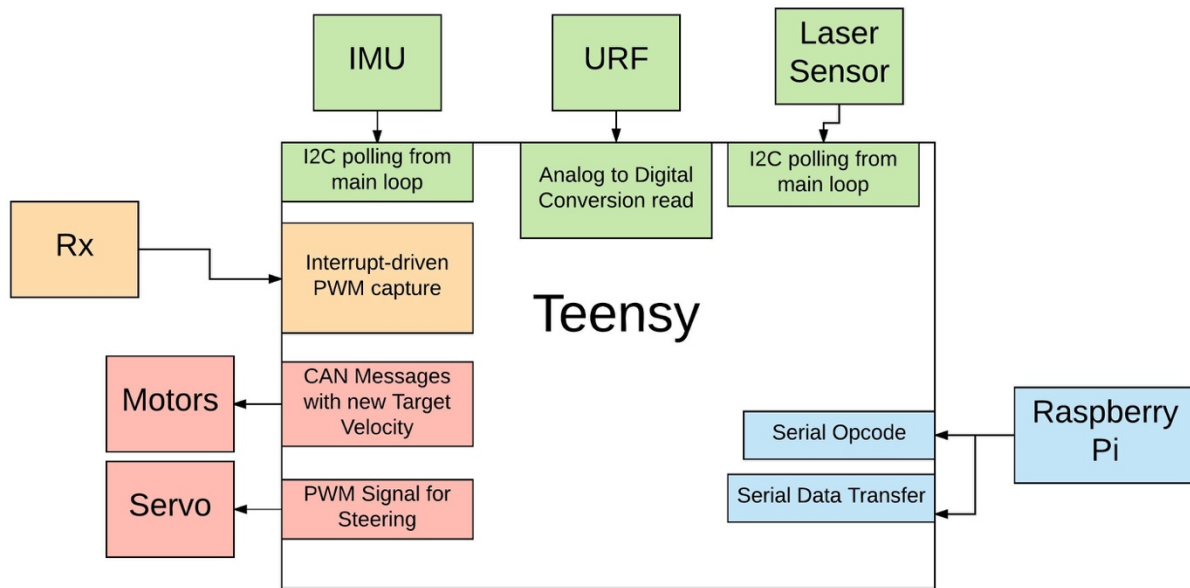


Figure 34: Teensy Task Decomposition

In Figure 28 we depicted a scenario of using the loop configuration for our software. All the tasks were possible and seemed fairly straight-forward in this manner. With a loop configuration in mind, the teensy will keep busy but won't be overloaded. We ultimately chose to incorporate the loop model because it helps simplify the chances for error while also creating a more straightforward approach. In the loop we communicate with the Raspberry Pi, motor controllers, IMU, and URF or laser sensor. It's a bit to do but we are sure that the Teensy can handle at least this much operating at 180 Mhz.

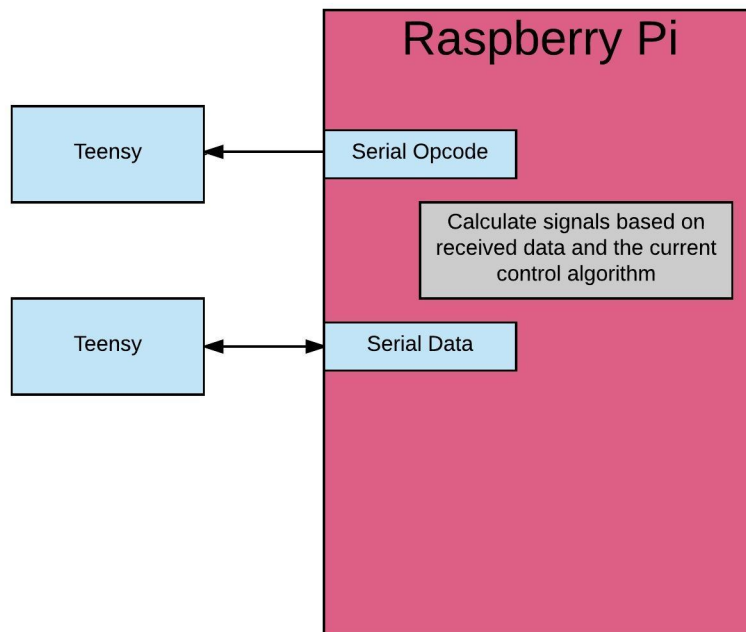


Figure 35: Raspberry Pi Software

The Raspberry Pi is more straightforward. Our responsibility is to create a SPI protocol which happens every loop cycle and to update the data we receive to specific variables. Most of the hard programming will be left up to the user to design and implement their own control algorithms.

7. MANUFACTURING

Over the winter and spring quarters, we worked to concurrently complete manufacture and assembly and fulfill the ESV competition expectations. From CDR, we had approximately a month to prepare for the ESV evaluation. We continued broad project work while creating the demo, and the demo supported implementation of our final design. Figure 36 below presents our planned timeline from CDR to completion.

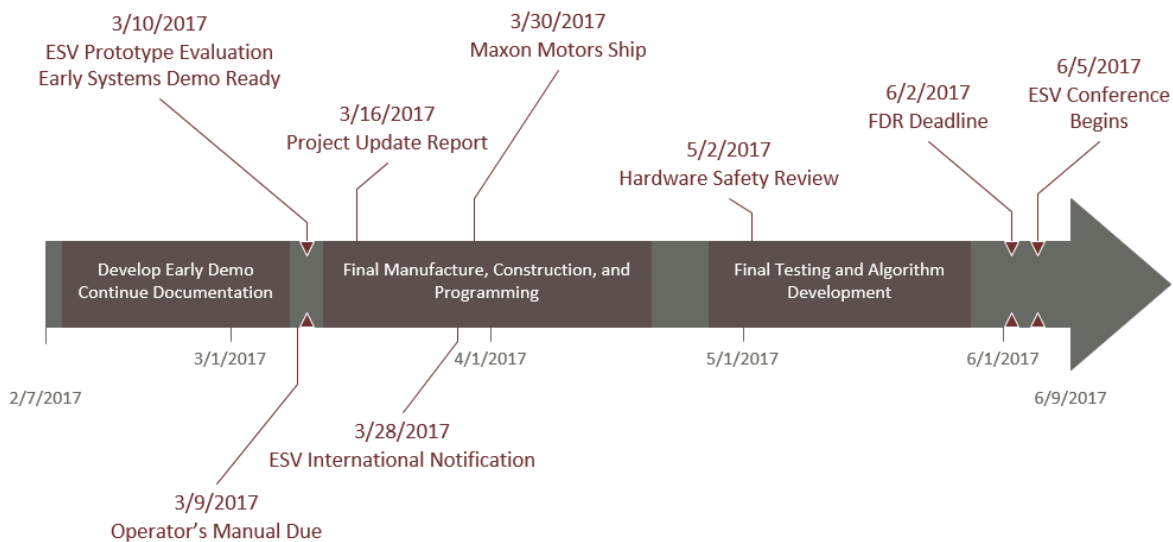


Figure 36: Planning timeline from CDR, including critical dates, milestones and deliverables.

7.1 ESV SYSTEMS DEMO

It would have been prohibitively difficult for us to manufacture the entire prototype in a month, and the Maxon motors were backordered until late March. Because of this, we did not have a full system demonstration for the ESV regional review. We intended to instead demonstrate that we will have a fully functional prototype by the ESV conference. We showed the judges that our system could communicate with hardware via the Matlab environment, and demonstrated responsive steering and throttle based on the orientation of the car.

7.2 SOURCING

We ordered the Traxxas Slash, Raspberry Pi, Sensors, and electronic prototyping equipment at the end of winter quarter and have been working with those parts to inform our design. The Maxon motors and drivers are ordered but are not scheduled to ship until the end of March. Through the partnership that Charlie has formed with Maxon, we ordered the motors and drivers at a significantly discounted cost. By February 20th, we anticipate having the rest of our parts and materials ordered. We ordered the motherboard from OSH Park, and received three unpopulated copies of the board for five dollars per square inch. The components for the motherboard were ordered from Digikey, and the raw material for manufacturing was ordered from McMaster Carr. The full bill of materials describes the exact source location for each item.

7.3 BUDGET AND BILL OF MATERIALS

Calculating the expenses of specific component costs and providing estimates for components not currently specified totals the project cost at \$1502.72. We factored in tax rates and estimated shipping costs along with all material costs into our total. We used the standard California tax rate of approximately eight percent

for budget calculations. For shipping costs, we used an estimate of twelve percent of the material cost. As a non-profit project, we have been awarded \$350.00 from the Mechanical Engineering Student Fee Allocation Committee (MESFAC) of Cal Poly. We were also awarded a grant from CPConnect in the form of \$500.00. Our sponsor Dr. Birdsong also has \$1000.00 set aside for this project in the event we need extra funding. Figure 37 below shows the allocated funds for our product and the final project cost. The full Bill of Materials can be found in Appendix I.

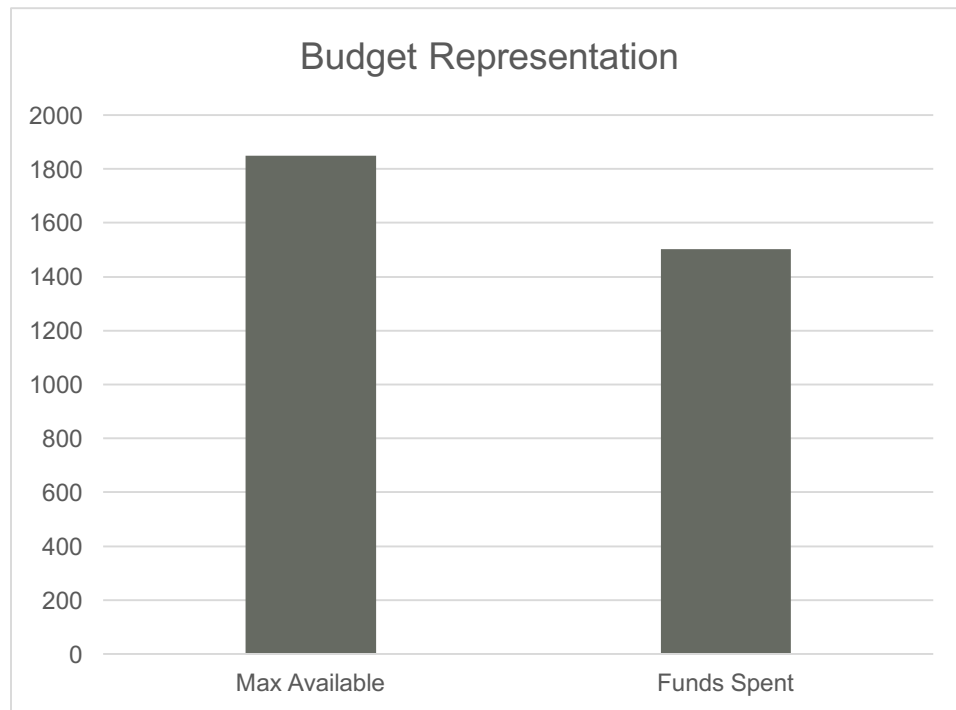


Figure 37: Current budget compared to PDR expectations and total available budget.

7.4 MANUFACTURING

The mechanical parts were designed in such a way that as few parts as possible must be manufactured with specialized tooling and knowledge. Save for the motor housing and shaft couplers which must be produced using a CNC mill, everything can be manufactured using manual machine tools. During the manufacturing phase, Jay will be working on tooling and manufacturing the CNC parts while Chris works on the remaining simplified parts.

While the Cal Poly machine shops have limited tooling available and it is often worn out, it is recommended that projects acquire their own drills. For our purposes, we will have to acquire or borrow the tooling presented in Table 10.

Table 10: Tooling required to manufacture the custom components.

Tooling Type
M2 pre-tap drill
M2 tap
M3 pre-tap drill
M3 tap
M3 clearance drill

The full set of manufacturing drawings can be found in Appendix J.

7.4.1 Motor Housing

The motor housings were 3-D printed out of PLA. The designs were updated to incorporate room for heat set inserts to be bonded in the plastic. The inserts are M3 brass inserts that are installed via a soldering iron with a special tip for pressing them into the plastic. The soldering iron heats the conductive metal and melts the plastic around the insert. This provides a bond that would be stronger than simply tapping the plastic and threading a screw. Originally, our plan was to CNC motor housings, but 3D printing greatly simplifies the manufacturing process while not degrading the overall build quality. As a part of the shift to 3D printing, we also changed the designs slightly. The updated drawings are present in Appendix J.

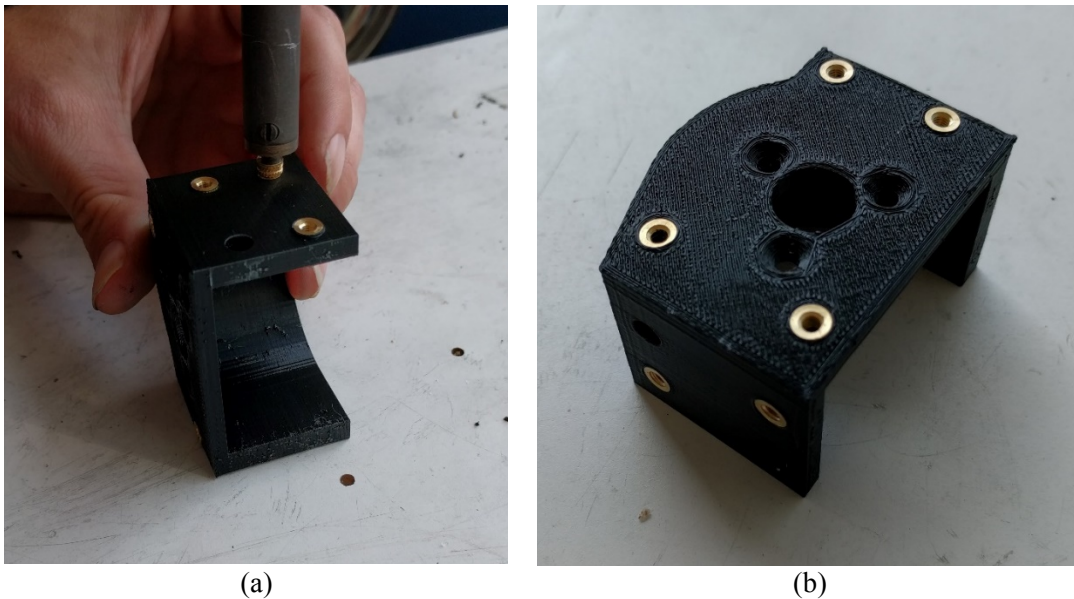


Figure 38: Heat setting threaded inserts into the motor housings using the soldering iron tip (a). A motor housing with most of the inserts in place (b).

The installation of the heat set inserts worked very well, with the inserts providing more than adequate strength and quality of attachment. While the PLA is not extremely strong, it is more than adequate for

the small loads that it must bear. If there are issues with durability, the same models can be adapted for CNC machine easily, as they were initially designed with machining in mind.

7.4.2 Shaft Couplers

The shaft couplers are made of two similar parts differing only in hole pattern, and will be manufactured on the CNC using three operations. Starting with rectangular stock, the net half circle and u joint upright is shaped while a portion of the original stock is held in the vice. The part is flipped over into a set of custom soft jaws with a mating half circle cutout on one side of the vice, and the rest of the original stock is removed. The part is turned semicircle up and clamped in another portion of the soft jaws that locates the semicircle and u joint post, and the necessary holes are drilled to complete the part. With a properly designed set of soft jaws, multiple parts can be in the CNC at once, increasing the turnover speed and reducing setup overhead. We will be using identical shaft couplers on all four motors. The assembled shaft coupler can be found in drawing 225, and the component halves in 225A and 225B.

Since we 3D printed the motor housings, we were able to repurpose the stock intended for that purpose for the soft-jaws, which are shown in the CNC in Figure 39.

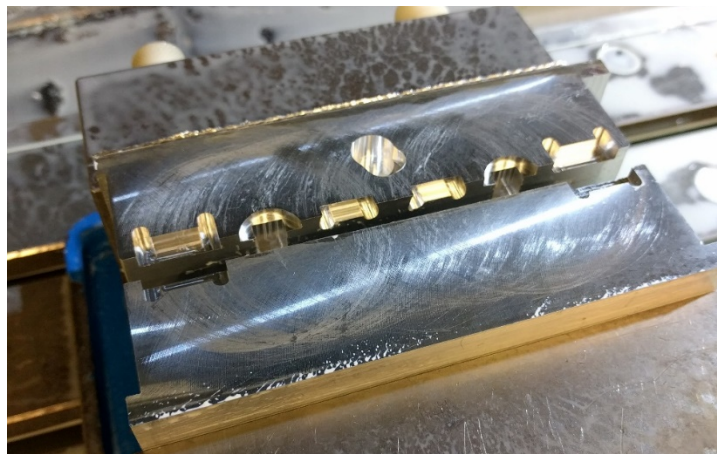
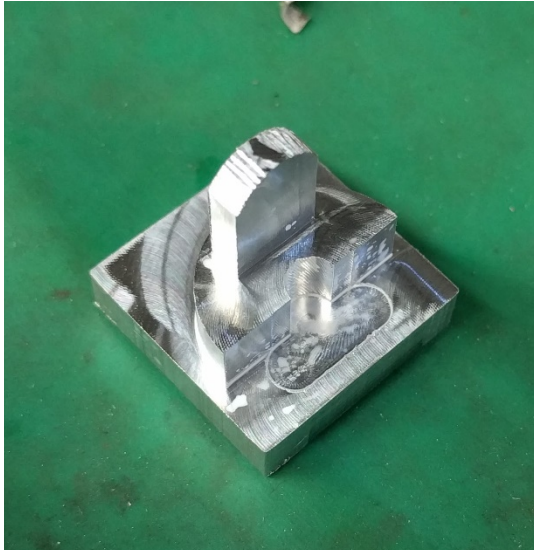


Figure 39: Finished soft-jaws for manufacturing the custom shaft couplers.

Both halves of a coupler can be produced at once, with the first operation being on the furthest outside, the next being the middle, and the final operation being the closest to the middle. This organization serves two purposes. First, ensuring that there are parts symmetrical across the jaw centerline during each operation is important. The most material removal happens in the first operation, so that operation should take place at the edges of the softjaws, where the material can support the most moment induced by the cutting tool. The shaft couplers can be seen in two different stages of the manufacturing process in Figure 40.



(a)



(b)

Figure 40: Shaft coupler half after the first operation (a). Both halves in the soft-jaws after the final operation (b).

During the first pair of halves, we were focused on verifying the toolpath to prevent crashes. We found that the softjaw position shown in Figure 40b was too tight of a fit, so we reduced the outer diameter of the next coupler pairs by a few thousandths of an inch. After finishing the first pair, we assembled it on the motor with the Traxxas slip-yoke to confirm that the geometry and tolerances were properly set. This successful test fit is shown in Figure 41.

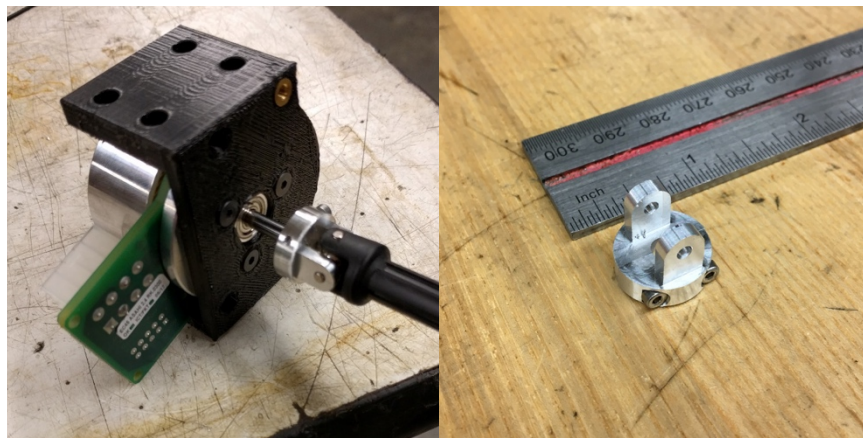


Figure 41: Test fits of the first shaft coupler, completed before manufacturing the rest.

We confirmed that the geometry and tolerances were up to par, and were ready to proceed with the other three pairs of coupler halves. Since we had already confirmed the toolpath, we were able to proceed with manufacturing the rest of the couplers extremely quickly. Each set took approximately 10 minutes to produce after installing tooling and verifying toolpaths. For future manufacturing runs of shaft couplers with an experienced CNC tech, one should expect about an hour of tooling set-up and approximately 10 minutes per coupler pair.

7.4.3 Steering Posts

The steering posts were the only part that required a lathe to manufacture. We started with a 12" long 7mm round stock and cut it to about 0.25" longer than the required length using the vertical band saw. Next, each post was faced to the necessary length. The outer diameter was achieved by turning just over half the length down to 5mm, and then turning the other side down to the same size. This operation could be omitted by simply purchasing 5mm tight tolerance rod and only facing it to length. The lathe would still need to be utilized for drilling through the center of the post, as well as tapping each side.

7.4.4 Suspension System Mounts

The A-Arm and turnbuckle mounts have very simple geometry that only required a few different operations. First, $\frac{3}{4}$ " x $\frac{3}{4}$ " aluminum stock was cut to about 200 thousandths of an inch larger than the width. The Bridgeport mill (~1800 RPM) was then used to cut the stock to the appropriate length, width and height. Excess material was then removed to reach the final shape of each respective mount. Next, clearance and pre-tap holes were made on the drill press using the purchased drill bits. For the turnbuckle mounts, the necessary holes were tapped with M3 thread and tap magic cutting fluid. The A-arm mount dowel slot was drilled using the M3 clearance drill bit (~3.20 mm OD, #30). This provided a slight clearance between the dowel and inner diameter of the hole. Chamfered edges were made by simply grinding the edges down with a belt sander and holding the part with a pair of vice-grips.



Figure 42. Suspension mounts manufactures on the Bridgeport mill (top left) and cut with the water jet (right). Manufacturing of front suspension mounts on Bridgeport mill (bottom).

The front and rear suspension mounts have a two-dimensional profile that allows them to be easily manufactured by the waterjet (Figure 42 – Top Right). The water jet uses a .dxf file created from the CAD model to cut the 5/16” thick aluminum stock. Pre-tap holes were made with a #40 drill bit and then tapped with M3 thread and tap magic. Clearance holes for mounting to the top of the chassis mounts were drilled with the M4 clearance drill.

7.4.5 Chassis

The base shape of the chassis was cut using the waterjet. Features included in this first operation was the outer profile, mounting holes for the motor housings, and the slot for the servo. The mill with a 3/16” end mill was used to cut a flat surface into the chassis where the servo could mount, allowing the servo horn not to interfere with the chassis. The mounting holes were made on the drill press and tapped with M3 thread. Slots also needed to be made on the underside of the chassis to allow space for the steering bellcranks. These were made with a 3/4” forstner bit, and then 1/4” forstner but drilled deeper to enclose the mounting screw.

As we were prototyping the arrangement of the components on the chassis, we realized that it would be more beneficial to mount the battery on the bottom of the car. This gives us the space to run cabling, and developing a bottom motor housing was a simple task.

The final modification of the chassis was drilling the mounting holes for the motor controllers, motherboard and IMU mount. To reduce the likelihood of error and potentially ruining the chassis, a template was cut out of acrylic with the laser cutter. This template allowed us to easily and accurately drill holes for all the components.

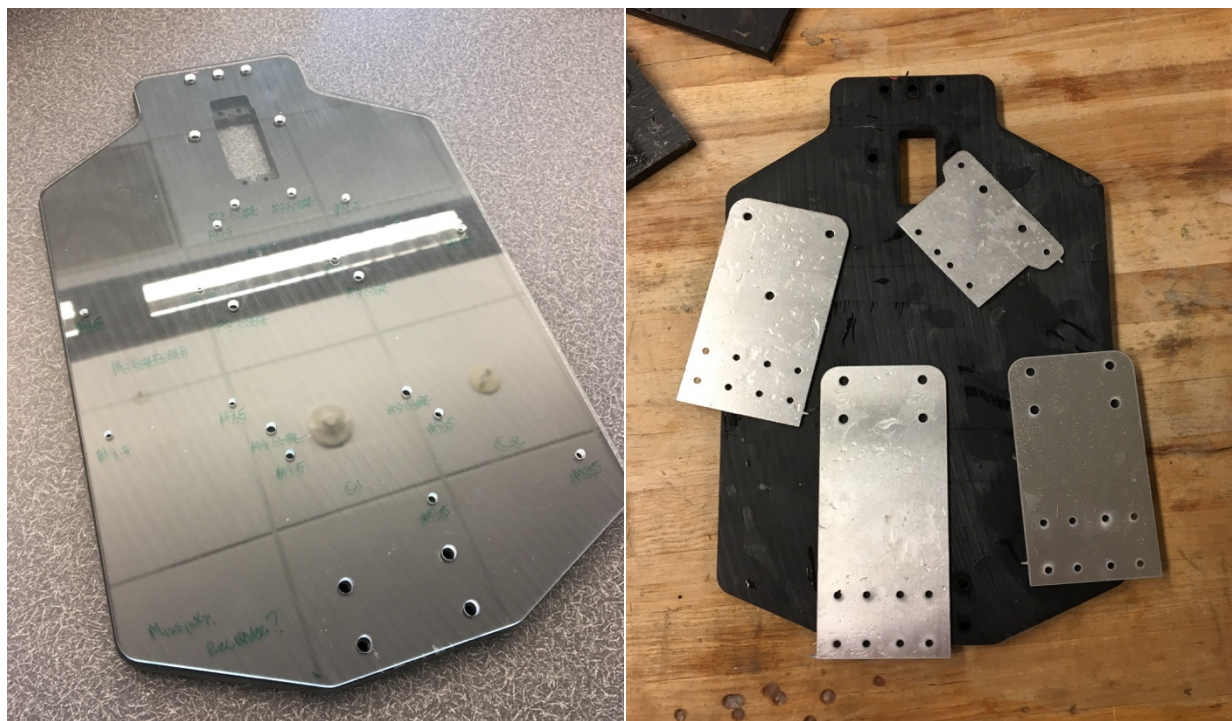


Figure 43. Chassis template made with the laser cutter and mounted to the chassis for hole locating (left) and 2D profiles cut on water jet for chassis and chassis mounts (right).

7.4.6 Chassis Mounts

The chassis mounts were made out of 0.08" sheet 3003 aluminum. The outer profiles and hole patterns were made using the flat pattern in Solidworks and exporting the face as a .dxf file. The profile was then cut with the water jet. The sheets were bent using the sheet metal bend brake. The flat top mounts are in drawings 204 and 304, while the bent bottom mounts are in drawings 203 and 303. Figure 43(right) shows the chassis mounts after they were cut with the water jet.

7.4.7 Sensor Mounts

Our design is extremely adaptable with plenty of space to grow. Currently, we have designed two sensor mounts. An IMU mount is located in the center of the chassis and is compatible with the BN0055 IMU. A sensor array is located at the front of the vehicle and contains a laser rangefinder, Raspberry Pi camera, and ultrasonic rangefinder Figure . The mounting holes on the top chassis mount can be used to incorporate a wide variety of custom designed sensor mounts in the future. New parts can be quickly designed and prototyped using rapid prototyping technology such as 3D printing.

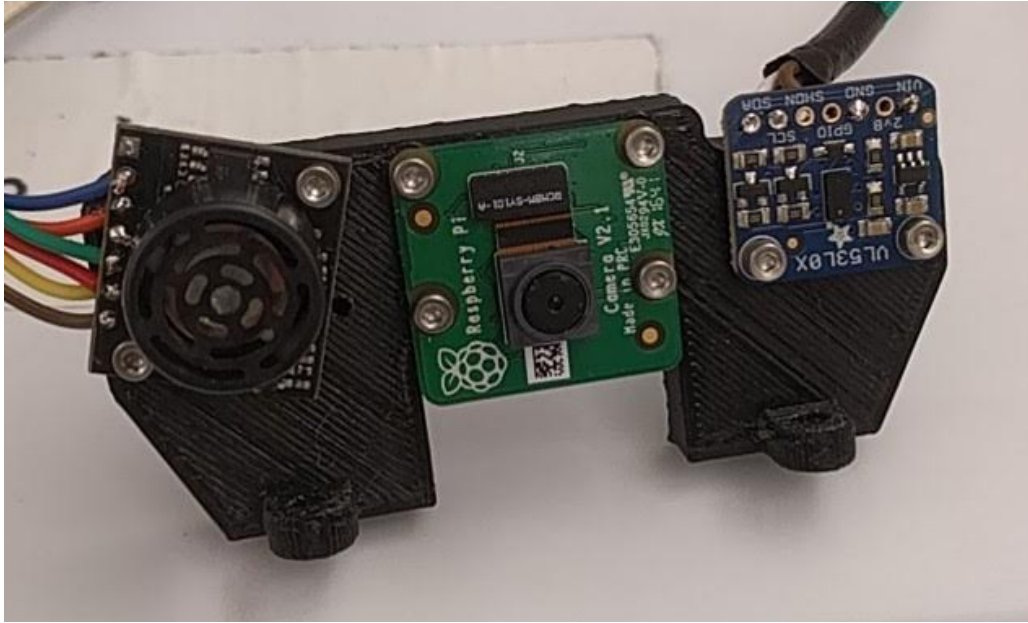


Figure 44: 3D printed sensor array featuring (from left to right) an ultrasonic distance sensor, raspberry pi camera, and laser range finder.

7.4.8 Motherboard

After designing the motherboard with the chips and connections we thought necessary to run the platform, we ordered a small prototype batch from OSH park, a US based community PCB batch ordering service. OSH Park accepts user designs and ships boards within 12 days of ordering. Pricing is \$5 per square in, and you get three copies of the board. Our board is 7.5 in², which costs \$37.50 plus tax. We sourced our components from Digikey. Components were be placed and soldered by hand in the mechatronics lab in building 192. The PCB has silkscreened labeled indicating orientation for all symmetric polarized components. The motherboard schematic is found in drawing 420, and the board layout is found in drawing 421. The components used on the motherboard are specified in drawings 422-431.

To get the first version of the motherboard working with all of our peripherals, we had to make several wire and resistor modifications to route signals differently than designed. These wire modifications are shown in Figure 44. The isolating-switching voltage regulator was designed to be mounted upside down, but the board footprint was improperly designed. For this version, we felt it was most important to get functionality like the CAN bus and serial interface working, which we successfully did.

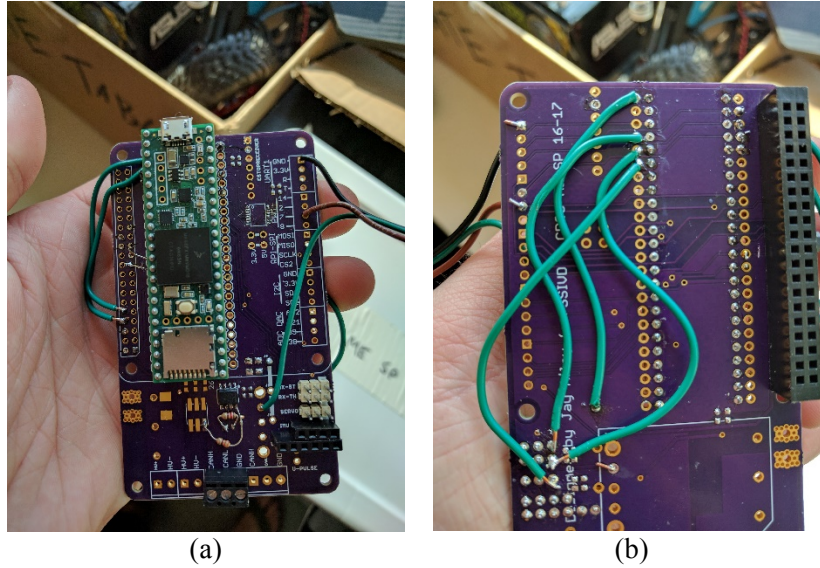


Figure 44: Wire modifications on top (a) and bottom (b) faces made to the first revision of the motherboard.

Based on the modifications we had to make to the motherboard, we redesigned certain aspects and ordered another batch from OSH park. In addition to resolving minor design and connection problems, we changed the RGB led to have a lower current so the Teensy pins could adequately sink/source the required current. We also added a power indicator LED, shown in Figure 45, which lights up when there is 3.3V coming from the Teensy. We also made the design safer by ensuring that the high voltage contacts were not easily shorted against each other, which was a problem in the first revision. We added a diode to prevent back-powering the switching regulator, and added through-holes under the input fuse, as we had torn the pads off one of the boards from the first revision. Before populating the active components, we thoroughly tested continuity across the power lines to ensure that no components would experience over-voltage or shorts.

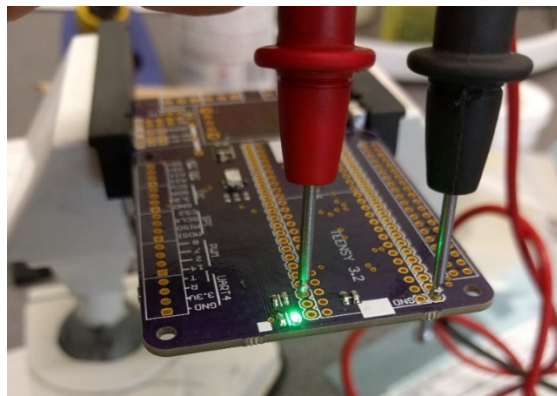


Figure 45: Probing the second iteration of the motherboard to ensure it is safe for integration of components and microcontrollers.

After soldering the components to the motherboard, we realized that the footprint had been designed improperly for the RGB indicator LED. In addition to the orientation being wrong, the footprint had been designed for a common-anode LED, while it was actually a common-cathode LED. By cutting some traces

and using small-gauge wire to route the correct voltage, we were able to get the indicator successfully integrated on the board.

During testing of the second revision of the motherboard, we had both the Teensy and Raspberry Pi connected to USB ports – somehow this faulted, and it appears that we overvoltage the Teensy, destroying that microcontroller. We believe that one of the USB supplies browned out, creating a voltage difference on the connected 5V lines from each USB cable. In addition to the 5V lines being connected, both processors have their own 3.3V regulators, which are then connected with a shared 3.3V line. To help prevent this from happening in the future, we severed the shared 3.3V line so that each processor had its own. We also are ensuring that only one USB is plugged in at a given time. Fortunately, we ordered a new Teensy to replace the fried one.

When testing the motherboard's capabilities to run the motors and microcontroller off of battery power alone, we noticed that the Raspberry Pi appeared to be browning out. With a multimeter, we found that the diode was causing a voltage drop, supplying only 4.76V nominally to the 5V line. The Raspberry Pi has a minimum input voltage of 4.75V, and any ripples in our output voltage were developing the brown-out. Fortunately, the motherboard had been designed with footprints to adjust the output of the regulator – by putting a 33k Ω resistor between the output voltage and the sense pin, we were able to tune the output voltage after the diode to 5.00V. After implementing this resistor, we no longer encountered issues with the Raspberry Pi browning out.

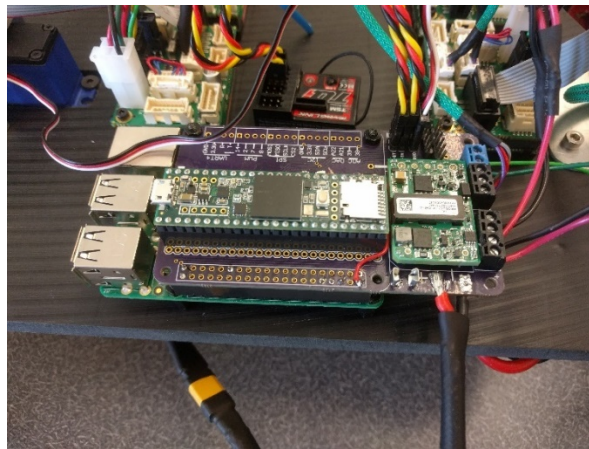


Figure 46: Final implementation of the motherboard, with Raspberry Pi and Teensy.

The motherboard performs excellently, and acts as a solid base to structure an intelligent vehicle research platform around. There are minor suggestions for further improvement, which are fully discussed in section -.

7.4.9 Wiring

To interface with our sensors and the motor drivers, we had to build a series of custom cables for our system. These cables are built with the green expandable sleeving to protect the wires and create visual organization on our final system since there are many cables that will have to be routed all over the car. At both ends of the expandable sleeving we incorporated heat-shrink to add strain relief and prevent fraying. We first build the cables to interface with the motor drivers. With the motor drivers, Maxon includes a spring-loaded

connector for power & ground. We created a spliced y-cable to connect two motor drivers to each of the high-voltage output screw terminals. The wires were first soldered together, then wrapped tightly in high gage bus wire and soldered again – this allows for better strain relief on a linear splice, which can be a common point of failure if improperly done. We had to create four sets of cables for the motors, which connect to the drivers with two cables each – a 10pin ribbon IDC cable and an asymmetrical cable to connect the motor coils and hall sensors. The asymmetrical cable has an 8pos Molex Mini-fit jr cable connecting to the motor, 4pos Molex Mini-fit jr cable connecting the motor coils to the driver, and a 6pos Molex Microfit 3.0 cable connecting the hall sensors to the driver. The connectors and crimp pins for the motor cable are shown below in Figure 47.



Figure 47: Connectors and crimp pins for the motor to driver cable harness.

While unintuitive to have an asymmetrical cable, it allows the motor driver to support DC and BLDC motors in both sensed and sensorless configurations. The completed motor to driver cable is shown in Figure 48.



Figure 48: Fully assembled motor to driver cable harness.

To connect the CAN bus of the drivers and the motherboard, we had to develop another set of cables. 3 symmetric cables with 4pos Molex Clik-Mate 1.5mm connect the drivers to one CAN bus, and a Clik-Mate to exposed stranded wire connects the driver bus to the motherboard. An unfinished CAN cable is shown in Figure 49, breaking out CAN high / low via a twisted pair and breaking out a ground line.



Figure 49: Open ended CAN connector, with twisted wire pair for CAN high and low.

For each of our sensors, we soldered wires to the sensor boards, and broke them out to a female header row. These female header rows plug directly in to male header on the motherboard, so the order of the cables must be correct. The wire arrangement for all of the motor driver cables is shown in Drawing 450.

7.5 ASSEMBLY

Part of the idea behind our design is that the motor blocks can be removed from the chassis. This separates our design into three main sections: the electronics, the chassis, and the motor blocks. Everything is attached with non-permanent mechanical fasteners. The suspension system is first connected to the motor blocks to create four different subassemblies. The motor housing can be joined together through the chassis and suspension mounts, where the chassis mounts then connect the chassis to the motor blocks. It is anticipated that all electrical components will be wired on the chassis before mounting to the rest of the assembly. However, they can also be assembled after the motor blocks are mounted. The overall design allows for easy access to the electrical components on the vehicle. Figure 50 shows the compartmentalized design that allows for easy modifications for future designs.

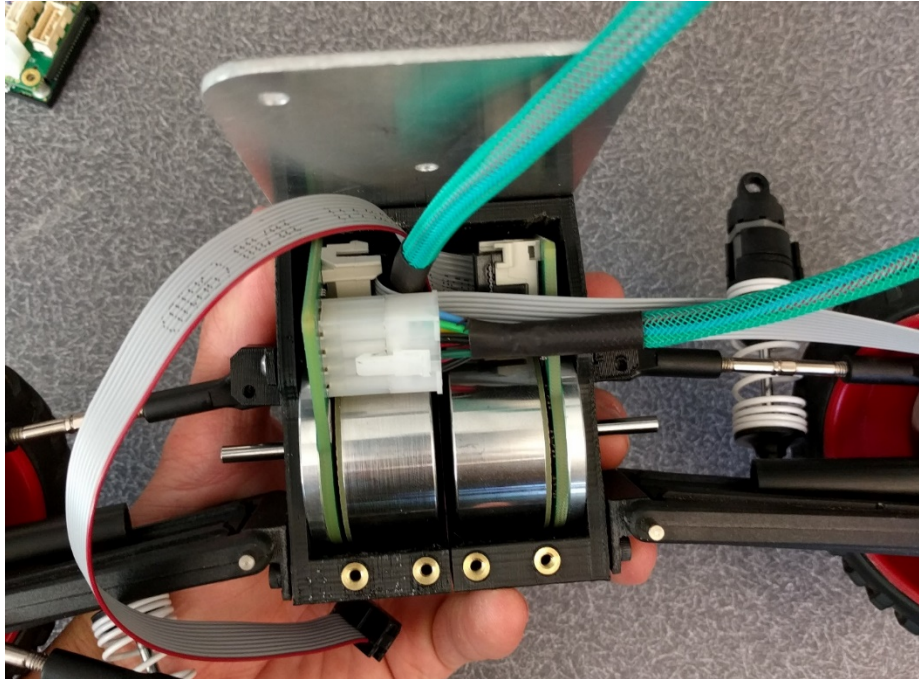


Figure 50: Motor integration on the front drivetrain.

The final assembly, shown in Figure 51 is sturdy and durable.

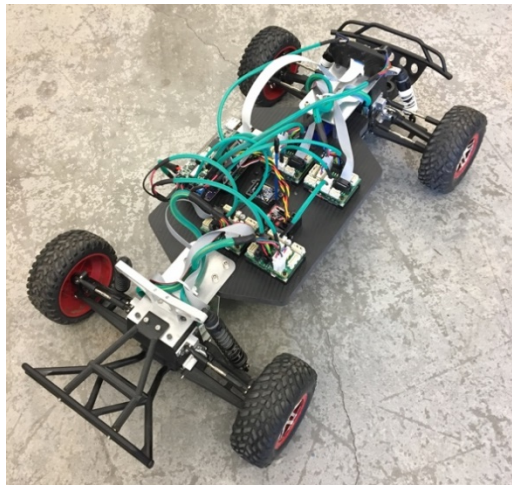


Figure 51: Fully assembled vehicle

7.6 HARDWARE SAFETY CONSIDERATIONS

In developing our system, we have identified that the primary safety concerns lie with the motion of the platform, the battery, and the electrical system. The motion of the system is the intended use, so it is difficult to ease the safety concerns with that, but we have limited the speed and acceleration in the motor drivers to that of our engineering specifications (2.5ft/s^2 and 10mph). For the battery, we use a good balancing charger that monitors individual cell voltage and can indicate poor battery health. For charging and storage,

we have a fire-resistant Li-Po bag. We also configured the firmware to enter a fault state if the battery voltage drops too low. Electrically, we have several fast-acting fuses throughout the system. We have a 30A fuse directly after the battery, meaning any short in the system will result in a cutoff of the battery power. On the motherboard, we have a 6A fuse in-line with our switching power supply, preventing damage and overdraw on our low voltage system. Each motor driver comes with a 10A fuse to handle any possible shorts or hardware failures. The multiple layers of fuses ensure that if something goes wrong anywhere along the system that we can handle it without damaging other parts of the system. It also allows us to be confident that we will not damage our hardware or hurt the end user through electrical shorts.

7.7 SOFTWARE BREAKDOWN

7.7.1 Teensy Software File Layout

There are many files attached to the main Arduino loop, `Teensy_Firmware`. In this section we will go over which files you need to be aware of and which you can ignore.

IMU: (From a documented Library)

The BNO055 takes up quite a few files. `Adafruit_BNO055.cpp`, `Adafruit_BNO055.h`, `Adafruit_Sensor.h`, `imumaths.h`, `matrix.h`, `structs.h`, `quaternion.h`, `vector.h`, `BNO.cpp`, and `BNO.h` all have to do with the IMU. If you need to look into how the IMU is handled start with `BNO.cpp` and `BNO.h` as these are the files that incorporate all the rest.

Indicator LED: (User made files)

`Fault_handler.cpp` and `fault_handler.h` hold the functions we use to initialize and set the on-board LED. There's not too much here, although if you're looking to change the LED color check out our pre-made selection in `fault_handler.h`.

Throttle/Steering Input and URF Distance Sensor : (User made files)

All handling of the radio receiver's PWM input is handled through `input_handler.cpp` and `input_handler.h`. Here you will find the interrupt that catches the starting timestamp of the PWM and the handling of the calculated difference between that and the ending. We limit the throttle input from -500 to 500 and the servo input from -400 to 400. The URF sensor's initializing and polling functions are also handled here.

Servo Output: (User made files)

Sending a PWM pulse to the servo happens through `output_handler.cpp` and `output_handler.h`. The useful functions here are `initServo` and `writeServo`.

CAN Library: (From a documented Library)

`FlexCAN.cpp`, `FlexCAN.h`, and `kinetis_flexcan.h` provide a CAN library that allowed us to simply call functions to start the Teensy's CAN bus and to read and send the CAN messages. All CAN library support comes from these two files.

Motor Controller Interfacing: (User made files)

Building from the CAN library in FlexCAN.cpp, uLaren_CAN_Driver.cpp and uLaren_CAN_Driver.h have the heavy task of creating all the specific CAN functionality to use the motor controllers. Setting up their network, initializing them, setting the target velocity and much more is found in these files.

Laser Sensor (From a documented Library)

Adafruit_VL53L0X.cpp, Adafruit_VL53L0X.h, and the twenty other files that start with "vl53l0x" all provide support for the VL53L0X Laser Sensor. The only notable files are the Adafruit_VL53L0X.cpp and the Adafruit_VL53L0X.h. The other files support these two.

Main Loop Support (User made files)

As the system is right now, we have one file that supports the main loop, loop.h. This file contains the states the main loop is allowed to cycle through.

7.7.2 Teensy Main Loop Brief

There are two critical things to know that pertain to the main loop file Teensy_Firmware: the state transitions and the global variables used. We'll start with explaining the state transitions.

INITIALIZE_PERIPHERALS

In this state we initialize any sensors we wish to use. The indicator led gets set to white here. Once we're done here we move on to INITIALIZE_CONTROLLERS.

INITIALIZE_CONTROLLERS

This state is used specifically for the Maxon motor controllers. We start by resetting the nodes in case they were previously in a fault state. We continue by initializing the CAN network for every node (motor controller) on the network and going around one by one and initializing them to various modes and settings. The notable settings are changing to Profile Velocity Mode and turning the controller into the Switched-On state. The code in uLaren_CAN_Driver.cpp is well documented for all settings. Once initialized, the code moves on to the WAIT_FOR_ARM state.

WAIT_FOR_ARM

As the name implies, we wait for the user to arm the system and motor controllers by turning and holding the steering as far right as possible. This is indicated on the LED by transitioning to a yellow color. Once armed, we then transition to the LINK_COMMUNICATION state.

LINK_COMMUNICATION

Here we set each motor controller to the Operation Enabled state which is the final state and enables the motors to be "running". In this state the indicator LED becomes red. The initial velocity here is set to '0' which gives the motors holding power. This stage is rather complex and

sometimes we need to try to rearm the controller. The code here handles this case and is quite impressive in its robustness. Once all motor controllers are armed, we go to either the `RUNNING_NOMINALLY` state or `RUNNING_SIMULINK` state depending on whether the defined variable `SIMULINK` is set to a '0' or '1' respectively.

`RUNNING_NOMINALLY`

This stage is where the loop will stay at until we encounter an error. This state starts by checking to see if any CAN messages can be processed. It then attempts to write a new value to the motor controllers if 20 milliseconds have gone by; if it hasn't then it moves on. We then check the voltage level of the motor controllers to regulate the battery level. We decided to set a minimum voltage level of 22V and if this level isn't met the LED indicator changes to a purple color and shuts down all the motor controllers. In most cases we continue through the state and attempt to write to the servo if 10 milliseconds have gone by. In this state the indicator LED is green.

`RUNNING_SIMULINK`

This state is very similar to `RUNNING_NOMINALLY`. The difference is that instead of taking input directly from the radio receiver we send it to Simulink first and use the values that Simulink sends back. To implement this, we had to use the Raspberry Pi/Simulink as the SPI master while the Teensy was the receiver. We used a one-byte opcode to indicate to the Teensy which functionality the Raspberry Pi/Simulink is trying to use. To achieve this setting we incorporated a switch case using the one-byte prefixed opcode. Other than the switch case this state also incorporates all functionality that the `RUNNING_NOMINALLY` state has. Here the indicator LED is supposed to be cyan but it looks more white in reality.

`INDICATE_AND_LOG_ERROR`

`WAIT_FOR_CLEAR`

These states have yet to be implemented. They were developed in the prototype phase and serve as a base for future projects to use.

7.7.3 Teensy's Pertinent Variables

SIMULINK: The `SIMULINK` defined variable is used to switch between the `RUNNING_NOMINALLY` functionality and the `RUNNING_SIMULINK` functionality by writing a '0' or '1' respectively.

MC_VOLTAGE_THRESHOLD: This variable is used to set a minimum limit that the perceived voltage by the motor controllers must not dip below. This value is counted in 0.1V and we recommend not changing its value.

CANbus: The `CANbus` global variable is the key to all CAN communication. Key functions regarding this variable are specified in `FlexCAN.h`.

Next_state: This crucial global variable is how we interpret which state we are currently in.

All data and output variables are self-explanatory and are meant to be apparent in their meaning.

Timing variables are used to control how often we exercise certain functionality. As an example, we use the motor's timing variables to write approximately 50 times per second.

PRINT: Located in uLaren_CAN_Driver.h, if set to a '1' many items will be printed in the serial port and is quite helpful for debugging purposes.

SCALE FACTOR: Be careful with this defined variable as it controls the factor we scale the throttle by. For normal use we advise not going above 2 to maintain similitude. The motors however are capable of going much higher (up to 8 is theoretically possible but is very strongly advised against and could destroy the motors).

7.7.4 CAN User Guide

Introduction

To communicate with our Maxon motor drivers, we had to create a partial implementation of the CANopen CAN in Automation (CiA) protocol. We encountered many difficulties while doing this, as it was our first time working with CAN, but we have successfully pieced together an implementation that works nominally. CAN stands for Communication Area Network, and is a popular communication protocol, particularly in Automotive applications. CANopen is an 'Application Layer' on the CAN protocol, meaning that it is a pre-defined way of controlling hardware using a CAN bus. This brief will overview and consolidate our discoveries and documentation, acting as a springboard for someone who wishes to create their own CANopen partial implementation.

CAN Brief

CAN uses a differential voltage signal, meaning that the hardware level of communication is interference and noise resistant as well as compatible across slightly different voltages (3.3V CAN and 5V CAN are fully cross-compatible). The protocol has built in collision avoidance, preventing one node on the bus from communicating while another is in progress. The protocol can be considered to be made up of data 'frames.' Each frame contains several different fields. The critical fields to be aware of are message ID, data length, and data. Message ID indicates which node on the bus should interpret the message, and how it should interpret it. Data length specifies how many bytes of data will be transmitted, and data contains the actual message to be passed from one node to another. Other parts of the CAN frame, including the CRC field, are automatically calculated and used for data validation and collision avoidance. A typical CAN frame is presented below in Figure 1.

request, the request frame indicates which object to pass the data from. The response frame contains the object that was read from as well as the data that was read.

Process Data Objects are a way to write to or read from multiple objects on a node. These are often used periodically or responsively to pass data in between slave nodes. For our purposes, PDOs can be used to read or write pre-defined sets of data with less back and forth required. We can use a rxPDO (receive from the perspective of the slave node) to pass data into multiple predefined objects using one frame, and no return frame is passed. A txPDO can be used in multiple ways, including synchronously, asynchronously, and asynchronously requested. For synchronous use, the bus master can command all nodes with the configured synchronous PDO to record data simultaneously, and have them all pass this data after the next sync command. For standard asynchronous use, the slave nodes will transmit the PDO whenever one of the contained objects changes (only as frequently as configured). The other method of asynchronous use is to inhibit the automatic transmission and only respond with the data from configured objects when a request frame is sent.

CANopen Message Structure

CiA compliant messages are interpreted with information from two parts, the message ID (called COB-ID or CAN Object ID) and the data part of the frame. The following section outlines the basic format for NMT, Heartbeating, SDO, and PDO communication. Discussion regarding specific roadblocks and discoveries we had during development will then close out this portion of the SSIVD documentation. This documentation is from the perspective of a partial CiA master, and it is not meant to support development of a CiA node partial implementation. The information here may still be helpful as it consolidates pieces of information that are not readily available in a single location.

SDO's

The first piece of information to start with is the COB-ID. For CANopen, a COB-ID consists of one field of 4 bits called a function code and another field of 7 bits for a device id. The seven bits of the device id is how you differentiate between devices on the network, allowing for a maximum of 127 devices. The 4 bits of the COB-ID for the function code is used to describe what type of a message the node is sending. For example, sending an SDO to a node would require a COB-ID of 0x600 for the function code + the node id of the device you are communicating with. When the node responds to this, it will respond with a function code of 0x580 + its own node id. Note that when you look up the function code it pertains to how the node recognizes it, so using our previous example we'd know that 0x600 is an SDO RX and a 0x580 is an SDO TX. That's it for the COB-ID.

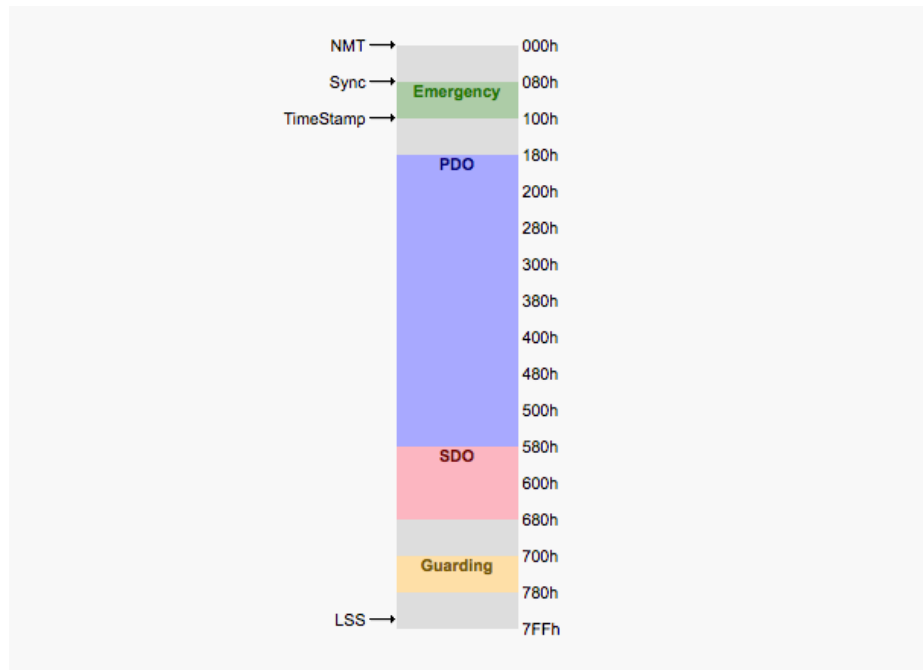


Figure 53: COB-ID Values Broken Up by Function Codes

The next place to start is the payload data. There are 8 bytes in this region of a CAN message. In CANopen, the first 4 bytes are used to describe the message being sent leaving the last 4 bytes as the place to store the actual data. Figure 60 is included below as a visual reference.

COB-ID	Command	Index		Subindex	Service Data (Parameters)			
11 Bit	Byte 0	Byte 1 (LSB)	Byte 2 (MSB)	Byte 3	Byte 4 (LSB)	Byte 5	Byte 6	Byte 7 (MSB)

Service data longer than 4 bytes (Objects 1008h, 1009h, 100Ah) are transferred by the segmented protocol.

Figure 54: Notable CANopen Fields

As you can see from Figure 60, the first four bytes are used to hold a Command Specifier, Index, and a Subindex. The Index and Subindex refer to the object in the Object Dictionary that you are attempting to write to or read from. You can find these specific objects for our motor controllers in the Maxon Firmware Specification for that specific device. The tricky part is the Command Specifier. This took us a long time to figure out. We read somewhere the wrong values to use for this and were quite frustrated when they weren't working. There are different values to use depending on whether it's a read or a write. 0x40 will work for any read request. For Write Requests, the CS value changes depending of the length of the data you wish to write; but remember, this length must match the length of the object as specified in its Object Dictionary. Figure 4 below describes the correct values to use as a write Command Specifier.

Command Code	Meaning
0x23	Write Dictionary Object reply, expedited, 4 bytes sent
0x27	Write Dictionary Object reply, expedited, 3 bytes sent
0x2B	Write Dictionary Object reply, expedited, 2 bytes sent
0x2F	Write Dictionary Object reply, expedited, 1 bytes sent

Figure 55: Command Codes for a Write Request Frame

This about sums up a SDO CANopen message frame. We talked about the COB-ID and how SDO frames have function codes of 0x580 and 0x600 and the need to add the node id to these base values. We went over the payload format and what values are expected to reside in each byte. Below you will find an example of a SDO Read Request followed by a SDO Write Request. One final note of advice is to recognize the format of the Index and Parameter fields. Both of these fields expect the Least Significant Byte (LSB) to be written first, followed by the corresponding bytes of increasing significance ending with the Most Significant Byte (MSB).

```

//statusword request
msg.id = 0x600 + node_id;
msg.ext = 0;
msg.len = 4;
msg.timeout = 0;
//
msg.buf[0] = 0x40;
msg.buf[2] = 0x60;
msg.buf[1] = 0x41;
msg.buf[3] = 0x00;
//

```

Figure 56: Example of a Read Request SDO

```

//tell MC's to go to shutdown state
msg.id = 0x600 + node_id;
msg.ext = 0;
msg.len = 6;
msg.timeout = 0;
//
msg.buf[0] = 0x2B; //68
msg.buf[2] = 0x60; //60
msg.buf[1] = 0x40;
msg.buf[3] = 0x00;
//
msg.buf[5] = 0x00;
msg.buf[4] = 0b00000110;
msg.buf[6] = 0;
msg.buf[7] = 0;

```

Figure 57: Example of a Write Request SDO

PDO's

Since we've gone over the process of using SDO's we will now explain how to use Process Data Objects (PDO). PDO's can be thought of almost like a custom SDO. As previously mentioned PDO's can be configured in many different ways. For our use we configured each node to accept the Target Velocity and the accompanying Controlword that enables the controller to start to reach the new Target Velocity (we will go over how to use the Maxon controllers, Target Velocity, and Controlwords later on). Once we set this up we simply use the function code of PDO4 (0x500) + the id of the node we are directing as the COB-ID. We then set bytes 0 and 1 to be the new Controlword and bytes 2-5 as the value of the new Target Velocity. Most of the work on PDO's come from the EPOS Software as we configure the motor control to accept the PDO we envision. To configure a PDO, you must first set the number of objects to 0. You can then modify the assigned objects, and the datalength for each. After completing this to satisfaction, you can set the number of mapped objects to your desired number.

```

//initiate target velocity
//tell MC's to go to operation enabled state w/target velocity enabled
msg.id = 0x500 + node_id;
msg.ext = 0;
msg.len = 6;
msg.timeout = 0;
//
msg.buf[0] = 0x0F;
msg.buf[1] = 0x00;
memcpy(&(msg.buf[2]), (void *)&throttle, 1);
memcpy(&(msg.buf[3]), ((char *)&throttle + 1), 1);
memcpy(&(msg.buf[4]), ((char *)&throttle + 2), 1);
memcpy(&(msg.buf[5]), ((char *)&throttle + 3), 1);
msg.buf[6] = 0;
msg.buf[7] = 0;

```

Figure 58: Example of a Configured PDO

Using CANopen with the Maxon 50/5 Controller

Starting up the CAN Network

The network initialization is very straightforward. It takes one command to turn on the network of every node (there is also the option to select which nodes to turn on). Figure 65 below is what we used to do this. The message id must be '0' for this to work but data byte 1 is where you select the node id of the specific node you wish to enable (a '0' will turn on every node in the network).

```

//"Start Remote Node"
msg.id = 0;
msg.ext = 0;
msg.len = 2;
msg.timeout = 0;
msg.buf[0] = 0x01;
msg.buf[1] = 0;

```

Figure 59: Start CAN Network Example

Initializing the Controller

There are a few things to be done for initializing the motor controller. As written in the firmware specification, there is a state diagram to be followed to get the controller into Operation Enabled mode. For good practice we start it up through the Shutdown State and bring it to Switched On, where we then wait for the user to arm the system. To do this we must write new values to the controlword. The controlword is the key to controlling the state of the device. Below you will find examples as to the messages we sent for Shutdown and Switched On. We also need to set the mode of the controller into Profile Velocity Mode as this is the mode we chose to run in our project.

```

//tell MC's to go to shutdown state
msg.id = 0x600 + node_id;
msg.ext = 0;
msg.len = 6;
msg.timeout = 0;
//
msg.buf[0] = 0x2B;
msg.buf[2] = 0x60;
msg.buf[1] = 0x40;
msg.buf[3] = 0x00;
//
msg.buf[5] = 0x00;
msg.buf[4] = 0b00000110;
msg.buf[6] = 0;
msg.buf[7] = 0;

```

Figure 60: Controlword Write of Shutdown State

```

//initialize MC's to profile velocity mode
msg.id = 0x600 + node_id;
msg.len = 5;
msg.timeout = 0;
msg.buf[0] = 0x2F;
msg.buf[2] = 0x60;
msg.buf[1] = 0x60;
msg.buf[3] = 0;
msg.buf[4] = PROFILE_VELOCITY_MODE;

```

Figure 61: Profile Velocity Mode Selection

```

//tell MC's to go to switch-on state
msg.id = 0x600 + node_id;
msg.ext = 0;
msg.len = 6;
msg.timeout = 0;
//
msg.buf[0] = 0x2B;
msg.buf[2] = 0x60;
msg.buf[1] = 0x40;
msg.buf[3] = 0x00;
//
msg.buf[5] = 0x00;
msg.buf[4] = 0b00000111;

```

Figure 62: Controword Write of Switched-On State

Arming the Controller and Motor

Arming the controller only takes one step and again it is a change to the controlword. This will bring the controller finally to the Operation Enabled state. It is advised that you wait 500-1000ms to allow the controller to switch into this armed state.

```
//tell MC's to go to operation enabled state
msg.id = 0x600 + node_id;
msg.ext = 0;
msg.len = 6;
msg.timeout = 0;
//
msg.buf[0] = 0x2B;
msg.buf[2] = 0x60;
msg.buf[1] = 0x40;
msg.buf[3] = 0x00;
//
msg.buf[5] = 0x01;
msg.buf[4] = 0b00001111;
```

Figure 63: Controlword Write of Operation Enabled State

Nominal Use

At this point in time the motors will have a holding torque, as the current written Target Velocity is '0'. To change this value we simply write a 4 byte integer into the Target Velocity Index in the Object Dictionary. An example is shown below. That last thing we need to do is enable the controller to achieve this Target Velocity. Again we will change the controlword, finally writing a '0' into bit 8 and watching as the motor finally spins!

```
//write to Target Velocity
msg.id = 0x600 + node_id;
msg.ext = 0;
msg.len = 8;
msg.timeout = 0;
msg.buf[0] = 0x23;
msg.buf[2] = 0x60;
msg.buf[1] = 0xFF;
msg.buf[3] = 0;
//
memcpy(&(msg.buf[4]), (void *)&throttle, 1);
memcpy(&(msg.buf[5]), ((char *)&throttle + 1), 1);
memcpy(&(msg.buf[6]), ((char *)&throttle + 2), 1);
memcpy(&(msg.buf[7]), ((char *)&throttle + 3), 1);
```

Figure 64: Writing to Index 0x60FF (Target Velocity)

```

//initiate target velocity
//tell MC's to go to operation enabled state w/target velocity enabled
msg.id = 0x600 + node_id;
msg.ext = 0;
msg.len = 6;
msg.timeout = 0;
//
msg.buf[0] = 0x2B;
msg.buf[2] = 0x60;
msg.buf[1] = 0x40;
msg.buf[3] = 0x00;
//
msg.buf[5] = 0b00000000;
msg.buf[4] = 0b00001111;
msg.buf[6] = 0;
msg.buf[7] = 0;

```

Figure 65: Controlword Write with Target Velocity Enabled

7.8 RECOMMENDATIONS FOR CONTINUED SOFTWARE DEVELOPMENT

7.8.1 Simulink

Currently, our model is made primarily using MATLAB code that we run through Simulink using Interpreted MATLAB Function Blocks. These blocks, quite unfortunately, are unable to be run on a target hardware, which means that the computer must be made to run Simulink on its own CPU while the MATLAB code calls the Raspberry Pi to do specific instructions over Ethernet. We ended up running into this issue late in our development while trying to integrate all of our subsystems together.

The goal for this project was to let Simulink manipulate an small-scale vehicle in real-time. While we achieved this, having a cord connected to the Raspberry Pi at all times was not what we had in mind. To allow Simulink to run without a tether there are a few options. It appears to be easiest to use S-Function blocks to write a device driver that will do the same functionality as we implemented in MATLAB. We are unsure how long this task might take and someone more familiar with Simulink would be more suited to this task.

Notable features in the MATLAB code include establishing a connection to the raspberry pi, creating a serial data object, and using this serial object to send data to the Teensy microcontroller. The first two features are easy enough to do in MATLAB and the first would not be necessary when configured to run on the raspberry pi itself. What someone would need to do first is to initialize a serial path using the default tx and rx pins on the raspberry pi. The next necessity is to use the serial port to transmit and receive data. We established an arbitrary connection method that allows the Teensy and Raspberry Pi microcontrollers to ask for and receive specific information. Table 1 below shows the mapping we use for various requests.

Table 11: Serial Connection Sequences

Initial Raspberry Pi	Byte from	Corresponding Functionality	Byte Length the Teensy Expects
11		Write the Steering Value (-400 to 400)	2
12		Write the Right Front Motor (-1000 to 1000)	2
13		Write the Left Front Motor (-1000 to 1000)	2
14		Write the Left Rear Motor (-1000 to 1000)	2
15		Write the Right Rear Motor (-1000 to 1000)	2
21		Read the Steering Input Value (-400 to 400)	2
22		Read URF Distance Sensor	2
23		Read GYRO_X position data	2
24		Read GYRO_Y position data	2
25		Read GYRO_Z position data	2
26		Read Throttle Input Value (-500 to 500)	2

As you can see from the table above, the method to connecting to the Teensy requires a sequential method. The Teensy expects an instruction byte to map the proceeding data to a specific variable. For example, if we wanted to tell the right rear motor to operate at 200 rpm, we would send one byte with an opcode of '15' followed by two bytes in 'int16' form to specify a value between 1000 and -1000.

7.8.2 Adding a Sensor

All the sensors communicate directly to the Teensy microcontroller. If you need to hook up another sensor there are two distinct steps you need to go through. The first is to demonstrate reliable connection between the Teensy and the new sensor. For almost every sensor imaginable there is already someone who was created a reliable demo for you and has put it on the web for free. Search around and see if you can find this; if not, you'll have to build one the old fashioned way through documentation.

Once you have a demo, open up the Arduino IDE (if it's not already installed visit our Software Installation document) and plug in the source code for the demo. Remember to include any accompanying files the source demo might need. From here, upload it to Teensy and watch it work (if it doesn't, debug and use the internet to correct any mistakes). Once you have a working demo you are done with the first step.

The next step is to insert your main sensor variable(s) into the top of the page next to all the other variables. For good coding practices and to stick with our design, you should develop one function to initialize your sensor. Once you have this, call that function from the INITIALIZE_PERIPHERALS case in our main loop and refer to Figure 1. Also, take this time to copy the needed files from your demo into our Teensy_Firmware folder.

```

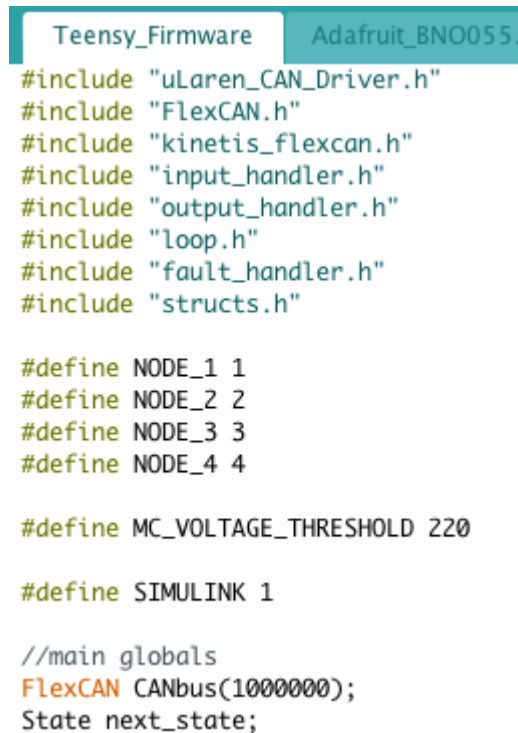
void loop() {
  CAN_message_t msg;

  switch(next_state)
  {
    case(INITIALIZE_PERIPHERALS):
      //initialize other things
      if (PRINT)
      {
        Serial.println("Initializing Peripherals");
      }
      initPWMin();
      initServo();
      next_state = INITIALIZE_CONTROLLERS;
      break;
  }
}

```

Figure 66: Insert Initialize Function in the INITIALIZE_PERIPHERALS Case

For normal operation, the car will either run in the RUNNING_NOMINALLY case or the RUNNING_SIMULINK case, depending on whether you are running Simulink (this can be turned on or off from the SIMULINK defined variable at the top of Teensy_Firmware.cpp).



```

Teensy_Firmware
Adafruit_BNO055

#include "uLaren_CAN_Driver.h"
#include "FlexCAN.h"
#include "kinetis_flexcan.h"
#include "input_handler.h"
#include "output_handler.h"
#include "loop.h"
#include "fault_handler.h"
#include "structs.h"

#define NODE_1 1
#define NODE_2 2
#define NODE_3 3
#define NODE_4 4

#define MC_VOLTAGE_THRESHOLD 220

#define SIMULINK 1

//main globals
FlexCAN CANbus(1000000);
State next_state;

```

Figure 67: Where to Find the SIMULINK Variable ('1' means on, '0' means off)

In the correct RUNNING_ "____" loop, insert whatever code you want to continually run. If you are wanting to incorporate this sensor into Simulink, read on.

The first thing you are going to want to do is to locate the switch case in the RUNNING_SIMULINK case. Here is where the magic happens and the transferring of variables to the Raspberry Pi/Simulink takes place. Pick a number (any number from 27 – 127 will work) and setup a case block for your variable as we have done in numbers 21-26. You will want to do a Serial1.write() with the first parameter being your variable and the second number being the number of bytes your sensor variable takes up (all of ours ended up being two bytes long).

```
case 22: //URF write to pi
    Serial1.write((const uint8_t*)&URF_dist, 2);
    break;
case 23: //gyroX write to pi
    Serial1.write((const uint8_t*)&g2, 2);
    break;
-- .. .. .
```

Figure 68: Two Examples of Sending a Sensor Variable to Simulink

The second and last step is to incorporate the serial connection on the Raspberry Pi/Simulink side. This is surprisingly easy to do since our model is mostly running through MATLAB code. Figure 4 provides an example function we have that incorporates the same URF data sequence that we see coming from the Teensy in Figure 3.

```
function y = readURFInput(~)

    global myserialdevice;
    global myconnection;

    if (myconnection > 0)
        write(myserialdevice, 22);
        x = typecast(read(myserialdevice,2), 'int16');

        y = double(x);
    else
        y = 0;
    end
end
```

Figure 69: Example of Sending Serial Opcode and Receiving the Sensor Data

To change this function to incorporate your own sensor data, all you need to change is the values in the write and read functions. First change the '22' opcode to the same value you chose to use in the Teensy switch block. Then modify the read function to match the number of bytes you set the Teensy to send (if your variable was two bytes you don't need to change anything on the read function). Also, if your variable is different than two bytes a change to the parameter 'int16' is needed to be appropriately changed ('int8' and 'int32' are usable parameters).

The last thing to do is to create an "Interpreted MATLAB Function" block in Simulink and add your function as a parameter. Create the necessary wiring from the "Create Connection Variable" block we made and design your own algorithm!

7.8.3 Teensy Loop

We wanted to implement error handling from the `RUNNING_NOMINAL` and `RUNNING_SIMULINK` states but we will leave this up to future contributors. When we encountered an error we expected to first transition to the `INDICATE_AND_LOG_ERRORS` case. Once done there, we would transition to the `WAIT_FOR_CLEAR` state and wait for the user to rearm the system using some predefined action (possibly turning the steering completely to the left). At this point there are many different transitions you could make. One possible solution would be to go back to the `INITIALIZE_PERIPHERALS` state to give the system a reboot of some kind. Another could be to just transition back to a running state. Ultimately the decision is up to you and creating your own state(s) could end up being the best solution.

8. DESIGN VERIFICATION

8.1 TESTING PLAN

We performed a series of tests with and on the vehicle to ensure that it meets all of our design specifications and customer requirements. We used quantitative methods— mass properties and CG position, as well as compared real results to that of our steering model developed earlier. To test the more qualitative parameters, we examined the system for any defects and provided recommendations for future fixes or improvements. Simulink modeling will help us to tune the model vehicle parameters to allow for more effective control systems.

See Appendix H for the full description of our design verification plan.

8.2 BUILD QUALITY EVALUATION

In lieu of a destructive test that would potentially render our hard work useless, a qualitative evaluation was conducted to establish potential areas of structural weakness. Where possible, modifications were made to strengthen any weak areas. However, there are a few areas where the mechanical system could be improved in the future to create a more durable and dynamically similar vehicle to that of a full scale vehicle. These will be discussed in a later section.

8.2.1 Modifications

Bump steer occurred due to the position of the steering turnbuckles on the steering blocks. As a result, we raised the mounting position and threaded a larger screw through ball joint as shown:



Figure 70. Steering linkage modification to mitigate bump steer.

This adjustment effectively reduced the bump steer, as well as the toe angle. We also noticed that the servo did not sit evenly in its slot due to the structural ribs on the mounting flanges. As a result, the chassis slot was filed down to incorporate a slot, as shown below:

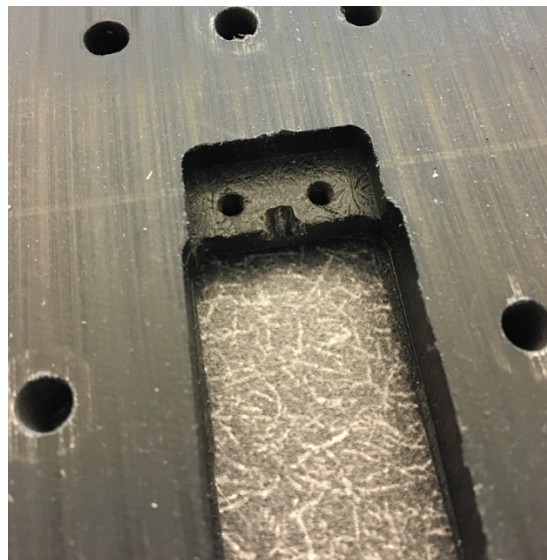


Figure 71. Grooves filed in servo slot to incorporate the structural ribs on the servo.

This change should be added as an extra manufacturing step for future chassis using the same servo. Spacers were also added between the hex standoffs and chassis mount on the front end of the chassis. This was due to a slight difference in height between the top of the standoff and top of the motor housing that was initially thought to be negligible.

Changes were also made to stiffen the suspension by adding Traxxas supplied spacers between the top of the spring and ultrashock endcap. The size and amount of spacers used set the sag to about 30% in both the front and rear. The original progressive springs in the rear were moved to the front, and a 10% stiffer set of Integy springs were installed in the rear. Clips and zip ties were also added for cable management.

Another important note is that there were no necessary modifications to the motor shaft when aluminum shaft couplers were used. This greatly simplified the manufacturing process and eliminated any chances of unnecessary damage to the motors.

Finally, during expo we noticed that the a-arm mounts became loose over time. This was due to the eccentric vibration of the wheels when ran on the display stand. The loosening may not be an issue when not on the displaying stand, but it will be worth monitoring in the future. Future modifications of the 3D print could be made that incorporate the A-arm mounts into the motor housing design.

8.3 QUANTITATIVE TESTING

8.3.1 Center of Gravity Position

The center of gravity(CG) can be determined based upon the static weight distribution of the vehicle. Four standard kitchen scales (11 lb max) were purchased to give an accurate reading of the reaction forces at each tire-ground interface (Figure 72). First, we determined the longitudinal position of the CG through the relationships:

$$b = \frac{R_R L}{W} \quad , \quad a = L - b$$

As a result, we needed to measure the wheelbase, and reaction force at both the front and rear axles.

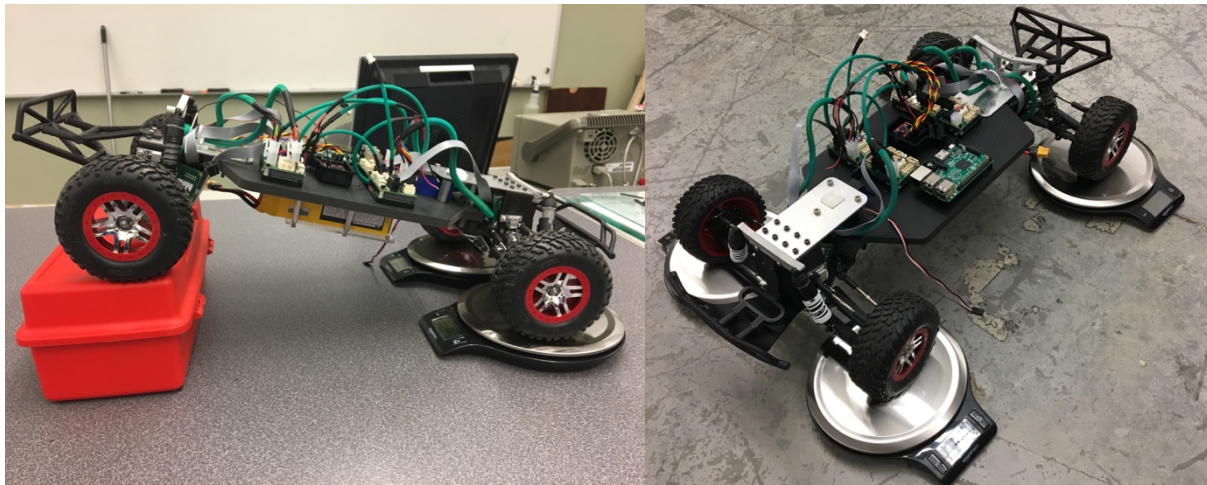


Figure 72. Lifted rear end for measuring CG height(left) and static weight distribution (right).

This meant that we would have uncertainty in the calculated value due to measurement error. It should be noted that the reading on the scale fluctuated based upon the position of the tire on the scale, but this variation was ignored for purposes of the uncertainty analysis. Any reported uncertainty is due to resolution uncertainty only. Through statics relationships, we were able to determine the approximate CG height by lifting the rear end of the vehicle approximately 120 mm. This causes a weight shift towards the front of the vehicle, which can be used to determine the position of the CG through the relationship:

$$h = \left[a - L \left(\frac{R_f}{W} \right) \right] \cot(\theta) + r$$

where, $\theta = \sin^{-1}(H/L)$

The results from this testing can be seen in Table 12, while the data collection and uncertainty analysis is tabulated in Appendix K.

Table 12. Position of center of gravity with propagated resolution uncertainty.

Variable	Value	Unc.	Units	Description
W	3.85	+/-0.001	[kg]	Total weight of vehicle
b	196.8	+/-2.5	[mm]	Distance from front to CG
a	204.2	+/-2.5	[mm]	Distance from rear to CG
h	75.1	+/-1.2	[mm]	Height of CG from ground

8.3.2 Steering Model Verification & Repeatability

In our firmware, we control the steering servo by controlling the pulse-width of a PWM signal, which the servo interprets as an angular position setpoint. While we know what values we are passing the servo, it is important to understand what steering angle these values actually produce so we can develop a proper correlation to the bicycle dynamic model. Using a protractor, we measured the angle of the inside wheel relative to forward that was produced in response to a given input to our steering function. Our system has some backlash induced hysteresis, which we attempted to correct for to find the ‘center’ output steering angle. Our control scheme does not account for the hysteresis, and it is something that can likely be improved by tightening mechanical tolerances. The steering data is presented in Figure 73, with the resulting input to angle correlation shown.

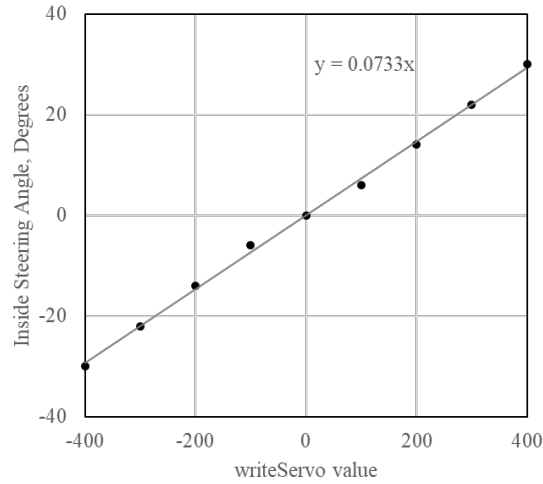


Figure 73: Inside steering angle versus the input value to our servo function.

For testing purposes, we developed a steering profile to be applied to both the model and the platform. The input function, shown in Figure 74, should produce an ‘obstacle avoidance’ profile – a lane change like maneuver in one direction, and a lane change line maneuver to return to the original path.

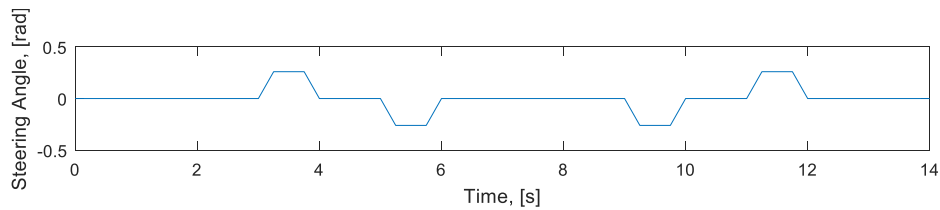


Figure 74: Steering angle profile for the repeated steering profile test.

We developed a simple function to allow for the repeated testing and data collection for an input profile. The steering profile, including steering angle, throttle, and times, were pre-defined in arrays, and the loop would interpolate between points to create a motion ramp. For this test, the user is required to hold down the throttle. If they let go, the car comes to a stop quickly. This allows the user to abort the test if it is nearing a wall or getting out of control. The abort functionality was thoroughly tested on a stand before placing the vehicle on the ground. The 14s profile took the car about 10 meters. The car is at starting position and Chris is standing at the finish location in Figure 75a, and the finish location spread of 3 discreet tests of the same profile is shown in Figure 75b.

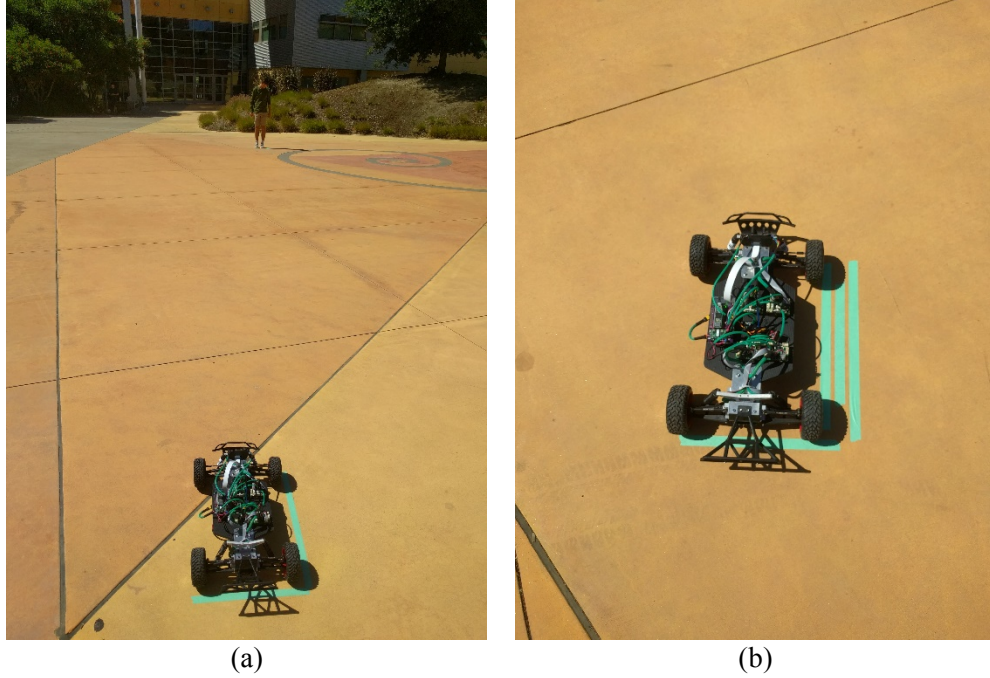


Figure 75: Starting point of the repeated steering profile test (a) with Chris standing at the endpoint. Marked end locations of the repeated steering profile test (b) demonstrating high repeatability across large distances.

During these tests, we were collecting raw linear acceleration data and filtered Euler vector orientation data from the BNO055 imu and logging it for comparison to the model. We used the Euler vector to develop the output yaw relative to the starting orientation. Figure 76 shows the comparison of model and real yaw responses.

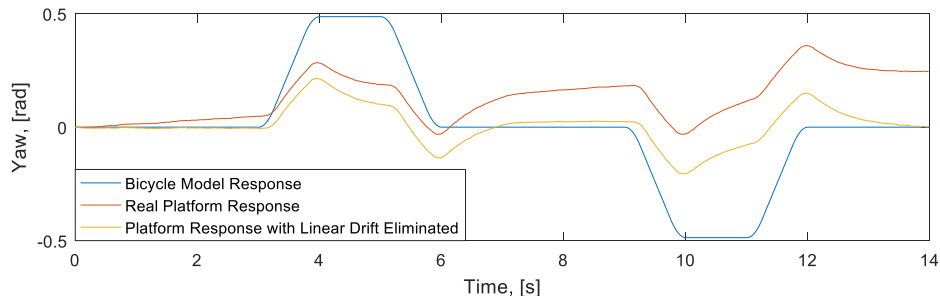


Figure 76: Comparison of steering profile yaw from the bicycle model, the platform response, and filtered platform response.

Note that Euler vector angles are from 0 to 360 degrees, and the output had to be conditioned to be a signed output. Knowing that our expected yaw angles were to be less than 180 degrees (as the car was not making a u-turn), we were able to condition the response with a simple if statement applied to each data point, shown in pseudocode below:

```
if (anglein > 180): angleout = - (360 - anglein)
else: angleout = anglein
```

We know that the drift is due to a misalignment in our steering center, which we would expect to produce a linear drift. The drift does appear to be very close to linear, and we can filter it out using an incremental function applied to the whole array. We also know that the ending yaw should be the same as the starting yaw, because this is meant as an obstacle avoidance algorithm. This incremental function is shown in pseudocode below:

```
for each yaw in yaw_array:
    yaw = yaw - yaw@tmax * (t / tmax)
```

Implementing this filter allows us to view what the response would look like if the steering was tuned more accurately, which can be done in the future.

We also felt it was important to look at the acceleration response of the vehicle during the steering profile. Since the profile is to be operating at a constant velocity, and is operating on flat ground, we are most interested in the lateral acceleration, or acceleration to the left or right from the frame of reference of the vehicle. The bicycle model response and platform response measured by the IMU are shown in Figure 77.

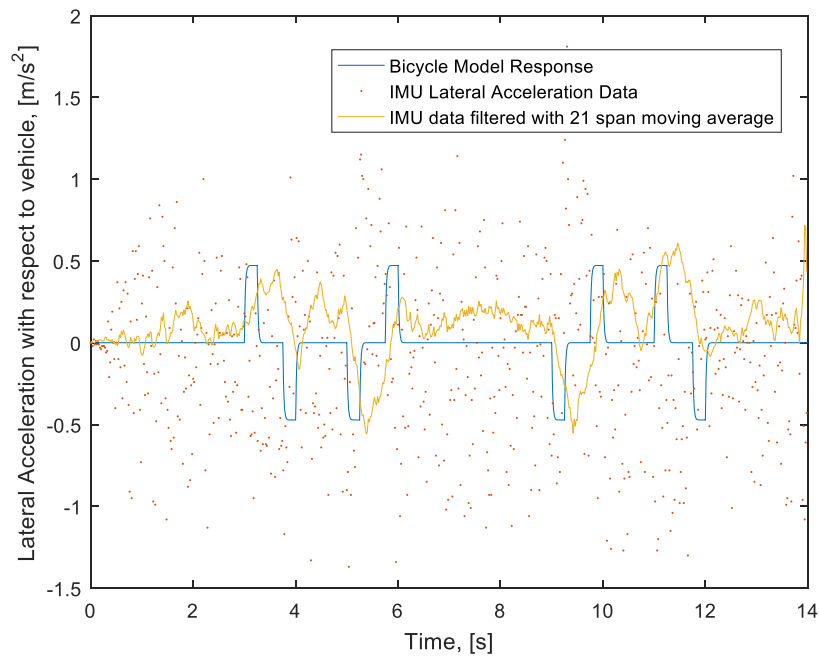


Figure 77: Comparison of steering profile lateral acceleration from the bicycle model, the platform response, and filtered platform response.

The raw data has an extremely high noise floor, and it makes it incredibly hard to decipher meaningful information from the unfiltered IMU data. To attempt to decipher the actual response, we can smooth the data with minimal overhead by using a moving average. We found that a span of 21 was a good compromise between the quality of output data and the phase delay induced by using a moving average. The data still has noise, but the magnitude of peaks and valleys more closely matches what we expected from the bicycle model. The number and order of peaks and valleys does not perfectly match the bicycle model, but they do align with the differences we saw in the real and model yaw response. The peaks are shifted about a quarter

second behind where they should be, which makes sense with the moving average span of 21. The sample frequency of the IMU is 50Hz in this data, meaning that the span covers 0.42 seconds. In hopes of better understanding the noise floor, we can take the Fast-Fourier Transform (FFT) of the raw IMU data, shown in Figure 78.

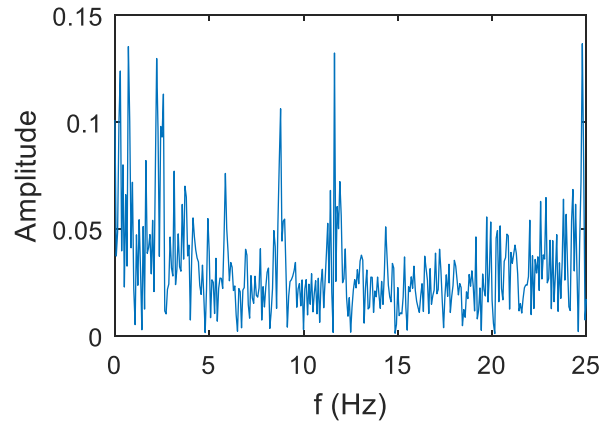


Figure 78: FFT of the raw IMU lateral acceleration data.

The IMU can only sample at 100Hz, and we were only sampling at 50Hz, so we are unable to characterize higher frequency vibrational modes. Within the range of frequencies that we can analyze, we can see that there are vibrational modes at approximately 3, 6, 9, and 12Hz, but none of those modes have overwhelming amplitudes. These modes may also be a result of aliasing, where we are sampling higher-frequency vibration modes at a lower rate, showing false modes. The unfiltered data will be useless for trying to complete stability control. A moving average filter could be applied on the Teensy, but the output data still has non-negligible noise and inherently has a phase shift. Further discussion on how to get more appropriate acceleration data is present in section 8.4, Possible Improvements.

8.4 POSSIBLE IMPROVEMENTS

8.4.1 Geometry & Structure

The build quality evaluation provided us with insight into what areas would require modifications for future adaptations of the vehicle. Although these will be discussed in more detail in later sections a few to note are:

- Installing 1/7th scale wheels that mimic realistic tire dynamics
- Increase the wheelbase for an aspect ratio that compares to a standard SUV (1.7 for Ford Edge)
- Perform tire characterization tests for cornering coefficients
- Install end caps on A-Arm dowels to prevent them from sliding out
- Lock/clamp the battery in place
- Mount IMU closer to CG (below chassis)
- Manufacture or purchase a protective cage for the electronics
- Test clamping capabilities of 3D printed shaft couplers

8.4.2 Circuit Design

With minor modifications, this revision of the circuit board worked nominally. Specifically, the LED footprint had to be modified and augmented, and the 3.3V line between the Pi and Teensy was severed. The Teensy to Pi GPIO pin was also modified, as the pin connected on the Pi is not a valid GPIO pin.

Design wise, we would have liked to have selected dimmer LEDs for both power indication and state indication, or increased the size of the current limiting resistors. We found that we only needed one screw terminal breaking out the CAN bus, but that we would have liked to have 4 screw terminals breaking out the high voltage power and ground for the motor drivers. The screw terminals only easily accept one wire, and trying to route four power cables into two ports was unfruitful. To remedy this in our implementation, we created y-splice cables joining two power cables into one wire to connect to the screw terminal.

8.4.3 Sensors & System Architecture

The serial communication link between the Raspberry Pi and the Teensy microcontroller was the right choice for our application – it helps create a ‘black box’ that handles the hardware interface. It does, however come with some limitations. Our serial port operates at 115200 bits per second, which means that we can only pass 14.4 kilobytes per second. With the control loop operating at 50 Hz, we can only pass 288 bytes per loop. In our simple demonstration implementation, we are passing 22 bytes per control loop, meaning that we have already consumed 7.5% of the maximum load. The serial communication also does consume clock cycles on both the Pi and the Teensy, introducing further delays on both ends of the communication scheme. While this is manageable at the bus load we are at, implementation of more data-heavy sensors such as LIDAR may require development of Raspberry Pi driver blocks so that the sensors are driven directly off of the Raspberry Pi GPIO. In the future, it may be desirable to develop an SPI interface as it can operate significantly faster than the standard serial interface, but we had significant difficulty implementing it and had to proceed, so serial was the right choice for us. In the scope of what we were trying to accomplish, the Serial communication allows for easier, faster development and integration of new sensors. It does, however come with limitations when hoping to implement more data throughput.

The interpreted function blocks cannot deploy to the Raspberry Pi hardware, but these can be recreated using S-Function blocks or Device Driver blocks which can deploy to the hardware. It was quite an unfortunate scenario when we tried to embed the Simulink model and have our whole platform running remotely only to have Simulink be unable to do so. Earlier in our development process we had gotten Simulink running great with the Raspberry Pi’s hardware and had been writing and reading from the Serial pins. There was no reason to believe a model that could use the Raspberry Pi as effectively as it had been doing could not just embed the same code onto the Raspberry Pi itself.

When we went to select the *External* option and hit *Run* it wouldn’t build. We weren’t sure what had happened. We looked online for similar errors only to find out it wasn’t possible to use the Interpreted MATLAB Functions in an embedded situation. It turns out that the MATLAB functions that are so easy and nice to use don’t work when embedded. The only solution is to use S-Function blocks or Device Driver blocks which require a whole process of writing C and C++ code with specific MATLAB files that are used in their specialized process of embedding code. Needless to say there is not much documentation on it, and the ones that we have found have problems we don’t have time to fix. If we had, say, two more weeks to

work on this project I am confident that we could implement a solution but time is not something we have anymore.

Additionally, the IMU acceleration data should be processed to filter the noise out of the system as best as possible. This can be implemented in multiple ways, including some configurations of the BNO055 IMU itself. This includes calibrating the IMU, pulling the calibration register data, and pushing the calibration data back to the IMU upon every power-up. For the highest possible accuracy, this should be done for each sensor on the BNO055. The IMU doesn't have any non-volatile memory, so it is critical to pull this data, store it, and push it every time the IMU is powered on. More information on how to calibrate the sensor can be found in the BNO055 datasheet, the first page of which is present in the Drawings section. On the Teensy, several different data filtering methods can be used, the simplest of which would be a low-pass filter. This can be done with a moving average, where the Teensy averages the last n readings from the IMU. As n increases, the low-pass filter can filter out lower and lower frequency noise. A more accurate way to filter noise would be to develop a system specific Kalman filter, but this is often a project in and of itself. With calibration and a moving average low-pass filter, the lateral acceleration data can be conditioned to be usable.

Currently, our system does not have built-in data logging outside of the repeated profile testing firmware. This can be implemented on the Teensy using the micro-sd card slot, or it can be implemented on the Raspberry Pi. It was not critical for the scope of our project to develop this data-logging, and we ran out of time to implement it as a 'nice to have.' It will be straightforward for someone to implement this in parallel to the development that we have already done. Removeable media is the preferred location for data-logging, but care should be taken that the data-logging does not slow down the response time of the system.

9. REPRODUCTION

Throughout the design and manufacturing phases it was important to keep in mind the reproducibility of individual parts for future SSIVD platforms. Because the vehicle we manufactured consists of parts that are time consuming to reproduce manually, we have identified alternative methods and part designs that would be sufficient for a robust and durable SSIVD.

9.1 MOTHERBOARD

The motherboard was designed in EAGLE cad, which is free for educational and non-commercial use. The board and schematic files are available on the GitHub, along with a full bill of materials. We used a prototype version of the motherboard for our system, but have put up version 1.0 on the GitHub. This version resolves the specific design errors that we had to correct with wire modifications, but does not address the 'nice-to-have' design points mentioned in the Possible Improvements section.

9.2 MECHANICAL SYSTEM

Future vehicle platforms will utilize automated processes to decrease the overall production time. As mentioned in the manufacturing section, many parts were produced via water jet, which is a simple process that only requires two dimensional .dxf files to operate. The majority of the manufacturing time would be spent drilling holes with the drill press and tapping any necessary holes. If made through CNC, the couplers will require a significant amount of time, approximately 4 pairs per hour after set up. However, the results of our FEM indicate that 3D printing the couplers is a feasible solution if they can generate the required clamping force. The other manual operation that would need to be completed is tapping and drilling the steering posts. This would be a quick operation that can be done on the lathe once the stock is cut to length. Tight tolerance rod would eliminate the need to turn the rod down to the 5mm diameter.

Table 13. Custom parts and suggested manufacturing methods.

Part/Operation	Mfg Method
Chassis Net Shape Profile Chassis Mounts Front Suspension Mount Profile Rear Suspension Mount Profile Battery Plate	Water Jet
Chassis Mounting Holes Front Suspension Mount Holes Rear Suspension Mount Holes	Drill Press
Steering Posts	Lathe
Motor Housings Steering Linkage A-Arm Mounts Rear Turnbuckle Mounts Front Turnbuckle Mounts Bumper Mounts IMU Mount URF + Camera Mount	3D Print
Shaft Coupler	3D Print/CNC

9.3 OVERALL SYSTEM COST

We estimate that any future systems will cost approximately \$1,050. Depending on manufacturing methods and costs, as well as product sourcing, this value could fluctuate significantly. The Traxxas estimates are based upon retail costs, so we highly suggest looking for third party sources that sell parts at a discounted process. The rough breakdown for future SSIVD costs can be seen below in Table 14. This cost does not include the cost of motors, and uses only an estimate for the electronics.

Table 14. Reproduction costs for future SSIVD platforms.

Total Cost	
Traxxas Parts	\$475.75
Electronics	\$450.00
Raw Materials and Hardware	\$125.84
Total	\$1,051.59

10. REFERENCES

- [1] D. Walsh, "Connected cars: AA test has effort shifting to higher gear," Crain's Detroit Business, 30 May 2014. [Online]. [Accessed 22 October 2016].
- [2] National Center for Statistics and Analysis. , "Pedestrians: 2013 Data," National Highway Traffic Safety Adminsistration, Washington DC, 2015, February.
- [3] L. Guo, L. Li, Y. Zhao and Z. Zhao, "Pedestrian Tracking based on Camshift with Kalman Prediction for Autonomous Vehicles," *International Journal of Advanced Robotic Systems*, vol. I, no. 13, pp. 1-9, 2016.
- [4] S. Munder, C. Schnorr and D. Gavrilu, "Pedestrian Detection and Tracking Using a Mixture of View-Based Shape-Texture Models," CVGPR Group, Amsterdam, The Netherlands, 2007.
- [5] US Software System Safety Working Group, "Adaptive Cruise Control System Overview," Anaheim, CA, 2005.
- [6] J. Nilsson, A. C. Odblow and J. Fredriksson, "Worst-Case Analysis of Automotive Collision Avoidance Systems," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 4, pp. 1899-1911, 2016.
- [7] T. Stevens, "A LIDAR BASED SEMI-AUTONOMOUS COLLISION AVOIDANCE SYSTEM," California Polytechnic University, San Luis Obispo, CA, 2015.
- [8] S. Lefevre, D. Vasquez and C. Laugier, "A survey on motion prediction and riskassessment for intelligent vehicles," *ROBOMECh Journal*, vol. 1, no. 1, pp. 1-14, 2014.
- [9] SAE International, "Automotive Stability Enhancement Systems," *Surface Vehicle Information Report*, vol. J2564, 2004.

- [10] Bavarian Motor Works, "Dynamic Traction Control," BMW, 2016. [Online]. Available: http://www.bmw.com/en/technology/insights/dynamic_traction_control. [Accessed 21 October 2016].
- [11] H. Lee and M. Tomizuka, "Adaptive Vehicle Traction Force Control for Intelligent Vehicle Highway Systems (IVHSs)," *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS*, vol. 50, no. 1, pp. 37-47, February 2003.
- [12] Insurance Institute for Highway Safety, "ESC and side airbag availability by make and model," Highway Loss Institute, 2016. [Online]. Available: <http://www.iihs.org/iihs/ratings/safety-features>. [Accessed 6 November 2016].
- [13] California State University, Northridge, "2016-2017 Catalog," Mechanical Engineering Department, CSU Northridge, 2016. [Online]. [Accessed 20 October 2016].
- [14] S. Solmaz and T. Coskun, "An automotive vehicle dynamics prototyping platform based on a remote control," *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. I, no. 21, pp. 439-451, 2013.
- [15] H. Manh La, R. Lim, J. Du, S. Zhang, G. Yan and W. Sheng, "Development of a Small-Scale Research Platform for Intelligent Transportation Systems," *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, , vol. 13, no. 4, pp. 1753-1762, 2012.
- [16] A. Fabio, "Independent Wheel Drive R/C Car," Hackaday, 2014. [Online]. Available: <http://hackaday.com/2014/07/12/independent-wheel-drive-rc-car/>. [Accessed 15 11 2016].
- [17] "Arduino Uno," Arduino, [Online]. Available: <https://www.arduino.cc/en/Main/ArduinoBoardUno>. [Accessed 23 10 2016].
- [18] "Arduino MEGA 2560," Arduino, [Online]. Available: <https://www.arduino.cc/en/Main/ArduinoBoardMega2560>. [Accessed 23 10 2016].
- [19] J. Man, "Raspberry Pi 3 Model B Technical Specifications," Element14, [Online]. Available: <https://www.element14.com/community/docs/DOC-80899/l/raspberry-pi-3-model-b-technical-specifications>. [Accessed 23 10 2016].
- [20] "Raspberry Pi Model A+," Adafruit, [Online]. Available: <https://www.adafruit.com/products/2266>. [Accessed 23 10 2016].
- [21] "BeagleBone Black," BeagleBone, [Online]. Available: <https://beagleboard.org/black>. [Accessed 23 10 2016].

- [22] "Hardware Support Index," Mathworks, [Online]. Available: <https://www.mathworks.com/hardware-support/index.html>. [Accessed 23 10 2016].
- [23] K. Amano, N. Goda, S. Nishida, Y. Ejima, T. Takeda and Y. Ohtani, "Estimation of the Timing of Human Visual Perception," *The Journal of Neuroscience*, p. 3981–3991, 2006.
- [24] Department of Mechanical Engineering, Cal Poly, "ME 428/429/430 Senior Design Project Reference Book and Success Guide," 2016.
- [25] Traxxas, "Traxxas Slash: 1/10 scale short-course truck," Traxxas, 2016. [Online]. [Accessed 9 November 2016].
- [26] Maxon Motor, "EC 45 flat Ø42.8 mm, brushless, 70 Watt, with Hall sensors," 2016. [Online]. Available: <http://www.maxonmotor.com/maxon/view/product/397172>.
- [27] Ford Motor Company, "2015 Ford Edge Technical Specifications," 2015. [Online]. Available: https://media.ford.com/content/dam/fordmedia/North%20America/US/2015_Specs/2015_Edge_Specs.pdf.
- [28] Maxon Motor, "EPOS4 Compact 50/5 CAN Hardware Reference," November 2016. [Online]. Available: http://www.maxonmotorusa.com/medias/sys_master/root/8823917740062/534130-Hardware-Reference-En.pdf. [Accessed 12 January 2017].
- [29] Texas Instruments, "Overview of 3.3V CAN (Controller Area Network) Transceivers," January 2013. [Online]. Available: <http://www.ti.com/lit/an/slla337/slla337.pdf>. [Accessed 27 January 2017].
- [30] R. Rajamani, Vehicle Dynamics and Control, New York, NY: Springer Science and Business Media, 2006.
- [31] CarSort, "Ford Edge SEL vs. Ford Explorer," 2015. [Online]. Available: <http://carsort.com/compare/Ford-Edge-vs-Ford-Explorer>.
- [32] W. Riley and A. George, "Design, Analysis and Testing of a Formula SAE Car Chassis," *Proceedings of the 2002 SAE Motorsports Engineering Conference and Exhibition*, vol. 01, p. 382, January 2002.
- [33] E. W. Weisstein, "Torsional Rigidity," From MathWorld--A Wolfram Web Resource, 2010. [Online]. Available: <http://mathworld.wolfram.com/TorsionalRigidity.html>. [Accessed 16 January 2016].
- [34] R. Budynas and K. Nisbett, Shigley's Mechanical Engineering Design, New York, NY: McGraw Hill, 2016.
- [35] B. Quimby, "Chapter 3 - Tension Members - Bearing on Holes," BG Structural Engineering, 27 July 2011. [Online]. Available:

<http://www.bgstructuralengineering.com/BGSCM13/BGSCM003/BGSCM00306.htm>. [Accessed 29 January 2017].

[36] NHTSA, "ESV 2017," 2016. [Online]. Available: <http://www-esv.nhtsa.dot.gov/>. [Accessed 22 10 2016].

APPENDICES

Appendix A Quality function deployment

Appendix B Boundary Sketch

Appendix C Brainstormng Tables and Figures

Appendix D Pugh and Decision Matrices

Appendix E Design Safety CHecklist

Appendix F Supporting Analysis

Appendix G Gantt Chart

Appendix H DVP&R and DFMEA

Appendix I Bill Of Materials

Appendix J Part and Assembly Drawings

Appendix K Testing Results and Data collection

Appendix L Reproduction Costs

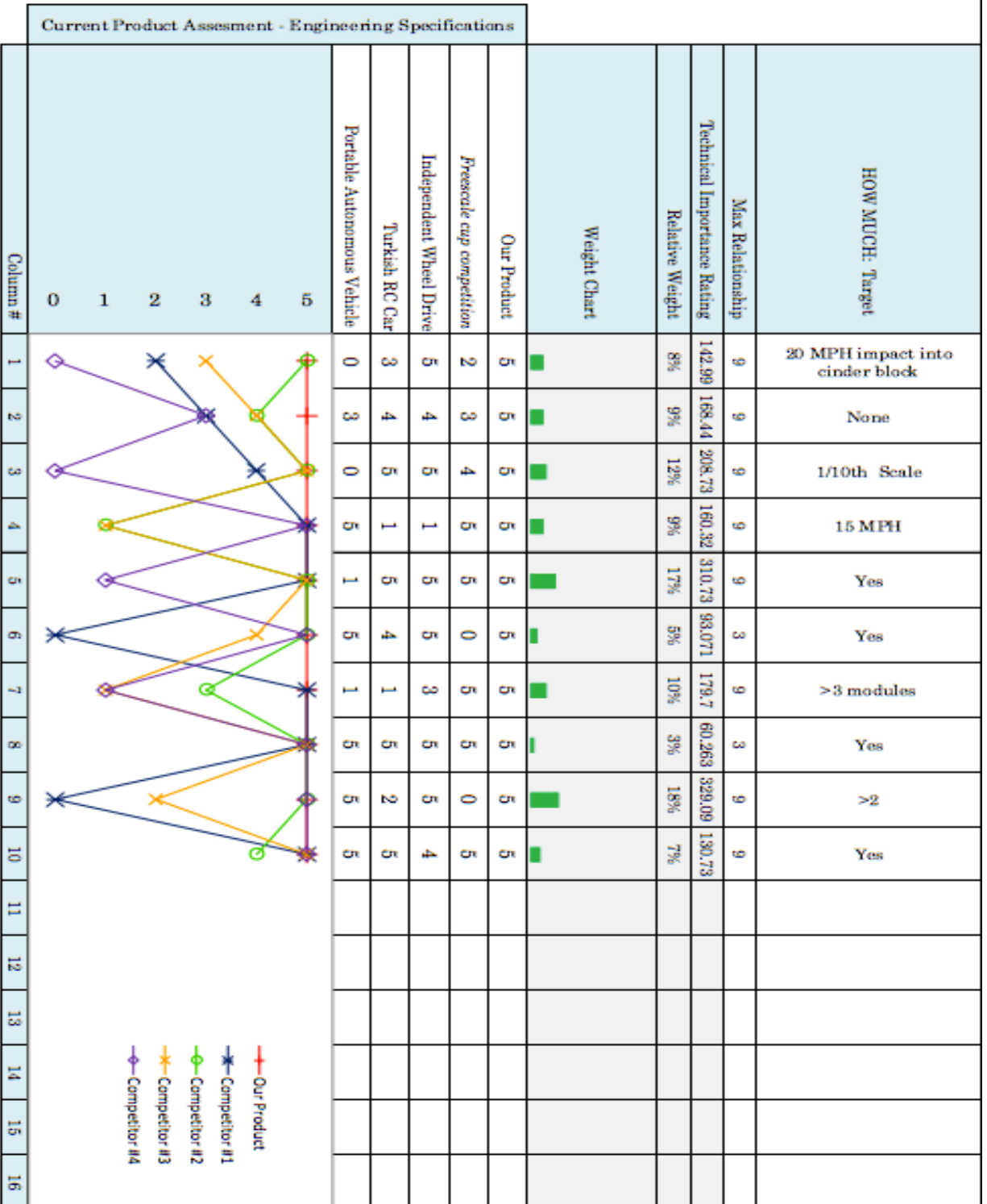
Appendix M Code Files

QFD Template Full.pdf

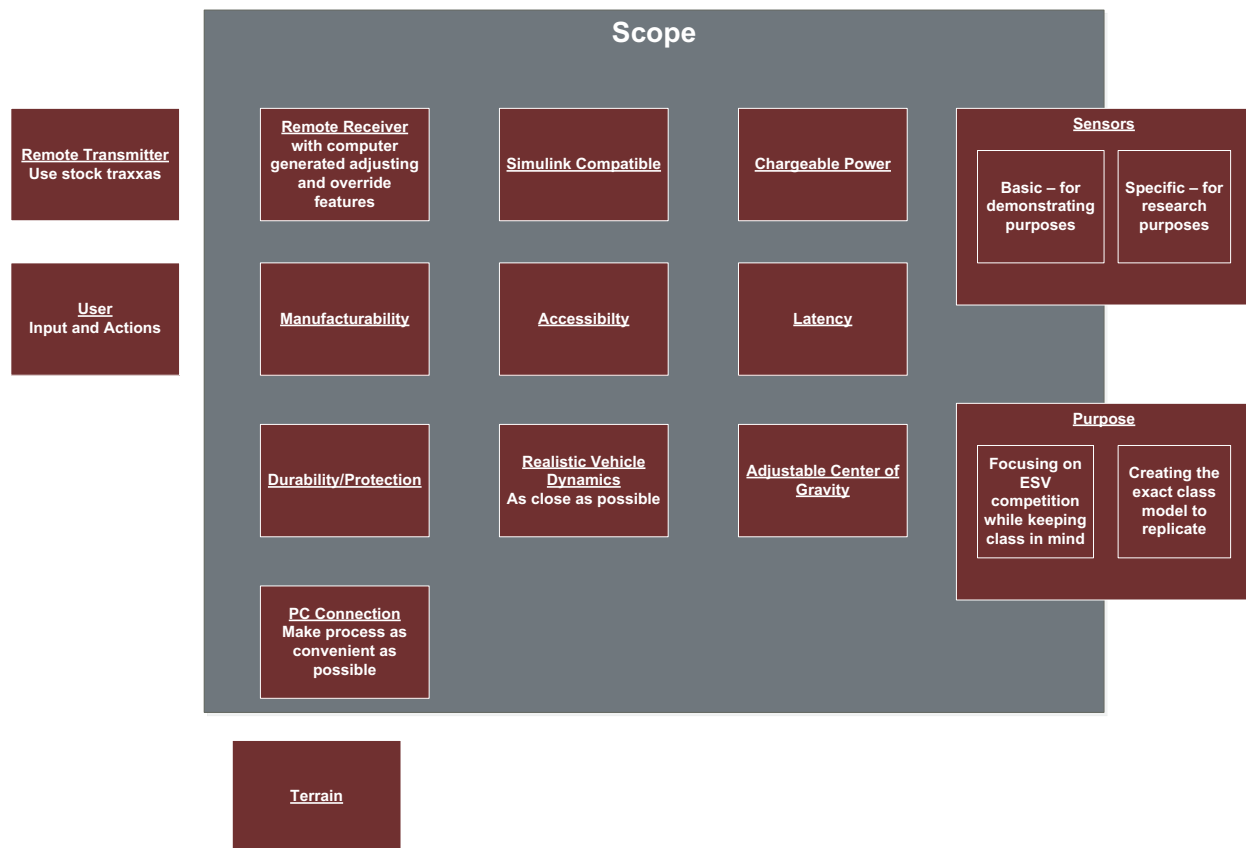


Direction of Improvement
Maximize ▲
Target ◇
Minimize ▼

A1



APPENDIX B BOUNDARY SKETCH



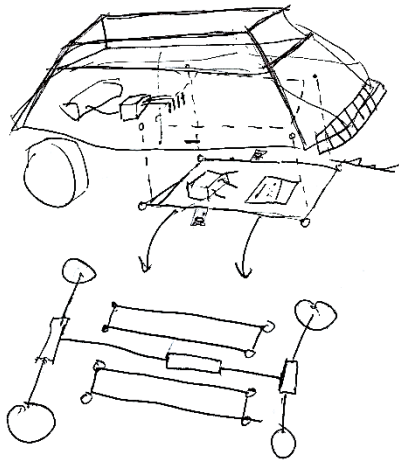
APPENDIX C BRAINSTORMNG TABLES AND FIGURES

Morph Chart

	Function					
	Electronics Protection	Protective Housing	Types of Sensors	Dynamic CG	Algorithms	Braking systems
Potential Options	Lid system	Cushion Bumper System	Microphone / sounds	Adjustable Mast	Adaptive cruise control	Friction clutch
	Wire Cage	Protective sides(go-kart)	IMU	Roll Bar w/ weights	Stop light detection	Hydraulic brakes
	Plexiglass / acrylic housing	Roll cage	Radar	Dynamic Angle Boom	Line-following	Shaft braking
	Custom PCB	Mechanical fuse	Wheel Encoder	Tub of water on top	Following	Tiny Disc Brakes
	Cast MCU in epoxy	Crumple zone	LIDAR	Battery in middle	Obstacle avoidance	Split diff, two motors
		Inflatable bumper	Pressure transducer tires	Placeable weights	Lane tracking	4 wheel steering
		Airbag	Road condition(friction)	weight rack	Dynamic stability control	Friction solenoid
		Deployable bumpers	Tachometer		Road sign detection/reading	
		Foam Bumpers	Hall Sensor		Pedestrian Detection	
		Vaccum Formed Shell	Digital Camera		Outside Camera system integration	
		Zip tie mounting points			Parking auto-pilot	
		Detachable middle component			Controller feedback	
		Central breadbox design			Light detection	
		Box with ports			Vehicle to vehicle coms	
					Collision avoidance	
					Stop-light cam	
					Adaptive speed limit	

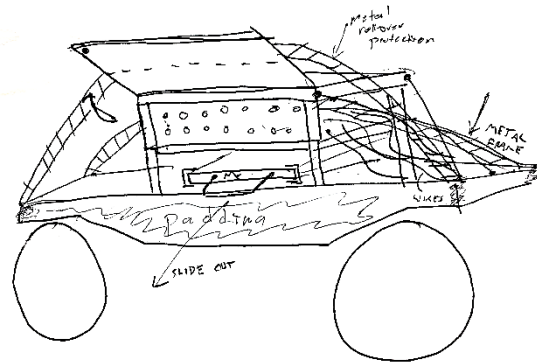
Brainstorming Examples

PRO FRAME



Drop out electronics board

FRAME

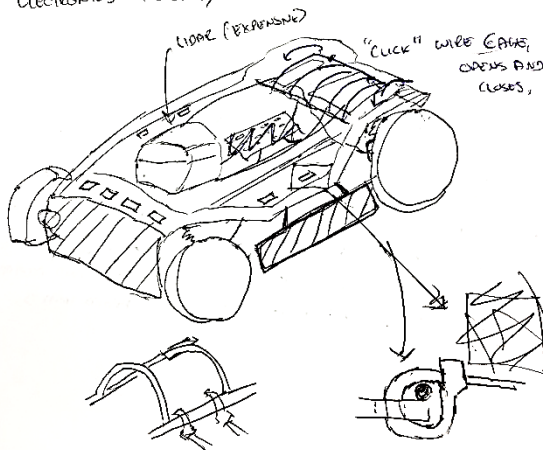


Frame and easy access sensor plug-ins

DRAWING

(DRAW PICTURE)

ELECTRONICS HOUSING

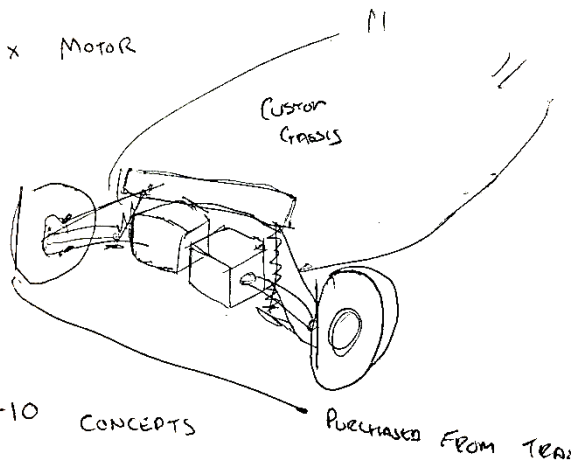


Overall concept ideation

4/16

VEHICLE RESEARCH

X MOTOR

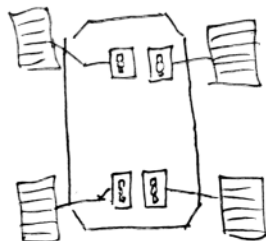


-10

CONCEPTS

3/15

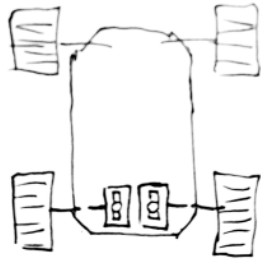
Dual Motor System w/ custom chassis



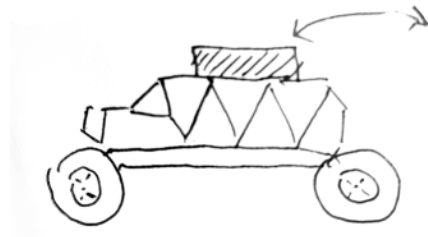
4 motor power system



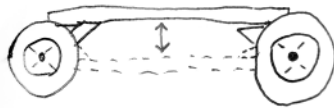
Friction solenoid



2 motor power system



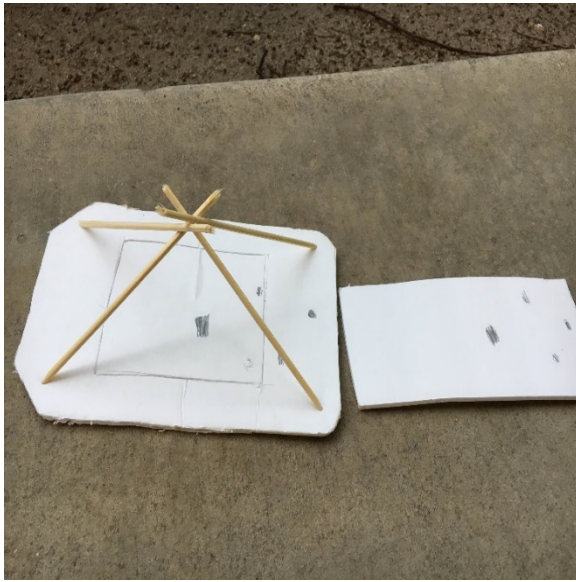
Rack with weights



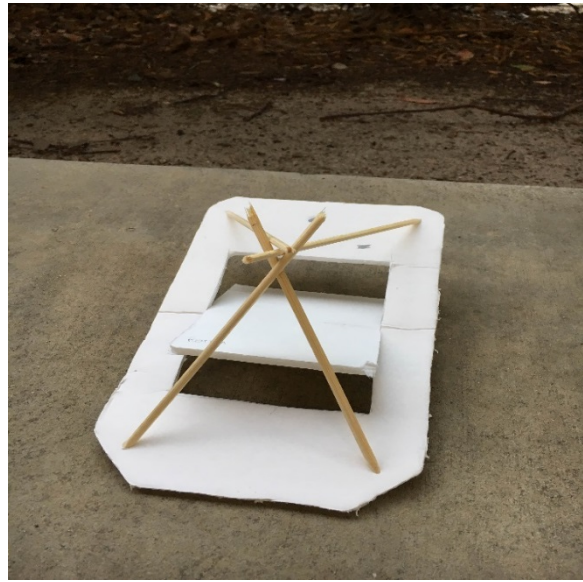
Adjustable chassis height

\

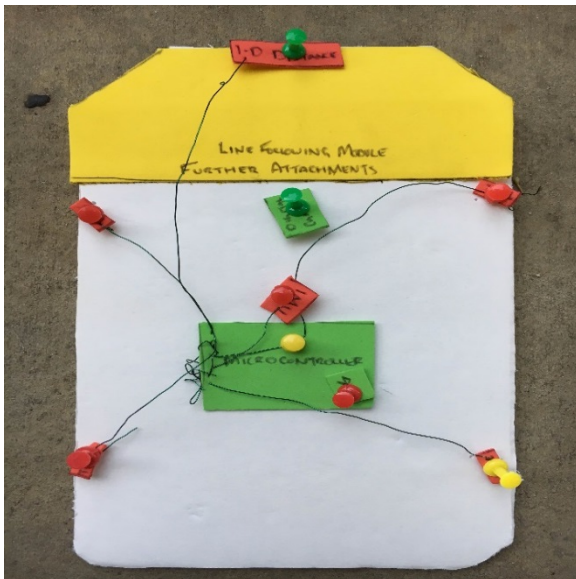
Concept Prototypes



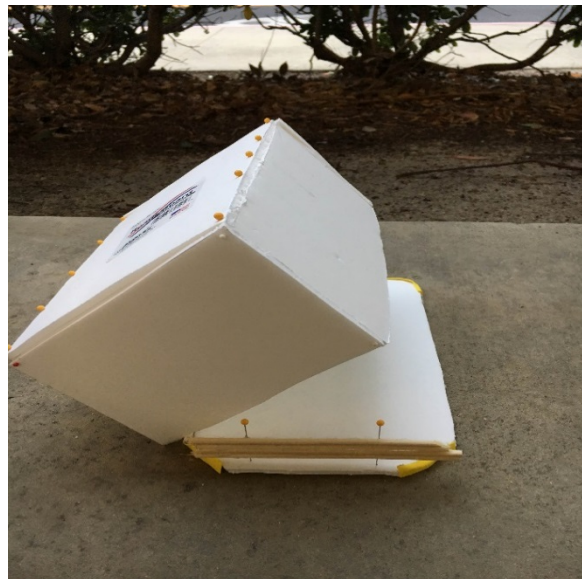
Electrical Layout – Slide in-out board with electrical connections on the bottom of the attachment (slide out piece shown upside down).



Electrical Layout – Flip board which allows the electrical components to be accessible from the bottom of the vehicle.



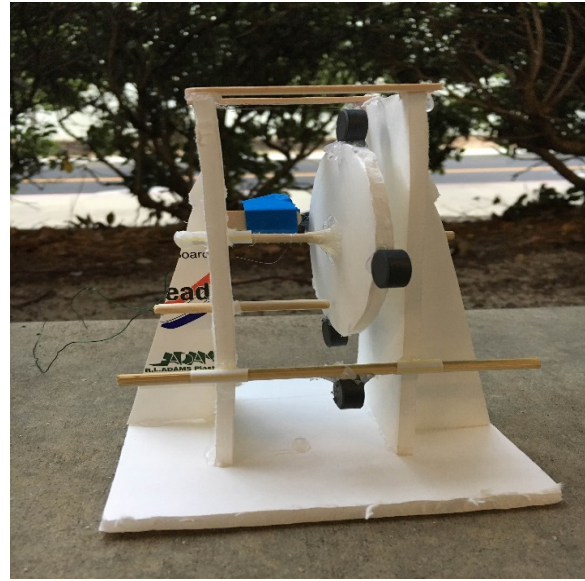
Electrical Layout – An example of the possible components we could use and the arrangement they could have. Included is a Microcontroller, multiple distance sensors, a Bluetooth sensor, an IMU, and prepared space for more sensors.



Protective Housing – This features a hinged cage design which allows the components to be covered and protective while also being readily available for human modification.



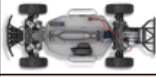




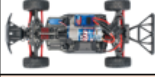
Protective Housing – A possible solution using a wired cage design where the pipe cleaners represent metal bars.



Braking System – This prototype features three different possible methods. The blue tape in the back of the picture holds an experimental eddy current system with thin copper wire wrapped approximately 100 turns that connected to a 9V battery. The next experimental design was magnetic braking which we glued a magnet to the stick in the front of the design and used the attractive magnetic forces to slow down the wheel. The last trial method is a solenoid method which is an applied-friction design and is shown here as the bottom stick in the middle of the design being applied to the disc.

APPENDIX D PUGH AND DECISION MATRICES

Function 1: RC Platform

	Concept					
	Traxxis Slash	Axial Yeti	Traxxis E-Revo 1/16th	Axial SCX10	HPI Trophy Flux	1/16th Slash 4WD
						
Criteria	1	2	3	4	5	6
1/10th scale	+	+		+	+	0
Durable Components	0	-	D	-	+	0
Parts Database	0	0		0	-	0
Cost	+	-	A	-	-	+
Space for Components	+	0		+	+	0
Suspension	0	0	T	0	0	0
Rubber Tires	0	0		0	0	0
Weight (Heavy=+)	+	+	U	+	+	0
Σ^+	4	2		3	4	1
Σ^-	0	-2	M	-2	-2	0
Σs	4	0		1	2	1

Function 2: Microcontroller

Criteria	Concepts		
	1	2	3
CPU Speed	*+	*+	D
RAM	*+	*+	A
ROM	*+	*+	T
Ports	* ₋	*0	U
Cost	*+	* ₋	M
Σ^+	4	3	
Σ^-	1	1	
Σs	3	2	

Function 3: Algorithms

Criteria	Concept							
	Adaptive Cruise	Dynamic Stability	Lane Following	Pedestrian Detection	Road Sign / Light detection	Auto Parking	Multi Vehicle Management	Obstacle Detection
	1	2	3	4	5	6	7	8
Additional Sensors Req'd (more is bad, baseline IMU / Tach)			-	=	=	-	-	=
Practicality		+	=	-	-	-	-	-
Intrigue	D	+	=	+	+	=	+	+
Student Challenge	A	+	=	+	+	=	+	+
Physical Simplicity	T	-	=	=	=	+	+	+
Wiring Simplicity	U	=	-	=	=	-	=	=
Σ^+	M	3	0	2	2	1	3	3
Σ^-		-2	-2	1	-1	-3	-2	-1
Σs		1	-2	-1	-1	-2	1	2

Function 4: Braking Method

	Friction solenoid	Eddy Current	Disc Brakes	4x Motors	Electromagnets	Permanent Magnets
Criteria	1	2	3	4	5	6
Braking Force	-	-		+	-	-
Feasability	+	0	D	+	0	0
Cost of implementation	+	0	A	0	0	0
Complexity of redesign	+	0	T	0	0	0
Σ^+	3	0	U	2	0	0
Σ^-	-1	-1	M	0	-1	-1
Σs	2	-1		2	-1	-1

Function 5: Protective Housing

	Concept			
	Wire Cage	Foam Bumpers	Box-Hinge	Drop Out Assembly
Criteria	1	2	3	4
Ease of Implementation	0		-	-
Structural Rigidity	+	D	+	+
Feasability	0	A	0	-
Cost of implementation	0	T	0	0
Complexity	0	U	0	-
Σ^+	1	M	1	1
Σ^-	0		-1	-3
Σs	1		0	-2

Function 6: CG Modification

	Concept	
	Rack System (Static)	Adjustable Column(Dynamic)
Criteria	1	2
Effectiveness		+
Feasability	D	-
Cost of implementation	A	0
Complexity	T	-
Σ^+	U	1
Σ^-	M	-2
Σs		-1

Function 7: Electrical Layout

	Concept			
	Breadboard	Mother / Daughterboard	Discrete Components	Component Stack
Criteria	1	2	3	4
Adaptability to new senso	+	+		-
Cost	-	-		-
Ease of manufacture	+	-	D	-
Durability	-	+	A	+
Accessibility	+	+	T	-
Self Contained	-	+	U	+
Σ^+	3	4	M	2
Σ^-	-3	-2		-4
Σs	0	2		-2

	Factor 1		Factor 3		Factor 5		Factor 6		Factor 8		Factor 8			
	Physically Robust		Adaptable		Cost		Realistic Dynamics		Manufacturability		Repairability			
	Weight	20%	Weight	15%	Weight	15%	Weight	10%	Weight	20%	Weight	20%	Sum wt.	100%
System	Rank	Wtd. Rank	Rank	Wtd. Rank	Rank	Wtd. Rank	Rank	Wtd. Rank	Rank	Wtd. Rank	Rank	Wtd. Rank	Total	Wtd Total
1	3	0.6	3	0.45	5	0.75	3	0.3	3	0.6	2	0.4	19	3.1
2	5	1	5	1	2.5	0.5	4	0.8	5	1	5	1	26.5	5.3
3	5	1	4	0.8	1	0.2	4	0.8	3	0.6	4	0.8	21	4.2
4	5	1	5	1	1	0.2	5	1	3.5	0.7	4	0.8	23.5	4.7
5	4	0.8	4	0.8	4	0.8	2	0.4	2	0.4	2.5	0.5	18.5	3.7
6	4	0.8	3	0.6	2.5	0.5	4	0.8	3.5	0.7	3.5	0.7	20.5	4.1
7	5	1	5	1	1	0.2	5	1	4	0.8	4	0.8	24	4.8
8	4	0.8	4	0.8	4	0.8	2	0.4	2	0.4	2.5	0.5	18.5	3.7
Notes:	Physically Robust refers to how well protected the electronics are and how well the vehicle can withstand impacts. Rank out of 5.		Adaptable means that new sensors and modifications can be easily added to the vehicle. Rank out of 5.		The cost refers to the overall price to develop a single unit. Rank out of 5.		A vehicle with realistic dynamics has systems and properties similar to a full scale vehicle.		Manufacturability refers to how easily the prototype can be reproduced.		Repairability refers to how easily broken components can be fixed or replaced.			

Decision Matrix used to aid final system design decision.

APPENDIX E DESIGN SAFETY CHECKLIST

ME 428/429/430 Senior Design Project

2016-2017

DESIGN HAZARD CHECKLIST	
Team:	46 - uLaren
Advisor:	Dr. Birdsong
Y N	
<input checked="" type="checkbox"/> <input type="checkbox"/>	1. Will any part of the design create hazardous revolving, reciprocating, running, shearing, punching, pressing, squeezing, drawing, cutting, rolling, mixing or similar action, including pinch points and sheer points?
<input checked="" type="checkbox"/> <input type="checkbox"/>	2. Can any part of the design undergo high accelerations/decelerations?
<input checked="" type="checkbox"/> <input type="checkbox"/>	3. Will the system have any large moving masses or large forces?
<input type="checkbox"/> <input checked="" type="checkbox"/>	4. Will the system produce a projectile?
<input type="checkbox"/> <input checked="" type="checkbox"/>	5. Would it be possible for the system to fall under gravity creating injury?
<input type="checkbox"/> <input checked="" type="checkbox"/>	6. Will a user be exposed to overhanging weights as part of the design?
<input type="checkbox"/> <input checked="" type="checkbox"/>	7. Will the system have any sharp edges?
<input type="checkbox"/> <input checked="" type="checkbox"/>	8. Will any part of the electrical systems not be grounded?
<input checked="" type="checkbox"/> <input type="checkbox"/>	9. Will there be any large batteries or electrical voltage in the system above 40 V?
<input checked="" type="checkbox"/> <input type="checkbox"/>	10. Will there be any stored energy in the system such as batteries, flywheels, hanging weights or pressurized fluids?
<input type="checkbox"/> <input checked="" type="checkbox"/>	11. Will there be any explosive or flammable liquids, gases, or dust fuel as part of the system?
<input type="checkbox"/> <input checked="" type="checkbox"/>	12. Will the user of the design be required to exert any abnormal effort or physical posture during the use of the design?
<input type="checkbox"/> <input checked="" type="checkbox"/>	13. Will there be any materials known to be hazardous to humans involved in either the design or the manufacturing of the design?
<input type="checkbox"/> <input checked="" type="checkbox"/>	14. Can the system generate high levels of noise?
<input type="checkbox"/> <input checked="" type="checkbox"/>	15. Will the device/system be exposed to extreme environmental conditions such as fog, humidity, cold, high temperatures, etc?
<input checked="" type="checkbox"/> <input type="checkbox"/>	16. Is it possible for the system to be used in an unsafe manner?
<input type="checkbox"/> <input checked="" type="checkbox"/>	17. Will there be any other potential hazards not listed above? If yes, please explain on reverse.
<p>For any "Y" responses, add (1) a complete description, (2) a list of corrective actions to be taken, and (3) date to be completed on the reverse side.</p>	

Figure 4: Design Hazard Checklist, Page 1

Description of Hazard	Planned Corrective Action	Planned Date	Actual Date
Design contains revolving action at wheels and potential pinch / shear points in structure	o Affix warning to car to disconnect battery power while programming or maintaining, as forced turning or movement or turning only occurs under battery power	5/10/17	5/10
Design can undergo high acceleration due to powerful motors	o This is part of intended operation, but we will ensure the software cannot prompt excessive acceleration without user input	4/10/17	5/20
Design will be able to go very fast presenting impact hazard	o Can install default software limitation and lock maximum speed to controller input levels	4/10/17	5/20
There will be a large Lithium Polymer battery in the system, running at 25.9V	o We will purchase a fire resistant bag for storing and charging o We will use a polarized connector to prevent shorting or applying reverse voltage o We will design several possible protective systems to prevent impact or puncture	3/10/17	3/20
It will be possible to use the system in an unsafe manner by driving into people or walls	o While this is out of our control, we will write a user manual with an introduction that introduces specific safety hazards that the user should be aware of o We will consider including a training mode that limits acceleration and speed	4/10/17	5/10

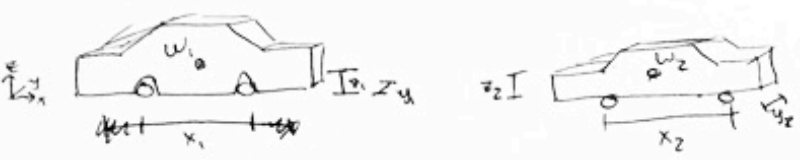
Figure 5: Design Hazard Checklist, Page 2

APPENDIX F SUPPORTING ANALYSIS

Similitude Analysis

SIMILITUDE ANALYSIS

BY DIMENSIONS:



$$W_1 = x_1 y_1 z_1 \rho$$

$$W_2 = x_2 y_2 z_2 \rho \quad \text{where} \quad x_2 = \frac{x_1}{10}, \quad y_2 = \frac{y_1}{10}, \quad z_2 = \frac{z_1}{10}$$

$$W_2 = \frac{x_1 y_1 z_1}{10 \times 10 \times 10} \rho$$

$$= \frac{W_1}{1000} \quad \text{So, weight scales by a factor of } \frac{1}{1000}$$

BY WEIGHT

$$W_2 = \frac{W_1}{10}$$

$$W_1 = x_1 y_1 z_1 \rho$$

$$W_2 = x_2 y_2 z_2 \rho$$

$$x_2 y_2 z_2 \rho = \frac{x_1 y_1 z_1 \rho}{10}$$

Assuming, each dimension scales by the same weight,

$$x_2 = x_1 \sqrt[3]{\frac{1}{10}} = x_1 (0.464)$$

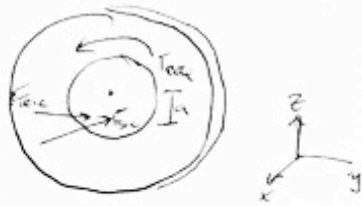
$$y_2 = y_1 \sqrt[3]{\frac{1}{10}} = y_1 (0.464)$$

$$z_2 = z_1 \sqrt[3]{\frac{1}{10}} = z_1 (0.464)$$

So, dimensions each scale by about 0.464.

Braking Force Analysis

Braking Force - Friction Solenoid, CONTINUED



$$F_{fric} = \mu F_{sol}, \text{ ASSUME } \mu \approx 0.7$$

$$T_{brk} = F_{fric} a \quad \text{WHERE } a \text{ IS DISTANCE FROM WHEEL AXEL TO SOLENOID FRICTION POINT. } a \approx 0.2''$$

$$T_{brk} = \mu F_{sol} a$$

$$\begin{aligned} F_{sol} &= \frac{T_{brk}}{\mu a} \\ &= \frac{0.076 \text{ lb}_f \cdot \text{ft}}{(0.7) (0.2'' / 12.5 \frac{\text{in}}{\text{ft}})} \\ &= 6.5 \text{ lb}_f \end{aligned}$$

* THIS A REASONABLE ACTUATING FORCE FROM SOLENOID

Motor Analysis

Maxon Motor Specifications		
Max Continuous Torque	128	<i>mNm</i>
Max Continuous Speed	4860	<i>rpm</i>

RC Platform Specifications		
Traxxas Slash 4X4 Mass	2.64	<i>kg</i>
Rough Estimate of Added Mass	1.5	<i>kg</i>
Estimated Total Mass	4.65	<i>kg</i>
Maximum Wheel Diameter	76	<i>mm</i>

With the assumed operation being on flat ground, we can calculate the estimated continuous speed and acceleration of our system with the selected motors. Velocity can be calculated based on angular velocity and radius.

$$V = \omega * r$$

Since we have angular velocity in rotations per minute and radius in millimeters, we must use unit conversions to get the ft/s needed to compare to our engineering specifications.

$$V = \omega * \frac{2\pi}{60} \frac{\frac{rad}{s}}{rpm} * r * \frac{1}{10 * 2.54 * 12} \frac{ft}{mm}$$

For 4860 rpm and a wheel diameter of 76mm, we can calculate the max sustained speed to be 63 ft/s or 43.3 mph. This is far above our specified speed, but the motors can be controlled at a lower voltage for slower speeds.

Acceleration can be calculated based on mass and force, which we can calculate as a function of torque.

$$A = \frac{F}{m} = \frac{\frac{T}{r}}{m}$$

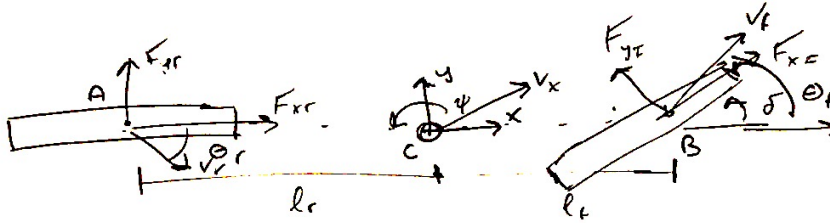
We must convert the final acceleration to ft/s² to compare directly with our engineering specifications.

$$A = \frac{\frac{T * \frac{1}{1000} \frac{Nm}{mNm}}{r}}{m} * 3.28 \frac{ft}{m}$$

For four wheels with 128 mNm of torque, a 76mm wheel diameter, and a mass of 4.65kg, we estimate that we will be capable of an acceleration of 9.5 ft/s². While this exceeds our specification, we can control the motors at a lower voltage to produce lower torque.

Simulink Steering Model Derivation

STATE SPACE MODEL - DERIVATION



$$\sum F_y: m(\ddot{y} + \dot{\psi} v_x) = F_{yf} + F_{yr}$$

$$\sum M_C: I_z \ddot{\psi} = l_f F_{yf} - l_r F_{yr}$$

FOR SMALL SLIP ANGLES, $F_y \propto C_{\alpha}$

$$F_{yf} = 2C_{\alpha}(\delta - \theta_{vf}) \quad \text{WHERE} \quad \theta_{vf} = \frac{\dot{y} + l_f \dot{\psi}}{v_x}$$

$$F_{yr} = 2C_{\alpha}(-\theta_{vr}) \quad \text{WHERE} \quad \theta_{vr} = \frac{\dot{y} - l_r \dot{\psi}}{v_x}$$

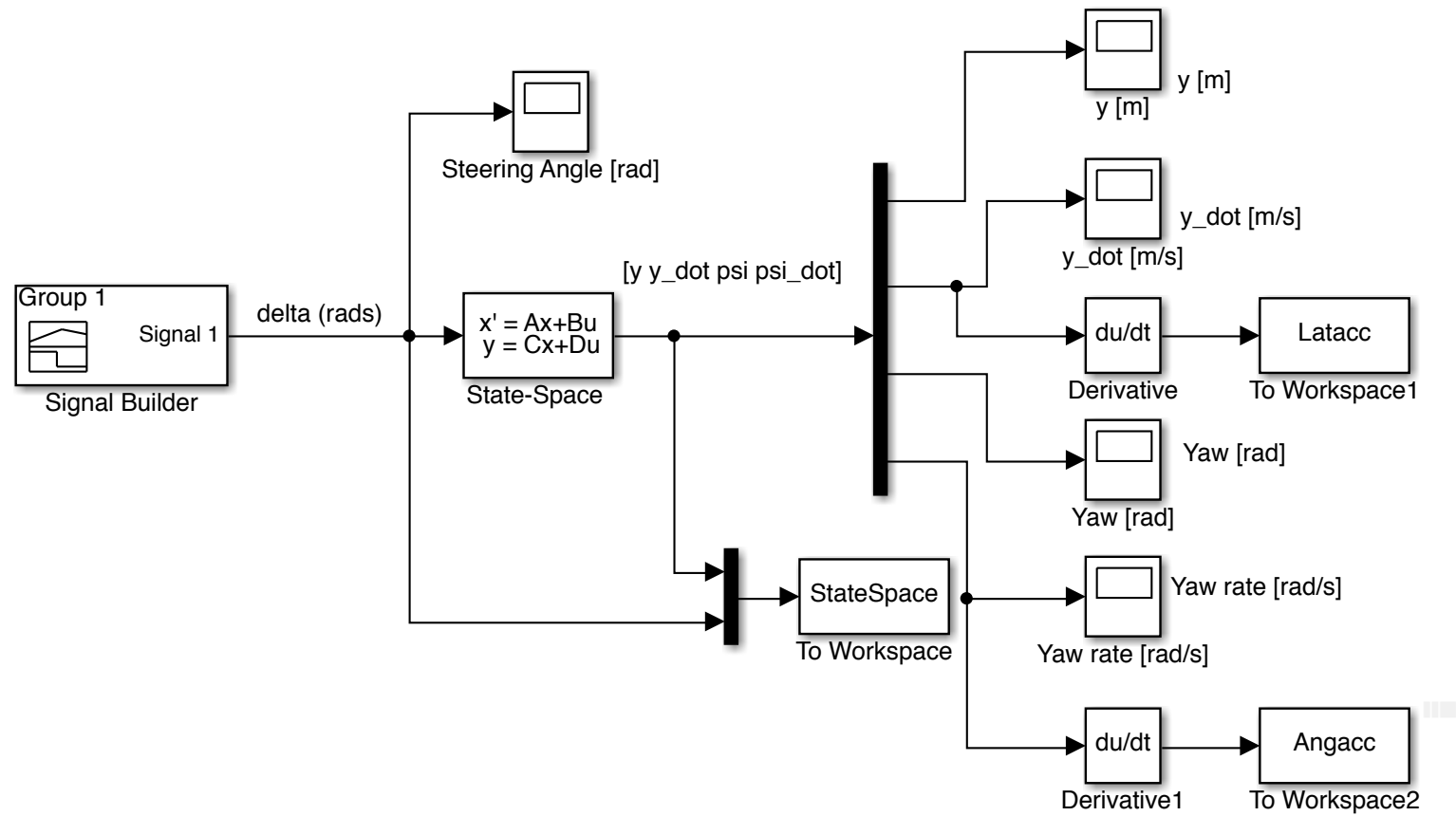
REARRANGING AND SOLVING FOR \ddot{y} , $\ddot{\psi}$, RESPECTIVELY

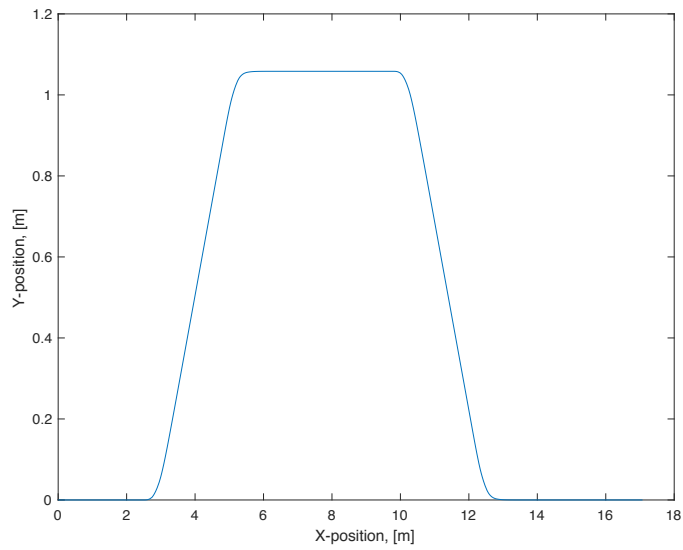
$$\ddot{y} = -\frac{2C_{\alpha} + 2C_{\alpha} l_r}{m v_x} \dot{y} - \dot{\psi} \left[v_x + \frac{2C_{\alpha} l_f - 2C_{\alpha} l_r}{m v_x} \right] + \frac{2C_{\alpha}}{m} \delta$$

$$\ddot{\psi} = \frac{-2l_f C_{\alpha} - 2l_r C_{\alpha}}{I_z v_x} \dot{y} - \dot{\psi} \frac{2l_r^2 C_{\alpha} + 2l_f^2 C_{\alpha}}{I_z v_x} + \frac{2l_f C_{\alpha}}{I_z} \delta$$

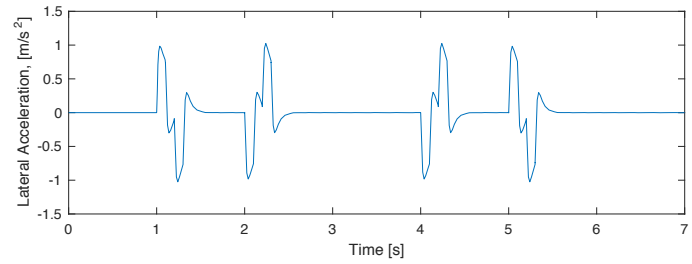
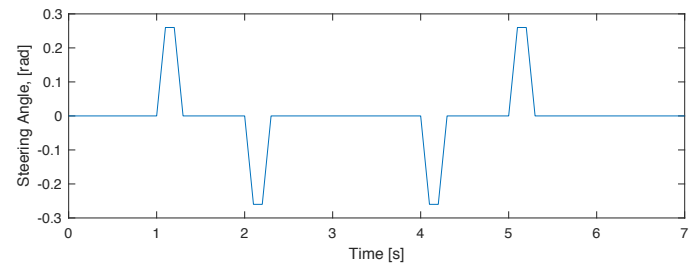
WHICH CAN BE SOLVED FOR AS A SYSTEM OF COUPLED, 1ST ORDER DIFFERENTIAL EQUATIONS

Simulink Steering Model

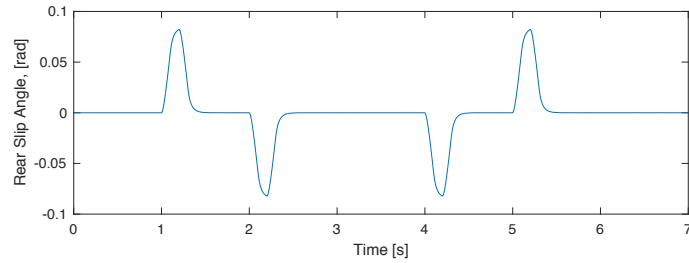
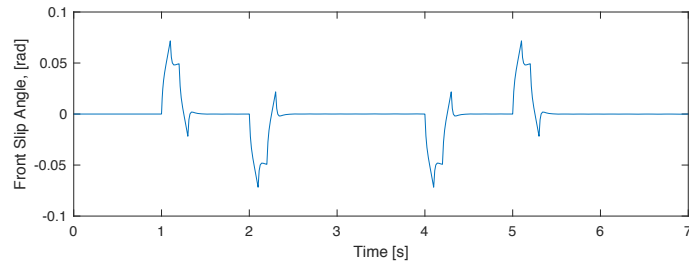




Global path of vehicle

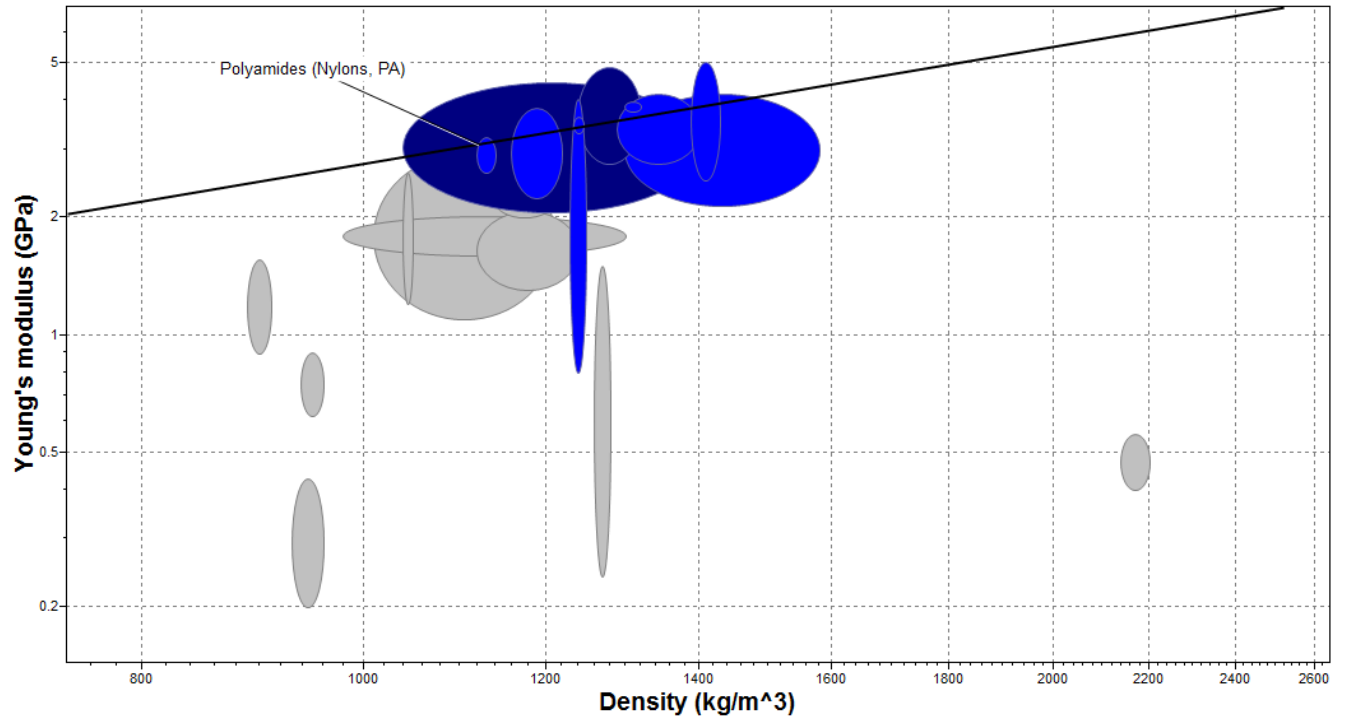


Top: Steering angle input, Bottom: Lateral acceleration

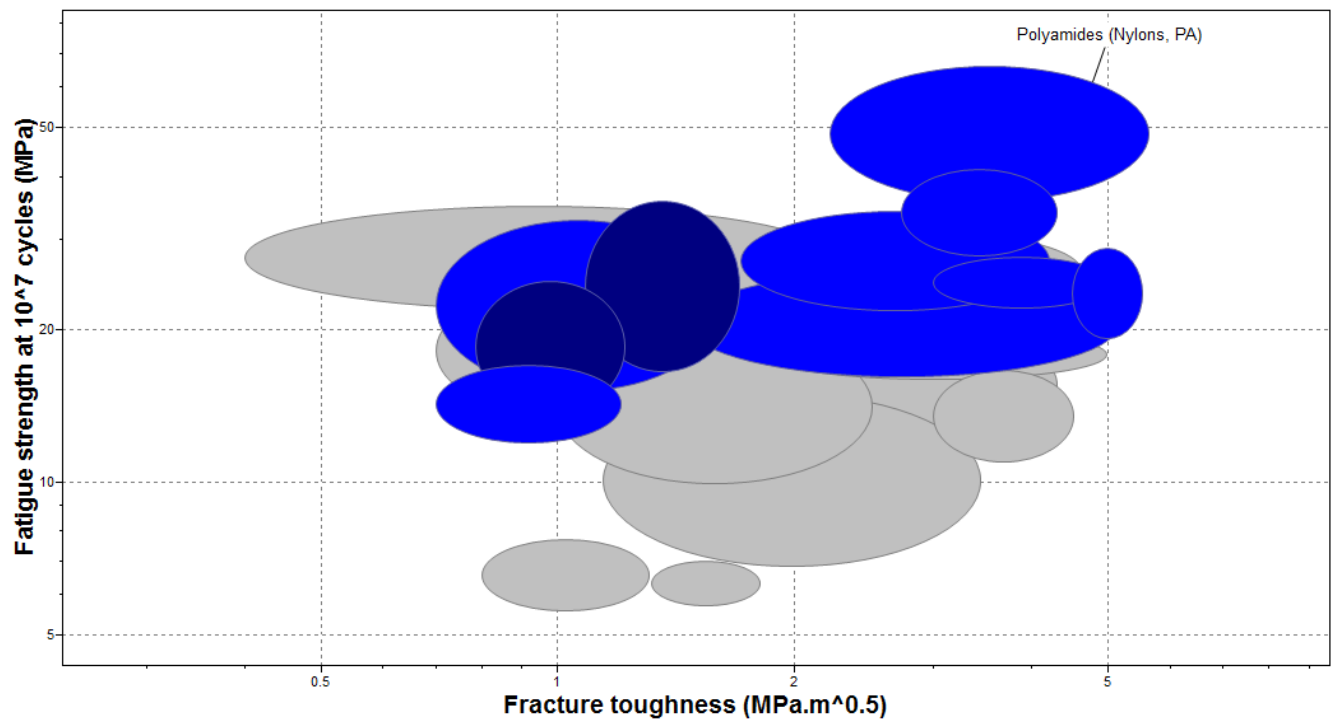


Top: Front slip angle, Bottom: Rear slip angle

Materials Selection – Ashby Charts

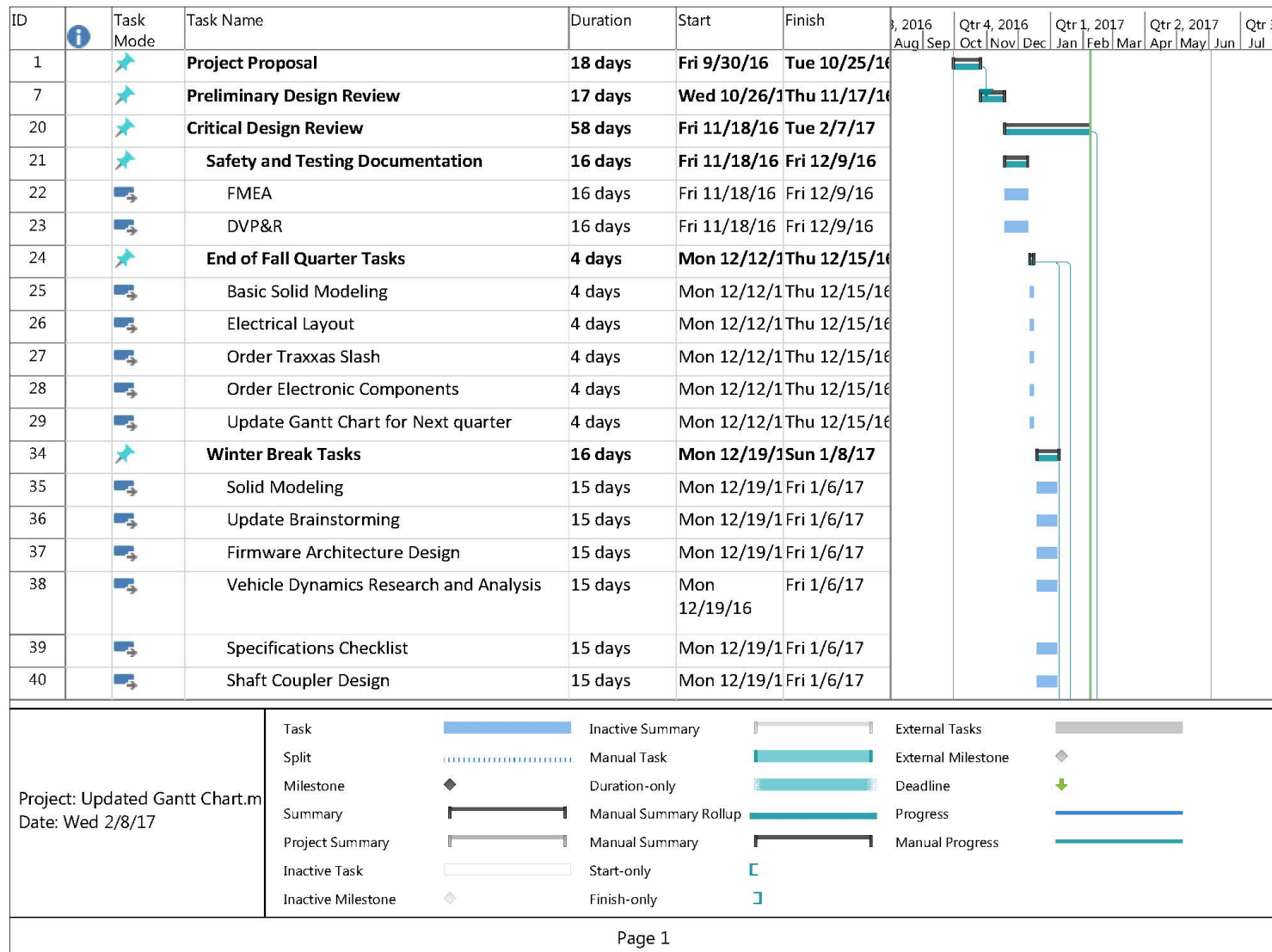


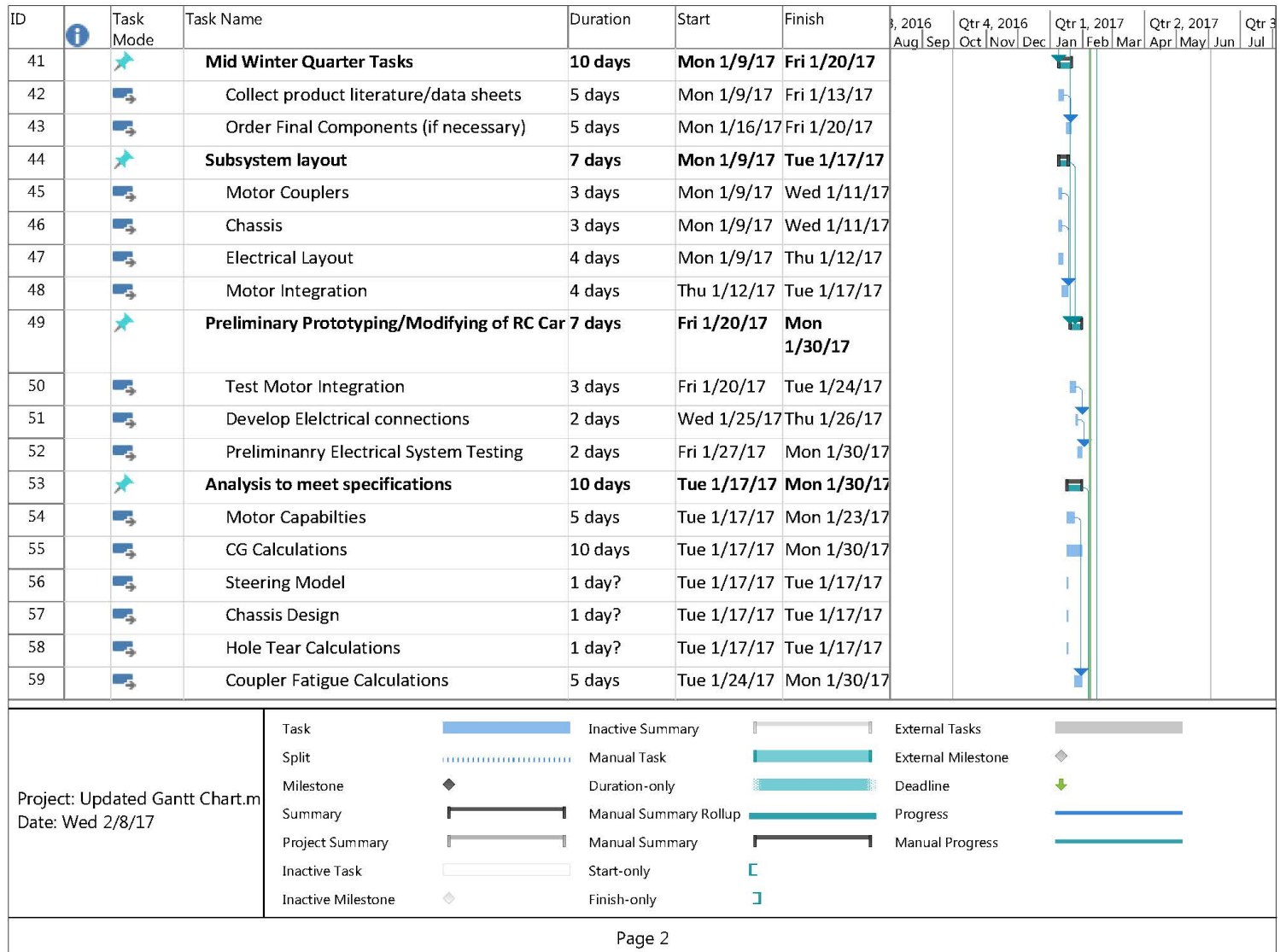
Young's Modulus and density. Ideally top left.

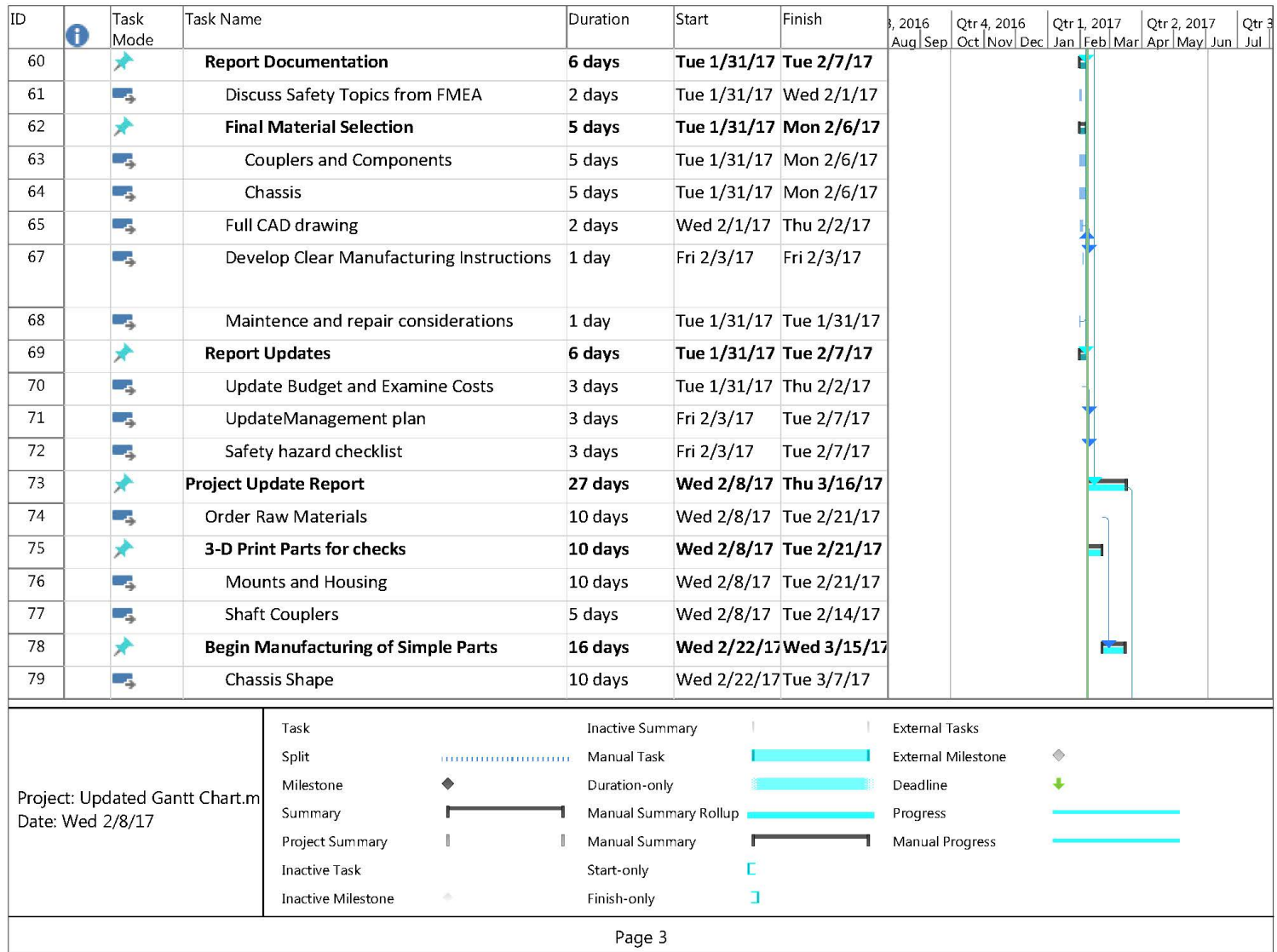


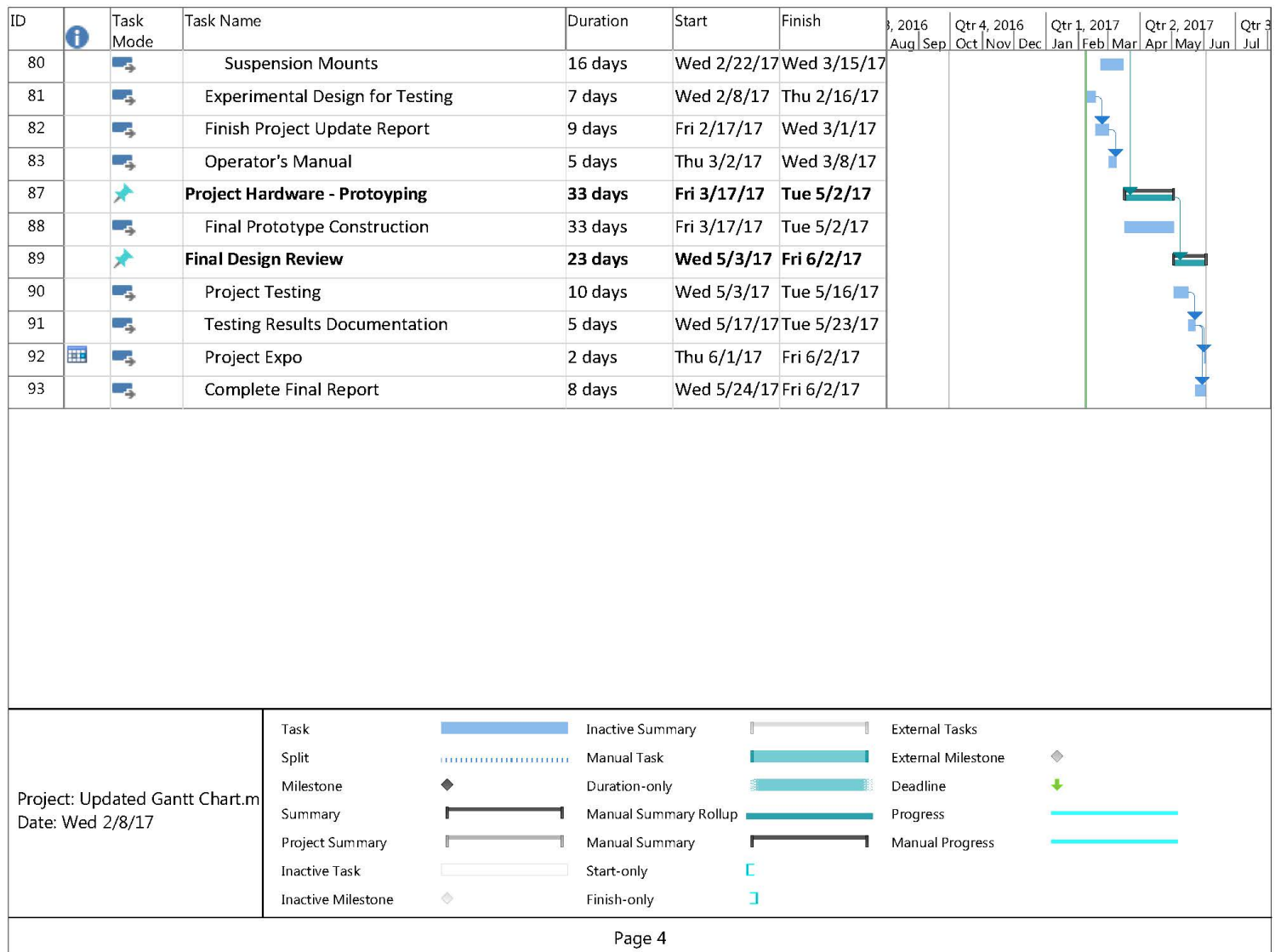
Fatigue strength and fracture toughness. Ideally top right.

APPENDIX G GANTT CHART









APPENDIX H DVP&R AND DFMEA

ME428 DVP&R Format													
Report Date		12/1/16	Sponsor	Dr. Birdsong					System	Intelligent Vehicle Platform	REPORTING ENGINEER:		
TEST PLAN													
Item No	Specification or Clause Reference	Test Description	Acceptance Criteria	Test Responsibility	Test Stage	SAMPLES TESTED		TIMING		TEST RESULTS			NOTES
						Quantity	Type	Start date	Finish date	Test Result	Quantity Pass	Quantity Fail	
1	Trackwidth	Center of wheel to center of wheel measurement	275 mm +/- 15%	Chris	DV	1	B	5/22/17	5/23/17	276 mm - Pass	1	0	Uncertainty predicted +/- 1 mm
2	Wheelbase	Center of wheel to center of wheel measurement	406 mm +/- 15%	Chris	CV/DV	1	B	5/22/17	5/23/17	401 mm - Pass	1	0	Uncertainty predicted +/- 1 mm
3	Weight	Measure on scale	5.8 kg Max	Chris	CV	1	B	5/22/17	5/23/17	3.8 kg - Pass	1	0	Uncertainty analysis predicted +/- 50 g
4	Longitudinal CG Position	Position from center of rear wheel	203 mm +/- 25%	Chris	DV	1	B	5/22/17	5/23/17	204 mm - Pass	1	0	Uncertainty analysis predicted +/- 2 mm
5	CG Height	Lift front axle, measure reaction force on scale	50 mm Min	Chris	DV	1	B	5/22/17	5/23/17	75 mm - Pass	1	0	Uncertainty analysis predicted +/- 2 mm
6	Top Vehicle Speed	Test max speed on concrete for 10 yards	6.7 +/- 1 m/s	Chris	DV	1	B	5/25/17	5/25/17	Set Electronically in motor controller software	1	1	
7	Maximum Acceleration	Test max acceleration on concrete	2 m/s ² +/- 50%	Chris	CV/DV	1	B	5/25/17	5/25/17	Set Electronically in motor controller software	1	1	
8	Turning Radius	Maximum steering angle, drive vehicle in circle	1600 mm +/- 25%	Chris	DV	1	B	N/A	N/A	Did not test			Theoretically calculation is a pass - 1300 mm
9	Rollover at top speed	Visually see inside wheel lose traction	Must Meet Standard	Chris	DV	1	B	N/A	N/A	Did not test			
10	Physically Robust	Visual inspection	Full Evaluation	Chris & Jay	DV	1	A	5/22/17	5/23/17	Pass	1	0	See Areas for improvement section in FDR.
11	Visual latency	Run sample collision avoidance algorithm	Must Meet Standard	Evan	DV		B	N/A	N/A	Did not test			
12	Works with Simulink	Development of algorithms	Must Meet Standard	Evan & Jay	CV	1	A	6/1/17	6/1/17	Pass	1	1	
13	Suspension	N/A (Yes or no)	Must Meet Standard	Jay	CV	1	B	6/1/17	6/1/17	Pass	1	1	
14	Digital I/O Ports	N/A (Yes or no)	NO ERROR (Must be able to hold 3	Evan	CV	1	B	6/1/17	6/1/17	Pass	1	1	
15	Independently Powered Wheels	N/A (Yes or no)	Must Meet Standard	Jay	CV/DV	1	B	6/1/17	6/1/17	Pass	1	1	
16	Tetherless	N/A (Yes or no)	Must Meet Standard	Everyone	CV	1	B	6/1/17	6/1/17	Fail - Not tetherless	1	1	Future modification necessary
17	Autonomous/Hybrid Control	N/A (Yes or no)	Must Meet Standard	Evan	CV/DV	1	A	6/1/17	6/1/17	Pass	1	1	
18	Battery Life	6 cycles of 10 min continued use with 15 min "reprogramming"	Must Meet Standard	Jay	CV/DV		B	N/A	N/A	Did not test			

Item / Function	Potential Failure Mode	Potential Effect(s) of Failure	Severity	Potential Cause(s) / Mechanism(s) of Failure	Occurrence	Criticality	Recommended Action(s)	Responsibility & Target Completion Date	Actions Taken	Severity	Occurrence	Criticality
Tests Simulink Algorithms	Loss of control	Car moves to fast	3	Bad Algorithm	7	21	Remote killswitch, sensors check/agree	Evan, 5/17/2017	Remote killswitch, maxiumum motor speed	3	7	21
				Component	3	9					3	9
				Faulty Sensor Data	2	6					2	6
		Wrong vehicle path	6	Bad Algorithm	7	42				6	7	21
				Component	3	18					3	9
				Not enough output info from sensors	4	24					4	12
				Faulty Sensor Data	2	12					2	6
Battery	Overheats	Loss of power	2	Excess current drainage	3	6	Position battery in safe location, limit operating time	Chris, 5/17/2017	LiPo safe bag for storage, voltage regulator to protect fragile electronics	2	3	9
				Overuse	3	6					3	9
	Over drained	Loss of power	2	Excess current drainage	3	6				2	3	9
				Overuse	3	6					3	9
	Punctured Casing	Exposure to toxic chemicals	8	Sharp object comes into contact with battery at high speeds	3	24				8	3	9
		Catches fire	9	Sharp object comes into contact with battery at high speeds	3	27				9	3	9
Protective Housing	Broken housing	Pinch points	6	High speed impacts from loss of control	4	24	Durable cage, killswitch to prevent loss of control, speed control	Jay, 5/17/2017	None taken, needs adaptation	6	4	12
				Fatigue failure from use	3	18					3	9
		Cage cannot protect components	5	High speed impacts from loss of control	4	20				5	4	12
				Fatigue failure from use	3	15					3	9
	Housing falls off	Damage to electronics	6	High speed impacts from loss of control	4	24				6	4	12
				Fatigue failure from	3	18					3	9
		Replacement of expensive parts	3	High speed impacts from loss of control	4	12				3	4	12
Motors	Burnout	No power to shafts	3	Motor lifetime	1	3	Design couplers for infinite life, detailed analysis	Jay, 5/17/2017	CNC couplers	3	1	3
	Coupler attachment breaks	Vehicle spins out of control	6	Fatigue failure in	3	18				6	3	9
				High speed impacts	4	24					4	12
Performs Like a Real Vehicle	Too stable	Roll over not possible	3	Center of gravity too low	4	12	Add wegiht/vary CG, limit motor supply current	Chris, 5/17/2017	Used existing Slash suspension and steering geometry, battery below chassis to lower CG	3	4	12
	Large power/weight ratio	Burn-out	2	Motors too powerful	5	10				2	5	15
		Can't test stbality	4	Motors too powerful	5	20				4	5	15
	Loss of traction	Can't test stablilty control	4	Excessive speeds	3	12				4	3	9
				Car too light	5	20					5	15
				Tires don't provide enough grip	3	12					3	9
		Vehicle crashes	5	Excessive speeds	3	15				5	3	9
				Car too light	5	25					5	15
				Tires don't provide enough grip	3	15					3	9

APPENDIX I BILL OF MATERIALS

CATEGORY	ITEM DESCRIPTION	QTY	COST PER UNIT	TOTAL COST	SOURCE
Shipping, etc.	Maxon Order + ?			16.00	
Traxxas Slash	Traxxas Slash 4x4 4WD	1	449.95	483.70	Amazon: https://www.amazon.com/Traxxas-68086-3-Electric-Vehicle-Colors/dp/B0115ERJOO/ref=sr_1_1?s=toys-and-games&ie=UTF8&qid=1486152828&sr=1-1&keywords=traxxas+slash+4x4&refinements=p_36%3A1253564011
Electronics	9-DOF Absolute Orientation IMU	1	34.95	34.95	Adafruit: https://www.adafruit.com/products/2472
	Raspberry Pi 3B	2	39.95	79.90	Adafruit: https://www.adafruit.com/products/3055
	Pi Cobbler Plus	1	6.95	6.95	Adafruit: https://www.adafruit.com/products/2029
	Maxbotix Ultrasonic Rangefinder	1	33.95	33.95	Adafruit: https://www.adafruit.com/products/983
	Raspberry Pi Cam	1	29.95	29.95	Adafruit: https://www.adafruit.com/products/3099
	Pi Cam Cable	1	1.95	1.95	Adafruit: https://www.adafruit.com/products/1648
	Keyfob Remote Control Button	1	6.95	6.95	Adafruit: https://www.adafruit.com/products/1648
	Keyfob Remote Receiver	1	4.95	4.95	Adafruit: https://www.adafruit.com/products/1097
	4GB SD Card	2	9.95	19.90	Adafruit: https://www.adafruit.com/products/1121
	Teensy 3.6 Microcontroller	1	37.50	37.50	Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?Detail&itemSeq=218432068&uq=636220001070721314

Circuit Board and Parts	DC Level Shifter	1	18.73	18.73	Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?Detail&itemSeq=218432067&uq=636220001070721314
	CAN Transceiver	2	2.19	4.38	Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?Detail&itemSeq=218432071&uq=636220001070721314
	RGB Led Indicator	2	0.57	1.14	Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?Detail&itemSeq=218432070&uq=636220001070721314
	Voltage Level Translator	2	0.85	1.70	Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?Detail&itemSeq=218432069&uq=636220001070721314
	24 Pin Female/Male Header Connector	2	1.64	3.28	Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?Detail&itemSeq=218432066&uq=636220001070721314
	4 Pin Female/Male Header Connector	2	0.60	1.20	Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?Detail&itemSeq=218432065&uq=636220001070721314
	7 Pin Female/Male Header Connector	1	0.78	0.78	Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?Detail&itemSeq=218432064&uq=636220001070721314
	6 Position Header Connector Through Hole Tin	2	0.64	1.28	Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?Detail&itemSeq=218432062&uq=636220001070721314
	40 Pin Female/Male Header Connector	2	2.31	2.31	Digikey: http://www.digikey.com/scripts/DkSearch/dksus.dll?Detail&itemSeq=218432061&uq=636220001070721314
	Custom Circuit Board	1	37.50	37.50	Osh Park:
	Battery	1	38.70	38.70	HobbyKing: https://hobbyking.com/en_us/zippy-compact-4000mah-7s-25c-lipo-pack.html

	0.1 μ F Capacitor	25	Pack	0.33	http://www.digikey.com/products/en?newproducts=0&keywords=311-1343-1-ND&pkeyword=311-1343-1-ND&v=
	0.01 μ F Capacitor	25	Pack	0.33	http://www.digikey.com/products/en?newproducts=0&keywords=311-1085-1-ND&pkeyword=311-1085-1-ND&v=
	1 μ F Capacitor	3	Pack	0.99	http://www.digikey.com/products/en?keywords=478-8234-1-ND
	10 μ F Capacitor	3	Pack	1.14	http://www.digikey.com/products/en?keywords=399-3684-1-ND
	5 A Fuse	3	Pack	7.65	http://www.digikey.com/products/en?keywords=F2969CT-ND
	26.7 Ω Resistor	10	Pack	0.13	http://www.digikey.com/products/en?keywords=311-26.7HRCT-ND
	3.3 Ω Resistor	10	Pack	0.10	http://www.digikey.com/products/en?keywords=311-3.3GRCT-ND
	Fine Pitch Screw Terminals 4pos	5	Pack	5.25	http://www.digikey.com/products/en?keywords=ED10563-ND
	6pos Female header	1	0.68	0.68	http://www.digikey.com/products/en/connectors-interconnects/rectangular-connectors-headers-receptacles-female-sockets/315?k=S7004-ND&pkeyword=S7004-ND
	Screw Terminal 3 Pos	2	0.50	1.00	http://www.digikey.com/products/en?keywords=ED2741-ND
	Screw Terminal 2 Pos	2	0.38	0.76	http://www.digikey.com/products/en?keywords=ED2740-ND
Motor	EC45 Motor (PN:397172)	5	20.00 (Sponsorship Price)	100.00	Maxon: http://www.maxonmotorusa.com/maxon/view/product/motor/ecmotor/ecflat/ecflat45/397172
	EPOS4 Compact 50/5 CAN Controller (PN: 541718)	5	20.00 (Sponsorship Price)	100.00	Maxon: http://www.maxonmotorusa.com/maxon/view/product/control/Positionierung/EPOS-4/541718

Raw Materials	Aluminum Round Stock 1 ft - 5 mm Tight Tol. Rod	1	7.99	8.63	McMaster – Carr 6940T12
	Aluminum Round Stock 1 ft - 10 mm Unpolished	1	1.80	1.94	McMaster – Carr 4634T36
	Aluminum Bar Stock 6" x 1" x 3/4"	1	4.27	4.61	McMaster – Carr 8975K14
	Aluminum Bar Stock 12" x 1.625" x 1.625"	1	19.12	20.6	McMaster – Carr 9008K48
	Aluminum Bar Stock 12" x 1" x 3/4"	1	6.71	7.25	McMaster – Carr 8975K14
	Aluminum Sheet Stock 24" x 4" x 0.08"	1	13.50	14.58	McMaster – Carr 89015K192
	Nylon Sheet Stock 12" x 12" x 3/8"	1	38.52	41.60	McMaster – Carr 8540K117
Fasteners and Spacers	Miscellaneous	1	30.00	30.00	Local or Amazon

APPENDIX K TESTING RESULTS AND DATA COLLECTION

Weight Data

Weight Data				Vehicle Measurements		
Rear Left [g]	Rear Right [g]	Front Left [g]	Front Right [g]	U_scale [g]	Length [mm]	U_length [mm]
867	1023	1016	946	0.5	401	5

Longitudinal CG position and uncertainty

Calculations		
R_r	18540.9 [N]	Rear Weight
W	37788.1 [N]	Total Weight
b	196.8 [mm]	Distance from front to CG
a	204.2 [mm]	Distance from rear to CG
Wr_unc	1890.7 g	Rear weight + .707
Wf_unc	1962.7 g	Front Weight + .707
L_unc	406.0 mm	Wheelbase + Unc
s1_a	0.0 mm	Sensitivity to rear weight
s2_a	0.0 mm	Sensitivity to front weight
s3_a	2.5 mm	Sensitivity to wheelbase
U total	2.5 mm	Total uncertainty in "a"

Modified weight data

Measurement	Value	Unc.	Units	Description
FWl	1937	0.71	g	Front Weight Level
FWr	1996	0.71	g	Front Weight Raised
W	3832	1.00	g	Total Weight
L	401	5.00	mm	Wheelbase
H	118	1.00	mm	Rear Lift Height
r	55	1.00	mm	Effective wheel radius

Vertical CG position and uncertainty analysis.

Calculate		
CG Height	75.1 mm	
s_FWl	-0.2 g	Sensitivity to front weight
s_FWr	0.2 g	Sensitivity to rear weight
s_W	0.0 g	Sensitivity to total weight
s_L	0.5 mm	Sensitivity to wheelbase
s_H	-0.2 mm	Sensitivity to lift height
s_r	1.0 mm	Sensitivity to wheel radius
Uncertainty	1.2 mm	Total Uncertainty

APPENDIX L REPRODUCTION COSTS

Traxxas Parts

System	Description	Part #	Quantity	Cost (Each)	Total
Front Suspension and Steering	Front Driveshaft Assembly	6851X	2	\$10.00	\$20.00
	Caster Blocks	6832	1	\$5.00	\$5.00
	Steering Blocks	6837	1	\$5.00	\$5.00
	Steering Turnbuckle (58 mm)	5539	1	\$7.50	\$7.50
	Suspension Turnbuckle (49 mm)	3643	1	\$7.50	\$7.50
	Bellcranks	6845	1	\$12.00	\$12.00
	Spring Progressive +10%	6863	1	\$5.00	\$5.00
	Steering Linkage	6846	1	\$3.00	\$3.00
	Bumper	6853	1	\$10.00	\$10.00
Rear Suspension	Rear Driveshaft Assembly	6852X	2	\$10.00	\$20.00
	Carriers	1952	1	\$3.00	\$3.00
	Suspension Turnbuckle (39 mm)	3644	1	\$7.00	\$7.00
	Bumper	6838	1	\$10.00	\$10.00
	Spring Progressive +10%	6867	1	\$6.50	\$6.50
Both (Front and Rear)	A-Arm (Both)	3655X	2	\$10.00	\$20.00
	Ultrashocks	2662	1	\$42.00	\$42.00
	Wheel Nuts	3647	1	\$3.25	\$3.25
	Wheel Hubs	1654	2	\$2.00	\$4.00
	Ball Bearing	5119	1	\$7.00	\$7.00
	Ball Bearing	5116	3	\$3.50	\$10.50
	Suspension Pins	6834	1	\$4.00	\$4.00
	Wheels	5873	2	\$50.00	\$100.00
Chassis/Electronics	Receiver	6533	1	\$50.00	\$50.00
	Transmitter	6517	1	\$60.00	\$60.00
	Servo	2075	1	\$40.00	\$40.00
Hardware	Shoulder Screw	3642X	2	\$3.00	\$6.00
	M3 x 15 mm rounded (6)	2579	2	\$2.50	\$5.00
	M3 x 6mm flat (6)	3932	1	\$2.50	\$2.50
Grand Total					\$475.75

Mechanical Hardware and Raw Materials

Description	Quantity	Price (Each)	Total
Nylon Sheet Stock (12" x 12" x 3/8")	1	\$38.52	\$38.52
Aluminum Round Stock (1 ft - 5 mm Tight Tol. Rod)	1	\$7.99	\$7.99
Aluminum Bar Stock (1' x 2" x 5/16")	1	\$5.48	\$5.48
Aluminum Bar Stock (2' x 3/4" x 3/4")	1	\$8.78	\$8.78
Aluminum 3003 Sheet Stock (2' x 6" x 0.08")	1	\$8.96	\$8.96
Acrylic sheet (6" x 4" x 7/64")	1	\$6.42	\$6.42
Heat-Set Inserts M3, 3.8 mm (100 pack)	1	\$12.30	\$12.30
Aluminum 45 mm threaded hex standoff, M3 thread	6	\$1.57	\$9.42
Aluminum 10 mm threaded hex standoff, M4 thread	3	\$1.27	\$3.81
Aluminum 20 mm threaded hex standoff, M4 thread	3	\$1.57	\$4.71
M3 x 10 mm socket (50 pack)	1	\$5.43	\$5.43
M3 x 6mm rounded (100 pack)	1	\$6.88	\$6.88
M3 x 4 mm flat (50 pack)	1	\$7.14	\$7.14
M4 x 14 mm socket (50 pack)	1	\$9.84	\$9.84
M2.5 x 20 mm socket (25 pack)	1	\$7.10	\$7.10
M2 x 12 mm socket (50 pack)	1	\$5.00	\$5.00
Grand Total			125.84

APPENDIX J PART AND ASSEMBLY DRAWINGS

100 – Top Level Assembly

- 101 – Top Level Assembly Exploded**
- 102 – Chassis Mount Spacer A**
- 103 – Chassis Mount Spacer B**
- 104 – Chassis Mount Spacer C**
- 105 – M4 x 07 14 MM Socket Screw**
- 106 – M4 x 0.7 Nut**
- 107 – M4 x 0.7 22 MM Socket Screw**
- 108 – M4 x 0.7 10 MM Socket Screws**

200 – Front Motor Block Assembly

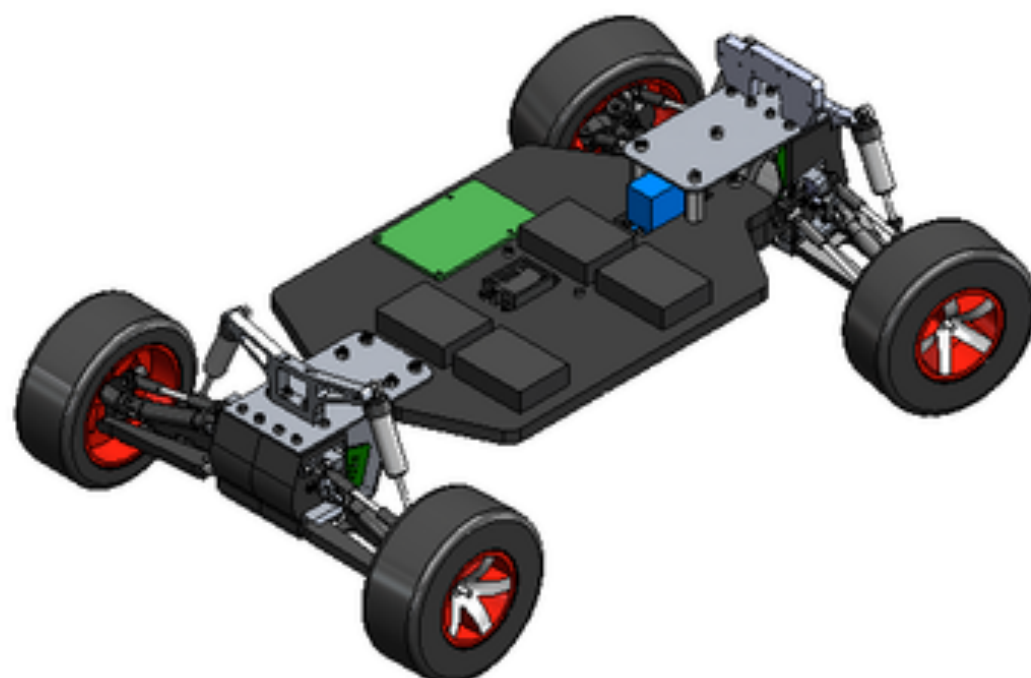
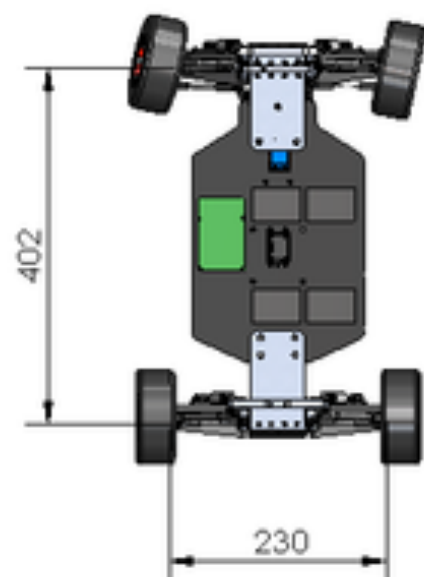
- 201 – Front Motor Block Assembly Exploded**
- 202 – Front Suspension Mount Drawing**
- 203 – Front Bottom Chassis Mount Drawing**
- 204 – Front Top Chassis Mount Drawing**
- 205 – Steering Post Dowel Drawing**
- 206 – Left Steering Bell crank**
- 207 – Right Steering Bell crank**
- 208 – M3 x 0.5 6mm Socket Screw**
- 209 - M3 x 0.5 12mm Socket Screw**
- 211 – Steering Linkage Drawing**
- 212 – M4 x 0.7 6mm Shoulder**
- 213 – Small Steering Turnbuckle**

220 – Front Right Motor Block Assembly

- 221 – Front Right Traxxas Assembly²⁴¹**
- 222 – Front Right Motor Housing Drawing**
- 223 - Front Turnbuckle Mount Drawing**
- 224 – Front A-Arm Mount Drawing**
- 225 – Shaft Coupler Assembly**

- 225A** – Shaft Coupler A Drawing
 - 225B** – Shaft Coupler B Drawing
 - 226** – Maxon Motor Data Sheet
 - 227** – M3 x 0.5 4mm Flat Screw
 - 228** – M2 x 0.4 12mm Socket Screw
 - 229** – M3 x 0.5 5mm Rounded Screw
 - 230** – M3 X 0.5 14mm Rounded Screw
- 240** – Front Left Motor Block Assembly
 - 241** - Front Left Traxxas Assembly
 - 242** – Front Left Motor Housing Drawing
- 300** – Rear Motor Block Assembly
 - 301** - Rear Motor Block Assembly Exploded
 - 302** – Rear Suspension Mount Drawing
 - 303** – Rear Bottom Chassis Mount Drawing
 - 304** – Rear Top Chassis Mount Drawing
- 310** – Rear Right Motor Block Assembly
 - 311** – Rear Right Motor Housing Drawing
 - 312** – Rear Right Traxxas Assembly
 - 314** - Rear Turnbuckle Mount Drawing
 - 315** – Rear A-Arm Mount Drawing
- 320** – Rear Left Motor Block Assembly
 - 321** – Rear Left Motor Housing Drawing
 - 322** – Rear Left Traxxas Assembly
- 400** – Chassis Assembly
 - 401** – Chassis Drawing
 - 402** – Traxxas Servo
 - 403** – Traxxas Receiver
 - 404** – Maxon Motor Driver Datasheet
 - 405** – IMU Datasheet
 - 406** – Pi Camera Datasheet

- 407** – Ultrasonic Distance Sensor Datasheet
- 408** – Raspberry Pi Datasheet
- 409** – Battery Datasheet
- 410** – Battery Plate
- 411** – 45 MM Hex Standoff
- 412** – Sensor Array
- 413** – IMU Mount
- 414** – Traxxas Battery Cage
- 415** – M3 X 0.5 14mm Flat
- 416** – Laser Rangefinder Datasheet
- 420** – Motherboard Schematic
 - 421** – Motherboard Layout
 - 422** – Teensy 3.6 Datasheet
 - 423** – Voltage Regulator Datasheet
 - 424** – CAN Transceiver Datasheet
 - 425** – Indicator LED Datasheet
 - 426** – Power LED Datasheet
 - 427** – Schottky Diode Datasheet
 - 428** – Screw Terminal Datasheet
 - 429** – Male Header Datasheet
 - 430** – Female Header Datasheet
 - 431** – LittleFuse Datasheet
- 450** – Cabling Diagram
 - 451** – Minifit Jr Receptacle
 - 452** – Minifit Jr Crimp Receptacle
 - 453** – Microfit 3.0 Receptacle
 - 454** – Microfit 3.0 Crimp Receptacle
 - 455** – Klik-Mate 1.5
 - 456** – Klik-Mate 1.5 Crimp Pin



SCALE: 1:4

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: VARIOUS

TITLE: TOP LEVEL ASSEMBLY

DRAWN BY: CG

DWG #: 100

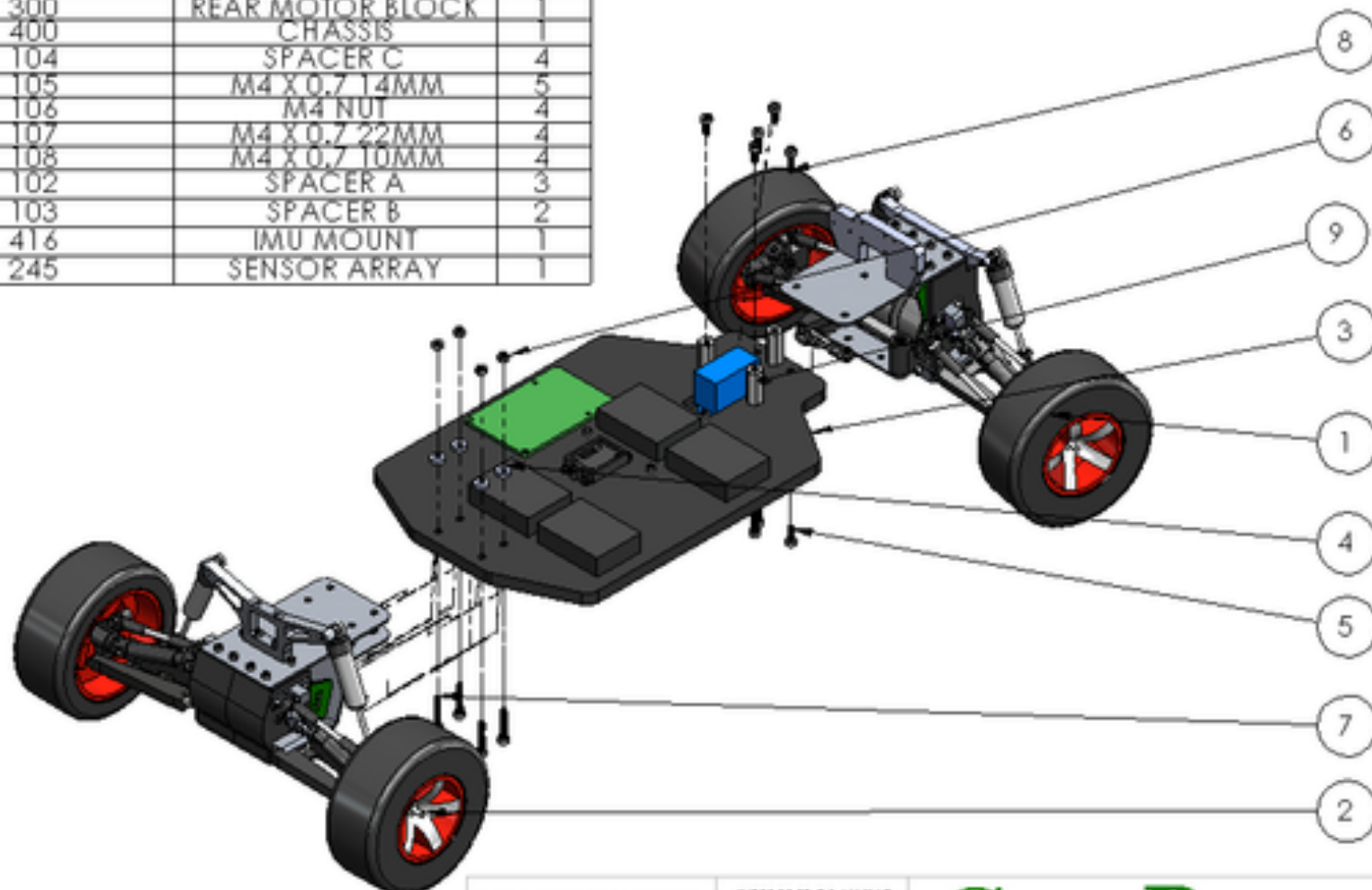
SHEET 1 OF 1

SCALE: 1:8

REV SIZE

A

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	200	FRONT MOTOR BLOCK	1
2	300	REAR MOTOR BLOCK	1
3	400	CHASSIS	1
4	104	SPACER C	4
5	105	M4 X 0.7 14MM	5
6	106	M4 NUT	4
7	107	M4 X 0.7 22MM	4
8	108	M4 X 0.7 10MM	4
9	102	SPACER A	3
10	103	SPACER B	2
11	416	IMU MOUNT	1
12	245	SENSOR ARRAY	1



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: VARIOUS

TITLE: TOP LEVEL ASSEMBLY

DRAWN BY: CG

DWG #: 101

SHEET 1 OF 1

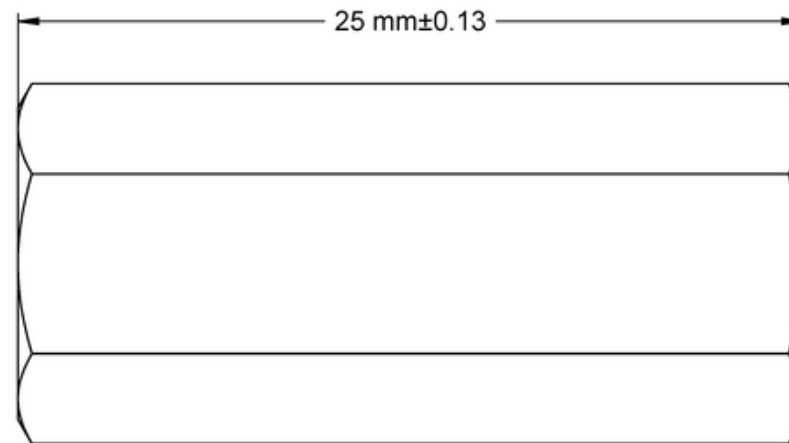
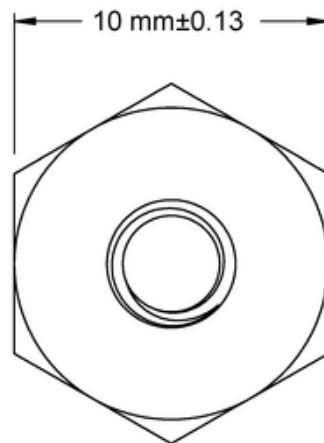
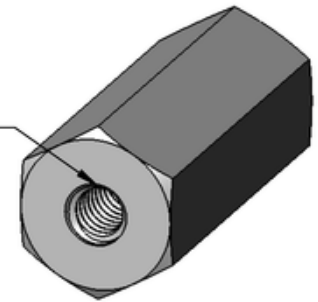
SCALE: 4

REV

SIZE

A

M4 x 0.7 mm Thread



McMASTER-CARR CAD

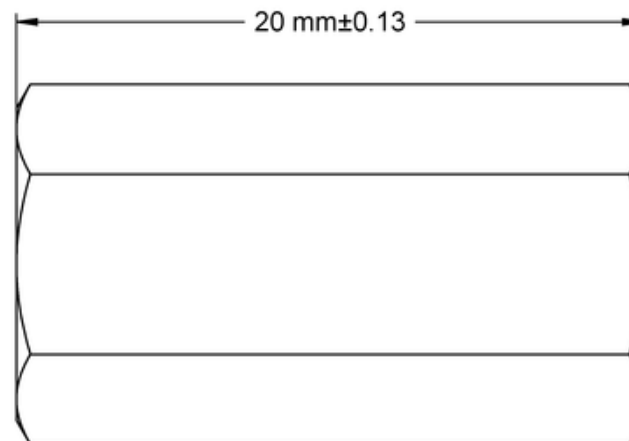
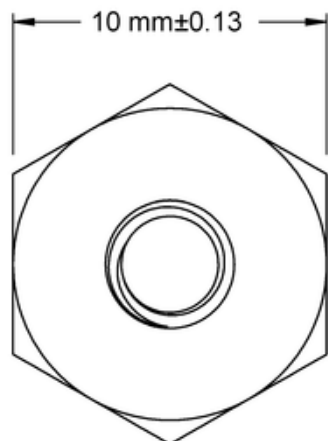
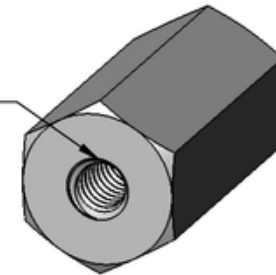
<http://www.mcmaster.com>
© 2015 McMaster-Carr Supply Company

Information in this drawing is provided for reference only.

PART
NUMBER **95947A746**

Female Threaded
Hex Standoff

M4 x 0.7 mm Thread



McMASTER-CARR CAD

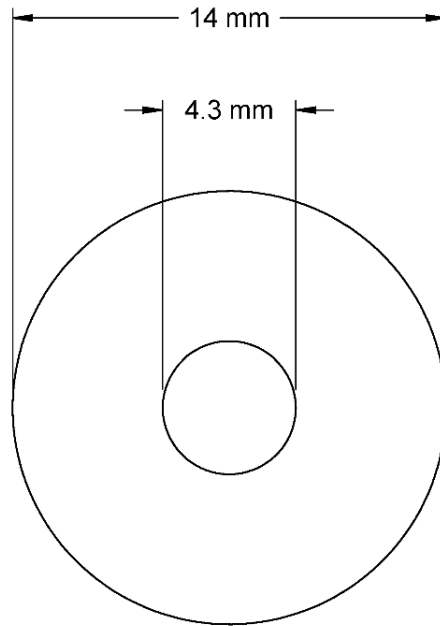
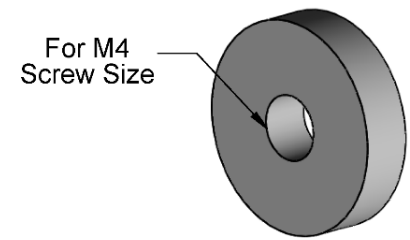
<http://www.mcmaster.com>
© 2015 McMaster-Carr Supply Company

Information in this drawing is provided for reference only.

PART
NUMBER

95947A744

Female Threaded
Hex Standoff



Washer may vary from
2.5 mm to 3.4 mm in thickness.

McMASTER-CARR CAD

<http://www.mcmaster.com>

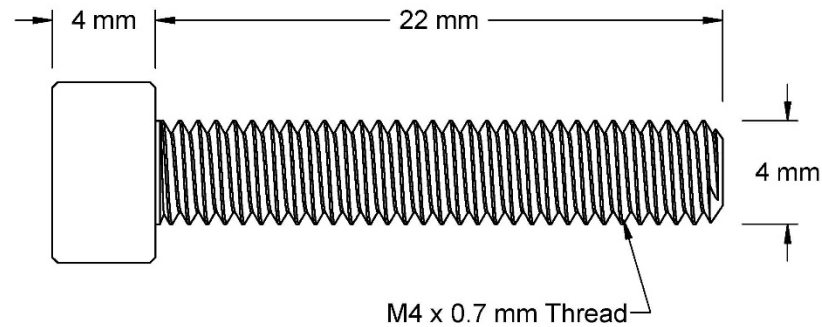
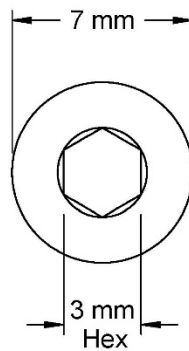
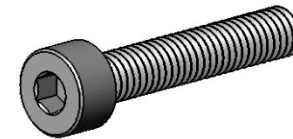
© 2014 McMaster-Carr Supply Company

Information in this drawing is provided for reference only.

PART
NUMBER

104

Metric Oversized
Washer



McMASTER-CARR CAD

<http://www.mcmaster.com>
© 2014 McMaster-Carr Supply Company

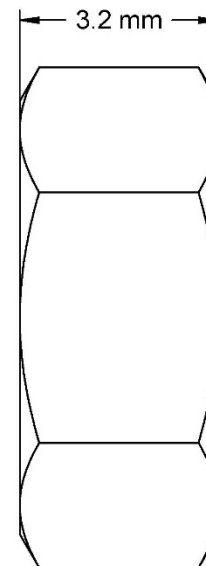
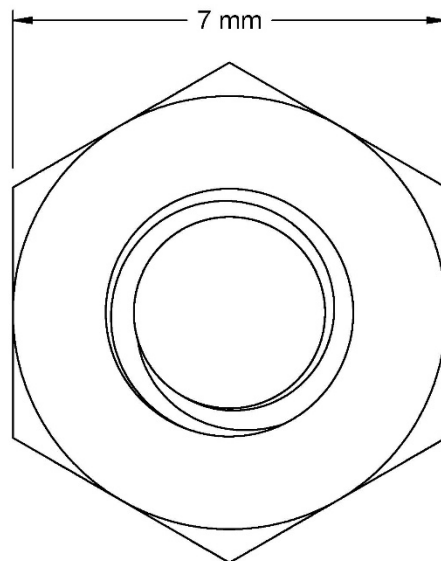
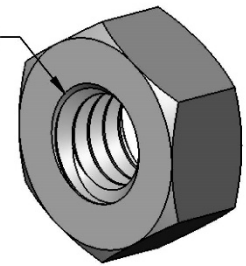
Information in this drawing is provided for reference only.

PART
NUMBER

105

Metric Alloy Steel
Socket Head Cap Screw

M4 x 0.7 mm Thread



McMASTER-CARR CAD

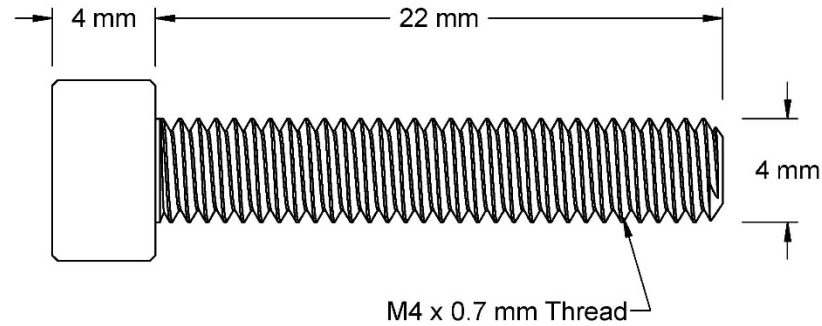
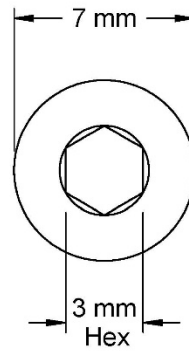
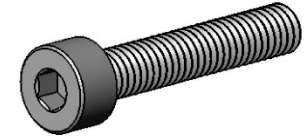
<http://www.mcmaster.com>
© 2015 McMaster-Carr Supply Company

Information in this drawing is provided for reference only.

PART
NUMBER

106

Metric
Hex Nut



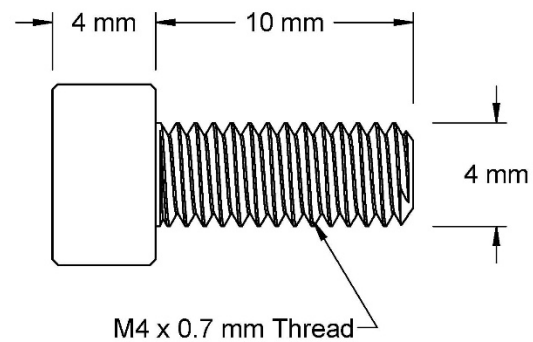
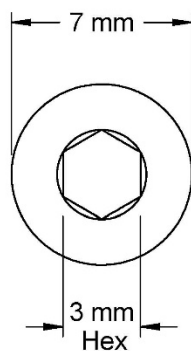
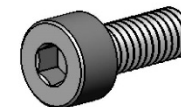
McMASTER-CARR CAD

<http://www.mcmaster.com>
© 2014 McMaster-Carr Supply Company

Information in this drawing is provided for reference only.

PART
NUMBER **107**

Metric Alloy Steel
Socket Head Cap Screw

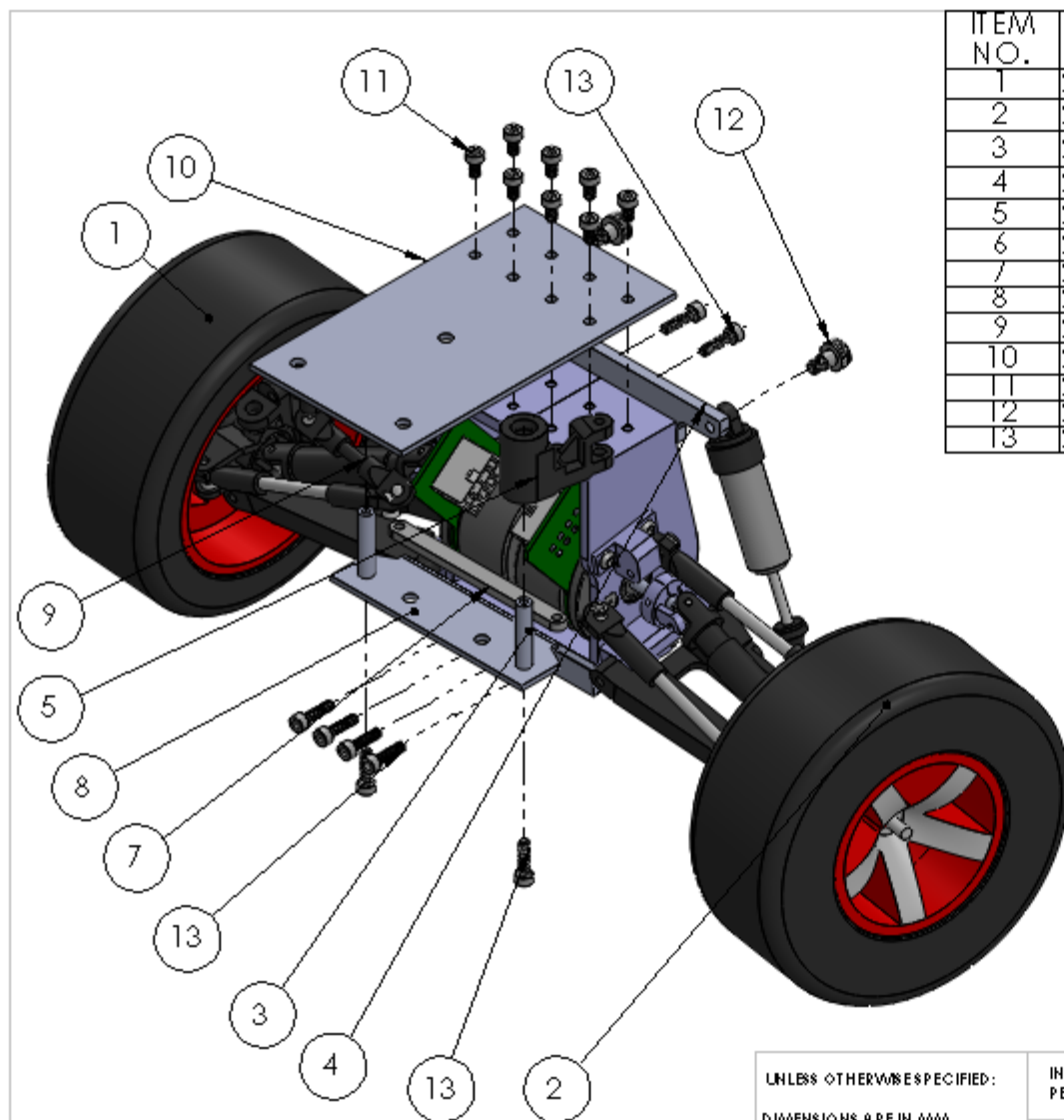


McMASTER-CARR CAD

<http://www.mcmaster.com>
© 2014 McMaster-Carr Supply Company
Information in this drawing is provided for reference only.

PART
NUMBER **108**

Metric Alloy Steel
Socket Head Cap Screw



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	230	FL MOTOR ASSEMBLY	1
2	220	FR MOTOR ASSEMBLY	1
3	205	STEERING POST DOWEL	2
4	202	F SUSP. MOUNT	1
5	207	RIGHT STEERING POST	1
6	206	LEFT STEERING POST	1
7	211	STEERING LINKAGE	1
8	203	FB CHASSIS MOUNT	1
9	212	S. STEER TURNBUCKLE	1
10	204	FT CHASSIS MOUNT	1
11	208	M3 X 0.5 6MM	8
12	210	M4 X 0.7 SHOULDER	2
13	209	M3 X 0.5 12 MM	8

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACED DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: VARIOUS

TITLE: F MOTOR BLOCK ASSEMBLY EXPLODED

DRAWN BY: CG

DWG #: 201

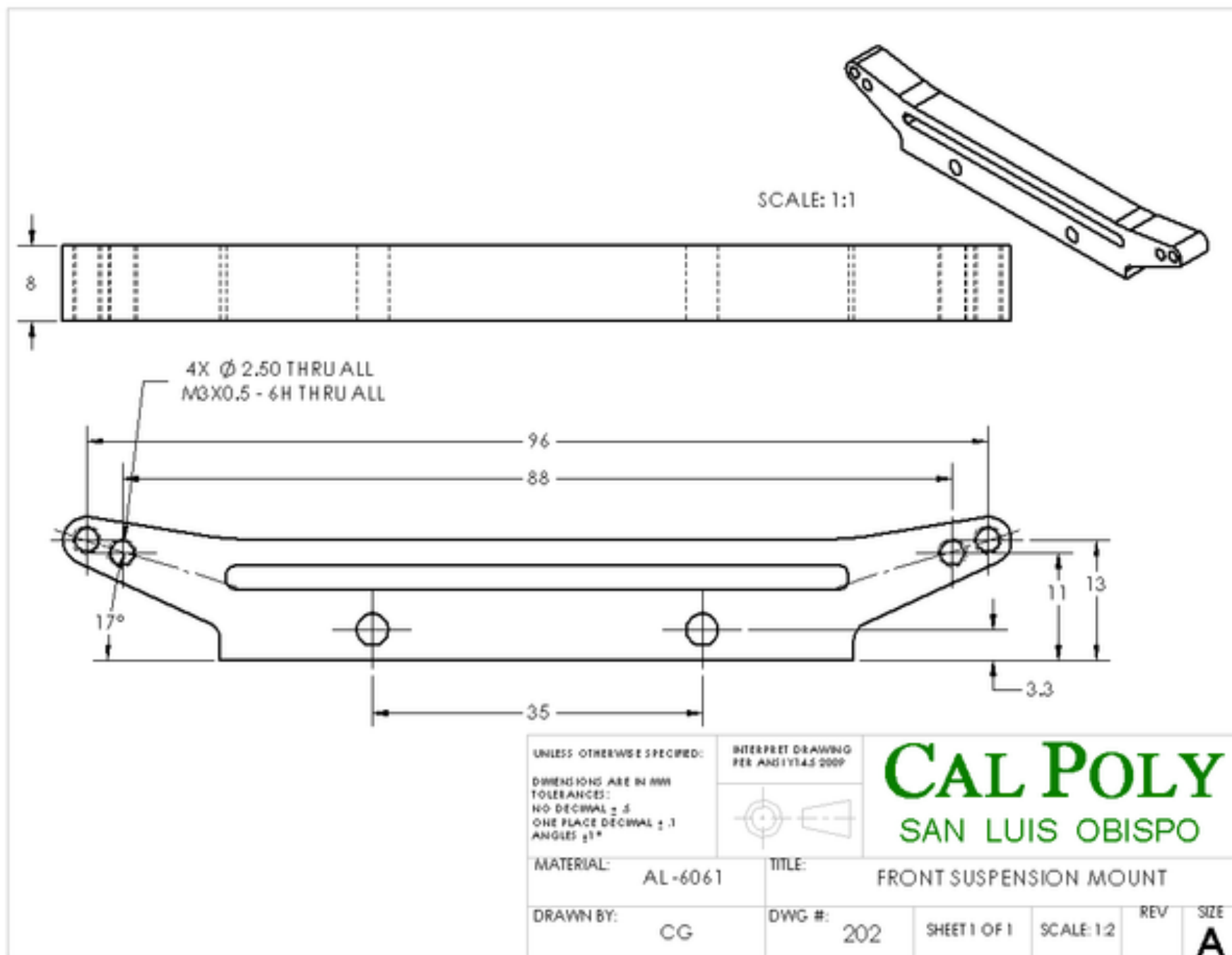
SHEET 1 OF 1

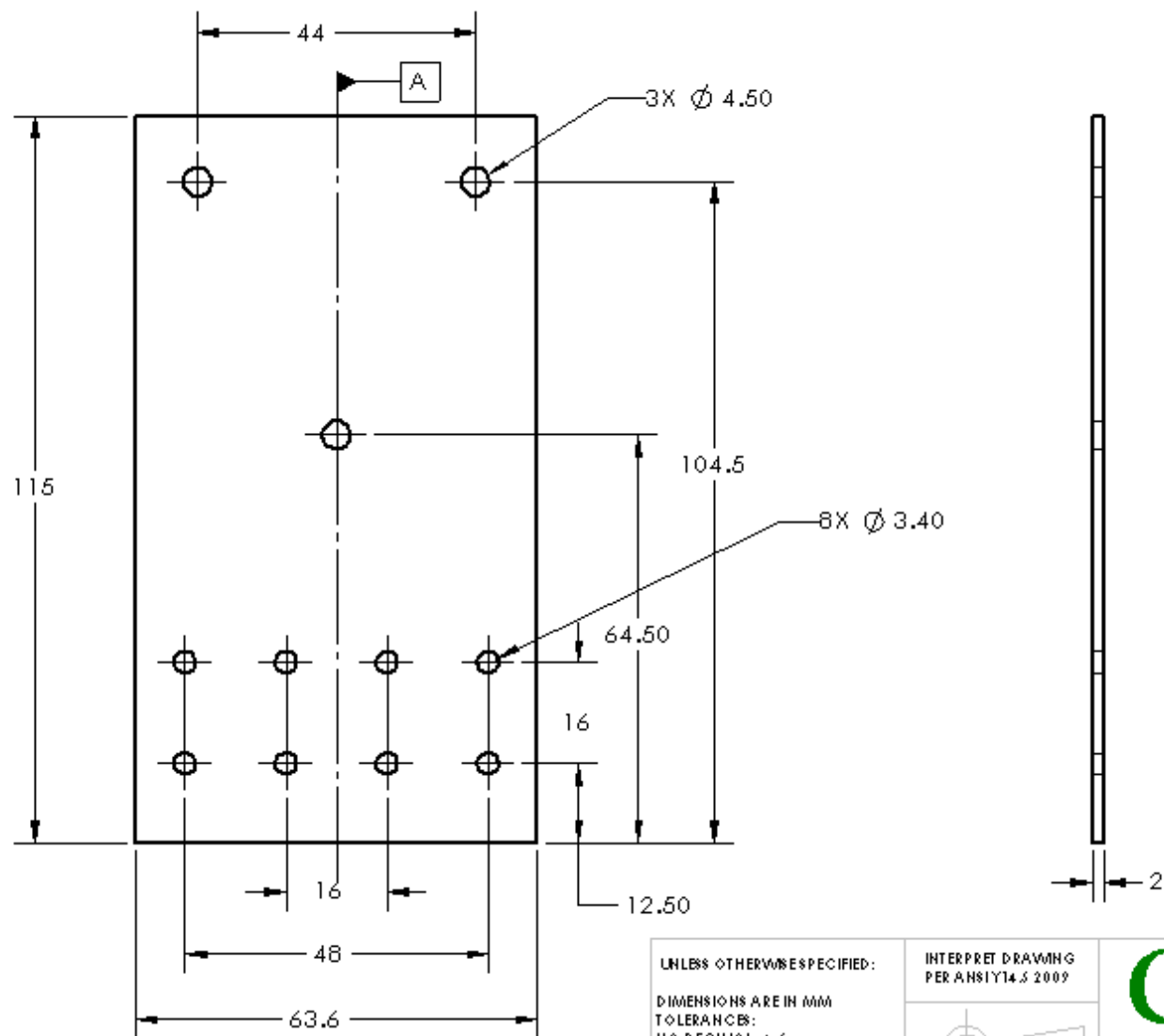
SCALE: 1:4

REV

SIZE

A





UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACED DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL -6061

TITLE: FRONT TOP CHASSIS MOUNT

DRAWN BY: CG

DWG #: 204

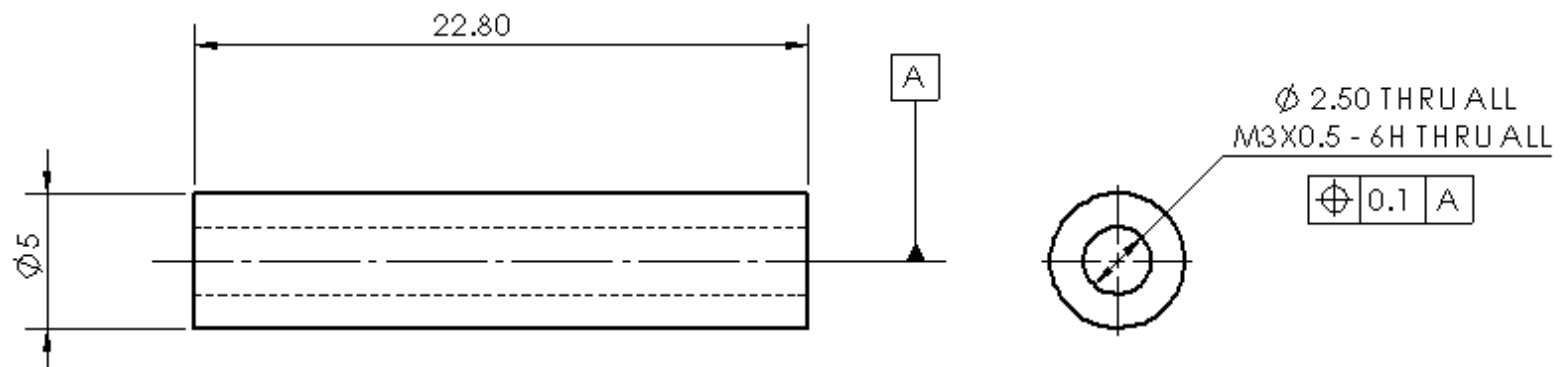
SHEET 1 OF 1

SCALE: 1:1

REV

SIZE

A



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: STEERING POST DOWEL

DRAWN BY: CG

DWG #: 205

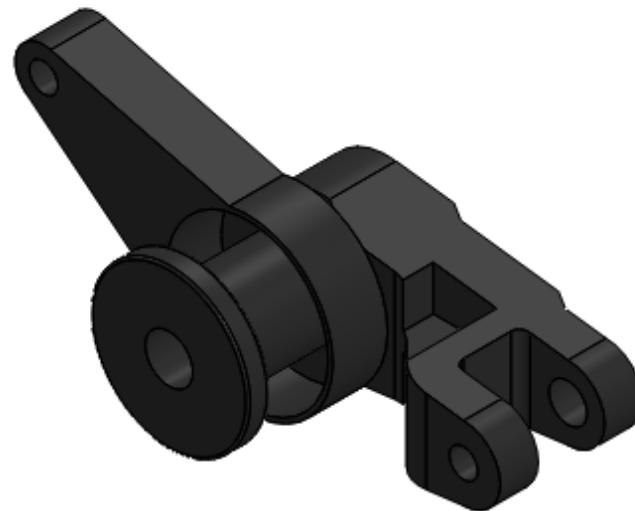
SHEET 1 OF 1

SCALE: 4:1

REV

SIZE

A



TRAXXAS SLASH
ASSEMBLY
PART
MODEL # 6845X

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM

TOLERANCES:

NO DECIMAL $\pm .5$

ONE PLACE DECIMAL $\pm .1$

ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: MISC.

TITLE: STEERING BELLCRANK LEFT

DRAWN BY: CG

DWG #: 206

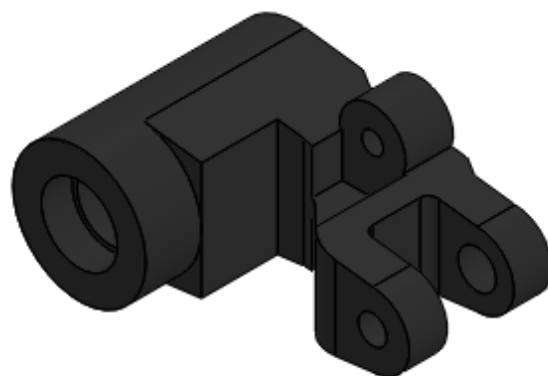
SHEET 1 OF 1

SCALE: 2:1


REV

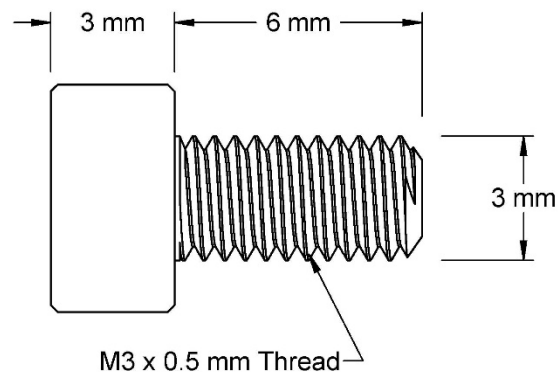
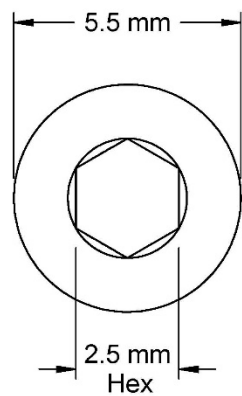
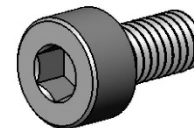
SIZE

A



TRAXXAS SLASH
ASSEMBLY PART
MODEL # 6845X

UNLESS OTHERWISE SPECIFIED:		INTERPRET DRAWING PER ANSI Y14.5 2009		<div>CAL POLY</div> <div>SAN LUIS OBISPO</div>			
DIMENSIONS ARE IN MM TOLERANCES: NO DECIMAL $\pm .5$ ONE PLACE DECIMAL $\pm .1$ ANGLES $\pm 1^\circ$							
MATERIAL: misc		TITLE: STEERING BELLCRANK RIGHT					
DRAWN BY: CG		DWG #: 207		SHEET 1 OF 1	SCALE: 2:1	REV	SIZE A

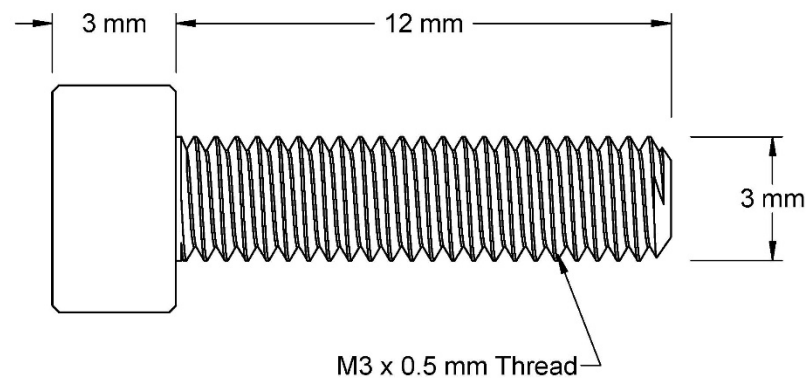
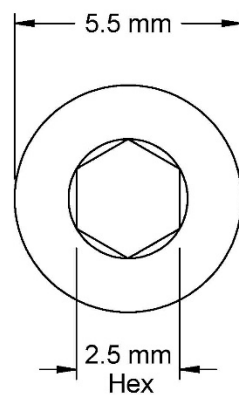
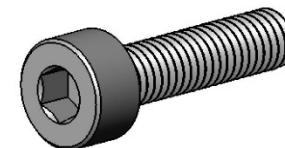


McMASTER-CARR CAD

<http://www.mcmaster.com>
© 2014 McMaster-Carr Supply Company
Information in this drawing is provided for reference only.

PART
NUMBER **208**

Metric Alloy Steel
Socket Head Cap Screw



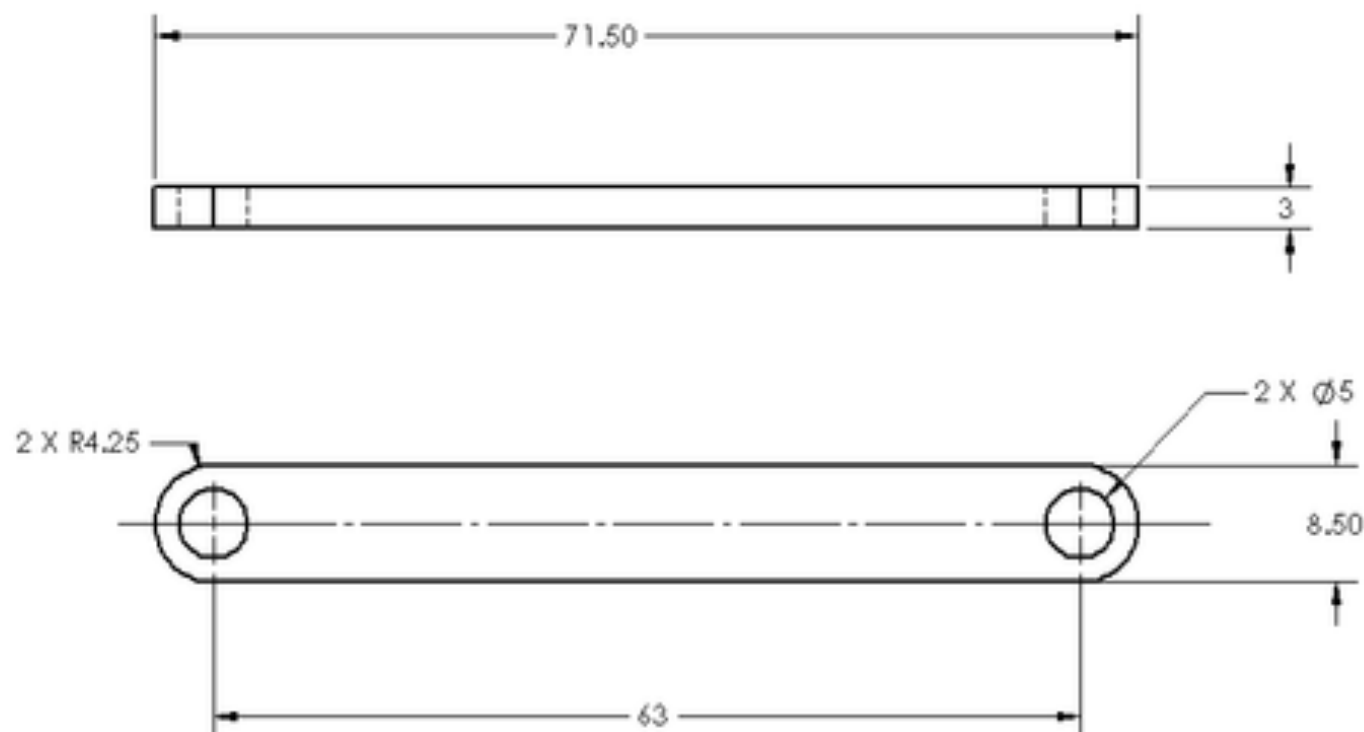
McMASTER-CARR CAD

<http://www.mcmaster.com>
© 2014 McMaster-Carr Supply Company

Information in this drawing is provided for reference only.

PART
NUMBER **209**

Metric Alloy Steel
Socket Head Cap Screw



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: PLA

TITLE: STEERING LINKAGE

DRAWN BY: CG

DWG #: 211

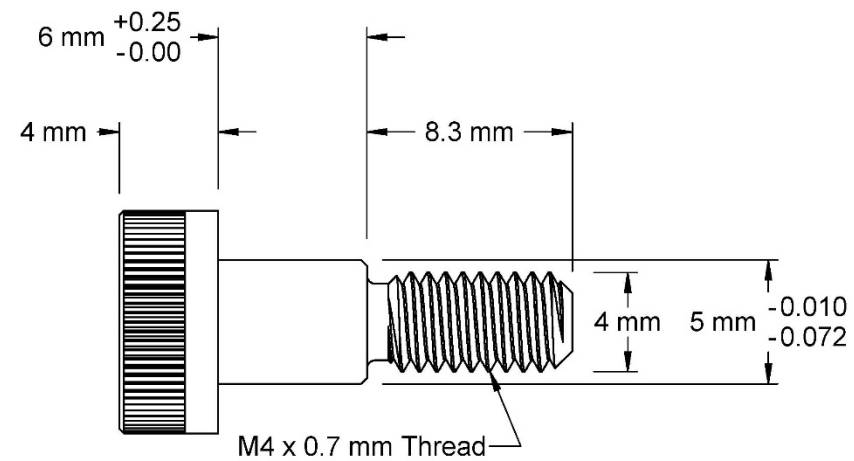
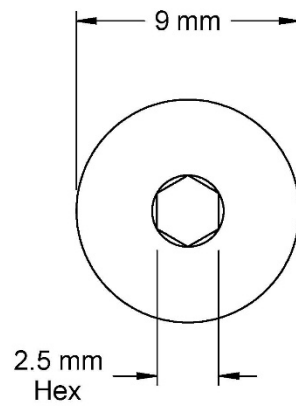
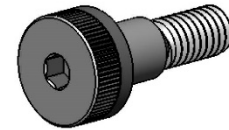
SHEET 1 OF 1

SCALE: 2:1

REV

SIZE

A



McMASTER-CARR CAD

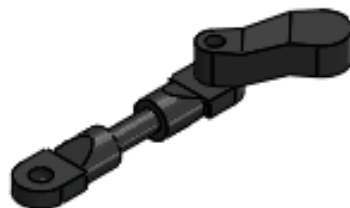
<http://www.mcmaster.com>
© 2013 McMaster-Carr Supply Company

Information in this drawing is provided for reference only.

PART
NUMBER **212**

Shoulder
Screw

TRAXXAS STEERING
TURNBUCKLE 4X4 SLASH



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: MISC

TITLE: SMALL STEERING TURNBUCKLE

DRAWN BY: CG

DWG #: 213

SHEET 1 OF 1

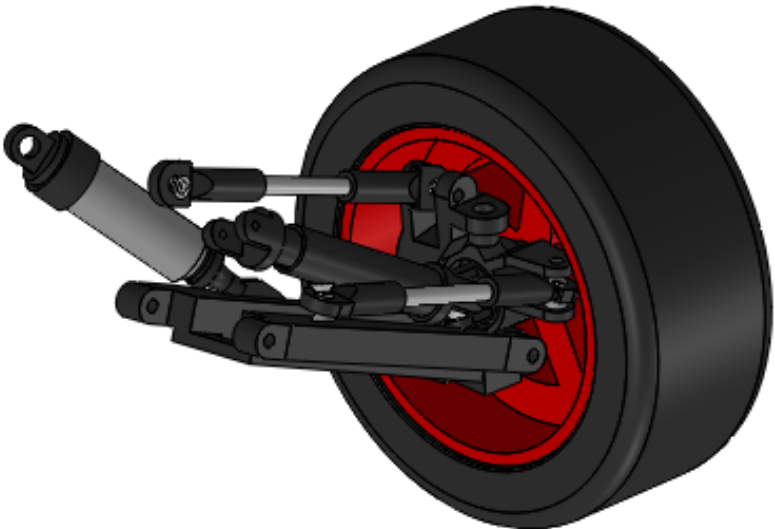
SCALE: 1:1

REV

SHEET

A

TRAXXAS SUSPENSION AND
DRIVETRAIN ASSEMBLY:



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACED DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: MISC

TITLE: TRAXXAS ASSEMBLY FRONT RIGHT

DRAWN BY: CG

DWG #: 221

SHEET 1 OF 1

SCALE: 2:3

REV

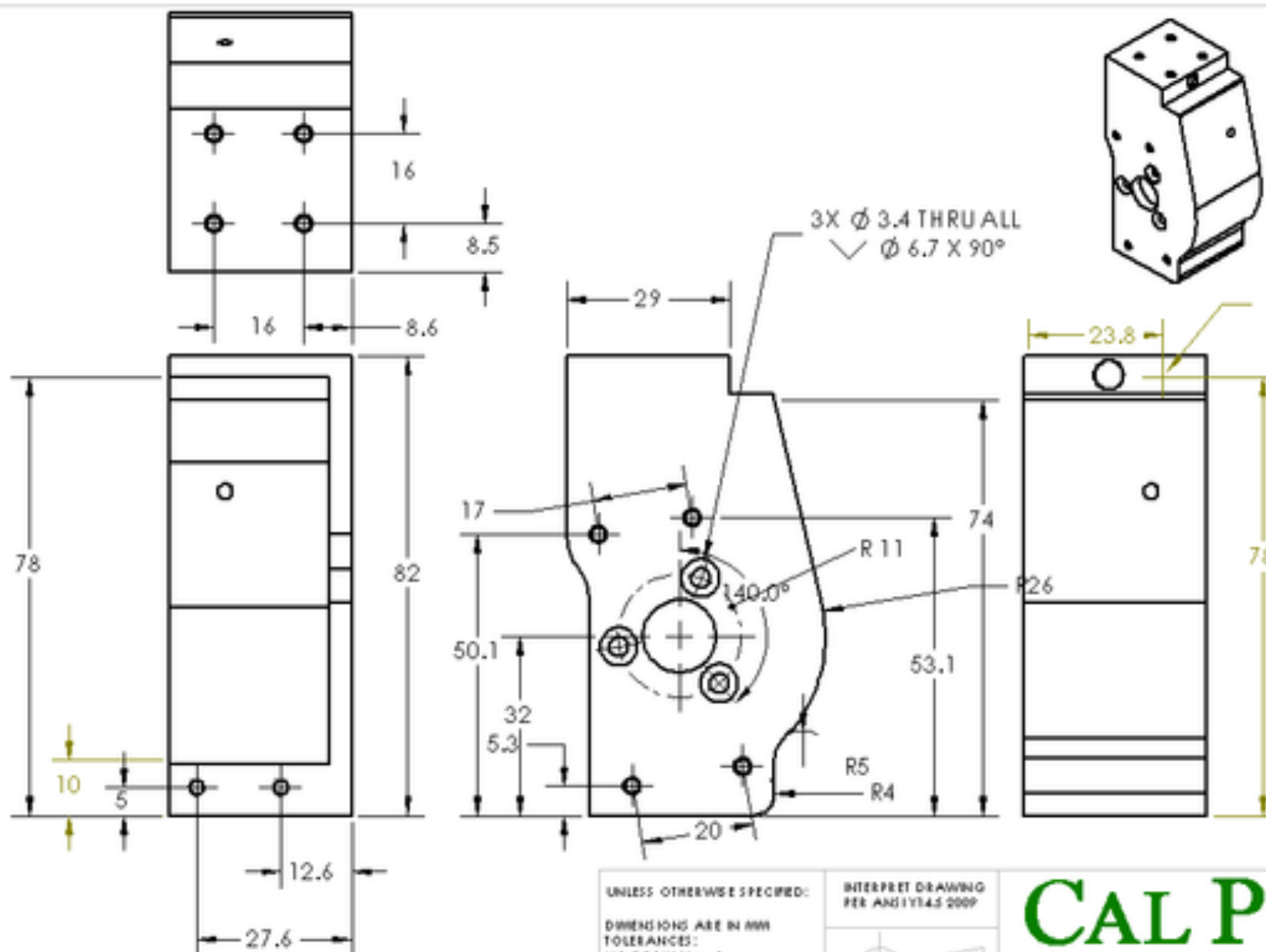
SIZE

A

SCALE 1:2



$\varnothing 2.5 \pm 0.1$
M3X0.5 - 6H



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y34.5 2009



MATERIAL: AL-6061

TITLE: FR MOTOR HOUSING

DRAWN BY: CG

DWG #: 222A

SHEET 1 OF 1

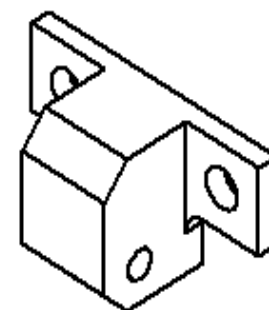
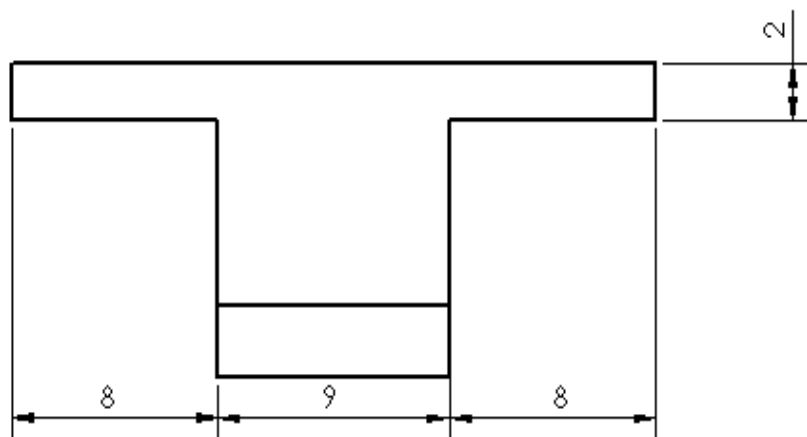
SCALE: 1

REV

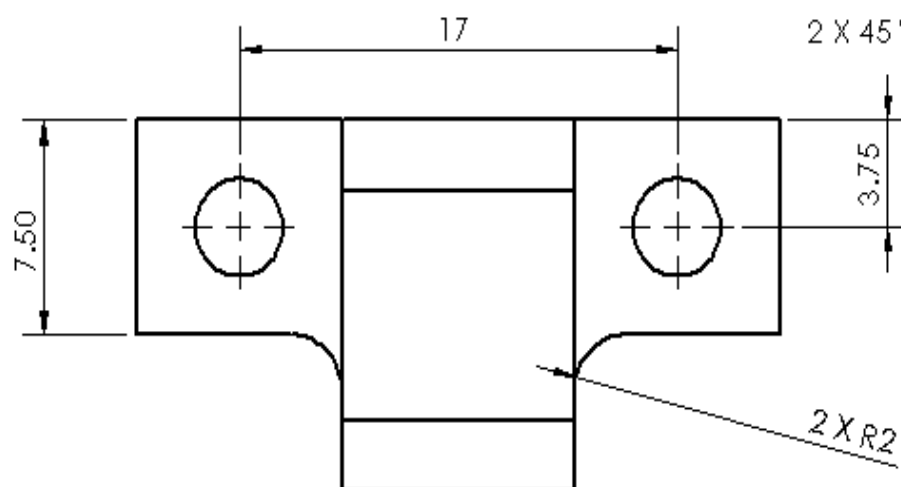
SIZE

A

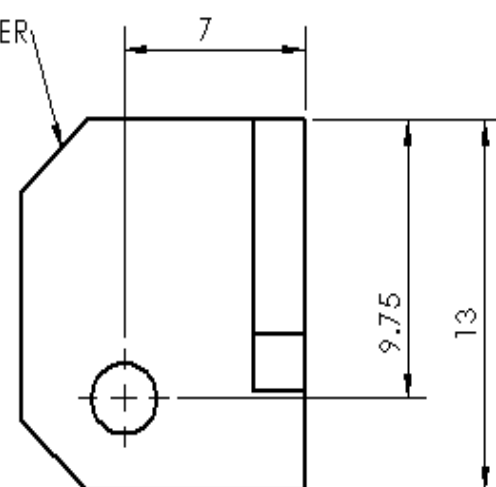
CAL POLY
SAN LUIS OBISPO



SCALE: 2:1



2 X 45° CHAMFER



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: FRONT TURNBUCKLE MOUNT

DRAWN BY: CG

DWG #: 223

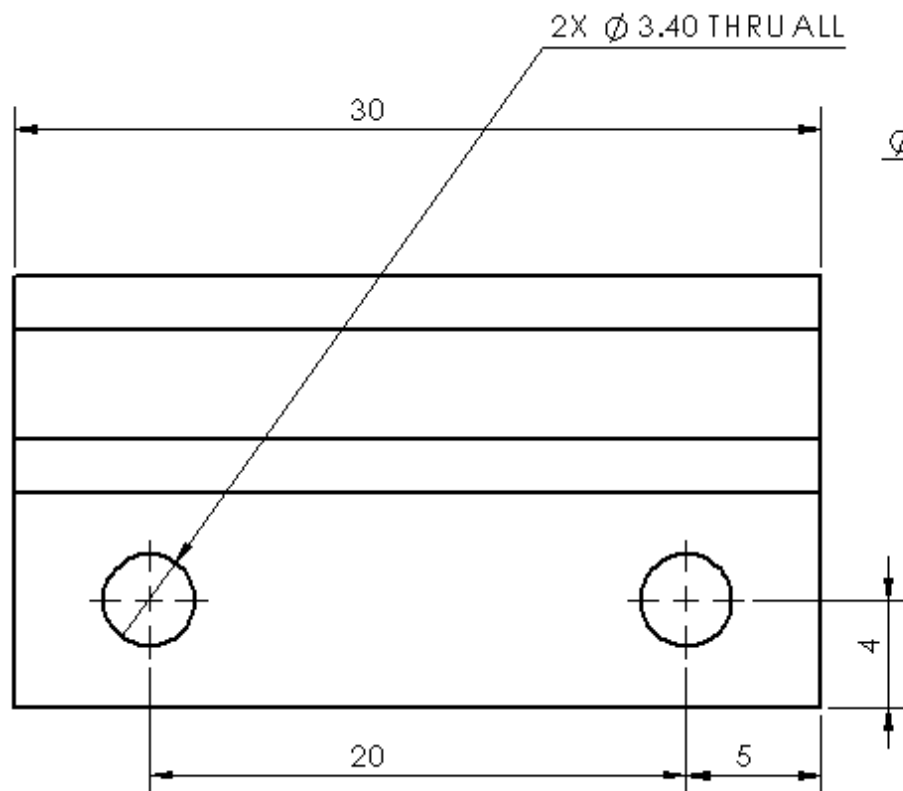
SHEET 1 OF 1

SCALE: 4:1

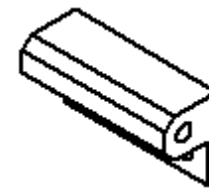
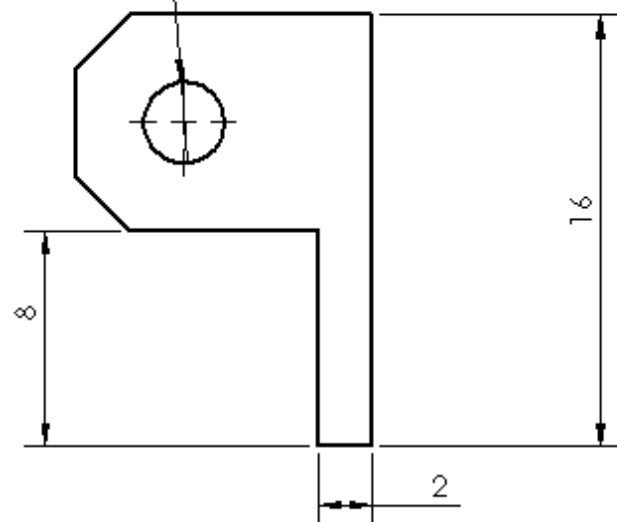
REV

SIZE

A



Ø3 max. THRU



SCALE: 1:1

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM

TOLERANCES:

NO DECIMAL $\pm .5$

ONE PLACE DECIMAL $\pm .1$

ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: FRONT A-ARM MOUNT

DRAWN BY: CG

DWG #: 224

SHEET 1 OF 1

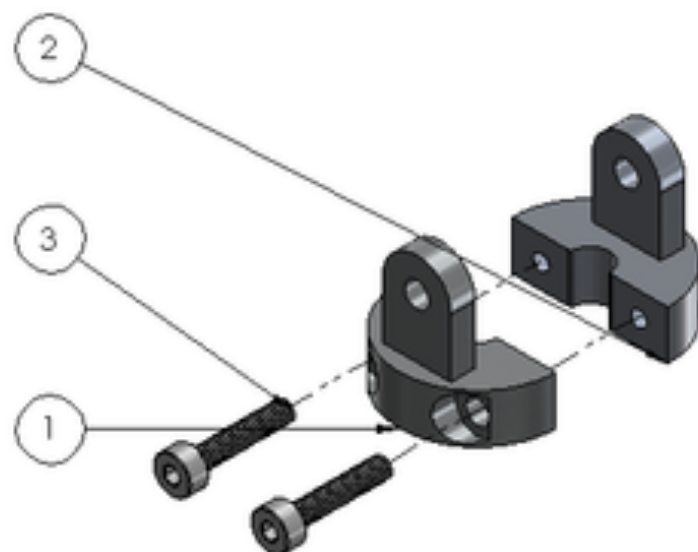
SCALE: 4:1

REV

SIZE

A

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	225A	SHAFT COUPLER SIDE A	1
2	225B	SHAFT COUPLER SIDE B	1
3	231	M2.5 X 0.4 12MM SOCKET	2



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: VARIOUS

TITLE: SHAFT COUPLER ASSEMBLY

DRAWN BY: CG

DWG #: 225

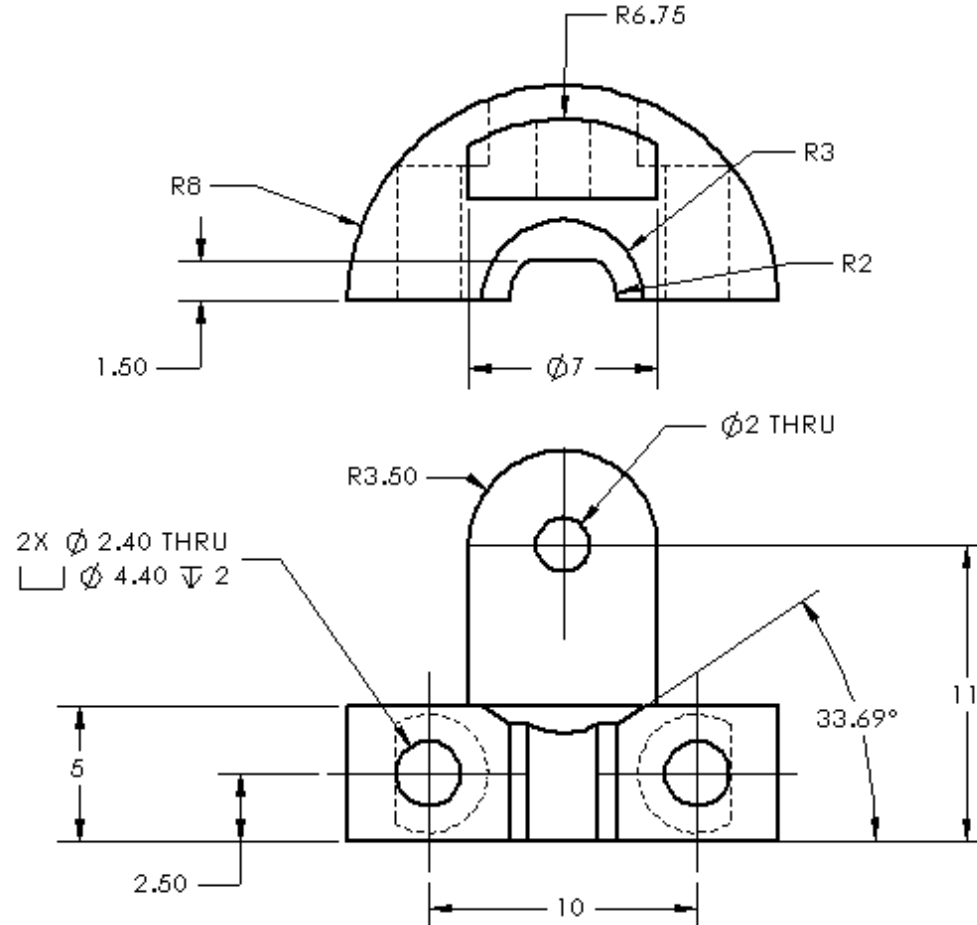
SHEET 1 OF 1

SCALE: 2:1

REV

SIZE

A



SCALE: 1:1

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACED DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: SHAFT COUPLER SIDE A

DRAWN BY: CG

DWG #: 225A

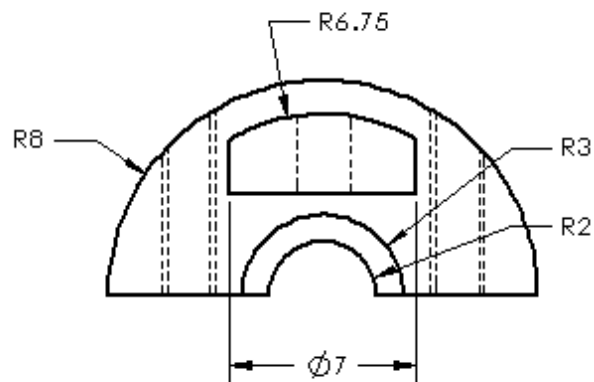
SHEET 1 OF 1

SCALE: 4:1

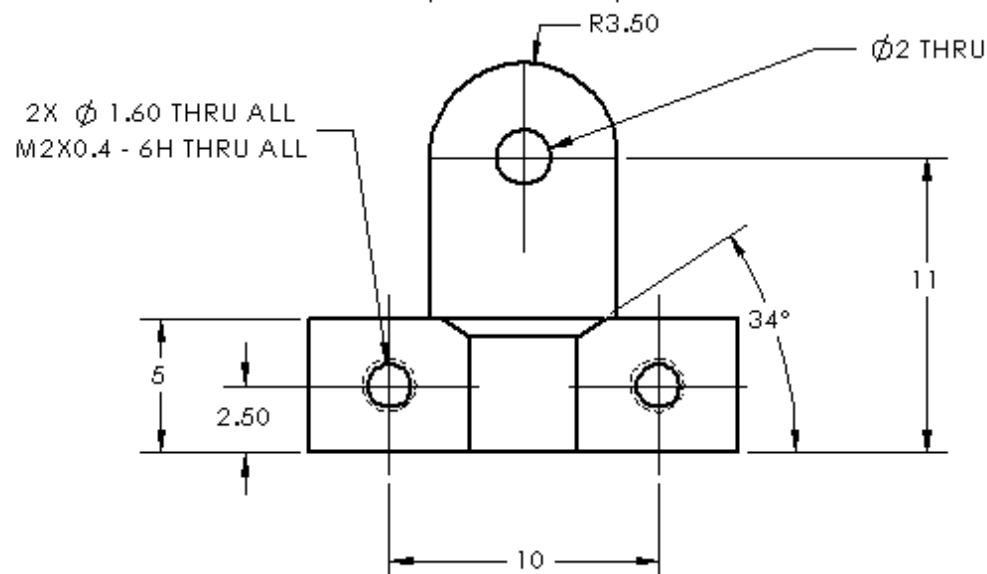
REV

SIZE

A



SCALE: 1:1



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL - 6061

TITLE: SHAFT COUPLER SIDE B

DRAWN BY: CG

DWG #: 225B

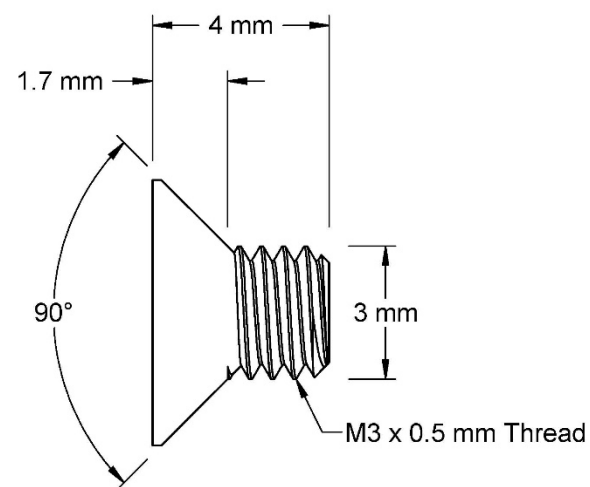
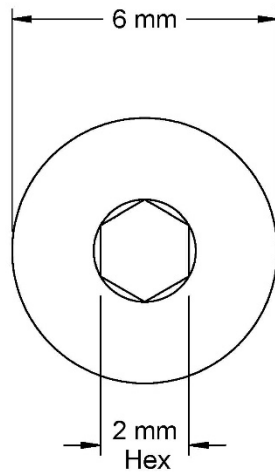
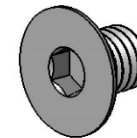
SHEET 1 OF 1

SCALE: 4:1

REV

SIZE

A

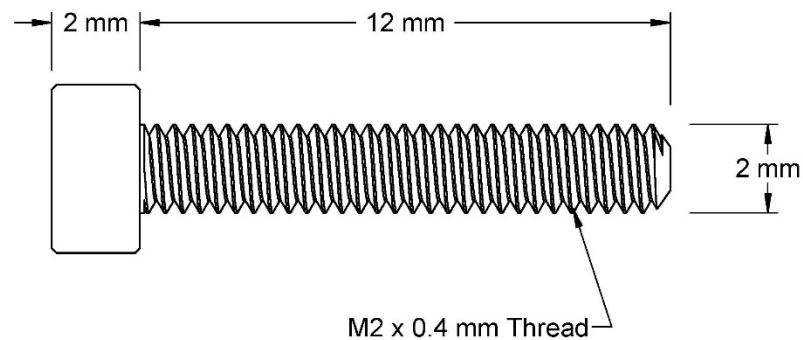
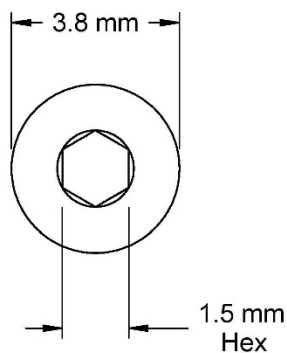
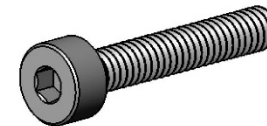


McMASTER-CARR CAD

<http://www.mcmaster.com>
© 2014 McMaster-Carr Supply Company
Information in this drawing is provided for reference only.

PART
NUMBER **227**

Stainless Steel Flat-Head
Socket Cap Screw



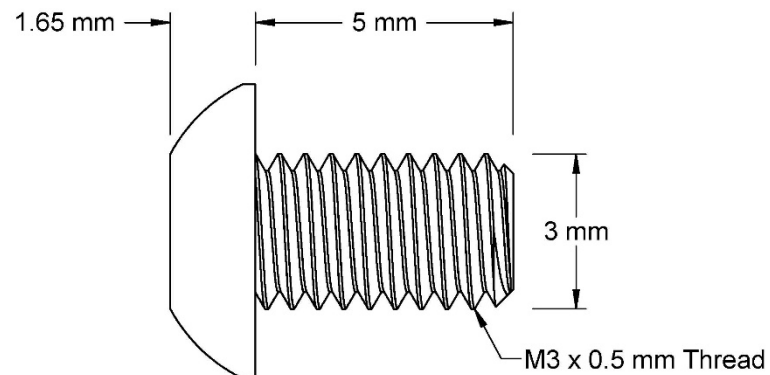
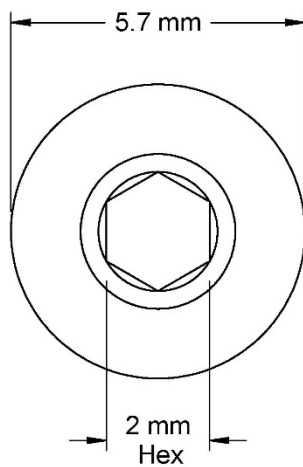
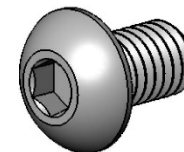
McMASTER-CARR CAD

<http://www.mcmaster.com>
© 2014 McMaster-Carr Supply Company
Information in this drawing is provided for reference only.

PART
NUMBER

228

Metric Alloy Steel
Socket Head Cap Screw

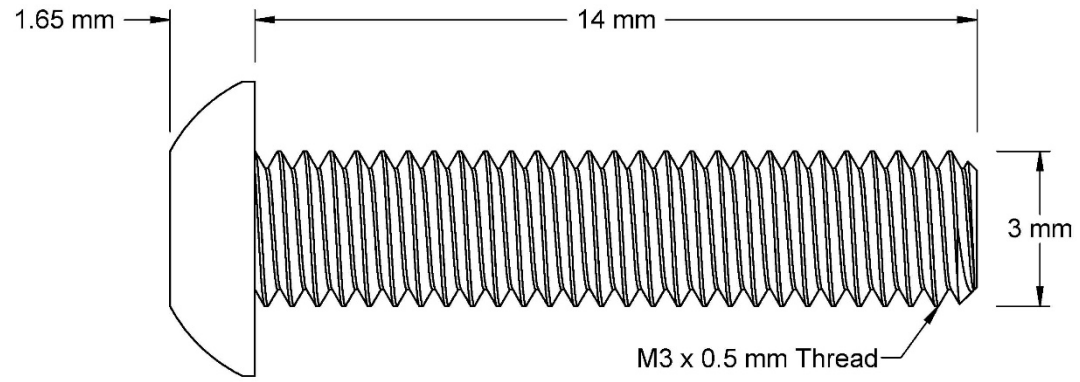
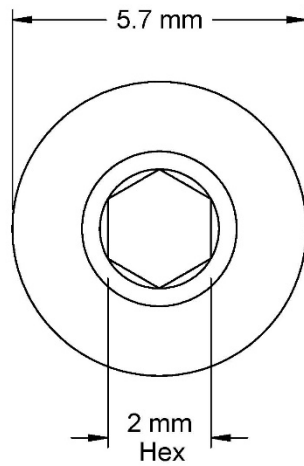
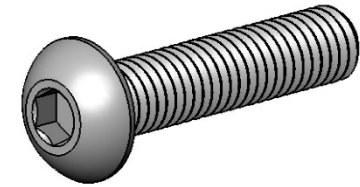


McMASTER-CARR CAD

<http://www.mcmaster.com>
© 2014 McMaster-Carr Supply Company
Information in this drawing is provided for reference only.

PART
NUMBER **229**

Alloy Steel Button-Head
Socket Cap Screw



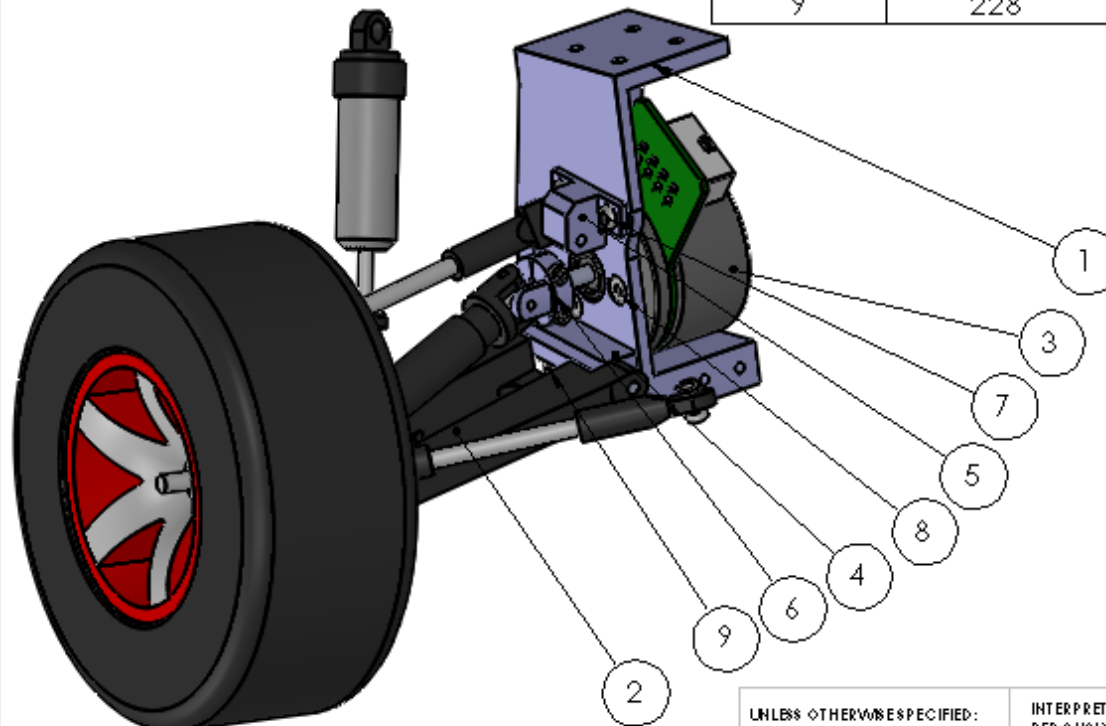
McMASTER-CARR CAD

<http://www.mcmaster.com>
© 2014 McMaster-Carr Supply Company
Information in this drawing is provided for reference only.

PART
NUMBER **230**

Alloy Steel Button-Head
Socket Cap Screw

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	242	FL MOTOR HOUSING	1
2	241	FLTRAXXAS ASSEM	2
3	226	MAXON MOTOR	1
4	224	A - ARM MOUNT FRONT	1
5	223	TURNBUCKLE MOUNT FRONT	1
6	225	SHAFT COUPLER	1
7	229	M3 X 0.5 5MM ROUND	2
8	227	M3 X 0.5 4MM FLAT	3
9	228	M3 X 0.5 12MM SOCKET	2



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM

TOLERANCES:

NO DECIMAL $\pm .5$

ONE PLACED DECIMAL $\pm .1$

ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: FRONT SUSPENSION MOUNT

DRAWN BY: CG

DWG #: 240

SHEET 1 OF 1

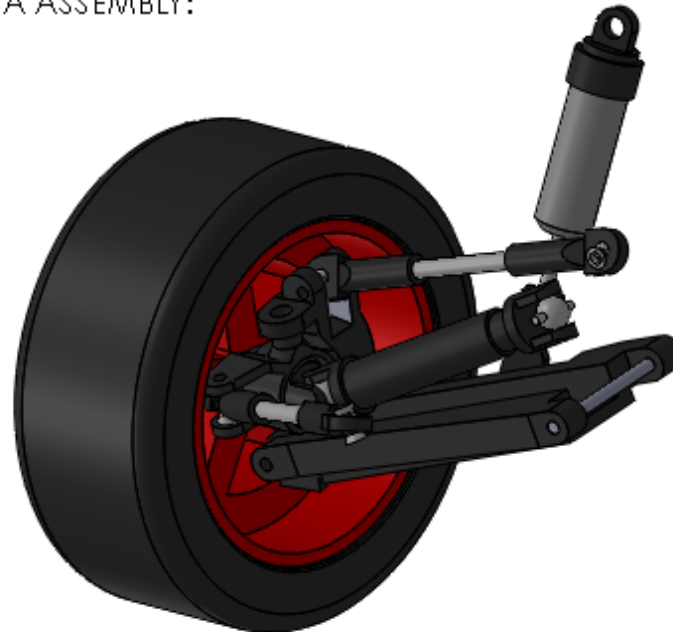
SCALE: 1:4

REV

SIZE

A

TRAXXAS DRIVETRAIN AND
SUSPENSION ASSEMBLY:



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM

TOLERANCES:

NO DECIMAL $\pm .5$

ONE PLACE DECIMAL $\pm .1$

ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: MISC

TITLE: FRONT LEFT TRAXXAS ASSEMBLY

DRAWN BY: CG

DWG #: 241

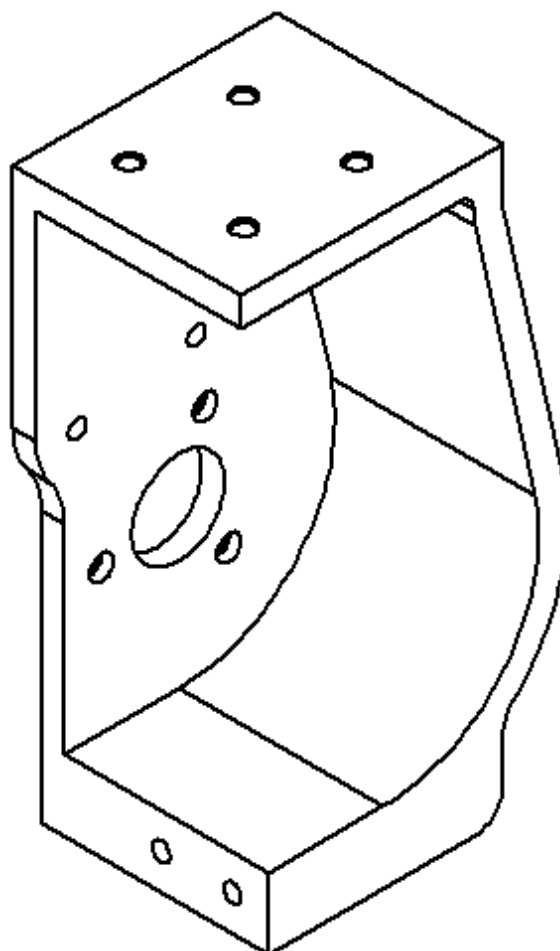
SHEET 1 OF 1

SCALE: 2:3

REV

SIZE

A



SEE PART 222A:
MIRRORED

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: FL MOTOR HOUSING

DRAWN BY: CG

DWG #: 242

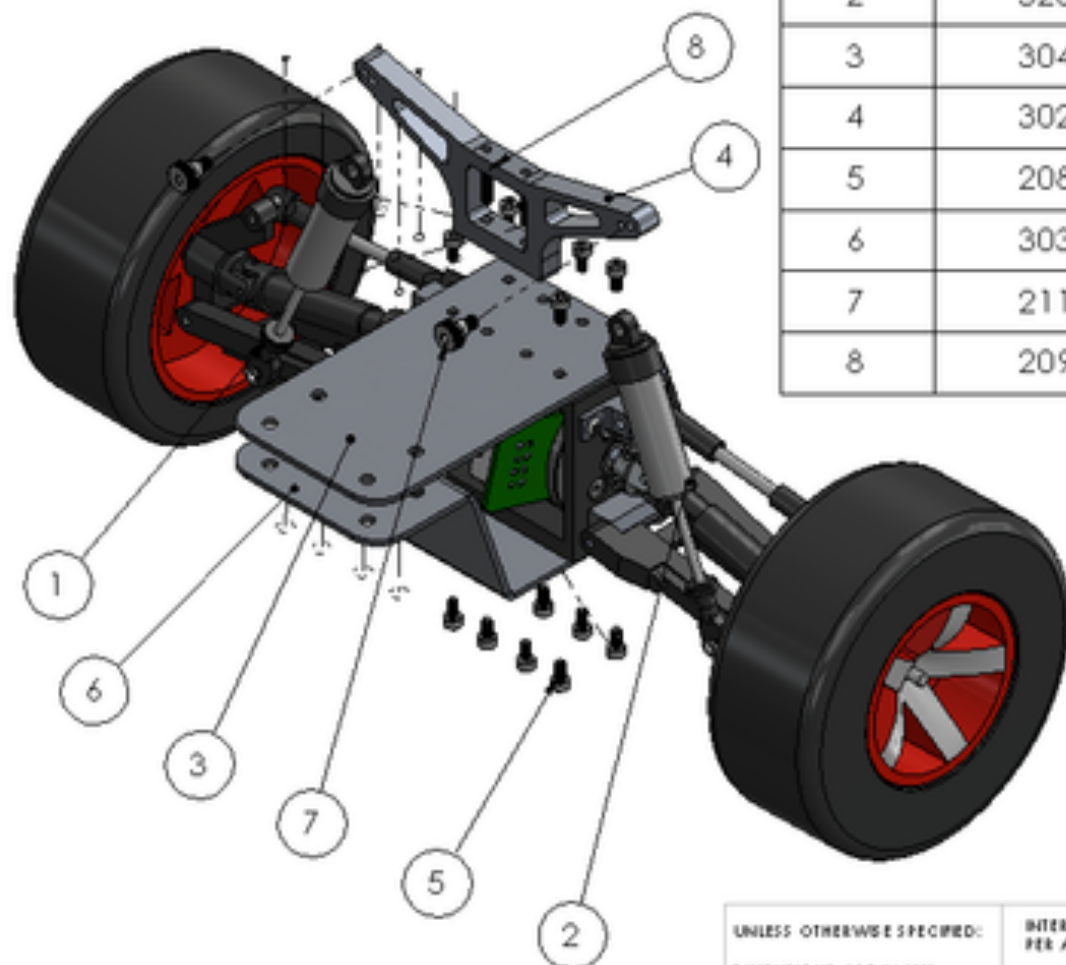
SHEET 1 OF 1

SCALE: 2:3

REV

SIZE

A



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	310	TRAXXAS ASSEM RR	1
2	320	TRAXXAS ASSEM RI	1
3	304	REAR TOP CHASSIS MOUNT	1
4	302	SUSPENSION MOUNT REAR	1
5	208	M3 X 0.5 6MM SOCKET	14
6	303	REAR BOTTOM CHASSIS MOUNT	1
7	211	M4 X 0.7 6MM SHOULDER	2
8	209	M3 X 0.5 12MM SOCKET	2

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: VARIOUS

TITLE: REAR MOTOR BLOCK ASSEM. EXPLODED

DRAWN BY: CG

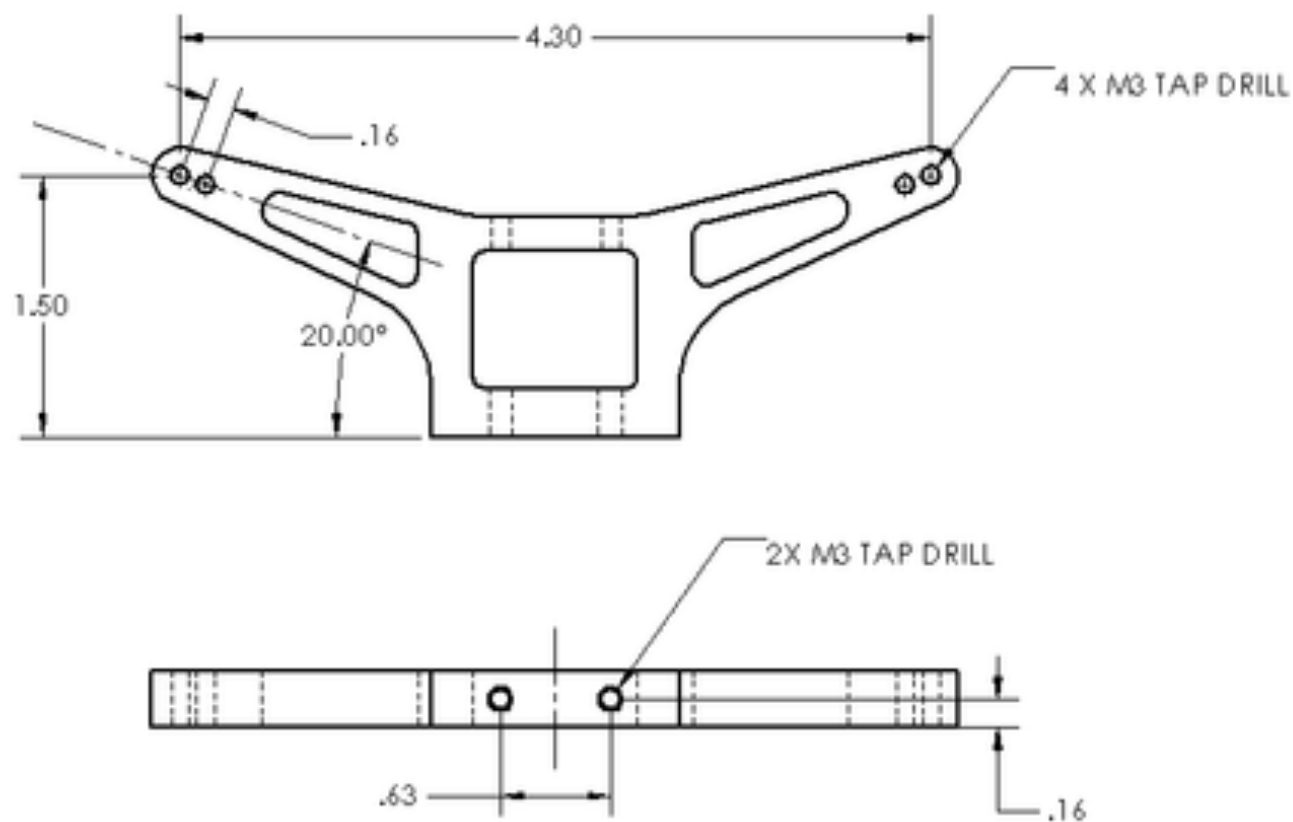
DWG #: 301

SHEET 1 OF 1

SCALE: 2

REV

SIZE
A



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM

TOLERANCES:

NO DECIMAL $\pm .5$

ONE PLACE DECIMAL $\pm .1$

ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: REAR HOLE DETAIL

DRAWN BY: CG

DWG #: 302A

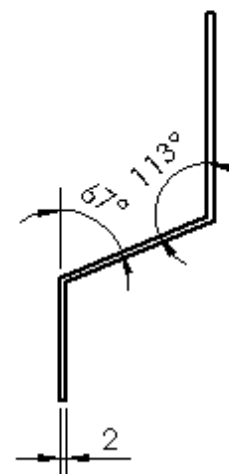
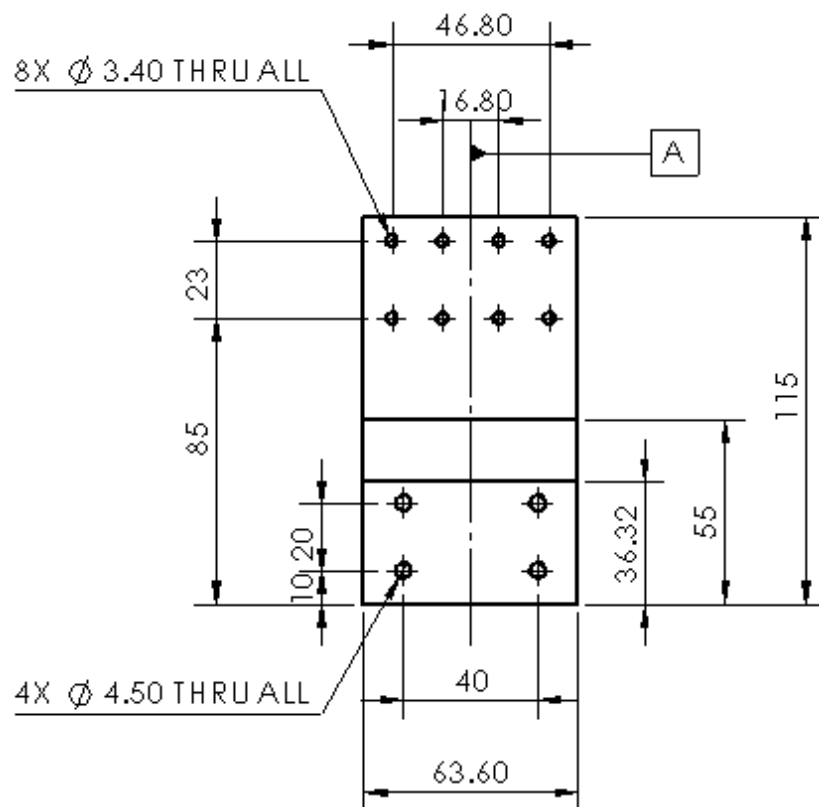
SHEET 1 OF 1

SCALE: 1:1

REV

SIZE

A



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: REAR BOTTOM CHASSIS MOUNT

DRAWN BY: CG

DWG #: 303

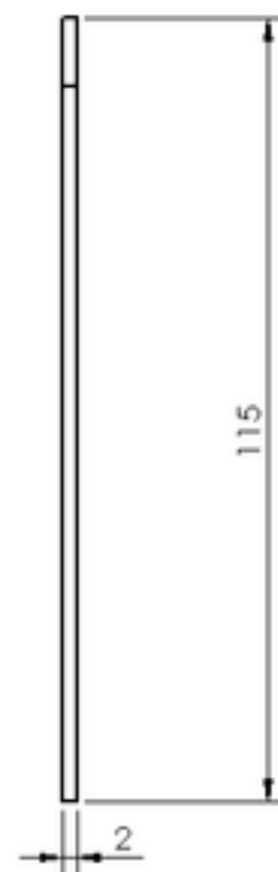
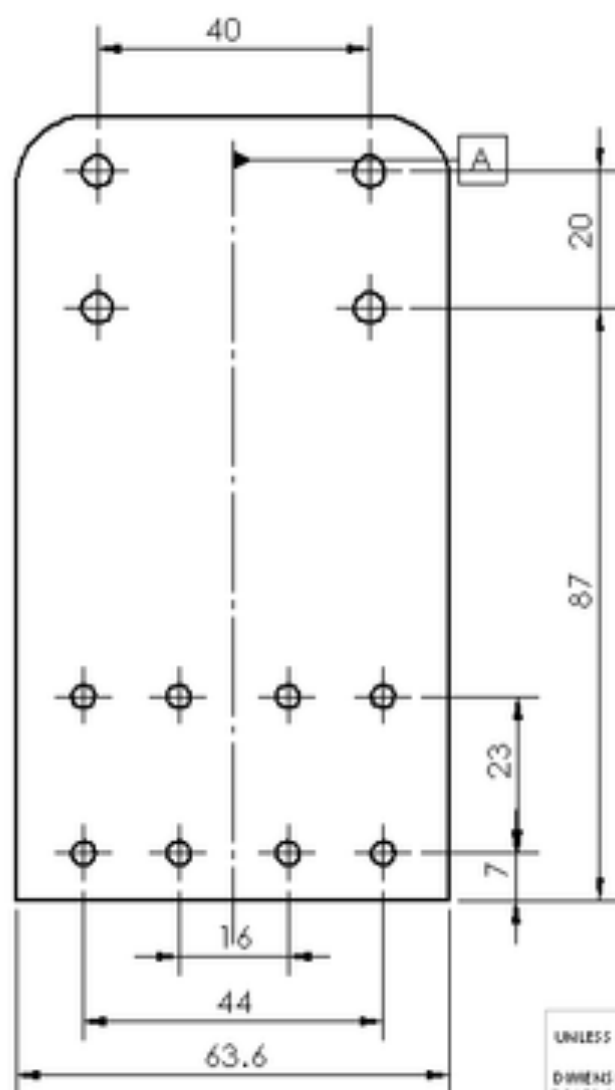
SHEET 1 OF 1

SCALE: 1:2

REV

SIZE

A



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: REAR TOP CHASSIS MOUNT

DRAWN BY: CG

DWG #: 304

SHEET 1 OF 1

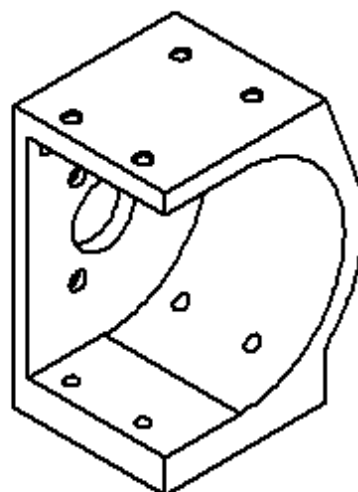
SCALE: 1:1

REV

SIZE

A

SEE DRAWING 321 - MIRRORED PART



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: REAR RIGHT MOTOR HOUSING

DRAWN BY: CG

DWG #: 311

SHEET 1 OF 1

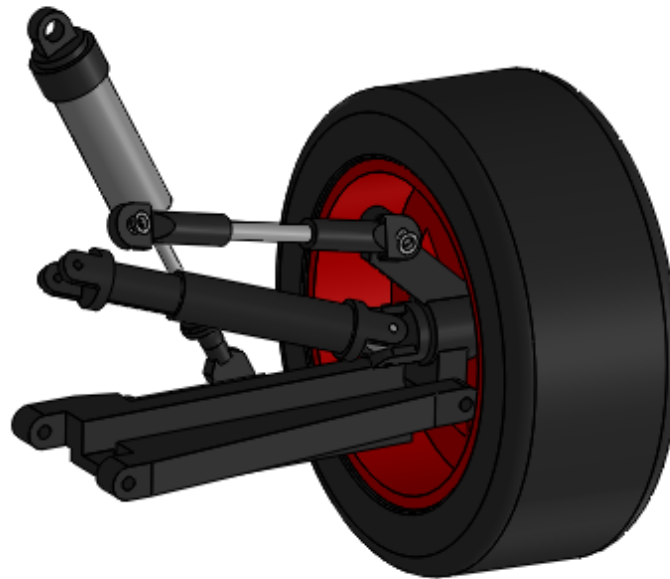
SCALE: 1:1

REV

SIZE

A

TRAXXAS DRIVETRAIN
AND SUSPENSION
ASSEMBLY



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACED DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: MISC

TITLE: REAR RIGHT TRAXXAS ASSEMBLY

DRAWN BY: CG

DWG #: 312

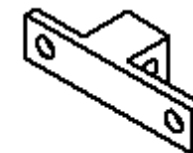
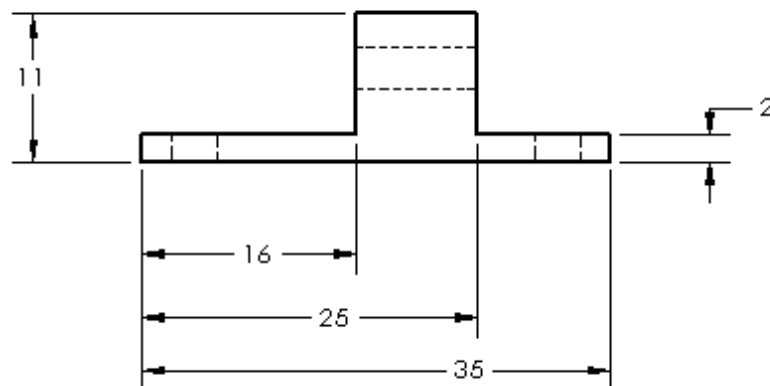
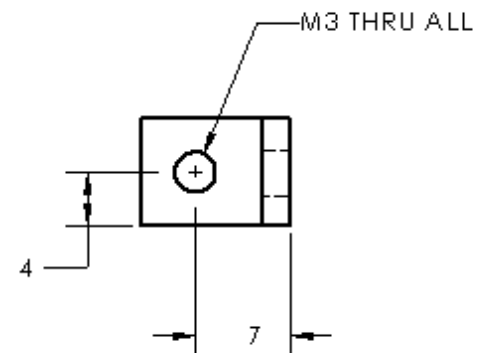
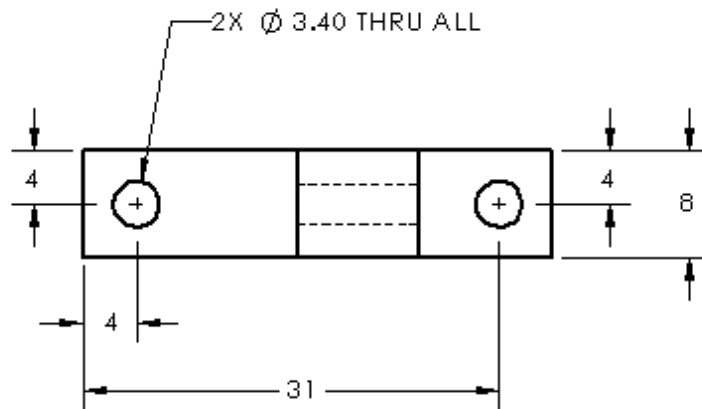
SHEET 1 OF 1

SCALE: 2:3

REV

SIZE

A



SCALE: 1:1

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM

TOLERANCES:

NO DECIMAL $\pm .5$

ONE PLACE DECIMAL $\pm .1$

ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL-6061

TITLE: REAR TURNBUCKLE MOUNT

DRAWN BY: CG

DWG #: 314

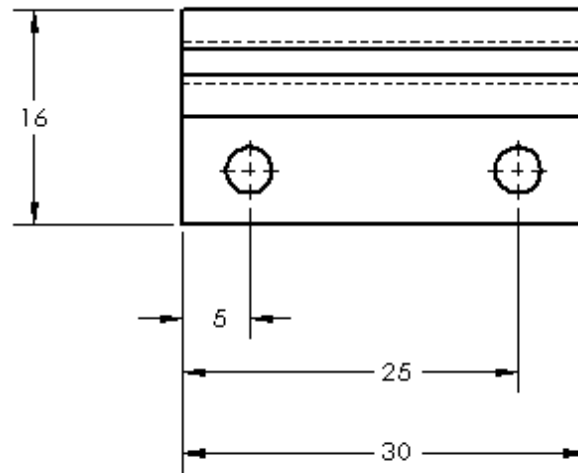
SHEET 1 OF 1

SCALE: 2:1

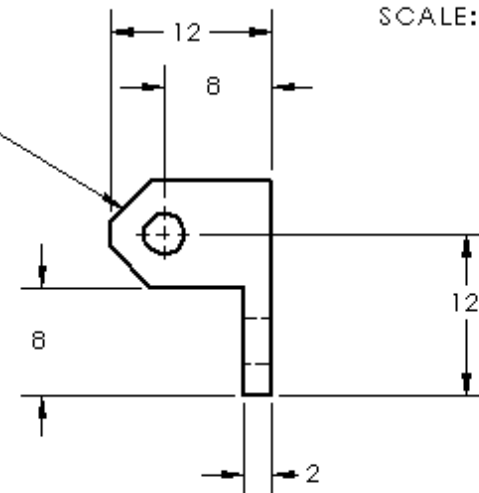
REV

SIZE

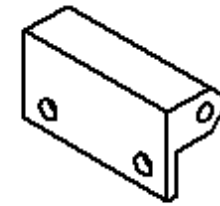
A



2 X 45°
CHAMFER



SCALE: 1:1



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL -6061

TITLE: A-ARM MOUNT REAR

DRAWN BY: CG

DWG #: 315

SHEET 1 OF 1

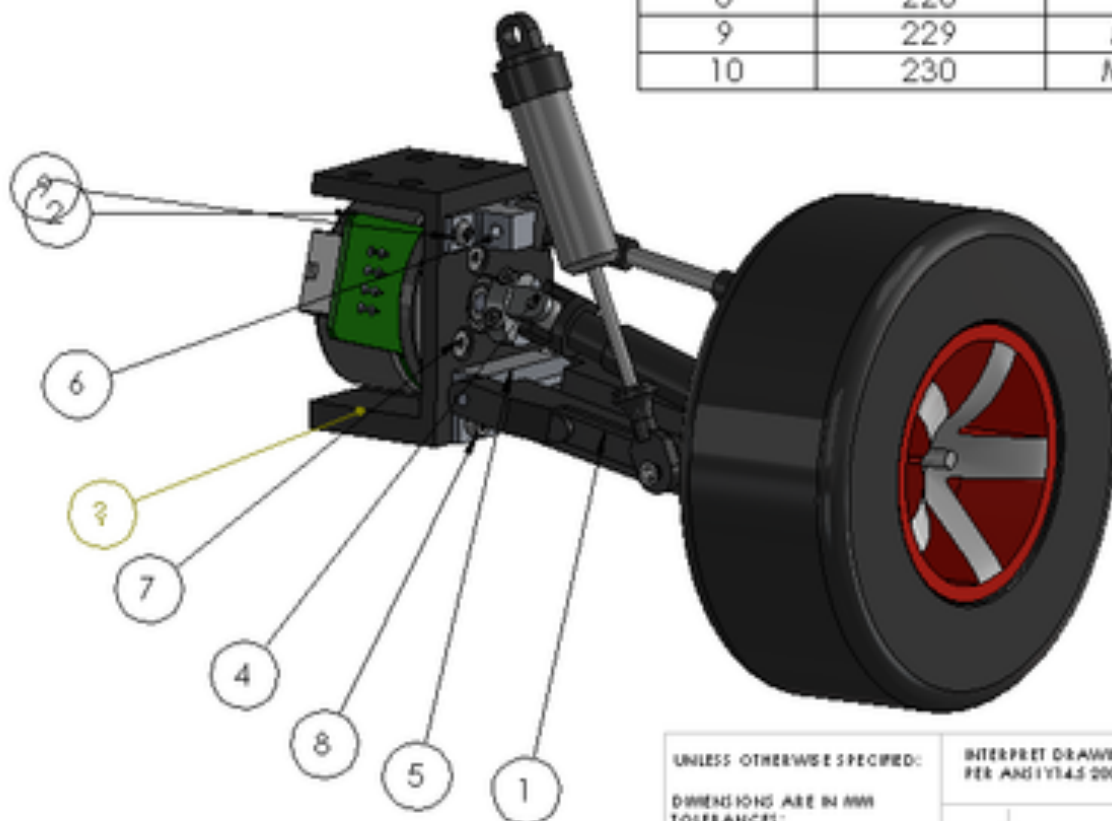
SCALE: 2:1

REV

SIZE

A

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	322	REAR LEFT TRAXXAS ASSEM	1
2	226	MAXON MOTOR	1
3	323	REAR LEFT MOTOR HOUSING	1
4	225	SHAFT COUPLER	1
5	314	REAR A-ARM MOUNT	1
6	313	REAR TURNBUCKLE MOUNT	1
7	227	M3 X0.5 4MM FLAT	3
8	228	M3 X 0.5 12MM SOCKET	2
9	229	M3 X 0.5 5MM ROUNDED	2
10	230	M3 X 0.5 14MM ROUNDED	1



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: VARIOUS

TITLE: REAR LEFT MOTOR BLOCK ASSEMBLY

DRAWN BY: CG

DWG #: 320

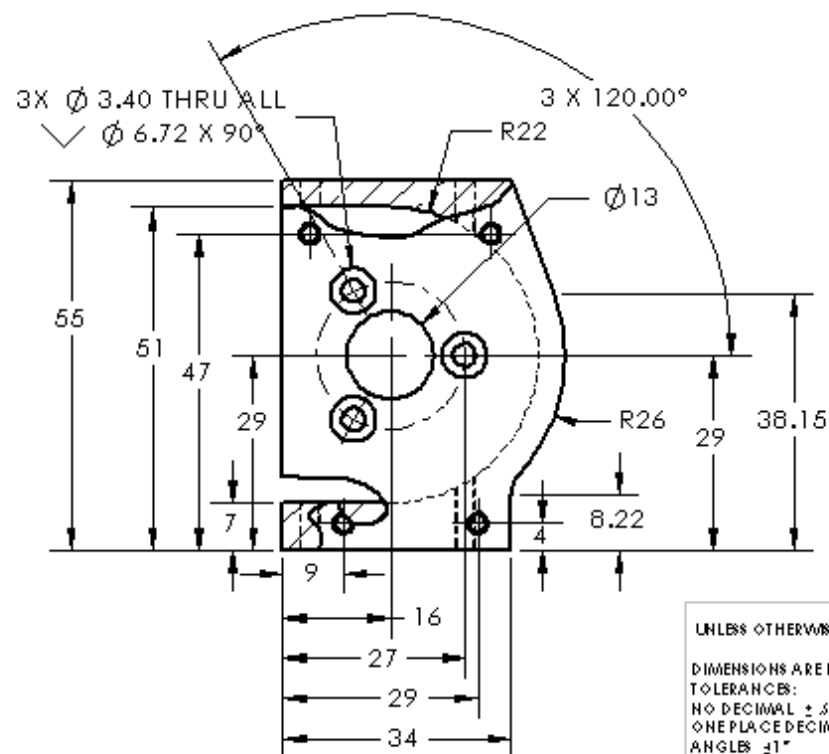
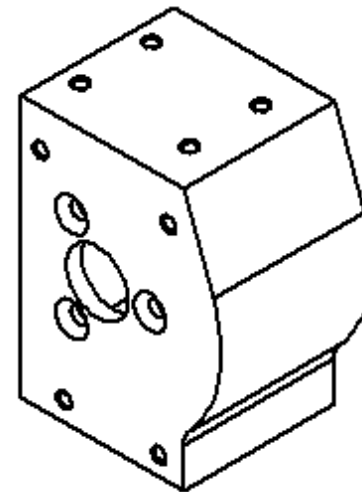
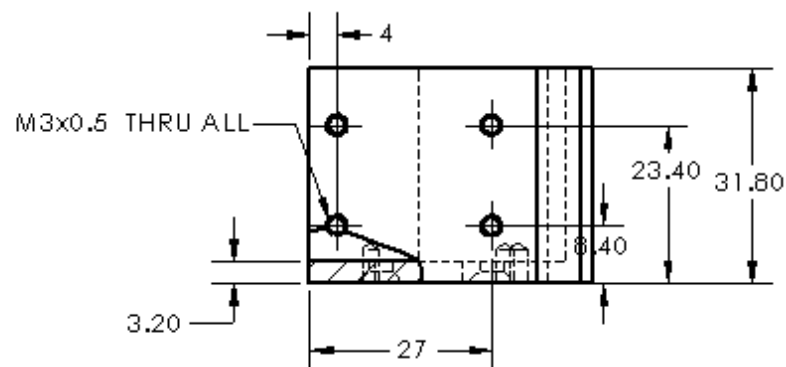
SHEET 1 OF 1

SCALE: 2:3

REV

SIZE

A



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM

TOLERANCES:

NO DECIMAL $\pm .5$

ONE PLACE DECIMAL $\pm .1$

ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: AL -6061

TITLE: REAR LEFT MOTOR HOUSING

DRAWN BY: CG

DWG #: 323

SHEET 1 OF 1

SCALE: 1:1

REV

SIZE

A

4X4 TRAXXAS SLASH
DRIVETRAIN AND
SUSPENSION
ASSEMBLY



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: MISC

TITLE: REAR LEFT TRAXXAS ASSEMBLY

DRAWN BY: CG

DWG #: 322

SHEET 1 OF 1

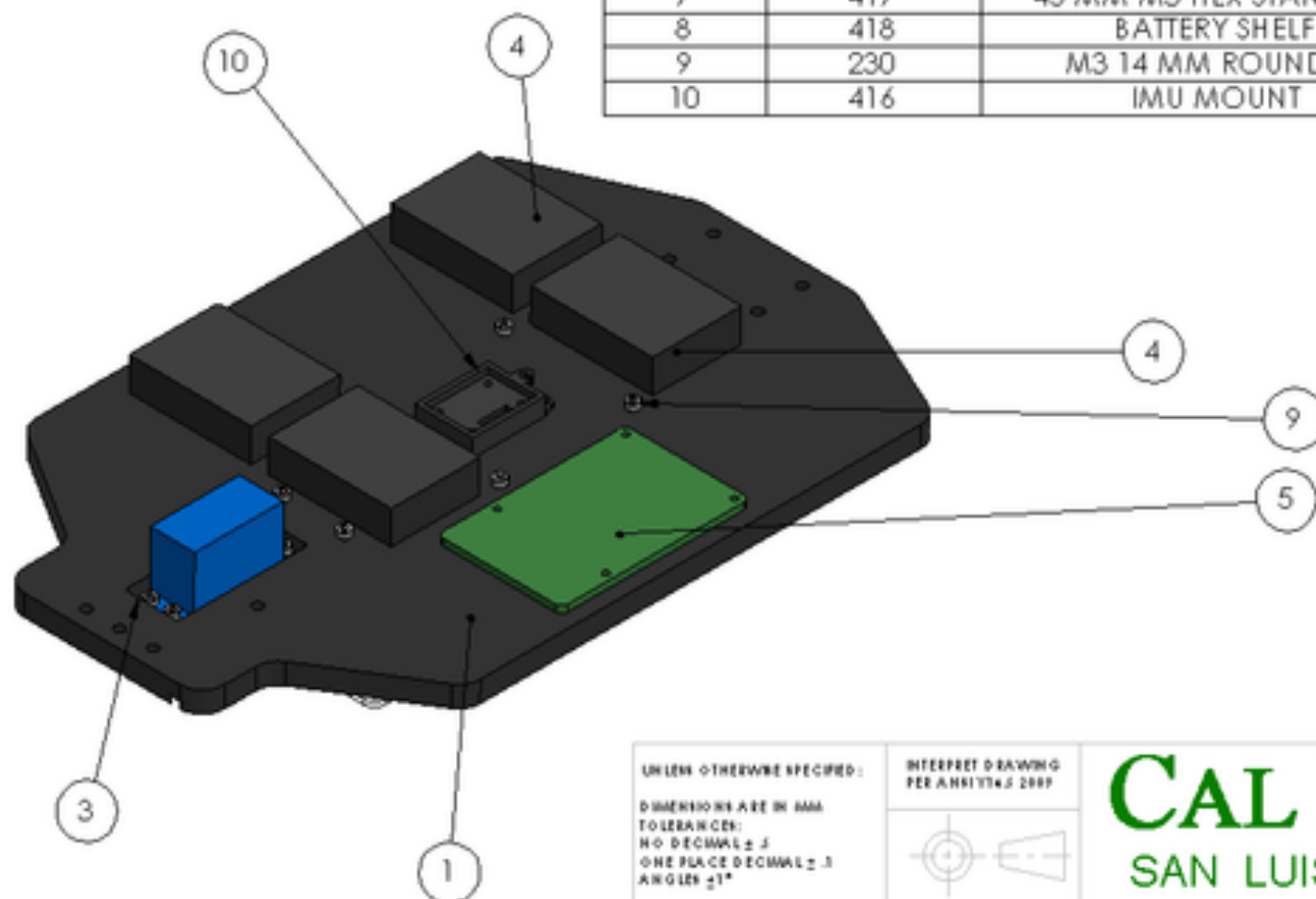
SCALE: 2:3

REV

SIZE

A

ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	401	CHASSIS	1
3	402	TRAXXAS SERVO	1
4	404	MAXON MOTOR DRIVER	4
5	408	RASPBERRY PI/CIRCUIT BOARD	1
6	209	M3 X 0.5 10MM SOCKET	4
7	417	45 MM M3 HEX STANDOFF	6
8	418	BATTERY SHELF	1
9	230	M3 14 MM ROUNDED	6
10	416	IMU MOUNT	1



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: VARIOUS

TITLE: CHASSIS ASSEMBLY

DRAWN BY: CG

DWG #: 400

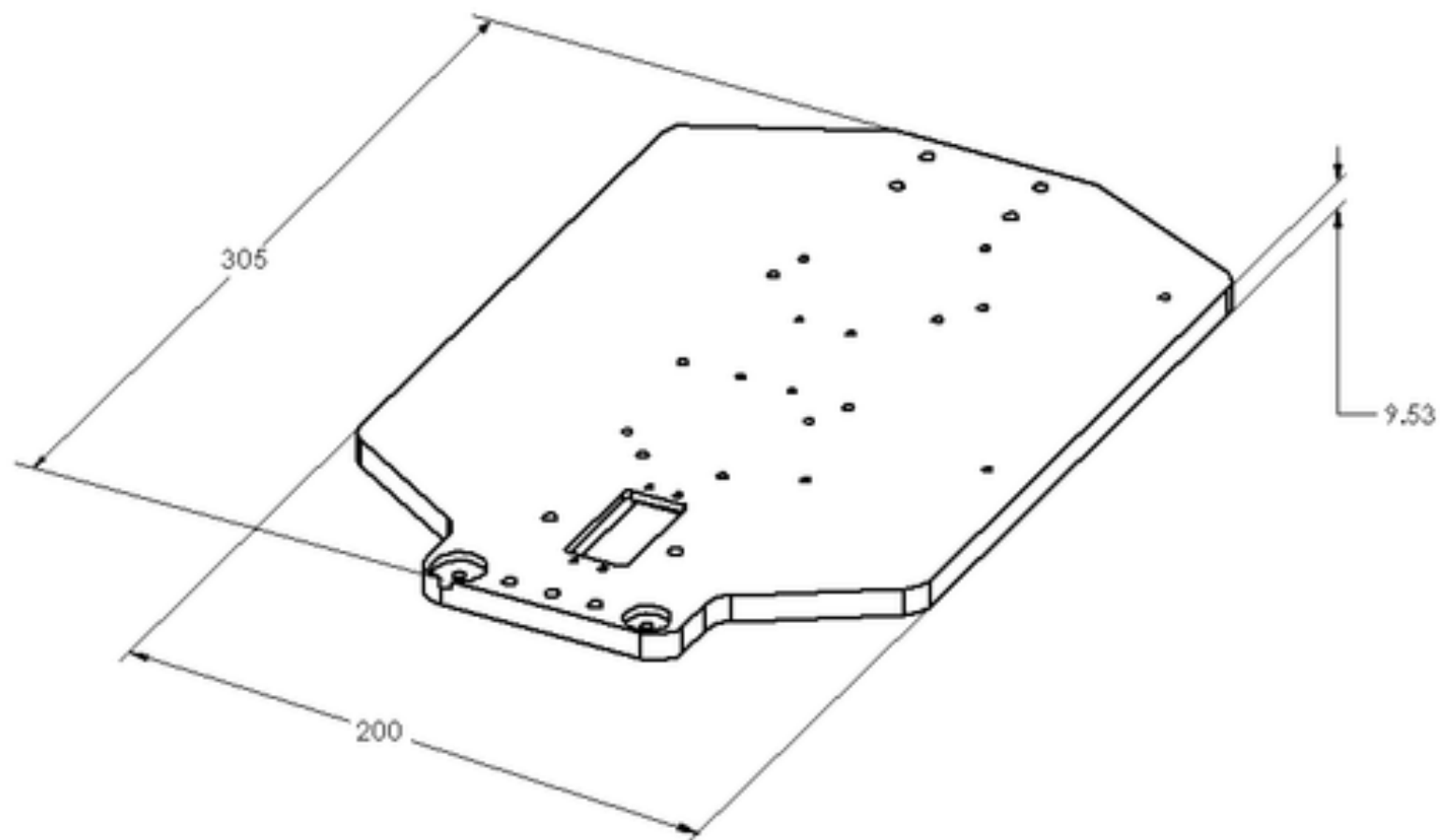
SHEET 1 OF 1

SCALE: 1:2

REV

SIZE

A



UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN MM
TOLERANCES:
NO DECIMAL $\pm .5$
ONE PLACE DECIMAL $\pm .1$
ANGLES $\pm 1^\circ$

INTERPRET DRAWING
PER ANSI Y14.5 2009



CAL POLY
SAN LUIS OBISPO

MATERIAL: Nylon (6)

TITLE: Chassis

DRAWN BY: CG

DWG #: 401A

SHEET 1 OF 1

SCALE: 1:2

REV

SIZE

A

402 – Traxxas Servo

Waterproof Digital Micro Servo

- 41.7 oz-in torque
- Transit time 0.11 sec/60°

Part #2080



403 – Traxxas Receiver



Waterproof Receiver Box (#3924) with TQi TSM Traxxas Link Receiver (#6533)

EPOS4**Feature Chart**










maxon motor control's EPOS4 products are small-sized, full digital, smart positioning control units. Their high power density allows flexible use for brushed DC and brushless EC (BLDC) motors up to approximately 750 Watts with various feedback options, such as Hall sensors, incremental encoders as well as absolute sensors in a multitude of drive applications.


EPOS4 controllers are specially designed to be commanded and controlled as a slave node in the CANopen network. In addition, the units can be operated via any USB or RS232 communication port of a Windows or Linux workstation. Moreover, the integrated extension interface allows optional communication interfaces, such as EtherCAT or other additional functionalities.

Latest technology, such as field-oriented control (FOC) and acceleration/velocity feed forward in combination with highest control cycle rates allow sophisticated, ease-of-use motion control.



Legend: ✓ = included / nnnnnn = order number / ** = available shortly

	EPOS4 Module 24/1.5 (536630)	EPOS4 Compact 24/1.5 CAN (546714)	EPOS4 Module 50/5 (534130)	EPOS4 Compact 50/5 CAN (541718)	EPOS4 Module 50/8 (504384)	EPOS4 Compact 50/8 CAN (520885)	EPOS4 Module 50/15 (504383)	EPOS4 Compact 50/15 CAN (520886)
 for comparison purposes: US Half Dollar coin (Ø30.6 mm)								
Communication Interfaces								
CANopen Slave	max. 1 Mbit/s							
CANopen Application Layer and Communication Profile	CiA 301							
CANopen Layer Setting Services and Protocol (LSS)	CiA 305**							
CANopen Device Profile Drives and Motion Control	CiA 402							
USB 2.0 / USB 3.0	Full speed							
Gateway function USB-to-CAN	✓							
RS232	max. 115'200 bit/s							
Gateway function RS232-to-CAN	✓							
EtherCAT Slave (IEC 61158)	with optional extension module**							
Motors								
Brushed DC motors up to (continuous / max.)	36 W / 108 W	36 W / 108 W	250 W / 750 W	250 W / 750 W	400 W / 1'500 W	400 W / 1'500 W	750 W / 1'500 W	750 W / 1'500 W
Brushless EC motors (BLDC) up to (continuous / max.)	36 W / 108 W	36 W / 108 W	250 W / 750 W	250 W / 750 W	400 W / 1'500 W	400 W / 1'500 W	750 W / 1'500 W	750 W / 1'500 W

 BOSCH	BNO055 Data sheet	Page 2
--	----------------------	--------

BNO055

INTELLIGENT ABSOLUTE ORIENTATION SENSOR, 9-AXIS SENSOR FUSION ALL-IN-ONE WINDOWS 8.x COMPLIANT SENSOR HUB

Basic Description

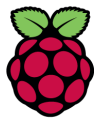
Key features:

- | | |
|---|--|
| <ul style="list-style-type: none"> • Outputs fused sensor data • 3 sensors in one device • Small package • Power Management • Common voltage supplies • Digital interface • Consumer electronics suite | <p>Quaternion, Euler angles, Rotation vector,
Linear acceleration, Gravity, Heading
an advanced triaxial 16bit gyroscope, a versatile,
leading edge triaxial 14bit accelerometer and a
full performance geomagnetic sensor</p> <p>LGA package 28 pins
Footprint 3.8 x 5.2 mm², height 1.13 mm²
Intelligent Power Management: normal,
low power and suspend mode available
V_{DD} voltage range: 2.4V to 3.6V
HID-I2C (Windows 8 compatible), I²C, UART
V_{DDIO} voltage range: 1.7V to 3.6V
MSL1, RoHS compliant, halogen-free
Operating temperature: -40°C ... +85°C</p> |
|---|--|

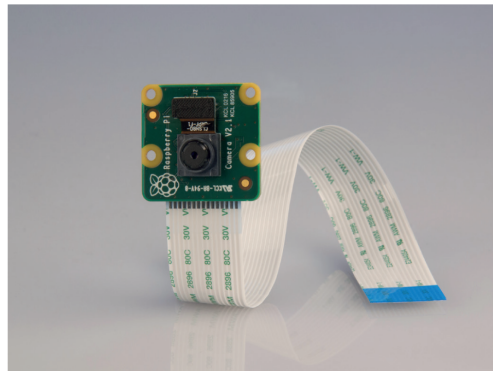
Key features of integrated sensors:

Accelerometer features

- | | |
|--|---|
| <ul style="list-style-type: none"> • Programmable functionality • On-chip interrupt controller | <p>Acceleration ranges $\pm 2g/\pm 4g/\pm 8g/\pm 16g$
Low-pass filter bandwidths 1kHz - <8Hz
Operation modes:</p> <ul style="list-style-type: none"> - Normal - Suspend - Low power - Standby - Deep suspend <p>Motion-triggered interrupt-signal generation for</p> <ul style="list-style-type: none"> - any-motion (slope) detection - slow or no motion recognition - high-g detection |
|--|---|



Raspberry Pi



Camera Module

Product Name	Raspberry Pi Camera Module
Product Description	High Definition camera module compatible with all Raspberry Pi models. Provides high sensitivity, low crosstalk and low noise image capture in an ultra small and lightweight design. The camera module connects to the Raspberry Pi board via the CSI connector designed specifically for interfacing to cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data to the processor.
RS Part Numer	913-2664
Specifications	
Image Sensor	Sony IMX 219 PQ CMOS image sensor in a fixed-focus module.
Resolution	8-megapixel
Still picture resolution	3280 x 2464
Max image transfer rate	1080p: 30fps (encode and decode) 720p: 60fps
Connection to Raspberry Pi	15-pin ribbon cable, to the dedicated 15-pin MIPI Camera Serial Interface (CSI-2).
Image control functions	Automatic exposure control Automatic white balance Automatic band filter Automatic 50/60 Hz luminance detection Automatic black level calibration
Temp range	Operating: -20° to 60° Stable image: -20° to 60°
Lens size	1/4"
Dimensions	23.86 x 25 x 9mm
Weight	3g

HRLV-MaxSonar® - EZ™ Series

High Resolution, Precision, Low Voltage Ultrasonic Range Finder

MB1003, MB1013, MB1023, MB1033, MB1043

The HRLV-MaxSonar-EZ sensor line is the most cost-effective solution for applications where precision range-finding, low-voltage operation, and low-cost are needed. This sensor component module allows users of other more costly precision rangefinders to lower the cost of their systems without sacrificing performance.

The HRLV-MaxSonar-EZ sensor line provides high accuracy and high resolution ultrasonic proximity detection and ranging in air, in a package less than one cubic inch. This sensor line features 1-mm resolution, target-size and operating-voltage compensation for improved accuracy, superior rejection of outside noise sources, internal speed-of-sound temperature compensation and optional external speed-of-sound temperature compensation. This ultrasonic sensor detects objects from 1-mm to 5-meters, senses range to objects from 30-cm to 5-meters, with large objects closer than 30-cm are typically reported as 30-cm¹. The interface output formats are pulse width, analog voltage, and serial digital in either RS232 or TTL. Factory calibration is standard. ¹See Close Range Operation



Precision Range Sensing

- Range-finding at a fraction of the cost of other precision rangefinders
- Reading-to-reading stability of 1-mm at 1-meter is typical
- Accuracy is factory-matched at 1-meter to 0.1% providing a typical large target accuracy of 1% or better for most voltages and uses²
- Calibrated acoustic detection zones allows selection of the part number that matches a specific application
- Compensation for target size variation and operating voltage range
- Standard internal temperature compensation and optional external temperature compensation

Range Outputs

- Pulse width, (1uS/mm)
- Analog Voltage, (5mm resolution)
- Serial, (RS232 or TTL using solder-able jumper or volume orders available as no-cost factory installed jumper)

Easy to Use Component Module

- Gracefully handles other ultrasonic sensors⁴
- Stable and reliable range readings and excellent noise rejection make the sensor easy to use
- Easy to use interface with distance provided in a variety of outputs
- Target size compensation provides greater consistency and accuracy when switching targets
- Sensor automatically handles acoustic noise^{2,3}
- Sensor ignores other acoustic noise sources
- Small and easy to mount
- Calibrated sensor eliminates most sensor to sensor variations
- Very low power ranger, excellent for multiple sensors or battery based systems

General Characteristics

- Low-cost ultrasonic rangefinder
- Size less than 1 cubic inch with easy mounting

- Object proximity detection from 1-mm to 5-meters
- Resolution of 1-mm
- Excellent³ Mean Time Between Failure (MTBF)
- Triggered operation yields a real-time
- 100mS measurement cycle
- Free run operation uses a 2Hz filter, with 100mS measurement and output cycle
- Operating temperature range from -15°C to +65°C, provided proper frost prevention is employed
- Operating voltage from 2.5V to 5.5V
- Nominal current draw of 2.5mA at 3.3V, and 3.1mA at 5V
- Low current draw reduces current drain for battery operation
- Fast first reading after power-up eases battery requirements

Notes:

² Users are encouraged to evaluate the sensor performance in their application.

³ By design.

⁴ See page 5 for multi-sensor operation

Close Range Operation

Applications requiring 100% reading-to-reading reliability should not use MaxSonar sensors at a distance closer than 30cm. Although most users find MaxSonar sensors to work reliably from 0 to 30cm for detecting objects in many applications, MaxBotix® Inc. does not guarantee operational reliability for objects closer than the minimum reported distance. Because of ultrasonic physics, these sensors are unable to achieve 100% reliability at close distances.

Warning: Personal Safety Applications

We do not recommend or endorse this product be used as a component in any personal safety applications. This product is not designed, intended or authorized for such use. These sensors and controls do not include the self-checking redundant circuitry needed for such use. Such unauthorized use may create a failure of the MaxBotix® Inc. product which may result in personal injury or death. MaxBotix® Inc. will not be held liable for unauthorized use of this component.



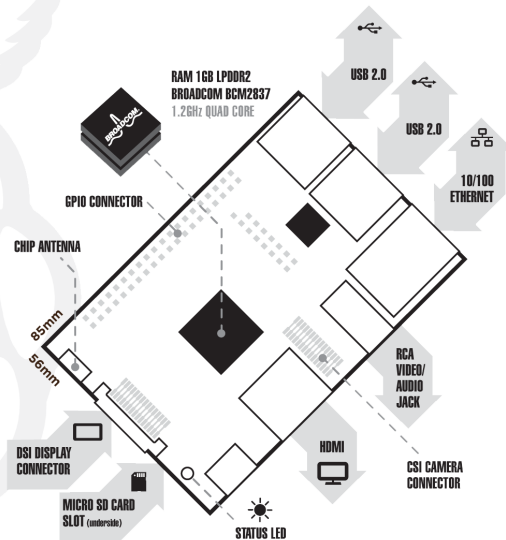
Raspberry Pi

Raspberry Pi 3 Model B

Product Name Raspberry Pi 3

Product Description The Raspberry Pi 3 Model B is the third generation Raspberry Pi. This powerful credit-card sized single board computer can be used for many applications and supersedes the original Raspberry Pi Model B+ and Raspberry Pi 2 Model B. Whilst maintaining the popular board format the Raspberry Pi 3 Model B brings you a more powerful processor, 10x faster than the first generation Raspberry Pi. Additionally it adds wireless LAN & Bluetooth connectivity making it the ideal solution for powerful connected designs.

RS Part Number 896-8660



www.rs-components.com/raspberrypi



Raspberry Pi

Raspberry Pi 3 Model B

Specifications

Processor	Broadcom BCM2387 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure 1GB LPDDR2
Memory	Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
Operating System	
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V1, 2.5A
Connectors:	
Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4) Composite RCA (PAL and NTSC)
Audio Output	Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	Push/pull Micro SDIO

Key Benefits

- Low cost
- 10x faster processing
- Consistent board format
- Added connectivity

Key Applications

- Low cost PC/tablet/laptop
- Media centre
- Industrial/Home automation
- Print server
- Web camera
- Wireless access point
- Environmental sensing/monitoring (e.g. weather station)
- IoT applications
- Robotics
- Server/cloud server
- Security monitoring
- Gaming

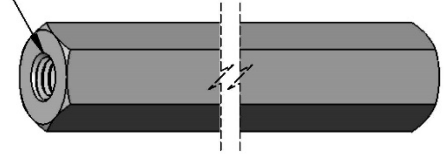


www.rs-components.com/raspberrypi

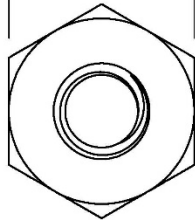
ZIPPY Compact 4000mAh 7S 25C Lipo Pack



M3 x 0.5 mm Thread



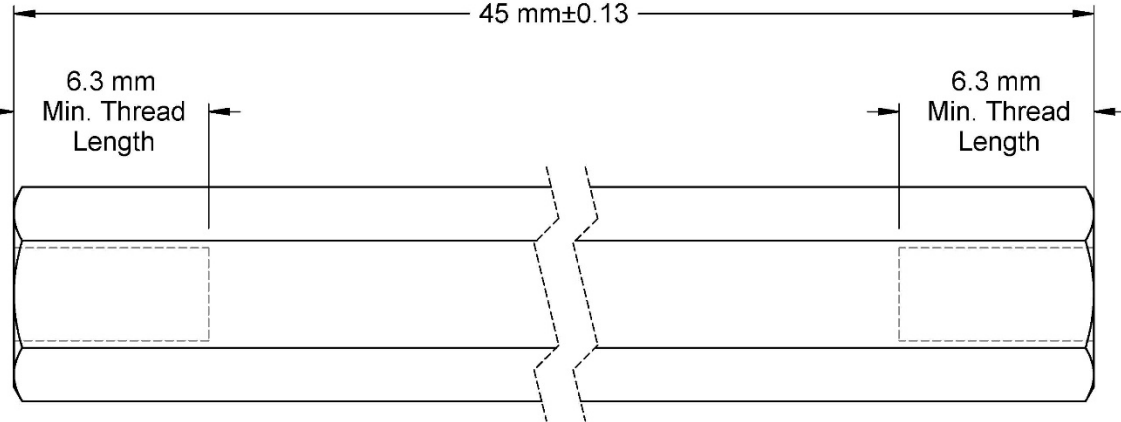
6 mm±0.13



45 mm±0.13

6.3 mm
Min. Thread
Length

6.3 mm
Min. Thread
Length



McMASTER-CARR CAD

<http://www.mcmaster.com>
© 2015 McMaster-Carr Supply Company

Information in this drawing is provided for reference only.

PART
NUMBER **411**

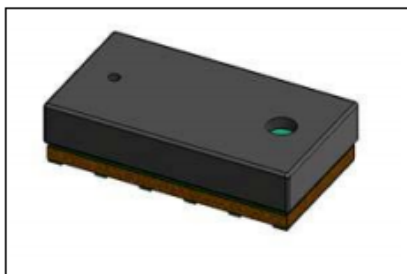
Female Threaded
Hex Standoff



VL53L0X

World smallest Time-of-Flight ranging and gesture detection sensor

Datasheet - production data



Features

- Fully integrated miniature module
 - 940nm Laser VCSEL
 - VCSEL driver
 - Ranging sensor with advanced embedded micro controller
 - 4.4 x 2.4 x 1.0mm
- Fast, accurate distance ranging
 - Measures absolute range up to 2m
 - Reported range is independent of the target reflectance
 - Operates in high infrared ambient light levels
 - Advanced embedded optical cross-talk compensation to simplify cover glass selection
- Eye safe
 - Class 1 laser device compliant with latest standard IEC 60825-1:2014 - 3rd edition
- Easy integration
 - Single reflowable component
 - No additional optics
 - Single power supply
 - I2C interface for device control and data transfer
 - Xshutdown (Reset) and interrupt GPIO
 - Programmable I2C address

Applications

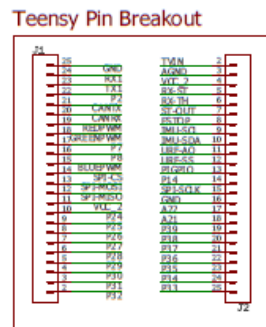
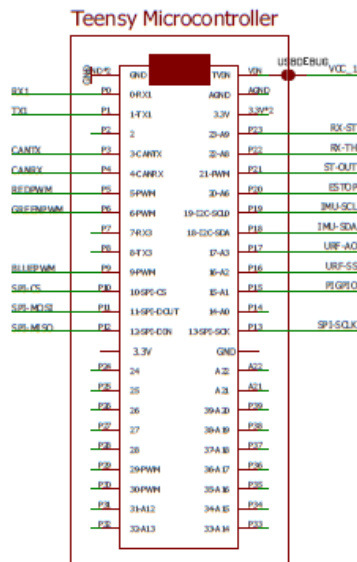
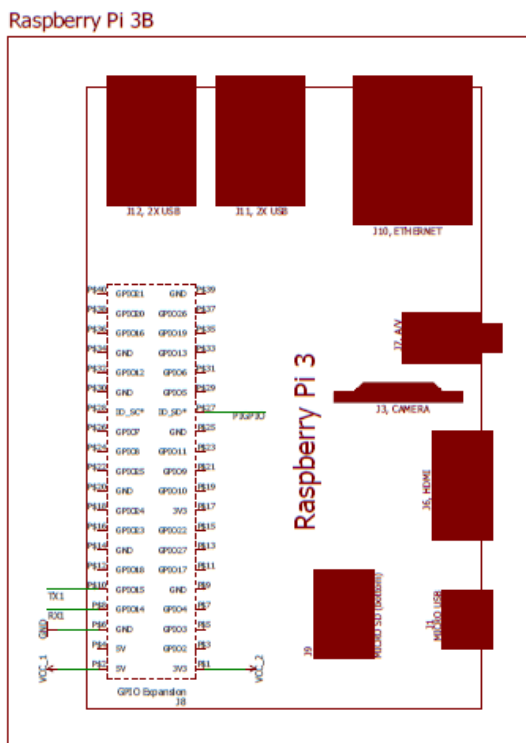
- User detection for Personal Computers/ Laptops/Tablets and IoT (Energy saving).
- Robotics (obstacle detection).
- White goods (hand detection in automatic faucets, soap dispensers etc...)
- 1D gesture recognition.
- Laser assisted Auto-Focus. Enhances and speeds-up camera AF system performance, especially in difficult scenes (low light levels, low contrast) or fast moving video mode.

Description

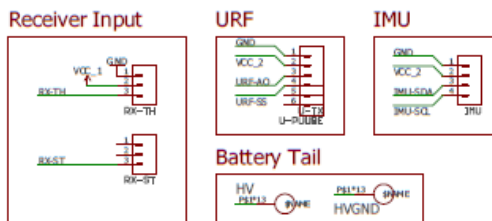
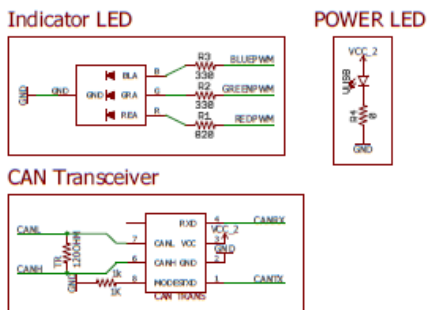
The VL53L0X is a new generation Time-of-Flight (ToF) laser-ranging module housed in the smallest package on the market today, providing accurate distance measurement whatever the target reflectances unlike conventional technologies. It can measure absolute distances up to 2m, setting a new benchmark in ranging performance levels, opening the door to various new applications.

The VL53L0X integrates a leading-edge SPAD array (Single Photon Avalanche Diodes) and embeds ST's second generation FlightSense™ patented technology.

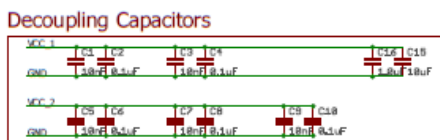
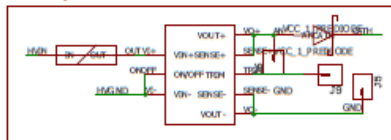
The VL53L0X's 940nm VCSEL emitter (Vertical Cavity Surface-Emitting Laser), is totally invisible to the human eye, coupled with internal physical infrared filters, it enables longer ranging distance, higher immunity to ambient light and better robustness to cover-glass optical cross-talk.



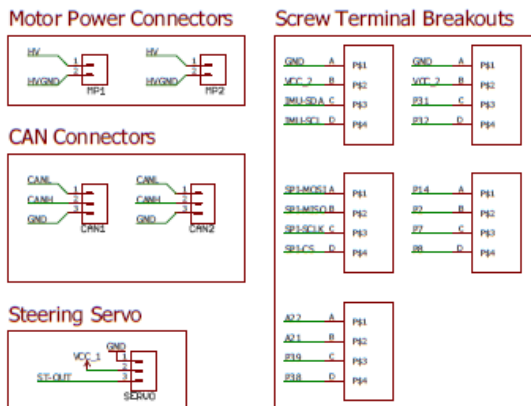
Processors



Input Connectors

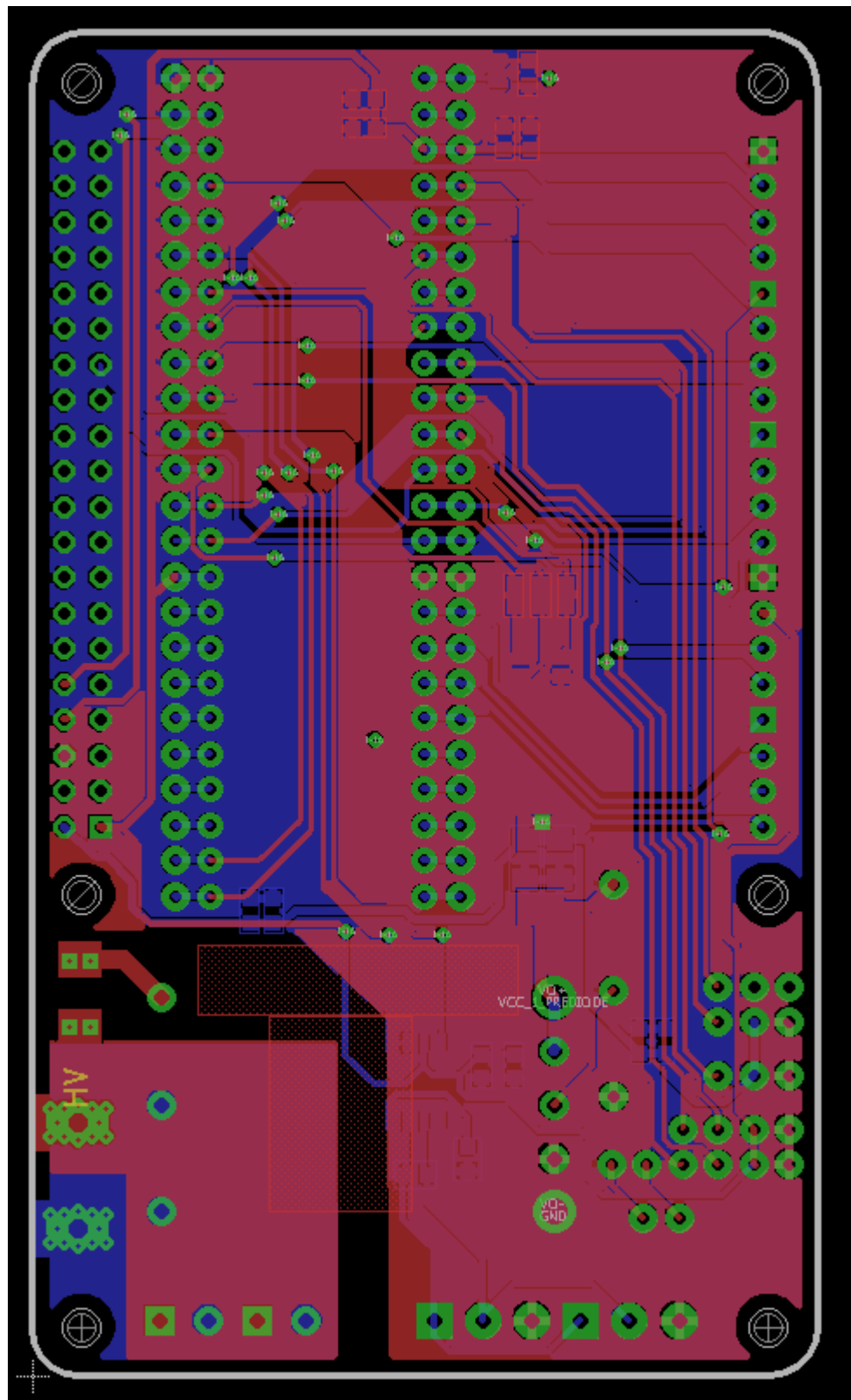


Peripheral Components



Output Connectors

421 – Motherboard Layout

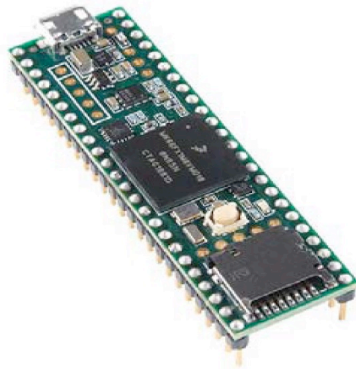


422 – Teensy 3.6 Datasheet



Teensy 3.6 (Headers)

DEV-14058 RoHS



Description: The Teensy is a breadboard-friendly development board with loads of features in a, well, teensy package. Each Teensy 3.6 comes with headers already attached and pre-flashed with a bootloader so you can program it using the onboard USB connection; no external programmer needed! You can program for the Teensy in your favorite program editor using C, or you can install the Teensyduino add-on for the Arduino IDE and write Arduino sketches for it!

The processor on the Teensy also has access to the USB and can emulate any kind of USB device you need it to be, making it great for USB-MIDI and other HID projects. The 32-bit, 180MHz processor brings a few other features to the table as well, such as multiple channels of Direct Memory Access, several high-resolution ADCs and even an I2S digital audio interface! There are also four separate interval timers, plus a delay timer! Oh yeah, and all digital pins have interrupt capability and are 3.3V tolerant.

All of this functionality is jammed into a 62.3mm x 18mm board with all headers on a 0.1" grid so you can slap it on a breadboard and get to work! The Teensy 3.6 (as well as its sibling, the Teensy 3.5) is larger, faster and capable of more complex projects, especially with its onboard microSD card port. An upgraded ARM Cortex MCU (180MHz from 72MHz), more memory (1M from 256K)—as well as more RAM, EEPROM and accessible pins—make up the key new features of this board. The Teensy 3.6 is slightly scaled up from the Teensy 3.5.

Note: This does not come with a USB cable.

Dimensions: 62.3mm x 18mm x 4.2mm (2.5in x 0.7in x 0.2in)

Features:

- 180 MHz ARM Cortex-M4 with Floating Point Unit
- 1M Flash, 256K RAM, 4K EEPROM
- Microcontroller Chip MK66FX1M0VMD18
- USB High Speed (480Mbit/sec) Port
- 2 CAN Bus Ports
- 32 General Purpose DMA Channels
- 22 PWM Outputs
- 4 I2C Ports
- 11 Touch-Sensing Inputs
- 62 I/O Pins (42 breadboard friendly)
- 25 Analog Inputs to 2 ADCs with 13-bit resolution
- 2 Analog Outputs (DACs) with 12-bit resolution
- USB Full-Speed (12Mbit/sec) Port
- Ethernet mac, capable of full 100Mbit/sec speed
- Native (4-bit SDIO) microSD card port
- I2S Audio Port, 4-Channel Digital Audio Input and Output
- 14 Hardware Timers
- Cryptographic Acceleration Unit
- Random Number Generator
- CRC Computation Unit
- 6 Serial Ports (2 with FIFO and Fast Baud Rates)
- 3 SPI Ports (1 with FIFO)
- Real-Time Clock (RTC)
- Pre-soldered Headers

SHHD003A0A Hammerhead* Series; DC-DC Converter Power Modules

18-75Vdc Input; 5.0Vdc, 3A, 15W Output



RoHS Compliant

Applications

- Wireless Networks
- Hybrid power architectures
- Optical and Access Network Equipment
- Enterprise Networks including Power over Ethernet (PoE)
- Industrial markets

Options

- Negative Remote On/Off logic
- Surface Mount/Tape and Reel (-SR Suffix)

Description

The SHHD003A0A Hammerhead series power modules are isolated dc-dc converters that operate over an ultra-wide input voltage range of 18 Vdc -75Vdc and provide a single precisely regulated output voltage at 5.0Vdc. This series is a low cost, smaller size alternative to the existing LW/LAW/LC/SC/SW with enhanced performance parameters. The output is fully isolated from the input, allowing versatile polarity configurations and grounding connections. The modules exhibit high efficiency of 87.0% typical at full load. Built-in filtering for both input and output minimizes the need for external filtering. The module is fully self-protected with output over-current and over-voltage, over-temperature and input under voltage shutdown control. Optional features include negative or positive on/off logic and SMT connections.

Features

- Compliant to RoHS II EU "Directive 2011/65/EU (-Z versions)
- Compliant to REACH Directive (EC) No 1907/2006
- Ultra-wide Input Voltage Range, 18Vdc to 75Vdc
- No minimum load
- High efficiency – 87.0% at full load ($V_{in}=24$ or 48Vdc)
- Constant switching frequency
- Low output ripple and noise
- Small Size and low profile, follows industry standard 1x1 footprint
27.9mm x 24.4mm x 8.5mm (MAX)
(1.10 x 0.96 x 0.335 in)
- Surface mount (SMT) or Through hole (TH)
- Reflow process compliant, both SMT and TH versions
- Positive Remote On/Off logic
- Output overcurrent/voltage protection (hiccup)
- Over-temperature protection
- Output Voltage adjust: 90% to 110% of $V_{o,nom}$
- Wide operating temperature range (-40°C to 85°C)
- CAN/CSA[†] C22.2 No. 60950-1-07, 2nd Edition + A1:2011 (MOD), ANSI/UL[#] 60950-1-2011, December 19, 2011; DIN EN 60950-1 (VDE[‡] 0805-1):2011-01 DIN EN 60950-1/A12 (VDE 0805-1/A12):2011-08 EN 60950-1:2006 + A1:2009 + A1:2010 + A12:2011 IEC 60950-1:2005 (2nd Edition); am1:2009
- CE mark meets 2006/95/EC directive[§]
- Meets the voltage and current requirements for ETSI 300-132-2 and complies with and licensed for Basic insulation rating per EN60950-1
- 2250 Vdc Isolation tested in compliance with IEEE 802.3[¶] PoE standards
- ISO** 9001 and ISO 14001 certified manufacturing facilities

* Trademark of General Electric Company

UL is a registered trademark of Underwriters Laboratories, Inc.

† CSA is a registered trademark of Canadian Standards Association.

‡ VDE is a trademark of Verband Deutscher Elektrotechniker e.V.

§ This product is intended for integration into end-user equipment. All of the required procedures of end-use equipment should be followed.

¶ IEEE and 802 are registered trademarks of the Institute of Electrical and Electronics Engineers, Incorporated.

** ISO is a registered trademark of the International Organization of Standards.

Product
FolderSample &
BuyTechnical
DocumentsTools &
SoftwareSupport &
Community**SN65HVD230, SN65HVD231, SN65HVD232**

SLOS346N – MARCH 2001 – REVISED JULY 2015

SN65HVD23x 3.3-V CAN Bus Transceivers**1 Features**

- Operates with a single 3.3 V Supply
- Compatible With ISO 11898-2 Standard
- Low Power Replacement for the PCA82C250 Footprint
- Bus Pin ESD Protection Exceeds ± 16 kV HBM
- High Input Impedance Allows for Up to 120 Nodes on a Bus
- Adjustable Driver Transition Times for Improved Emissions Performance
 - SN65HVD230 and SN65HVD231
- SN65HVD230: Low Current Standby Mode
 - 370 μ A Typical
- SN65HVD231: Ultra Low Current Sleep Mode
 - 40 nA Typical
- Designed for Data Rates⁽¹⁾ up to 1 Mbps
- Thermal Shutdown Protection
- Open Circuit Fail-Safe Design
- Glitch Free Power Up and Power Down Protection for Hot Plugging Applications

⁽¹⁾ The signaling rate of a line is the number of voltage transitions that are made per second expressed in the units bps (bits per second).

2 Applications

- Industrial Automation, Control, Sensors and Drive Systems
- Motor and Robotic Control
- Building and Climate Control (HVAC)
- Telecom and Basestation Control and Status
- CAN Bus Standards Such as CANopen, DeviceNet, and CAN Kingdom

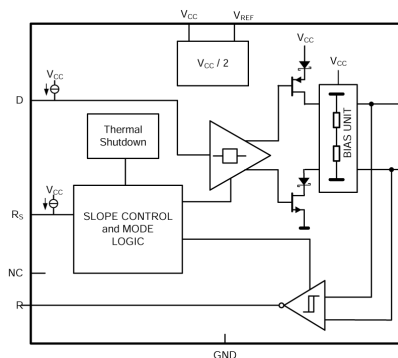
3 Description

The SN65HVD230, SN65HVD231, and SN65HVD232 controller area network (CAN) transceivers are compatible to the specifications of the ISO 11898-2 High Speed CAN Physical Layer standard (transceiver). These devices are designed for data rates up to 1 megabit per second (Mbps), and include many protection features providing device and CAN network robustness. The SN65HVD23x transceivers are designed for use with the Texas Instruments 3.3 V μ Ps, MCUs and DSPs with CAN controllers, or with equivalent protocol controller devices. The devices are intended for use in applications employing the CAN serial communication physical layer in accordance with the ISO 11898 standard.

Device Information⁽¹⁾

PART NUMBER	PACKAGE	BODY SIZE (NOM)
SN65HVD230	SOIC (8)	4.90 mm \times 3.91 mm
SN65HVD231		
SN65HVD232		

⁽¹⁾ For all available packages, see the orderable addendum at the end of the datasheet.

Equivalent Input and Output Schematic Diagrams

An IMPORTANT NOTICE at the end of this data sheet addresses availability, warranty, changes, use in safety-critical applications, intellectual property matters and other important disclaimers. PRODUCTION DATA.

t

Kingbright

3.5x2.8mm SURFACE MOUNT LED LAMP



ATTENTION
OBSERVE PRECAUTIONS
FOR HANDLING
ELECTROSTATIC
DISCHARGE
SENSITIVE
DEVICES

Part Number: AAA3528LSEEZGKBKS

Hyper Red
Green
Blue

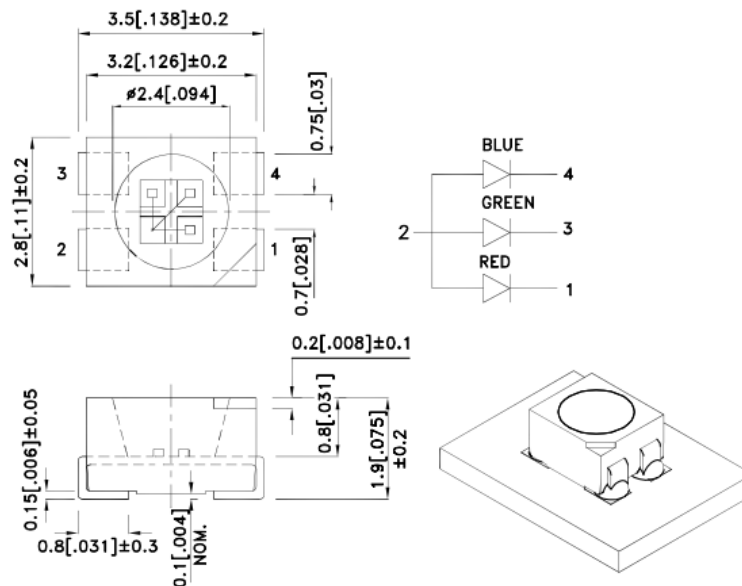
Features

- Suitable for all SMD assembly and solder process.
- Available on tape and reel.
- Package: 2000pcs / reel.
- Moisture sensitivity level : level 3.
- Low current IF=2mA operating.
- RoHS compliant.

Descriptions

- The Hyper Red source color devices are made with AlGaInP on GaAs substrate Light Emitting Diode.
- The Green source color devices are made with InGaN on Sapphire Light Emitting Diode.
- The Blue source color devices are made with InGaN on Sapphire Light Emitting Diode.
- Electrostatic discharge and power surge could damage the LEDs.
- It is recommended to use a wrist band or anti-electrostatic glove when handling the LEDs.
- All devices, equipments and machineries must be electrically grounded.

Package Dimensions



Notes:

1. All dimensions are in millimeters (inches).
2. Tolerance is $\pm 0.25(0.01^\circ)$ unless otherwise noted.
3. The specifications, characteristics and technical data described in the datasheet are subject to change without prior notice.
4. The device has a single mounting surface. The device must be mounted according to the specifications.



SPEC NO: DSAO4456

APPROVED: Wynec

REV NO: V.2B

CHECKED: Allen Liu

DATE: SEP/18/2015

DRAWN: M.Liu

PAGE: 1 OF 8

ERP: 1201008965

Kingbright

1.6X0.8mm SMD CHIP LED LAMP

Part Number: APT1608LZGCK

Green



ATTENTION
OBSERVE PRECAUTIONS
FOR HANDLING
ELECTROSTATIC
DISCHARGE
SENSITIVE
DEVICES

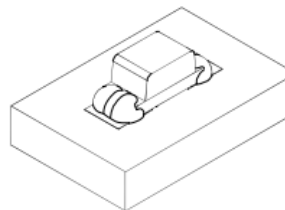
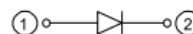
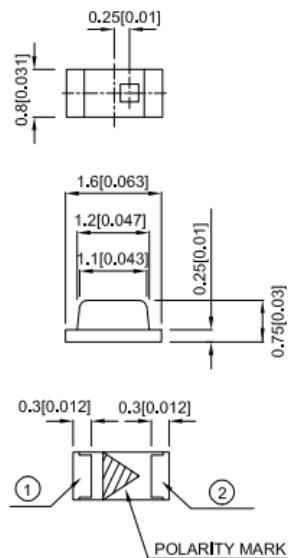
Features

- 1.6mmX0.8mm SMT LED, 0.75mm thickness.
- Low power consumption.
- Wide viewing angle.
- Ideal for backlight and indicator.
- Package: 2000pcs / reel .
- Moisture sensitivity level : level 3.
- Low current IF=2mA operating.
- RoHS compliant.

Descriptions

- The Green source color devices are made with InGaN on Sapphire Light Emitting Diode.
- Electrostatic discharge and power surge could damage the LEDs.
- It is recommended to use a wrist band or anti-electrostatic glove when handling the LEDs.
- All devices, equipments and machineries must be electrically grounded.

Package Dimensions



Notes:

1. All dimensions are in millimeters (inches).
2. Tolerance is $\pm 0.1(0.004")$ unless otherwise noted.
3. The specifications, characteristics and technical data described in the datasheet are subject to change without prior notice.
4. The device has a single mounting surface. The device must be mounted according to the specifications.



SPEC NO: DSAN8392

REV NO: V.2B

DATE: APR/28/2015

PAGE: 1 OF 5

APPROVED: WYNEC

CHECKED: Allen Liu

DRAWN: Q.M.Chen

ERP: 1203014454

Schottky Barrier Rectifier Diode

Lead-less Chip Form



GENERAL DESCRIPTION

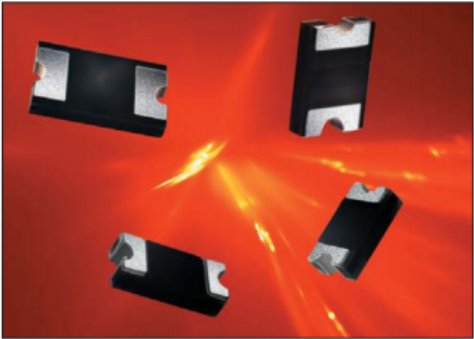
AVX Schottky rectifier diodes offer unique lead-less chip packaging technology which eliminates the lead frame wire bond to give the chip top-bottom symmetry for fewer mounting problems, better heat transfer, and current handling capability (compared to SOD devices).

FEATURES

- Lead-less chip form
- Low Vf
- High current capability
- Low power loss/high efficiency
- UL 94V-0 class package material
- Halogen free

APPLICATIONS

- Switch mode power supplies
- High frequency rectification
- Portable battery powered devices
- Reverse bias protection



MECHANICAL DATA

Case: FRP substrate with epoxy underfill
Terminations: 100% Sn plated (Pb-free), solderable per MIL-STD-750, Method 2026.
Operating Temperature: -55°C to 125°C
Storage Temperature: -55°C to 150°C

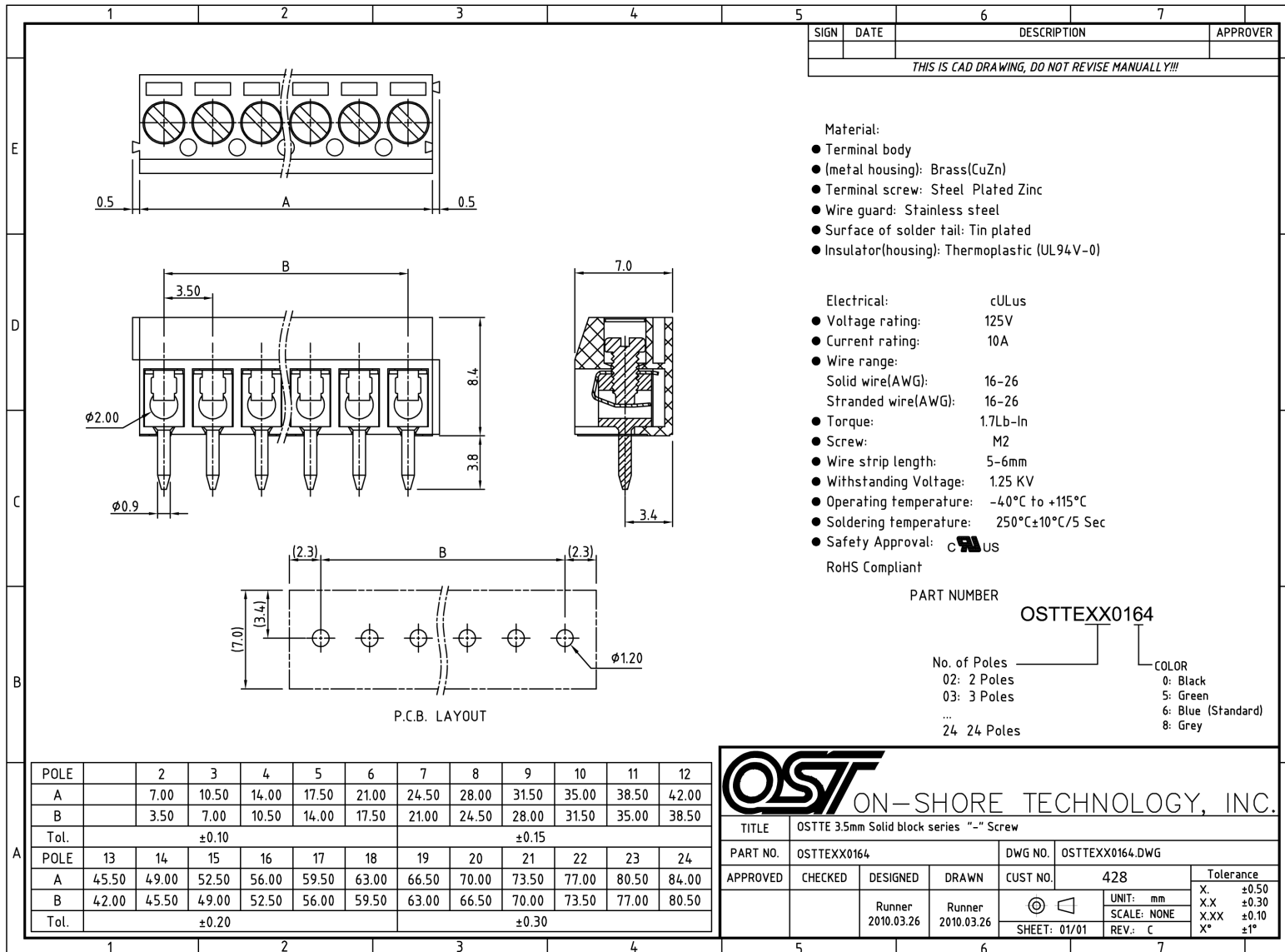
HOW TO ORDER

SD	3220	S	020	S	3R0
Series Schottky Diodes	Size 3220 2114 2010 1206 0805 0603	Thickness S = Standard T = Thin	Voltage 020 = 20V 030 = 30V 040 = 40V 060 = 60V 100 = 100V 150 = 150V 200 = 200V	Vf S = Standard L = Low	Current 0R1 = 0.1 0R2 = 0.2 0R3 = 0.3 0R5 = 0.5 1R0 = 1.0



AVX SCHOTTKY DIODE CURRENTS BY CASE SIZE

Size		Max Forward Current								
EIAJ	JEDEC	.1A	.2A	.3A	.5A	1A	2A	3A	5A	8A
0603	SOD-523	●	●	●						
0805	SOD-323	●	●	●	●	●				
1206	SOD-123				●	●	●	●		
2010	SMA (D0-214AC)					●	●	●	●	
2114	SMB (D0-214AA)							●	●	●
3220	SMC (D0-214AB)							●	●	



F-217



(2.54 mm) .100"



THROUGH-HOLE .025" SQ POST HEADER

Board Mates:

SSW, SSQ, SSM, ESW, ESQ, BCS, BSW, CES, SLW

Cable Mates:

IDSD, IDSS, SMSD, SMSS

SPECIFICATIONS

For complete specifications see www.samtec.com?TSW or www.samtec.com?HTSW

Insulator Material:

TSW: Black Glass Filled Polyester
HTSW: Natural Liquid Crystal Polymer

Terminal Material:

Phosphor Bronze

Plating:

Au or Sn over 50 μ" (1.27 μm) Ni

Operating Temp Range:

-55 °C to +125 °C with Gold

-55 °C to +105 °C with Tin

Voltage Rating:

550 VAC mated with SSW;

500 VAC mated with BCS or ESQ;

450 VAC -RA/-RE mated with BCS or SSM

400 VAC mated with CES

RoHS Compliant: Yes

Lead-Free Solderable:

HTSW: Yes

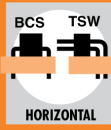
TSW: No, Lead Wave Only

CURRENT RATING (PER PIN)						
TSW mated with						
ESW	SSW	SLW	SSQ	SSM	BCS	SNT
5.2 A	5.7 A	5.2 A	6.3 A	5.2 A	4.6 A	4.3 A
1 POSITION POWERED PER ROW						

RECOGNITIONS

For complete scope of recognitions see www.samtec.com/quality

APPLICATIONS



ALSO AVAILABLE (MOQ Required)

- Other platings
- Contact Samtec.

Note: Some lengths, styles and options are non-standard, non-returnable.

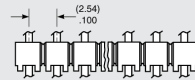
TYPE STRIP

TSW = Standard Strip

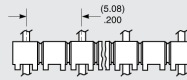
HTSW = Hi-Temp Strip

PIN CENTERS

-1 = .100" (2.54 mm) Centers, All Positions Filled

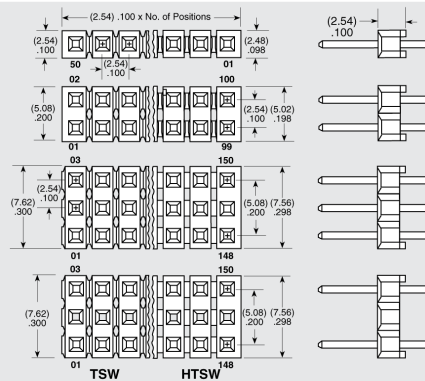


-2 = .200" (5.08 mm) Centers, Every Other Position Filled



NO. PINS PER ROW

Straight Pin Versions



-S

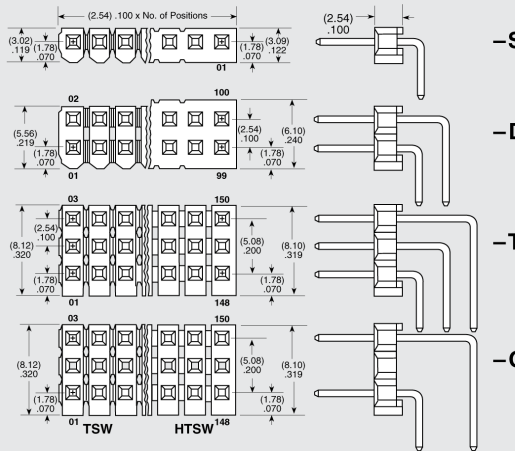
-D

-T

-Q

01 thru 50
= .100" (2.54 mm)
Center Version

Right-Angle Versions



-S

-D

-T

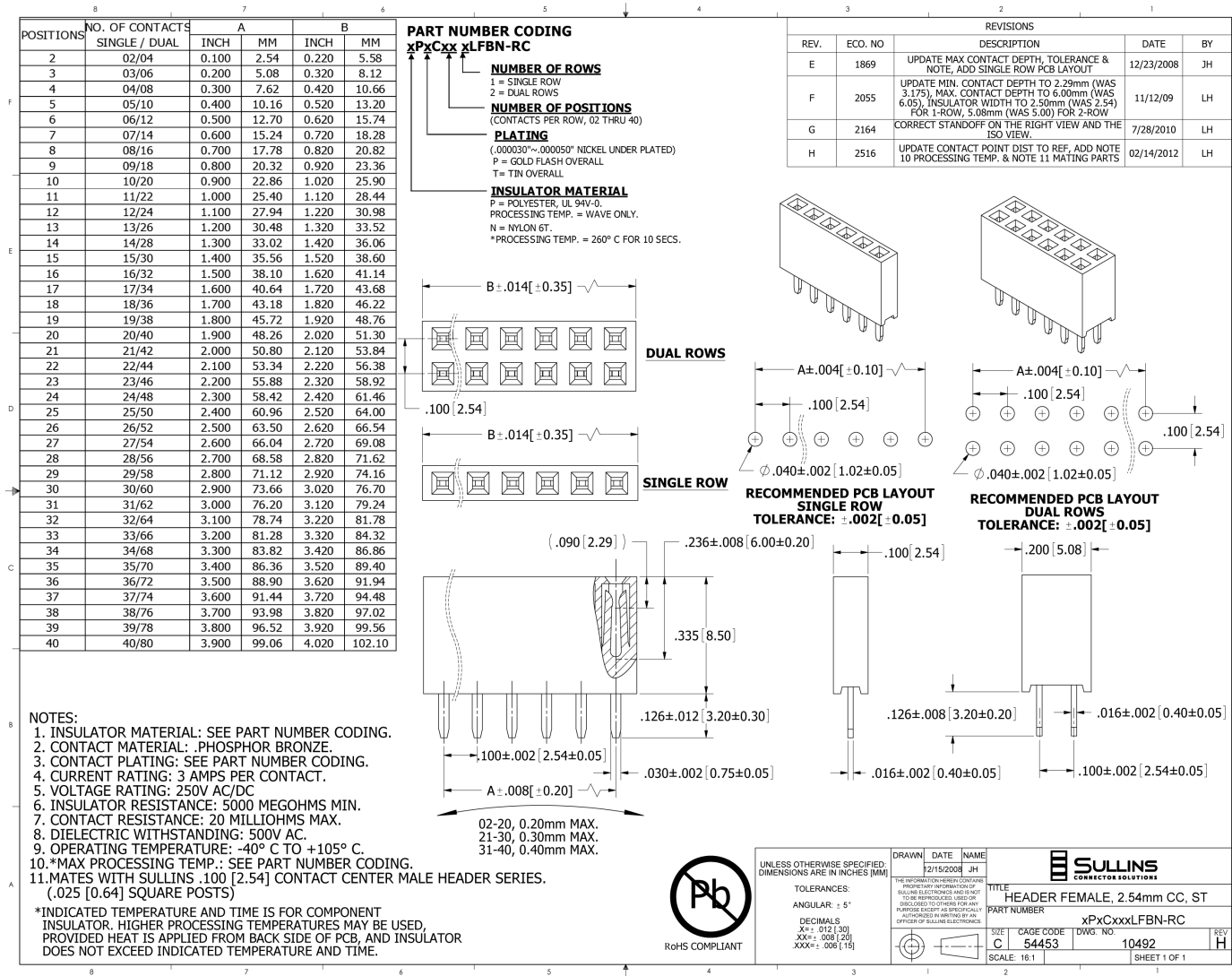
-Q

02 thru 25
= .200" (5.08 mm)
Center Version

WWW.SAMTEC.COM

Due to technical progress, all designs, specifications and components are subject to change without notice.

430 – Female Header Datasheet





Surface Mount Fuses

NANO²® > 157 Fuse and Holder Combination

157 Series – Standard Nano²® Fuse and Clip Assembly



Agency Approvals

AGENCY	AGENCY FILE NUMBER	AMPERE RANGE
	E14721	0.062A ~ 10A
	NBK030205-E10480A NBK030205-E10480B NBK101105-E184655	1A - 1.6A 2A - 5A 6.3A - 10A

Electrical Characteristics for Series

% of Ampere Rating	Opening Time at 25°C
100%	4 hours Minimum
200%	5 secs. Maximum

Electrical Specifications by Item

Ampere Rating (A)	Amp Code	Max Voltage Rating (V)	Interrupting Rating (A)	Fuse Furnished	Nominal Cold Resistance (Ohms)	Nominal Melting Pt (A ² sec)	Agency Approvals	
0.062	.062	125	50A @ 125 VAC/VDC 300A @ 32 VDC	0451.062	5.5372	0.00019	X	
0.080	.080	125		0451.080	4.0500	0.00033	X	
0.100	.100	125		0451.100	3.1000	0.00138	X	
0.125	.125	125		0451.125	1.7059	0.00286	X	
0.160	.160	125		0453.160	1.2157	0.0048	X	
0.200	.200	125		0453.200	1.3971	0.00652	X	
0.250	.250	125		0453.250	1.0496	0.01126	X	
0.315	.315	125		0453.315	0.3881	0.0311	X	
0.375	.375	125		0453.375	0.6100	0.0442	X	
0.400	.400	125		0453.400	0.5600	0.0551	X	
0.500	.500	125		0453.500	0.4200	0.0824	X	
0.630	.630	125		0453.630	0.3050	0.1381	X	
0.750	.750	125		0453.750	0.2450	0.2143	X	
0.800	.800	125		0453.800	0.2120	0.2654	X	
1.0	.001	125		0453001.	0.1530	0.6029	X	X
1.25	1.25	125		04531.25	0.078	0.664	X	X
1.5	.015	125		045301.5	0.0634	0.853	X	X
1.6	.016	125		045301.6	0.0580	1.060	X	X
2.0	.002	125		0453002.	0.0373	0.530	X	X
2.5	.025	125		045302.5	0.0288	1.029	X	X
3.0	.003	125		0453003.	0.0229	1.650	X	X
3.15	3.15	125		04533.15	0.0215	1.920	X	X
3.5	.035	125		045303.5	0.0203	2.469	X	X
4.0	.004	125		0453004.	0.0163	3.152	X	X
5.0	.005	125		0453005.	0.0127	5.566	X	X
6.3	.063	125		045306.3	0.0098	9.17	X	X
7.0	.007	125		0453007.	0.0092	10.32	X	X
8.0	.008	125		0453008.	0.0079	20.23	X	X
10.0	.010	125	35A @ 125 VAC / 50A @ 125 VDC 300A @ 32VDC	0453010.	0.0058	26.46	X	X

1. Cold resistance measured at less than 10% of rated current at 23°C.

2. Pt values stated for 8ms opening time.

3. Agency Approval Table Key: X=Approved or Certified, P=Pending and Blank=Not Approved

4. Have special electrical characteristic needs? Contact Littelfuse to learn more about application specific options.

Description

The 157 Series – Standard Nano Fuse/Clip assembly is a small, square, very fast acting surface mount fuse that is assembled in surface mountable fuse clips. The fuse clip and pre-installed fuse combination can be automatically placed in PC Board in one efficient manufacturing operation. It permits quick and easy replacement of fuses without performing desoldering process, even in the field and without exposing the PC Board to detrimental effects of rework solder heat.

Features

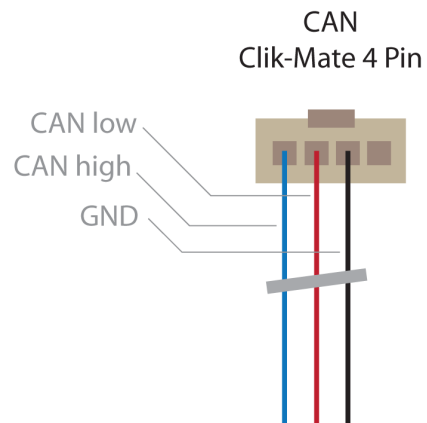
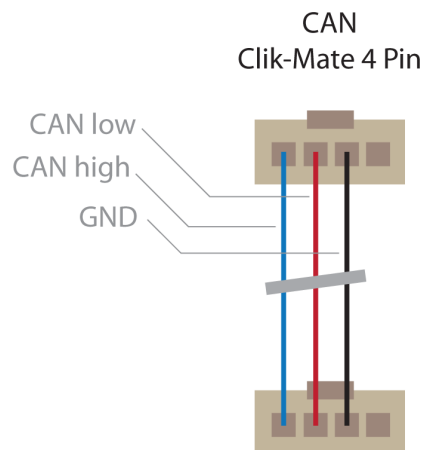
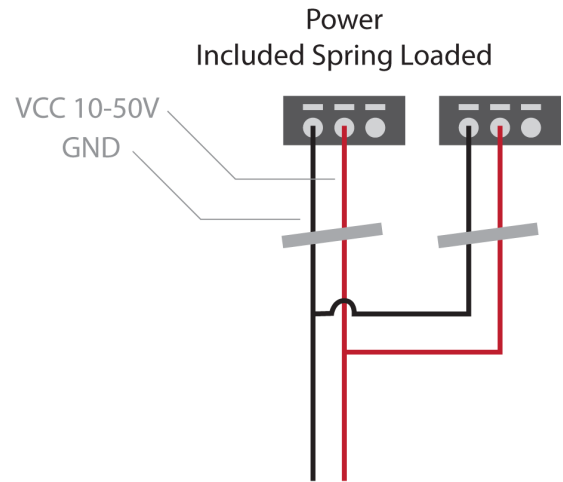
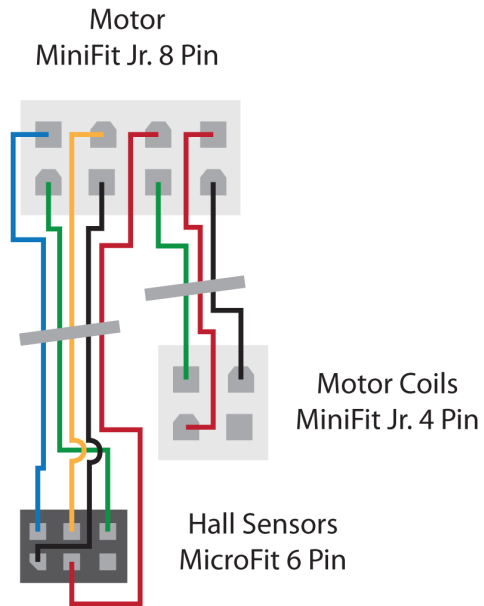
- Surface Mountable, Very Fast Acting Fuse.
- Fully compatible with RoHS/Pb-Free solder alloys and higher temperature profiles associated with leadfree assembly.
- Easily replaceable on PC Board (Field Replaceable)
- RoHS compliant and Halogen Free
- Available in ratings of 0.062 ~ 10 Amperes.

Applications

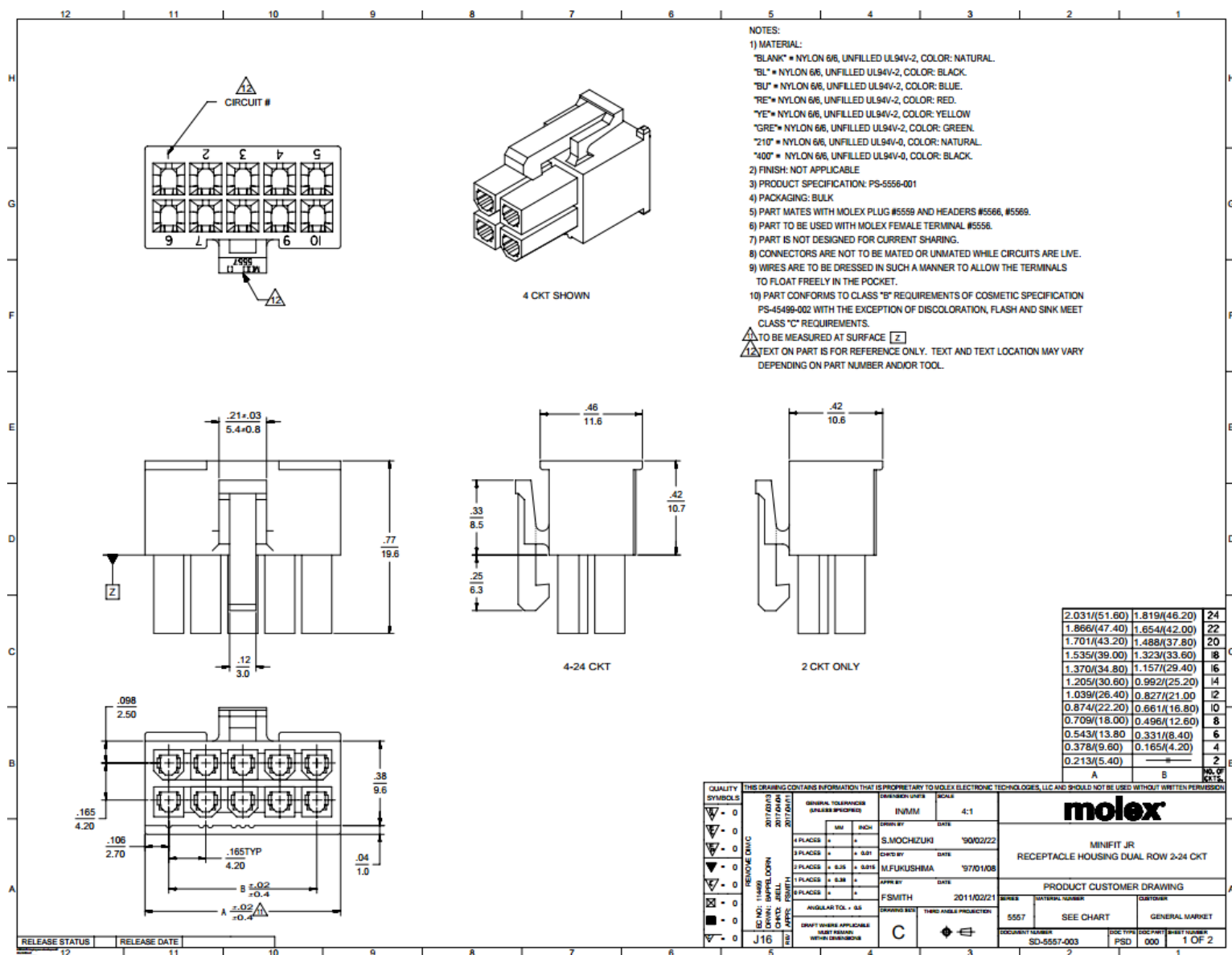
- Instrumentation
- Base Stations
- Telecommunications

Drawing 450 - Cabling Diagrams

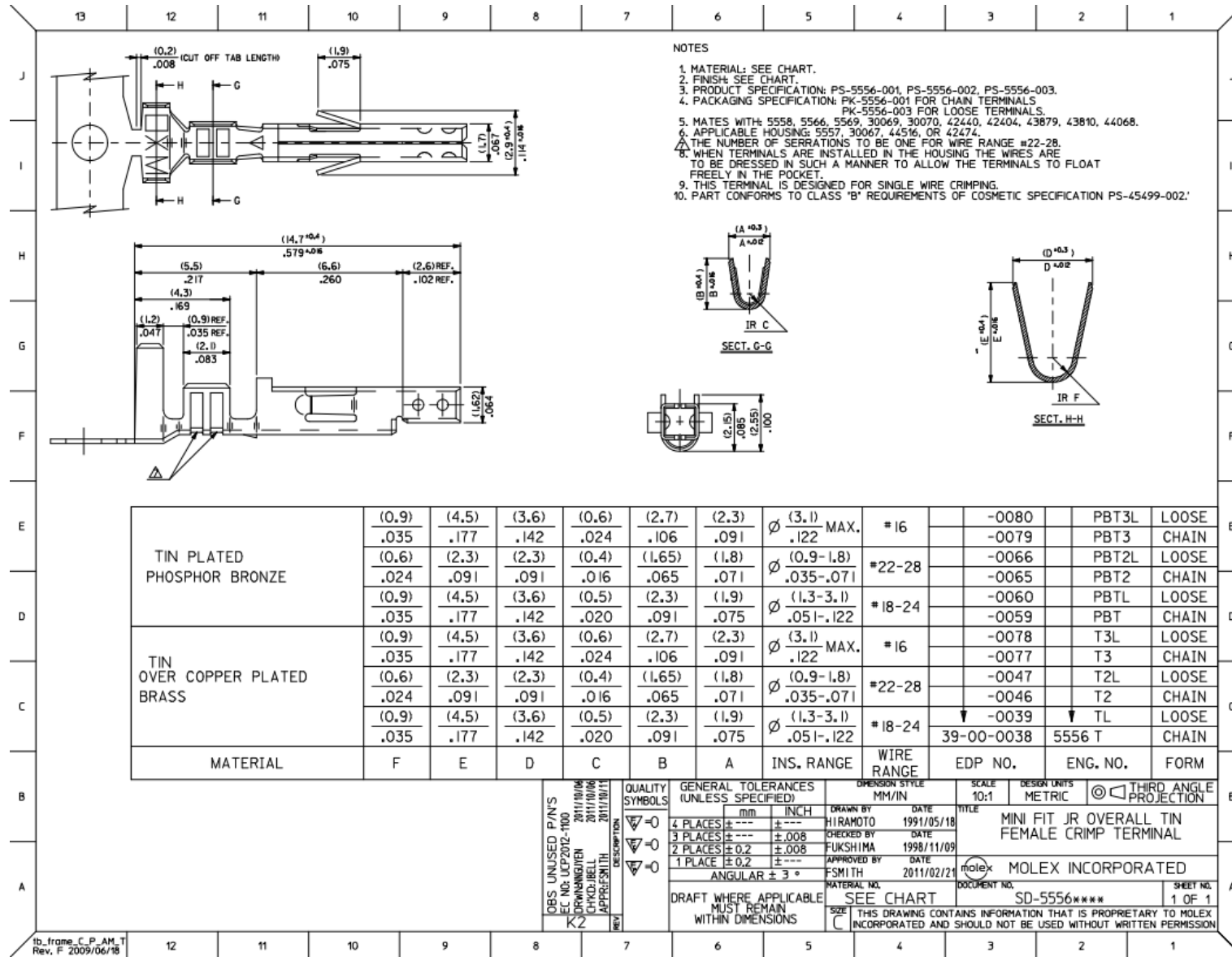
Looking at back of connectors



451 – Minifit Jr Receptacle



452 – Minifit Jr Crimp Receptacle



PART CHARACTERISTICS

NUMBER OF POSITION	ASSEMBLY ITEM NUMBER
02	43025-0208
04	43025-0408
06	43025-0608
08	43025-0808
10	43025-1008
12	43025-1208
14	43025-1408
16	43025-1608
18	43025-1808
20	43025-2008
22	43025-2208
24	43025-2408

QTY. NO.	DIM. 'A'	DIM. 'B'
2	.275 (7.12)	N/A
4	.270 (6.86)	.118 (3.00)
6	.268 (6.80)	.236 (6.00)
8	.266 (6.73)	.354 (9.00)
10	.264 (6.68)	.472 (12.00)
12	.262 (6.63)	.591 (15.00)
14	.260 (6.58)	.709 (18.00)
16	.258 (6.52)	.827 (21.00)
18	.256 (6.47)	.945 (24.00)
20	.254 (6.42)	1.063 (27.00)
22	.252 (6.37)	1.181 (30.00)
24	.250 (6.32)	1.299 (33.00)

HOUSING SHOWN WITH FIRST CIRCUIT IDENTIFIER RIB (SEE NOTE #10)

RECEPTACLE #43025-****

PLUG #43025-****

MATED MICRO-FIT CONNECTOR

SECTION "A"- "A" WITH TERMINAL (SCALE 10X)

SECTION "B"- "B" (SCALE 10X)

RECEPTACLE ISO VIEW (8 CIRCUIT SHOWN) (SEE NOTE 9 FOR TESTING)

NOTES:

- HOUSING MATERIAL: UNFILLED NYLON, RATED UL 94V-0, COLOR IS BLACK, (LOW HALOGEN)
- FINISH: N/A
- PRODUCT SPECIFICATION: PS-43045
- PACKAGING SPECIFICATION: PK-43025-001
- THIS RECEPTACLE MATES WITH 43020, 43045
- THIS RECEPTACLE TO BE USED WITH MOLEX FEMALE TERMINAL SERIES 43030 OR 46235. SEE SECTION "A"- "A" FOR TERMINAL ORIENTATION IN HOUSING.
- FOR OVERHOLDING PARAMETERS SEE ENGINEERING SPECIFICATION #SDS-43025-1000
- TOP PULL TABS ARE NOT AVAILABLE ON 2 AND 4 CIRCUIT PARTS
- MOLEX RECOMMENDS THE USE OF MICRO-FIT TEST PLUG, SERIES NO. 44242-**** WHENEVER TESTING IS PERFORMED. TEST PLUGS MUST NOT BE USED FOR MAKE OR BREAK UNDER LOAD. MOLEX DOES NOT RECOMMEND USING STANDARD MATING COMPONENTS FOR HARSHNESS TESTING PURPOSES.
- SOME HOUSINGS MAY HAVE A SMALL GATE BLEMISH NEAR THE GATE THAT DOES NOT AFFECT FUNCTIONALITY
- HOUSINGS HAVE EITHER AN IDENTIFIER RIB OR ENGRAVED "F" SYMBOL TO INDICATE CIRCUIT #1. IDENTIFIER TYPE IS TOOL DEPENDENT AND NOT SELECTABLE.
- DIMENSION "A" MEASURED AT DATUM $\perp A$
- THIS PART CONFORMS TO CLASS "B" REQUIREMENTS OF COSMETIC SPECIFICATION PS-45499-002

REVISIONS

REV	DATE	DESCRIPTION
1	07/07/2007	INITIAL RELEASE
2	08/01/2008	REVISION 1

QUALITY SYMBOLS

▽=0 4 PLACES ± .010

▽=0 3 PLACES ± .010

▽=0 2 PLACES ± 0.25

▽=0 1 PLACE ± 0.35

▽=0 0 PLACES ± .010

GENERAL TOLERANCES (UNLESS SPECIFIED)

DIMENSION	TOLERANCE
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010

REVISIONS

REV	DATE	DESCRIPTION
1	07/07/2007	INITIAL RELEASE
2	08/01/2008	REVISION 1

QUALITY SYMBOLS

▽=0 4 PLACES ± .010

▽=0 3 PLACES ± .010

▽=0 2 PLACES ± 0.25

▽=0 1 PLACE ± 0.35

▽=0 0 PLACES ± .010

GENERAL TOLERANCES (UNLESS SPECIFIED)

DIMENSION	TOLERANCE
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010

REVISIONS

REV	DATE	DESCRIPTION
1	07/07/2007	INITIAL RELEASE
2	08/01/2008	REVISION 1

QUALITY SYMBOLS

▽=0 4 PLACES ± .010

▽=0 3 PLACES ± .010

▽=0 2 PLACES ± 0.25

▽=0 1 PLACE ± 0.35

▽=0 0 PLACES ± .010

GENERAL TOLERANCES (UNLESS SPECIFIED)

DIMENSION	TOLERANCE
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010

REVISIONS

REV	DATE	DESCRIPTION
1	07/07/2007	INITIAL RELEASE
2	08/01/2008	REVISION 1

QUALITY SYMBOLS

▽=0 4 PLACES ± .010

▽=0 3 PLACES ± .010

▽=0 2 PLACES ± 0.25

▽=0 1 PLACE ± 0.35

▽=0 0 PLACES ± .010

GENERAL TOLERANCES (UNLESS SPECIFIED)

DIMENSION	TOLERANCE
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010

REVISIONS

REV	DATE	DESCRIPTION
1	07/07/2007	INITIAL RELEASE
2	08/01/2008	REVISION 1

QUALITY SYMBOLS

▽=0 4 PLACES ± .010

▽=0 3 PLACES ± .010

▽=0 2 PLACES ± 0.25

▽=0 1 PLACE ± 0.35

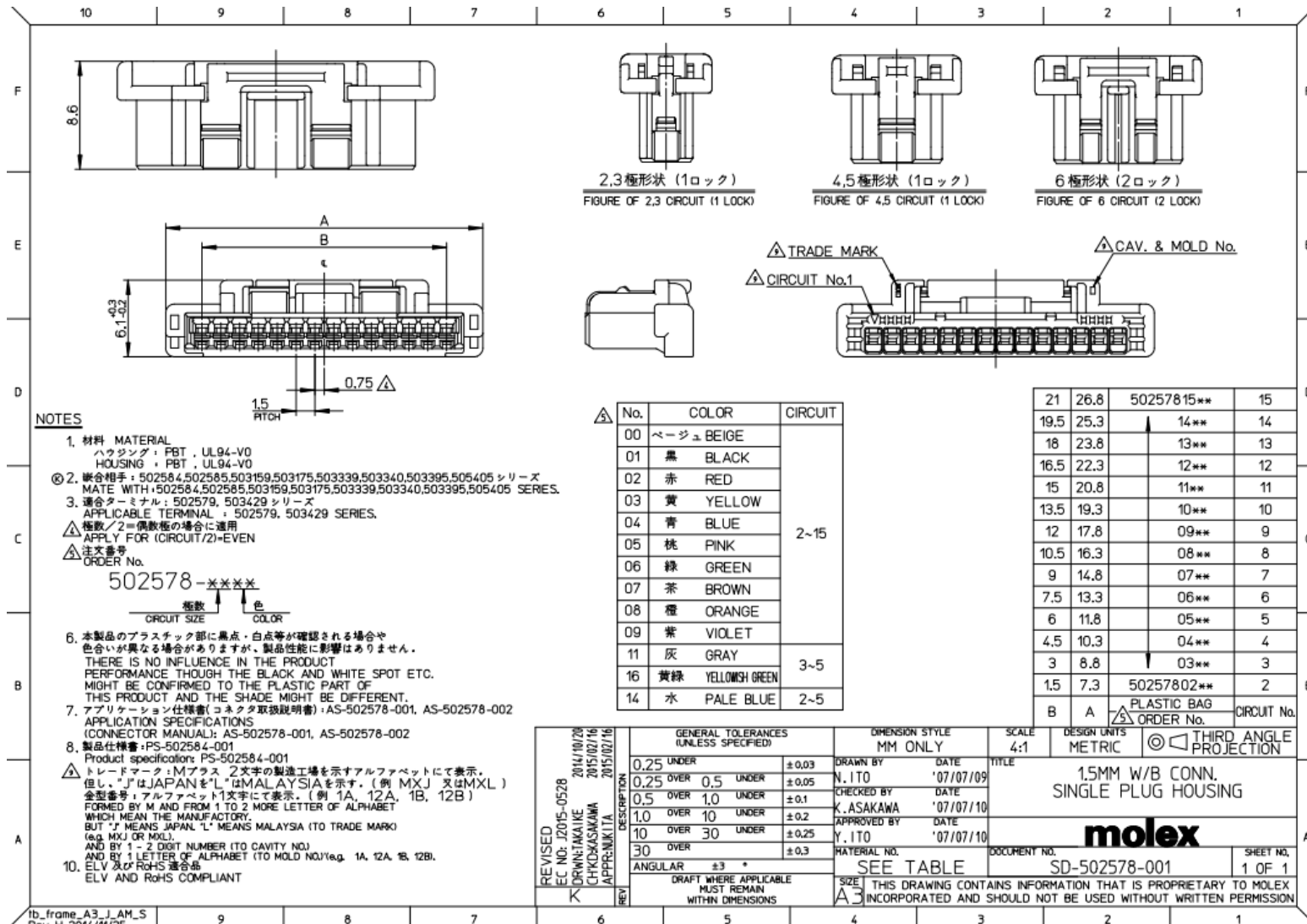
▽=0 0 PLACES ± .010

GENERAL TOLERANCES (UNLESS SPECIFIED)

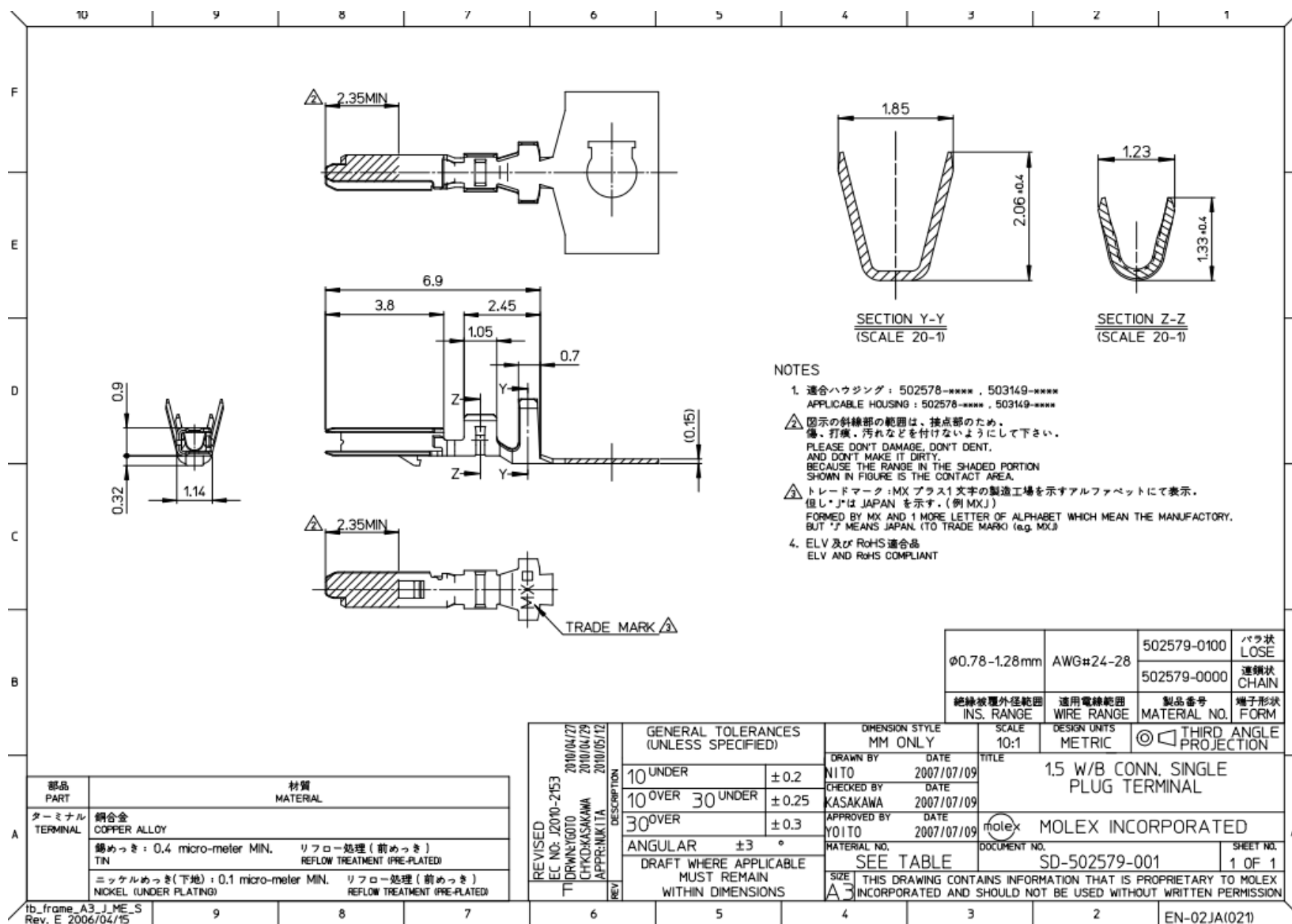
DIMENSION	TOLERANCE
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL DIMENSIONS	± .010
ALL	

	13	12	11	10	9	8	7	6	5	4	3	2	1																																																				
J	<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th>WIRE SIZE</th><th>MAX INSULATION</th><th>"A" ±.012 ±0.30</th><th>"C" ±.012 ±0.30</th><th>"D" ±.005 ±0.13</th><th>"E" ±.012 ±0.30</th><th>"G" ±.005 ±0.13</th><th>"H" ±.012 ±0.30</th></tr> </thead> <tbody> <tr> <td>20-24</td><td>.073/1.85</td><td>.090/2.29</td><td>.100/2.54</td><td>.030/0.76 R.</td><td>.075/1.91</td><td>.025/0.64 R.</td><td>.082/2.08</td></tr> <tr> <td>26-30</td><td>.050/1.27</td><td>.075/1.91</td><td>.083/2.10</td><td>.025/0.64 R.</td><td>.058/1.47</td><td>.020/0.51 R.</td><td>.065/1.65</td></tr> <tr> <td>18 OR 0.75mm²</td><td>.073/1.85</td><td>.091/2.31</td><td>.099/2.51</td><td>.030/0.76 R.</td><td>.086/2.18</td><td>.029/0.74 R.</td><td>.094/2.39</td></tr> </tbody> </table>													WIRE SIZE	MAX INSULATION	"A" ±.012 ±0.30	"C" ±.012 ±0.30	"D" ±.005 ±0.13	"E" ±.012 ±0.30	"G" ±.005 ±0.13	"H" ±.012 ±0.30	20-24	.073/1.85	.090/2.29	.100/2.54	.030/0.76 R.	.075/1.91	.025/0.64 R.	.082/2.08	26-30	.050/1.27	.075/1.91	.083/2.10	.025/0.64 R.	.058/1.47	.020/0.51 R.	.065/1.65	18 OR 0.75mm ²	.073/1.85	.091/2.31	.099/2.51	.030/0.76 R.	.086/2.18	.029/0.74 R.	.094/2.39																				
WIRE SIZE	MAX INSULATION	"A" ±.012 ±0.30	"C" ±.012 ±0.30	"D" ±.005 ±0.13	"E" ±.012 ±0.30	"G" ±.005 ±0.13	"H" ±.012 ±0.30																																																										
20-24	.073/1.85	.090/2.29	.100/2.54	.030/0.76 R.	.075/1.91	.025/0.64 R.	.082/2.08																																																										
26-30	.050/1.27	.075/1.91	.083/2.10	.025/0.64 R.	.058/1.47	.020/0.51 R.	.065/1.65																																																										
18 OR 0.75mm ²	.073/1.85	.091/2.31	.099/2.51	.030/0.76 R.	.086/2.18	.029/0.74 R.	.094/2.39																																																										
I																																																																	
H	<p style="text-align: center;">SECTION V-V SECTION W-W</p>																																																																
G																																																																	
F	<p style="text-align: center;">SEE NOTE 9</p>																																																																
E	<p style="text-align: center;">MINIMUM AREA OF SELECT GOLD PLATING 2 PLACES (INSIDE ONLY) (PLATING B AND C ONLY)</p>																																																																
D	<p style="text-align: center;">SELECT TIN AREA (BOTH SIDES) (PLATING B AND C ONLY)</p>																																																																
C	<p style="text-align: center;">COINED AREA (SCALE 20:1)</p>																																																																
B	<p>NOTES</p> <ol style="list-style-type: none"> MATERIAL: PHOSPHOR BRONZE ALLOY TERMINAL PLATING: "A" - HOT TIN DIP: .000040/0.00102 MIN. OVERALL "B" - .000015/0.00038 MIN. SELECT GOLD .000100/0.00254 MIN. SELECT TIN ALL OVER .000050/0.00127 MIN. NICKEL OVERALL. "C" - .000030/0.00076 MIN. SELECT GOLD .000100/0.00254 MIN. SELECT TIN ALL OVER .000050/0.00127 MIN. NICKEL OVERALL. PLATING FINISHES B AND C ARE POST PLATED. PRODUCT SPECIFICATION: PS-43045, PS-43650, PS-44300-001. PACKAGING SPECIFICATION: PK-43030-001 (REEL), PK-43030-003 (LOOSE). TERMINAL FOR USE IN A MICRO FIT RECEPTACLE #43025-XXXX, 43645-XXXX, AND 44133-XXXX. FOR TERMINAL ORIENTATION IN RECEPTACLE SEE DRAWING #SDA-43025 OR SDA-43645. THIS TERMINAL IS DESIGNED IN METRIC. MOLEX RECOMMENDS THE USE OF MICRO-FIT TEST PLUG (SERIES 44242) WHENEVER CONTINUITY TESTING IS PERFORMED. TEST PLUGS MUST NOT BE USED TO MAKE OR BREAK UNDER LOAD. MOLEX DOES NOT RECOMMEND USING STANDARD MATING COMPONENTS (SERIES 43020, 43045, 43640, 43650, OR 43031) FOR HARNESS TESTING PURPOSES. MARKING MAY VARY ON PARTS. CONSULT FACTORY. PART NUMBER 43030-003B IS DESIGNED FOR 18 AWG 1061 STYLE WIRE OR A .073/1.85 MAXIMUM INSULATION OUTSIDE DIAMETER. 																																																																
A	<table border="1" style="width: 100%; border-collapse: collapse; font-size: small;"> <tr> <td rowspan="4" style="vertical-align: top;"> REMOVE 43030-0024 REC NO: 2007-1028 DRINKS/LUPFER 28/11/15 APPROVED BY 28/11/15 </td> <td>QUALITY SYMBOLS</td> <td colspan="2">GENERAL TOLERANCES (UNLESS SPECIFIED)</td> <td colspan="2">DIMENSION STYLE IN/MM</td> <td>SCALE 10:1</td> <td>DESIGN UNITS METRIC</td> <td>THIRD ANGLE PROJECTION</td> </tr> <tr> <td rowspan="3"> ∇-O ∇-O ∇-O </td> <td>4 PLACES ± .010</td> <td>INCH</td> <td>DRAWN BY</td> <td>DATE</td> <td rowspan="3">TITLE mico-fit (3.0) FEMALE CRIMP TERMINAL</td> <td rowspan="3">molex</td> <td rowspan="3">SHEET NO. 1 OF 1</td> </tr> <tr> <td>3 PLACES ± .010</td> <td>mm</td> <td>CHECKED BY</td> <td>DATE</td> </tr> <tr> <td>2 PLACES ± 0.25</td> <td>mm</td> <td>APPROVED BY</td> <td>DATE</td> </tr> <tr> <td></td> <td>1 PLACE ± 0.35</td> <td>mm</td> <td>MARGULIS</td> <td>2002/08/03</td> <td colspan="2">DOCUMENT NO.</td> <td colspan="2">SD-43030-XXXX</td> </tr> <tr> <td></td> <td>0 PLACE ± .010</td> <td>mm</td> <td colspan="2">ANGULAR ±1/2°</td> <td colspan="2">MATERIAL NO.</td> <td colspan="2">SEE CHART</td> </tr> <tr> <td></td> <td></td> <td colspan="2">DRAFT WHERE APPLICABLE</td> <td colspan="2">SIZE</td> <td colspan="3">THIS DRAWING CONTAINS INFORMATION THAT IS PROPRIETARY TO MOLEX INCORPORATED AND SHOULD NOT BE USED WITHOUT WRITTEN PERMISSION</td> </tr> </table>													REMOVE 43030-0024 REC NO: 2007-1028 DRINKS/LUPFER 28/11/15 APPROVED BY 28/11/15	QUALITY SYMBOLS	GENERAL TOLERANCES (UNLESS SPECIFIED)		DIMENSION STYLE IN/MM		SCALE 10:1	DESIGN UNITS METRIC	THIRD ANGLE PROJECTION	∇-O ∇-O ∇-O	4 PLACES ± .010	INCH	DRAWN BY	DATE	TITLE mico-fit (3.0) FEMALE CRIMP TERMINAL	molex	SHEET NO. 1 OF 1	3 PLACES ± .010	mm	CHECKED BY	DATE	2 PLACES ± 0.25	mm	APPROVED BY	DATE		1 PLACE ± 0.35	mm	MARGULIS	2002/08/03	DOCUMENT NO.		SD-43030-XXXX			0 PLACE ± .010	mm	ANGULAR ±1/2°		MATERIAL NO.		SEE CHART				DRAFT WHERE APPLICABLE		SIZE		THIS DRAWING CONTAINS INFORMATION THAT IS PROPRIETARY TO MOLEX INCORPORATED AND SHOULD NOT BE USED WITHOUT WRITTEN PERMISSION		
REMOVE 43030-0024 REC NO: 2007-1028 DRINKS/LUPFER 28/11/15 APPROVED BY 28/11/15	QUALITY SYMBOLS	GENERAL TOLERANCES (UNLESS SPECIFIED)		DIMENSION STYLE IN/MM		SCALE 10:1	DESIGN UNITS METRIC	THIRD ANGLE PROJECTION																																																									
	∇-O ∇-O ∇-O	4 PLACES ± .010	INCH	DRAWN BY	DATE	TITLE mico-fit (3.0) FEMALE CRIMP TERMINAL	molex	SHEET NO. 1 OF 1																																																									
		3 PLACES ± .010	mm	CHECKED BY	DATE																																																												
		2 PLACES ± 0.25	mm	APPROVED BY	DATE																																																												
	1 PLACE ± 0.35	mm	MARGULIS	2002/08/03	DOCUMENT NO.		SD-43030-XXXX																																																										
	0 PLACE ± .010	mm	ANGULAR ±1/2°		MATERIAL NO.		SEE CHART																																																										
		DRAFT WHERE APPLICABLE		SIZE		THIS DRAWING CONTAINS INFORMATION THAT IS PROPRIETARY TO MOLEX INCORPORATED AND SHOULD NOT BE USED WITHOUT WRITTEN PERMISSION																																																											

455 – Clik-Mate 1.5



456 – Clik-Mate Crimp Pin



1.1.1 Teensy_Firmware.ino

Here they are:

```
#include "uLaren_CAN_Driver.h"
```

```
#include "FlexCAN.h"
```

```
#include "kinetis_flexcan.h"
```

```
#include "input_handler.h"
```

```
#include "output_handler.h"
```

```
#include "loop.h"
```

```
#include "fault_handler.h"
```

```
#include "structs.h"
```

```
#include "Adafruit_VL53L0X.h"
```

```
#define NODE_1 1
```

```
#define NODE_2 2
```

```
#define NODE_3 3
```

```
#define NODE_4 4
```

```
#define MC_VOLTAGE_THRESHOLD 220
```

```
#define SIMULINK 1
```

```
//main globals
```

```
FlexCAN CANbus(1000000);
```

```
State next_state;
```

```

//Laser Sensor

Adafruit_VL53L0X lox = Adafruit_VL53L0X();

VL53L0X_RangingMeasurementData_t measure;


//other data globals

// throttle input value (ranges from ~-500 to 500)

volatile int16_t THR_in;

//steering input value (ranges from ~-500 to 500)

volatile int16_t ST_in;

// URF measured distance, in mm (ranges from ~0-5000)

uint16_t URF_dist;

// IMU structure

extern IMUstruct IMUdat;


// Output variables

long g1 = 0;

long g2 = 0;

long g3 = 0;

imu::Vector<3> gyro1; // Stores the 16-bit signed gyro sensor output

int16_t servo_out = 0;

int16_t throttle_out = 0;

int16_t throttle_out_LF = 0;

int16_t throttle_out_RF = 0;

int16_t throttle_out_LR = 0;

int16_t throttle_out_RR = 0;

```

```

//timing variables

unsigned long start_time_motors;

unsigned long current_time_motors;

unsigned long start_time_servo;

unsigned long current_time_servo;

unsigned long start_time_voltage;

unsigned long current_time_voltage;


//loop variables

int ret = 0;

int error = NO_ERROR;

int voltage = 0;

int data_int = 0;

/*****/

/***** NOTES *****/

/*

* ERROR_CAN_WRITE: a write to can error usually means one controller is offline.

*   In this scenario we will go to INITIALIZE_CONTROLLERS and try to reboot them

*/


void setup() {

    //begin serial port operation

    Serial.begin(115200);

    if (SIMULINK)

```

```
{  
  
    Serial1.begin(115200);  
  
    pinMode(PI_GPIO, OUTPUT);  
  
    delay(10);  
  
    digitalWrite(PI_GPIO, LOW);  
  
}  
  
//startup CAN network  
  
CANbus.begin();  
  
next_state = INITIALIZE_PERIPHERALS;  
  
indicatorSet(rgbWHITE);  
  
}  
  
void loop() {  
  
    CAN_message_t msg;  
  
  
  
    switch(next_state)  
  
    {  
  
        case(INITIALIZE_PERIPHERALS):  
  
            //initialize other things  
  
            if (PRINT)  
  
            {  
  
                Serial.println("Initializing Peripherals");  
  
            }  
  
            initPWMin();  
  
            initServo();  
  

```

```
if (!lox.begin())

{

    //error state

    indicatorSet(rgbPURPLE);

    while(1);

}

next_state = INITIALIZE_CONTROLLERS;

break;

case(INITIALIZE_CONTROLLERS):

    if (PRINT)

    {

        Serial.println("Initializing Controllers");

    }

    ret = reset_nodes();

    if (ret > 0)

    {

        error = ret;

    }

    delay(1000);


    ret = initialize_CAN();

    if (ret > 0)

    {

        error = ret;

    }

}
```

```
delay(50);
```

```
ret = initialize_MC(NODE_1);
```

```
if (ret > 0)
```

```
{
```

```
    error = ret;
```

```
}
```

```
process_available_msgs();
```

```
delay(100);
```

```
ret = initialize_MC(NODE_2);
```

```
if (ret > 0)
```

```
{
```

```
    error = ret;
```

```
}
```

```
process_available_msgs();
```

```
delay(100);
```

```
ret = initialize_MC(NODE_3);
```

```
if (ret > 0)
```

```
{
```

```
    error = ret;
```

```
}
```

```
process_available_msgs();
```

```
delay(100);

ret = initialize_MC(NODE_4);

if (ret > 0)

{

    error = ret;

}


if (error == ERROR_CAN_WRITE)

{

    //reinitialize controllers

    if (PRINT)

    {

        Serial.println("Stopping Nodes.");

    }


    delay(500);

    stop_remote_node(NODE_1);

    stop_remote_node(NODE_2);

    stop_remote_node(NODE_3);

    stop_remote_node(NODE_4);

    delay(500);

    process_available_msgs();

    if (PRINT)

    {

        Serial.println("Reinitializing Controllers.");
```

```
}

    next_state = INITIALIZE_CONTROLLERS;

    error = 0;
}
else
{
    next_state = WAIT_FOR_ARM;

    indicatorSet(rgbYELLOW);
}

break;
case(WAIT_FOR_ARM):
    if (PRINT)
    {
        Serial.println("Awaiting arming sequence... ");
    }

    while (ST_in < 400)
    {
        //do nothing until armed

        if (PRINT)
        {
            Serial.println(ST_in);
        }
    }
}
```

```
    delay(500);  
}
```

```
if (ST_in >= 400)  
{  
    next_state = LINK_COMMUNICATION;  
    indicatorSet(rgbRED);  
}  
else  
{  
    next_state = WAIT_FOR_ARM;  
}
```

```
break;
```

```
case(LINK_COMMUNICATION):
```

```
    //arm  
    link_node(NODE_1);  
    delay(500);  
    link_node(NODE_2);  
    delay(500);  
    link_node(NODE_3);  
    delay(500);  
    link_node(NODE_4);  
    delay(500);
```

```

if (SIMULINK)
{
    next_state = RUNNING_SIMULINK;

    //write simulink pin high (pin 15) (PI_GPIO)

    digitalWrite(PI_GPIO, HIGH);

    indicatorSet(rgbCYAN);
}

else{

    next_state = RUNNING_NOMINALLY;

    indicatorSet(rgbGREEN);
}

start_time_motors = micros();

start_time_servo = micros();

start_time_voltage = millis();

if (PRINT)

{

    Serial.println("Running under normal operations.");

}

break;

case(RUNNING_NOMINALLY):

    //check for messages

    while (CANbus.available())

    {

        if (CANbus.read(msg) != 0)

```

```

{
    //Serial.println("ello matey");
    print_incoming_CAN_message(msg);
}
}

//write to motor controllers

current_time_motors = micros();

if ((current_time_motors - start_time_motors) >= 20000) //20ms => 50hz
{
    start_time_motors = micros();

    write_velocity_and_enable_MC(NODE_1, -THR_in * SCALE_FACTOR);
    write_velocity_and_enable_MC(NODE_2, THR_in * SCALE_FACTOR);
    write_velocity_and_enable_MC(NODE_3, THR_in * SCALE_FACTOR);
    write_velocity_and_enable_MC(NODE_4, -THR_in * SCALE_FACTOR);

}

//check voltage level

current_time_voltage = millis();

if ((current_time_voltage - start_time_voltage) >= 800)
{
    voltage = query_voltage_level(NODE_1);

    start_time_voltage = millis();
}

```

```

    if (voltage < MC_VOLTAGE_THRESHOLD)
    {
        Serial.println("Voltage to motors is below our threshold. System is shutting down. Charge your
battery!");

        //shutdown drivers

        shutdown_MC(NODE_1);

        shutdown_MC(NODE_2);

        shutdown_MC(NODE_3);

        shutdown_MC(NODE_4);


        //make LED purple

        indicatorSet(rgbPURPLE);

        while(1);
    }

    indicatorSet(rgbGREEN);
}

```

```

// Push Actuator Data-----

```

```

// Set steering angle

```

```

current_time_servo = micros();

```

```

if ((current_time_servo - start_time_servo) >= 10000)

```

```

{

```

```

    start_time_servo = micros();

```

```

    writeServo(ST_in);

```

```
}
```

```
break;
```

```
case(RUNNING_SIMULINK):
```

```
    //check for CAN messages
```

```
    while (CANbus.available())
```

```
    {
```

```
        if (CANbus.read(msg) != 0)
```

```
        {
```

```
            //Serial.println("ello matey");
```

```
            print_incoming_CAN_message(msg);
```

```
        }
```

```
    }
```

```
    //write to motor controllers if we need to (50Hz)
```

```
    current_time_motors = micros();
```

```
    if ((current_time_motors - start_time_motors) >= 20000) //20ms => 50hz
```

```
    {
```

```
        start_time_motors = micros();
```

```
        lox.rangingTest(&measure, false);
```

```
        write_velocity_and_enable_MC(NODE_1, -throttle_out_RF * SCALE_FACTOR);
```

```
        write_velocity_and_enable_MC(NODE_2, throttle_out_LF * SCALE_FACTOR);
```

```
        write_velocity_and_enable_MC(NODE_3, throttle_out_LR * SCALE_FACTOR);
```

```
write_velocity_and_enable_MC(NODE_4, -throttle_out_RR * SCALE_FACTOR);  
  
Serial.println("I wrote to motors!");  
  
}
```

```
//process serial data (simulink) (simulink running at appx. 50Hz)
```

```
while (Serial1.available() > 0)  
{  
  
  data_int = Serial1.read();  
  
  Serial.println(data_int);  
  
  switch(data_int)  
  {  
  
    case 11: //servo read from pi  
      while (!Serial1.available())  
      {  
  
        ;  
  
      }  
  
      servo_out = Serial1.read();  
  
      while (!Serial1.available())  
  
      {  
  
        ;  
  
      }  
  
      servo_out |= (Serial1.read() << 8);  
  
      break;  
  
    case 12: //throttle RF read from pi  
  
      while (!Serial1.available())
```

```

{
    ;
}

throttle_out_RF = Serial1.read();

while (!Serial1.available())

{
    ;
}

throttle_out_RF |= (Serial1.read() << 8);

break;

case 13: //throttle LF read from pi

while (!Serial1.available())

{
    ;
}

throttle_out_LF = Serial1.read();

while (!Serial1.available())

{
    ;
}

throttle_out_LF |= (Serial1.read() << 8);

break;

case 14: //throttle LR read from pi

while (!Serial1.available())

{

```

```

;
}

throttle_out_LR = Serial1.read();

while (!Serial1.available())

{

;

}

throttle_out_LR |= (Serial1.read() << 8);

break;

case 15: //throttle RR read from pi

while (!Serial1.available())

{

;

}

throttle_out_RR = Serial1.read();

while (!Serial1.available())

{

;

}

throttle_out_RR |= (Serial1.read() << 8);

break;

case 21: //servo write to pi

if (PRINT)

{

Serial.println(ST_in);

```

```

}

Serial1.write((const uint8_t*)&ST_in, 2);

break;

case 22: //URF write to pi

Serial1.write((const uint8_t*)&URF_dist, 2);

break;

case 23: //gyroX write to pi

Serial1.write((const uint8_t*)&g2, 2);

break;

case 24: //gyroY write to pi

Serial1.write((const uint8_t*)&g3, 2);

break;

case 25: //gyroZ write to pi

Serial1.write((const uint8_t*)&g1, 2);

break;

case 26: //throttle write to pi

if (PRINT)

{

Serial.println(THR_in);

}

Serial1.write((const uint8_t*)&THR_in, 2);

break;

case 27: //laser write to pi

Serial1.write((const uint8_t*)&measure.RangeMilliMeter, 2);

break;

```

```

default:

    Serial.print("In Default case. Should NOT BE HERE. **");

    Serial.println(data_int);

    break;

}

}

//check voltage level (1Hz)

current_time_voltage = millis();

if ((current_time_voltage - start_time_voltage) >= 1000)

{

    voltage = query_voltage_level(NODE_1);

    start_time_voltage = millis();

    if (voltage < MC_VOLTAGE_THRESHOLD)

    {

        Serial.println("Voltage to motors is below our threshold. System is shutting down. Charge your
battery!");

        //shutdown drivers

        shutdown_MC(NODE_1);

        shutdown_MC(NODE_2);

        shutdown_MC(NODE_3);

        shutdown_MC(NODE_4);

        //make LED purple

        indicatorSet(rgbPURPLE);

```

```

    }

    indicatorSet(rgbCYAN);
}

// Push Actuator Data-----

// Set steering angle (100Hz)

current_time_servo = micros();

if ((current_time_servo - start_time_servo) >= 10000)
{
    start_time_servo = micros();

    writeServo(servo_out);
}

break;

case(INDICATE_AND_LOG_ERROR):

    if (PRINT)
    {
        Serial.println("Indicating and logging error.");
    }

    next_state = WAIT_FOR_CLEAR;

    break;

case(WAIT_FOR_CLEAR):

    delay(1000);

```

```

        next_state = WAIT_FOR_ARM;

        break;

default:

    if (PRINT)

    {

        Serial.println("In default case. Should not be here. Ever.");

        Serial.print("Current state is: ");

        Serial.println(next_state);

    }

    exit(0);

    break;

}

}

```

1.1.2 ULaren_CAN_Driver.cpp

Below:

```
/* CAN Firmware Functions made for Teensy 3.6
```

```
by the uLaren Senior Project Team */
```

```
/******
```

```
/* current functionality works but re-initialization does not work*/
```

```
*****
```

```
#include "flexCAN.h"
```

```
#include "uLaren_CAN_Driver.h"
```

```
#include <string.h>
```

```
#define WAIT_FOR_RESPONSE_TIME_SLOW_US 100
```

```
#define WAIT_FOR_RESPONSE_TIME_FAST_US 50
```

```
#define NODE_1 1
```

```
#define NODE_2 2
```

```
#define NODE_3 3
```

```
#define NODE_4 4
```

```
extern FlexCAN CANbus;
```

```
int initialize_CAN()
```

```
{
```

```
    CAN_message_t msg;
```

```
    int ret = 0;
```

```
    //"Start Remote Node"
```

```
    msg.id = 0;
```

```
    msg.ext = 0;
```

```
    msg.len = 2;
```

```
    msg.timeout = 0;
```

```
    msg.buf[0] = 0x01;
```

```
    msg.buf[1] = 0;
```

```
        if (CANbus.write(msg) == 0)
    {
        ret = ERROR_CAN_WRITE;
    }

    //print_outgoing_CAN_message(msg);

    delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);

    if (CANbus.read(msg) != 0)
    {
        if (PRINT)
        {
            Serial.println("Received Start Node Confirmation");
        }

    }

    else
    {
        if (PRINT)
        {
            Serial.println("DID NOT Receive Start Node Confirmation");
        }

    }
}
```

```
return ret;
```

```
}
```

```
int resetFault (int node_id)
```

```
{
```

```
    CAN_message_t msg;
```

```
    int ret = 0;
```

```
    //make sure node_id is valid!
```

```
    if (node_id > 0 && node_id < 127)
```

```
    {
```

```
    }
```

```
    else
```

```
    {
```

```
        if (PRINT)
```

```
        {
```

```
            Serial.print("***** I received a bad node_id of: ");
```

```
            Serial.println(node_id);
```

```
        }
```

```
        exit(0);
```

```
    }
```

```
//tell MC's to reset faults

msg.id = 0x600 + node_id;

msg.ext = 0;

msg.len = 6;

msg.timeout = 0;

//

msg.buf[0] = 0x2B; //68

msg.buf[2] = 0x60; //60

msg.buf[1] = 0x40;

msg.buf[3] = 0x00;

//

msg.buf[5] = 0x00;

msg.buf[4] = 0b10000000;

msg.buf[6] = 0;

msg.buf[7] = 0;


if (CANbus.write(msg) == 0)

{

    ret = ERROR_CAN_WRITE;

    if (PRINT)

    {

        Serial.println("ERROR: CAN Write.");

        Serial.print("Node id: ");

        Serial.println(node_id);

    }

}
```

```
}  
  
else {  
  
    //print_outgoing_CAN_message(msg);  
  
    delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);  
  
    if (CANbus.read(msg) != 0)  
  
    {  
  
        if (PRINT)  
  
        {  
  
            Serial.println("Received Shutdown Confirmation");  
  
        }  
  
    }  
  
    else  
  
    {  
  
        if (PRINT)  
  
        {  
  
            Serial.println("DID NOT Receive Shutdown Confirmation");  
  
        }  
  
    }  
  
}  
  
//tell MC to reset communication  
  
msg.id = 0x00;  
  
msg.ext = 0;
```

```
msg.len = 2;

msg.timeout = 0;

//

msg.buf[0] = 0x01; //68

msg.buf[2] = 0x00 + node_id; //60

msg.buf[1] = 0x00;

msg.buf[3] = 0x00;

//

msg.buf[5] = 0x00;

msg.buf[4] = 0x00;

msg.buf[6] = 0;

msg.buf[7] = 0;


if (CANbus.write(msg) == 0)

{

    ret = ERROR_CAN_WRITE;

    if (PRINT)

    {

        Serial.println("ERROR: CAN Write.");

        Serial.print("Node id: ");

        Serial.println(node_id);

    }

}

else {
```

```

//print_outgoing_CAN_message(msg);

delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);

if (CANbus.read(msg) != 0)

{

    if (PRINT)

    {

        Serial.println("Received Shutdown Confirmation");

    }

}

else

{

    if (PRINT)

    {

        Serial.println("DID NOT Receive Shutdown Confirmation");

    }

}

}

}

}

int initialize_MC(int node_id)

{

    CAN_message_t msg;

    int ret = 0;

```

```
//make sure node_id is valid!

if (node_id > 0 && node_id < 127)
{

}

else
{
    if (PRINT)
    {
        Serial.print("***** I received a bad node_id of: ");
        Serial.println(node_id);
    }

    exit(0);
}
```

```
//tell MC's to go to shutdown state
```

```
msg.id = 0x600 + node_id;
```

```
msg.ext = 0;
```

```
msg.len = 6;
```

```
msg.timeout = 0;
```

```
//
```

```
msg.buf[0] = 0x2B;
```

```
msg.buf[2] = 0x60;
```

```

        msg.buf[1] = 0x40;

        msg.buf[3] = 0x00;

        //

        msg.buf[5] = 0x00;

        msg.buf[4] = 0b000000110;
msg.buf[6] = 0;
msg.buf[7] = 0;


        if (CANbus.write(msg) == 0)
{
    ret = ERROR_CAN_WRITE;

    if (PRINT)
    {
        Serial.println("ERROR: CAN Write.");

        Serial.print("Node id: ");

        Serial.println(node_id);
    }

}

else {

    //print_outgoing_CAN_message(msg);

    delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);

    if (CANbus.read(msg) != 0)

    {

        if (PRINT)

```

```
{  
    Serial.println("Received Shutdown Confirmation");  
}  
  
}  
else  
{  
    if (PRINT)  
    {  
        Serial.println("DID NOT Receive Shutdown Confirmation");  
    }  
  
}  
}  
  
    //initialize MC's to profile velocity mode  
msg.id = 0x600 + node_id;  
    msg.len = 5;  
msg.timeout = 0;  
    msg.buf[0] = 0x2F;  
    msg.buf[2] = 0x60;  
    msg.buf[1] = 0x60;  
    msg.buf[3] = 0;  
    msg.buf[4] = PROFILE_VELOCITY_MODE;  
  
    if (CANbus.write(msg) == 0)
```

```
{  
  
    ret = ERROR_CAN_WRITE;  
  
    if (PRINT)  
    {  
  
        Serial.println("ERROR: CAN Write.");  
  
        Serial.print("Node id: ");  
  
        Serial.println(node_id);  
    }  
  
}  
  
else {  
  
    //print_outgoing_CAN_message(msg);  
  
    delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);  
  
    if (CANbus.read(msg) != 0)  
    {  
  
        if (PRINT)  
        {  
  
            Serial.println("Received Profile Velocity Confirmation");  
        }  
  
    }  
  
    else  
    {  
  
        if (PRINT)  
        {  
  

```

```
Serial.println("DID NOT Receive Profile Velocity Confirmation");  
  
}  
  
}  
  
}
```

```
    //initialize MCs motion profile type : linear  
msg.id = 0x600 + node_id;  
  
    msg.len = 6;  
  
    msg.buf[0] = 0x2B;  
  
    msg.buf[2] = 0x60;  
  
    msg.buf[1] = 0x86;  
  
    msg.buf[3] = 0;  
  
    msg.buf[4] = 0;  
  
    msg.buf[5] = 0;  
  
    if (CANbus.write(msg) == 0)  
{  
    ret = ERROR_CAN_WRITE;  
  
    if (PRINT)  
    {  
        Serial.println("ERROR: CAN Write.");  
  
        Serial.print("Node id: ");  
  
        Serial.println(node_id);  
    }  
}
```

```

}

else {

    //print_outgoing_CAN_message(msg);

    delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);

    if (CANbus.read(msg) != 0)

    {

        if (PRINT)

        {

            Serial.println("Received Motion Profile Confirmation");

            //print_incoming_CAN_message(msg);

        }

    }

    else

    {

        if (PRINT)

        {

            Serial.println("DID NOT Receive Motion Profile Confirmation");

        }

    }

}

```

```

/*write max profile velocity

```

```
msg.len = 8;

msg.buf[0] = 0x68;

msg.buf[1] = 0x60;

msg.buf[2] = 0x7F;

msg.buf[3] = 0;

//velocity goes in here. refer to 6.2.81 & 2-15

msg.buf[4] = 0;

msg.buf[5] = 0;

msg.buf[6] = 0;

msg.buf[7] = 0;


write_message_to_all_MCs(msg);

*/
```

```
//tell MC's to go to switch-on state
```

```
msg.id = 0x600 + node_id;
```

```
msg.ext = 0;
```

```
msg.len = 6;
```

```
msg.timeout = 0;
```

```
//
```

```
msg.buf[0] = 0x2B;
```

```
msg.buf[2] = 0x60;
```

```
msg.buf[1] = 0x40;
```

```
msg.buf[3] = 0x00;
```

```
//
```

```
msg.buf[5] = 0x00;

msg.buf[4] = 0b00000111;


if (CANbus.write(msg) == 0)
{
    ret = ERROR_CAN_WRITE;

    if (PRINT)
    {
        Serial.println("ERROR: CAN Write.");

        Serial.print("Node id: ");

        Serial.println(node_id);
    }

}

else {

    //print_outgoing_CAN_message(msg);

    delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);

    if (CANbus.read(msg) != 0)
    {
        if (PRINT)
        {
            Serial.println("Received Switch ON Confirmation");
        }

    }

}
```

```
else
{
    if (PRINT)
    {
        Serial.println("DID NOT Receive Switch ON Confirmation");
    }

}

}

return ret;
}
```

```
int arm_MC(int node_id)
{
    CAN_message_t msg;

    int ret = 0;

    //make sure node_id is valid!
    if (node_id > 0 && node_id < 127)
    {

    }

    else
    {

        if (PRINT)
```

```
{  
    Serial.print("***** I received a bad node_id of: ");  
    Serial.println(node_id);  
}
```

```
exit(0);
```

```
}
```

```
//tell MC's to go to operation enabled state
```

```
msg.id = 0x600 + node_id;
```

```
msg.ext = 0;
```

```
msg.len = 6;
```

```
msg.timeout = 0;
```

```
//
```

```
msg.buf[0] = 0x2B;
```

```
msg.buf[2] = 0x60;
```

```
msg.buf[1] = 0x40;
```

```
msg.buf[3] = 0x00;
```

```
//
```

```
msg.buf[5] = 0x01;
```

```
msg.buf[4] = 0b00001111;
```

```
if (CANbus.write(msg) == 0)
```

```
{
```

```
    ret = ERROR_CAN_WRITE;
```

```

}

/*else {

    //print_outgoing_CAN_message(msg);

    delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);

    if (CANbus.read(msg) != 0)

    {

        Serial.println("Received Operation Enabled Confirmation");

        //print_incoming_CAN_message(msg);

    }

    else

    {

        Serial.println("DID NOT Receive Operation Enabled Confirmation");

    }

}*/

return ret;

}

```

```

int write_throttle_to_MC(int node_id, int throttle)

{

    CAN_message_t msg;

    int ret = 0;

    //make sure node_id is valid!

    if (node_id > 0 && node_id < 127)

```

```
{

}

else

{

    if (PRINT)

    {

        Serial.print("***** I received a bad node_id of: ");

        Serial.println(node_id);

    }

    exit(0);

}

//make sure throttle does not exceed limits

if (throttle > 500)

{

    throttle = 500;

}

else if (throttle < -500)

{

    throttle = -500;

}

else if (throttle < 10 && throttle > -10)

{
```

```

    throttle = 0;
}

//write to Target Velocity

msg.id = 0x600 + node_id;

msg.ext = 0;

msg.len = 8;

msg.timeout = 0;

msg.buf[0] = 0x23;

msg.buf[2] = 0x60;

msg.buf[1] = 0xFF;

msg.buf[3] = 0;

//

memcpy(&(msg.buf[4]), (void *)&throttle, 1);

memcpy(&(msg.buf[5]), ((char *)&throttle + 1), 1);

memcpy(&(msg.buf[6]), ((char *)&throttle + 2), 1);

memcpy(&(msg.buf[7]), ((char *)&throttle + 3), 1);


//write message and receive

if (CANbus.write(msg) == 0)

{

    if (PRINT)

    {

        Serial.println("error writing CAN message");

    }

}

```

```

    ret = ERROR_CAN_WRITE;

}

//it will stop responding if a can write occurs but i want to test this out
else
{
    if (PRINT)
    {
        //Serial.print("Awaiting throttle write confirmation... ");
    }
    while (CANbus.read(msg) == 0)
    {
        //wait for handshake

    }

    if (PRINT)
    {
        //Serial.println("Received confirmation");
    }
}

return ret;
}

```

```

int initiate_target_velocity (int node_id)

```

```

{
    CAN_message_t msg;

    int ret = 0;


    //initiate target velocity

    //tell MC's to go to operation enabled state w/target velocity enabled

    msg.id = 0x600 + node_id;

    msg.ext = 0;

    msg.len = 6;

    msg.timeout = 0;

    //

    msg.buf[0] = 0x2B;

    msg.buf[2] = 0x60;

    msg.buf[1] = 0x40;

    msg.buf[3] = 0x00;

    //

    msg.buf[5] = 0b00000000;

    msg.buf[4] = 0b00001111;

    msg.buf[6] = 0;

    msg.buf[7] = 0;


    //write message

    if (CANbus.write(msg) == 0)

    {

        ret = ERROR_CAN_WRITE;
    }
}

```

```

}

else

{

//it will stop responding but i want to test

//print_outgoing_CAN_message(msg);

delayMicroseconds(WAIT_FOR_RESPONSE_TIME_FAST_US);

if (PRINT)

{

//Serial.print("Awaiting initiate target velocity confirmation... ");

}

while (CANbus.read(msg) == 0)

{

//wait for handshake

}

if (PRINT)

{

//Serial.println("Received confirmation");

}

}

return ret;

}

```

```

int write_velocity_and_enable_MC(int node_id, int throttle)

```

```

{

    CAN_message_t msg;

    int ret = 0;

    int error = 0;


    //make sure node_id is valid!

    if (node_id > 0 && node_id < 127)
    {


    }

    else
    {

        if (PRINT)
        {

            Serial.print("***** I received a bad node_id of: ");

            Serial.println(node_id);

        }


        exit(0);

    }


    //make sure throttle does not exceed limits

    if (throttle > (500 * SCALE_FACTOR + 200))
    {

```

```

    throttle = 500 * SCALE_FACTOR;

}

else if (throttle < (-500 * SCALE_FACTOR - 200))

{

    throttle = -500 * SCALE_FACTOR;

}


//initiate throttle dead zone

if (throttle < (10 * SCALE_FACTOR) && throttle > (-10 * SCALE_FACTOR))

{

    throttle = 0;

}


//initiate target velocity

//tell MC's to go to operation enabled state w/target velocity enabled

msg.id = 0x500 + node_id;

msg.ext = 0;

msg.len = 6;

msg.timeout = 0;

//

msg.buf[0] = 0x0F;

msg.buf[1] = 0x00;

memcpy(&(msg.buf[2]), (void *)&throttle, 1);

memcpy(&(msg.buf[3]), ((char *)&throttle) + 1, 1);

memcpy(&(msg.buf[4]), ((char *)&throttle) + 2, 1);

```

```
memcpy(&(msg.buf[5]), ((char *)&throttle) + 3, 1);
```

```
msg.buf[6] = 0;
```

```
msg.buf[7] = 0;
```

```
if (PRINT)
```

```
{
```

```
    print_outgoing_CAN_message(msg);
```

```
}
```

```
ret = CANbus.write(msg);
```

```
if (ret == 1)
```

```
{
```

```
    error = NO_ERROR;
```

```
}
```

```
else {
```

```
    error = ERROR_CAN_WRITE;
```

```
}
```

```
return 0;
```

```
}
```

```
void print_outgoing_CAN_message(CAN_message_t msg)
```

```
{
```

```
    //if (PRINT)
```

```
    //{
```

```
Serial.println("***Outgoing CAN Message***");

Serial.print("ID: ");

Serial.println(msg.id, HEX);

Serial.print("Length: ");

Serial.println(msg.len);

Serial.print("Byte[0]: ");

Serial.println(msg.buf[0], HEX);

Serial.print("Byte[1]: ");

Serial.println(msg.buf[1], HEX);

Serial.print("Byte[2]: ");

Serial.println(msg.buf[2], HEX);

Serial.print("Byte[3]: ");

Serial.println(msg.buf[3], HEX);

Serial.print("Byte[4]: ");

Serial.println(msg.buf[4], HEX);

Serial.print("Byte[5]: ");

Serial.println(msg.buf[5], HEX);

Serial.print("Byte[6]: ");

Serial.println(msg.buf[6], HEX);

Serial.print("Byte[7]: ");

Serial.println(msg.buf[7], HEX);

//}

}
```

```
void print_incoming_CAN_message(CAN_message_t msg)
{
    //if (PRINT)
    //{
        Serial.println("***Incoming CAN Message***");
        Serial.print("ID: ");
        Serial.println(msg.id, HEX);
        Serial.print("Length: ");
        Serial.println(msg.len);
        Serial.print("Byte[0]: ");
        Serial.println(msg.buf[0], HEX);
        Serial.print("Byte[1]: ");
        Serial.println(msg.buf[1], HEX);
        Serial.print("Byte[2]: ");
        Serial.println(msg.buf[2], HEX);
        Serial.print("Byte[3]: ");
        Serial.println(msg.buf[3], HEX);
        Serial.print("Byte[4]: ");
        Serial.println(msg.buf[4], HEX);
        Serial.print("Byte[5]: ");
        Serial.println(msg.buf[5], HEX);
        Serial.print("Byte[6]: ");
        Serial.println(msg.buf[6], HEX);
        Serial.print("Byte[7]: ");
        Serial.println(msg.buf[7], HEX);
    }
```

```
//}
```

```
}
```

```
int send_statusword_request(int node_id)
```

```
{
```

```
    CAN_message_t msg;
```

```
    int ret = 0;
```

```
    msg.id = 0x600 + node_id;
```

```
    msg.ext = 0;
```

```
    msg.len = 4;
```

```
    msg.timeout = 0;
```

```
    //
```

```
    msg.buf[0] = 0x40;
```

```
    msg.buf[2] = 0x60;
```

```
    msg.buf[1] = 0x41;
```

```
    msg.buf[3] = 0x00;
```

```
    //
```

```
    msg.buf[4] = 0x00;
```

```
    msg.buf[5] = 0;
```

```
    msg.buf[6] = 0;
```

```
    msg.buf[7] = 0;
```

```
    if (CANbus.write(msg) == 0)
```

```
{  
  
    if (PRINT)  
  
    {  
  
        Serial.println("error writing CAN message");  
  
    }  
  
  
    exit(0);  
}  
  
  
print_outgoing_CAN_message(msg);  
delayMicroseconds(WAIT_FOR_RESPONSE_TIME_FAST_US);  
/*if (CANbus.read(msg) != 0)  
{  
  
    if (PRINT)  
  
    {  
  
        Serial.println("Received Statusword");  
  
    }  
  
  
    print_incoming_CAN_message(msg);  
  
    ret = 1;  
}  
else  
  
{  
  
    if (PRINT)  
  
    {
```

```
        Serial.println("DID NOT Receive Statusword");
    }

    ret = 0;
}*/

return ret;
}

void check_available_msg()
{
    CAN_message_t msg;

    if (CANbus.available())
    {
        if (PRINT)
        {
            Serial.println("I have something to receive");
        }

        if (CANbus.read(msg) != 0)
        {
            if (PRINT)
            {
                Serial.println("I read a message!");
            }
        }
    }
}
```

```
}
```

```
    print_incoming_CAN_message(msg);
```

```
}
```

```
}
```

```
else
```

```
{
```

```
    if (PRINT)
```

```
    {
```

```
        Serial.println("nothing");
```

```
    }
```

```
}
```

```
}
```

```
int reset_nodes()
```

```
{
```

```
    CAN_message_t msg;
```

```
    int ret = 0;
```

```
    //reset node coms
```

```
    msg.id = 0;
```

```
    msg.ext = 0;
```

```
    msg.len = 2;
```

```
    msg.timeout = 0;
```

```
msg.buf[0] = 0x82;

msg.buf[1] = 0;

msg.buf[2] = 0;

msg.buf[3] = 0;

msg.buf[4] = 0;

msg.buf[5] = 0;

msg.buf[6] = 0;

msg.buf[7] = 0;


if (CANbus.write(msg) == 0)
{
    ret = ERROR_CAN_WRITE;

    if (PRINT)
    {
        Serial.println("error writing CAN message");
    }

}

//print_outgoing_CAN_message(msg);

delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);

if (CANbus.read(msg) != 0)
{
    if (PRINT)
    {
        Serial.println("Received Reset Node Confirmation");
    }
}
```

```
}
```

```
}
```

```
else
```

```
{
```

```
    if (PRINT)
```

```
    {
```

```
        Serial.println("DID NOT Receive Reset Node Confirmation");
```

```
    }
```

```
}
```

```
return ret;
```

```
}
```

```
void process_available_msgs()
```

```
{
```

```
    CAN_message_t msg;
```

```
    delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);
```

```
    while (CANbus.available())
```

```
    {
```

```
        if (CANbus.read(msg) != 0)
```

```
        {
```

```
    print_incoming_CAN_message(msg);  
}  
  
else  
{  
    if (PRINT)  
    {  
        Serial.println("***OOPS the CAN read went wrong!***");  
    }  
  
}  
}  
}
```

```
int stop_remote_node(int node_id)
```

```
{  
    CAN_message_t msg;  
    int ret = 0;  
  
    //reset node coms  
    msg.id = 0;  
    msg.ext = 0;  
    msg.len = 2;  
    msg.timeout = 0;  
    msg.buf[0] = 0x02;  
    msg.buf[1] = 0;
```

```
msg.buf[2] = 0;

msg.buf[3] = 0;

msg.buf[4] = 0;

msg.buf[5] = 0;

msg.buf[6] = 0;

msg.buf[7] = 0;


ret = CANbus.write(msg);

//print_outgoing_CAN_message(msg);

delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);

if (CANbus.read(msg) != 0)

{

    if (PRINT)

    {

        Serial.println("Received Stop Node Confirmation");

    }

}

else

{

    if (PRINT)

    {

        Serial.println("DID NOT Receive Stop Node Confirmation");

    }

}
```

```
}
```

```
return ret;
```

```
}
```

```
int diagnose_controller_message(CAN_message_t msg)
```

```
{
```

```
    int message_status = 0;
```

```
    if (msg.id > 580 && msg.id < 680)
```

```
    {
```

```
        //this is a confirmation message
```

```
        if (msg.id == 581)
```

```
        {
```

```
            if (PRINT)
```

```
            {
```

```
                Serial.print("Node 1:");
```

```
            }
```

```
        }
```

```
    }
```

```
    //even actually do this sometime if it will help
```

```
    return message_status;
```

```
}
```

```
int link_node(int node_id)

{

    CAN_message_t msg;

    int ret = 0;

    int armed = 0;

    int error = 0;


    //make sure node_id is valid!

    if (node_id > 0 && node_id < 127)

    {


    }

    else

    {

        if (PRINT)

        {

            Serial.print("***** I received a bad node_id of: ");

            Serial.println(node_id);

        }


        exit(0);

    }


    //try to clear out can read buffer (is there a read buffer?)

    delay(200);
```

```
while (CANbus.available())

{

    CANbus.read(msg);

}

/*****/

//arm

if (PRINT)

{

    Serial.println("*****ARMING*****");

    Serial.print("On node: ");

    Serial.println(node_id);

}


arm_MC(node_id);

delay(1000);

send_statusword_request(node_id);

delay(200);

while (CANbus.available())

{

    CANbus.read(msg);

    if (PRINT)

    {

        print_incoming_CAN_message(msg);

    }

}
```

```
//previously searched for a 1A0 node id but this isn't always the node id sent when the controller gets armed
```

```
if (msg.buf[4] == 55 && msg.buf[5] == 22) //37 and 16 in hex
```

```
{
```

```
    //were armed. lets go
```

```
    armed = 1;
```

```
}
```

```
}
```

```
//attempt to rearm node
```

```
while (armed == 0)
```

```
{
```

```
    if (PRINT)
```

```
    {
```

```
        Serial.println("*****ATTEMPTING A RE-ARM*****");
```

```
        Serial.print("On node: ");
```

```
        Serial.println(node_id);
```

```
    }
```

```
    armed = rearm_MC(node_id);
```

```
}
```

```
//write
```

```
if (PRINT)
{
    Serial.println("*****WRITING VELOCITY and INITIATING*****");
}
```

```
write_velocity_and_enable_MC(node_id,0);
```

```
delay(50);
```

```
while (CANbus.available())
```

```
{
    CANbus.read(msg);
    //print_incoming_CAN_message(msg);
}
```

```
/*
```

```
//arm
```

```
Serial.println("*****ARMING*****");
```

```
arm_MC(NODE_1);
```

```
delay(2000);
```

```
while (CANbus.available())
```

```
{
    CANbus.read(msg);
    print_incoming_CAN_message(msg);
}
```

```
//write
```

```
Serial.println("*****WRITING VELOCITY*****");
```

```
write_throttle_to_MC(NODE_1,0);

delay(1000);

while (CANbus.available())

{

    CANbus.read(msg);

    print_incoming_CAN_message(msg);

}


//initiate

Serial.println("*****INITIATING*****");

initiate_target_velocity(NODE_1);

delay(1000);

while (CANbus.available())

{

    CANbus.read(msg);

    print_incoming_CAN_message(msg);

}

//stall

while (1)

{

    if (CANbus.available())

    {

        CANbus.read(msg);

        print_incoming_CAN_message(msg);

    }

}
```

```

    //stall here so i can figure out what messages i should be expecting

    */

    return ret;
}

int rearm_MC(int node_id)
{
    CAN_message_t msg;

    int ret = 0;

    int error = 0;

    /*
    ret = initialize_MC(node_id);

    if (ret > 0)
    {
        error = ret;
    }

    ret = arm_MC(node_id);

    if (ret > 0)
    {
        error = ret;
    }
    */

    delay(200);

```

```
resetFault(node_id);  
  
delay(200);  
  
initialize_MC(node_id);  
  
delay(200);  
  
while (CANbus.available())  
{  
  
    CANbus.read(msg);  
  
}  
  
arm_MC(node_id);  
  
delay(500);  
  
send_statusword_request(node_id);  
  
delay(200);
```

```
while (CANbus.available())  
{  
  
    CANbus.read(msg);  
  
    if (PRINT)  
    {  
  
        print_incoming_CAN_message(msg);  
  
    }  
  
}
```

//previously searched for a 1A0 node id but this isn't always the node id sent when the controller gets armed

```
if (msg.buf[4] == 55 && msg.buf[5] == 22) //37 and 16 in hex  
  
{  
  
    //were armed. lets go
```

```

        ret = 1;
    }

}

return ret;
}

int query_voltage_level(int node_id)
{
    CAN_message_t msg;

    int ret = 0;

    //send message to find voltage level

    msg.id = 0x600 + node_id;

    msg.ext = 0;

    msg.len = 4;

    msg.timeout = 0;

    //

    msg.buf[0] = 0x40;

    msg.buf[2] = 0x22;

    msg.buf[1] = 0x00;

    msg.buf[3] = 0x01;

    //

    msg.buf[5] = 0;

```

```
msg.buf[4] = 0;

msg.buf[6] = 0;

msg.buf[7] = 0;


//write message

if (CANbus.write(msg) == 0)

{

    ret = ERROR_CAN_WRITE;

}

else

{

//it will stop responding but i want to test

    //print_outgoing_CAN_message(msg);

    //Serial.println("waiting for response");

    //delayMicroseconds(WAIT_FOR_RESPONSE_TIME_FAST_US);

    while (CANbus.read(msg) == 0)

    {

        //wait for handshake

    }


    memcpy((char *)&ret, &(msg.buf[4]), 1);

    memcpy((char *)&ret + 1, &(msg.buf[5]), 1);


    //print_incoming_CAN_message(msg);

    //Serial.println(ret);
```

```
}
```

```
return ret;
```

```
}
```

```
int shutdown_MC(int node_id)
```

```
{
```

```
    CAN_message_t msg;
```

```
    int ret = 0;
```

```
    //tell MC's to go to shutdown state
```

```
    msg.id = 0x600 + node_id;
```

```
    msg.ext = 0;
```

```
    msg.len = 6;
```

```
    msg.timeout = 0;
```

```
    //
```

```
    msg.buf[0] = 0x2B; //68
```

```
    msg.buf[2] = 0x60; //60
```

```
    msg.buf[1] = 0x40;
```

```
    msg.buf[3] = 0x00;
```

```
    //
```

```
    msg.buf[5] = 0x00;
```

```
    msg.buf[4] = 0b000000110;
```

```
    msg.buf[6] = 0;
```

```
    msg.buf[7] = 0;
```

```
if (CANbus.write(msg) == 0)
{
    ret = ERROR_CAN_WRITE;

    if (PRINT)
    {
        Serial.println("ERROR: CAN Write.");

        Serial.print("Node id: ");

        Serial.println(node_id);
    }

}

else {

    //print_outgoing_CAN_message(msg);

    delayMicroseconds(WAIT_FOR_RESPONSE_TIME_SLOW_US);

    if (CANbus.read(msg) != 0)
    {
        if (PRINT)
        {
            Serial.println("Received Shutdown Confirmation");
        }

    }

}

else
{
```

```
    if (PRINT)
    {
        Serial.println("DID NOT Receive Shutdown Confirmation");
    }

}

}

return ret;
}
```

1.1.3 ULaren_CAN_Driver.h

Below:

```
#ifndef __ULAREN_CAN_DRIVER_H__
#define __ULAREN_CAN_DRIVER_H__
```

```
#include "FlexCAN.h"
```

```
#define NO_ERROR 0
```

```
#define ERROR_CAN_WRITE 1
```

```
#define PROFILE_VELOCITY_MODE 3
```

```
#define PRINT 0
```

```
#define SCALE_FACTOR 2
```

```

int initialize_CAN();
int initialize_MC(int node_id);
int arm_MC(int node_id);
int write_throttle_to_MC(int node_id, int throttle);
int initiate_target_velocity (int node_id);
int write_velocity_and_enable_MC(int node_id, int throttle);
void print_outgoing_CAN_message(CAN_message_t msg);
void print_incoming_CAN_message(CAN_message_t msg);
int send_statusword_request(int node_id);
void check_available_msg();
int reset_nodes();
void process_available_msgs();
int stop_remote_node(int node_id);
int diagnose_controller_message(CAN_message_t msg);
int link_node(int node_id);
int rearm_MC(int node_id);
int query_voltage_level(int node_id);
int shutdown_MC(int node_id);

```

#endif

1.1.4 Vector.h

Below:

```
/*
```

Inertial Measurement Unit Maths Library

Copyright (C) 2013-2014 Samuel Cowen

www.camelsoftware.com

Bug fixes and cleanups by Gé Vissers (gvissers@gmail.com)

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

*/

```
#ifndef IMUMATH_VECTOR_HPP
```

```
#define IMUMATH_VECTOR_HPP
```

```
#include <string.h>
```

```
#include <stdint.h>
```

```
#include <math.h>
```

```
namespace imu
```

```
{
```

```
template <uint8_t N> class Vector
```

```
{
```

```
public:
```

```
    Vector()
```

```
{  
    memset(p_vec, 0, sizeof(double)*N);  
}
```

Vector(double a)

```
{  
    memset(p_vec, 0, sizeof(double)*N);  
    p_vec[0] = a;  
}
```

Vector(double a, double b)

```
{  
    memset(p_vec, 0, sizeof(double)*N);  
    p_vec[0] = a;  
    p_vec[1] = b;  
}
```

Vector(double a, double b, double c)

```
{  
    memset(p_vec, 0, sizeof(double)*N);  
    p_vec[0] = a;  
    p_vec[1] = b;  
    p_vec[2] = c;  
}
```

Vector(double a, double b, double c, double d)

```
{  
    memset(p_vec, 0, sizeof(double)*N);  
    p_vec[0] = a;
```

```
p_vec[1] = b;  
p_vec[2] = c;  
p_vec[3] = d;  
}
```

```
Vector(const Vector<N> &v)  
{  
    for (int x = 0; x < N; x++)  
        p_vec[x] = v.p_vec[x];  
}
```

```
~Vector()  
{  
}
```

```
uint8_t n() { return N; }
```

```
double magnitude() const  
{  
    double res = 0;  
    for (int i = 0; i < N; i++)  
        res += p_vec[i] * p_vec[i];  
  
    return sqrt(res);  
}
```

```
void normalize()  
{  
    double mag = magnitude();
```

```

    if (isnan(mag) || mag == 0.0)
        return;

    for (int i = 0; i < N; i++)
        p_vec[i] /= mag;
}

```

```

double dot(const Vector& v) const
{
    double ret = 0;
    for (int i = 0; i < N; i++)
        ret += p_vec[i] * v.p_vec[i];

    return ret;
}

```

```

// The cross product is only valid for vectors with 3 dimensions,
// with the exception of higher dimensional stuff that is beyond
// the intended scope of this library.
// Only a definition for N==3 is given below this class, using
// cross() with another value for N will result in a link error.

```

```

Vector cross(const Vector& v) const;

```

```

Vector scale(double scalar) const
{
    Vector ret;
    for(int i = 0; i < N; i++)
        ret.p_vec[i] = p_vec[i] * scalar;
    return ret;
}

```

```
}
```

```
Vector invert() const
```

```
{
```

```
    Vector ret;
```

```
    for(int i = 0; i < N; i++)
```

```
        ret.p_vec[i] = -p_vec[i];
```

```
    return ret;
```

```
}
```

```
Vector& operator=(const Vector& v)
```

```
{
```

```
    for (int x = 0; x < N; x++ )
```

```
        p_vec[x] = v.p_vec[x];
```

```
    return *this;
```

```
}
```

```
double& operator [](int n)
```

```
{
```

```
    return p_vec[n];
```

```
}
```

```
double operator [](int n) const
```

```
{
```

```
    return p_vec[n];
```

```
}
```

```
double& operator ()(int n)
```

```
{
```

```
    return p_vec[n];  
}
```

```
double operator()(int n) const  
{  
    return p_vec[n];  
}
```

```
Vector operator+(const Vector& v) const  
{  
    Vector ret;  
    for(int i = 0; i < N; i++)  
        ret.p_vec[i] = p_vec[i] + v.p_vec[i];  
    return ret;  
}
```

```
Vector operator-(const Vector& v) const  
{  
    Vector ret;  
    for(int i = 0; i < N; i++)  
        ret.p_vec[i] = p_vec[i] - v.p_vec[i];  
    return ret;  
}
```

```
Vector operator * (double scalar) const  
{  
    return scale(scalar);  
}
```

Vector operator / (double scalar) const

```
{  
    Vector ret;  
    for(int i = 0; i < N; i++)  
        ret.p_vec[i] = p_vec[i] / scalar;  
    return ret;  
}
```

void toDegrees()

```
{  
    for(int i = 0; i < N; i++)  
        p_vec[i] *= 57.2957795131; //180/pi  
}
```

void toRadians()

```
{  
    for(int i = 0; i < N; i++)  
        p_vec[i] *= 0.01745329251; //pi/180  
}
```

double& x() { return p_vec[0]; }

double& y() { return p_vec[1]; }

double& z() { return p_vec[2]; }

double x() const { return p_vec[0]; }

double y() const { return p_vec[1]; }

double z() const { return p_vec[2]; }

private:

```

    double p_vec[N];
};

template <>
inline Vector<3> Vector<3>::cross(const Vector& v) const
{
    return Vector(
        p_vec[1] * v.p_vec[2] - p_vec[2] * v.p_vec[1],
        p_vec[2] * v.p_vec[0] - p_vec[0] * v.p_vec[2],
        p_vec[0] * v.p_vec[1] - p_vec[1] * v.p_vec[0]
    );
}

} // namespace

#endif

```

1.1.5 Structs.h

Below:

```

#include "Adafruit_Sensor.h"
#include "Adafruit_BNO055.h"
#include "imumaths.h"

struct IMUstructp{
    bool err;    // Indicates a communication error

    imu::Vector<3> accel; // Stores the 16-bit signed accelerometer sensor output
    imu::Vector<3> gyro;  // Stores the 16-bit signed gyro sensor output
    imu::Vector<3> mag;   // Stores the 16-bit signed magnetometer sensor output

```

```
imu::Quaternion quat; // Stores the 16-bit signed quaternion output
imu::Vector<3> eul; // Stores the 16-bit signed Euler angle output
imu::Vector<3> lia; // Stores the 16-bit signed linear acceleration output
imu::Vector<3> grv; // Stores the 16-bit signed gravity vector output
};
```

```
typedef struct IMUstructp IMUstruct;
```

1.1.6 Quarternion.h

Below:

```
/*
Inertial Measurement Unit Maths Library
Copyright (C) 2013-2014 Samuel Cowen
www.camelsoftware.com
```

Bug fixes and cleanups by Gé Vissers (gvissers@gmail.com)

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

```
*/
```

```
#ifndef IMUMATH_QUATERNION_HPP
```

```
#define IMUMATH_QUATERNION_HPP
```

```
#include <stdlib.h>
```

```
#include <string.h>
```

```
#include <stdint.h>
```

```
#include <math.h>
```

```
#include "matrix.h"
```

```
namespace imu
```

```
{
```

```
class Quaternion
```

```
{
```

```
public:
```

```
    Quaternion(): _w(1.0), _x(0.0), _y(0.0), _z(0.0) {}
```

```
    Quaternion(double w, double x, double y, double z):
```

```
        _w(w), _x(x), _y(y), _z(z) {}
```

```
    Quaternion(double w, Vector<3> vec):
```

```
        _w(w), _x(vec.x()), _y(vec.y()), _z(vec.z()) {}
```

```
    double& w()
```

```
{
```

```
    return _w;  
}
```

```
double& x()
```

```
{  
    return _x;  
}
```

```
double& y()
```

```
{  
    return _y;  
}
```

```
double& z()
```

```
{  
    return _z;  
}
```

```
double w() const
```

```
{  
    return _w;  
}
```

```
double x() const
```

```
{  
    return _x;  
}
```

```
double y() const
```

```
{  
    return _y;  
}
```

```
double z() const
```

```
{
```

```
    return _z;  
}
```

```
double magnitude() const  
{  
    return sqrt(_w*_w + _x*_x + _y*_y + _z*_z);  
}
```

```
void normalize()  
{  
    double mag = magnitude();  
    *this = this->scale(1/mag);  
}
```

```
Quaternion conjugate() const  
{  
    return Quaternion(_w, -_x, -_y, -_z);  
}
```

```
void fromAxisAngle(const Vector<3>& axis, double theta)  
{  
    _w = cos(theta/2);  
    //only need to calculate sine of half theta once  
    double sht = sin(theta/2);  
    _x = axis.x() * sht;  
    _y = axis.y() * sht;  
    _z = axis.z() * sht;  
}
```

```

void fromMatrix(const Matrix<3>& m)
{
    double tr = m.trace();

    double S;
    if (tr > 0)
    {
        S = sqrt(tr+1.0) * 2;
        _w = 0.25 * S;
        _x = (m(2, 1) - m(1, 2)) / S;
        _y = (m(0, 2) - m(2, 0)) / S;
        _z = (m(1, 0) - m(0, 1)) / S;
    }
    else if (m(0, 0) > m(1, 1) && m(0, 0) > m(2, 2))
    {
        S = sqrt(1.0 + m(0, 0) - m(1, 1) - m(2, 2)) * 2;
        _w = (m(2, 1) - m(1, 2)) / S;
        _x = 0.25 * S;
        _y = (m(0, 1) + m(1, 0)) / S;
        _z = (m(0, 2) + m(2, 0)) / S;
    }
    else if (m(1, 1) > m(2, 2))
    {
        S = sqrt(1.0 + m(1, 1) - m(0, 0) - m(2, 2)) * 2;
        _w = (m(0, 2) - m(2, 0)) / S;
        _x = (m(0, 1) + m(1, 0)) / S;
        _y = 0.25 * S;
        _z = (m(1, 2) + m(2, 1)) / S;
    }
}

```

```

else
{
    S = sqrt(1.0 + m(2, 2) - m(0, 0) - m(1, 1)) * 2;
    _w = (m(1, 0) - m(0, 1)) / S;
    _x = (m(0, 2) + m(2, 0)) / S;
    _y = (m(1, 2) + m(2, 1)) / S;
    _z = 0.25 * S;
}
}

void toAxisAngle(Vector<3>& axis, double& angle) const
{
    double sqw = sqrt(1-_w*_w);
    if (sqw == 0) //it's a singularity and divide by zero, avoid
        return;

    angle = 2 * acos(_w);
    axis.x() = _x / sqw;
    axis.y() = _y / sqw;
    axis.z() = _z / sqw;
}

```

```

Matrix<3> toMatrix() const
{
    Matrix<3> ret;
    ret.cell(0, 0) = 1 - 2*_y*_y - 2*_z*_z;
    ret.cell(0, 1) = 2*_x*_y - 2*_w*_z;
    ret.cell(0, 2) = 2*_x*_z + 2*_w*_y;

```

```

ret.cell(1, 0) = 2*_x*_y + 2*_w*_z;
ret.cell(1, 1) = 1 - 2*_x*_x - 2*_z*_z;
ret.cell(1, 2) = 2*_y*_z - 2*_w*_x;

ret.cell(2, 0) = 2*_x*_z - 2*_w*_y;
ret.cell(2, 1) = 2*_y*_z + 2*_w*_x;
ret.cell(2, 2) = 1 - 2*_x*_x - 2*_y*_y;
return ret;
}

```

```

// Returns euler angles that represent the quaternion. Angles are
// returned in rotation order and right-handed about the specified
// axes:
//
// v[0] is applied 1st about z (ie, roll)
// v[1] is applied 2nd about y (ie, pitch)
// v[2] is applied 3rd about x (ie, yaw)
//
// Note that this means result.x() is not a rotation about x;
// similarly for result.z().
//
Vector<3> toEuler() const
{
    Vector<3> ret;

    double sqw = _w*_w;
    double sqx = _x*_x;
    double sqy = _y*_y;
    double sqz = _z*_z;

```

```

ret.x() = atan2(2.0*(_x*_y+_z*_w),(sqx-sqy-sqz+sqw));
ret.y() = asin(-2.0*(_x*_z-_y*_w)/(sqx+sqy+sqz+sqw));
ret.z() = atan2(2.0*(_y*_z+_x*_w),(-sqx-sqy+sqz+sqw));

return ret;
}

```

Vector<3> toAngularVelocity(double dt) const

```

{
    Vector<3> ret;

    Quaternion one(1.0, 0.0, 0.0, 0.0);
    Quaternion delta = one - *this;
    Quaternion r = (delta/dt);
    r = r * 2;
    r = r * one;

    ret.x() = r.x();
    ret.y() = r.y();
    ret.z() = r.z();
    return ret;
}

```

Vector<3> rotateVector(const Vector<2>& v) const

```

{
    return rotateVector(Vector<3>(v.x(), v.y()));
}

```

Vector<3> rotateVector(const Vector<3>& v) const

```

{
    Vector<3> qv(_x, _y, _z);
    Vector<3> t = qv.cross(v) * 2.0;
    return v + t*_w + qv.cross(t);
}

```

Quaternion operator*(const Quaternion& q) const

```

{
    return Quaternion(
        _w*q._w - _x*q._x - _y*q._y - _z*q._z,
        _w*q._x + _x*q._w + _y*q._z - _z*q._y,
        _w*q._y - _x*q._z + _y*q._w + _z*q._x,
        _w*q._z + _x*q._y - _y*q._x + _z*q._w
    );
}

```

Quaternion operator+(const Quaternion& q) const

```

{
    return Quaternion(_w + q._w, _x + q._x, _y + q._y, _z + q._z);
}

```

Quaternion operator-(const Quaternion& q) const

```

{
    return Quaternion(_w - q._w, _x - q._x, _y - q._y, _z - q._z);
}

```

Quaternion operator/(double scalar) const

```

{

```

```

        return Quaternion(_w / scalar, _x / scalar, _y / scalar, _z / scalar);
    }

Quaternion operator*(double scalar) const
{
    return scale(scalar);
}

Quaternion scale(double scalar) const
{
    return Quaternion(_w * scalar, _x * scalar, _y * scalar, _z * scalar);
}

private:
    double _w, _x, _y, _z;
};

} // namespace

#endif

```

1.1.7 Output_handler.h

Below:

```

/*
 * This .h file defines function and class prototypes that are referenced in the main program file
 */

// Servo PWM initialization
void initServo();

// Pass writeServo -500 to 500 and have it mapped to full left to full right

```

```
void writeServo(int16_t pos);
```

```
// Pass writetempMotor -500 to 500 and have it mapped to full reverse to full forward
```

```
void writetempMotor(int16_t pos);
```

```
//-----
```

1.1.8 Output_handler.cpp

Below:

```
/*
```

```
* This .cpp file defines the content of functions to be called by the main routine
```

```
*/
```

```
// Allow access to arduino specific datatypes
```

```
#include <arduino.h>
```

```
#include <Servo.h>
```

```
// Servo-----
```

```
// Pin numbers for IO
```

```
#define servo_out 21
```

```
#define motor_out 7
```

```
//-----
```

```
// Functions to initialize and actuate Servo-----  
-----
```

```
void initServo(){
```

```
    // Set data direction
```

```
    pinMode(servo_out, OUTPUT);
```

```
    // Set PWM freq and resolution
```

```
    analogWriteFrequency(21,400); //Note this also changes PWM freq on LED pins to 400Hz
```

```

    analogWriteFrequency(7,400); //Note this also changes PWM freq on LED pins to 400Hz
    analogWriteRes(12);        //Note this also changes PWM res on all pins to 12 bit
}

// maps -500 to 500 to the appropriate pwm out for 1ms pulse to 2ms pulse
void writeServo(int16_t pos){
    analogWrite(servo_out, map(pos,-1500,1000,0,4096));
}

// maps -500 to 500 to the appropriate pwm out for 1ms pulse to 2ms pulse
void writetempMotor(int16_t pos){
    analogWrite(motor_out, map(pos,-1500,1000,0,4096));
}

//-----

```

1.1.9 Matrix.h

Below:

```

/*
    Inertial Measurement Unit Maths Library
    Copyright (C) 2013-2014 Samuel Cowen
    www.camelsoftware.com

    Bug fixes and cleanups by Gé Vissers (gvissers@gmail.com)

```

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <<http://www.gnu.org/licenses/>>.

*/

```
#ifndef IMUMATH_MATRIX_HPP
#define IMUMATH_MATRIX_HPP
```

```
#include <string.h>
#include <stdint.h>
```

```
#include "vector.h"
```

```
namespace imu
{
```

```
template <uint8_t N> class Matrix
{
public:
    Matrix()
    {
        memset(_cell_data, 0, N*N*sizeof(double));
    }
}
```

```
Matrix(const Matrix &m)
```

```
{  
    for (int ij = 0; ij < N*N; ++ij)  
    {  
        _cell_data[ij] = m._cell_data[ij];  
    }  
}
```

```
~Matrix()
```

```
{  
}
```

```
Matrix& operator=(const Matrix& m)
```

```
{  
    for (int ij = 0; ij < N*N; ++ij)  
    {  
        _cell_data[ij] = m._cell_data[ij];  
    }  
    return *this;  
}
```

```
Vector<N> row_to_vector(int i) const
```

```
{  
    Vector<N> ret;  
    for (int j = 0; j < N; j++)  
    {  
        ret[j] = cell(i, j);  
    }  
    return ret;  
}
```

```
}
```

```
Vector<N> col_to_vector(int j) const
```

```
{
```

```
    Vector<N> ret;
```

```
    for (int i = 0; i < N; i++)
```

```
    {
```

```
        ret[i] = cell(i, j);
```

```
    }
```

```
    return ret;
```

```
}
```

```
void vector_to_row(const Vector<N>& v, int i)
```

```
{
```

```
    for (int j = 0; j < N; j++)
```

```
    {
```

```
        cell(i, j) = v[j];
```

```
    }
```

```
}
```

```
void vector_to_col(const Vector<N>& v, int j)
```

```
{
```

```
    for (int i = 0; i < N; i++)
```

```
    {
```

```
        cell(i, j) = v[i];
```

```
    }
```

```
}
```

```
double operator()(int i, int j) const
```

```

{
    return cell(i, j);
}

double& operator()(int i, int j)
{
    return cell(i, j);
}

```

```

double cell(int i, int j) const
{
    return _cell_data[i*N+j];
}

double& cell(int i, int j)
{
    return _cell_data[i*N+j];
}

```

```

Matrix operator+(const Matrix& m) const
{
    Matrix ret;
    for (int ij = 0; ij < N*N; ++ij)
    {
        ret._cell_data[ij] = _cell_data[ij] + m._cell_data[ij];
    }
    return ret;
}

```

```

Matrix operator-(const Matrix& m) const

```

```

{
    Matrix ret;

    for (int ij = 0; ij < N*N; ++ij)
    {
        ret._cell_data[ij] = _cell_data[ij] - m._cell_data[ij];
    }

    return ret;
}

```

Matrix operator*(double scalar) const

```

{
    Matrix ret;

    for (int ij = 0; ij < N*N; ++ij)
    {
        ret._cell_data[ij] = _cell_data[ij] * scalar;
    }

    return ret;
}

```

Matrix operator*(const Matrix& m) const

```

{
    Matrix ret;

    for (int i = 0; i < N; i++)
    {
        Vector<N> row = row_to_vector(i);

        for (int j = 0; j < N; j++)
        {
            ret(i, j) = row.dot(m.col_to_vector(j));
        }
    }
}

```

```
    }  
    return ret;  
}
```

Matrix transpose() const

```
{  
    Matrix ret;  
    for (int i = 0; i < N; i++)  
    {  
        for (int j = 0; j < N; j++)  
        {  
            ret(j, i) = cell(i, j);  
        }  
    }  
    return ret;  
}
```

Matrix<N-1> minor_matrix(int row, int col) const

```
{  
    Matrix<N-1> ret;  
    for (int i = 0, im = 0; i < N; i++)  
    {  
        if (i == row)  
            continue;  
  
        for (int j = 0, jm = 0; j < N; j++)  
        {  
            if (j != col)  
            {
```

```

        ret(im, jm++) = cell(i, j);
    }
}
im++;
}
return ret;
}

```

```

double determinant() const
{
    // specialization for N == 1 given below this class
    double det = 0.0, sign = 1.0;
    for (int i = 0; i < N; ++i, sign = -sign)
        det += sign * cell(0, i) * minor_matrix(0, i).determinant();
    return det;
}

```

```

Matrix invert() const
{
    Matrix ret;
    double det = determinant();

    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            ret(i, j) = minor_matrix(j, i).determinant() / det;
            if ((i+j)%2 == 1)
                ret(i, j) = -ret(i, j);
        }
    }
}

```

```
    }  
    }  
    return ret;  
}
```

```
double trace() const  
{  
    double tr = 0.0;  
    for (int i = 0; i < N; ++i)  
        tr += cell(i, i);  
    return tr;  
}
```

```
private:  
    double _cell_data[N*N];  
};
```

```
template<>  
inline double Matrix<1>::determinant() const  
{  
    return cell(0, 0);  
}  
  
};
```

```
#endif
```

1.1.10 Loop.h

Below:

```
#ifndef _LOOP_H_
```

```
#define _LOOP_H_
```

```
enum state {
```

```
    INITIALIZE_PERIPHERALS,
```

```
    INITIALIZE_CONTROLLERS,
```

```
    WAIT_FOR_ARM,
```

```
    LINK_COMMUNICATION,
```

```
    RUNNING_NOMINALLY,
```

```
    RUNNING_SIMULINK,
```

```
    INDICATE_AND_LOG_ERROR,
```

```
    WAIT_FOR_CLEAR,
```

```
};
```

```
typedef enum state State;
```

```
#endif
```

1.1.11 Kinetis_flexcan.h

Below:

```
/*
```

```
 * File:   kinetis_flexcan.h
```

```
 * Purpose: Register and bit definitions
```

```
 */
```

```
#ifndef __KINETIS_FLEXCAN_H
```

```
#define __KINETIS_FLEXCAN_H
```

```
#include <stdint.h>
```

```
/* Common bit definition */
```

```
#define BIT0      (1L)
#define BIT1      (BIT0<<1)
#define BIT2      (BIT0<<2)
#define BIT3      (BIT0<<3)
#define BIT4      (BIT0<<4)
#define BIT5      (BIT0<<5)
#define BIT6      (BIT0<<6)
#define BIT7      (BIT0<<7)
#define BIT8      (BIT0<<8)
#define BIT9      (BIT0<<9)
#define BIT10     (BIT0<<10)
#define BIT11     (BIT0<<11)
#define BIT12     (0x00001000L)
#define BIT13     (0x00002000L)
#define BIT14     (0x00004000L)
#define BIT15     (0x00008000L)
#define BIT16     (0x00010000L)
#define BIT17     (0x00020000L)
#define BIT18     (0x00040000L)
#define BIT19     (0x00080000L)
#define BIT20     (0x00100000L)
#define BIT21     (0x00200000L)
#define BIT22     (0x00400000L)
#define BIT23     (0x00800000L)
#define BIT24     (0x01000000L)
```

```
#define BIT25      (0x02000000L)
#define BIT26      (0x04000000L)
#define BIT27      (0x08000000L)
#define BIT28      (0x10000000L)
#define BIT29      (0x20000000L)
#define BIT30      (0x40000000L)
#define BIT31      (0x80000000L)
```

```
/* FlexCAN module I/O Base Addresss */
```

```
#define FLEXCAN0_BASE      (0x40024000L)
#define FLEXCAN1_BASE      (0x400A4000L)
```

```
typedef volatile uint32_t vuint32_t;
```

```
/*
 *
 * FlexCAN0 (FLEXCAN0)
 *
 *****/
```

```
/* Register read/write macros */
```

```
#define FLEXCAN0_MCR      (*(vuint32_t*)(FLEXCAN0_BASE))
#define FLEXCAN0_CTRL1    (*(vuint32_t*)(FLEXCAN0_BASE+4))
#define FLEXCAN0_TIMER    (*(vuint32_t*)(FLEXCAN0_BASE+8))
#define FLEXCAN0_TCR      (*(vuint32_t*)(FLEXCAN0_BASE+0x0C))
#define FLEXCAN0_RXMGMASK  (*(vuint32_t*)(FLEXCAN0_BASE+0x10))
#define FLEXCAN0_RX14MASK  (*(vuint32_t*)(FLEXCAN0_BASE+0x14))
#define FLEXCAN0_RX15MASK  (*(vuint32_t*)(FLEXCAN0_BASE+0x18))
```

```

#define FLEXCAN0_ECR          (*(vuint32_t*)(FLEXCAN0_BASE+0x1C))
#define FLEXCAN0_ESR1        (*(vuint32_t*)(FLEXCAN0_BASE+0x20))
#define FLEXCAN0_IMASK2      (*(vuint32_t*)(FLEXCAN0_BASE+0x24))
#define FLEXCAN0_IMASK1      (*(vuint32_t*)(FLEXCAN0_BASE+0x28))
#define FLEXCAN0_IFLAG2      (*(vuint32_t*)(FLEXCAN0_BASE+0x2C))
#define FLEXCAN0_IFLAG1      (*(vuint32_t*)(FLEXCAN0_BASE+0x30))
#define FLEXCAN0_CTRL2       (*(vuint32_t*)(FLEXCAN0_BASE+0x34))
#define FLEXCAN0_ESR2        (*(vuint32_t*)(FLEXCAN0_BASE+0x38))
#define FLEXCAN0_FUREQ       (*(vuint32_t*)(FLEXCAN0_BASE+0x3C))
#define FLEXCAN0_FUACK       (*(vuint32_t*)(FLEXCAN0_BASE+0x40))
#define FLEXCAN0_CRCCR       (*(vuint32_t*)(FLEXCAN0_BASE+0x44))
#define FLEXCAN0_RXFGMASK    (*(vuint32_t*)(FLEXCAN0_BASE+0x48))
#define FLEXCAN0_RXFIR       (*(vuint32_t*)(FLEXCAN0_BASE+0x4C))
#define FLEXCAN0_DBG1        (*(vuint32_t*)(FLEXCAN0_BASE+0x58))
#define FLEXCAN0_DBG2        (*(vuint32_t*)(FLEXCAN0_BASE+0x5C))

#define FLEXCAN0_IMEUR        FLEXCAN0_FUREQ
#define FLEXCAN0_LRFR        FLEXCAN0_FUACK

/* Message Buffers */
#define FLEXCAN0_MB0_CS       (*(vuint32_t*)(FLEXCAN0_BASE+0x80))
#define FLEXCAN0_MB0_ID       (*(vuint32_t*)(FLEXCAN0_BASE+0x84))
#define FLEXCAN0_MB0_WORD0    (*(vuint32_t*)(FLEXCAN0_BASE+0x88))
#define FLEXCAN0_MB0_WORD1    (*(vuint32_t*)(FLEXCAN0_BASE+0x8C))

#define FLEXCAN0_MBn_CS(n)    (*(vuint32_t*)(FLEXCAN0_BASE+0x80+n*0x10))
#define FLEXCAN0_MBn_ID(n)    (*(vuint32_t*)(FLEXCAN0_BASE+0x84+n*0x10))
#define FLEXCAN0_MBn_WORD0(n) (*(vuint32_t*)(FLEXCAN0_BASE+0x88+n*0x10))

```

```
#define FLEXCAN0_MBn_WORD1(n)          (*(vuint32_t*)(FLEXCAN0_BASE+0x8C+n*0x10))
```

```
/* Rx Individual Mask Registers */
```

```
#define FLEXCAN0_RXIMR0                (*(vuint32_t*)(FLEXCAN0_BASE+0x880))
```

```
#define FLEXCAN0_RXIMRn(n)             (*(vuint32_t*)(FLEXCAN0_BASE+0x880+n*4))
```

```
/* Rx FIFO ID Filter Table Element 0 to 127 */
```

```
#define FLEXCAN0_IDFLT_TAB0            (*(vuint32_t*)(FLEXCAN0_BASE+0xE0))
```

```
#define FLEXCAN0_IDFLT_TAB(n)          (*(vuint32_t*)(FLEXCAN0_BASE+0xE0+(n*4)))
```

```
//#define FLEXCAN0_IDFLT_TAB(n)        (*(vuint32_t*)(FLEXCAN0_BASE+0xE0+(n<<2)))
```

```
/* Memory Error Control Register */
```

```
#define FLEXCAN0_MECR  
    (*(vuint32_t*)(FLEXCAN0_BASE+0x3B70))
```

```
/* Error Injection Address Register */
```

```
#define FLEXCAN0_ERRIAR  
    (*(vuint32_t*)(FLEXCAN0_BASE+0x3B74))
```

```
/* Error Injection Data Pattern Register */
```

```
#define FLEXCAN0_ERRIDPR                (*(vuint32_t*)(FLEXCAN0_BASE+0x3B78))
```

```
/* Error Injection Parity Pattern Register */
```

```
#define FLEXCAN0_ERRIPPR                (*(vuint32_t*)(FLEXCAN0_BASE+0x3B7C))
```

```
/* Error Report Address Register */
```

```
#define FLEXCAN0_RERRAR  
    (*(vuint32_t*)(FLEXCAN0_BASE+0x3B80))
```

```
/* Error Report Data Register */
```

```

#define FLEXCAN0_RERRDR
    *(vuint32_t*)(FLEXCAN0_BASE+0x3B84))

/* Error Report Syndrome Register */

#define FLEXCAN0_RERRSYNR
    *(vuint32_t*)(FLEXCAN0_BASE+0x3B88))

/* Error Status Register */

#define FLEXCAN0_ERRSR
    *(vuint32_t*)(FLEXCAN0_BASE+0x3B8C))

/*****
*
* FlexCAN1 (FLEXCAN1)
*
*****/

/* Register read/write macros */

#define FLEXCAN1_MCR
    (*(vuint32_t*)(FLEXCAN1_BASE))

#define FLEXCAN1_CTRL1
    (*(vuint32_t*)(FLEXCAN1_BASE+4))

#define FLEXCAN1_TIMER
    (*(vuint32_t*)(FLEXCAN1_BASE+8))

#define FLEXCAN1_TCR
    (*(vuint32_t*)(FLEXCAN1_BASE+0x0C))

#define FLEXCAN1_RXMGMASK
    (*(vuint32_t*)(FLEXCAN1_BASE+0x10))

#define FLEXCAN1_RX14MASK
    (*(vuint32_t*)(FLEXCAN1_BASE+0x14))

#define FLEXCAN1_RX15MASK
    (*(vuint32_t*)(FLEXCAN1_BASE+0x18))

#define FLEXCAN1_ECR
    (*(vuint32_t*)(FLEXCAN1_BASE+0x1C))

#define FLEXCAN1_ESR1
    (*(vuint32_t*)(FLEXCAN1_BASE+0x20))

#define FLEXCAN1_IMASK2
    (*(vuint32_t*)(FLEXCAN1_BASE+0x24))

#define FLEXCAN1_IMASK1
    (*(vuint32_t*)(FLEXCAN1_BASE+0x28))

#define FLEXCAN1_IFLAG2
    (*(vuint32_t*)(FLEXCAN1_BASE+0x2C))

```

```

#define FLEXCAN1_IFLAG1      (*(vuint32_t*)(FLEXCAN1_BASE+0x30))
#define FLEXCAN1_CTRL2      (*(vuint32_t*)(FLEXCAN1_BASE+0x34))
#define FLEXCAN1_ESR2       (*(vuint32_t*)(FLEXCAN1_BASE+0x38))
#define FLEXCAN1_FUREQ      (*(vuint32_t*)(FLEXCAN1_BASE+0x3C))
#define FLEXCAN1_FUACK      (*(vuint32_t*)(FLEXCAN1_BASE+0x40))
#define FLEXCAN1_CRCR       (*(vuint32_t*)(FLEXCAN1_BASE+0x44))
#define FLEXCAN1_RXFGMASK   (*(vuint32_t*)(FLEXCAN1_BASE+0x48))
#define FLEXCAN1_RXFIR      (*(vuint32_t*)(FLEXCAN1_BASE+0x4C))
#define FLEXCAN1_DBG1       (*(vuint32_t*)(FLEXCAN1_BASE+0x58))
#define FLEXCAN1_DBG2       (*(vuint32_t*)(FLEXCAN1_BASE+0x5C))

#define FLEXCAN1_IMEUR      FLEXCAN1_FUREQ
#define FLEXCAN1_LRFR       FLEXCAN1_FUACK

/* Message Buffers */
#define FLEXCAN1_MB0_CS      (*(vuint32_t*)(FLEXCAN1_BASE+0x80))
#define FLEXCAN1_MB0_ID      (*(vuint32_t*)(FLEXCAN1_BASE+0x84))
#define FLEXCAN1_MB0_WORD0   (*(vuint32_t*)(FLEXCAN1_BASE+0x88))
#define FLEXCAN1_MB0_WORD1   (*(vuint32_t*)(FLEXCAN1_BASE+0x8C))

#define FLEXCAN1_MBn_CS(n)   (*(vuint32_t*)(FLEXCAN1_BASE+0x80+n*0x10))
#define FLEXCAN1_MBn_ID(n)   (*(vuint32_t*)(FLEXCAN1_BASE+0x84+n*0x10))
#define FLEXCAN1_MBn_WORD0(n) (*(vuint32_t*)(FLEXCAN1_BASE+0x88+n*0x10))
#define FLEXCAN1_MBn_WORD1(n) (*(vuint32_t*)(FLEXCAN1_BASE+0x8C+n*0x10))

/* Rx Individual Mask Registers */
#define FLEXCAN1_RXIMR0      (*(vuint32_t*)(FLEXCAN1_BASE+0x880))
#define FLEXCAN1_RXIMRn(n)   (*(vuint32_t*)(FLEXCAN1_BASE+0x880+n*4))

```

/* Rx FIFO ID Filter Table Element 0 to 127 */

#define FLEXCAN1_IDFLT_TAB0 (*(vuint32_t*)(FLEXCAN1_BASE+0xE0))

#define FLEXCAN1_IDFLT_TAB(n) (*(vuint32_t*)(FLEXCAN1_BASE+0xE0+(n<<2)))

/* Memory Error Control Register */

#define FLEXCAN1_MECR
(vuint32_t)(FLEXCAN1_BASE+0x7B70))

/* Error Injection Address Register */

#define FLEXCAN1_ERRIAR
(vuint32_t)(FLEXCAN1_BASE+0x3B74))

/* Error Injection Data Pattern Register */

#define FLEXCAN1_ERRIDPR
(vuint32_t)(FLEXCAN1_BASE+0x3B78))

/* Error Injection Parity Pattern Register */

#define FLEXCAN1_ERRIPPR
(vuint32_t)(FLEXCAN1_BASE+0x3B7C))

/* Error Report Address Register */

#define FLEXCAN1_RERRAR
(vuint32_t)(FLEXCAN1_BASE+0x3B80))

/* Error Report Data Register */

#define FLEXCAN1_RERRDR
(vuint32_t)(FLEXCAN1_BASE+0x3B84))

/* Error Report Syndrome Register */

#define FLEXCAN1_RERRSYNR
(vuint32_t)(FLEXCAN1_BASE+0x3B88))

```
/* Error Status Register */
```

```
#define FLEXCAN1_ERRSR  
    *(vuint32_t*)(FLEXCAN1_BASE+0x3B8C))
```

```
/* Bit definitions and macros for FLEXCAN_MCR */
```

```
#define FLEXCAN_MCR_MAXMB(x)    (((x)&0x0000007F)<<0)  
#define FLEXCAN_MCR_IDAM(x)    (((x)&0x00000003)<<8)  
#define FLEXCAN_MCR_MAXMB_MASK    (0x0000007F)  
#define FLEXCAN_MCR_IDAM_MASK    (0x00000300)  
#define FLEXCAN_MCR_IDAM_BIT_NO    (8)  
#define FLEXCAN_MCR_AEN    (0x00001000)  
#define FLEXCAN_MCR_LPRIQ_EN    (0x00002000)  
#define FLEXCAN_MCR_IRMQ    (0x00010000)  
#define FLEXCAN_MCR_SRQ_DIS    (0x00020000)  
#define FLEXCAN_MCR_DOZE    (0x00040000)  
#define FLEXCAN_MCR_WAK_SRC    (0x00080000)  
#define FLEXCAN_MCR_LPM_ACK    (0x00100000)  
#define FLEXCAN_MCR_WRN_EN    (0x00200000)  
#define FLEXCAN_MCR_SLF_WAK    (0x00400000)  
#define FLEXCAN_MCR_SUPV    (0x00800000)  
#define FLEXCAN_MCR_FRZ_ACK    (0x01000000)  
#define FLEXCAN_MCR_SOFT_RST    (0x02000000)  
#define FLEXCAN_MCR_WAK_MSK    (0x04000000)  
#define FLEXCAN_MCR_NOT_RDY    (0x08000000)  
#define FLEXCAN_MCR_HALT    (0x10000000)  
#define FLEXCAN_MCR_FEN    (0x20000000)  
#define FLEXCAN_MCR_FRZ    (0x40000000)  
#define FLEXCAN_MCR_MDIS    (0x80000000)
```

```

/* Bit definitions and macros for FLEXCAN_CTRL */

#define FLEXCAN_CTRL_PROPSEG(x)    (((x)&0x00000007L)<<0)

#define FLEXCAN_CTRL_LOM          (0x00000008)

#define FLEXCAN_CTRL_LBUF         (0x00000010)

#define FLEXCAN_CTRL_TSYNC        (0x00000020)

#define FLEXCAN_CTRL_BOFF_REC     (0x00000040)

#define FLEXCAN_CTRL_SMP          (0x00000080)

#define FLEXCAN_CTRL_RWRN_MSK     (0x00000400)

#define FLEXCAN_CTRL_TWRN_MSK     (0x00000800)

#define FLEXCAN_CTRL_LPB          (0x00001000L)

#define FLEXCAN_CTRL_CLK_SRC      (0x00002000)

#define FLEXCAN_CTRL_ERR_MSK      (0x00004000)

#define FLEXCAN_CTRL_BOFF_MSK     (0x00008000)

#define FLEXCAN_CTRL_PSEG2(x)     (((x)&0x00000007L)<<16)

#define FLEXCAN_CTRL_PSEG1(x)     (((x)&0x00000007L)<<19)

#define FLEXCAN_CTRL_RJW(x)       (((x)&0x00000003L)<<22)

#define FLEXCAN_CTRL_PRESDIV(x)    (((x)&0x000000FFL)<<24)

```

```

/* Bit definitions and macros for FLEXCAN_CTRL2 */

#define FLEXCAN_CTRL2_IMEUEN      (BIT31)

#define FLEXCAN_CTRL2_RFFN        (0x0F000000L)

#define FLEXCAN_CTRL2_RFFN_BIT_NO (24)

#define FLEXCAN_CTRL2_TASD        (0x00F80000L)

#define FLEXCAN_CTRL2_TASD_BIT_NO (19)

#define FLEXCAN_CTRL2_MRP         (BIT18)

#define FLEXCAN_CTRL2_RRS         (BIT17)

#define FLEXCAN_CTRL2_EACEN       (BIT16)

#define FLEXCAN_CTRL2_MUMASK      (BIT1)

#define FLEXCAN_CTRL2_FUMASK      (BIT0)

```

```
#define FLEXCAN_CTRL2_LOSTRMSK          (BIT2)

#define FLEXCAN_CTRL2_LOSTRMMSK        (BIT1)

#define FLEXCAN_CTRL2_IMEUMASK          (BIT0)

#define FLEXCAN_set_rffn(ctrl2,rffn)    ctrl2 = ((ctrl2) & ~FLEXCAN_CTRL2_RFFN) | ((rffn & 0xF)<<FLEXCAN_CTRL2_RFFN_BIT_NO)
```

```
/* Bit definitions and macros for FLEXCAN_TIMER */
```

```
#define FLEXCAN_TIMER_TIMER(x)          (((x)&0x0000FFFF)<<0)
```

```
/* Bit definitions and macros for FLEXCAN_TCR */
```

```
#define FLEXCAN_TCR_DSCACK              (0x00000100)
```

```
#define FLEXCAN_TCR_BIT_CLS             (0x00000200)
```

```
#define FLEXCAN_TCR_TRD                 (0x00000400)
```

```
/* Bit definitions and macros for FLEXCAN_RXGMASK */
```

```
#define FLEXCAN_RXGMASK_MI0            (0x00000001)
```

```
#define FLEXCAN_RXGMASK_MI1            (0x00000002)
```

```
#define FLEXCAN_RXGMASK_MI2            (0x00000004)
```

```
#define FLEXCAN_RXGMASK_MI3            (0x00000008)
```

```
#define FLEXCAN_RXGMASK_MI4            (0x00000010)
```

```
#define FLEXCAN_RXGMASK_MI5            (0x00000020)
```

```
#define FLEXCAN_RXGMASK_MI6            (0x00000040)
```

```
#define FLEXCAN_RXGMASK_MI7            (0x00000080)
```

```
#define FLEXCAN_RXGMASK_MI8            (0x00000100)
```

```
#define FLEXCAN_RXGMASK_MI9            (0x00000200)
```

```
#define FLEXCAN_RXGMASK_MI10           (0x00000400)
```

```
#define FLEXCAN_RXGMASK_MI11           (0x00000800)
```

```
#define FLEXCAN_RXGMASK_MI12           (0x00001000)
```

```
#define FLEXCAN_RXGMASK_MI13    (0x00002000)
#define FLEXCAN_RXGMASK_MI14    (0x00004000)
#define FLEXCAN_RXGMASK_MI15    (0x00008000)
#define FLEXCAN_RXGMASK_MI16    (0x00010000)
#define FLEXCAN_RXGMASK_MI17    (0x00020000)
#define FLEXCAN_RXGMASK_MI18    (0x00040000)
#define FLEXCAN_RXGMASK_MI19    (0x00080000)
#define FLEXCAN_RXGMASK_MI20    (0x00100000)
#define FLEXCAN_RXGMASK_MI21    (0x00200000)
#define FLEXCAN_RXGMASK_MI22    (0x00400000)
#define FLEXCAN_RXGMASK_MI23    (0x00800000)
#define FLEXCAN_RXGMASK_MI24    (0x01000000)
#define FLEXCAN_RXGMASK_MI25    (0x02000000)
#define FLEXCAN_RXGMASK_MI26    (0x04000000)
#define FLEXCAN_RXGMASK_MI27    (0x08000000)
#define FLEXCAN_RXGMASK_MI28    (0x10000000)
#define FLEXCAN_RXGMASK_MI29    (0x20000000)
#define FLEXCAN_RXGMASK_MI30    (0x40000000)
#define FLEXCAN_RXGMASK_MI31    (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RX14MASK */
```

```
#define FLEXCAN_RX14MASK_MI0    (0x00000001)
#define FLEXCAN_RX14MASK_MI1    (0x00000002)
#define FLEXCAN_RX14MASK_MI2    (0x00000004)
#define FLEXCAN_RX14MASK_MI3    (0x00000008)
#define FLEXCAN_RX14MASK_MI4    (0x00000010)
#define FLEXCAN_RX14MASK_MI5    (0x00000020)
#define FLEXCAN_RX14MASK_MI6    (0x00000040)
#define FLEXCAN_RX14MASK_MI7    (0x00000080)
```

```
#define FLEXCAN_RX14MASK_MI8      (0x00000100)
#define FLEXCAN_RX14MASK_MI9      (0x00000200)
#define FLEXCAN_RX14MASK_MI10     (0x00000400)
#define FLEXCAN_RX14MASK_MI11     (0x00000800)
#define FLEXCAN_RX14MASK_MI12     (0x00001000)
#define FLEXCAN_RX14MASK_MI13     (0x00002000)
#define FLEXCAN_RX14MASK_MI14     (0x00004000)
#define FLEXCAN_RX14MASK_MI15     (0x00008000)
#define FLEXCAN_RX14MASK_MI16     (0x00010000)
#define FLEXCAN_RX14MASK_MI17     (0x00020000)
#define FLEXCAN_RX14MASK_MI18     (0x00040000)
#define FLEXCAN_RX14MASK_MI19     (0x00080000)
#define FLEXCAN_RX14MASK_MI20     (0x00100000)
#define FLEXCAN_RX14MASK_MI21     (0x00200000)
#define FLEXCAN_RX14MASK_MI22     (0x00400000)
#define FLEXCAN_RX14MASK_MI23     (0x00800000)
#define FLEXCAN_RX14MASK_MI24     (0x01000000)
#define FLEXCAN_RX14MASK_MI25     (0x02000000)
#define FLEXCAN_RX14MASK_MI26     (0x04000000)
#define FLEXCAN_RX14MASK_MI27     (0x08000000)
#define FLEXCAN_RX14MASK_MI28     (0x10000000)
#define FLEXCAN_RX14MASK_MI29     (0x20000000)
#define FLEXCAN_RX14MASK_MI30     (0x40000000)
#define FLEXCAN_RX14MASK_MI31     (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RX15MASK */
```

```
#define FLEXCAN_RX15MASK_MI0      (0x00000001)
#define FLEXCAN_RX15MASK_MI1      (0x00000002)
#define FLEXCAN_RX15MASK_MI2      (0x00000004)
```

```
#define FLEXCAN_RX15MASK_MI3      (0x00000008)
#define FLEXCAN_RX15MASK_MI4      (0x00000010)
#define FLEXCAN_RX15MASK_MI5      (0x00000020)
#define FLEXCAN_RX15MASK_MI6      (0x00000040)
#define FLEXCAN_RX15MASK_MI7      (0x00000080)
#define FLEXCAN_RX15MASK_MI8      (0x00000100)
#define FLEXCAN_RX15MASK_MI9      (0x00000200)
#define FLEXCAN_RX15MASK_MI10     (0x00000400)
#define FLEXCAN_RX15MASK_MI11     (0x00000800)
#define FLEXCAN_RX15MASK_MI12     (0x00001000)
#define FLEXCAN_RX15MASK_MI13     (0x00002000)
#define FLEXCAN_RX15MASK_MI14     (0x00004000)
#define FLEXCAN_RX15MASK_MI15     (0x00008000)
#define FLEXCAN_RX15MASK_MI16     (0x00010000)
#define FLEXCAN_RX15MASK_MI17     (0x00020000)
#define FLEXCAN_RX15MASK_MI18     (0x00040000)
#define FLEXCAN_RX15MASK_MI19     (0x00080000)
#define FLEXCAN_RX15MASK_MI20     (0x00100000)
#define FLEXCAN_RX15MASK_MI21     (0x00200000)
#define FLEXCAN_RX15MASK_MI22     (0x00400000)
#define FLEXCAN_RX15MASK_MI23     (0x00800000)
#define FLEXCAN_RX15MASK_MI24     (0x01000000)
#define FLEXCAN_RX15MASK_MI25     (0x02000000)
#define FLEXCAN_RX15MASK_MI26     (0x04000000)
#define FLEXCAN_RX15MASK_MI27     (0x08000000)
#define FLEXCAN_RX15MASK_MI28     (0x10000000)
#define FLEXCAN_RX15MASK_MI29     (0x20000000)
#define FLEXCAN_RX15MASK_MI30     (0x40000000)
#define FLEXCAN_RX15MASK_MI31     (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_ECR */
```

```
#define FLEXCAN_ECR_TX_ERR_COUNTER(x) (((x)&0x000000FF)<<0)
```

```
#define FLEXCAN_ECR_RX_ERR_COUNTER(x) (((x)&0x000000FF)<<8)
```

```
/* Bit definitions and macros for FLEXCAN_ESR1 */
```

```
#define FLEXCAN_ESR_WAK_INT      (0x00000001)
```

```
#define FLEXCAN_ESR_ERR_INT      (0x00000002)
```

```
#define FLEXCAN_ESR_BOFF_INT     (0x00000004)
```

```
#define FLEXCAN_ESR_RX          (0x00000008)
```

```
#define FLEXCAN_ESR_FLT_CONF(x)  (((x)&0x00000003)<<4)
```

```
#define FLEXCAN_ESR_FLT_CONF_MASK      (0x00000030)
```

```
#define FLEXCAN_ESR_TX          (0x00000040)
```

```
#define FLEXCAN_ESR_IDLE       (0x00000080)
```

```
#define FLEXCAN_ESR_RX_WRN     (0x00000100)
```

```
#define FLEXCAN_ESR_TX_WRN     (0x00000200)
```

```
#define FLEXCAN_ESR_STF_ERR     (0x00000400)
```

```
#define FLEXCAN_ESR_FRM_ERR     (0x00000800)
```

```
#define FLEXCAN_ESR_CRC_ERR     (0x00001000)
```

```
#define FLEXCAN_ESR_ACK_ERR     (0x00002000)
```

```
#define FLEXCAN_ESR_BIT0_ERR    (0x00004000)
```

```
#define FLEXCAN_ESR_BIT1_ERR    (0x00008000)
```

```
#define FLEXCAN_ESR_RWRN_INT    (0x00010000)
```

```
#define FLEXCAN_ESR_TWRN_INT    (0x00020000)
```

```
#define FLEXCAN_ESR_get_fault_code(esr) (((esr) & FLEXCAN_ESR_FLT_CONF_MASK)>>4)
```

```
#define CAN_ERROR_ACTIVE          0
```

```
#define CAN_ERROR_PASSIVE         1
```

```
#define CAN_ERROR_BUS_OFF         2
```

```
/* Bit definition for FLEXCAN_ESR2 */
```

```
#define FLEXCAN_ESR2_IMB          (0x00002000)

#define FLEXCAN_ESR2_VPS          (0x00004000)

#define FLEXCAN_ESR2_LTM          (0x007F0000L)

#define FLEXCAN_ESR2_LTM_BIT_NO    (16)

#define FLEXCAN_ESR2_LOSTRLF      (0x00000004)

#define FLEXCAN_ESR2_LOSTRMF      (0x00000002)

#define FLEXCAN_ESR2_IMEUF        (0x00000001)

#define FLEXCAN_get_LTM(esr2_value) (((esr2_value) &
(FLEXCAN_ESR2_LTM))>>(FLEXCAN_ESR2_LTM_BIT_NO))
```

```
/* Bit definitions and macros for FLEXCAN_IMASK1 */
```

```
#define FLEXCAN_IMASK1_BUF0M      (0x00000001)

#define FLEXCAN_IMASK1_BUF1M      (0x00000002)

#define FLEXCAN_IMASK1_BUF2M      (0x00000004)

#define FLEXCAN_IMASK1_BUF3M      (0x00000008)

#define FLEXCAN_IMASK1_BUF4M      (0x00000010)

#define FLEXCAN_IMASK1_BUF5M      (0x00000020)

#define FLEXCAN_IMASK1_BUF6M      (0x00000040)

#define FLEXCAN_IMASK1_BUF7M      (0x00000080)

#define FLEXCAN_IMASK1_BUF8M      (0x00000100)

#define FLEXCAN_IMASK1_BUF9M      (0x00000200)

#define FLEXCAN_IMASK1_BUF10M     (0x00000400)

#define FLEXCAN_IMASK1_BUF11M     (0x00000800)

#define FLEXCAN_IMASK1_BUF12M     (0x00001000)

#define FLEXCAN_IMASK1_BUF13M     (0x00002000)

#define FLEXCAN_IMASK1_BUF14M     (0x00004000)

#define FLEXCAN_IMASK1_BUF15M     (0x00008000)

#define FLEXCAN_IMASK1_BUF16M     (0x00010000)
```

```
#define FLEXCAN_IMASK1_BUF17M    (0x00020000)
#define FLEXCAN_IMASK1_BUF18M    (0x00040000)
#define FLEXCAN_IMASK1_BUF19M    (0x00080000)
#define FLEXCAN_IMASK1_BUF20M    (0x00100000)
#define FLEXCAN_IMASK1_BUF21M    (0x00200000)
#define FLEXCAN_IMASK1_BUF22M    (0x00400000)
#define FLEXCAN_IMASK1_BUF23M    (0x00800000)
#define FLEXCAN_IMASK1_BUF24M    (0x01000000)
#define FLEXCAN_IMASK1_BUF25M    (0x02000000)
#define FLEXCAN_IMASK1_BUF26M    (0x04000000)
#define FLEXCAN_IMASK1_BUF27M    (0x08000000)
#define FLEXCAN_IMASK1_BUF28M    (0x10000000)
#define FLEXCAN_IMASK1_BUF29M    (0x20000000)
#define FLEXCAN_IMASK1_BUF30M    (0x40000000)
#define FLEXCAN_IMASK1_BUF31M    (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_IFLAG1 */
```

```
#define FLEXCAN_IFLAG1_BUF0I     (0x00000001)
#define FLEXCAN_IFLAG1_BUF1I     (0x00000002)
#define FLEXCAN_IFLAG1_BUF2I     (0x00000004)
#define FLEXCAN_IFLAG1_BUF3I     (0x00000008)
#define FLEXCAN_IFLAG1_BUF4I     (0x00000010)
#define FLEXCAN_IFLAG1_BUF5I     (0x00000020)
#define FLEXCAN_IFLAG1_BUF6I     (0x00000040)
#define FLEXCAN_IFLAG1_BUF7I     (0x00000080)
#define FLEXCAN_IFLAG1_BUF8I     (0x00000100)
#define FLEXCAN_IFLAG1_BUF9I     (0x00000200)
#define FLEXCAN_IFLAG1_BUF10I    (0x00000400)
#define FLEXCAN_IFLAG1_BUF11I    (0x00000800)
```

```

#define FLEXCAN_IFLAG1_BUF12I    (0x00001000)
#define FLEXCAN_IFLAG1_BUF13I    (0x00002000)
#define FLEXCAN_IFLAG1_BUF14I    (0x00004000)
#define FLEXCAN_IFLAG1_BUF15I    (0x00008000)
#define FLEXCAN_IFLAG1_BUF16I    (0x00010000)
#define FLEXCAN_IFLAG1_BUF17I    (0x00020000)
#define FLEXCAN_IFLAG1_BUF18I    (0x00040000)
#define FLEXCAN_IFLAG1_BUF19I    (0x00080000)
#define FLEXCAN_IFLAG1_BUF20I    (0x00100000)
#define FLEXCAN_IFLAG1_BUF21I    (0x00200000)
#define FLEXCAN_IFLAG1_BUF22I    (0x00400000)
#define FLEXCAN_IFLAG1_BUF23I    (0x00800000)
#define FLEXCAN_IFLAG1_BUF24I    (0x01000000)
#define FLEXCAN_IFLAG1_BUF25I    (0x02000000)
#define FLEXCAN_IFLAG1_BUF26I    (0x04000000)
#define FLEXCAN_IFLAG1_BUF27I    (0x08000000)
#define FLEXCAN_IFLAG1_BUF28I    (0x10000000)
#define FLEXCAN_IFLAG1_BUF29I    (0x20000000)
#define FLEXCAN_IFLAG1_BUF30I    (0x40000000)
#define FLEXCAN_IFLAG1_BUF31I    (0x80000000)

```

```

/* Bit definitions and macros for FLEXCAN_MB_CS */

```

```

#define FLEXCAN_MB_CS_TIMESTAMP(x)  (((x)&0x0000FFFF)<<0)
#define FLEXCAN_MB_CS_TIMESTAMP_MASK (0x0000FFFFL)
#define FLEXCAN_MB_CS_LENGTH(x)    (((x)&0x0000000F)<<16)
#define FLEXCAN_MB_CS_RTR          (0x00100000)
#define FLEXCAN_MB_CS_IDE           (0x00200000)
#define FLEXCAN_MB_CS_SRR           (0x00400000)
#define FLEXCAN_MB_CS_CODE(x)       (((x)&0x0000000F)<<24)

```

```

#define FLEXCAN_MB_CS_CODE_MASK      (0x0F000000L)
#define FLEXCAN_MB_CS_DLC_MASK      (0x000F0000L)
#define FLEXCAN_MB_CODE_RX_INACTIVE  (0)
#define FLEXCAN_MB_CODE_RX_EMPTY    (4)
#define FLEXCAN_MB_CODE_RX_FULL     (2)
#define FLEXCAN_MB_CODE_RX_OVERRUN  (6)
#define FLEXCAN_MB_CODE_RX_BUSY     (1)

#define FLEXCAN_MB_CS_IDE_BIT_NO     (21)
#define FLEXCAN_MB_CS_RTR_BIT_NO     (20)
#define FLEXCAN_MB_CS_DLC_BIT_NO     (16)

#define FLEXCAN_MB_CODE_TX_INACTIVE  (8)
#define FLEXCAN_MB_CODE_TX_ABORT     (9)
#define FLEXCAN_MB_CODE_TX_ONCE     (0x0C)
#define FLEXCAN_MB_CODE_TX_RESPONSE (0x0A)
#define FLEXCAN_MB_CODE_TX_RESPONSE_TEMPO (0x0E)
#define FLEXCAN_get_code(cs)         (((cs) & FLEXCAN_MB_CS_CODE_MASK)>>24)
#define FLEXCAN_get_length(cs)       (((cs) & FLEXCAN_MB_CS_DLC_MASK)>>16)

/* Bit definitions and macros for FLEXCAN_MB_ID */
#define FLEXCAN_MB_ID_STD_MASK      (0x1FFC0000L)
#define FLEXCAN_MB_ID_EXT_MASK      (0x1FFFFFFFL)
#define FLEXCAN_MB_ID_IDEXT(x)      (((x)&0x0003FFFF)<<0)
#define FLEXCAN_MB_ID_IDSTD(x)      (((x)&0x000007FF)<<18)
#define FLEXCAN_MB_ID_PRIO(x)       (((x)&0x00000007)<<29)
#define FLEXCAN_MB_ID_PRIO_BIT_NO   (29)
#define FLEXCAN_MB_ID_STD_BIT_NO    (18)
#define FLEXCAN_MB_ID_EXT_BIT_NO    (0)

```

```
/* Bit definitions and macros for FLEXCAN_MB_WORD0 */
```

```
#define FLEXCAN_MB_WORD0_DATA3(x) (((x)&0x000000FF)<<0)
```

```
#define FLEXCAN_MB_WORD0_DATA2(x) (((x)&0x000000FF)<<8)
```

```
#define FLEXCAN_MB_WORD0_DATA1(x) (((x)&0x000000FF)<<16)
```

```
#define FLEXCAN_MB_WORD0_DATA0(x) (((x)&0x000000FF)<<24)
```

```
/* Bit definitions and macros for FLEXCAN_MB_WORD1 */
```

```
#define FLEXCAN_MB_WORD1_DATA7(x) (((x)&0x000000FF)<<0)
```

```
#define FLEXCAN_MB_WORD1_DATA6(x) (((x)&0x000000FF)<<8)
```

```
#define FLEXCAN_MB_WORD1_DATA5(x) (((x)&0x000000FF)<<16)
```

```
#define FLEXCAN_MB_WORD1_DATA4(x) (((x)&0x000000FF)<<24)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR0 */
```

```
#define FLEXCAN_RXIMR0_MI0      (0x00000001)
```

```
#define FLEXCAN_RXIMR0_MI1      (0x00000002)
```

```
#define FLEXCAN_RXIMR0_MI2      (0x00000004)
```

```
#define FLEXCAN_RXIMR0_MI3      (0x00000008)
```

```
#define FLEXCAN_RXIMR0_MI4      (0x00000010)
```

```
#define FLEXCAN_RXIMR0_MI5      (0x00000020)
```

```
#define FLEXCAN_RXIMR0_MI6      (0x00000040)
```

```
#define FLEXCAN_RXIMR0_MI7      (0x00000080)
```

```
#define FLEXCAN_RXIMR0_MI8      (0x00000100)
```

```
#define FLEXCAN_RXIMR0_MI9      (0x00000200)
```

```
#define FLEXCAN_RXIMR0_MI10     (0x00000400)
```

```
#define FLEXCAN_RXIMR0_MI11     (0x00000800)
```

```
#define FLEXCAN_RXIMR0_MI12     (0x00001000)
```

```
#define FLEXCAN_RXIMR0_MI13     (0x00002000)
```

```
#define FLEXCAN_RXIMR0_MI14    (0x00004000)
#define FLEXCAN_RXIMR0_MI15    (0x00008000)
#define FLEXCAN_RXIMR0_MI16    (0x00010000)
#define FLEXCAN_RXIMR0_MI17    (0x00020000)
#define FLEXCAN_RXIMR0_MI18    (0x00040000)
#define FLEXCAN_RXIMR0_MI19    (0x00080000)
#define FLEXCAN_RXIMR0_MI20    (0x00100000)
#define FLEXCAN_RXIMR0_MI21    (0x00200000)
#define FLEXCAN_RXIMR0_MI22    (0x00400000)
#define FLEXCAN_RXIMR0_MI23    (0x00800000)
#define FLEXCAN_RXIMR0_MI24    (0x01000000)
#define FLEXCAN_RXIMR0_MI25    (0x02000000)
#define FLEXCAN_RXIMR0_MI26    (0x04000000)
#define FLEXCAN_RXIMR0_MI27    (0x08000000)
#define FLEXCAN_RXIMR0_MI28    (0x10000000)
#define FLEXCAN_RXIMR0_MI29    (0x20000000)
#define FLEXCAN_RXIMR0_MI30    (0x40000000)
#define FLEXCAN_RXIMR0_MI31    (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR1 */
```

```
#define FLEXCAN_RXIMR1_MI0     (0x00000001)
#define FLEXCAN_RXIMR1_MI1     (0x00000002)
#define FLEXCAN_RXIMR1_MI2     (0x00000004)
#define FLEXCAN_RXIMR1_MI3     (0x00000008)
#define FLEXCAN_RXIMR1_MI4     (0x00000010)
#define FLEXCAN_RXIMR1_MI5     (0x00000020)
#define FLEXCAN_RXIMR1_MI6     (0x00000040)
#define FLEXCAN_RXIMR1_MI7     (0x00000080)
#define FLEXCAN_RXIMR1_MI8     (0x00000100)
```

```

#define FLEXCAN_RXIMR1_MI9      (0x00000200)
#define FLEXCAN_RXIMR1_MI10     (0x00000400)
#define FLEXCAN_RXIMR1_MI11     (0x00000800)
#define FLEXCAN_RXIMR1_MI12     (0x00001000)
#define FLEXCAN_RXIMR1_MI13     (0x00002000)
#define FLEXCAN_RXIMR1_MI14     (0x00004000)
#define FLEXCAN_RXIMR1_MI15     (0x00008000)
#define FLEXCAN_RXIMR1_MI16     (0x00010000)
#define FLEXCAN_RXIMR1_MI17     (0x00020000)
#define FLEXCAN_RXIMR1_MI18     (0x00040000)
#define FLEXCAN_RXIMR1_MI19     (0x00080000)
#define FLEXCAN_RXIMR1_MI20     (0x00100000)
#define FLEXCAN_RXIMR1_MI21     (0x00200000)
#define FLEXCAN_RXIMR1_MI22     (0x00400000)
#define FLEXCAN_RXIMR1_MI23     (0x00800000)
#define FLEXCAN_RXIMR1_MI24     (0x01000000)
#define FLEXCAN_RXIMR1_MI25     (0x02000000)
#define FLEXCAN_RXIMR1_MI26     (0x04000000)
#define FLEXCAN_RXIMR1_MI27     (0x08000000)
#define FLEXCAN_RXIMR1_MI28     (0x10000000)
#define FLEXCAN_RXIMR1_MI29     (0x20000000)
#define FLEXCAN_RXIMR1_MI30     (0x40000000)
#define FLEXCAN_RXIMR1_MI31     (0x80000000)

```

/* Bit definitions and macros for FLEXCAN_RXIMR2 */

```

#define FLEXCAN_RXIMR2_MI0      (0x00000001)
#define FLEXCAN_RXIMR2_MI1      (0x00000002)
#define FLEXCAN_RXIMR2_MI2      (0x00000004)
#define FLEXCAN_RXIMR2_MI3      (0x00000008)

```

```
#define FLEXCAN_RXIMR2_MI4      (0x00000010)
#define FLEXCAN_RXIMR2_MI5      (0x00000020)
#define FLEXCAN_RXIMR2_MI6      (0x00000040)
#define FLEXCAN_RXIMR2_MI7      (0x00000080)
#define FLEXCAN_RXIMR2_MI8      (0x00000100)
#define FLEXCAN_RXIMR2_MI9      (0x00000200)
#define FLEXCAN_RXIMR2_MI10     (0x00000400)
#define FLEXCAN_RXIMR2_MI11     (0x00000800)
#define FLEXCAN_RXIMR2_MI12     (0x00001000)
#define FLEXCAN_RXIMR2_MI13     (0x00002000)
#define FLEXCAN_RXIMR2_MI14     (0x00004000)
#define FLEXCAN_RXIMR2_MI15     (0x00008000)
#define FLEXCAN_RXIMR2_MI16     (0x00010000)
#define FLEXCAN_RXIMR2_MI17     (0x00020000)
#define FLEXCAN_RXIMR2_MI18     (0x00040000)
#define FLEXCAN_RXIMR2_MI19     (0x00080000)
#define FLEXCAN_RXIMR2_MI20     (0x00100000)
#define FLEXCAN_RXIMR2_MI21     (0x00200000)
#define FLEXCAN_RXIMR2_MI22     (0x00400000)
#define FLEXCAN_RXIMR2_MI23     (0x00800000)
#define FLEXCAN_RXIMR2_MI24     (0x01000000)
#define FLEXCAN_RXIMR2_MI25     (0x02000000)
#define FLEXCAN_RXIMR2_MI26     (0x04000000)
#define FLEXCAN_RXIMR2_MI27     (0x08000000)
#define FLEXCAN_RXIMR2_MI28     (0x10000000)
#define FLEXCAN_RXIMR2_MI29     (0x20000000)
#define FLEXCAN_RXIMR2_MI30     (0x40000000)
#define FLEXCAN_RXIMR2_MI31     (0x80000000)
```

/* Bit definitions and macros for FLEXCAN_RXIMR3 */

```
#define FLEXCAN_RXIMR3_MI0      (0x00000001)
#define FLEXCAN_RXIMR3_MI1      (0x00000002)
#define FLEXCAN_RXIMR3_MI2      (0x00000004)
#define FLEXCAN_RXIMR3_MI3      (0x00000008)
#define FLEXCAN_RXIMR3_MI4      (0x00000010)
#define FLEXCAN_RXIMR3_MI5      (0x00000020)
#define FLEXCAN_RXIMR3_MI6      (0x00000040)
#define FLEXCAN_RXIMR3_MI7      (0x00000080)
#define FLEXCAN_RXIMR3_MI8      (0x00000100)
#define FLEXCAN_RXIMR3_MI9      (0x00000200)
#define FLEXCAN_RXIMR3_MI10     (0x00000400)
#define FLEXCAN_RXIMR3_MI11     (0x00000800)
#define FLEXCAN_RXIMR3_MI12     (0x00001000)
#define FLEXCAN_RXIMR3_MI13     (0x00002000)
#define FLEXCAN_RXIMR3_MI14     (0x00004000)
#define FLEXCAN_RXIMR3_MI15     (0x00008000)
#define FLEXCAN_RXIMR3_MI16     (0x00010000)
#define FLEXCAN_RXIMR3_MI17     (0x00020000)
#define FLEXCAN_RXIMR3_MI18     (0x00040000)
#define FLEXCAN_RXIMR3_MI19     (0x00080000)
#define FLEXCAN_RXIMR3_MI20     (0x00100000)
#define FLEXCAN_RXIMR3_MI21     (0x00200000)
#define FLEXCAN_RXIMR3_MI22     (0x00400000)
#define FLEXCAN_RXIMR3_MI23     (0x00800000)
#define FLEXCAN_RXIMR3_MI24     (0x01000000)
#define FLEXCAN_RXIMR3_MI25     (0x02000000)
#define FLEXCAN_RXIMR3_MI26     (0x04000000)
#define FLEXCAN_RXIMR3_MI27     (0x08000000)
```

```
#define FLEXCAN_RXIMR3_MI28    (0x10000000)
#define FLEXCAN_RXIMR3_MI29    (0x20000000)
#define FLEXCAN_RXIMR3_MI30    (0x40000000)
#define FLEXCAN_RXIMR3_MI31    (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR4 */
```

```
#define FLEXCAN_RXIMR4_MI0      (0x00000001)
#define FLEXCAN_RXIMR4_MI1      (0x00000002)
#define FLEXCAN_RXIMR4_MI2      (0x00000004)
#define FLEXCAN_RXIMR4_MI3      (0x00000008)
#define FLEXCAN_RXIMR4_MI4      (0x00000010)
#define FLEXCAN_RXIMR4_MI5      (0x00000020)
#define FLEXCAN_RXIMR4_MI6      (0x00000040)
#define FLEXCAN_RXIMR4_MI7      (0x00000080)
#define FLEXCAN_RXIMR4_MI8      (0x00000100)
#define FLEXCAN_RXIMR4_MI9      (0x00000200)
#define FLEXCAN_RXIMR4_MI10     (0x00000400)
#define FLEXCAN_RXIMR4_MI11     (0x00000800)
#define FLEXCAN_RXIMR4_MI12     (0x00001000)
#define FLEXCAN_RXIMR4_MI13     (0x00002000)
#define FLEXCAN_RXIMR4_MI14     (0x00004000)
#define FLEXCAN_RXIMR4_MI15     (0x00008000)
#define FLEXCAN_RXIMR4_MI16     (0x00010000)
#define FLEXCAN_RXIMR4_MI17     (0x00020000)
#define FLEXCAN_RXIMR4_MI18     (0x00040000)
#define FLEXCAN_RXIMR4_MI19     (0x00080000)
#define FLEXCAN_RXIMR4_MI20     (0x00100000)
#define FLEXCAN_RXIMR4_MI21     (0x00200000)
#define FLEXCAN_RXIMR4_MI22     (0x00400000)
```

```
#define FLEXCAN_RXIMR4_MI23      (0x00800000)
#define FLEXCAN_RXIMR4_MI24      (0x01000000)
#define FLEXCAN_RXIMR4_MI25      (0x02000000)
#define FLEXCAN_RXIMR4_MI26      (0x04000000)
#define FLEXCAN_RXIMR4_MI27      (0x08000000)
#define FLEXCAN_RXIMR4_MI28      (0x10000000)
#define FLEXCAN_RXIMR4_MI29      (0x20000000)
#define FLEXCAN_RXIMR4_MI30      (0x40000000)
#define FLEXCAN_RXIMR4_MI31      (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR5 */
```

```
#define FLEXCAN_RXIMR5_MI0       (0x00000001)
#define FLEXCAN_RXIMR5_MI1       (0x00000002)
#define FLEXCAN_RXIMR5_MI2       (0x00000004)
#define FLEXCAN_RXIMR5_MI3       (0x00000008)
#define FLEXCAN_RXIMR5_MI4       (0x00000010)
#define FLEXCAN_RXIMR5_MI5       (0x00000020)
#define FLEXCAN_RXIMR5_MI6       (0x00000040)
#define FLEXCAN_RXIMR5_MI7       (0x00000080)
#define FLEXCAN_RXIMR5_MI8       (0x00000100)
#define FLEXCAN_RXIMR5_MI9       (0x00000200)
#define FLEXCAN_RXIMR5_MI10      (0x00000400)
#define FLEXCAN_RXIMR5_MI11      (0x00000800)
#define FLEXCAN_RXIMR5_MI12      (0x00001000)
#define FLEXCAN_RXIMR5_MI13      (0x00002000)
#define FLEXCAN_RXIMR5_MI14      (0x00004000)
#define FLEXCAN_RXIMR5_MI15      (0x00008000)
#define FLEXCAN_RXIMR5_MI16      (0x00010000)
#define FLEXCAN_RXIMR5_MI17      (0x00020000)
```

```
#define FLEXCAN_RXIMR5_MI18    (0x00040000)
#define FLEXCAN_RXIMR5_MI19    (0x00080000)
#define FLEXCAN_RXIMR5_MI20    (0x00100000)
#define FLEXCAN_RXIMR5_MI21    (0x00200000)
#define FLEXCAN_RXIMR5_MI22    (0x00400000)
#define FLEXCAN_RXIMR5_MI23    (0x00800000)
#define FLEXCAN_RXIMR5_MI24    (0x01000000)
#define FLEXCAN_RXIMR5_MI25    (0x02000000)
#define FLEXCAN_RXIMR5_MI26    (0x04000000)
#define FLEXCAN_RXIMR5_MI27    (0x08000000)
#define FLEXCAN_RXIMR5_MI28    (0x10000000)
#define FLEXCAN_RXIMR5_MI29    (0x20000000)
#define FLEXCAN_RXIMR5_MI30    (0x40000000)
#define FLEXCAN_RXIMR5_MI31    (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR6 */
```

```
#define FLEXCAN_RXIMR6_MI0     (0x00000001)
#define FLEXCAN_RXIMR6_MI1     (0x00000002)
#define FLEXCAN_RXIMR6_MI2     (0x00000004)
#define FLEXCAN_RXIMR6_MI3     (0x00000008)
#define FLEXCAN_RXIMR6_MI4     (0x00000010)
#define FLEXCAN_RXIMR6_MI5     (0x00000020)
#define FLEXCAN_RXIMR6_MI6     (0x00000040)
#define FLEXCAN_RXIMR6_MI7     (0x00000080)
#define FLEXCAN_RXIMR6_MI8     (0x00000100)
#define FLEXCAN_RXIMR6_MI9     (0x00000200)
#define FLEXCAN_RXIMR6_MI10    (0x00000400)
#define FLEXCAN_RXIMR6_MI11    (0x00000800)
#define FLEXCAN_RXIMR6_MI12    (0x00001000)
```

```
#define FLEXCAN_RXIMR6_MI13    (0x00002000)
#define FLEXCAN_RXIMR6_MI14    (0x00004000)
#define FLEXCAN_RXIMR6_MI15    (0x00008000)
#define FLEXCAN_RXIMR6_MI16    (0x00010000)
#define FLEXCAN_RXIMR6_MI17    (0x00020000)
#define FLEXCAN_RXIMR6_MI18    (0x00040000)
#define FLEXCAN_RXIMR6_MI19    (0x00080000)
#define FLEXCAN_RXIMR6_MI20    (0x00100000)
#define FLEXCAN_RXIMR6_MI21    (0x00200000)
#define FLEXCAN_RXIMR6_MI22    (0x00400000)
#define FLEXCAN_RXIMR6_MI23    (0x00800000)
#define FLEXCAN_RXIMR6_MI24    (0x01000000)
#define FLEXCAN_RXIMR6_MI25    (0x02000000)
#define FLEXCAN_RXIMR6_MI26    (0x04000000)
#define FLEXCAN_RXIMR6_MI27    (0x08000000)
#define FLEXCAN_RXIMR6_MI28    (0x10000000)
#define FLEXCAN_RXIMR6_MI29    (0x20000000)
#define FLEXCAN_RXIMR6_MI30    (0x40000000)
#define FLEXCAN_RXIMR6_MI31    (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR7 */
```

```
#define FLEXCAN_RXIMR7_MI0     (0x00000001)
#define FLEXCAN_RXIMR7_MI1     (0x00000002)
#define FLEXCAN_RXIMR7_MI2     (0x00000004)
#define FLEXCAN_RXIMR7_MI3     (0x00000008)
#define FLEXCAN_RXIMR7_MI4     (0x00000010)
#define FLEXCAN_RXIMR7_MI5     (0x00000020)
#define FLEXCAN_RXIMR7_MI6     (0x00000040)
#define FLEXCAN_RXIMR7_MI7     (0x00000080)
```

```
#define FLEXCAN_RXIMR7_MI8      (0x00000100)
#define FLEXCAN_RXIMR7_MI9      (0x00000200)
#define FLEXCAN_RXIMR7_MI10     (0x00000400)
#define FLEXCAN_RXIMR7_MI11     (0x00000800)
#define FLEXCAN_RXIMR7_MI12     (0x00001000)
#define FLEXCAN_RXIMR7_MI13     (0x00002000)
#define FLEXCAN_RXIMR7_MI14     (0x00004000)
#define FLEXCAN_RXIMR7_MI15     (0x00008000)
#define FLEXCAN_RXIMR7_MI16     (0x00010000)
#define FLEXCAN_RXIMR7_MI17     (0x00020000)
#define FLEXCAN_RXIMR7_MI18     (0x00040000)
#define FLEXCAN_RXIMR7_MI19     (0x00080000)
#define FLEXCAN_RXIMR7_MI20     (0x00100000)
#define FLEXCAN_RXIMR7_MI21     (0x00200000)
#define FLEXCAN_RXIMR7_MI22     (0x00400000)
#define FLEXCAN_RXIMR7_MI23     (0x00800000)
#define FLEXCAN_RXIMR7_MI24     (0x01000000)
#define FLEXCAN_RXIMR7_MI25     (0x02000000)
#define FLEXCAN_RXIMR7_MI26     (0x04000000)
#define FLEXCAN_RXIMR7_MI27     (0x08000000)
#define FLEXCAN_RXIMR7_MI28     (0x10000000)
#define FLEXCAN_RXIMR7_MI29     (0x20000000)
#define FLEXCAN_RXIMR7_MI30     (0x40000000)
#define FLEXCAN_RXIMR7_MI31     (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR8 */
```

```
#define FLEXCAN_RXIMR8_MI0      (0x00000001)
#define FLEXCAN_RXIMR8_MI1      (0x00000002)
#define FLEXCAN_RXIMR8_MI2      (0x00000004)
```

```
#define FLEXCAN_RXIMR8_MI3      (0x00000008)
#define FLEXCAN_RXIMR8_MI4      (0x00000010)
#define FLEXCAN_RXIMR8_MI5      (0x00000020)
#define FLEXCAN_RXIMR8_MI6      (0x00000040)
#define FLEXCAN_RXIMR8_MI7      (0x00000080)
#define FLEXCAN_RXIMR8_MI8      (0x00000100)
#define FLEXCAN_RXIMR8_MI9      (0x00000200)
#define FLEXCAN_RXIMR8_MI10     (0x00000400)
#define FLEXCAN_RXIMR8_MI11     (0x00000800)
#define FLEXCAN_RXIMR8_MI12     (0x00001000)
#define FLEXCAN_RXIMR8_MI13     (0x00002000)
#define FLEXCAN_RXIMR8_MI14     (0x00004000)
#define FLEXCAN_RXIMR8_MI15     (0x00008000)
#define FLEXCAN_RXIMR8_MI16     (0x00010000)
#define FLEXCAN_RXIMR8_MI17     (0x00020000)
#define FLEXCAN_RXIMR8_MI18     (0x00040000)
#define FLEXCAN_RXIMR8_MI19     (0x00080000)
#define FLEXCAN_RXIMR8_MI20     (0x00100000)
#define FLEXCAN_RXIMR8_MI21     (0x00200000)
#define FLEXCAN_RXIMR8_MI22     (0x00400000)
#define FLEXCAN_RXIMR8_MI23     (0x00800000)
#define FLEXCAN_RXIMR8_MI24     (0x01000000)
#define FLEXCAN_RXIMR8_MI25     (0x02000000)
#define FLEXCAN_RXIMR8_MI26     (0x04000000)
#define FLEXCAN_RXIMR8_MI27     (0x08000000)
#define FLEXCAN_RXIMR8_MI28     (0x10000000)
#define FLEXCAN_RXIMR8_MI29     (0x20000000)
#define FLEXCAN_RXIMR8_MI30     (0x40000000)
#define FLEXCAN_RXIMR8_MI31     (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR9 */
```

```
#define FLEXCAN_RXIMR9_MI0      (0x00000001)
#define FLEXCAN_RXIMR9_MI1      (0x00000002)
#define FLEXCAN_RXIMR9_MI2      (0x00000004)
#define FLEXCAN_RXIMR9_MI3      (0x00000008)
#define FLEXCAN_RXIMR9_MI4      (0x00000010)
#define FLEXCAN_RXIMR9_MI5      (0x00000020)
#define FLEXCAN_RXIMR9_MI6      (0x00000040)
#define FLEXCAN_RXIMR9_MI7      (0x00000080)
#define FLEXCAN_RXIMR9_MI8      (0x00000100)
#define FLEXCAN_RXIMR9_MI9      (0x00000200)
#define FLEXCAN_RXIMR9_MI10     (0x00000400)
#define FLEXCAN_RXIMR9_MI11     (0x00000800)
#define FLEXCAN_RXIMR9_MI12     (0x00001000)
#define FLEXCAN_RXIMR9_MI13     (0x00002000)
#define FLEXCAN_RXIMR9_MI14     (0x00004000)
#define FLEXCAN_RXIMR9_MI15     (0x00008000)
#define FLEXCAN_RXIMR9_MI16     (0x00010000)
#define FLEXCAN_RXIMR9_MI17     (0x00020000)
#define FLEXCAN_RXIMR9_MI18     (0x00040000)
#define FLEXCAN_RXIMR9_MI19     (0x00080000)
#define FLEXCAN_RXIMR9_MI20     (0x00100000)
#define FLEXCAN_RXIMR9_MI21     (0x00200000)
#define FLEXCAN_RXIMR9_MI22     (0x00400000)
#define FLEXCAN_RXIMR9_MI23     (0x00800000)
#define FLEXCAN_RXIMR9_MI24     (0x01000000)
#define FLEXCAN_RXIMR9_MI25     (0x02000000)
#define FLEXCAN_RXIMR9_MI26     (0x04000000)
```

```
#define FLEXCAN_RXIMR9_MI27      (0x08000000)
#define FLEXCAN_RXIMR9_MI28      (0x10000000)
#define FLEXCAN_RXIMR9_MI29      (0x20000000)
#define FLEXCAN_RXIMR9_MI30      (0x40000000)
#define FLEXCAN_RXIMR9_MI31      (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR10 */
```

```
#define FLEXCAN_RXIMR10_MI0      (0x00000001)
#define FLEXCAN_RXIMR10_MI1      (0x00000002)
#define FLEXCAN_RXIMR10_MI2      (0x00000004)
#define FLEXCAN_RXIMR10_MI3      (0x00000008)
#define FLEXCAN_RXIMR10_MI4      (0x00000010)
#define FLEXCAN_RXIMR10_MI5      (0x00000020)
#define FLEXCAN_RXIMR10_MI6      (0x00000040)
#define FLEXCAN_RXIMR10_MI7      (0x00000080)
#define FLEXCAN_RXIMR10_MI8      (0x00000100)
#define FLEXCAN_RXIMR10_MI9      (0x00000200)
#define FLEXCAN_RXIMR10_MI10     (0x00000400)
#define FLEXCAN_RXIMR10_MI11     (0x00000800)
#define FLEXCAN_RXIMR10_MI12     (0x00001000)
#define FLEXCAN_RXIMR10_MI13     (0x00002000)
#define FLEXCAN_RXIMR10_MI14     (0x00004000)
#define FLEXCAN_RXIMR10_MI15     (0x00008000)
#define FLEXCAN_RXIMR10_MI16     (0x00010000)
#define FLEXCAN_RXIMR10_MI17     (0x00020000)
#define FLEXCAN_RXIMR10_MI18     (0x00040000)
#define FLEXCAN_RXIMR10_MI19     (0x00080000)
#define FLEXCAN_RXIMR10_MI20     (0x00100000)
#define FLEXCAN_RXIMR10_MI21     (0x00200000)
```

```
#define FLEXCAN_RXIMR10_MI22    (0x00400000)
#define FLEXCAN_RXIMR10_MI23    (0x00800000)
#define FLEXCAN_RXIMR10_MI24    (0x01000000)
#define FLEXCAN_RXIMR10_MI25    (0x02000000)
#define FLEXCAN_RXIMR10_MI26    (0x04000000)
#define FLEXCAN_RXIMR10_MI27    (0x08000000)
#define FLEXCAN_RXIMR10_MI28    (0x10000000)
#define FLEXCAN_RXIMR10_MI29    (0x20000000)
#define FLEXCAN_RXIMR10_MI30    (0x40000000)
#define FLEXCAN_RXIMR10_MI31    (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR11 */
```

```
#define FLEXCAN_RXIMR11_MI0      (0x00000001)
#define FLEXCAN_RXIMR11_MI1      (0x00000002)
#define FLEXCAN_RXIMR11_MI2      (0x00000004)
#define FLEXCAN_RXIMR11_MI3      (0x00000008)
#define FLEXCAN_RXIMR11_MI4      (0x00000010)
#define FLEXCAN_RXIMR11_MI5      (0x00000020)
#define FLEXCAN_RXIMR11_MI6      (0x00000040)
#define FLEXCAN_RXIMR11_MI7      (0x00000080)
#define FLEXCAN_RXIMR11_MI8      (0x00000100)
#define FLEXCAN_RXIMR11_MI9      (0x00000200)
#define FLEXCAN_RXIMR11_MI10     (0x00000400)
#define FLEXCAN_RXIMR11_MI11     (0x00000800)
#define FLEXCAN_RXIMR11_MI12     (0x00001000)
#define FLEXCAN_RXIMR11_MI13     (0x00002000)
#define FLEXCAN_RXIMR11_MI14     (0x00004000)
#define FLEXCAN_RXIMR11_MI15     (0x00008000)
#define FLEXCAN_RXIMR11_MI16     (0x00010000)
```

```
#define FLEXCAN_RXIMR11_MI17    (0x00020000)
#define FLEXCAN_RXIMR11_MI18    (0x00040000)
#define FLEXCAN_RXIMR11_MI19    (0x00080000)
#define FLEXCAN_RXIMR11_MI20    (0x00100000)
#define FLEXCAN_RXIMR11_MI21    (0x00200000)
#define FLEXCAN_RXIMR11_MI22    (0x00400000)
#define FLEXCAN_RXIMR11_MI23    (0x00800000)
#define FLEXCAN_RXIMR11_MI24    (0x01000000)
#define FLEXCAN_RXIMR11_MI25    (0x02000000)
#define FLEXCAN_RXIMR11_MI26    (0x04000000)
#define FLEXCAN_RXIMR11_MI27    (0x08000000)
#define FLEXCAN_RXIMR11_MI28    (0x10000000)
#define FLEXCAN_RXIMR11_MI29    (0x20000000)
#define FLEXCAN_RXIMR11_MI30    (0x40000000)
#define FLEXCAN_RXIMR11_MI31    (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR12 */
```

```
#define FLEXCAN_RXIMR12_MI0     (0x00000001)
#define FLEXCAN_RXIMR12_MI1     (0x00000002)
#define FLEXCAN_RXIMR12_MI2     (0x00000004)
#define FLEXCAN_RXIMR12_MI3     (0x00000008)
#define FLEXCAN_RXIMR12_MI4     (0x00000010)
#define FLEXCAN_RXIMR12_MI5     (0x00000020)
#define FLEXCAN_RXIMR12_MI6     (0x00000040)
#define FLEXCAN_RXIMR12_MI7     (0x00000080)
#define FLEXCAN_RXIMR12_MI8     (0x00000100)
#define FLEXCAN_RXIMR12_MI9     (0x00000200)
#define FLEXCAN_RXIMR12_MI10    (0x00000400)
#define FLEXCAN_RXIMR12_MI11    (0x00000800)
```

```
#define FLEXCAN_RXIMR12_MI12    (0x00001000)
#define FLEXCAN_RXIMR12_MI13    (0x00002000)
#define FLEXCAN_RXIMR12_MI14    (0x00004000)
#define FLEXCAN_RXIMR12_MI15    (0x00008000)
#define FLEXCAN_RXIMR12_MI16    (0x00010000)
#define FLEXCAN_RXIMR12_MI17    (0x00020000)
#define FLEXCAN_RXIMR12_MI18    (0x00040000)
#define FLEXCAN_RXIMR12_MI19    (0x00080000)
#define FLEXCAN_RXIMR12_MI20    (0x00100000)
#define FLEXCAN_RXIMR12_MI21    (0x00200000)
#define FLEXCAN_RXIMR12_MI22    (0x00400000)
#define FLEXCAN_RXIMR12_MI23    (0x00800000)
#define FLEXCAN_RXIMR12_MI24    (0x01000000)
#define FLEXCAN_RXIMR12_MI25    (0x02000000)
#define FLEXCAN_RXIMR12_MI26    (0x04000000)
#define FLEXCAN_RXIMR12_MI27    (0x08000000)
#define FLEXCAN_RXIMR12_MI28    (0x10000000)
#define FLEXCAN_RXIMR12_MI29    (0x20000000)
#define FLEXCAN_RXIMR12_MI30    (0x40000000)
#define FLEXCAN_RXIMR12_MI31    (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR13 */
```

```
#define FLEXCAN_RXIMR13_MI0      (0x00000001)
#define FLEXCAN_RXIMR13_MI1      (0x00000002)
#define FLEXCAN_RXIMR13_MI2      (0x00000004)
#define FLEXCAN_RXIMR13_MI3      (0x00000008)
#define FLEXCAN_RXIMR13_MI4      (0x00000010)
#define FLEXCAN_RXIMR13_MI5      (0x00000020)
#define FLEXCAN_RXIMR13_MI6      (0x00000040)
```

```
#define FLEXCAN_RXIMR13_MI7      (0x00000080)
#define FLEXCAN_RXIMR13_MI8      (0x00000100)
#define FLEXCAN_RXIMR13_MI9      (0x00000200)
#define FLEXCAN_RXIMR13_MI10     (0x00000400)
#define FLEXCAN_RXIMR13_MI11     (0x00000800)
#define FLEXCAN_RXIMR13_MI12     (0x00001000)
#define FLEXCAN_RXIMR13_MI13     (0x00002000)
#define FLEXCAN_RXIMR13_MI14     (0x00004000)
#define FLEXCAN_RXIMR13_MI15     (0x00008000)
#define FLEXCAN_RXIMR13_MI16     (0x00010000)
#define FLEXCAN_RXIMR13_MI17     (0x00020000)
#define FLEXCAN_RXIMR13_MI18     (0x00040000)
#define FLEXCAN_RXIMR13_MI19     (0x00080000)
#define FLEXCAN_RXIMR13_MI20     (0x00100000)
#define FLEXCAN_RXIMR13_MI21     (0x00200000)
#define FLEXCAN_RXIMR13_MI22     (0x00400000)
#define FLEXCAN_RXIMR13_MI23     (0x00800000)
#define FLEXCAN_RXIMR13_MI24     (0x01000000)
#define FLEXCAN_RXIMR13_MI25     (0x02000000)
#define FLEXCAN_RXIMR13_MI26     (0x04000000)
#define FLEXCAN_RXIMR13_MI27     (0x08000000)
#define FLEXCAN_RXIMR13_MI28     (0x10000000)
#define FLEXCAN_RXIMR13_MI29     (0x20000000)
#define FLEXCAN_RXIMR13_MI30     (0x40000000)
#define FLEXCAN_RXIMR13_MI31     (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR14 */
```

```
#define FLEXCAN_RXIMR14_MI0      (0x00000001)
#define FLEXCAN_RXIMR14_MI1      (0x00000002)
```

```
#define FLEXCAN_RXIMR14_MI2      (0x00000004)
#define FLEXCAN_RXIMR14_MI3      (0x00000008)
#define FLEXCAN_RXIMR14_MI4      (0x00000010)
#define FLEXCAN_RXIMR14_MI5      (0x00000020)
#define FLEXCAN_RXIMR14_MI6      (0x00000040)
#define FLEXCAN_RXIMR14_MI7      (0x00000080)
#define FLEXCAN_RXIMR14_MI8      (0x00000100)
#define FLEXCAN_RXIMR14_MI9      (0x00000200)
#define FLEXCAN_RXIMR14_MI10     (0x00000400)
#define FLEXCAN_RXIMR14_MI11     (0x00000800)
#define FLEXCAN_RXIMR14_MI12     (0x00001000)
#define FLEXCAN_RXIMR14_MI13     (0x00002000)
#define FLEXCAN_RXIMR14_MI14     (0x00004000)
#define FLEXCAN_RXIMR14_MI15     (0x00008000)
#define FLEXCAN_RXIMR14_MI16     (0x00010000)
#define FLEXCAN_RXIMR14_MI17     (0x00020000)
#define FLEXCAN_RXIMR14_MI18     (0x00040000)
#define FLEXCAN_RXIMR14_MI19     (0x00080000)
#define FLEXCAN_RXIMR14_MI20     (0x00100000)
#define FLEXCAN_RXIMR14_MI21     (0x00200000)
#define FLEXCAN_RXIMR14_MI22     (0x00400000)
#define FLEXCAN_RXIMR14_MI23     (0x00800000)
#define FLEXCAN_RXIMR14_MI24     (0x01000000)
#define FLEXCAN_RXIMR14_MI25     (0x02000000)
#define FLEXCAN_RXIMR14_MI26     (0x04000000)
#define FLEXCAN_RXIMR14_MI27     (0x08000000)
#define FLEXCAN_RXIMR14_MI28     (0x10000000)
#define FLEXCAN_RXIMR14_MI29     (0x20000000)
#define FLEXCAN_RXIMR14_MI30     (0x40000000)
```

```
#define FLEXCAN_RXIMR14_MI31      (0x80000000)
```

```
/* Bit definitions and macros for FLEXCAN_RXIMR15 */
```

```
#define FLEXCAN_RXIMR15_MI0      (0x00000001)
```

```
#define FLEXCAN_RXIMR15_MI1      (0x00000002)
```

```
#define FLEXCAN_RXIMR15_MI2      (0x00000004)
```

```
#define FLEXCAN_RXIMR15_MI3      (0x00000008)
```

```
#define FLEXCAN_RXIMR15_MI4      (0x00000010)
```

```
#define FLEXCAN_RXIMR15_MI5      (0x00000020)
```

```
#define FLEXCAN_RXIMR15_MI6      (0x00000040)
```

```
#define FLEXCAN_RXIMR15_MI7      (0x00000080)
```

```
#define FLEXCAN_RXIMR15_MI8      (0x00000100)
```

```
#define FLEXCAN_RXIMR15_MI9      (0x00000200)
```

```
#define FLEXCAN_RXIMR15_MI10     (0x00000400)
```

```
#define FLEXCAN_RXIMR15_MI11     (0x00000800)
```

```
#define FLEXCAN_RXIMR15_MI12     (0x00001000)
```

```
#define FLEXCAN_RXIMR15_MI13     (0x00002000)
```

```
#define FLEXCAN_RXIMR15_MI14     (0x00004000)
```

```
#define FLEXCAN_RXIMR15_MI15     (0x00008000)
```

```
#define FLEXCAN_RXIMR15_MI16     (0x00010000)
```

```
#define FLEXCAN_RXIMR15_MI17     (0x00020000)
```

```
#define FLEXCAN_RXIMR15_MI18     (0x00040000)
```

```
#define FLEXCAN_RXIMR15_MI19     (0x00080000)
```

```
#define FLEXCAN_RXIMR15_MI20     (0x00100000)
```

```
#define FLEXCAN_RXIMR15_MI21     (0x00200000)
```

```
#define FLEXCAN_RXIMR15_MI22     (0x00400000)
```

```
#define FLEXCAN_RXIMR15_MI23     (0x00800000)
```

```
#define FLEXCAN_RXIMR15_MI24     (0x01000000)
```

```
#define FLEXCAN_RXIMR15_MI25     (0x02000000)
```

```

#define FLEXCAN_RXIMR15_MI26      (0x04000000)
#define FLEXCAN_RXIMR15_MI27      (0x08000000)
#define FLEXCAN_RXIMR15_MI28      (0x10000000)
#define FLEXCAN_RXIMR15_MI29      (0x20000000)
#define FLEXCAN_RXIMR15_MI30      (0x40000000)
#define FLEXCAN_RXIMR15_MI31      (0x80000000)

```

/* Bit definitions for CRC register */

```

#define FLEXCAN_CRCCR_MBCRC_BIT_NO      (16)
#define FLEXCAN_CRCCR_MBCRC_MASK        (0x007F0000)
#define FLEXCAN_CRCCR_CRC_BIT_NO        (0)
#define FLEXCAN_CRCCR_CRC_MASK          (0x00007FFF)

```

/* Bit definition for Individual Matching Elements Update Register (IMEUR) */

```

#define FLEXCAN_IMEUR_IMEUP_MASK        (0x0000007F)
#define FLEXCAN_IMEUR_IMEUP_BIT_NO      (0)
#define FLEXCAN_IMEUR_IMEUREQ_MASK      (0x00000100)
#define FLEXCAN_IMEUR_IMEUACK_MASK      (0x00000200)
#define FLEXCAN_Set_IMEUP(imeur, imeup)  imeur = (imeur & ~(FLEXCAN_IMEUR_IMEUP_MASK)) |
(imeup & FLEXCAN_IMEUR_IMEUP_MASK)
#define FLEXCAN_Get_IMEUP(imeur)         (imeur & FLEXCAN_IMEUR_IMEUP_MASK)

```

/* Bit definition for Lost Rx Frames Register (LRFR)

```

*/
#define FLEXCAN_LRFR_LOSTRLP_MASK(0x007F0000)
#define FLEXCAN_LRFR_LFIFOMTC_MASK      (0x00008000)
#define FLEXCAN_LRFR_LOSTRMP_MASK      (0x000001FF)
#define FLEXCAN_LRFR_LOSTRLP_BIT_NO     (16)
#define FLEXCAN_LRFR_LFIFOMTC_BIT_NO    (15)

```

```
#define FLEXCAN_LRFR_LOSTRMP_BIT_NO          (0)

#define FLEXCAN_Get_LostMBLocked(lrfr)      ((lrfr &
FLEXCAN_LRFR_LOSTRLP_MASK)>>(FLEXCAN_LRFR_LOSTRLP_BIT_NO))

#define FLEXCAN_Get_LostMBUpdated(lrfr)     ((lrfr & FLEXCAN_LRFR_LOSTRMP_MASK))
```

```
/* Bit definition for Memory Error Control Register */
```

```
#define FLEXCAN_MECR_NCEFAFRZ_MASK          (0x00000080)
#define FLEXCAN_MECR_RERRDIS_MASK          (0x00000100)
#define FLEXCAN_MECR_EXTERRIE_MAKS         (0x00002000)
#define FLEXCAN_MECR_FAERRIE_MAKS          (0x00004000)
#define FLEXCAN_MECR_HAERRIE_MAKS          (0x00008000)
#define FLEXCAN_MECR_CEI_MSK_MAKS          (0x00010000)
#define FLEXCAN_MECR_FANCEI_MSK_MAKS       (0x00040000)
#define FLEXCAN_MECR_HANCEI_MSK_MAKS       (0x00080000)
#define FLEXCAN_MECR_ECRWRDIS_MSK_MAKS     (0x80000000)
```

```
/* Bit definition for Error Report Address Register (RERRAR) */
```

```
#define FLEXCAN_RERRAR_NCE_MASK             (0x01000000)
#define FLEXCAN_RERRAR_SAID_MASK           (0x00070000)
#define FLEXCAN_ERRADDR_MASK               (0x00003FFF)
```

```
/* Bit definition for Error Report Syndrome Register (RERRSYNR) */
```

```
#define FLEXCAN_RERRSYNR_BE3_MASK           (0x80000000)
#define FLEXCAN_RERRSYNR_SYND3_MASK        (0x1F000000)
#define FLEXCAN_RERRSYNR_SYND3_BIT_NO      (24)
#define FLEXCAN_RERRSYNR_BE2_MASK           (0x00800000)
#define FLEXCAN_RERRSYNR_SYND2_MASK        (0x001F0000)
#define FLEXCAN_RERRSYNR_SYND2_BIT_NO      (16)
#define FLEXCAN_RERRSYNR_BE1_MASK           (0x00008000)
```

```

#define FLEXCAN_RERRSYNR_SYND1_MASK    (0x00001F00)

#define FLEXCAN_RERRSYNR_SYND1_BIT_NO  (8)

#define FLEXCAN_RERRSYNR_BE0_MASK      (0x00000080)

#define FLEXCAN_RERRSYNR_SYND0_MASK    (0x0000001F)

#define FLEXCAN_RERRSYNR_SYND0_BIT_NO  (0)


#define FLEXCAN_RERRSYNR_check_BEn_Bit(errsynr,n) (errsynr &
FLEXCAN_RERRSYNR_BE##n##_MASK)

#define FLEXCAN_RERRSYNR_get_SYNDn(errsynr,n) (errsynr & FLEXCAN_RERRSYNR_SYND##n##_MASK)

#define FLEXCAN_RERRSYNR_check_SYNDn_Bit(errsynr,n) ((errsynr &
FLEXCAN_RERRSYNR_SYND##n##_MASK)>>FLEXCAN_RERRSYNR_SYND##n##_BIT_NO)


/* Bit definition for Error Status Register (ERRSR) */
#define FLEXCAN_ERRSR_CEIOF_MASK  (0x00000001)

#define FLEXCAN_ERRSR_FANCEIOF_MASK  (0x00000004)

#define FLEXCAN_ERRSR_HANCEIOF_MASK  (0x00000008)

#define FLEXCAN_ERRSR_CEIF_MASK      (0x00010000)

#define FLEXCAN_ERRSR_FANCEIF_MASK   (0x00040000)

#define FLEXCAN_ERRSR_HANCEIF_MASK   (0x00080000)


/*****/

#endif // __KINETIS_FLEXCAN_H

```

1.1.12 Input_handler.h

Below:

```

/*

* This .h file defines function and class prototypes that are referenced in the main program file

*/

```

```
#define PI_GPIO 28
```

```
// Initialize PWM input interrupts
```

```
void initPWMin();
```

```
//-----
```

```
// Poll URF data
```

```
void initURF();
```

```
void pollURF();
```

```
//-----
```

1.1.13 Input_handler.cpp

Below:

```
/*
```

```
 * This .cpp file defines the content of functions to be called by the main routine
```

```
*/
```

```
// Allow access to arduino specific datatypes
```

```
#include <arduino.h>
```

```
// Define variables and constants. Voltatile is used for variables that are only modified in ISRs
```

```
// as the compiler is liable to treat them as constants.
```

```
// RX input-----
```

```
// Pin numbers for input
```

```
#define THROTTLEPIN 22
```

```

#define STEERPIN 23

// Variables for throttle start time
uint16_t THR_st;

// Variables for steering start time
uint16_t ST_st;

// throttle input value (ranges from ~-500 to 500) / defined in main
extern volatile int16_t THR_in;

//steering input value (ranges from ~-500 to 500) / defined in main
extern volatile int16_t ST_in;

//-----

// URF-----

// Pin numbers for IO
#define URF_in 17
#define URF_trig 16
extern uint16_t URF_dist;

//-----

// Specify contents of functions to be called by main routine

// Functions to initialize URF and output URF value (configured as analog input)-----
-----

void initURF(){
    // Set data directions
    pinMode(URF_in, INPUT);
    pinMode(URF_trig, OUTPUT);

    // Put URF in continuous mode
    digitalWrite(URF_trig, HIGH);

```

```
}
```

```
void pollURF(){
```

```
    URF_dist = analogRead(URF_in) * 5;
```

```
    Serial.println(analogRead(URF_in));
```

```
}
```

```
//-----
```

```
// Steering ISR to determine the time difference between rising edge and falling edge-----
```

```
-----
```

```
// Called and run upon the change of the input pin tied to the receiver
```

```
void ST_int() {
```

```
    int16_t x = 0;
```

```
    // If the pin is now HIGH, record the current time
```

```
    if (digitalRead(STEERPIN) == HIGH)
```

```
    {
```

```
        ST_st = micros();
```

```
    }
```

```
    else
```

```
    {
```

```
        // If the pin is now low, calculate, adjust, and output the time
```

```
        // between when the timer was started and now. Range between -500 and 500, approximately
```

```
        x = (micros() - ST_st - 1500);
```

```
        if (x < 530 && x > -530) //allow for discrepancies
```

```
        {
```

```
            if (x > 500)
```

```
            {
```

```
                x = 400;
```

```

    }

    if (x < -500)
    {
        x = -400;
    }

    ST_in = x;
    //Serial.println(ST_in);
}
}
}

// Functionally Identical to steering ISR
void THR_int() {
    int16_t x;

    if (digitalRead(THROTTLEPIN) == HIGH)
    {
        THR_st = micros();
    }
    else
    {
        x = (micros() - THR_st - 1500);
        if (x < 530 && x > -530) //allow for discrepancies
        {
            if (x > 500)
            {
                x = 500;
            }
        }
    }
}

```

```

    }

    if (x < -500)
    {
        x = -500;
    }

    THR_in = x;
}
}
}

// This function is called in the primary setup, it initializes the pinmodes and assigns the ISRs to the
// appropriate pins
void initPWMin() {
    // Set the two pins to be inputs
    pinMode(THROTTLEPIN, INPUT);
    pinMode(STEERPIN, INPUT);

    // Attach an ISR to be called whenever
    // the pin changes from LOW to HIGH or vice versa
    attachInterrupt(digitalPinToInterrupt(STEERPIN), ST_int, CHANGE);
    attachInterrupt(digitalPinToInterrupt(THROTTLEPIN), THR_int, CHANGE);
}

```

//-----

1.1.14 Imu_maths.h

Below:

/*

Inertial Measurement Unit Maths Library

Copyright (C) 2013-2014 Samuel Cowen

www.camelsoftware.com

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

*/

```
#ifndef IMUMATH_H
```

```
#define IMUMATH_H
```

```
#include "vector.h"
```

```
#include "matrix.h"
```

```
#include "quaternion.h"
```

```
#endif
```

1.1.15 FlexCAN.h

Below:

```
/*
```

```
    Inertial Measurement Unit Maths Library
```

```
    Copyright (C) 2013-2014 Samuel Cowen
```

```
    www.camelsoftware.com
```

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <<http://www.gnu.org/licenses/>>.

```
*/
```

```
#ifndef IMUMATH_H
```

```
#define IMUMATH_H
```

```
#include "vector.h"
```

```
#include "matrix.h"
```

```
#include "quaternion.h"
```

```
#endif
```

1.1.16 FlexCAN.cpp

Below:

```
// -----
```

```
// a simple Arduino Teensy 3.1/3.2/3.5/3.6 CAN driver
```

```
// by teachop
```

```
// dual CAN support for MK66FX1M0 and updates for MK64FX512 by Pawelsky
```

```
//
```

```
#include "FlexCAN.h"
```

```
#include "kinetis_flexcan.h"
```

```
static const int txb = 8; // with default settings, all buffers before this are consumed by the FIFO
```

```
static const int txBuffers = 8;
```

```
static const int rxb = 0;
```

```
#define FLEXCANb_MCR(b)      (*(vuint32_t*)(b))
```

```
#define FLEXCANb_CTRL1(b)    (*(vuint32_t*)(b+4))
```

```
#define FLEXCANb_RXMGMASK(b) (*(vuint32_t*)(b+0x10))
```

```
#define FLEXCANb_IFLAG1(b)   (*(vuint32_t*)(b+0x30))
```

```
#define FLEXCANb_RXFGMASK(b) (*(vuint32_t*)(b+0x48))
```

```
#define FLEXCANb_MBn_CS(b, n) (*(vuint32_t*)(b+0x80+n*0x10))
```

```
#define FLEXCANb_MBn_ID(b, n) (*(vuint32_t*)(b+0x84+n*0x10))
```

```
#define FLEXCANb_MBn_WORD0(b, n) (*(vuint32_t*)(b+0x88+n*0x10))
```

```
#define FLEXCANb_MBn_WORD1(b, n) (*(vuint32_t*)(b+0x8C+n*0x10))
```

```
#define FLEXCANb_IDFLT_TAB(b, n) (*(vuint32_t*)(b+0xE0+(n*4)))
```

```
// -----
```

```
FlexCAN::FlexCAN(uint32_t baud, uint8_t id, uint8_t txAlt, uint8_t rxAlt)
```

```

{
    flexcanBase = FLEXCAN0_BASE;
#ifdef __MK66FX1M0__
    if(id > 0) flexcanBase = FLEXCAN1_BASE;
#endif

    // set up the pins
    if(flexcanBase == FLEXCAN0_BASE)
    {
#ifdef defined(__MK66FX1M0__) || defined(__MK64FX512__)
        // 3=PTA12=CAN0_TX, 4=PTA13=CAN0_RX (default)
        // 29=PTB18=CAN0_TX, 30=PTB19=CAN0_RX (alternative)

        if(txAlt == 1) CORE_PIN29_CONFIG = PORT_PCR_MUX(2); else CORE_PIN3_CONFIG =
PORT_PCR_MUX(2);

        if(rxAlt == 1) CORE_PIN30_CONFIG = PORT_PCR_MUX(2); else CORE_PIN4_CONFIG =
PORT_PCR_MUX(2);// | PORT_PCR_PE | PORT_PCR_PS;
#else
        // 3=PTA12=CAN0_TX, 4=PTA13=CAN0_RX (default)
        // 32=PTB18=CAN0_TX, 25=PTB19=CAN0_RX (alternative)

        if(txAlt == 1) CORE_PIN32_CONFIG = PORT_PCR_MUX(2); else CORE_PIN3_CONFIG =
PORT_PCR_MUX(2);

        if(rxAlt == 1) CORE_PIN25_CONFIG = PORT_PCR_MUX(2); else CORE_PIN4_CONFIG =
PORT_PCR_MUX(2);// | PORT_PCR_PE | PORT_PCR_PS;
#endif
    }
#ifdef __MK66FX1M0__
    else if(flexcanBase == FLEXCAN1_BASE)
    {
        // 33=PTE24=CAN1_TX, 34=PTE25=CAN1_RX (default)

        // NOTE: Alternative CAN1 pins are not broken out on Teensy 3.6

```

```

CORE_PIN33_CONFIG = PORT_PCR_MUX(2);

CORE_PIN34_CONFIG = PORT_PCR_MUX(2); // | PORT_PCR_PE | PORT_PCR_PS;
}

#endif

// select clock source 16MHz xtal
OSC0_CR |= OSC_ERCLKEN;

if(flexcanBase == FLEXCAN0_BASE) SIM_SCGC6 |= SIM_SCGC6_FLEXCAN0;
#ifdef __MK66FX1M0__
else if(flexcanBase == FLEXCAN1_BASE) SIM_SCGC3 |= SIM_SCGC3_FLEXCAN1;
#endif

FLEXCANb_CTRL1(flexcanBase) &= ~FLEXCAN_CTRL_CLK_SRC;

// enable CAN
FLEXCANb_MCR(flexcanBase) |= FLEXCAN_MCR_FRZ;
FLEXCANb_MCR(flexcanBase) &= ~FLEXCAN_MCR_MDIS;
while(FLEXCANb_MCR(flexcanBase) & FLEXCAN_MCR_LPM_ACK)
;

// soft reset
FLEXCANb_MCR(flexcanBase) ^= FLEXCAN_MCR_SOFT_RST;
while(FLEXCANb_MCR(flexcanBase) & FLEXCAN_MCR_SOFT_RST)
;

// wait for freeze ack
while(!(FLEXCANb_MCR(flexcanBase) & FLEXCAN_MCR_FRZ_ACK))
;

// disable self-reception
FLEXCANb_MCR(flexcanBase) |= FLEXCAN_MCR_SRX_DIS;

//enable RX FIFO

```

```

FLEXCANb_MCR(flexcanBase) |= FLEXCAN_MCR_FEN;

// segment splits and clock divisor based on baud rate
if ( 50000 == baud ) {
    FLEXCANb_CTRL1(flexcanBase) = (FLEXCAN_CTRL_PROPSEG(2) | FLEXCAN_CTRL_RJW(1)
                                   | FLEXCAN_CTRL_PSEG1(7) | FLEXCAN_CTRL_PSEG2(3) |
FLEXCAN_CTRL_PRESDIV(19));
} else if ( 100000 == baud ) {
    FLEXCANb_CTRL1(flexcanBase) = (FLEXCAN_CTRL_PROPSEG(2) | FLEXCAN_CTRL_RJW(1)
                                   | FLEXCAN_CTRL_PSEG1(7) | FLEXCAN_CTRL_PSEG2(3) | FLEXCAN_CTRL_PRESDIV(9));
} else if ( 250000 == baud ) {
    FLEXCANb_CTRL1(flexcanBase) = (FLEXCAN_CTRL_PROPSEG(2) | FLEXCAN_CTRL_RJW(1)
                                   | FLEXCAN_CTRL_PSEG1(7) | FLEXCAN_CTRL_PSEG2(3) | FLEXCAN_CTRL_PRESDIV(3));
} else if ( 500000 == baud ) {
    FLEXCANb_CTRL1(flexcanBase) = (FLEXCAN_CTRL_PROPSEG(2) | FLEXCAN_CTRL_RJW(1)
                                   | FLEXCAN_CTRL_PSEG1(7) | FLEXCAN_CTRL_PSEG2(3) | FLEXCAN_CTRL_PRESDIV(1));
} else if ( 1000000 == baud ) {
    FLEXCANb_CTRL1(flexcanBase) = (FLEXCAN_CTRL_PROPSEG(2) | FLEXCAN_CTRL_RJW(0)
                                   | FLEXCAN_CTRL_PSEG1(1) | FLEXCAN_CTRL_PSEG2(1) | FLEXCAN_CTRL_PRESDIV(1));
} else { // 125000
    FLEXCANb_CTRL1(flexcanBase) = (FLEXCAN_CTRL_PROPSEG(2) | FLEXCAN_CTRL_RJW(1)
                                   | FLEXCAN_CTRL_PSEG1(7) | FLEXCAN_CTRL_PSEG2(3) | FLEXCAN_CTRL_PRESDIV(7));
}

// Default mask is allow everything
defaultMask.rtr = 0;
defaultMask.ext = 0;
defaultMask.id = 0;
}

```

```

// -----
void FlexCAN::end(void)
{
    // enter freeze mode
    FLEXCANb_MCR(flexcanBase) |= (FLEXCAN_MCR_HALT);
    while(!(FLEXCANb_MCR(flexcanBase) & FLEXCAN_MCR_FRZ_ACK))
        ;
}

// -----
void FlexCAN::begin(const CAN_filter_t &mask)
{
    FLEXCANb_RXMGMASK(flexcanBase) = 0;

    //enable reception of all messages that fit the mask
    if (mask.ext) {
        FLEXCANb_RXFGMASK(flexcanBase) = ((mask.rtr?1:0) << 31) | ((mask.ext?1:0) << 30) | ((mask.id &
        FLEXCAN_MB_ID_EXT_MASK) << 1);
    } else {
        FLEXCANb_RXFGMASK(flexcanBase) = ((mask.rtr?1:0) << 31) | ((mask.ext?1:0) << 30) |
        (FLEXCAN_MB_ID_IDSTD(mask.id) << 1);
    }

    // start the CAN
    FLEXCANb_MCR(flexcanBase) &= ~(FLEXCAN_MCR_HALT);
    // wait till exit of freeze mode

```

```
while(FLEXCANb_MCR(flexcanBase) & FLEXCAN_MCR_FRZ_ACK);
```

```
// wait till ready
```

```
while(FLEXCANb_MCR(flexcanBase) & FLEXCAN_MCR_NOT_RDY);
```

```
//set tx buffers to inactive
```

```
for (int i = txb; i < txb + txBuffers; i++) {
```

```
    FLEXCANb_MBn_CS(flexcanBase, i) = FLEXCAN_MB_CS_CODE(FLEXCAN_MB_CODE_TX_INACTIVE);
```

```
}
```

```
}
```

```
// -----
```

```
void FlexCAN::setFilter(const CAN_filter_t &filter, uint8_t n)
```

```
{
```

```
    if ( 8 > n ) {
```

```
        if (filter.ext) {
```

```
            FLEXCANb_IDFLT_TAB(flexcanBase, n) = ((filter.rtr?1:0) << 31) | ((filter.ext?1:0) << 30) | ((filter.id & FLEXCAN_MB_ID_EXT_MASK) << 1);
```

```
        } else {
```

```
            FLEXCANb_IDFLT_TAB(flexcanBase, n) = ((filter.rtr?1:0) << 31) | ((filter.ext?1:0) << 30) | (FLEXCAN_MB_ID_IDSTD(filter.id) << 1);
```

```
        }
```

```
    }
```

```
}
```

```
// -----
```

```
int FlexCAN::available(void)
```

```

{
    //In FIFO mode, the following interrupt flag signals availability of a frame
    return (FLEXCANb_IFLAG1(flexcanBase) & FLEXCAN_IMASK1_BUF5M)? 1:0;
}

// -----
int FlexCAN::read(CAN_message_t &msg)
{
    unsigned long int startMillis;

    startMillis = msg.timeout? millis() : 0;

    while( !available() ) {
        if ( !msg.timeout || (msg.timeout<=(millis()-startMillis)) ) {
            // early EXIT nothing here
            return 0;
        }
        yield();
    }

    // get identifier and dlc
    msg.len = FLEXCAN_get_length(FLEXCANb_MBn_CS(flexcanBase, rxb));
    msg.ext = (FLEXCANb_MBn_CS(flexcanBase, rxb) & FLEXCAN_MB_CS_IDE)? 1:0;
    msg.id = (FLEXCANb_MBn_ID(flexcanBase, rxb) & FLEXCAN_MB_ID_EXT_MASK);
    if(!msg.ext) {
        msg.id >>= FLEXCAN_MB_ID_STD_BIT_NO;
    }
}

```

```

// copy out message
uint32_t dataIn = FLEXCANb_MBn_WORD0(flexcanBase, rxb);
msg.buf[3] = dataIn;
dataIn >>=8;
msg.buf[2] = dataIn;
dataIn >>=8;
msg.buf[1] = dataIn;
dataIn >>=8;
msg.buf[0] = dataIn;
if ( 4 < msg.len ) {
    dataIn = FLEXCANb_MBn_WORD1(flexcanBase, rxb);
    msg.buf[7] = dataIn;
    dataIn >>=8;
    msg.buf[6] = dataIn;
    dataIn >>=8;
    msg.buf[5] = dataIn;
    dataIn >>=8;
    msg.buf[4] = dataIn;
}
for( int loop=msg.len; loop<8; ++loop ) {
    msg.buf[loop] = 0;
}

//notify FIFO that message has been read
FLEXCANb_IFLAG1(flexcanBase) = FLEXCAN_IMASK1_BUF5M;

return 1;
}

```

```

// -----
int FlexCAN::write(const CAN_message_t &msg)
{
    unsigned long int startMillis;

    startMillis = msg.timeout? millis() : 0;

    // find an available buffer
    int buffer = -1;
    for ( int index = txb; ; ) {
        if ((FLEXCANb_MBn_CS(flexcanBase, index) & FLEXCAN_MB_CS_CODE_MASK) ==
FLEXCAN_MB_CS_CODE(FLEXCAN_MB_CODE_TX_INACTIVE)) {
            buffer = index;
            break;// found one
        }
        if ( !msg.timeout ) {
            if ( ++index >= (txb+txBuffers) ) {
                return 0;// early EXIT no buffers available
            }
        } else {
            // blocking mode, only 1 txb used to guarantee frames in order
            if ( msg.timeout <= (millis()-startMillis) ) {
                return 0;// timed out
            }
            yield();
        }
    }
}

```

```

// transmit the frame

FLEXCANb_MBn_CS(flexcanBase, buffer) =
FLEXCAN_MB_CS_CODE(FLEXCAN_MB_CODE_TX_INACTIVE);

if(msg.ext) {

    FLEXCANb_MBn_ID(flexcanBase, buffer) = (msg.id & FLEXCAN_MB_ID_EXT_MASK);

} else {

    FLEXCANb_MBn_ID(flexcanBase, buffer) = FLEXCAN_MB_ID_IDSTD(msg.id);

}

FLEXCANb_MBn_WORD0(flexcanBase, buffer) =
(msg.buf[0]<<24)|(msg.buf[1]<<16)|(msg.buf[2]<<8)|msg.buf[3];

FLEXCANb_MBn_WORD1(flexcanBase, buffer) =
(msg.buf[4]<<24)|(msg.buf[5]<<16)|(msg.buf[6]<<8)|msg.buf[7];

if(msg.ext) {

    FLEXCANb_MBn_CS(flexcanBase, buffer) = FLEXCAN_MB_CS_CODE(FLEXCAN_MB_CODE_TX_ONCE)
                                         | FLEXCAN_MB_CS_LENGTH(msg.len) | FLEXCAN_MB_CS_SRR |
FLEXCAN_MB_CS_IDE;

} else {

    FLEXCANb_MBn_CS(flexcanBase, buffer) = FLEXCAN_MB_CS_CODE(FLEXCAN_MB_CODE_TX_ONCE)
                                         | FLEXCAN_MB_CS_LENGTH(msg.len);

}

return 1;

}

```

1.1.17 Fault_handler.h

Below:

```

/*

* This .h file defines function and class prototypes that are referenced in the main program file

*//*

// define colors

#define rgbRED 0x00ff0000

```

```

#define rgbGREEN 0x0000ff00
#define rgbBLUE 0x000000ff
#define rgbPURPLE 0x00ff00ff
#define rgbYELLOW 0x00ffff00
#define rgbWHITE 0x00ffffff
#define rgbCYAN 0x0000ffff*/
//define colors part2
#define rgbRED 0x0000ffff
#define rgbGREEN 0x00ff00ff
#define rgbBLUE 0x00ffff00
#define rgbPURPLE 0x0000ff00
#define rgbYELLOW 0x000000ff
#define rgbWHITE 0x00000000
#define rgbCYAN 0x00ff0000
// Fault State Indication
void initFault();
void detectFault();
void indicatorSet(uint32_t RGBval);
//-----

```

1.1.18 Fault_handler.cpp

Below:

```

/*
 * This .cpp file defines the content of functions to be called by the main routine
 */

// Allow access to arduino specific datatypes
#include <arduino.h>
#include "structs.h"
#include "fault_handler.h"

```

```
// Define variables and constants. Volatile is used for variables that are only modified in ISRs
// as the compiler is liable to treat them as constants.
```

```
// Fault / Indicator LED-----
```

```
// Pin numbers for IO
```

```
#define LED_r 5
```

```
#define LED_g 6
```

```
#define LED_b 9
```

```
extern uint8_t errorState;
```

```
extern IMUstruct IMUdat;
```

```
// define external variables used to check for faults
```

```
// throttle input value (ranges from ~-500 to 500) / defined in main
```

```
extern volatile int16_t THR_in;
```

```
//steering input value (ranges from ~-500 to 500) / defined in main
```

```
extern volatile int16_t ST_in;
```

```
//-----
```

```
// Specify contents of functions to be called by main routine
```

```
// Set LED using indicatorSet(0x00RRGGBB)
```

```
void indicatorSet(uint32_t RGBval){
```

```
    analogWrite(LED_r,map((RGBval & 0x00ff0000),0,256,0,4096));
```

```
    analogWrite(LED_g,map((RGBval & 0x0000ff00),0,256,0,4096));
```

```
    analogWrite(LED_b,map((RGBval & 0x000000ff),0,256,0,4096));
```

```
}
```

```
// -----
```

```

// Functions to initialize and detect fault states-----
---

// initialize indicator LED pins

void initFault(){

    // Set led pins to output

    pinMode(LED_r,OUTPUT);

    pinMode(LED_g,OUTPUT);

    pinMode(LED_b,OUTPUT);

}


// passively detects predefined fault states due to bad data etc

void detectFault(){

    if (false){//if ( (THR_in > 600 || THR_in < 600) || (ST_in > 600 || ST_in < 600) ) {

        // receiver error, LED red errorstate 1

        indicatorSet(rgbRED);

        errorState = 1;

    }

    else if (IMUdat.err == 1){

        // IMU comm error, LED purple errorstate 2

        indicatorSet(rgbPURPLE);

        errorState = 2;

    }

    else {

        // no known fault

        indicatorSet(rgbGREEN);

        errorState = 0;

    }

}

//-----

```

1.1.19 BNO.h

Below:

```
/*
 * This .h file defines function and class prototypes that are referenced in the main program file
 */
#include "structs.h"

// IMU configuration
void initIMU();
IMUstruct pollIMU();
//-----
```

1.1.20 BNO.cpp

Below: /*

```
 * This .cpp file defines the content of functions to be called by the main routine
 */
```

```
// Allow access to arduino specific datatypes
```

```
#include <arduino.h>
```

```
#include <Wire.h>
```

```
#include "Adafruit_Sensor.h"
```

```
#include "Adafruit_BNO055.h"
```

```
#include "imumaths.h"
```

```
#include "BNO.h"
```

```
// Define variables and constants. Volatile is used for variables that are only modified in ISRs
```

```
// as the compiler is liable to treat them as constants.
```

```
// IMU-----
```

```
Adafruit_BNO055 bno = Adafruit_BNO055();
```

```
// Function output
```

```
IMUstruct IMUdat;
```

```
//-----
```

```
// Functions to initialize and utilize IMU comms-----  
---
```

```
// communication functions
```

```
// initialize IMU
```

```
void initIMU(){
```

```
    delay(1000);
```

```
    Serial.println("Starting IMU");
```

```
    if(!bno.begin())
```

```
    {
```

```
        /* There was a problem the BNO055 ... check your connections */
```

```
        Serial.println("No BNO055 detected");
```

```
        IMUdat.err = 1;
```

```
    }
```

```
    bno.setExtCrystalUse(true);
```

```
    delay(1000);
```

```
    Serial.println("BNO ready");
```

```
}
```

```
IMUstruct pollIMU(void){
```

```

IMUdat.accel = bno.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER);
IMUdat.gyro = bno.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
IMUdat.quat = bno.getQuat();
IMUdat.mag = bno.getVector(Adafruit_BNO055::VECTOR_MAGNETOMETER);
IMUdat.eul = bno.getVector(Adafruit_BNO055::VECTOR_EULER);
IMUdat.lia = bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
IMUdat.grv = bno.getVector(Adafruit_BNO055::VECTOR_GRAVITY);
return IMUdat;
}
//-----

```

1.1.21 Adafruit_Sensor.h

Below:

```

/*
 * Copyright (C) 2008 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/* Update by K. Townsend (Adafruit Industries) for lighter typedefs, and

```

* extended sensor support to include color, voltage and current */

#ifndef _ADAFRUIT_SENSOR_H

#define _ADAFRUIT_SENSOR_H

#if ARDUINO >= 100

#include "Arduino.h"

#include "Print.h"

#else

#include "WProgram.h"

#endif

/* Intentionally modeled after sensors.h in the Android API:

*

https://github.com/android/platform_hardware_libhardware/blob/master/include/hardware/sensors.h
*/

/* Constants */

#define SENSORS_GRAVITY_EARTH (9.80665F) /**< Earth's gravity in m/s^2 */

#define SENSORS_GRAVITY_MOON (1.6F) /**< The moon's gravity in m/s^2 */

#define SENSORS_GRAVITY_SUN (275.0F) /**< The sun's gravity in m/s^2 */

#define SENSORS_GRAVITY_STANDARD (SENSORS_GRAVITY_EARTH)

#define SENSORS_MAGFIELD_EARTH_MAX (60.0F) /**< Maximum magnetic field on Earth's surface */

#define SENSORS_MAGFIELD_EARTH_MIN (30.0F) /**< Minimum magnetic field on Earth's surface */

#define SENSORS_PRESSURE_SEALEVELHPA (1013.25F) /**< Average sea level pressure is 1013.25 hPa */

#define SENSORS_DPS_TO_RADS (0.017453293F) /**< Degrees/s to rad/s multiplier */

#define SENSORS_GAUSS_TO_MICROTESLA (100) /**< Gauss to micro-Tesla multiplier */

```
/** Sensor types */
```

```
typedef enum
```

```
{
```

```
    SENSOR_TYPE_ACCELEROMETER    = (1), /**< Gravity + linear acceleration */
```

```
    SENSOR_TYPE_MAGNETIC_FIELD    = (2),
```

```
    SENSOR_TYPE_ORIENTATION    = (3),
```

```
    SENSOR_TYPE_GYROSCOPE    = (4),
```

```
    SENSOR_TYPE_LIGHT    = (5),
```

```
    SENSOR_TYPE_PRESSURE    = (6),
```

```
    SENSOR_TYPE_PROXIMITY    = (8),
```

```
    SENSOR_TYPE_GRAVITY    = (9),
```

```
    SENSOR_TYPE_LINEAR_ACCELERATION = (10), /**< Acceleration not including gravity */
```

```
    SENSOR_TYPE_ROTATION_VECTOR    = (11),
```

```
    SENSOR_TYPE_RELATIVE_HUMIDITY    = (12),
```

```
    SENSOR_TYPE_AMBIENT_TEMPERATURE = (13),
```

```
    SENSOR_TYPE_VOLTAGE    = (15),
```

```
    SENSOR_TYPE_CURRENT    = (16),
```

```
    SENSOR_TYPE_COLOR    = (17)
```

```
} sensors_type_t;
```

```
/** struct sensors_vec_s is used to return a vector in a common format. */
```

```
typedef struct {
```

```
    union {
```

```
        float v[3];
```

```
        struct {
```

```
            float x;
```

```
            float y;
```

```
            float z;
```

```

};

/* Orientation sensors */

struct {

    float roll; /**< Rotation around the longitudinal axis (the plane body, 'X axis'). Roll is positive
and increasing when moving downward. -90◊<=roll<=90◊ */

    float pitch; /**< Rotation around the lateral axis (the wing span, 'Y axis'). Pitch is positive and
increasing when moving upwards. -180◊<=pitch<=180◊) */

    float heading; /**< Angle between the longitudinal axis (the plane body) and magnetic north,
measured clockwise when viewing from the top of the device. 0-359◊ */

};

};

int8_t status;

uint8_t reserved[3];
} sensors_vec_t;

/** struct sensors_color_s is used to return color data in a common format. */
typedef struct {

    union {

        float c[3];

        /* RGB color space */

        struct {

            float r; /**< Red component */

            float g; /**< Green component */

            float b; /**< Blue component */

        };

    };

};

uint32_t rgba; /**< 24-bit RGBA value */
} sensors_color_t;

```

```

/* Sensor event (36 bytes) */

/** struct sensor_event_s is used to provide a single sensor event in a common format. */

typedef struct
{
    int32_t version;          /**< must be sizeof(struct sensors_event_t) */
    int32_t sensor_id;        /**< unique sensor identifier */
    int32_t type;             /**< sensor type */
    int32_t reserved0;        /**< reserved */
    int32_t timestamp;        /**< time is in milliseconds */
    union
    {
        float    data[4];
        sensors_vec_t  acceleration;    /**< acceleration values are in meter per second per second
(m/s^2) */
        sensors_vec_t  magnetic;        /**< magnetic vector values are in micro-Tesla (uT) */
        sensors_vec_t  orientation;     /**< orientation values are in degrees */
        sensors_vec_t  gyro;           /**< gyroscope values are in rad/s */
        float    temperature;          /**< temperature is in degrees centigrade (Celsius) */
        float    distance;             /**< distance in centimeters */
        float    light;               /**< light in SI lux units */
        float    pressure;            /**< pressure in hectopascal (hPa) */
        float    relative_humidity;    /**< relative humidity in percent */
        float    current;             /**< current in milliamps (mA) */
        float    voltage;             /**< voltage in volts (V) */
        sensors_color_t color;         /**< color in RGB component values */
    };
} sensors_event_t;

/* Sensor details (40 bytes) */

```

```

/** struct sensor_s is used to describe basic information about a specific sensor. */
typedef struct
{
    char    name[12];           /**< sensor name */
    int32_t version;           /**< version of the hardware + driver */
    int32_t sensor_id;         /**< unique sensor identifier */
    int32_t type;               /**< this sensor's type (ex. SENSOR_TYPE_LIGHT) */
    float   max_value;          /**< maximum value of this sensor's value in SI units */
    float   min_value;          /**< minimum value of this sensor's value in SI units */
    float   resolution;         /**< smallest difference between two values reported by this sensor */
    int32_t min_delay;          /**< min delay in microseconds between events. zero = not a
constant rate */
} sensor_t;

```

```

class Adafruit_Sensor {
public:
    // Constructor(s)
    Adafruit_Sensor() {}
    virtual ~Adafruit_Sensor() {}

    // These must be defined by the subclass
    virtual void enableAutoRange(bool enabled) {};
    virtual bool getEvent(sensors_event_t*) = 0;
    virtual void getSensor(sensor_t*) = 0;

private:
    bool _autoRange;
};

```

#endif

1.1.22 Adafruit_BNO055.h

Below:

```
/******
```

This is a library for the BNO055 orientation sensor

Designed specifically to work with the Adafruit BNO055 Breakout.

Pick one up today in the adafruit shop!

-----> <http://www.adafruit.com/products>

These sensors use I2C to communicate, 2 pins are required to interface.

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing products
from Adafruit!

Written by KTOWN for Adafruit Industries.

MIT license, all text above must be included in any redistribution

```
*****/
```

```
#ifndef __ADAFRUIT_BNO055_H__
```

```
#define __ADAFRUIT_BNO055_H__
```

```
#if (ARDUINO >= 100)
```

```
#include "Arduino.h"
```

```
#else
```

```
#include "WProgram.h"
```

```
#endif
```

```
#ifdef __AVR_ATtiny85__
```

```
#include <TinyWireM.h>
```

```
#define Wire TinyWireM
```

```
#else
```

```
#include <Wire.h>
```

```
#endif
```

```
#include "Adafruit_Sensor.h"
```

```
#include "imumaths.h"
```

```
#define BNO055_ADDRESS_A (0x28)
```

```
#define BNO055_ADDRESS_B (0x29)
```

```
#define BNO055_ID      (0xA0)
```

```
#define NUM_BNO055_OFFSET_REGISTERS (22)
```

```
typedef struct
```

```
{
```

```
    uint16_t accel_offset_x;
```

```
    uint16_t accel_offset_y;
```

```
    uint16_t accel_offset_z;
```

```
    uint16_t gyro_offset_x;
```

```
    uint16_t gyro_offset_y;
```

```
    uint16_t gyro_offset_z;
```

```
    uint16_t mag_offset_x;
```

```
    uint16_t mag_offset_y;
```

```
    uint16_t mag_offset_z;
```

```

uint16_t accel_radius;

uint16_t mag_radius;

} adafruit_bno055_offsets_t;

```

```

class Adafruit_BNO055 : public Adafruit_Sensor

```

```

{

```

```

    public:

```

```

        typedef enum

```

```

        {

```

```

            /* Page id register definition */

```

```

            BNO055_PAGE_ID_ADDR                = 0X07,

```

```

            /* PAGE0 REGISTER DEFINITION START*/

```

```

            BNO055_CHIP_ID_ADDR                = 0x00,

```

```

            BNO055_ACCEL_REV_ID_ADDR           = 0x01,

```

```

            BNO055_MAG_REV_ID_ADDR            = 0x02,

```

```

            BNO055_GYRO_REV_ID_ADDR           = 0x03,

```

```

            BNO055_SW_REV_ID_LSB_ADDR         = 0x04,

```

```

            BNO055_SW_REV_ID_MSB_ADDR         = 0x05,

```

```

            BNO055_BL_REV_ID_ADDR             = 0X06,

```

```

            /* Accel data register */

```

```

            BNO055_ACCEL_DATA_X_LSB_ADDR      = 0X08,

```

```

            BNO055_ACCEL_DATA_X_MSB_ADDR      = 0X09,

```

```

            BNO055_ACCEL_DATA_Y_LSB_ADDR      = 0X0A,

```

```

            BNO055_ACCEL_DATA_Y_MSB_ADDR      = 0X0B,

```

```

            BNO055_ACCEL_DATA_Z_LSB_ADDR      = 0X0C,

```

```

            BNO055_ACCEL_DATA_Z_MSB_ADDR      = 0X0D,

```

/* Mag data register */

BNO055_MAG_DATA_X_LSB_ADDR	= 0X0E,
BNO055_MAG_DATA_X_MSB_ADDR	= 0X0F,
BNO055_MAG_DATA_Y_LSB_ADDR	= 0X10,
BNO055_MAG_DATA_Y_MSB_ADDR	= 0X11,
BNO055_MAG_DATA_Z_LSB_ADDR	= 0X12,
BNO055_MAG_DATA_Z_MSB_ADDR	= 0X13,

/* Gyro data registers */

BNO055_GYRO_DATA_X_LSB_ADDR	= 0X14,
BNO055_GYRO_DATA_X_MSB_ADDR	= 0X15,
BNO055_GYRO_DATA_Y_LSB_ADDR	= 0X16,
BNO055_GYRO_DATA_Y_MSB_ADDR	= 0X17,
BNO055_GYRO_DATA_Z_LSB_ADDR	= 0X18,
BNO055_GYRO_DATA_Z_MSB_ADDR	= 0X19,

/* Euler data registers */

BNO055_EULER_H_LSB_ADDR	= 0X1A,
BNO055_EULER_H_MSB_ADDR	= 0X1B,
BNO055_EULER_R_LSB_ADDR	= 0X1C,
BNO055_EULER_R_MSB_ADDR	= 0X1D,
BNO055_EULER_P_LSB_ADDR	= 0X1E,
BNO055_EULER_P_MSB_ADDR	= 0X1F,

/* Quaternion data registers */

BNO055_QUATERNION_DATA_W_LSB_ADDR	= 0X20,
BNO055_QUATERNION_DATA_W_MSB_ADDR	= 0X21,
BNO055_QUATERNION_DATA_X_LSB_ADDR	= 0X22,

BNO055_QUATERNION_DATA_X_MSB_ADDR	= 0X23,
BNO055_QUATERNION_DATA_Y_LSB_ADDR	= 0X24,
BNO055_QUATERNION_DATA_Y_MSB_ADDR	= 0X25,
BNO055_QUATERNION_DATA_Z_LSB_ADDR	= 0X26,
BNO055_QUATERNION_DATA_Z_MSB_ADDR	= 0X27,

/* Linear acceleration data registers */

BNO055_LINEAR_ACCEL_DATA_X_LSB_ADDR	= 0X28,
BNO055_LINEAR_ACCEL_DATA_X_MSB_ADDR	= 0X29,
BNO055_LINEAR_ACCEL_DATA_Y_LSB_ADDR	= 0X2A,
BNO055_LINEAR_ACCEL_DATA_Y_MSB_ADDR	= 0X2B,
BNO055_LINEAR_ACCEL_DATA_Z_LSB_ADDR	= 0X2C,
BNO055_LINEAR_ACCEL_DATA_Z_MSB_ADDR	= 0X2D,

/* Gravity data registers */

BNO055_GRAVITY_DATA_X_LSB_ADDR	= 0X2E,
BNO055_GRAVITY_DATA_X_MSB_ADDR	= 0X2F,
BNO055_GRAVITY_DATA_Y_LSB_ADDR	= 0X30,
BNO055_GRAVITY_DATA_Y_MSB_ADDR	= 0X31,
BNO055_GRAVITY_DATA_Z_LSB_ADDR	= 0X32,
BNO055_GRAVITY_DATA_Z_MSB_ADDR	= 0X33,

/* Temperature data register */

BNO055_TEMP_ADDR	= 0X34,
------------------	---------

/* Status registers */

BNO055_CALIB_STAT_ADDR	= 0X35,
BNO055_SELFTEST_RESULT_ADDR	= 0X36,
BNO055_INTR_STAT_ADDR	= 0X37,

BNO055_SYS_CLK_STAT_ADDR = 0X38,
BNO055_SYS_STAT_ADDR = 0X39,
BNO055_SYS_ERR_ADDR = 0X3A,

/* Unit selection register */

BNO055_UNIT_SEL_ADDR = 0X3B,
BNO055_DATA_SELECT_ADDR = 0X3C,

/* Mode registers */

BNO055_OPR_MODE_ADDR = 0X3D,
BNO055_PWR_MODE_ADDR = 0X3E,

BNO055_SYS_TRIGGER_ADDR = 0X3F,
BNO055_TEMP_SOURCE_ADDR = 0X40,

/* Axis remap registers */

BNO055_AXIS_MAP_CONFIG_ADDR = 0X41,
BNO055_AXIS_MAP_SIGN_ADDR = 0X42,

/* SIC registers */

BNO055_SIC_MATRIX_0_LSB_ADDR = 0X43,
BNO055_SIC_MATRIX_0_MSB_ADDR = 0X44,
BNO055_SIC_MATRIX_1_LSB_ADDR = 0X45,
BNO055_SIC_MATRIX_1_MSB_ADDR = 0X46,
BNO055_SIC_MATRIX_2_LSB_ADDR = 0X47,
BNO055_SIC_MATRIX_2_MSB_ADDR = 0X48,
BNO055_SIC_MATRIX_3_LSB_ADDR = 0X49,
BNO055_SIC_MATRIX_3_MSB_ADDR = 0X4A,

BNO055_SIC_MATRIX_4_LSB_ADDR	= 0X4B,
BNO055_SIC_MATRIX_4_MSB_ADDR	= 0X4C,
BNO055_SIC_MATRIX_5_LSB_ADDR	= 0X4D,
BNO055_SIC_MATRIX_5_MSB_ADDR	= 0X4E,
BNO055_SIC_MATRIX_6_LSB_ADDR	= 0X4F,
BNO055_SIC_MATRIX_6_MSB_ADDR	= 0X50,
BNO055_SIC_MATRIX_7_LSB_ADDR	= 0X51,
BNO055_SIC_MATRIX_7_MSB_ADDR	= 0X52,
BNO055_SIC_MATRIX_8_LSB_ADDR	= 0X53,
BNO055_SIC_MATRIX_8_MSB_ADDR	= 0X54,

/* Accelerometer Offset registers */

ACCEL_OFFSET_X_LSB_ADDR	= 0X55,
ACCEL_OFFSET_X_MSB_ADDR	= 0X56,
ACCEL_OFFSET_Y_LSB_ADDR	= 0X57,
ACCEL_OFFSET_Y_MSB_ADDR	= 0X58,
ACCEL_OFFSET_Z_LSB_ADDR	= 0X59,
ACCEL_OFFSET_Z_MSB_ADDR	= 0X5A,

/* Magnetometer Offset registers */

MAG_OFFSET_X_LSB_ADDR	= 0X5B,
MAG_OFFSET_X_MSB_ADDR	= 0X5C,
MAG_OFFSET_Y_LSB_ADDR	= 0X5D,
MAG_OFFSET_Y_MSB_ADDR	= 0X5E,
MAG_OFFSET_Z_LSB_ADDR	= 0X5F,
MAG_OFFSET_Z_MSB_ADDR	= 0X60,

/* Gyroscope Offset registers */

GYRO_OFFSET_X_LSB_ADDR	= 0X61,
------------------------	---------

GYRO_OFFSET_X_MSB_ADDR	= 0X62,
GYRO_OFFSET_Y_LSB_ADDR	= 0X63,
GYRO_OFFSET_Y_MSB_ADDR	= 0X64,
GYRO_OFFSET_Z_LSB_ADDR	= 0X65,
GYRO_OFFSET_Z_MSB_ADDR	= 0X66,

/* Radius registers */

ACCEL_RADIUS_LSB_ADDR	= 0X67,
ACCEL_RADIUS_MSB_ADDR	= 0X68,
MAG_RADIUS_LSB_ADDR	= 0X69,
MAG_RADIUS_MSB_ADDR	= 0X6A

} adafruit_bno055_reg_t;

typedef enum

{	
POWER_MODE_NORMAL	= 0X00,
POWER_MODE_LOWPOWER	= 0X01,
POWER_MODE_SUSPEND	= 0X02

} adafruit_bno055_powermode_t;

typedef enum

{	
/* Operation mode settings*/	
OPERATION_MODE_CONFIG	= 0X00,
OPERATION_MODE_ACONLY	= 0X01,
OPERATION_MODE_MAGONLY	= 0X02,
OPERATION_MODE_GYRONLY	= 0X03,
OPERATION_MODE_ACCMAG	= 0X04,
OPERATION_MODE_ACCGYRO	= 0X05,

```

OPERATION_MODE_MAGGYRO          = 0X06,
OPERATION_MODE_AMG               = 0X07,
OPERATION_MODE_IMUPLUS           = 0X08,
OPERATION_MODE_COMPASS           = 0X09,
OPERATION_MODE_M4G               = 0X0A,
OPERATION_MODE_NDOF_FMC_OFF      = 0X0B,
OPERATION_MODE_NDOF              = 0X0C
} adafruit_bno055_opmode_t;

```

typedef enum

```

{
    REMAP_CONFIG_P0              = 0x21,
    REMAP_CONFIG_P1              = 0x24, // default
    REMAP_CONFIG_P2              = 0x24,
    REMAP_CONFIG_P3              = 0x21,
    REMAP_CONFIG_P4              = 0x24,
    REMAP_CONFIG_P5              = 0x21,
    REMAP_CONFIG_P6              = 0x21,
    REMAP_CONFIG_P7              = 0x24
} adafruit_bno055_axis_remap_config_t;

```

typedef enum

```

{
    REMAP_SIGN_P0                = 0x04,
    REMAP_SIGN_P1                = 0x00, // default
    REMAP_SIGN_P2                = 0x06,
    REMAP_SIGN_P3                = 0x02,
    REMAP_SIGN_P4                = 0x03,
    REMAP_SIGN_P5                = 0x01,

```

```

    REMAP_SIGN_P6                = 0x07,
    REMAP_SIGN_P7                = 0x05
} adafruit_bno055_axis_remap_sign_t;

typedef struct
{
    uint8_t accel_rev;
    uint8_t mag_rev;
    uint8_t gyro_rev;
    uint16_t sw_rev;
    uint8_t bl_rev;
} adafruit_bno055_rev_info_t;

typedef enum
{
    VECTOR_ACCELEROMETER = BNO055_ACCEL_DATA_X_LSB_ADDR,
    VECTOR_MAGNETOMETER = BNO055_MAG_DATA_X_LSB_ADDR,
    VECTOR_GYROSCOPE    = BNO055_GYRO_DATA_X_LSB_ADDR,
    VECTOR_EULER        = BNO055_EULER_H_LSB_ADDR,
    VECTOR_LINEARACCEL  = BNO055_LINEAR_ACCEL_DATA_X_LSB_ADDR,
    VECTOR_GRAVITY      = BNO055_GRAVITY_DATA_X_LSB_ADDR
} adafruit_vector_type_t;

#if defined (ARDUINO_SAMD_ZERO) && ! (ARDUINO_SAMD_FEATHER_M0)
#error "On an arduino Zero, BNO055's ADR pin must be high. Fix that, then delete this line."

    Adafruit_BNO055 ( int32_t sensorID = -1, uint8_t address = BNO055_ADDRESS_B );
#else
    Adafruit_BNO055 ( int32_t sensorID = -1, uint8_t address = BNO055_ADDRESS_A );
#endif

```

```

bool begin      ( adafruit_bno055_opmode_t mode = OPERATION_MODE_NDOF );
void setMode    ( adafruit_bno055_opmode_t mode );
void getRevInfo ( adafruit_bno055_rev_info_t* );
void displayRevInfo ( void );
void setExtCrystalUse ( boolean usextal );
void getSystemStatus ( uint8_t *system_status,
                      uint8_t *self_test_result,
                      uint8_t *system_error);
void displaySystemStatus ( void );
void getCalibration ( uint8_t* system, uint8_t* gyro, uint8_t* accel, uint8_t* mag);

imu::Vector<3> getVector ( adafruit_vector_type_t vector_type );
imu::Quaternion getQuat ( void );
int8_t      getTemp ( void );

/* Adafruit_Sensor implementation */
bool getEvent ( sensors_event_t* );
void getSensor ( sensor_t* );

/* Functions to deal with raw calibration data */
bool getSensorOffsets(uint8_t* calibData);
bool getSensorOffsets(adafruit_bno055_offsets_t &offsets_type);
void setSensorOffsets(const uint8_t* calibData);
void setSensorOffsets(const adafruit_bno055_offsets_t &offsets_type);
bool isFullyCalibrated(void);

private:
byte read8 ( adafruit_bno055_reg_t );
bool readLen ( adafruit_bno055_reg_t, byte* buffer, uint8_t len );

```

```
bool write8 ( adafruit_bno055_reg_t, byte value );

uint8_t _address;

int32_t _sensorID;

adafruit_bno055_opmode_t _mode;

};
```

#endif

1.1.23 Adafruit_BNO055.cpp

Below:

```
/******
```

This is a library for the BNO055 orientation sensor

Designed specifically to work with the Adafruit BNO055 Breakout.

Pick one up today in the adafruit shop!

-----> <http://www.adafruit.com/products>

These sensors use I2C to communicate, 2 pins are required to interface.

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing products
from Adafruit!

Written by KTOWN for Adafruit Industries.

MIT license, all text above must be included in any redistribution

```
*****/
```

```

#if ARDUINO >= 100

#include "Arduino.h"

#else

#include "WProgram.h"

#endif


#include <math.h>

#include <limits.h>


#include "Adafruit_BNO055.h"


/*****

CONSTRUCTOR

*****/

/*****/

/*!
  @brief Instantiates a new Adafruit_BNO055 class
*/

/*****/

Adafruit_BNO055::Adafruit_BNO055(int32_t sensorID, uint8_t address)
{
  _sensorID = sensorID;
  _address = address;
}


/*****

PUBLIC FUNCTIONS

*****/

```

```

/*****
/*!
  @brief Sets up the HW
*/
*****/

bool Adafruit_BNO055::begin(adafruit_bno055_opmode_t mode)
{
  /* Enable I2C */
  Wire.begin();

  // BNO055 clock stretches for 500us or more!
#ifdef ESP8266
  Wire.setClockStretchLimit(1000); // Allow for 1000us of clock stretching
#endif

  /* Make sure we have the right device */
  uint8_t id = read8(BNO055_CHIP_ID_ADDR);
  if(id != BNO055_ID)
  {
    delay(1000); // hold on for boot
    id = read8(BNO055_CHIP_ID_ADDR);
    if(id != BNO055_ID) {
      return false; // still not? ok bail
    }
  }

  /* Switch to config mode (just in case since this is the default) */
  setMode(OPERATION_MODE_CONFIG);

```

```

/* Reset */

write8(BNO055_SYS_TRIGGER_ADDR, 0x20);
while (read8(BNO055_CHIP_ID_ADDR) != BNO055_ID)
{
    delay(10);
}
delay(50);

/* Set to normal power mode */

write8(BNO055_PWR_MODE_ADDR, POWER_MODE_NORMAL);
delay(10);

write8(BNO055_PAGE_ID_ADDR, 0);

/* Set the output units */
/*
uint8_t unitsel = (0 << 7) | // Orientation = Android
                (0 << 4) | // Temperature = Celsius
                (0 << 2) | // Euler = Degrees
                (1 << 1) | // Gyro = Rads
                (0 << 0); // Accelerometer = m/s^2
write8(BNO055_UNIT_SEL_ADDR, unitsel);
*/

/* Configure axis mapping (see section 3.4) */
/*
write8(BNO055_AXIS_MAP_CONFIG_ADDR, REMAP_CONFIG_P2); // P0-P7, Default is P1
delay(10);

```

```

write8(BNO055_AXIS_MAP_SIGN_ADDR, REMAP_SIGN_P2); // P0-P7, Default is P1

delay(10);

*/

write8(BNO055_SYS_TRIGGER_ADDR, 0x0);

delay(10);

/* Set the requested operating mode (see section 3.3) */

setMode(mode);

delay(20);

return true;
}

/*****

/*!

@brief Puts the chip in the specified operating mode

*/

*****/

void Adafruit_BNO055::setMode(adafruit_bno055_opmode_t mode)
{
    _mode = mode;

    write8(BNO055_OPR_MODE_ADDR, _mode);

    delay(30);
}

/*****

/*!

@brief Use the external 32.768KHz crystal

*/

```

```

/*****/

void Adafruit_BNO055::setExtCrystalUse(boolean usextal)
{
    adafruit_bno055_opmode_t modeback = _mode;

    /* Switch to config mode (just in case since this is the default) */
    setMode(OPERATION_MODE_CONFIG);
    delay(25);
    write8(BNO055_PAGE_ID_ADDR, 0);
    if (usextal) {
        write8(BNO055_SYS_TRIGGER_ADDR, 0x80);
    } else {
        write8(BNO055_SYS_TRIGGER_ADDR, 0x00);
    }
    delay(10);
    /* Set the requested operating mode (see section 3.3) */
    setMode(modeback);
    delay(20);
}

/*****/

/*!
    @brief Gets the latest system status info
*/

/*****/

void Adafruit_BNO055::getSystemStatus(uint8_t *system_status, uint8_t *self_test_result, uint8_t
*system_error)
{

```

```
write8(BNO055_PAGE_ID_ADDR, 0);
```

```
/* System Status (see section 4.3.58)
```

```
-----
```

```
0 = Idle
```

```
1 = System Error
```

```
2 = Initializing Peripherals
```

```
3 = System Initialization
```

```
4 = Executing Self-Test
```

```
5 = Sensor fusion algorithm running
```

```
6 = System running without fusion algorithms */
```

```
if (system_status != 0)
```

```
    *system_status = read8(BNO055_SYS_STAT_ADDR);
```

```
/* Self Test Results (see section )
```

```
-----
```

```
1 = test passed, 0 = test failed
```

```
Bit 0 = Accelerometer self test
```

```
Bit 1 = Magnetometer self test
```

```
Bit 2 = Gyroscope self test
```

```
Bit 3 = MCU self test
```

```
0x0F = all good! */
```

```
if (self_test_result != 0)
```

```
    *self_test_result = read8(BNO055_SELFTEST_RESULT_ADDR);
```

```
/* System Error (see section 4.3.59)
```

```
-----
```

```
0 = No error
```

```
1 = Peripheral initialization error
```

```
2 = System initialization error
```

```
3 = Self test result failed
```

```
4 = Register map value out of range
```

```
5 = Register map address out of range
```

```
6 = Register map write error
```

```
7 = BNO low power mode not available for selected operation mode
```

```
8 = Accelerometer power mode not available
```

```
9 = Fusion algorithm configuration error
```

```
A = Sensor configuration error */
```

```
if (system_error != 0)
```

```
    *system_error = read8(BNO055_SYS_ERR_ADDR);
```

```
delay(200);
```

```
}
```

```
/**-----
```

```
/*!
```

```
@brief Gets the chip revision numbers
```

```
*/
```

```
/**-----
```

```
void Adafruit_BNO055::getRevInfo(adafruit_bno055_rev_info_t* info)
```

```
{
```

```
    uint8_t a, b;
```

```

memset(info, 0, sizeof(adafruit_bno055_rev_info_t));

/* Check the accelerometer revision */
info->accel_rev = read8(BNO055_ACCEL_REV_ID_ADDR);

/* Check the magnetometer revision */
info->mag_rev = read8(BNO055_MAG_REV_ID_ADDR);

/* Check the gyroscope revision */
info->gyro_rev = read8(BNO055_GYRO_REV_ID_ADDR);

/* Check the SW revision */
info->bl_rev = read8(BNO055_BL_REV_ID_ADDR);

a = read8(BNO055_SW_REV_ID_LSB_ADDR);
b = read8(BNO055_SW_REV_ID_MSB_ADDR);
info->sw_rev = (((uint16_t)b) << 8) | ((uint16_t)a);
}

/*****

/*!
 @brief Gets current calibration state. Each value should be a uint8_t
 pointer and it will be set to 0 if not calibrated and 3 if
 fully calibrated.

 */

*****/

void Adafruit_BNO055::getCalibration(uint8_t* sys, uint8_t* gyro, uint8_t* accel, uint8_t* mag) {
    uint8_t calData = read8(BNO055_CALIB_STAT_ADDR);

    if (sys != NULL) {

```

```

    *sys = (calData >> 6) & 0x03;
}
if (gyro != NULL) {
    *gyro = (calData >> 4) & 0x03;
}
if (accel != NULL) {
    *accel = (calData >> 2) & 0x03;
}
if (mag != NULL) {
    *mag = calData & 0x03;
}
}

/*****/
/*!
    @brief Gets the temperature in degrees celsius
*/
/*****/

int8_t Adafruit_BNO055::getTemp(void)
{
    int8_t temp = (int8_t)(read8(BNO055_TEMP_ADDR));
    return temp;
}

/*****/
/*!
    @brief Gets a vector reading from the specified source
*/
/*****/

```

```

imu::Vector<3> Adafruit_BNO055::getVector(adafruit_vector_type_t vector_type)
{
    imu::Vector<3> xyz;
    uint8_t buffer[6];
    memset (buffer, 0, 6);

    int16_t x, y, z;
    x = y = z = 0;

    /* Read vector data (6 bytes) */
    readLen((adafruit_bno055_reg_t)vector_type, buffer, 6);

    x = ((int16_t)buffer[0]) | (((int16_t)buffer[1]) << 8);
    y = ((int16_t)buffer[2]) | (((int16_t)buffer[3]) << 8);
    z = ((int16_t)buffer[4]) | (((int16_t)buffer[5]) << 8);

    /* Convert the value to an appropriate range (section 3.6.4) */
    /* and assign the value to the Vector type */
    switch(vector_type)
    {
        case VECTOR_MAGNETOMETER:
            /* 1uT = 16 LSB */
            xyz[0] = ((double)x)/16.0;
            xyz[1] = ((double)y)/16.0;
            xyz[2] = ((double)z)/16.0;
            break;
        case VECTOR_GYROSCOPE:
            /* 1rps = 900 LSB */
            xyz[0] = ((double)x)/900.0;

```

```

    xyz[1] = ((double)y)/900.0;
    xyz[2] = ((double)z)/900.0;

    break;
case VECTOR_EULER:
    /* 1 degree = 16 LSB */
    xyz[0] = ((double)x)/16.0;
    xyz[1] = ((double)y)/16.0;
    xyz[2] = ((double)z)/16.0;

    break;
case VECTOR_ACCELEROMETER:
case VECTOR_LINEARACCEL:
case VECTOR_GRAVITY:
    /* 1m/s^2 = 100 LSB */
    xyz[0] = ((double)x)/100.0;
    xyz[1] = ((double)y)/100.0;
    xyz[2] = ((double)z)/100.0;

    break;
}

return xyz;
}

/*****
/*!
 @brief Gets a quaternion reading from the specified source
 */
*****/

imu::Quaternion Adafruit_BNO055::getQuat(void)
{

```

```

uint8_t buffer[8];

memset (buffer, 0, 8);


int16_t x, y, z, w;

x = y = z = w = 0;


/* Read quat data (8 bytes) */
readLen(BNO055_QUATERNION_DATA_W_LSB_ADDR, buffer, 8);

w = (((uint16_t)buffer[1]) << 8) | ((uint16_t)buffer[0]);
x = (((uint16_t)buffer[3]) << 8) | ((uint16_t)buffer[2]);
y = (((uint16_t)buffer[5]) << 8) | ((uint16_t)buffer[4]);
z = (((uint16_t)buffer[7]) << 8) | ((uint16_t)buffer[6]);


/* Assign to Quaternion */

/* See http://ae-bst.resource.bosch.com/media/products/dokumente/bno055/BST\_BNO055\_DS000\_12~1.pdf
3.6.5.5 Orientation (Quaternion) */
const double scale = (1.0 / (1<<14));

imu::Quaternion quat(scale * w, scale * x, scale * y, scale * z);

return quat;
}


/*****

/*!

@brief Provides the sensor_t data for this sensor

*/

*****/

void Adafruit_BNO055::getSensor(sensor_t *sensor)

{

```

```

/* Clear the sensor_t object */
memset(sensor, 0, sizeof(sensor_t));

/* Insert the sensor name in the fixed length char array */
strncpy (sensor->name, "BNO055", sizeof(sensor->name) - 1);
sensor->name[sizeof(sensor->name)- 1] = 0;
sensor->version  = 1;
sensor->sensor_id = _sensorID;
sensor->type     = SENSOR_TYPE_ORIENTATION;
sensor->min_delay = 0;
sensor->max_value = 0.0F;
sensor->min_value = 0.0F;
sensor->resolution = 0.01F;
}

/*****
/*!
  @brief Reads the sensor and returns the data as a sensors_event_t
*/
*****/

bool Adafruit_BNO055::getEvent(sensors_event_t *event)
{
  /* Clear the event */
  memset(event, 0, sizeof(sensors_event_t));

  event->version = sizeof(sensors_event_t);
  event->sensor_id = _sensorID;
  event->type     = SENSOR_TYPE_ORIENTATION;
  event->timestamp = millis();

```

```

/* Get a Euler angle sample for orientation */
imu::Vector<3> euler = getVector(Adafruit_BNO055::VECTOR_EULER);
event->orientation.x = euler.x();
event->orientation.y = euler.y();
event->orientation.z = euler.z();

return true;
}

/*****
/*!
@brief Reads the sensor's offset registers into a byte array
*/
*****/

bool Adafruit_BNO055::getSensorOffsets(uint8_t* calibData)
{
    if (isFullyCalibrated())
    {
        adafruit_bno055_opmode_t lastMode = _mode;
        setMode(OPERATION_MODE_CONFIG);

        readLen(ACCEL_OFFSET_X_LSB_ADDR, calibData, NUM_BNO055_OFFSET_REGISTERS);

        setMode(lastMode);
        return true;
    }
    return false;
}

```

```

/*****
/!
@brief Reads the sensor's offset registers into an offset struct
*/
/*****/

bool Adafruit_BNO055::getSensorOffsets(adafruit_bno055_offsets_t &offsets_type)
{
    if (isFullyCalibrated())
    {
        adafruit_bno055_opmode_t lastMode = _mode;

        setMode(OPERATION_MODE_CONFIG);

        delay(25);

        offsets_type.accel_offset_x = (read8(ACCEL_OFFSET_X_MSB_ADDR) << 8) |
(read8(ACCEL_OFFSET_X_LSB_ADDR));

        offsets_type.accel_offset_y = (read8(ACCEL_OFFSET_Y_MSB_ADDR) << 8) |
(read8(ACCEL_OFFSET_Y_LSB_ADDR));

        offsets_type.accel_offset_z = (read8(ACCEL_OFFSET_Z_MSB_ADDR) << 8) |
(read8(ACCEL_OFFSET_Z_LSB_ADDR));

        offsets_type.gyro_offset_x = (read8(GYRO_OFFSET_X_MSB_ADDR) << 8) |
(read8(GYRO_OFFSET_X_LSB_ADDR));

        offsets_type.gyro_offset_y = (read8(GYRO_OFFSET_Y_MSB_ADDR) << 8) |
(read8(GYRO_OFFSET_Y_LSB_ADDR));

        offsets_type.gyro_offset_z = (read8(GYRO_OFFSET_Z_MSB_ADDR) << 8) |
(read8(GYRO_OFFSET_Z_LSB_ADDR));

        offsets_type.mag_offset_x = (read8(MAG_OFFSET_X_MSB_ADDR) << 8) |
(read8(MAG_OFFSET_X_LSB_ADDR));

        offsets_type.mag_offset_y = (read8(MAG_OFFSET_Y_MSB_ADDR) << 8) |
(read8(MAG_OFFSET_Y_LSB_ADDR));
    }
}

```

```
    offsets_type.mag_offset_z = (read8(MAG_OFFSET_Z_MSB_ADDR) << 8) |
(read8(MAG_OFFSET_Z_LSB_ADDR));
```

```
    offsets_type.accel_radius = (read8(ACCEL_RADIUS_MSB_ADDR) << 8) |
(read8(ACCEL_RADIUS_LSB_ADDR));
```

```
    offsets_type.mag_radius = (read8(MAG_RADIUS_MSB_ADDR) << 8) |
(read8(MAG_RADIUS_LSB_ADDR));
```

```
    setMode(lastMode);
```

```
    return true;
```

```
}
```

```
return false;
```

```
}
```

```
/**/
```

```
/*!
```

```
@brief Writes an array of calibration values to the sensor's offset registers
```

```
*/
```

```
/**/
```

```
void Adafruit_BNO055::setSensorOffsets(const uint8_t* calibData)
```

```
{
```

```
    adafruit_bno055_opmode_t lastMode = _mode;
```

```
    setMode(OPERATION_MODE_CONFIG);
```

```
    delay(25);
```

```
    /* A writeLen() would make this much cleaner */
```

```
    write8(ACCEL_OFFSET_X_LSB_ADDR, calibData[0]);
```

```
    write8(ACCEL_OFFSET_X_MSB_ADDR, calibData[1]);
```

```
    write8(ACCEL_OFFSET_Y_LSB_ADDR, calibData[2]);
```

```

write8(ACCEL_OFFSET_Y_MSB_ADDR, calibData[3]);
write8(ACCEL_OFFSET_Z_LSB_ADDR, calibData[4]);
write8(ACCEL_OFFSET_Z_MSB_ADDR, calibData[5]);


write8(GYRO_OFFSET_X_LSB_ADDR, calibData[6]);
write8(GYRO_OFFSET_X_MSB_ADDR, calibData[7]);
write8(GYRO_OFFSET_Y_LSB_ADDR, calibData[8]);
write8(GYRO_OFFSET_Y_MSB_ADDR, calibData[9]);
write8(GYRO_OFFSET_Z_LSB_ADDR, calibData[10]);
write8(GYRO_OFFSET_Z_MSB_ADDR, calibData[11]);


write8(MAG_OFFSET_X_LSB_ADDR, calibData[12]);
write8(MAG_OFFSET_X_MSB_ADDR, calibData[13]);
write8(MAG_OFFSET_Y_LSB_ADDR, calibData[14]);
write8(MAG_OFFSET_Y_MSB_ADDR, calibData[15]);
write8(MAG_OFFSET_Z_LSB_ADDR, calibData[16]);
write8(MAG_OFFSET_Z_MSB_ADDR, calibData[17]);


write8(ACCEL_RADIUS_LSB_ADDR, calibData[18]);
write8(ACCEL_RADIUS_MSB_ADDR, calibData[19]);


write8(MAG_RADIUS_LSB_ADDR, calibData[20]);
write8(MAG_RADIUS_MSB_ADDR, calibData[21]);


setMode(lastMode);
}

/*****
/*!

```

@brief Writes to the sensor's offset registers from an offset struct

```
*/  
  
/*****  
  
void Adafruit_BNO055::setSensorOffsets(const adafruit_bno055_offsets_t &offsets_type)  
{  
    adafruit_bno055_opmode_t lastMode = _mode;  
    setMode(OPERATION_MODE_CONFIG);  
    delay(25);  
  
    write8(ACCEL_OFFSET_X_LSB_ADDR, (offsets_type.accel_offset_x) & 0xFF);  
    write8(ACCEL_OFFSET_X_MSB_ADDR, (offsets_type.accel_offset_x >> 8) & 0xFF);  
    write8(ACCEL_OFFSET_Y_LSB_ADDR, (offsets_type.accel_offset_y) & 0xFF);  
    write8(ACCEL_OFFSET_Y_MSB_ADDR, (offsets_type.accel_offset_y >> 8) & 0xFF);  
    write8(ACCEL_OFFSET_Z_LSB_ADDR, (offsets_type.accel_offset_z) & 0xFF);  
    write8(ACCEL_OFFSET_Z_MSB_ADDR, (offsets_type.accel_offset_z >> 8) & 0xFF);  
  
    write8(GYRO_OFFSET_X_LSB_ADDR, (offsets_type.gyro_offset_x) & 0xFF);  
    write8(GYRO_OFFSET_X_MSB_ADDR, (offsets_type.gyro_offset_x >> 8) & 0xFF);  
    write8(GYRO_OFFSET_Y_LSB_ADDR, (offsets_type.gyro_offset_y) & 0xFF);  
    write8(GYRO_OFFSET_Y_MSB_ADDR, (offsets_type.gyro_offset_y >> 8) & 0xFF);  
    write8(GYRO_OFFSET_Z_LSB_ADDR, (offsets_type.gyro_offset_z) & 0xFF);  
    write8(GYRO_OFFSET_Z_MSB_ADDR, (offsets_type.gyro_offset_z >> 8) & 0xFF);  
  
    write8(MAG_OFFSET_X_LSB_ADDR, (offsets_type.mag_offset_x) & 0xFF);  
    write8(MAG_OFFSET_X_MSB_ADDR, (offsets_type.mag_offset_x >> 8) & 0xFF);  
    write8(MAG_OFFSET_Y_LSB_ADDR, (offsets_type.mag_offset_y) & 0xFF);  
    write8(MAG_OFFSET_Y_MSB_ADDR, (offsets_type.mag_offset_y >> 8) & 0xFF);  
    write8(MAG_OFFSET_Z_LSB_ADDR, (offsets_type.mag_offset_z) & 0xFF);  
    write8(MAG_OFFSET_Z_MSB_ADDR, (offsets_type.mag_offset_z >> 8) & 0xFF);  
}
```

```

write8(ACCEL_RADIUS_LSB_ADDR, (offsets_type.accel_radius) & 0xFF);
write8(ACCEL_RADIUS_MSB_ADDR, (offsets_type.accel_radius >> 8) & 0xFF);

write8(MAG_RADIUS_LSB_ADDR, (offsets_type.mag_radius) & 0xFF);
write8(MAG_RADIUS_MSB_ADDR, (offsets_type.mag_radius >> 8) & 0xFF);

setMode(lastMode);
}

```

```

bool Adafruit_BNO055::isFullyCalibrated(void)
{
    uint8_t system, gyro, accel, mag;
    getCalibration(&system, &gyro, &accel, &mag);
    if (system < 3 || gyro < 3 || accel < 3 || mag < 3)
        return false;
    return true;
}

```

```

/*****

PRIVATE FUNCTIONS

*****/

/*****/

/*!
    @brief Writes an 8 bit value over I2C

*/

*****/

```

```
bool Adafruit_BNO055::write8(adafruit_bno055_reg_t reg, byte value)
```

```
{  
  Wire.beginTransaction(_address);  
  #if ARDUINO >= 100  
    Wire.write((uint8_t)reg);  
    Wire.write((uint8_t)value);  
  #else  
    Wire.send(reg);  
    Wire.send(value);  
  #endif  
  Wire.endTransmission();  
  
  /* ToDo: Check for error! */  
  return true;  
}
```

```
/*  
*****  
*!  
  @brief Reads an 8 bit value over I2C  
*/
```

```
*****
```

```
byte Adafruit_BNO055::read8(adafruit_bno055_reg_t reg )
```

```
{  
  byte value = 0;  
  
  Wire.beginTransaction(_address);  
  #if ARDUINO >= 100  
    Wire.write((uint8_t)reg);  
  #else
```

```

    Wire.send(reg);

#endif

Wire.endTransmission();

Wire.requestFrom(_address, (byte)1);

#if ARDUINO >= 100
    value = Wire.read();
#else
    value = Wire.receive();
#endif

return value;
}

/*****
 *!
 * @brief Reads the specified number of bytes over I2C
 */
*****/

bool Adafruit_BNO055::readLen(adafruit_bno055_reg_t reg, byte * buffer, uint8_t len)
{
    Wire.beginTransmission(_address);

    #if ARDUINO >= 100
        Wire.write((uint8_t)reg);
    #else
        Wire.send(reg);
    #endif

    Wire.endTransmission();

    Wire.requestFrom(_address, (byte)len);

```

```

for (uint8_t i = 0; i < len; i++)
{
    #if ARDUINO >= 100
        buffer[i] = Wire.read();
    #else
        buffer[i] = Wire.receive();
    #endif
}

/* ToDo: Check for errors! */

return true;
}

```

1.1.24 VL53l0x_types.h

```

/*****

```

Copyright © 2015, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND

ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

/**

* @file vl53l0_types.h

* @brief VL53L0 types definition

*/

#ifndef VL53L0X_TYPES_H_

#define VL53L0X_TYPES_H_

/** @defgroup porting_type Basic type definition

* @ingroup VL53L0X_platform_group

*

* @brief file vl53l0_types.h files hold basic type definition that may requires porting

*

* contains type that must be defined for the platform\n

* when target platform and compiler provide stdint.h and stddef.h it is enough to include it.\n

* If stdint.h is not available review and adapt all signed and unsigned 8/16/32 bits basic types. \n

* If stddef.h is not available review and adapt NULL definition .

*/

```
#include <stdint.h>
```

```
#include <stddef.h>
```

```
#ifndef NULL
```

```
#error "Error NULL definition should be done. Please add required include "
```

```
#endif
```

```
#if ! defined(STDINT_H) && !defined(_GCC_STDINT_H) &&!defined(__STDINT_DECLS) &&  
!defined(_GCC_WRAP_STDINT_H)
```

```
#pragma message("Please review type definition of STDINT define for your platform and add to list  
above ")
```

```
/*
```

```
* target platform do not provide stdint or use a different #define than above
```

```
* to avoid seeing the message below addapt the #define list above or implement
```

```
* all type and delete these pragma
```

```
*/
```

```
/** \ingroup VL53L0X_portingType_group
```

```
* @{
```

```
*/
```

```
typedef unsigned long long uint64_t;
```

```
/** @brief Typedef defining 32 bit unsigned int type.\n
```

* The developer should modify this to suit the platform being deployed.

*/

typedef unsigned int uint32_t;

/** @brief Typedef defining 32 bit int type.\n

* The developer should modify this to suit the platform being deployed.

*/

typedef int int32_t;

/** @brief Typedef defining 16 bit unsigned short type.\n

* The developer should modify this to suit the platform being deployed.

*/

typedef unsigned short uint16_t;

/** @brief Typedef defining 16 bit short type.\n

* The developer should modify this to suit the platform being deployed.

*/

typedef short int16_t;

/** @brief Typedef defining 8 bit unsigned char type.\n

* The developer should modify this to suit the platform being deployed.

*/

typedef unsigned char uint8_t;

/** @brief Typedef defining 8 bit char type.\n

* The developer should modify this to suit the platform being deployed.

*/

typedef signed char int8_t;

```
/** @} */
```

```
#endif /* _STDINT_H */
```

```
/** use where fractional values are expected
```

```
*
```

```
* Given a floating point value f it's .16 bit point is (int)(f*(1<<16))*/
```

```
typedef uint32_t FixPoint1616_t;
```

```
#endif /* VL53L0X_TYPES_H_ */
```

1.1.25 VL53L0X_TUNING.H

```
/******
```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without

modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND

NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#ifndef _VL53L0X_TUNING_H_
```

```
#define _VL53L0X_TUNING_H_
```

```
#include "vl53l0x_def.h"
```

```
#ifdef __cplusplus
```

```
extern "C" {
```

```
#endif
```

```
uint8_t DefaultTuningSettings[] = {
```

```
    /* update 02/11/2015_v36 */
```

```
    0x01, 0xFF, 0x01,
```

```
    0x01, 0x00, 0x00,
```

```
    0x01, 0xFF, 0x00,
```

0x01, 0x09, 0x00,
0x01, 0x10, 0x00,
0x01, 0x11, 0x00,

0x01, 0x24, 0x01,
0x01, 0x25, 0xff,
0x01, 0x75, 0x00,

0x01, 0xFF, 0x01,
0x01, 0x4e, 0x2c,
0x01, 0x48, 0x00,
0x01, 0x30, 0x20,

0x01, 0xFF, 0x00,
0x01, 0x30, 0x09, /* mja changed from 0x64. */
0x01, 0x54, 0x00,
0x01, 0x31, 0x04,
0x01, 0x32, 0x03,
0x01, 0x40, 0x83,
0x01, 0x46, 0x25,
0x01, 0x60, 0x00,
0x01, 0x27, 0x00,
0x01, 0x50, 0x06,
0x01, 0x51, 0x00,
0x01, 0x52, 0x96,
0x01, 0x56, 0x08,
0x01, 0x57, 0x30,
0x01, 0x61, 0x00,
0x01, 0x62, 0x00,

0x01, 0x64, 0x00,
0x01, 0x65, 0x00,
0x01, 0x66, 0xa0,

0x01, 0xFF, 0x01,
0x01, 0x22, 0x32,
0x01, 0x47, 0x14,
0x01, 0x49, 0xff,
0x01, 0x4a, 0x00,

0x01, 0xFF, 0x00,
0x01, 0x7a, 0x0a,
0x01, 0x7b, 0x00,
0x01, 0x78, 0x21,

0x01, 0xFF, 0x01,
0x01, 0x23, 0x34,
0x01, 0x42, 0x00,
0x01, 0x44, 0xff,
0x01, 0x45, 0x26,
0x01, 0x46, 0x05,
0x01, 0x40, 0x40,
0x01, 0x0E, 0x06,
0x01, 0x20, 0x1a,
0x01, 0x43, 0x40,

0x01, 0xFF, 0x00,
0x01, 0x34, 0x03,
0x01, 0x35, 0x44,

0x01, 0xFF, 0x01,
0x01, 0x31, 0x04,
0x01, 0x4b, 0x09,
0x01, 0x4c, 0x05,
0x01, 0x4d, 0x04,

0x01, 0xFF, 0x00,
0x01, 0x44, 0x00,
0x01, 0x45, 0x20,
0x01, 0x47, 0x08,
0x01, 0x48, 0x28,
0x01, 0x67, 0x00,
0x01, 0x70, 0x04,
0x01, 0x71, 0x01,
0x01, 0x72, 0xfe,
0x01, 0x76, 0x00,
0x01, 0x77, 0x00,

0x01, 0xFF, 0x01,
0x01, 0x0d, 0x01,

0x01, 0xFF, 0x00,
0x01, 0x80, 0x01,
0x01, 0x01, 0xF8,

0x01, 0xFF, 0x01,
0x01, 0x8e, 0x01,

```
    0x01, 0x00, 0x01,  
    0x01, 0xFF, 0x00,  
    0x01, 0x80, 0x00,  
  
    0x00, 0x00, 0x00  
};
```

```
#ifdef __cplusplus  
{  
#endif
```

```
#endif /* _VL53L0X_TUNING_H_ */
```

1.1.26 VL53L0X_PLATFORM_H

Below:

```
/******
```

Copyright © 2015, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the
names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED. IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#ifndef _VL53L0X_PLATFORM_H_
```

```
#define _VL53L0X_PLATFORM_H_
```

```
#include "vl53l0x_def.h"
```

```
#include "vl53l0x_platform_log.h"
```

```
#include "vl53l0x_i2c_platform.h"
```

```
#ifdef __cplusplus
```

```
extern "C" {
```

```
#endif
```

```
/**
```

```
* @file vl53l0x_platform.h
```

```
*
```

```
* @brief All end user OS/platform/application porting
```

```
*/
```

```
/**
```

```
 * @defgroup VL53L0X_platform_group VL53L0 Platform Functions
```

```
 * @brief VL53L0 Platform Functions
```

```
 * @{
```

```
*/
```

```
/**
```

```
 * @struct VL53L0X_Dev_t
```

```
 * @brief Generic PAL device type that does link between API and platform abstraction layer
```

```
 *
```

```
*/
```

```
typedef struct {
```

```
    VL53L0X_DevData_t Data;          /*!< embed ST Ewok Dev data as "Data"*/
```

```
    /*!< user specific field */
```

```
    uint8_t I2cDevAddr;              /*!< i2c device address user specific field */
```

```
    uint8_t comms_type;              /*!< Type of comms : VL53L0X_COMMS_I2C or VL53L0X_COMMS_SPI  
*/
```

```
    uint16_t comms_speed_khz;        /*!< Comms speed [kHz] : typically 400kHz for I2C */
```

```
} VL53L0X_Dev_t;
```

```
/**
```

```
 * @brief Declare the device Handle as a pointer of the structure @a VL53L0X_Dev_t.
```

```
 *
```

```
*/
```

```
typedef VL53L0X_Dev_t* VL53L0X_DEV;
```

```
/**
```

```
 * @def PALDevDataGet
```

```
 * @brief Get ST private structure @a VL53L0X_DevData_t data access
```

```
 *
```

```
 * @param Dev    Device Handle
```

```
 * @param field  ST structure field name
```

```
 * It maybe used and as real data "ref" not just as "get" for sub-structure item
```

```
 * like PALDevDataGet(FilterData.field)[i] or PALDevDataGet(FilterData.MeasurementIndex)++
```

```
 */
```

```
#define PALDevDataGet(Dev, field) (Dev->Data.field)
```

```
/**
```

```
 * @def PALDevDataSet(Dev, field, data)
```

```
 * @brief Set ST private structure @a VL53L0X_DevData_t data field
```

```
 * @param Dev    Device Handle
```

```
 * @param field  ST structure field name
```

```
 * @param data   Data to be set
```

```
 */
```

```
#define PALDevDataSet(Dev, field, data) (Dev->Data.field)=(data)
```

```
/**
```

```
 * @defgroup VL53L0X_registerAccess_group PAL Register Access Functions
```

```
 * @brief  PAL Register Access Functions
```

```
 * @{
```

```
 */
```

/**

* Lock comms interface to serialize all commands to a shared I2C interface for a specific device

* @param Dev Device Handle

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_Error VL53L0X_LockSequenceAccess(VL53L0X_DEV Dev);

/**

* Unlock comms interface to serialize all commands to a shared I2C interface for a specific device

* @param Dev Device Handle

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_Error VL53L0X_UnlockSequenceAccess(VL53L0X_DEV Dev);

/**

* Writes the supplied byte buffer to the device

* @param Dev Device Handle

* @param index The register index

* @param pdata Pointer to uint8_t buffer containing the data to be written

* @param count Number of bytes in the supplied byte buffer

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_Error VL53L0X_WriteMulti(VL53L0X_DEV Dev, uint8_t index, uint8_t *pdata, uint32_t count);

/**

```

* Reads the requested number of bytes from the device

* @param Dev    Device Handle

* @param index  The register index

* @param pdata  Pointer to the uint8_t buffer to store read data

* @param count  Number of uint8_t's to read

* @return VL53L0X_ERROR_NONE    Success

* @return "Other error code"    See ::VL53L0X_Error

*/

```

```

VL53L0X_Error VL53L0X_ReadMulti(VL53L0X_DEV Dev, uint8_t index, uint8_t *pdata, uint32_t count);

```

```

/**

* Write single byte register

* @param Dev    Device Handle

* @param index  The register index

* @param data    8 bit register data

* @return VL53L0X_ERROR_NONE    Success

* @return "Other error code"    See ::VL53L0X_Error

*/

```

```

VL53L0X_Error VL53L0X_WrByte(VL53L0X_DEV Dev, uint8_t index, uint8_t data);

```

```

/**

* Write word register

* @param Dev    Device Handle

* @param index  The register index

* @param data    16 bit register data

* @return VL53L0X_ERROR_NONE    Success

* @return "Other error code"    See ::VL53L0X_Error

*/

```

```

VL53L0X_Error VL53L0X_WrWord(VL53L0X_DEV Dev, uint8_t index, uint16_t data);

```

/**

* Write double word (4 byte) register

* @param Dev Device Handle

* @param index The register index

* @param data 32 bit register data

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_Error VL53L0X_WrDWord(VL53L0X_DEV Dev, uint8_t index, uint32_t data);

/**

* Read single byte register

* @param Dev Device Handle

* @param index The register index

* @param data pointer to 8 bit data

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_Error VL53L0X_RdByte(VL53L0X_DEV Dev, uint8_t index, uint8_t *data);

/**

* Read word (2byte) register

* @param Dev Device Handle

* @param index The register index

* @param data pointer to 16 bit data

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_Error VL53L0X_RdWord(VL53L0X_DEV Dev, uint8_t index, uint16_t *data);
```

```
/**
```

```
 * Read dword (4byte) register
```

```
 * @param Dev    Device Handle
```

```
 * @param index  The register index
```

```
 * @param data   pointer to 32 bit data
```

```
 * @return VL53L0X_ERROR_NONE    Success
```

```
 * @return "Other error code"    See ::VL53L0X_Error
```

```
 */
```

```
VL53L0X_Error VL53L0X_RdDWord(VL53L0X_DEV Dev, uint8_t index, uint32_t *data);
```

```
/**
```

```
 * Threat safe Update (read/modify/write) single byte register
```

```
 *
```

```
 * Final_reg = (Initial_reg & and_data) | or_data
```

```
 *
```

```
 * @param Dev    Device Handle
```

```
 * @param index  The register index
```

```
 * @param AndData 8 bit and data
```

```
 * @param OrData  8 bit or data
```

```
 * @return VL53L0X_ERROR_NONE    Success
```

```
 * @return "Other error code"    See ::VL53L0X_Error
```

```
 */
```

```
VL53L0X_Error VL53L0X_UpdateByte(VL53L0X_DEV Dev, uint8_t index, uint8_t AndData, uint8_t OrData);
```

```
/** @} end of VL53L0X_registerAccess_group */
```

```

/**
 * @brief execute delay in all polling API call
 *
 * A typical multi-thread or RTOs implementation is to sleep the task for some 5ms (with 100Hz max rate
 faster polling is not needed)
 * if nothing specific is need you can define it as an empty/void macro
 * @code
 * #define VL53L0X_PollingDelay(...) (void)0
 * @endcode
 * @param Dev    Device Handle
 * @return VL53L0X_ERROR_NONE    Success
 * @return "Other error code"    See ::VL53L0X_Error
 */
VL53L0X_Error VL53L0X_PollingDelay(VL53L0X_DEV Dev); /* usually best implemented as a real function
 */

/** @} end of VL53L0X_platform_group */

#ifdef __cplusplus
}
#endif

#endif /* _VL53L0X_PLATFORM_H_ */

```

1.1.27 VL53L0x_platform.cpp

Below:

/******

Copyright © 2015, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the
names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

/**

```

* @file VL53L0X_i2c.c
*
* Copyright (C) 2014 ST MicroElectronics
*
* provide variable word size byte/Word/dword VL6180x register access via i2c
*
*/
#include "vl53l0x_platform.h"
#include "vl53l0x_i2c_platform.h"
#include "vl53l0x_api.h"

#define LOG_FUNCTION_START(fmt, ... )      _LOG_FUNCTION_START(TRACE_MODULE_PLATFORM,
fmt, ##__VA_ARGS__)

#define LOG_FUNCTION_END(status, ... )      _LOG_FUNCTION_END(TRACE_MODULE_PLATFORM,
status, ##__VA_ARGS__)

#define LOG_FUNCTION_END_FMT(status, fmt, ... )
_LOG_FUNCTION_END_FMT(TRACE_MODULE_PLATFORM, status, fmt, ##__VA_ARGS__)

/**
* @def I2C_BUFFER_CONFIG
*
* @brief Configure Device register I2C access
*
* @li 0 : one GLOBAL buffer \n
*   Use one global buffer of MAX_I2C_XFER_SIZE byte in data space \n
*   This solution is not multi-Device compliant nor multi-thread cpu safe \n
*   It can be the best option for small 8/16 bit MCU without stack and limited ram (STM8s, 80C51 ...)
*
* @li 1 : ON_STACK/local \n
*   Use local variable (on stack) buffer \n

```

```

* This solution is multi-thread with use of i2c resource lock or mutex see VL6180x_GetI2CAccess() \n
*
* @li 2 : User defined \n
* Per Device potentially dynamic allocated. Requires VL6180x_GetI2cBuffer() to be implemented.
* @ingroup Configuration
*/
#define I2C_BUFFER_CONFIG 1

/** Maximum buffer size to be used in i2c */
#define VL53L0X_MAX_I2C_XFER_SIZE 64 /* Maximum buffer size to be used in i2c */

#if I2C_BUFFER_CONFIG == 0
    /* GLOBAL config buffer */
    uint8_t i2c_global_buffer[VL53L0X_MAX_I2C_XFER_SIZE];

    #define DECL_I2C_BUFFER
    #define VL53L0X_GetLocalBuffer(Dev, n_byte) i2c_global_buffer

#elif I2C_BUFFER_CONFIG == 1
    /* ON STACK */
    #define DECL_I2C_BUFFER uint8_t LocBuffer[VL53L0X_MAX_I2C_XFER_SIZE];
    #define VL53L0X_GetLocalBuffer(Dev, n_byte) LocBuffer

#elif I2C_BUFFER_CONFIG == 2
    /* user define buffer type declare DECL_I2C_BUFFER as access via VL53L0X_GetLocalBuffer */
    #define DECL_I2C_BUFFER

#else
    #error "invalid I2C_BUFFER_CONFIG "
#endif

```

```
#define VL53L0X_I2C_USER_VAR    /* none but could be for a flag var to get/pass to mutex  
interruptible return flags and try again */
```

```
#define VL53L0X_GetI2CAccess(Dev) /* todo mutex acquire */
```

```
#define VL53L0X_DoneI2CAcces(Dev) /* todo mutex release */
```

```
VL53L0X_Error VL53L0X_LockSequenceAccess(VL53L0X_DEV Dev){
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    return Status;
```

```
}
```

```
VL53L0X_Error VL53L0X_UnlockSequenceAccess(VL53L0X_DEV Dev){
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    return Status;
```

```
}
```

```
// the ranging_sensor_comms.dll will take care of the page selection
```

```
VL53L0X_Error VL53L0X_WriteMulti(VL53L0X_DEV Dev, uint8_t index, uint8_t *pdata, uint32_t count){
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    int32_t status_int = 0;
```

```
    uint8_t deviceAddress;
```

```
    if (count >= VL53L0X_MAX_I2C_XFER_SIZE){
```

```
        Status = VL53L0X_ERROR_INVALID_PARAMS;
```

```
    }
```

```

    deviceAddress = Dev->I2cDevAddr;

    status_int = VL53L0X_write_multi(deviceAddress, index, pdata, count);

    if (status_int != 0)
        Status = VL53L0X_ERROR_CONTROL_INTERFACE;

    return Status;
}

// the ranging_sensor_comms.dll will take care of the page selection
VL53L0X_Error VL53L0X_ReadMulti(VL53L0X_DEV Dev, uint8_t index, uint8_t *pdata, uint32_t count){
    VL53L0X_I2C_USER_VAR
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t status_int;
    uint8_t deviceAddress;

    if (count >= VL53L0X_MAX_I2C_XFER_SIZE){
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }

    deviceAddress = Dev->I2cDevAddr;

    status_int = VL53L0X_read_multi(deviceAddress, index, pdata, count);

    if (status_int != 0)
        Status = VL53L0X_ERROR_CONTROL_INTERFACE;

    return Status;
}

```

```
}
```

```
VL53L0X_Error VL53L0X_WrByte(VL53L0X_DEV Dev, uint8_t index, uint8_t data){
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    int32_t status_int;
```

```
    uint8_t deviceAddress;
```

```
    deviceAddress = Dev->I2cDevAddr;
```

```
    status_int = VL53L0X_write_byte(deviceAddress, index, data);
```

```
    if (status_int != 0)
```

```
        Status = VL53L0X_ERROR_CONTROL_INTERFACE;
```

```
    return Status;
```

```
}
```

```
VL53L0X_Error VL53L0X_WrWord(VL53L0X_DEV Dev, uint8_t index, uint16_t data){
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    int32_t status_int;
```

```
    uint8_t deviceAddress;
```

```
    deviceAddress = Dev->I2cDevAddr;
```

```
    status_int = VL53L0X_write_word(deviceAddress, index, data);
```

```
    if (status_int != 0)
```

```
        Status = VL53L0X_ERROR_CONTROL_INTERFACE;
```

```
    return Status;
}
```

```
VL53L0X_Error VL53L0X_WrDWord(VL53L0X_DEV Dev, uint8_t index, uint32_t data){
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    int32_t status_int;
```

```
    uint8_t deviceAddress;
```

```
    deviceAddress = Dev->I2cDevAddr;
```

```
    status_int = VL53L0X_write_dword(deviceAddress, index, data);
```

```
    if (status_int != 0)
```

```
        Status = VL53L0X_ERROR_CONTROL_INTERFACE;
```

```
    return Status;
```

```
}
```

```
VL53L0X_Error VL53L0X_UpdateByte(VL53L0X_DEV Dev, uint8_t index, uint8_t AndData, uint8_t OrData){
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    int32_t status_int;
```

```
    uint8_t deviceAddress;
```

```
    uint8_t data;
```

```
    deviceAddress = Dev->I2cDevAddr;
```

```
    status_int = VL53L0X_read_byte(deviceAddress, index, &data);
```

```

if (status_int != 0)

    Status = VL53L0X_ERROR_CONTROL_INTERFACE;


if (Status == VL53L0X_ERROR_NONE) {

    data = (data & AndData) | OrData;

    status_int = VL53L0X_write_byte(deviceAddress, index, data);


    if (status_int != 0)

        Status = VL53L0X_ERROR_CONTROL_INTERFACE;
}


return Status;
}


VL53L0X_Error VL53L0X_RdByte(VL53L0X_DEV Dev, uint8_t index, uint8_t *data){

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    int32_t status_int;

    uint8_t deviceAddress;


    deviceAddress = Dev->I2cDevAddr;


    status_int = VL53L0X_read_byte(deviceAddress, index, data);


    if (status_int != 0)

        Status = VL53L0X_ERROR_CONTROL_INTERFACE;


    return Status;
}

```

```

VL53L0X_Error VL53L0X_RdWord(VL53L0X_DEV Dev, uint8_t index, uint16_t *data){
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    int32_t status_int;

    uint8_t deviceAddress;

    deviceAddress = Dev->I2cDevAddr;

    status_int = VL53L0X_read_word(deviceAddress, index, data);

    if (status_int != 0)
        Status = VL53L0X_ERROR_CONTROL_INTERFACE;

    return Status;
}

```

```

VL53L0X_Error VL53L0X_RdDWord(VL53L0X_DEV Dev, uint8_t index, uint32_t *data){
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    int32_t status_int;

    uint8_t deviceAddress;

    deviceAddress = Dev->I2cDevAddr;

    status_int = VL53L0X_read_dword(deviceAddress, index, data);

    if (status_int != 0)
        Status = VL53L0X_ERROR_CONTROL_INTERFACE;

    return Status;
}

```

```

}

#define VL53L0X_POLLINGDELAY_LOOPNB 250

VL53L0X_Error VL53L0X_PollingDelay(VL53L0X_DEV Dev){
    VL53L0X_Error status = VL53L0X_ERROR_NONE;
    volatile uint32_t i;
    LOG_FUNCTION_START("");

    for(i=0;i<VL53L0X_POLLINGDELAY_LOOPNB;i++){
        //Do nothing
        asm("nop");
    }

    LOG_FUNCTION_END(status);
    return status;
}

```

1.1.28 VL53L0x_platform_log.h

Below:

```

/*****

```

Copyright © 2015, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#ifndef _VL53L0X_PLATFORM_LOG_H_
#define _VL53L0X_PLATFORM_LOG_H_
```

```
#include <stdio.h>
#include <string.h>
/* LOG Functions */
```

```
#ifdef __cplusplus
extern "C" {
#endif
```

```

/**
 * @file vl53l0_platform_log.h
 *
 * @brief platform log function definition
 */

// #define VL53L0X_LOG_ENABLE 0

enum {
    TRACE_LEVEL_NONE,
    TRACE_LEVEL_ERRORS,
    TRACE_LEVEL_WARNING,
    TRACE_LEVEL_INFO,
    TRACE_LEVEL_DEBUG,
    TRACE_LEVEL_ALL,
    TRACE_LEVEL_IGNORE
};

enum {
    TRACE_FUNCTION_NONE = 0,
    TRACE_FUNCTION_I2C = 1,
    TRACE_FUNCTION_ALL = 0x7fffffff //all bits except sign
};

enum {
    TRACE_MODULE_NONE      = 0x0,
    TRACE_MODULE_API        = 0x1,
    TRACE_MODULE_PLATFORM   = 0x2,
    TRACE_MODULE_ALL        = 0x7fffffff //all bits except sign
};

```

```
};
```

```
#ifdef VL53L0X_LOG_ENABLE
```

```
#include <sys/time.h>
```

```
extern uint32_t _trace_level;
```

```
int32_t VL53L0X_trace_config(char *filename, uint32_t modules, uint32_t level, uint32_t functions);
```

```
void trace_print_module_function(uint32_t module, uint32_t level, uint32_t function, const char  
*format, ...);
```

```
//extern FILE * log_file;
```

```
#define LOG_GET_TIME() (int)clock()
```

```
#define _LOG_FUNCTION_START(module, fmt, ... ) \
```

```
    trace_print_module_function(module, _trace_level, TRACE_FUNCTION_ALL, "%ld <START> %s  
"fmt"\n", LOG_GET_TIME(), __FUNCTION__, ##__VA_ARGS__);
```

```
#define _LOG_FUNCTION_END(module, status, ... )\
```

```
    trace_print_module_function(module, _trace_level, TRACE_FUNCTION_ALL, "%ld <END> %s %d\n",  
LOG_GET_TIME(), __FUNCTION__, (int)status, ##__VA_ARGS__)
```

```
#define _LOG_FUNCTION_END_FMT(module, status, fmt, ... )\
```

```
    trace_print_module_function(module, _trace_level, TRACE_FUNCTION_ALL, "%ld <END> %s %d  
"fmt"\n", LOG_GET_TIME(), __FUNCTION__, (int)status, ##__VA_ARGS__)
```

```
// __func__ is gcc only
```

```
//#define VL53L0X_ErrLog( fmt, ...) fprintf(stderr, "VL53L0X_ErrLog %s" fmt "\n", __func__,  
##__VA_ARGS__)
```

```
#else /* VL53L0X_LOG_ENABLE no logging */
```

```
    #define VL53L0X_ErrLog(...) (void)0
```

```
    #define _LOG_FUNCTION_START(module, fmt, ... ) (void)0
```

```
    #define _LOG_FUNCTION_END(module, status, ... ) (void)0
```

```
    #define _LOG_FUNCTION_END_FMT(module, status, fmt, ... ) (void)0
```

```
#endif /* else */
```

```
#define VL53L0X_COPYSTRING(str, ...) strcpy(str, ##__VA_ARGS__)
```

```
#ifdef __cplusplus
```

```
}
```

```
#endif
```

```
#endif /* _VL53L0X_PLATFORM_LOG_H_ */
```

1.1.29 VL53L0x_i2c_comms.cpp

Below:

```
#include "vl53l0x_i2c_platform.h"
```

```
#include "vl53l0x_def.h"
```

```
//#define I2C_DEBUG
```

```
int VL53L0X_i2c_init(void) {  
    Wire.begin();  
    return VL53L0X_ERROR_NONE;  
}
```

```
int VL53L0X_write_multi(uint8_t deviceAddress, uint8_t index, uint8_t *pdata, uint32_t count) {  
    Wire.beginTransmission(deviceAddress);  
    Wire.write(index);  
#ifdef I2C_DEBUG  
    Serial.print("\tWriting "); Serial.print(count); Serial.print(" to addr 0x"); Serial.print(index, HEX);  
    Serial.print(": ");  
#endif  
    while(count--) {  
        Wire.write((uint8_t)pdata[0]);  
#ifdef I2C_DEBUG  
        Serial.print("0x"); Serial.print(pdata[0], HEX); Serial.print(", ");  
#endif  
        pdata++;  
    }  
#ifdef I2C_DEBUG  
    Serial.println();  
#endif  
    Wire.endTransmission();  
    return VL53L0X_ERROR_NONE;  
}
```

```
int VL53L0X_read_multi(uint8_t deviceAddress, uint8_t index, uint8_t *pdata, uint32_t count) {  
    Wire.beginTransmission(deviceAddress);  
    Wire.write(index);
```

```

Wire.endTransmission();

Wire.requestFrom(deviceAddress, (byte)count);

#ifdef I2C_DEBUG

    Serial.print("\tReading "); Serial.print(count); Serial.print(" from addr 0x"); Serial.print(index, HEX);
    Serial.print(": ");

#endif

    while (count-- > 0) {

        pdata[0] = Wire.read();

#ifdef I2C_DEBUG

        Serial.print("0x"); Serial.print(pdata[0], HEX); Serial.print(", ");

#endif

        pdata++;

    }

#ifdef I2C_DEBUG

    Serial.println();

#endif

    return VL53L0X_ERROR_NONE;

}

int VL53L0X_write_byte(uint8_t deviceAddress, uint8_t index, uint8_t data) {

    return VL53L0X_write_multi(deviceAddress, index, &data, 1);

}

int VL53L0X_write_word(uint8_t deviceAddress, uint8_t index, uint16_t data) {

    uint8_t buff[2];

    buff[1] = data & 0xFF;

    buff[0] = data >> 8;

    return VL53L0X_write_multi(deviceAddress, index, buff, 2);

```

```
}
```

```
int VL53L0X_write_dword(uint8_t deviceAddress, uint8_t index, uint32_t data) {
```

```
    uint8_t buff[4];
```

```
    buff[3] = data & 0xFF;
```

```
    buff[2] = data >> 8;
```

```
    buff[1] = data >> 16;
```

```
    buff[0] = data >> 24;
```

```
    return VL53L0X_write_multi(deviceAddress, index, buff, 4);
```

```
}
```

```
int VL53L0X_read_byte(uint8_t deviceAddress, uint8_t index, uint8_t *data) {
```

```
    return VL53L0X_read_multi(deviceAddress, index, data, 1);
```

```
}
```

```
int VL53L0X_read_word(uint8_t deviceAddress, uint8_t index, uint16_t *data) {
```

```
    uint8_t buff[2];
```

```
    int r = VL53L0X_read_multi(deviceAddress, index, buff, 2);
```

```
    uint16_t tmp;
```

```
    tmp = buff[0];
```

```
    tmp <<= 8;
```

```
    tmp |= buff[1];
```

```
    *data = tmp;
```

```
    return r;
```

```
}
```

```

int VL53L0X_read_dword(uint8_t deviceAddress, uint8_t index, uint32_t *data) {
    uint8_t buff[4];
    int r = VL53L0X_read_multi(deviceAddress, index, buff, 4);

    uint32_t tmp;
    tmp = buff[0];
    tmp <= 8;
    tmp |= buff[1];
    tmp <= 8;
    tmp |= buff[2];
    tmp <= 8;
    tmp |= buff[3];

    *data = tmp;

    return r;
}

```

1.1.30 VL53L0X_device.h

Below:

```

/*****

```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright

notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

/**

* Device specific defines. To be adapted by implementer for the targeted

* device.

*/

#ifndef _VL53L0X_DEVICE_H_

#define _VL53L0X_DEVICE_H_

#include "vl53l0x_types.h"

```

/** @defgroup VL53L0X_DevSpecDefines_group VL53L0X cut1.1 Device Specific Defines
 * @brief VL53L0X cut1.1 Device Specific Defines
 * @{
 */

```

```

/** @defgroup VL53L0X_DeviceError_group Device Error
 * @brief Device Error code
 *
 * This enum is Device specific it should be updated in the implementation
 * Use @a VL53L0X_GetStatusErrorString() to get the string.
 * It is related to Status Register of the Device.
 * @{
 */

```

```

typedef uint8_t VL53L0X_DeviceError;

```

```

#define VL53L0X_DEVICEERROR_NONE ((VL53L0X_DeviceError) 0)

/*!< 0 NoError */

#define VL53L0X_DEVICEERROR_VCSELCONTINUITYTESTFAILURE ((VL53L0X_DeviceError) 1)
#define VL53L0X_DEVICEERROR_VCSELWATCHDOGTESTFAILURE ((VL53L0X_DeviceError) 2)
#define VL53L0X_DEVICEERROR_NOVHVVALUEFOUND ((VL53L0X_DeviceError) 3)
#define VL53L0X_DEVICEERROR_MSRCNOTARGET ((VL53L0X_DeviceError) 4)
#define VL53L0X_DEVICEERROR_SNRCHECK ((VL53L0X_DeviceError) 5)
#define VL53L0X_DEVICEERROR_RANGEPHASECHECK ((VL53L0X_DeviceError) 6)
#define VL53L0X_DEVICEERROR_SIGMATHRESHOLDCHECK ((VL53L0X_DeviceError) 7)
#define VL53L0X_DEVICEERROR_TCC ((VL53L0X_DeviceError) 8)
#define VL53L0X_DEVICEERROR_PHASECONSISTENCY ((VL53L0X_DeviceError) 9)
#define VL53L0X_DEVICEERROR_MINCLIP ((VL53L0X_DeviceError) 10)

```

```
#define VL53L0X_DEVICEERROR_RANGECOMPLETE      ((VL53L0X_DeviceError) 11)
#define VL53L0X_DEVICEERROR_ALGOUNDERFLOW      ((VL53L0X_DeviceError) 12)
#define VL53L0X_DEVICEERROR_ALGOOVERFLOW       ((VL53L0X_DeviceError) 13)
#define VL53L0X_DEVICEERROR_RANGEIGNORETHRESHOLD ((VL53L0X_DeviceError) 14)
```

```
/** @} end of VL53L0X_DeviceError_group */
```

```
/** @defgroup VL53L0X_CheckEnable_group Check Enable list
```

```
* @brief Check Enable code
```

```
*
```

```
* Define used to specify the LimitCheckId.
```

```
* Use @a VL53L0X_GetLimitCheckInfo() to get the string.
```

```
* @{
```

```
*/
```

```
#define VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE    0
#define VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE 1
#define VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP     2
#define VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD 3
#define VL53L0X_CHECKENABLE_SIGNAL_RATE_MSRC    4
#define VL53L0X_CHECKENABLE_SIGNAL_RATE_PRE_RANGE 5
```

```
#define VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS    6
```

```
/** @} end of VL53L0X_CheckEnable_group */
```

```
/** @defgroup VL53L0X_GpioFunctionality_group Gpio Functionality
```

```

* @brief Defines the different functionalities for the device GPIO(s)
* @{
*/

typedef uint8_t VL53L0X_GpioFunctionality;

#define VL53L0X_GPIOFUNCTIONALITY_OFF \
    ((VL53L0X_GpioFunctionality) 0) /*!< NO Interrupt */
#define VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_LOW \
    ((VL53L0X_GpioFunctionality) 1) /*!< Level Low (value < thresh_low) */
#define VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_HIGH \
    ((VL53L0X_GpioFunctionality) 2) /*!< Level High (value > thresh_high) */
#define VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_OUT \
    ((VL53L0X_GpioFunctionality) 3)
    /*!< Out Of Window (value < thresh_low OR value > thresh_high) */
#define VL53L0X_GPIOFUNCTIONALITY_NEW_MEASURE_READY \
    ((VL53L0X_GpioFunctionality) 4) /*!< New Sample Ready */

/** @} end of VL53L0X_GpioFunctionality_group */

/* Device register map */

/** @defgroup VL53L0X_DefineRegisters_group Define Registers
* @brief List of all the defined registers
* @{
*/
#define VL53L0X_REG_SYSRANGE_START 0x000
    /** mask existing bit in #VL53L0X_REG_SYSRANGE_START*/
#define VL53L0X_REG_SYSRANGE_MODE_MASK 0x0F

```

```

/** bit 0 in #VL53L0X_REG_SYSRANGE_START write 1 toggle state in
 * continuous mode and arm next shot in single shot mode */
#define VL53L0X_REG_SYSRANGE_MODE_START_STOP 0x01

/** bit 1 write 0 in #VL53L0X_REG_SYSRANGE_START set single shot mode */
#define VL53L0X_REG_SYSRANGE_MODE_SINGLESOT 0x00

/** bit 1 write 1 in #VL53L0X_REG_SYSRANGE_START set back-to-back
 * operation mode */
#define VL53L0X_REG_SYSRANGE_MODE_BACKTOBACK 0x02

/** bit 2 write 1 in #VL53L0X_REG_SYSRANGE_START set timed operation
 * mode */
#define VL53L0X_REG_SYSRANGE_MODE_TIMED 0x04

/** bit 3 write 1 in #VL53L0X_REG_SYSRANGE_START set histogram operation
 * mode */
#define VL53L0X_REG_SYSRANGE_MODE_HISTOGRAM 0x08


#define VL53L0X_REG_SYSTEM_THRESH_HIGH 0x000C
#define VL53L0X_REG_SYSTEM_THRESH_LOW 0x000E


#define VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG 0x0001
#define VL53L0X_REG_SYSTEM_RANGE_CONFIG 0x0009
#define VL53L0X_REG_SYSTEM_INTERMEASUREMENT_PERIOD 0x0004


#define VL53L0X_REG_SYSTEM_INTERRUPT_CONFIG_GPIO 0x000A

#define VL53L0X_REG_SYSTEM_INTERRUPT_GPIO_DISABLED 0x00
#define VL53L0X_REG_SYSTEM_INTERRUPT_GPIO_LEVEL_LOW 0x01
#define VL53L0X_REG_SYSTEM_INTERRUPT_GPIO_LEVEL_HIGH 0x02

```

```

#define VL53L0X_REG_SYSTEM_INTERRUPT_GPIO_OUT_OF_WINDOW 0x03

#define VL53L0X_REG_SYSTEM_INTERRUPT_GPIO_NEW_SAMPLE_READY 0x04


#define VL53L0X_REG_GPIO_HV_MUX_ACTIVE_HIGH 0x0084


#define VL53L0X_REG_SYSTEM_INTERRUPT_CLEAR 0x000B


/* Result registers */
#define VL53L0X_REG_RESULT_INTERRUPT_STATUS 0x0013
#define VL53L0X_REG_RESULT_RANGE_STATUS 0x0014


#define VL53L0X_REG_RESULT_CORE_PAGE 1
#define VL53L0X_REG_RESULT_CORE_AMBIENT_WINDOW_EVENTS_RTN 0x00BC
#define VL53L0X_REG_RESULT_CORE_RANGING_TOTAL_EVENTS_RTN 0x00C0
#define VL53L0X_REG_RESULT_CORE_AMBIENT_WINDOW_EVENTS_REF 0x00D0
#define VL53L0X_REG_RESULT_CORE_RANGING_TOTAL_EVENTS_REF 0x00D4
#define VL53L0X_REG_RESULT_PEAK_SIGNAL_RATE_REF 0x00B6


/* Algo register */

#define VL53L0X_REG_ALGO_PART_TO_PART_RANGE_OFFSET_MM 0x0028


#define VL53L0X_REG_I2C_SLAVE_DEVICE_ADDRESS 0x008a


/* Check Limit registers */
#define VL53L0X_REG_MSRC_CONFIG_CONTROL 0x0060


#define VL53L0X_REG_PRE_RANGE_CONFIG_MIN_SNR 0x0027

```

```

#define VL53L0X_REG_PRE_RANGE_CONFIG_VALID_PHASE_LOW      0x0056
#define VL53L0X_REG_PRE_RANGE_CONFIG_VALID_PHASE_HIGH    0x0057
#define VL53L0X_REG_PRE_RANGE_MIN_COUNT_RATE_RTN_LIMIT    0x0064

#define VL53L0X_REG_FINAL_RANGE_CONFIG_MIN_SNR            0x0067
#define VL53L0X_REG_FINAL_RANGE_CONFIG_VALID_PHASE_LOW    0x0047
#define VL53L0X_REG_FINAL_RANGE_CONFIG_VALID_PHASE_HIGH    0x0048
#define VL53L0X_REG_FINAL_RANGE_CONFIG_MIN_COUNT_RATE_RTN_LIMIT 0x0044

#define VL53L0X_REG_PRE_RANGE_CONFIG_SIGMA_THRESH_HI      0x0061
#define VL53L0X_REG_PRE_RANGE_CONFIG_SIGMA_THRESH_LO      0x0062

/* PRE RANGE registers */
#define VL53L0X_REG_PRE_RANGE_CONFIG_VCSEL_PERIOD          0x0050
#define VL53L0X_REG_PRE_RANGE_CONFIG_TIMEOUT_MACROP_HI     0x0051
#define VL53L0X_REG_PRE_RANGE_CONFIG_TIMEOUT_MACROP_LO     0x0052

#define VL53L0X_REG_SYSTEM_HISTOGRAM_BIN                   0x0081
#define VL53L0X_REG_HISTOGRAM_CONFIG_INITIAL_PHASE_SELECT 0x0033
#define VL53L0X_REG_HISTOGRAM_CONFIG_READOUT_CTRL          0x0055

#define VL53L0X_REG_FINAL_RANGE_CONFIG_VCSEL_PERIOD        0x0070
#define VL53L0X_REG_FINAL_RANGE_CONFIG_TIMEOUT_MACROP_HI   0x0071
#define VL53L0X_REG_FINAL_RANGE_CONFIG_TIMEOUT_MACROP_LO   0x0072
#define VL53L0X_REG_CROSSTALK_COMPENSATION_PEAK_RATE_MCPS  0x0020

#define VL53L0X_REG_MSRC_CONFIG_TIMEOUT_MACROP              0x0046

```

```

#define VL53L0X_REG_SOFT_RESET_GO2_SOFT_RESET_N          0x00bf

#define VL53L0X_REG_IDENTIFICATION_MODEL_ID              0x00c0

#define VL53L0X_REG_IDENTIFICATION_REVISION_ID           0x00c2


#define VL53L0X_REG_OSC_CALIBRATE_VAL                    0x00f8


#define VL53L0X_SIGMA_ESTIMATE_MAX_VALUE                  65535
/* equivalent to a range sigma of 655.35mm */


#define VL53L0X_REG_GLOBAL_CONFIG_VCSEL_WIDTH            0x032
#define VL53L0X_REG_GLOBAL_CONFIG_SPAD_ENABLES_REF_0     0x0B0
#define VL53L0X_REG_GLOBAL_CONFIG_SPAD_ENABLES_REF_1     0x0B1
#define VL53L0X_REG_GLOBAL_CONFIG_SPAD_ENABLES_REF_2     0x0B2
#define VL53L0X_REG_GLOBAL_CONFIG_SPAD_ENABLES_REF_3     0x0B3
#define VL53L0X_REG_GLOBAL_CONFIG_SPAD_ENABLES_REF_4     0x0B4
#define VL53L0X_REG_GLOBAL_CONFIG_SPAD_ENABLES_REF_5     0x0B5


#define VL53L0X_REG_GLOBAL_CONFIG_REF_EN_START_SELECT    0xB6
#define VL53L0X_REG_DYNAMIC_SPAD_NUM_REQUESTED_REF_SPAD  0x4E /* 0x14E */
#define VL53L0X_REG_DYNAMIC_SPAD_REF_EN_START_OFFSET     0x4F /* 0x14F */
#define VL53L0X_REG_POWER_MANAGEMENT_GO1_POWER_FORCE     0x80


/*
 * Speed of light in um per 1E-10 Seconds
 */

#define VL53L0X_SPEED_OF_LIGHT_IN_AIR 2997

```

```
#define VL53L0X_REG_VHV_CONFIG_PAD_SCL_SDA__EXTSUP_HV 0x0089
```

```
#define VL53L0X_REG_ALGO_PHASECAL_LIM 0x0030 /* 0x130 */
```

```
#define VL53L0X_REG_ALGO_PHASECAL_CONFIG_TIMEOUT 0x0030
```

```
/** @} VL53L0X_DefineRegisters_group */
```

```
/** @} VL53L0X_DevSpecDefines_group */
```

```
#endif
```

```
/* _VL53L0X_DEVICE_H_ */
```

1.1.31 VL53L0x_def.h

Below:

```
VI5 /*****
```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

/**

* @file VL53L0X_def.h

*

* @brief Type definitions for VL53L0X API.

*

*/

#ifndef _VL53L0X_DEF_H_

#define _VL53L0X_DEF_H_

```

#ifdef __cplusplus
extern "C" {
#endif

/** @defgroup VL53L0X_globaldefine_group VL53L0X Defines
 *
 * @brief VL53L0X Defines
 *
 * @{
 */

```

```

/** PAL SPECIFICATION major version */
#define VL53L0X10_SPECIFICATION_VER_MAJOR 1

/** PAL SPECIFICATION minor version */
#define VL53L0X10_SPECIFICATION_VER_MINOR 2

/** PAL SPECIFICATION sub version */
#define VL53L0X10_SPECIFICATION_VER_SUB 7

/** PAL SPECIFICATION sub version */
#define VL53L0X10_SPECIFICATION_VER_REVISION 1440

```

```

/** VL53L0X PAL IMPLEMENTATION major version */
#define VL53L0X10_IMPLEMENTATION_VER_MAJOR 1

/** VL53L0X PAL IMPLEMENTATION minor version */
#define VL53L0X10_IMPLEMENTATION_VER_MINOR 0

/** VL53L0X PAL IMPLEMENTATION sub version */
#define VL53L0X10_IMPLEMENTATION_VER_SUB 9

/** VL53L0X PAL IMPLEMENTATION sub version */
#define VL53L0X10_IMPLEMENTATION_VER_REVISION 3673

```

```

/** PAL SPECIFICATION major version */

```

```

#define VL53L0X_SPECIFICATION_VER_MAJOR 1

/** PAL SPECIFICATION minor version */

#define VL53L0X_SPECIFICATION_VER_MINOR 2

/** PAL SPECIFICATION sub version */

#define VL53L0X_SPECIFICATION_VER_SUB 7

/** PAL SPECIFICATION sub version */

#define VL53L0X_SPECIFICATION_VER_REVISION 1440


/** VL53L0X PAL IMPLEMENTATION major version */

#define VL53L0X_IMPLEMENTATION_VER_MAJOR 1

/** VL53L0X PAL IMPLEMENTATION minor version */

#define VL53L0X_IMPLEMENTATION_VER_MINOR 0

/** VL53L0X PAL IMPLEMENTATION sub version */

#define VL53L0X_IMPLEMENTATION_VER_SUB 1

/** VL53L0X PAL IMPLEMENTATION sub version */

#define VL53L0X_IMPLEMENTATION_VER_REVISION 4606

#define VL53L0X_DEFAULT_MAX_LOOP 200

#define VL53L0X_MAX_STRING_LENGTH 32


#include "vl53l0x_device.h"

#include "vl53l0x_types.h"


/*****

* PRIVATE define do not edit

*****/

/** @brief Defines the parameters of the Get Version Functions

```

```
*/
```

```
typedef struct {
```

```
    uint32_t      revision; /*!< revision number */
```

```
    uint8_t       major;   /*!< major number */
```

```
    uint8_t       minor;   /*!< minor number */
```

```
    uint8_t       build;    /*!< build number */
```

```
} VL53L0X_Version_t;
```

```
/** @brief Defines the parameters of the Get Device Info Functions
```

```
*/
```

```
typedef struct {
```

```
    char Name[VL53L0X_MAX_STRING_LENGTH];
```

```
        /*!< Name of the Device e.g. Left_Distance */
```

```
    char Type[VL53L0X_MAX_STRING_LENGTH];
```

```
        /*!< Type of the Device e.g VL53L0X */
```

```
    char ProductId[VL53L0X_MAX_STRING_LENGTH];
```

```
        /*!< Product Identifier String    */
```

```
    uint8_t ProductType;
```

```
        /*!< Product Type, VL53L0X = 1, VL53L1 = 2 */
```

```
    uint8_t ProductRevisionMajor;
```

```
        /*!< Product revision major */
```

```
    uint8_t ProductRevisionMinor;
```

```
        /*!< Product revision minor */
```

```
} VL53L0X_DeviceInfo_t;
```

```
/** @defgroup VL53L0X_define_Error_group Error and Warning code returned by API
```

```
*      The following DEFINE are used to identify the PAL ERROR
```

```
*      @{  
*/
```

```
typedef int8_t VL53L0X_Error;
```

```
#define VL53L0X_ERROR_NONE          ((VL53L0X_Error) 0)
```

```
#define VL53L0X_ERROR_CALIBRATION_WARNING ((VL53L0X_Error) -1)
```

```
    /*!< Warning invalid calibration data may be in used
```

```
        \a      VL53L0X_InitData()
```

```
        \a VL53L0X_GetOffsetCalibrationData
```

```
        \a VL53L0X_SetOffsetCalibrationData */
```

```
#define VL53L0X_ERROR_MIN_CLIPPED      ((VL53L0X_Error) -2)
```

```
    /*!< Warning parameter passed was clipped to min before to be applied */
```

```
#define VL53L0X_ERROR_UNDEFINED          ((VL53L0X_Error) -3)
```

```
    /*!< Unqualified error */
```

```
#define VL53L0X_ERROR_INVALID_PARAMS      ((VL53L0X_Error) -4)
```

```
    /*!< Parameter passed is invalid or out of range */
```

```
#define VL53L0X_ERROR_NOT_SUPPORTED        ((VL53L0X_Error) -5)
```

```
    /*!< Function is not supported in current mode or configuration */
```

```
#define VL53L0X_ERROR_RANGE_ERROR          ((VL53L0X_Error) -6)
```

```
    /*!< Device report a ranging error interrupt status */
```

```
#define VL53L0X_ERROR_TIME_OUT             ((VL53L0X_Error) -7)
```

```
    /*!< Aborted due to time out */
```

```
#define VL53L0X_ERROR_MODE_NOT_SUPPORTED    ((VL53L0X_Error) -8)
```

```
    /*!< Asked mode is not supported by the device */
```

```
#define VL53L0X_ERROR_BUFFER_TOO_SMALL      ((VL53L0X_Error) -9)
```

```
    /*!< ... */
```

```
#define VL53L0X_ERROR_GPIO_NOT_EXISTING    ((VL53L0X_Error) -10)
```

```

    /*!< User tried to setup a non-existing GPIO pin */

#define VL53L0X_ERROR_GPIO_FUNCTIONALITY_NOT_SUPPORTED ((VL53L0X_Error) -11)

    /*!< unsupported GPIO functionality */

#define VL53L0X_ERROR_INTERRUPT_NOT_CLEARED          ((VL53L0X_Error) -12)

    /*!< Error during interrupt clear */

#define VL53L0X_ERROR_CONTROL_INTERFACE              ((VL53L0X_Error) -20)

    /*!< error reported from IO functions */

#define VL53L0X_ERROR_INVALID_COMMAND                ((VL53L0X_Error) -30)

    /*!< The command is not allowed in the current device state
    *      (power down) */

#define VL53L0X_ERROR_DIVISION_BY_ZERO                ((VL53L0X_Error) -40)

    /*!< In the function a division by zero occurs */

#define VL53L0X_ERROR_REF_SPAD_INIT                  ((VL53L0X_Error) -50)

    /*!< Error during reference SPAD initialization */

#define VL53L0X_ERROR_NOT_IMPLEMENTED                ((VL53L0X_Error) -99)

    /*!< Tells requested functionality has not been implemented yet or
    *      not compatible with the device */

/** @} VL53L0X_define_Error_group */

/** @defgroup VL53L0X_define_DeviceModes_group Defines Device modes
    *
    * Defines all possible modes for the device
    *
    * @{
    */

typedef uint8_t VL53L0X_DeviceModes;

#define VL53L0X_DEVICEMODE_SINGLE_RANGING            ((VL53L0X_DeviceModes) 0)
#define VL53L0X_DEVICEMODE_CONTINUOUS_RANGING        ((VL53L0X_DeviceModes) 1)
#define VL53L0X_DEVICEMODE_SINGLE_HISTOGRAM          ((VL53L0X_DeviceModes) 2)

```

```

#define VL53L0X_DEVICEMODE_CONTINUOUS_TIMED_RANGING ((VL53L0X_DeviceModes) 3)

#define VL53L0X_DEVICEMODE_SINGLE_ALS                ((VL53L0X_DeviceModes) 10)

#define VL53L0X_DEVICEMODE_GPIO_DRIVE                ((VL53L0X_DeviceModes) 20)

#define VL53L0X_DEVICEMODE_GPIO_OSC                  ((VL53L0X_DeviceModes) 21)

        /* ... Modes to be added depending on device */

/** @} VL53L0X_define_DeviceModes_group */

```

```

/** @defgroup VL53L0X_define_HistogramModes_group Defines Histogram modes

*      Defines all possible Histogram modes for the device

*      @{

*/

typedef uint8_t VL53L0X_HistogramModes;

```

```

#define VL53L0X_HISTOGRAMMODE_DISABLED                ((VL53L0X_HistogramModes) 0)

        /*!< Histogram Disabled */

#define VL53L0X_HISTOGRAMMODE_REFERENCE_ONLY((VL53L0X_HistogramModes) 1)

        /*!< Histogram Reference array only */

#define VL53L0X_HISTOGRAMMODE_RETURN_ONLY    ((VL53L0X_HistogramModes) 2)

        /*!< Histogram Return array only */

#define VL53L0X_HISTOGRAMMODE_BOTH                ((VL53L0X_HistogramModes) 3)

        /*!< Histogram both Reference and Return Arrays */

        /* ... Modes to be added depending on device */

/** @} VL53L0X_define_HistogramModes_group */

```

```

/** @defgroup VL53L0X_define_PowerModes_group List of available Power Modes

*      List of available Power Modes

```

```
*      @{\n*/
```

```
typedef uint8_t VL53L0X_PowerModes;
```

```
#define VL53L0X_POWERMODE_STANDBY_LEVEL1 ((VL53L0X_PowerModes) 0)
```

```
    /*!< Standby level 1 */
```

```
#define VL53L0X_POWERMODE_STANDBY_LEVEL2 ((VL53L0X_PowerModes) 1)
```

```
    /*!< Standby level 2 */
```

```
#define VL53L0X_POWERMODE_IDLE_LEVEL1  ((VL53L0X_PowerModes) 2)
```

```
    /*!< Idle level 1 */
```

```
#define VL53L0X_POWERMODE_IDLE_LEVEL2  ((VL53L0X_PowerModes) 3)
```

```
    /*!< Idle level 2 */
```

```
/** @} VL53L0X_define_PowerModes_group */
```

```
/** @brief Defines all parameters for the device
```

```
*/
```

```
typedef struct {
```

```
    VL53L0X_DeviceModes DeviceMode;
```

```
    /*!< Defines type of measurement to be done for the next measure */
```

```
    VL53L0X_HistogramModes HistogramMode;
```

```
    /*!< Defines type of histogram measurement to be done for the next
```

```
    *      measure */
```

```
    uint32_t MeasurementTimingBudgetMicroSeconds;
```

```
    /*!< Defines the allowed total time for a single measurement */
```

```
    uint32_t InterMeasurementPeriodMilliSeconds;
```

```
    /*!< Defines time between two consecutive measurements (between two
```

```

    *      measurement starts). If set to 0 means back-to-back mode */
uint8_t XTalkCompensationEnable;

/*!< Tells if Crosstalk compensation shall be enable or not      */
uint16_t XTalkCompensationRangeMilliMeter;

/*!< CrossTalk compensation range in millimeter      */
FixPoint1616_t XTalkCompensationRateMegaCps;

/*!< CrossTalk compensation rate in Mega counts per seconds.

    *      Expressed in 16.16 fixed point format.  */
int32_t RangeOffsetMicroMeters;

/*!< Range offset adjustment (mm).      */


uint8_t LimitChecksEnable[VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS];
/*!< This Array store all the Limit Check enable for this device. */
uint8_t LimitChecksStatus[VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS];
/*!< This Array store all the Status of the check linked to last
    * measurement. */
FixPoint1616_t LimitChecksValue[VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS];
/*!< This Array store all the Limit Check value for this device */


uint8_t WrapAroundCheckEnable;

/*!< Tells if Wrap Around Check shall be enable or not */
} VL53L0X_DeviceParameters_t;


/** @defgroup VL53L0X_define_State_group Defines the current status of the device
    *
    * Defines the current status of the device
    *
    * @{
    */

```

```

typedef uint8_t VL53L0X_State;

#define VL53L0X_STATE_POWERDOWN      ((VL53L0X_State) 0)
    /*!< Device is in HW reset      */
#define VL53L0X_STATE_WAIT_STATICINIT ((VL53L0X_State) 1)
    /*!< Device is initialized and wait for static initialization */
#define VL53L0X_STATE_STANDBY        ((VL53L0X_State) 2)
    /*!< Device is in Low power Standby mode */
#define VL53L0X_STATE_IDLE            ((VL53L0X_State) 3)
    /*!< Device has been initialized and ready to do measurements */
#define VL53L0X_STATE_RUNNING         ((VL53L0X_State) 4)
    /*!< Device is performing measurement */
#define VL53L0X_STATE_UNKNOWN         ((VL53L0X_State) 98)
    /*!< Device is in unknown state and need to be rebooted      */
#define VL53L0X_STATE_ERROR           ((VL53L0X_State) 99)
    /*!< Device is in error state and need to be rebooted */

/** @} VL53L0X_define_State_group */

/** @brief Structure containing the Dmax computation parameters and data
 */
typedef struct {
    int32_t AmbTuningWindowFactor_K;
        /*!< internal algo tuning (*1000) */
    int32_t RetSignalAt0mm;
        /*!< intermediate dmax computation value caching */
} VL53L0X_DMaxData_t;

```

```

/**
 * @struct VL53LOX_RangeData_t
 * @brief Range measurement data.
 */
typedef struct {
    uint32_t TimeStamp;          /*!< 32-bit time stamp. */
    uint32_t MeasurementTimeUsec;
                                /*!< Give the Measurement time needed by the device to do the
                                * measurement.*/

    uint16_t RangeMilliMeter;    /*!< range distance in millimeter. */

    uint16_t RangeDMaxMilliMeter;
                                /*!< Tells what is the maximum detection distance of the device
                                * in current setup and environment conditions (Filled when
                                * applicable) */

    FixPoint1616_t SignalRateRtnMegaCps;
                                /*!< Return signal rate (MCPS)\n these is a 16.16 fix point
                                * value, which is effectively a measure of target
                                * reflectance.*/

    FixPoint1616_t AmbientRateRtnMegaCps;
                                /*!< Return ambient rate (MCPS)\n these is a 16.16 fix point
                                * value, which is effectively a measure of the ambien
                                * t light.*/

    uint16_t EffectiveSpadRtnCount;
                                /*!< Return the effective SPAD count for the return signal.

```

```

        *      To obtain Real value it should be divided by 256 */

uint8_t ZoneId;

    /*!< Denotes which zone and range scheduler stage the range
        *      data relates to. */

uint8_t RangeFractionalPart;

    /*!< Fractional part of range distance. Final value is a
        *      FixPoint168 value. */

uint8_t RangeStatus;

    /*!< Range Status for the current measurement. This is device
        *      dependent. Value = 0 means value is valid.
        *      See \ref RangeStatusPage */
} VL53L0X_RangingMeasurementData_t;

```

```

#define VL53L0X_HISTOGRAM_BUFFER_SIZE 24

```

```

/**
 * @struct VL53L0X_HistogramData_t
 * @brief Histogram measurement data.
 */

typedef struct {
    /* Histogram Measurement data */
    uint32_t HistogramData[VL53L0X_HISTOGRAM_BUFFER_SIZE];
    /*!< Histogram data */
    uint8_t HistogramType; /*!< Indicate the types of histogram data :
    Return only, Reference only, both Return and Reference */
    uint8_t FirstBin; /*!< First Bin value */
    uint8_t BufferSize; /*!< Buffer Size - Set by the user.*/

```

```

uint8_t NumberOfBins;

/*!< Number of bins filled by the histogram measurement */

VL53L0X_DeviceError ErrorStatus;

/*!< Error status of the current measurement. \n
see @a ::VL53L0X_DeviceError @a VL53L0X_GetStatusErrorString() */
} VL53L0X_HistogramMeasurementData_t;

#define VL53L0X_REF_SPAD_BUFFER_SIZE 6

/**
 * @struct VL53L0X_SpadData_t
 * @brief Spad Configuration Data.
 */
typedef struct {
    uint8_t RefSpadEnables[VL53L0X_REF_SPAD_BUFFER_SIZE];
    /*!< Reference Spad Enables */
    uint8_t RefGoodSpadMap[VL53L0X_REF_SPAD_BUFFER_SIZE];
    /*!< Reference Spad Good Spad Map */
} VL53L0X_SpadData_t;

typedef struct {
    FixPoint1616_t OscFrequencyMHz; /* Frequency used */

    uint16_t LastEncodedTimeout;
    /* last encoded Time out used for timing budget*/

    VL53L0X_GpioFunctionality Pin0GpioFunctionality;
    /* store the functionality of the GPIO: pin0 */

```

```

uint32_t FinalRangeTimeoutMicroSecs;

/*!< Execution time of the final range*/

uint8_t FinalRangeVcselPulsePeriod;

/*!< Vcsel pulse period (pll clocks) for the final range measurement*/

uint32_t PreRangeTimeoutMicroSecs;

/*!< Execution time of the final range*/

uint8_t PreRangeVcselPulsePeriod;

/*!< Vcsel pulse period (pll clocks) for the pre-range measurement*/


uint16_t SigmaEstRefArray;

/*!< Reference array sigma value in 1/100th of [mm] e.g. 100 = 1mm */

uint16_t SigmaEstEffPulseWidth;

/*!< Effective Pulse width for sigma estimate in 1/100th
 * of ns e.g. 900 = 9.0ns */

uint16_t SigmaEstEffAmbWidth;

/*!< Effective Ambient width for sigma estimate in 1/100th of ns
 * e.g. 500 = 5.0ns */


uint8_t ReadDataFromDeviceDone; /* Indicate if read from device has
been done (==1) or not (==0) */

uint8_t ModuleId; /* Module ID */

uint8_t Revision; /* test Revision */

char ProductId[VL53L0X_MAX_STRING_LENGTH];

/* Product Identifier String */

uint8_t ReferenceSpadCount; /* used for ref spad management */

uint8_t ReferenceSpadType; /* used for ref spad management */

uint8_t RefSpadsInitialised; /* reports if ref spads are initialised. */

```

```

uint32_t PartUIDUpper; /*!< Unique Part ID Upper */

uint32_t PartUIDLower; /*!< Unique Part ID Lower */

FixPoint1616_t SignalRateMeasFixed400mm; /*!< Peek Signal rate
at 400 mm*/

} VL53L0X_DeviceSpecificParameters_t;

/**
 * @struct VL53L0X_DevData_t
 *
 * @brief VL53L0X PAL device ST private data structure \n
 * End user should never access any of these field directly
 *
 * These must never access directly but only via macro
 */
typedef struct {
    VL53L0X_DMaxData_t DMaxData;

    /*!< Dmax Data */

    int32_t Part2PartOffsetNVMMicroMeter;

    /*!< backed up NVM value */

    int32_t Part2PartOffsetAdjustmentNVMMicroMeter;

    /*!< backed up NVM value representing additional offset adjustment */

    VL53L0X_DeviceParameters_t CurrentParameters;

    /*!< Current Device Parameter */

    VL53L0X_RangingMeasurementData_t LastRangeMeasure;

    /*!< Ranging Data */

    VL53L0X_HistogramMeasurementData_t LastHistogramMeasure;

    /*!< Histogram Data */

    VL53L0X_DeviceSpecificParameters_t DeviceSpecificParameters;

```

```

/*!< Parameters specific to the device */
VL53L0X_SpadData_t SpadData;

/*!< Spad Data */
uint8_t SequenceConfig;

/*!< Internal value for the sequence config */
uint8_t RangeFractionalEnable;

/*!< Enable/Disable fractional part of ranging data */
VL53L0X_State PalState;

/*!< Current state of the PAL for this device */
VL53L0X_PowerModes PowerMode;

/*!< Current Power Mode      */
uint16_t SigmaEstRefArray;

/*!< Reference array sigma value in 1/100th of [mm] e.g. 100 = 1mm */
uint16_t SigmaEstEffPulseWidth;

/*!< Effective Pulse width for sigma estimate in 1/100th
* of ns e.g. 900 = 9.0ns */
uint16_t SigmaEstEffAmbWidth;

/*!< Effective Ambient width for sigma estimate in 1/100th of ns
* e.g. 500 = 5.0ns */
uint8_t StopVariable;

/*!< StopVariable used during the stop sequence */
uint16_t targetRefRate;

/*!< Target Ambient Rate for Ref spad management */
FixPoint1616_t SigmaEstimate;

/*!< Sigma Estimate - based on ambient & VCSEL rates and
* signal_total_events */
FixPoint1616_t SignalEstimate;

/*!< Signal Estimate - based on ambient & VCSEL rates and cross talk */
FixPoint1616_t LastSignalRefMcps;

```

```

    /*!< Latest Signal ref in Mcps */
    uint8_t *pTuningSettingsPointer;

    /*!< Pointer for Tuning Settings table */
    uint8_t UseInternalTuningSettings;

    /*!< Indicate if we use Tuning Settings table */
    uint16_t LinearityCorrectiveGain;

    /*!< Linearity Corrective Gain value in x1000 */
    uint16_t DmaxCalRangeMilliMeter;

    /*!< Dmax Calibration Range millimeter */
    FixPoint1616_t DmaxCalSignalRateRtnMegaCps;

    /*!< Dmax Calibration Signal Rate Return MegaCps */

} VL53L0X_DevData_t;

/** @defgroup VL53L0X_define_InterruptPolarity_group Defines the Polarity
 * of the Interrupt
 *
 * Defines the Polarity of the Interrupt
 *
 * @{
 */
typedef uint8_t VL53L0X_InterruptPolarity;

#define VL53L0X_INTERRUPTPOLARITY_LOW ((VL53L0X_InterruptPolarity) 0)
/*!< Set active low polarity best setup for falling edge. */
#define VL53L0X_INTERRUPTPOLARITY_HIGH ((VL53L0X_InterruptPolarity) 1)
/*!< Set active high polarity best setup for rising edge. */

/** @} VL53L0X_define_InterruptPolarity_group */

```

```

/** @defgroup VL53L0X_define_VcselPeriod_group Vcsel Period Defines
 *
 * Defines the range measurement for which to access the vcsel period.
 *
 * @{
 */

typedef uint8_t VL53L0X_VcselPeriod;

#define VL53L0X_VCSEL_PERIOD_PRE_RANGE ((VL53L0X_VcselPeriod) 0)
/*!<Identifies the pre-range vcsel period. */

#define VL53L0X_VCSEL_PERIOD_FINAL_RANGE ((VL53L0X_VcselPeriod) 1)
/*!<Identifies the final range vcsel period. */

/** @} VL53L0X_define_VcselPeriod_group */

/** @defgroup VL53L0X_define_SchedulerSequence_group Defines the steps
 *
 * carried out by the scheduler during a range measurement.
 *
 * @{
 *
 * Defines the states of all the steps in the scheduler
 *
 * i.e. enabled/disabled.
 *
 */

typedef struct {
    uint8_t      TccOn; /*!<Reports if Target Centre Check On */
    uint8_t      MsrcOn; /*!<Reports if MSRC On */
    uint8_t      DssOn; /*!<Reports if DSS On */
    uint8_t      PreRangeOn; /*!<Reports if Pre-Range On */
    uint8_t      FinalRangeOn; /*!<Reports if Final-Range On */
} VL53L0X_SchedulerSequenceSteps_t;

/** @} VL53L0X_define_SchedulerSequence_group */

```

```

/** @defgroup VL53L0X_define_SequenceStepId_group Defines the Polarity
*
* of the Interrupt
*
* Defines the the sequence steps performed during ranging..
*
* @{
*/

typedef uint8_t VL53L0X_SequenceStepId;

#define VL53L0X_SEQUENCESTEP_TCC ((VL53L0X_VcseIPeriod) 0)
/*!<Target CentreCheck identifier. */
#define VL53L0X_SEQUENCESTEP_DSS ((VL53L0X_VcseIPeriod) 1)
/*!<Dynamic Spad Selection function Identifier. */
#define VL53L0X_SEQUENCESTEP_MSRC ((VL53L0X_VcseIPeriod) 2)
/*!<Minimum Signal Rate Check function Identifier. */
#define VL53L0X_SEQUENCESTEP_PRE_RANGE ((VL53L0X_VcseIPeriod) 3)
/*!<Pre-Range check Identifier. */
#define VL53L0X_SEQUENCESTEP_FINAL_RANGE ((VL53L0X_VcseIPeriod) 4)
/*!<Final Range Check Identifier. */

#define VL53L0X_SEQUENCESTEP_NUMBER_OF_CHECKS 5
/*!<Number of Sequence Step Managed by the API. */

/** @} VL53L0X_define_SequenceStepId_group */

/* MACRO Definitions */

/** @defgroup VL53L0X_define_GeneralMacro_group General Macro Defines
*
* General Macro Defines
*
* @{

```

```
*/
```

```
/* Defines */
```

```
#define VL53L0X_SETPARAMETERFIELD(Dev, field, value) \  
    PALDevDataSet(Dev, CurrentParameters.field, value)
```

```
#define VL53L0X_GETPARAMETERFIELD(Dev, field, variable) \  
    variable = PALDevDataGet(Dev, CurrentParameters).field
```

```
#define VL53L0X_SETARRAYPARAMETERFIELD(Dev, field, index, value) \  
    PALDevDataSet(Dev, CurrentParameters.field[index], value)
```

```
#define VL53L0X_GETARRAYPARAMETERFIELD(Dev, field, index, variable) \  
    variable = PALDevDataGet(Dev, CurrentParameters).field[index]
```

```
#define VL53L0X_SETDEVICESPECIFICPARAMETER(Dev, field, value) \  
    PALDevDataSet(Dev, DeviceSpecificParameters.field, value)
```

```
#define VL53L0X_GETDEVICESPECIFICPARAMETER(Dev, field) \  
    PALDevDataGet(Dev, DeviceSpecificParameters).field
```

```
#define VL53L0X_FIXPOINT1616TOFIXPOINT97(Value) \  
    (uint16_t)((Value>>9)&0xFFFF)
```

```
#define VL53L0X_FIXPOINT97TOFIXPOINT1616(Value) \  
    (FixPoint1616_t)(Value<<9)
```

```

#define VL53L0X_FIXPOINT1616TOFIXPOINT88(Value) \
    (uint16_t)((Value>>8)&0xFFFF)
#define VL53L0X_FIXPOINT88TOFIXPOINT1616(Value) \
    (FixPoint1616_t)(Value<<8)

#define VL53L0X_FIXPOINT1616TOFIXPOINT412(Value) \
    (uint16_t)((Value>>4)&0xFFFF)
#define VL53L0X_FIXPOINT412TOFIXPOINT1616(Value) \
    (FixPoint1616_t)(Value<<4)

#define VL53L0X_FIXPOINT1616TOFIXPOINT313(Value) \
    (uint16_t)((Value>>3)&0xFFFF)
#define VL53L0X_FIXPOINT313TOFIXPOINT1616(Value) \
    (FixPoint1616_t)(Value<<3)

#define VL53L0X_FIXPOINT1616TOFIXPOINT08(Value) \
    (uint8_t)((Value>>8)&0x00FF)
#define VL53L0X_FIXPOINT08TOFIXPOINT1616(Value) \
    (FixPoint1616_t)(Value<<8)

#define VL53L0X_FIXPOINT1616TOFIXPOINT53(Value) \
    (uint8_t)((Value>>13)&0x00FF)
#define VL53L0X_FIXPOINT53TOFIXPOINT1616(Value) \
    (FixPoint1616_t)(Value<<13)

#define VL53L0X_FIXPOINT1616TOFIXPOINT102(Value) \
    (uint16_t)((Value>>14)&0x0FFF)
#define VL53L0X_FIXPOINT102TOFIXPOINT1616(Value) \
    (FixPoint1616_t)(Value<<12)

```

```
#define VL53L0X_MAKEUINT16(lsb, msb) (uint16_t)((((uint16_t)msb)<<8) + \
    (uint16_t)lsb)
```

```
/** @} VL53L0X_define_GeneralMacro_group */
```

```
/** @} VL53L0X_globaldefine_group */
```

```
#ifdef __cplusplus
```

```
}
```

```
#endif
```

```
#endif /* _VL53L0X_DEF_H_ */
```

1.1.32 3l0x_api.h

Below:

VI5 /*****

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright

notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#ifndef _VL53L0X_API_H_
```

```
#define _VL53L0X_API_H_
```

```
#include "vl53l0x_api_strings.h"
```

```
#include "vl53l0x_def.h"
```

```
#include "vl53l0x_platform.h"
```

```
#ifdef __cplusplus
```

```

extern "C"

{

#endif


#ifdef _MSC_VER

# ifdef VL53L0X_API_EXPORTS

#   define VL53L0X_API __declspec(dllexport)

# else

#   define VL53L0X_API

# endif

#else

# define VL53L0X_API

#endif


/** @defgroup VL53L0X_cut11_group VL53L0X cut1.1 Function Definition
 * @brief VL53L0X cut1.1 Function Definition
 * @{
 */


/** @defgroup VL53L0X_general_group VL53L0X General Functions
 * @brief General functions and definitions
 * @{
 */


/**
 * @brief Return the VL53L0X PAL Implementation Version
 *
 * @note This function doesn't access to the device
 *

```

* @param pVersion Pointer to current PAL Implementation Version

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetVersion(VL53L0X_Version_t *pVersion);

/**

* @brief Return the PAL Specification Version used for the current

* implementation.

*

* @note This function doesn't access to the device

*

* @param pPalSpecVersion Pointer to current PAL Specification Version

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetPalSpecVersion(

 VL53L0X_Version_t *pPalSpecVersion);

/**

* @brief Reads the Product Revision for a for given Device

* This function can be used to distinguish cut1.0 from cut1.1.

*

* @note This function Access to the device

*

* @param Dev Device Handle

* @param pProductRevisionMajor Pointer to Product Revision Major

* for a given Device

* @param pProductRevisionMinor Pointer to Product Revision Minor

* for a given Device

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetProductRevision(VL53L0X_DEV Dev,  
    uint8_t *pProductRevisionMajor, uint8_t *pProductRevisionMinor);
```

/**

* @brief Reads the Device information for given Device

*

* @note This function Access to the device

*

* @param Dev Device Handle

* @param pVL53L0X_DeviceInfo Pointer to current device info for a given

* Device

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetDeviceInfo(VL53L0X_DEV Dev,  
    VL53L0X_DeviceInfo_t *pVL53L0X_DeviceInfo);
```

/**

* @brief Read current status of the error register for the selected device

*

* @note This function Access to the device

*

* @param Dev Device Handle

* @param pDeviceErrorStatus Pointer to current error code of the device

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetDeviceErrorStatus(VL53L0X_DEV Dev,  
    VL53L0X_DeviceError *pDeviceErrorStatus);
```

/**

* @brief Human readable Range Status string for a given RangeStatus

*

* @note This function doesn't access to the device

*

* @param RangeStatus The RangeStatus code as stored on

* @a VL53L0X_RangingMeasurementData_t

* @param pRangeStatusString The returned RangeStatus string.

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetRangeStatusString(uint8_t RangeStatus,  
    char *pRangeStatusString);
```

/**

* @brief Human readable error string for a given Error Code

*

* @note This function doesn't access to the device

*

* @param ErrorCode The error code as stored on ::VL53L0X_DeviceError

* @param pDeviceErrorString The error string corresponding to the ErrorCode

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetDeviceErrorString(  
    VL53L0X_DeviceError ErrorCode, char *pDeviceErrorString);
```

```
/**
```

```
 * @brief Human readable error string for current PAL error status
```

```
 *
```

```
 * @note This function doesn't access to the device
```

```
 *
```

```
 * @param PalErrorCode    The error code as stored on @a VL53L0X_Error
```

```
 * @param pPalErrorString  The error string corresponding to the
```

```
 * PalErrorCode
```

```
 * @return VL53L0X_ERROR_NONE Success
```

```
 * @return "Other error code" See ::VL53L0X_Error
```

```
 */
```

```
VL53L0X_API VL53L0X_Error VL53L0X_GetPalErrorString(VL53L0X_Error PalErrorCode,  
    char *pPalErrorString);
```

```
/**
```

```
 * @brief Human readable PAL State string
```

```
 *
```

```
 * @note This function doesn't access to the device
```

```
 *
```

```
 * @param PalStateCode    The State code as stored on @a VL53L0X_State
```

```
 * @param pPalStateString  The State string corresponding to the
```

```
 * PalStateCode
```

```
 * @return VL53L0X_ERROR_NONE Success
```

```
 * @return "Other error code" See ::VL53L0X_Error
```

```
 */
```

```
VL53L0X_API VL53L0X_Error VL53L0X_GetPalStateString(VL53L0X_State PalStateCode,
```

```
char *pPalStateString);
```

```
/**
```

```
* @brief Reads the internal state of the PAL for a given Device
```

```
*
```

```
* @note This function doesn't access to the device
```

```
*
```

```
* @param Dev          Device Handle
```

```
* @param pPalState     Pointer to current state of the PAL for a
```

```
* given Device
```

```
* @return VL53L0X_ERROR_NONE    Success
```

```
* @return "Other error code"    See ::VL53L0X_Error
```

```
*/
```

```
VL53L0X_API VL53L0X_Error VL53L0X_GetPalState(VL53L0X_DEV Dev,
```

```
        VL53L0X_State *pPalState);
```

```
/**
```

```
* @brief Set the power mode for a given Device
```

```
* The power mode can be Standby or Idle. Different level of both Standby and
```

```
* Idle can exists.
```

```
* This function should not be used when device is in Ranging state.
```

```
*
```

```
* @note This function Access to the device
```

```
*
```

```
* @param Dev          Device Handle
```

```
* @param PowerMode     The value of the power mode to set.
```

```
* see ::VL53L0X_PowerModes
```

```
* Valid values are:
```

```
* VL53L0X_POWERMODE_STANDBY_LEVEL1,
```

```

*          VL53L0X_POWERMODE_IDLE_LEVEL1
* @return VL53L0X_ERROR_NONE          Success
* @return VL53L0X_ERROR_MODE_NOT_SUPPORTED  This error occurs when PowerMode
* is not in the supported list
* @return "Other error code"  See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_SetPowerMode(VL53L0X_DEV Dev,
          VL53L0X_PowerModes PowerMode);

```

```

/**
* @brief Get the power mode for a given Device
*
* @note This function Access to the device
*
* @param Dev          Device Handle
* @param pPowerMode    Pointer to the current value of the power
* mode. see ::VL53L0X_PowerModes
*
*          Valid values are:
*
*          VL53L0X_POWERMODE_STANDBY_LEVEL1,
*
*          VL53L0X_POWERMODE_IDLE_LEVEL1
* @return VL53L0X_ERROR_NONE  Success
* @return "Other error code"  See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetPowerMode(VL53L0X_DEV Dev,
          VL53L0X_PowerModes *pPowerMode);

```

```

/**
* Set or over-hide part to part calibration offset
* \sa VL53L0X_DataInit() VL53L0X_GetOffsetCalibrationDataMicroMeter()

```

```

*

* @note This function Access to the device
*
* @param Dev                Device Handle
* @param OffsetCalibrationDataMicroMeter  Offset (microns)
* @return VL53L0X_ERROR_NONE        Success
* @return "Other error code"        See ::VL53L0X_Error
*/

VL53L0X_API VL53L0X_Error VL53L0X_SetOffsetCalibrationDataMicroMeter(
    VL53L0X_DEV Dev, int32_t OffsetCalibrationDataMicroMeter);

/**
* @brief Get part to part calibration offset
*
* @par Function Description
* Should only be used after a successful call to @a VL53L0X_DataInit to backup
* device NVM value
*
* @note This function Access to the device
*
* @param Dev                Device Handle
* @param pOffsetCalibrationDataMicroMeter  Return part to part
* calibration offset from device (microns)
* @return VL53L0X_ERROR_NONE        Success
* @return "Other error code"        See ::VL53L0X_Error
*/

VL53L0X_API VL53L0X_Error VL53L0X_GetOffsetCalibrationDataMicroMeter(
    VL53L0X_DEV Dev, int32_t *pOffsetCalibrationDataMicroMeter);

```

```

/**
 * Set the linearity corrective gain
 *
 * @note This function Access to the device
 *
 * @param Dev          Device Handle
 * @param LinearityCorrectiveGain    Linearity corrective
 * gain in x1000
 * if value is 1000 then no modification is applied.
 * @return VL53L0X_ERROR_NONE      Success
 * @return "Other error code"      See ::VL53L0X_Error
 */
VL53L0X_API VL53L0X_Error VL53L0X_SetLinearityCorrectiveGain(VL53L0X_DEV Dev,
    int16_t LinearityCorrectiveGain);

```

```

/**
 * @brief Get the linearity corrective gain
 *
 * @par Function Description
 * Should only be used after a successful call to @a VL53L0X_DataInit to backup
 * device NVM value
 *
 * @note This function Access to the device
 *
 * @param Dev          Device Handle
 * @param pLinearityCorrectiveGain    Pointer to the linearity
 * corrective gain in x1000
 * if value is 1000 then no modification is applied.
 * @return VL53L0X_ERROR_NONE      Success

```

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetLinearityCorrectiveGain(VL53L0X_DEV Dev,
 uint16_t *pLinearityCorrectiveGain);

/**

* Set Group parameter Hold state

*

* @par Function Description

* Set or remove device internal group parameter hold

*

* @note This function is not Implemented

*

* @param Dev Device Handle

* @param GroupParamHold Group parameter Hold state to be set (on/off)

* @return VL53L0X_ERROR_NOT_IMPLEMENTED Not implemented

*/

VL53L0X_API VL53L0X_Error VL53L0X_SetGroupParamHold(VL53L0X_DEV Dev,
 uint8_t GroupParamHold);

/**

* @brief Get the maximal distance for actual setup

* @par Function Description

* Device must be initialized through @a VL53L0X_SetParameters() prior calling
* this function.

*

* Any range value more than the value returned is to be considered as

* "no target detected" or

* "no target in detectable range"\n

* @warning The maximal distance depends on the setup

*

* @note This function is not Implemented

*

* @param Dev Device Handle

* @param pUpperLimitMilliMeter The maximal range limit for actual setup

* (in millimeter)

* @return VL53L0X_ERROR_NOT_IMPLEMENTED Not implemented

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetUpperLimitMilliMeter(VL53L0X_DEV Dev,  
    uint16_t *pUpperLimitMilliMeter);
```

/**

* @brief Get the Total Signal Rate

* @par Function Description

* This function will return the Total Signal Rate after a good ranging is done.

*

* @note This function access to Device

*

* @param Dev Device Handle

* @param pTotalSignalRate Total Signal Rate value in Mega count per second

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_Error VL53L0X_GetTotalSignalRate(VL53L0X_DEV Dev,  
    FixPoint1616_t *pTotalSignalRate);
```

/** @} VL53L0X_general_group */

```
/** @defgroup VL53L0X_init_group VL53L0X Init Functions
```

```
* @brief VL53L0X Init Functions
```

```
* @{
```

```
*/
```

```
/**
```

```
* @brief Set new device address
```

```
*
```

```
* After completion the device will answer to the new address programmed.
```

```
* This function should be called when several devices are used in parallel
```

```
* before start programming the sensor.
```

```
* When a single device is used, there is no need to call this function.
```

```
*
```

```
* @note This function Access to the device
```

```
*
```

```
* @param Dev Device Handle
```

```
* @param DeviceAddress The new Device address
```

```
* @return VL53L0X_ERROR_NONE Success
```

```
* @return "Other error code" See ::VL53L0X_Error
```

```
*/
```

```
VL53L0X_API VL53L0X_Error VL53L0X_SetDeviceAddress(VL53L0X_DEV Dev,  
uint8_t DeviceAddress);
```

```
/**
```

```
*
```

```
* @brief One time device initialization
```

```
*
```

```
* To be called once and only once after device is brought out of reset
```

- * (Chip enable) and booted see @a VL53L0X_WaitDeviceBooted()
- *
- * @par Function Description
- * When not used after a fresh device "power up" or reset, it may return
- * @a #VL53L0X_ERROR_CALIBRATION_WARNING meaning wrong calibration data
- * may have been fetched from device that can result in ranging offset error\n
- * If application cannot execute device reset or need to run VL53L0X_DataInit
- * multiple time then it must ensure proper offset calibration saving and
- * restore on its own by using @a VL53L0X_GetOffsetCalibrationData() on first
- * power up and then @a VL53L0X_SetOffsetCalibrationData() in all subsequent init
- * This function will change the VL53L0X_State from VL53L0X_STATE_POWERDOWN to
- * VL53L0X_STATE_WAIT_STATICINIT.
- *
- * @note This function Access to the device
- *
- * @param Dev Device Handle
- * @return VL53L0X_ERROR_NONE Success
- * @return "Other error code" See ::VL53L0X_Error
- */

VL53L0X_API VL53L0X_Error VL53L0X_DataInit(VL53L0X_DEV Dev);

/**

- * @brief Set the tuning settings pointer
- *
- * This function is used to specify the Tuning settings buffer to be used
- * for a given device. The buffer contains all the necessary data to permit
- * the API to write tuning settings.
- * This function permit to force the usage of either external or internal
- * tuning settings.

```

*

* @note This function Access to the device

*

* @param Dev          Device Handle

* @param pTuningSettingBuffer    Pointer to tuning settings buffer.

* @param UseInternalTuningSettings    Use internal tuning settings value.

* @return VL53L0X_ERROR_NONE    Success

* @return "Other error code"    See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_SetTuningSettingBuffer(VL53L0X_DEV Dev,
    uint8_t *pTuningSettingBuffer, uint8_t UseInternalTuningSettings);

/**

* @brief Get the tuning settings pointer and the internal external switch

* value.

*

* This function is used to get the Tuning settings buffer pointer and the

* value.

* of the switch to select either external or internal tuning settings.

*

* @note This function Access to the device

*

* @param Dev          Device Handle

* @param ppTuningSettingBuffer    Pointer to tuning settings buffer.

* @param pUseInternalTuningSettings Pointer to store Use internal tuning

* settings value.

* @return VL53L0X_ERROR_NONE    Success

* @return "Other error code"    See ::VL53L0X_Error

*/

```

```
VL53L0X_API VL53L0X_Error VL53L0X_GetTuningSettingBuffer(VL53L0X_DEV Dev,  
    uint8_t **ppTuningSettingBuffer, uint8_t *pUseInternalTuningSettings);
```

```
/**
```

```
* @brief Do basic device init (and eventually patch loading)  
* This function will change the VL53L0X_State from  
* VL53L0X_STATE_WAIT_STATICINIT to VL53L0X_STATE_IDLE.  
* In this stage all default setting will be applied.
```

```
*
```

```
* @note This function Access to the device
```

```
*
```

```
* @param Dev      Device Handle
```

```
* @return VL53L0X_ERROR_NONE    Success
```

```
* @return "Other error code"    See ::VL53L0X_Error
```

```
*/
```

```
VL53L0X_API VL53L0X_Error VL53L0X_StaticInit(VL53L0X_DEV Dev);
```

```
/**
```

```
* @brief Wait for device booted after chip enable (hardware standby)  
* This function can be run only when VL53L0X_State is VL53L0X_STATE_POWERDOWN.
```

```
*
```

```
* @note This function is not Implemented
```

```
*
```

```
* @param Dev      Device Handle
```

```
* @return VL53L0X_ERROR_NOT_IMPLEMENTED Not implemented
```

```
*
```

```
*/
```

```
VL53L0X_API VL53L0X_Error VL53L0X_WaitDeviceBooted(VL53L0X_DEV Dev);
```

```

/**
 * @brief Do an hard reset or soft reset (depending on implementation) of the
 * device \nAfter call of this function, device must be in same state as right
 * after a power-up sequence.This function will change the VL53L0X_State to
 * VL53L0X_STATE_POWERDOWN.
 *
 * @note This function Access to the device
 *
 * @param Dev          Device Handle
 * @return VL53L0X_ERROR_NONE    Success
 * @return "Other error code"    See ::VL53L0X_Error
 */

```

```

VL53L0X_API VL53L0X_Error VL53L0X_ResetDevice(VL53L0X_DEV Dev);

```

```

/** @} VL53L0X_init_group */

```

```

/** @defgroup VL53L0X_parameters_group VL53L0X Parameters Functions
 * @brief Functions used to prepare and setup the device
 * @{
 */

```

```

/**
 * @brief Prepare device for operation
 * @par Function Description
 * Update device with provided parameters
 * @li Then start ranging operation.
 *
 * @note This function Access to the device
 */

```

```

* @param Dev          Device Handle
* @param pDeviceParameters  Pointer to store current device parameters.
* @return VL53L0X_ERROR_NONE  Success
* @return "Other error code"  See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_SetDeviceParameters(VL53L0X_DEV Dev,
    const VL53L0X_DeviceParameters_t *pDeviceParameters);

```

```

/**

```

```

* @brief Retrieve current device parameters
* @par Function Description
* Get actual parameters of the device
* @li Then start ranging operation.

```

```

*

```

```

* @note This function Access to the device

```

```

*

```

```

* @param Dev          Device Handle
* @param pDeviceParameters  Pointer to store current device parameters.
* @return VL53L0X_ERROR_NONE  Success
* @return "Other error code"  See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetDeviceParameters(VL53L0X_DEV Dev,
    VL53L0X_DeviceParameters_t *pDeviceParameters);

```

```

/**

```

```

* @brief Set a new device mode
* @par Function Description
* Set device to a new mode (ranging, histogram ...)

```

```

*

```

```

* @note This function doesn't Access to the device
*
* @param Dev          Device Handle
* @param DeviceMode    New device mode to apply
*
*          Valid values are:
*
*          VL53L0X_DEVICEMODE_SINGLE_RANGING
*
*          VL53L0X_DEVICEMODE_CONTINUOUS_RANGING
*
*          VL53L0X_DEVICEMODE_CONTINUOUS_TIMED_RANGING
*
*          VL53L0X_DEVICEMODE_SINGLE_HISTOGRAM
*
*          VL53L0X_HISTOGRAMMODE_REFERENCE_ONLY
*
*          VL53L0X_HISTOGRAMMODE_RETURN_ONLY
*
*          VL53L0X_HISTOGRAMMODE_BOTH
*
*
*
* @return VL53L0X_ERROR_NONE          Success
* @return VL53L0X_ERROR_MODE_NOT_SUPPORTED This error occurs when DeviceMode is
*
*          not in the supported list
*
*/
VL53L0X_API VL53L0X_Error VL53L0X_SetDeviceMode(VL53L0X_DEV Dev,
          VL53L0X_DeviceModes DeviceMode);

/**
* @brief Get current new device mode
* @par Function Description
* Get actual mode of the device(ranging, histogram ...)
*
*
* @note This function doesn't Access to the device
*
*
* @param Dev          Device Handle

```

```

* @param pDeviceMode      Pointer to current apply mode value
*
*      Valid values are:
*
*      VL53L0X_DEVICEMODE_SINGLE_RANGING
*
*      VL53L0X_DEVICEMODE_CONTINUOUS_RANGING
*
*      VL53L0X_DEVICEMODE_CONTINUOUS_TIMED_RANGING
*
*      VL53L0X_DEVICEMODE_SINGLE_HISTOGRAM
*
*      VL53L0X_HISTOGRAMMODE_REFERENCE_ONLY
*
*      VL53L0X_HISTOGRAMMODE_RETURN_ONLY
*
*      VL53L0X_HISTOGRAMMODE_BOTH
*
*
* @return VL53L0X_ERROR_NONE      Success
* @return VL53L0X_ERROR_MODE_NOT_SUPPORTED  This error occurs when
* DeviceMode is not in the supported list
* /
VL53L0X_API VL53L0X_Error VL53L0X_GetDeviceMode(VL53L0X_DEV Dev,
          VL53L0X_DeviceModes *pDeviceMode);

/**
* @brief Sets the resolution of range measurements.
*
* @par Function Description
*
* Set resolution of range measurements to either 0.25mm if
* fraction enabled or 1mm if not enabled.
*
*
* @note This function Accesses the device
*
*
* @param Dev      Device Handle
* @param Enable    Enable high resolution
*
*
* @return VL53L0X_ERROR_NONE      Success

```

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_SetRangeFractionEnable(VL53L0X_DEV Dev,  
    uint8_t Enable);
```

/**

* @brief Gets the fraction enable parameter indicating the resolution of

* range measurements.

*

* @par Function Description

* Gets the fraction enable state, which translates to the resolution of

* range measurements as follows :Enabled:=0.25mm resolution,

* Not Enabled:=1mm resolution.

*

* @note This function Accesses the device

*

* @param Dev Device Handle

* @param pEnable Output Parameter reporting the fraction enable state.

*

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetFractionEnable(VL53L0X_DEV Dev,  
    uint8_t *pEnable);
```

/**

* @brief Set a new Histogram mode

* @par Function Description

* Set device to a new Histogram mode

```

*

* @note This function doesn't Access to the device

*

* @param Dev          Device Handle

* @param HistogramMode    New device mode to apply

*          Valid values are:

*          VL53L0X_HISTOGRAMMODE_DISABLED

*          VL53L0X_DEVICEMODE_SINGLE_HISTOGRAM

*          VL53L0X_HISTOGRAMMODE_REFERENCE_ONLY

*          VL53L0X_HISTOGRAMMODE_RETURN_ONLY

*          VL53L0X_HISTOGRAMMODE_BOTH

*

* @return VL53L0X_ERROR_NONE          Success

* @return VL53L0X_ERROR_MODE_NOT_SUPPORTED    This error occurs when

* HistogramMode is not in the supported list

* @return "Other error code"    See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_SetHistogramMode(VL53L0X_DEV Dev,

          VL53L0X_HistogramModes HistogramMode);

/**

* @brief Get current new device mode

* @par Function Description

* Get current Histogram mode of a Device

*

* @note This function doesn't Access to the device

*

* @param Dev          Device Handle

* @param pHistogramMode    Pointer to current Histogram Mode value

```

```

*          Valid values are:
*
*          VL53L0X_HISTOGRAMMODE_DISABLED
*
*          VL53L0X_DEVICEMODE_SINGLE_HISTOGRAM
*
*          VL53L0X_HISTOGRAMMODE_REFERENCE_ONLY
*
*          VL53L0X_HISTOGRAMMODE_RETURN_ONLY
*
*          VL53L0X_HISTOGRAMMODE_BOTH

```

```

* @return VL53L0X_ERROR_NONE    Success

```

```

* @return "Other error code"    See ::VL53L0X_Error

```

```

*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetHistogramMode(VL53L0X_DEV Dev,
    VL53L0X_HistogramModes *pHistogramMode);

```

```

/**

```

```

* @brief Set Ranging Timing Budget in microseconds

```

```

*

```

```

* @par Function Description

```

```

* Defines the maximum time allowed by the user to the device to run a

```

```

* full ranging sequence for the current mode (ranging, histogram, ASL ...)

```

```

*

```

```

* @note This function Access to the device

```

```

*

```

```

* @param Dev          Device Handle

```

```

* @param MeasurementTimingBudgetMicroSeconds Max measurement time in

```

```

* microseconds.

```

```

*          Valid values are:

```

```

*          >= 17000 microseconds when wraparound enabled

```

```

*          >= 12000 microseconds when wraparound disabled

```

```

* @return VL53L0X_ERROR_NONE    Success

```

```

* @return VL53L0X_ERROR_INVALID_PARAMS This error is returned if

```

MeasurementTimingBudgetMicroSeconds out of range

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_SetMeasurementTimingBudgetMicroSeconds(
 VL53L0X_DEV Dev, uint32_t MeasurementTimingBudgetMicroSeconds);

/**

* @brief Get Ranging Timing Budget in microseconds

*

* @par Function Description

* Returns the programmed the maximum time allowed by the user to the

* device to run a full ranging sequence for the current mode

* (ranging, histogram, ASL ...)

*

* @note This function Access to the device

*

* @param Dev Device Handle

* @param pMeasurementTimingBudgetMicroSeconds Max measurement time in
* microseconds.

* Valid values are:

* >= 17000 microsecs when wraparound enabled

* >= 12000 microsecs when wraparound disabled

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetMeasurementTimingBudgetMicroSeconds(
 VL53L0X_DEV Dev, uint32_t *pMeasurementTimingBudgetMicroSeconds);

/**

```

* @brief Gets the VCSEL pulse period.
*
* @par Function Description
* This function retrieves the VCSEL pulse period for the given period type.
*
* @note This function Accesses the device
*
* @param Dev          Device Handle
* @param VcselPeriodType VCSEL period identifier (pre-range|final).
* @param pVCSELPulsePeriod Pointer to VCSEL period value.
* @return VL53L0X_ERROR_NONE    Success
* @return VL53L0X_ERROR_INVALID_PARAMS Error VcselPeriodType parameter not
* supported.
* @return "Other error code"     See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetVcselPulsePeriod(VL53L0X_DEV Dev,
    VL53L0X_VcselPeriod VcselPeriodType, uint8_t *pVCSELPulsePeriod);

```

```

/**
* @brief Sets the VCSEL pulse period.
*
* @par Function Description
* This function retrieves the VCSEL pulse period for the given period type.
*
* @note This function Accesses the device
*
* @param Dev          Device Handle
* @param VcselPeriodType VCSEL period identifier (pre-range|final).
* @param VCSELPulsePeriod VCSEL period value

```

```

* @return VL53L0X_ERROR_NONE      Success
* @return VL53L0X_ERROR_INVALID_PARAMS Error VcselPeriodType parameter not
*
* supported.
* @return "Other error code"      See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_SetVcselPulsePeriod(VL53L0X_DEV Dev,
    VL53L0X_VcselPeriod VcselPeriodType, uint8_t VCSELPulsePeriod);

```

```

/**

```

```

* @brief Sets the (on/off) state of a requested sequence step.

```

```

*

```

```

* @par Function Description

```

```

* This function enables/disables a requested sequence step.

```

```

*

```

```

* @note This function Accesses the device

```

```

*

```

```

* @param Dev      Device Handle

```

```

* @param SequenceStepId      Sequence step identifier.

```

```

* @param SequenceStepEnabled      Demanded state {0=Off,1=On}

```

```

*
* is enabled.

```

```

* @return VL53L0X_ERROR_NONE      Success

```

```

* @return VL53L0X_ERROR_INVALID_PARAMS Error SequenceStepId parameter not

```

```

*
* supported.

```

```

* @return "Other error code"      See ::VL53L0X_Error

```

```

*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_SetSequenceStepEnable(VL53L0X_DEV Dev,
    VL53L0X_SequenceStepId SequenceStepId, uint8_t SequenceStepEnabled);

```

```

/**

```

```

* @brief Gets the (on/off) state of a requested sequence step.
*
* @par Function Description
* This function retrieves the state of a requested sequence step, i.e. on/off.
*
* @note This function Accesses the device
*
* @param Dev          Device Handle
* @param SequenceStepId  Sequence step identifier.
* @param pSequenceStepEnabled  Out parameter reporting if the sequence step
*                               is enabled {0=Off,1=On}.
* @return VL53L0X_ERROR_NONE      Success
* @return VL53L0X_ERROR_INVALID_PARAMS  Error SequenceStepId parameter not
*                               supported.
* @return "Other error code"      See ::VL53L0X_Error
*/
VL53L0X_API VL53L0X_Error VL53L0X_GetSequenceStepEnable(VL53L0X_DEV Dev,
    VL53L0X_SequenceStepId SequenceStepId, uint8_t *pSequenceStepEnabled);

/**
* @brief Gets the (on/off) state of all sequence steps.
*
* @par Function Description
* This function retrieves the state of all sequence step in the scheduler.
*
* @note This function Accesses the device
*
* @param Dev          Device Handle
* @param pSchedulerSequenceSteps  Pointer to struct containing result.

```

```

* @return VL53L0X_ERROR_NONE      Success
* @return "Other error code"      See ::VL53L0X_Error
*/

VL53L0X_API VL53L0X_Error VL53L0X_GetSequenceStepEnables(VL53L0X_DEV Dev,
    VL53L0X_SchedulerSequenceSteps_t *pSchedulerSequenceSteps);

/**
* @brief Sets the timeout of a requested sequence step.
*
* @par Function Description
* This function sets the timeout of a requested sequence step.
*
* @note This function Accesses the device
*
* @param Dev      Device Handle
* @param SequenceStepId      Sequence step identifier.
* @param TimeOutMillisecs      Demanded timeout
* @return VL53L0X_ERROR_NONE      Success
* @return VL53L0X_ERROR_INVALID_PARAMS      Error SequenceStepId parameter not
* supported.
* @return "Other error code"      See ::VL53L0X_Error
*/

VL53L0X_API VL53L0X_Error VL53L0X_SetSequenceStepTimeout(VL53L0X_DEV Dev,
    VL53L0X_SequenceStepId SequenceStepId, FixPoint1616_t TimeOutMillisecs);

/**
* @brief Gets the timeout of a requested sequence step.
*
* @par Function Description

```

```

* This function retrieves the timeout of a requested sequence step.
*
* @note This function Accesses the device
*
* @param Dev          Device Handle
* @param SequenceStepId  Sequence step identifier.
* @param pTimeOutMilliSecs  Timeout value.
* @return VL53L0X_ERROR_NONE      Success
* @return VL53L0X_ERROR_INVALID_PARAMS  Error SequenceStepId parameter not
*                                     supported.
* @return "Other error code"      See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetSequenceStepTimeout(VL53L0X_DEV Dev,
    VL53L0X_SequenceStepId SequenceStepId,
    FixPoint1616_t *pTimeOutMilliSecs);

```

```

/**

```

```

* @brief Gets number of sequence steps managed by the API.
*
* @par Function Description
* This function retrieves the number of sequence steps currently managed
* by the API
*
* @note This function Accesses the device
*
* @param Dev          Device Handle
* @param pNumberOfSequenceSteps  Out parameter reporting the number of
*                               sequence steps.
* @return VL53L0X_ERROR_NONE      Success

```

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetNumberOfSequenceSteps(VL53L0X_DEV Dev,
 uint8_t *pNumberOfSequenceSteps);

/**

* @brief Gets the name of a given sequence step.

*

* @par Function Description

* This function retrieves the name of sequence steps corresponding to

* SequenceStepId.

*

* @note This function doesn't Accesses the device

*

* @param SequenceStepId Sequence step identifier.

* @param pSequenceStepsString Pointer to Info string

*

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetSequenceStepsInfo(
 VL53L0X_SequenceStepId SequenceStepId, char *pSequenceStepsString);

/**

* Program continuous mode Inter-Measurement period in milliseconds

*

* @par Function Description

* When trying to set too short time return INVALID_PARAMS minimal value

*

```

* @note This function Access to the device
*
* @param Dev Device Handle
* @param InterMeasurementPeriodMilliseconds Inter-Measurement Period in ms.
* @return VL53L0X_ERROR_NONE Success
* @return "Other error code" See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_SetInterMeasurementPeriodMilliseconds(
    VL53L0X_DEV Dev, uint32_t InterMeasurementPeriodMilliseconds);

```

```

/**

```

```

* Get continuous mode Inter-Measurement period in milliseconds
*
* @par Function Description
* When trying to set too short time return INVALID_PARAMS minimal value
*
* @note This function Access to the device
*

```

```

* @param Dev Device Handle
* @param pInterMeasurementPeriodMilliseconds Pointer to programmed
* Inter-Measurement Period in milliseconds.
* @return VL53L0X_ERROR_NONE Success
* @return "Other error code" See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetInterMeasurementPeriodMilliseconds(
    VL53L0X_DEV Dev, uint32_t *pInterMeasurementPeriodMilliseconds);

```

```

/**

```

```

* @brief Enable/Disable Cross talk compensation feature

```

```

*

* @note This function is not Implemented.

* Enable/Disable Cross Talk by set to zero the Cross Talk value

* by using @a VL53L0X_SetXTalkCompensationRateMegaCps().

*

* @param Dev          Device Handle

* @param XTalkCompensationEnable Cross talk compensation

* to be set 0=disabled else = enabled

* @return VL53L0X_ERROR_NOT_IMPLEMENTED Not implemented

*/

VL53L0X_API VL53L0X_Error VL53L0X_SetXTalkCompensationEnable(VL53L0X_DEV Dev,
    uint8_t XTalkCompensationEnable);

/**

* @brief Get Cross talk compensation rate

*

* @note This function is not Implemented.

* Enable/Disable Cross Talk by set to zero the Cross Talk value by

* using @a VL53L0X_SetXTalkCompensationRateMegaCps().

*

* @param Dev          Device Handle

* @param pXTalkCompensationEnable Pointer to the Cross talk compensation

* state 0=disabled or 1 = enabled

* @return VL53L0X_ERROR_NOT_IMPLEMENTED Not implemented

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetXTalkCompensationEnable(VL53L0X_DEV Dev,
    uint8_t *pXTalkCompensationEnable);

/**

```

```

* @brief Set Cross talk compensation rate
*
* @par Function Description
* Set Cross talk compensation rate.
*
* @note This function Access to the device
*
* @param Dev          Device Handle
* @param XTalkCompensationRateMegaCps Compensation rate in
* Mega counts per second (16.16 fix point) see datasheet for details
* @return VL53L0X_ERROR_NONE      Success
* @return "Other error code"      See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_SetXTalkCompensationRateMegaCps(VL53L0X_DEV Dev,
                          FixPoint1616_t XTalkCompensationRateMegaCps);

```

```

/**
* @brief Get Cross talk compensation rate
*
* @par Function Description
* Get Cross talk compensation rate.
*
* @note This function Access to the device
*
* @param Dev          Device Handle
* @param pXTalkCompensationRateMegaCps Pointer to Compensation rate
in Mega counts per second (16.16 fix point) see datasheet for details
* @return VL53L0X_ERROR_NONE      Success
* @return "Other error code"      See ::VL53L0X_Error

```

```

*/
VL53L0X_API VL53L0X_Error VL53L0X_GetXTalkCompensationRateMegaCps(VL53L0X_DEV Dev,
    FixPoint1616_t *pXTalkCompensationRateMegaCps);

```

```

/**
 * @brief Set Reference Calibration Parameters
 *
 * @par Function Description
 * Set Reference Calibration Parameters.
 *
 * @note This function Access to the device
 *
 * @param Dev          Device Handle
 * @param VhvSettings   Parameter for VHV
 * @param PhaseCal      Parameter for PhaseCal
 * @return VL53L0X_ERROR_NONE    Success
 * @return "Other error code"    See ::VL53L0X_Error
 */

```

```

VL53L0X_API VL53L0X_Error VL53L0X_SetRefCalibration(VL53L0X_DEV Dev,
    uint8_t VhvSettings, uint8_t PhaseCal);

```

```

/**
 * @brief Get Reference Calibration Parameters
 *
 * @par Function Description
 * Get Reference Calibration Parameters.
 *
 * @note This function Access to the device
 *

```

```

* @param Dev          Device Handle
* @param pVhvSettings Pointer to VHV parameter
* @param pPhaseCal    Pointer to PhaseCal Parameter
* @return VL53L0X_ERROR_NONE    Success
* @return "Other error code"    See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetRefCalibration(VL53L0X_DEV Dev,
    uint8_t *pVhvSettings, uint8_t *pPhaseCal);

```

```

/**

```

```

* @brief Get the number of the check limit managed by a given Device

```

```

*

```

```

* @par Function Description

```

```

* This function give the number of the check limit managed by the Device

```

```

*

```

```

* @note This function doesn't Access to the device

```

```

*

```

```

* @param pNumberOfLimitCheck    Pointer to the number of check limit.

```

```

* @return VL53L0X_ERROR_NONE    Success

```

```

* @return "Other error code"    See ::VL53L0X_Error

```

```

*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetNumberOfLimitCheck(
    uint16_t *pNumberOfLimitCheck);

```

```

/**

```

```

* @brief Return a description string for a given limit check number

```

```

*

```

```

* @par Function Description

```

```

* This function returns a description string for a given limit check number.

```

* The limit check is identified with the LimitCheckId.

*

* @note This function doesn't Access to the device

*

* @param Dev Device Handle

* @param LimitCheckId Limit Check ID

(0<= LimitCheckId < VL53L0X_GetNumberOfLimitCheck()).

* @param pLimitCheckString Pointer to the
description string of the given check limit.

* @return VL53L0X_ERROR_NONE Success

* @return VL53L0X_ERROR_INVALID_PARAMS This error is
returned when LimitCheckId value is out of range.

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetLimitCheckInfo(VL53L0X_DEV Dev,  
    uint16_t LimitCheckId, char *pLimitCheckString);
```

/**

* @brief Return a the Status of the specified check limit

*

* @par Function Description

* This function returns the Status of the specified check limit.

* The value indicate if the check is fail or not.

* The limit check is identified with the LimitCheckId.

*

* @note This function doesn't Access to the device

*

* @param Dev Device Handle

* @param LimitCheckId Limit Check ID

(0<= LimitCheckId < VL53L0X_GetNumberOfLimitCheck()).

* @param pLimitCheckStatus Pointer to the

Limit Check Status of the given check limit.

* LimitCheckStatus :

* 0 the check is not fail

* 1 the check if fail or not enabled

*

* @return VL53L0X_ERROR_NONE Success

* @return VL53L0X_ERROR_INVALID_PARAMS This error is
returned when LimitCheckId value is out of range.

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetLimitCheckStatus(VL53L0X_DEV Dev,
uint16_t LimitCheckId, uint8_t *pLimitCheckStatus);

/**

* @brief Enable/Disable a specific limit check

*

* @par Function Description

* This function Enable/Disable a specific limit check.

* The limit check is identified with the LimitCheckId.

*

* @note This function doesn't Access to the device

*

* @param Dev Device Handle

* @param LimitCheckId Limit Check ID

* (0<= LimitCheckId < VL53L0X_GetNumberOfLimitCheck()).

* @param LimitCheckEnable if 1 the check limit

* corresponding to LimitCheckId is Enabled

```

*             if 0 the check limit

* corresponding to LimitCheckId is disabled

* @return VL53L0X_ERROR_NONE      Success

* @return VL53L0X_ERROR_INVALID_PARAMS This error is returned

* when LimitCheckId value is out of range.

* @return "Other error code"      See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_SetLimitCheckEnable(VL53L0X_DEV Dev,
               uint16_t LimitCheckId, uint8_t LimitCheckEnable);

/**

* @brief Get specific limit check enable state

*

* @par Function Description

* This function get the enable state of a specific limit check.

* The limit check is identified with the LimitCheckId.

*

* @note This function Access to the device

*

* @param Dev      Device Handle

* @param LimitCheckId      Limit Check ID

* (0<= LimitCheckId < VL53L0X_GetNumberOfLimitCheck() ).

* @param pLimitCheckEnable      Pointer to the check limit enable

* value.

* if 1 the check limit

* corresponding to LimitCheckId is Enabled

* if 0 the check limit

* corresponding to LimitCheckId is disabled

* @return VL53L0X_ERROR_NONE      Success

```

```

* @return VL53L0X_ERROR_INVALID_PARAMS This error is returned
* when LimitCheckId value is out of range.
* @return "Other error code" See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetLimitCheckEnable(VL53L0X_DEV Dev,
    uint16_t LimitCheckId, uint8_t *pLimitCheckEnable);

```

```

/**

```

```

* @brief Set a specific limit check value

```

```

*

```

```

* @par Function Description

```

```

* This function set a specific limit check value.

```

```

* The limit check is identified with the LimitCheckId.

```

```

*

```

```

* @note This function Access to the device

```

```

*

```

```

* @param Dev Device Handle

```

```

* @param LimitCheckId Limit Check ID

```

```

* (0<= LimitCheckId < VL53L0X_GetNumberOfLimitCheck() ).

```

```

* @param LimitCheckValue Limit check Value for a given

```

```

* LimitCheckId

```

```

* @return VL53L0X_ERROR_NONE Success

```

```

* @return VL53L0X_ERROR_INVALID_PARAMS This error is returned when either

```

```

* LimitCheckId or LimitCheckValue value is out of range.

```

```

* @return "Other error code" See ::VL53L0X_Error

```

```

*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_SetLimitCheckValue(VL53L0X_DEV Dev,
    uint16_t LimitCheckId, FixPoint1616_t LimitCheckValue);

```

```

/**
 * @brief Get a specific limit check value
 *
 * @par Function Description
 * This function get a specific limit check value from device then it updates
 * internal values and check enables.
 * The limit check is identified with the LimitCheckId.
 *
 * @note This function Access to the device
 *
 * @param Dev          Device Handle
 * @param LimitCheckId  Limit Check ID
 * (0<= LimitCheckId < VL53L0X_GetNumberOfLimitCheck() ).
 * @param pLimitCheckValue  Pointer to Limit
 * check Value for a given LimitCheckId.
 * @return VL53L0X_ERROR_NONE      Success
 * @return VL53L0X_ERROR_INVALID_PARAMS  This error is returned
 * when LimitCheckId value is out of range.
 * @return "Other error code"      See ::VL53L0X_Error
 */
VL53L0X_API VL53L0X_Error VL53L0X_GetLimitCheckValue(VL53L0X_DEV Dev,
    uint16_t LimitCheckId, FixPoint1616_t *pLimitCheckValue);

```

```

/**
 * @brief Get the current value of the signal used for the limit check
 *
 * @par Function Description
 * This function get a the current value of the signal used for the limit check.
 * To obtain the latest value you should run a ranging before.

```

* The value reported is linked to the limit check identified with the

* LimitCheckId.

*

* @note This function Access to the device

*

* @param Dev Device Handle

* @param LimitCheckId Limit Check ID

* (0<= LimitCheckId < VL53L0X_GetNumberOfLimitCheck()).

* @param pLimitCheckCurrent Pointer to current Value for a

* given LimitCheckId.

* @return VL53L0X_ERROR_NONE Success

* @return VL53L0X_ERROR_INVALID_PARAMS This error is returned when

* LimitCheckId value is out of range.

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetLimitCheckCurrent(VL53L0X_DEV Dev,
    uint16_t LimitCheckId, FixPoint1616_t *pLimitCheckCurrent);
```

/**

* @brief Enable (or disable) Wrap around Check

*

* @note This function Access to the device

*

* @param Dev Device Handle

* @param WrapAroundCheckEnable Wrap around Check to be set

* 0=disabled, other = enabled

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_SetWrapAroundCheckEnable(VL53L0X_DEV Dev,  
    uint8_t WrapAroundCheckEnable);
```

```
/**
```

```
 * @brief Get setup of Wrap around Check
```

```
 *
```

```
 * @par Function Description
```

```
 * This function get the wrapAround check enable parameters
```

```
 *
```

```
 * @note This function Access to the device
```

```
 *
```

```
 * @param Dev          Device Handle
```

```
 * @param pWrapAroundCheckEnable Pointer to the Wrap around Check state
```

```
 *          0=disabled or 1 = enabled
```

```
 * @return VL53L0X_ERROR_NONE    Success
```

```
 * @return "Other error code"    See ::VL53L0X_Error
```

```
 */
```

```
VL53L0X_API VL53L0X_Error VL53L0X_GetWrapAroundCheckEnable(VL53L0X_DEV Dev,  
    uint8_t *pWrapAroundCheckEnable);
```

```
/**
```

```
 * @brief Set Dmax Calibration Parameters for a given device
```

```
 * When one of the parameter is zero, this function will get parameter
```

```
 * from NVM.
```

```
 * @note This function doesn't Access to the device
```

```
 *
```

```
 * @param Dev          Device Handle
```

```
 * @param RangeMilliMeter Calibration Distance
```

```
 * @param SignalRateRtnMegaCps Signal rate return read at CalDistance
```

```

* @return VL53L0X_ERROR_NONE    Success
* @return "Other error code"    See ::VL53L0X_Error
*/

VL53L0X_API VL53L0X_Error VL53L0X_SetDmaxCalParameters(VL53L0X_DEV Dev,
    uint16_t RangeMilliMeter, FixPoint1616_t SignalRateRtnMegaCps);

/**
* @brief Get Dmax Calibration Parameters for a given device
*
*
*
* @note This function Access to the device
*
* @param Dev          Device Handle
* @param pRangeMilliMeter    Pointer to Calibration Distance
* @param pSignalRateRtnMegaCps    Pointer to Signal rate return
* @return VL53L0X_ERROR_NONE    Success
* @return "Other error code"    See ::VL53L0X_Error
*/

VL53L0X_API VL53L0X_Error VL53L0X_GetDmaxCalParameters(VL53L0X_DEV Dev,
    uint16_t *pRangeMilliMeter, FixPoint1616_t *pSignalRateRtnMegaCps);

/** @} VL53L0X_parameters_group */

/** @defgroup VL53L0X_measurement_group VL53L0X Measurement Functions
* @brief Functions used for the measurements
* @{
*
*/

/**

```

```

* @brief Single shot measurement.
*
* @par Function Description
* Perform simple measurement sequence (Start measure, Wait measure to end,
* and returns when measurement is done).
* Once function returns, user can get valid data by calling
* VL53L0X_GetRangingMeasurement or VL53L0X_GetHistogramMeasurement
* depending on defined measurement mode
* User should Clear the interrupt in case this are enabled by using the
* function VL53L0X_ClearInterruptMask().
*
* @warning This function is a blocking function
*
* @note This function Access to the device
*
* @param Dev          Device Handle
* @return VL53L0X_ERROR_NONE    Success
* @return "Other error code"    See ::VL53L0X_Error
*/
VL53L0X_API VL53L0X_Error VL53L0X_PerformSingleMeasurement(VL53L0X_DEV Dev);

/**
* @brief Perform Reference Calibration
*
* @details Perform a reference calibration of the Device.
* This function should be run from time to time before doing
* a ranging measurement.
* This function will launch a special ranging measurement, so
* if interrupt are enable an interrupt will be done.

```

* This function will clear the interrupt generated automatically.

*

* @warning This function is a blocking function

*

* @note This function Access to the device

*

* @param Dev Device Handle

* @param pVhvSettings Pointer to vvh settings parameter.

* @param pPhaseCal Pointer to PhaseCal parameter.

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_PerformRefCalibration(VL53L0X_DEV Dev,  
    uint8_t *pVhvSettings, uint8_t *pPhaseCal);
```

/**

* @brief Perform XTalk Measurement

*

* @details Measures the current cross talk from glass in front

* of the sensor.

* This functions performs a histogram measurement and uses the results

* to measure the crosstalk. For the function to be successful, there

* must be no target in front of the sensor.

*

* @warning This function is a blocking function

*

* @warning This function is not supported when the final range

* vcsel clock period is set below 10 PCLKS.

*

```

* @note This function Access to the device
*
* @param Dev          Device Handle
* @param TimeoutMs     Histogram measurement duration.
* @param pXtalkPerSpad Output parameter containing the crosstalk
* measurement result, in MCPS/Spad. Format fixpoint 16:16.
* @param pAmbientTooHigh Output parameter which indicate that
* pXtalkPerSpad is not good if the Ambient is too high.
* @return VL53L0X_ERROR_NONE Success
* @return VL53L0X_ERROR_INVALID_PARAMS vcsel clock period not supported
* for this operation. Must not be less than 10PCLKS.
* @return "Other error code" See ::VL53L0X_Error
*/
VL53L0X_API VL53L0X_Error VL53L0X_PerformXTalkMeasurement(VL53L0X_DEV Dev,
    uint32_t TimeoutMs, FixPoint1616_t *pXtalkPerSpad,
    uint8_t *pAmbientTooHigh);

/**
* @brief Perform XTalk Calibration
*
* @details Perform a XTalk calibration of the Device.
* This function will launch a ranging measurement, if interrupts
* are enabled an interrupt will be done.
* This function will clear the interrupt generated automatically.
* This function will program a new value for the XTalk compensation
* and it will enable the cross talk before exit.
* This function will disable the VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD.
*
* @warning This function is a blocking function

```

```

*

* @note This function Access to the device

*

* @note This function change the device mode to

* VL53L0X_DEVICEMODE_SINGLE_RANGING

*

* @param Dev          Device Handle

* @param XTalkCalDistance  XTalkCalDistance value used for the XTalk

* computation.

* @param pXTalkCompensationRateMegaCps Pointer to new

* XTalkCompensation value.

* @return VL53L0X_ERROR_NONE  Success

* @return "Other error code"  See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_PerformXTalkCalibration(VL53L0X_DEV Dev,

    FixPoint1616_t XTalkCalDistance,

    FixPoint1616_t *pXTalkCompensationRateMegaCps);

/**

* @brief Perform Offset Calibration

*

* @details Perform a Offset calibration of the Device.

* This function will launch a ranging measurement, if interrupts are

* enabled an interrupt will be done.

* This function will clear the interrupt generated automatically.

* This function will program a new value for the Offset calibration value

* This function will disable the VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD.

*

* @warning This function is a blocking function

```

```

*

* @note This function Access to the device

*

* @note This function does not change the device mode.

*

* @param Dev          Device Handle

* @param CalDistanceMilliMeter Calibration distance value used for the

* offset compensation.

* @param pOffsetMicroMeter Pointer to new Offset value computed by the

* function.

*

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_PerformOffsetCalibration(VL53L0X_DEV Dev,

    FixPoint1616_t CalDistanceMilliMeter, int32_t *pOffsetMicroMeter);

/**

* @brief Start device measurement

*

* @details Started measurement will depend on device parameters set through

* @a VL53L0X_SetParameters()

* This is a non-blocking function.

* This function will change the VL53L0X_State from VL53L0X_STATE_IDLE to

* VL53L0X_STATE_RUNNING.

*

* @note This function Access to the device

*

```

```

* @param Dev          Device Handle
* @return VL53L0X_ERROR_NONE          Success
* @return VL53L0X_ERROR_MODE_NOT_SUPPORTED  This error occurs when
* DeviceMode programmed with @a VL53L0X_SetDeviceMode is not in the supported
* list:
*
*          Supported mode are:
*
*          VL53L0X_DEVICEMODE_SINGLE_RANGING,
*
*          VL53L0X_DEVICEMODE_CONTINUOUS_RANGING,
*
*          VL53L0X_DEVICEMODE_CONTINUOUS_TIMED_RANGING
* @return VL53L0X_ERROR_TIME_OUT  Time out on start measurement
* @return "Other error code"  See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_StartMeasurement(VL53L0X_DEV Dev);

```

```

/**
* @brief Stop device measurement
*
* @details Will set the device in standby mode at end of current measurement\n
*
*      Not necessary in single mode as device shall return automatically
*
*      in standby mode at end of measurement.
*
*      This function will change the VL53L0X_State from VL53L0X_STATE_RUNNING
*
*      to VL53L0X_STATE_IDLE.
*
* @note This function Access to the device
*
* @param Dev          Device Handle
* @return VL53L0X_ERROR_NONE  Success
* @return "Other error code"  See ::VL53L0X_Error
*/

```

```
VL53L0X_API VL53L0X_Error VL53L0X_StopMeasurement(VL53L0X_DEV Dev);
```

```
/**
```

```
 * @brief Return Measurement Data Ready
```

```
 *
```

```
 * @par Function Description
```

```
 * This function indicate that a measurement data is ready.
```

```
 * This function check if interrupt mode is used then check is done accordingly.
```

```
 * If perform function clear the interrupt, this function will not work,
```

```
 * like in case of @a VL53L0X_PerformSingleRangingMeasurement().
```

```
 * The previous function is blocking function, VL53L0X_GetMeasurementDataReady
```

```
 * is used for non-blocking capture.
```

```
 *
```

```
 * @note This function Access to the device
```

```
 *
```

```
 * @param Dev          Device Handle
```

```
 * @param pMeasurementDataReady Pointer to Measurement Data Ready.
```

```
 * 0=data not ready, 1 = data ready
```

```
 * @return VL53L0X_ERROR_NONE    Success
```

```
 * @return "Other error code"    See ::VL53L0X_Error
```

```
 */
```

```
VL53L0X_API VL53L0X_Error VL53L0X_GetMeasurementDataReady(VL53L0X_DEV Dev,  
    uint8_t *pMeasurementDataReady);
```

```
/**
```

```
 * @brief Wait for device ready for a new measurement command.
```

```
 * Blocking function.
```

```
 *
```

```
 * @note This function is not Implemented
```

```

*

* @param Dev    Device Handle

* @param MaxLoop  Max Number of polling loop (timeout).

* @return VL53L0X_ERROR_NOT_IMPLEMENTED  Not implemented

*/

VL53L0X_API VL53L0X_Error VL53L0X_WaitDeviceReadyForNewMeasurement(VL53L0X_DEV Dev,
    uint32_t MaxLoop);

```

```

/**

* @brief Retrieve the Reference Signal after a measurements

*

* @par Function Description

* Get Reference Signal from last successful Ranging measurement

* This function return a valid value after that you call the

* @a VL53L0X_GetRangingMeasurementData().

*

* @note This function Access to the device

*

* @param Dev          Device Handle

* @param pMeasurementRefSignal  Pointer to the Ref Signal to fill up.

* @return VL53L0X_ERROR_NONE    Success

* @return "Other error code"    See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetMeasurementRefSignal(VL53L0X_DEV Dev,
    FixPoint1616_t *pMeasurementRefSignal);

```

```

/**

* @brief Retrieve the measurements from device for a given setup

*

```

```

* @par Function Description
* Get data from last successful Ranging measurement
* @warning USER should take care about @a VL53L0X_GetNumberOfROIzones()
* before get data.
* PAL will fill a NumberOfROIzones times the corresponding data
* structure used in the measurement function.
*
* @note This function Access to the device
*
* @param Dev          Device Handle
* @param pRangingMeasurementData Pointer to the data structure to fill up.
* @return VL53L0X_ERROR_NONE    Success
* @return "Other error code"    See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetRangingMeasurementData(VL53L0X_DEV Dev,
    VL53L0X_RangingMeasurementData_t *pRangingMeasurementData);

```

```

/**

```

```

* @brief Retrieve the measurements from device for a given setup
*
* @par Function Description
* Get data from last successful Histogram measurement
* @warning USER should take care about @a VL53L0X_GetNumberOfROIzones()
* before get data.
* PAL will fill a NumberOfROIzones times the corresponding data structure
* used in the measurement function.
*
* @note This function is not Implemented
*

```

```

* @param Dev          Device Handle
* @param pHistogramMeasurementData Pointer to the histogram data structure.
* @return VL53L0X_ERROR_NOT_IMPLEMENTED Not implemented
*/

VL53L0X_API VL53L0X_Error VL53L0X_GetHistogramMeasurementData(VL53L0X_DEV Dev,
    VL53L0X_HistogramMeasurementData_t *pHistogramMeasurementData);

/**
* @brief Performs a single ranging measurement and retrieve the ranging
* measurement data
*
* @par Function Description
* This function will change the device mode to VL53L0X_DEVICEMODE_SINGLE_RANGING
* with @a VL53L0X_SetDeviceMode(),
* It performs measurement with @a VL53L0X_PerformSingleMeasurement()
* It get data from last successful Ranging measurement with
* @a VL53L0X_GetRangingMeasurementData.
* Finally it clear the interrupt with @a VL53L0X_ClearInterruptMask().
*
* @note This function Access to the device
*
* @note This function change the device mode to
* VL53L0X_DEVICEMODE_SINGLE_RANGING
*
* @param Dev          Device Handle
* @param pRangingMeasurementData Pointer to the data structure to fill up.
* @return VL53L0X_ERROR_NONE      Success
* @return "Other error code"      See ::VL53L0X_Error
*/

```

```
VL53L0X_API VL53L0X_Error VL53L0X_PerformSingleRangingMeasurement(VL53L0X_DEV Dev,  
    VL53L0X_RangingMeasurementData_t *pRangingMeasurementData);
```

```
/**
```

```
 * @brief Performs a single histogram measurement and retrieve the histogram
```

```
 * measurement data
```

```
 * Is equivalent to VL53L0X_PerformSingleMeasurement +
```

```
 * VL53L0X_GetHistogramMeasurementData
```

```
 *
```

```
 * @par Function Description
```

```
 * Get data from last successful Ranging measurement.
```

```
 * This function will clear the interrupt in case of these are enabled.
```

```
 *
```

```
 * @note This function is not Implemented
```

```
 *
```

```
 * @param Dev          Device Handle
```

```
 * @param pHistogramMeasurementData Pointer to the data structure to fill up.
```

```
 * @return VL53L0X_ERROR_NOT_IMPLEMENTED Not implemented
```

```
 */
```

```
VL53L0X_API VL53L0X_Error VL53L0X_PerformSingleHistogramMeasurement(VL53L0X_DEV Dev,  
    VL53L0X_HistogramMeasurementData_t *pHistogramMeasurementData);
```

```
/**
```

```
 * @brief Set the number of ROI Zones to be used for a specific Device
```

```
 *
```

```
 * @par Function Description
```

```
 * Set the number of ROI Zones to be used for a specific Device.
```

```
 * The programmed value should be less than the max number of ROI Zones given
```

```
 * with @a VL53L0X_GetMaxNumberOfROIZones().
```

* This version of API manage only one zone.

*

* @param Dev Device Handle

* @param NumberOfROIzones Number of ROI Zones to be used for a
* specific Device.

* @return VL53L0X_ERROR_NONE Success

* @return VL53L0X_ERROR_INVALID_PARAMS This error is returned if

* NumberOfROIzones != 1

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_SetNumberOfROIzones(VL53L0X_DEV Dev,  
    uint8_t NumberOfROIzones);
```

/**

* @brief Get the number of ROI Zones managed by the Device

*

* @par Function Description

* Get number of ROI Zones managed by the Device

* USER should take care about @a VL53L0X_GetNumberOfROIzones()

* before get data after a perform measurement.

* PAL will fill a NumberOfROIzones times the corresponding data

* structure used in the measurement function.

*

* @note This function doesn't Access to the device

*

* @param Dev Device Handle

* @param pNumberOfROIzones Pointer to the Number of ROI Zones value.

* @return VL53L0X_ERROR_NONE Success

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetNumberOfROIzones(VL53L0X_DEV Dev,
```

```

uint8_t *pNumberOfROI Zones);

/**
 * @brief Get the Maximum number of ROI Zones managed by the Device
 *
 * @par Function Description
 * Get Maximum number of ROI Zones managed by the Device.
 *
 * @note This function doesn't Access to the device
 *
 * @param Dev          Device Handle
 * @param pMaxNumberOfROI Zones Pointer to the Maximum Number
 * of ROI Zones value.
 * @return VL53L0X_ERROR_NONE    Success
 */
VL53L0X_API VL53L0X_Error VL53L0X_GetMaxNumberOfROI Zones(VL53L0X_DEV Dev,
    uint8_t *pMaxNumberOfROI Zones);

/** @} VL53L0X_measurement_group */

/** @defgroup VL53L0X_interrupt_group VL53L0X Interrupt Functions
 * @brief Functions used for interrupt managements
 * @{
 */

/**
 * @brief Set the configuration of GPIO pin for a given device
 *
 * @note This function Access to the device

```

```

*

* @param Dev          Device Handle
* @param Pin          ID of the GPIO Pin
* @param Functionality Select Pin functionality.
* Refer to ::VL53L0X_GpioFunctionality
* @param DeviceMode    Device Mode associated to the Gpio.
* @param Polarity      Set interrupt polarity. Active high
* or active low see ::VL53L0X_InterruptPolarity
* @return VL53L0X_ERROR_NONE          Success
* @return VL53L0X_ERROR_GPIO_NOT_EXISTING    Only Pin=0 is accepted.
* @return VL53L0X_ERROR_GPIO_FUNCTIONALITY_NOT_SUPPORTED This error occurs
* when Functionality programmed is not in the supported list:
*
*         Supported value are:
*
*         VL53L0X_GPIOFUNCTIONALITY_OFF,
*
*         VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_LOW,
*
*         VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_HIGH,
*
*         VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_OUT,
*
*         VL53L0X_GPIOFUNCTIONALITY_NEW_MEASURE_READY
* @return "Other error code" See ::VL53L0X_Error
*/

VL53L0X_API VL53L0X_Error VL53L0X_SetGpioConfig(VL53L0X_DEV Dev, uint8_t Pin,
        VL53L0X_DeviceModes DeviceMode, VL53L0X_GpioFunctionality Functionality,
        VL53L0X_InterruptPolarity Polarity);

/**
* @brief Get current configuration for GPIO pin for a given device
*
* @note This function Access to the device
*

```

- * @param Dev Device Handle
- * @param Pin ID of the GPIO Pin
- * @param pDeviceMode Pointer to Device Mode associated to the Gpio.
- * @param pFunctionality Pointer to Pin functionality.
- * Refer to ::VL53L0X_GpioFunctionality
- * @param pPolarity Pointer to interrupt polarity.
- * Active high or active low see ::VL53L0X_InterruptPolarity
- * @return VL53L0X_ERROR_NONE Success
- * @return VL53L0X_ERROR_GPIO_NOT_EXISTING Only Pin=0 is accepted.
- * @return VL53L0X_ERROR_GPIO_FUNCTIONALITY_NOT_SUPPORTED This error occurs
- * when Functionality programmed is not in the supported list:
- * Supported value are:
- * VL53L0X_GPIOFUNCTIONALITY_OFF,
- * VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_LOW,
- * VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_HIGH,
- * VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_OUT,
- * VL53L0X_GPIOFUNCTIONALITY_NEW_MEASURE_READY
- * @return "Other error code" See ::VL53L0X_Error
- */

```
VL53L0X_API VL53L0X_Error VL53L0X_GetGpioConfig(VL53L0X_DEV Dev, uint8_t Pin,
    VL53L0X_DeviceModes *pDeviceMode,
    VL53L0X_GpioFunctionality *pFunctionality,
    VL53L0X_InterruptPolarity *pPolarity);
```

/**

- * @brief Set low and high Interrupt thresholds for a given mode
- * (ranging, ALS, ...) for a given device
- *
- * @par Function Description

- * Set low and high Interrupt thresholds for a given mode (ranging, ALS, ...)
- * for a given device
- *
- * @note This function Access to the device
- *
- * @note DeviceMode is ignored for the current device
- *
- * @param Dev Device Handle
- * @param DeviceMode Device Mode for which change thresholds
- * @param ThresholdLow Low threshold (mm, lux ..., depending on the mode)
- * @param ThresholdHigh High threshold (mm, lux ..., depending on the mode)
- * @return VL53L0X_ERROR_NONE Success
- * @return "Other error code" See ::VL53L0X_Error
- */

```
VL53L0X_API VL53L0X_Error VL53L0X_SetInterruptThresholds(VL53L0X_DEV Dev,
    VL53L0X_DeviceModes DeviceMode, FixPoint1616_t ThresholdLow,
    FixPoint1616_t ThresholdHigh);
```

/**

- * @brief Get high and low Interrupt thresholds for a given mode
- * (ranging, ALS, ...) for a given device
- *
- * @par Function Description
- * Get high and low Interrupt thresholds for a given mode (ranging, ALS, ...)
- * for a given device
- *
- * @note This function Access to the device
- *
- * @note DeviceMode is ignored for the current device

```

*
* @param Dev          Device Handle
* @param DeviceMode    Device Mode from which read thresholds
* @param pThresholdLow Low threshold (mm, lux ..., depending on the mode)
* @param pThresholdHigh High threshold (mm, lux ..., depending on the mode)
* @return VL53L0X_ERROR_NONE Success
* @return "Other error code" See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetInterruptThresholds(VL53L0X_DEV Dev,
    VL53L0X_DeviceModes DeviceMode, FixPoint1616_t *pThresholdLow,
    FixPoint1616_t *pThresholdHigh);

```

```

/**

```

```

* @brief Return device stop completion status
*
* @par Function Description
* Returns stop completion status.
* User shall call this function after a stop command
*
* @note This function Access to the device
*
* @param Dev          Device Handle
* @param pStopStatus   Pointer to status variable to update
* @return VL53L0X_ERROR_NONE Success
* @return "Other error code" See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetStopCompletedStatus(VL53L0X_DEV Dev,
    uint32_t *pStopStatus);

```

/**

* @brief Clear given system interrupt condition

*

* @par Function Description

* Clear given interrupt(s).

*

* @note This function Access to the device

*

* @param Dev Device Handle

* @param InterruptMask Mask of interrupts to clear

* @return VL53L0X_ERROR_NONE Success

* @return VL53L0X_ERROR_INTERRUPT_NOT_CLEARED Cannot clear interrupts

*

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_ClearInterruptMask(VL53L0X_DEV Dev,  
    uint32_t InterruptMask);
```

/**

* @brief Return device interrupt status

*

* @par Function Description

* Returns currently raised interrupts by the device.

* User shall be able to activate/deactivate interrupts through

* @a VL53L0X_SetGpioConfig()

*

* @note This function Access to the device

*

```

* @param Dev          Device Handle
* @param pInterruptMaskStatus Pointer to status variable to update
* @return VL53L0X_ERROR_NONE    Success
* @return "Other error code"    See ::VL53L0X_Error
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_GetInterruptMaskStatus(VL53L0X_DEV Dev,
    uint32_t *pInterruptMaskStatus);

```

```

/**
* @brief Configure ranging interrupt reported to system
*
* @note This function is not Implemented
*

```

```

* @param Dev          Device Handle
* @param InterruptMask    Mask of interrupt to Enable/disable
* (0:interrupt disabled or 1: interrupt enabled)
* @return VL53L0X_ERROR_NOT_IMPLEMENTED    Not implemented
*/

```

```

VL53L0X_API VL53L0X_Error VL53L0X_EnableInterruptMask(VL53L0X_DEV Dev,
    uint32_t InterruptMask);

```

```

/** @} VL53L0X_interrupt_group */

```

```

/** @defgroup VL53L0X_SPADfunctions_group VL53L0X SPAD Functions

```

```

* @brief Functions used for SPAD managements
* @{
*/

```

```

/**

```

* @brief Set the SPAD Ambient Damper Threshold value

*

* @par Function Description

* This function set the SPAD Ambient Damper Threshold value

*

* @note This function Access to the device

*

* @param Dev Device Handle

* @param SpadAmbientDamperThreshold SPAD Ambient Damper Threshold value

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_SetSpadAmbientDamperThreshold(VL53L0X_DEV Dev,  
    uint16_t SpadAmbientDamperThreshold);
```

/**

* @brief Get the current SPAD Ambient Damper Threshold value

*

* @par Function Description

* This function get the SPAD Ambient Damper Threshold value

*

* @note This function Access to the device

*

* @param Dev Device Handle

* @param pSpadAmbientDamperThreshold Pointer to programmed

* SPAD Ambient Damper Threshold value

* @return VL53L0X_ERROR_NONE Success

* @return "Other error code" See ::VL53L0X_Error

*/

```
VL53L0X_API VL53L0X_Error VL53L0X_GetSpadAmbientDamperThreshold(VL53L0X_DEV Dev,  
    uint16_t *pSpadAmbientDamperThreshold);
```

```
/**
```

```
 * @brief Set the SPAD Ambient Damper Factor value
```

```
 *
```

```
 * @par Function Description
```

```
 * This function set the SPAD Ambient Damper Factor value
```

```
 *
```

```
 * @note This function Access to the device
```

```
 *
```

```
 * @param Dev          Device Handle
```

```
 * @param SpadAmbientDamperFactor    SPAD Ambient Damper Factor value
```

```
 * @return VL53L0X_ERROR_NONE        Success
```

```
 * @return "Other error code"        See ::VL53L0X_Error
```

```
 */
```

```
VL53L0X_API VL53L0X_Error VL53L0X_SetSpadAmbientDamperFactor(VL53L0X_DEV Dev,  
    uint16_t SpadAmbientDamperFactor);
```

```
/**
```

```
 * @brief Get the current SPAD Ambient Damper Factor value
```

```
 *
```

```
 * @par Function Description
```

```
 * This function get the SPAD Ambient Damper Factor value
```

```
 *
```

```
 * @note This function Access to the device
```

```
 *
```

```
 * @param Dev          Device Handle
```

```
 * @param pSpadAmbientDamperFactor    Pointer to programmed SPAD Ambient
```

```

* Damper Factor value

* @return VL53L0X_ERROR_NONE      Success

* @return "Other error code"      See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetSpadAmbientDamperFactor(VL53L0X_DEV Dev,
    uint16_t *pSpadAmbientDamperFactor);

/**
* @brief Performs Reference Spad Management
*
* @par Function Description
* The reference SPAD initialization procedure determines the minimum amount
* of reference spads to be enables to achieve a target reference signal rate
* and should be performed once during initialization.
*
* @note This function Access to the device
*
* @note This function change the device mode to
* VL53L0X_DEVICEMODE_SINGLE_RANGING
*
* @param Dev      Device Handle
* @param refSpadCount      Reports ref Spad Count
* @param isApertureSpads      Reports if spads are of type
*      aperture or non-aperture.
*      1:=aperture, 0:=Non-Aperture
* @return VL53L0X_ERROR_NONE      Success
* @return VL53L0X_ERROR_REF_SPAD_INIT      Error in the Ref Spad procedure.
* @return "Other error code"      See ::VL53L0X_Error
*/

```

```
VL53L0X_API VL53L0X_Error VL53L0X_PerformRefSpadManagement(VL53L0X_DEV Dev,
    uint32_t *refSpadCount, uint8_t *isApertureSpads);
```

```
/**
```

```
* @brief Applies Reference SPAD configuration
```

```
*
```

```
* @par Function Description
```

```
* This function applies a given number of reference spads, identified as
```

```
* either Aperture or Non-Aperture.
```

```
* The requested spad count and type are stored within the device specific
```

```
* parameters data for access by the host.
```

```
*
```

```
* @note This function Access to the device
```

```
*
```

```
* @param Dev Device Handle
```

```
* @param refSpadCount Number of ref spads.
```

```
* @param isApertureSpads Defines if spads are of type
```

```
* aperture or non-aperture.
```

```
* 1:=aperture, 0:=Non-Aperture
```

```
* @return VL53L0X_ERROR_NONE Success
```

```
* @return VL53L0X_ERROR_REF_SPAD_INIT Error in the in the reference
```

```
* spad configuration.
```

```
* @return "Other error code" See ::VL53L0X_Error
```

```
*/
```

```
VL53L0X_API VL53L0X_Error VL53L0X_SetReferenceSpads(VL53L0X_DEV Dev,
    uint32_t refSpadCount, uint8_t isApertureSpads);
```

```
/**
```

```
* @brief Retrieves SPAD configuration
```

```

*

* @par Function Description

* This function retrieves the current number of applied reference spads

* and also their type : Aperture or Non-Aperture.

*

* @note This function Access to the device

*

* @param Dev          Device Handle

* @param refSpadCount    Number ref Spad Count

* @param isApertureSpads    Reports if spads are of type

*                          aperture or non-aperture.

*                          1:=aperture, 0:=Non-Aperture

* @return VL53L0X_ERROR_NONE      Success

* @return VL53L0X_ERROR_REF_SPAD_INIT  Error in the in the reference

*                          spad configuration.

* @return "Other error code"      See ::VL53L0X_Error

*/

VL53L0X_API VL53L0X_Error VL53L0X_GetReferenceSpads(VL53L0X_DEV Dev,

    uint32_t *refSpadCount, uint8_t *isApertureSpads);

/** @} VL53L0X_SPADfunctions_group */

/** @} VL53L0X_cut11_group */

#ifdef __cplusplus
}
#endif

#endif /* _VL53L0X_API_H_ */

```

1.1.33 3l0x_api.cpp

Below:

```
/******
```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

```
*****/
```

```
#define USE_I2C_2V8
```

```
#include "vl53l0x_api.h"
```

```
#include "vl53l0x_tuning.h"
```

```
#include "vl53l0x_interrupt_threshold_settings.h"
```

```
#include "vl53l0x_api_core.h"
```

```
#include "vl53l0x_api_calibration.h"
```

```
#include "vl53l0x_api_strings.h"
```

```
#ifndef __KERNEL__
```

```
#include <stdlib.h>
```

```
#endif
```

```
#define LOG_FUNCTION_START(fmt, ...) \
```

```
    _LOG_FUNCTION_START	TRACE_MODULE_API, fmt, ##__VA_ARGS__
```

```
#define LOG_FUNCTION_END(status, ...) \
```

```
    _LOG_FUNCTION_END	TRACE_MODULE_API, status, ##__VA_ARGS__
```

```
#define LOG_FUNCTION_END_FMT(status, fmt, ...) \
```

```
    _LOG_FUNCTION_END_FMT	TRACE_MODULE_API, status, fmt, ##__VA_ARGS__
```

```
#ifdef VL53L0X_LOG_ENABLE
```

```
#define trace_print(level, ...) trace_print_module_function	TRACE_MODULE_API, \
```

```
    level, TRACE_FUNCTION_NONE, ##__VA_ARGS__
```

```
#endif
```

```
/* Group PAL General Functions */
```

```
VL53L0X_Error VL53L0X_GetVersion(VL53L0X_Version_t *pVersion)
```

```
{
```

```

VL53L0X_Error Status = VL53L0X_ERROR_NONE;

LOG_FUNCTION_START("");

pVersion->major = VL53L0X_IMPLEMENTATION_VER_MAJOR;
pVersion->minor = VL53L0X_IMPLEMENTATION_VER_MINOR;
pVersion->build = VL53L0X_IMPLEMENTATION_VER_SUB;

pVersion->revision = VL53L0X_IMPLEMENTATION_VER_REVISION;

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_GetPalSpecVersion(VL53L0X_Version_t *pPalSpecVersion)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    pPalSpecVersion->major = VL53L0X_SPECIFICATION_VER_MAJOR;
    pPalSpecVersion->minor = VL53L0X_SPECIFICATION_VER_MINOR;
    pPalSpecVersion->build = VL53L0X_SPECIFICATION_VER_SUB;

    pPalSpecVersion->revision = VL53L0X_SPECIFICATION_VER_REVISION;

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetProductRevision(VL53L0X_DEV Dev,
    uint8_t *pProductRevisionMajor, uint8_t *pProductRevisionMinor)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t revision_id;

    LOG_FUNCTION_START("");

    Status = VL53L0X_RdByte(Dev, VL53L0X_REG_IDENTIFICATION_REVISION_ID,
        &revision_id);

    *pProductRevisionMajor = 1;
    *pProductRevisionMinor = (revision_id & 0xF0) >> 4;

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetDeviceInfo(VL53L0X_DEV Dev,
    VL53L0X_DeviceInfo_t *pVL53L0X_DeviceInfo)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_get_device_info(Dev, pVL53L0X_DeviceInfo);

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetDeviceErrorStatus(VL53L0X_DEV Dev,
      VL53L0X_DeviceError *pDeviceErrorStatus)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t RangeStatus;

    LOG_FUNCTION_START("");

    Status = VL53L0X_RdByte(Dev, VL53L0X_REG_RESULT_RANGE_STATUS,
        &RangeStatus);

    *pDeviceErrorStatus = (VL53L0X_DeviceError)((RangeStatus & 0x78) >> 3);

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetDeviceErrorString(VL53L0X_DeviceError ErrorCode,
      char *pDeviceErrorString)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_get_device_error_string(ErrorCode, pDeviceErrorString);

    LOG_FUNCTION_END(Status);
}

```

```
        return Status;
    }
}
```

```
VL53L0X_Error VL53L0X_GetRangeStatusString(uint8_t RangeStatus,
    char *pRangeStatusString)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    LOG_FUNCTION_START("");

    Status = VL53L0X_get_range_status_string(RangeStatus,
        pRangeStatusString);

    LOG_FUNCTION_END(Status);
    return Status;
}
```

```
VL53L0X_Error VL53L0X_GetPalErrorString(VL53L0X_Error PalErrorCode,
    char *pPalErrorString)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    LOG_FUNCTION_START("");

    Status = VL53L0X_get_pal_error_string(PalErrorCode, pPalErrorString);

    LOG_FUNCTION_END(Status);
    return Status;
}
```

```
VL53L0X_Error VL53L0X_GetPalStateString(VL53L0X_State PalStateCode,
```

```

    char *pPalStateString)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_get_pal_state_string(PalStateCode, pPalStateString);

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_GetPalState(VL53L0X_DEV Dev, VL53L0X_State *pPalState)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    *pPalState = PALDevDataGet(Dev, PalState);

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_SetPowerMode(VL53L0X_DEV Dev, VL53L0X_PowerModes PowerMode)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    /* Only level1 of Power mode exists */
    if ((PowerMode != VL53L0X_POWERMODE_STANDBY_LEVEL1)

```

```

        && (PowerMode != VL53L0X_POWERMODE_IDLE_LEVEL1)) {
            Status = VL53L0X_ERROR_MODE_NOT_SUPPORTED;
        } else if (PowerMode == VL53L0X_POWERMODE_STANDBY_LEVEL1) {
            /* set the standby level1 of power mode */
            Status = VL53L0X_WrByte(Dev, 0x80, 0x00);
            if (Status == VL53L0X_ERROR_NONE) {
                /* Set PAL State to standby */
                PALDevDataSet(Dev, PalState, VL53L0X_STATE_STANDBY);
                PALDevDataSet(Dev, PowerMode,
                               VL53L0X_POWERMODE_STANDBY_LEVEL1);
            }

        } else {
            /* VL53L0X_POWERMODE_IDLE_LEVEL1 */
            Status = VL53L0X_WrByte(Dev, 0x80, 0x00);
            if (Status == VL53L0X_ERROR_NONE)
                Status = VL53L0X_StaticInit(Dev);

            if (Status == VL53L0X_ERROR_NONE)
                PALDevDataSet(Dev, PowerMode,
                               VL53L0X_POWERMODE_IDLE_LEVEL1);
        }

        LOG_FUNCTION_END(Status);
        return Status;
    }

VL53L0X_Error VL53L0X_GetPowerMode(VL53L0X_DEV Dev, VL53L0X_PowerModes *pPowerMode)

```

```

{

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t Byte;

    LOG_FUNCTION_START("");

    /* Only level1 of Power mode exists */

    Status = VL53L0X_RdByte(Dev, 0x80, &Byte);

    if (Status == VL53L0X_ERROR_NONE) {
        if (Byte == 1) {
            PALDevDataSet(Dev, PowerMode,
                          VL53L0X_POWERMODE_IDLE_LEVEL1);
        } else {
            PALDevDataSet(Dev, PowerMode,
                          VL53L0X_POWERMODE_STANDBY_LEVEL1);
        }
    }

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_SetOffsetCalibrationDataMicroMeter(VL53L0X_DEV Dev,
    int32_t OffsetCalibrationDataMicroMeter)
{

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_set_offset_calibration_data_micro_meter(Dev,

```

```

        OffsetCalibrationDataMicroMeter);

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_GetOffsetCalibrationDataMicroMeter(VL53L0X_DEV Dev,
    int32_t *pOffsetCalibrationDataMicroMeter)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_get_offset_calibration_data_micro_meter(Dev,
        pOffsetCalibrationDataMicroMeter);

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_SetLinearityCorrectiveGain(VL53L0X_DEV Dev,
    int16_t LinearityCorrectiveGain)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    if ((LinearityCorrectiveGain < 0) || (LinearityCorrectiveGain > 1000))
        Status = VL53L0X_ERROR_INVALID_PARAMS;

    else {
        PALDevDataSet(Dev, LinearityCorrectiveGain,

```

```

        LinearityCorrectiveGain);

    if (LinearityCorrectiveGain != 1000) {
        /* Disable FW Xtalk */
        Status = VL53L0X_WrWord(Dev,
            VL53L0X_REG_CROSSTALK_COMPENSATION_PEAK_RATE_MCPS, 0);
    }
}

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_GetLinearityCorrectiveGain(VL53L0X_DEV Dev,
    uint16_t *pLinearityCorrectiveGain)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    LOG_FUNCTION_START("");

    *pLinearityCorrectiveGain = PALDevDataGet(Dev, LinearityCorrectiveGain);

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_SetGroupParamHold(VL53L0X_DEV Dev, uint8_t GroupParamHold)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NOT_IMPLEMENTED;
    LOG_FUNCTION_START("");

```

```

/* not implemented on VL53L0X */

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_GetUpperLimitMilliMeter(VL53L0X_DEV Dev,
        uint16_t *pUpperLimitMilliMeter)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NOT_IMPLEMENTED;
    LOG_FUNCTION_START("");

    /* not implemented on VL53L0X */

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_GetTotalSignalRate(VL53L0X_DEV Dev,
        FixPoint1616_t *pTotalSignalRate)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    VL53L0X_RangingMeasurementData_t LastRangeDataBuffer;

    LOG_FUNCTION_START("");

    LastRangeDataBuffer = PALDevDataGet(Dev, LastRangeMeasure);

```

```

        Status = VL53L0X_get_total_signal_rate(
            Dev, &LastRangeDataBuffer, pTotalSignalRate);

    LOG_FUNCTION_END(Status);
    return Status;
}

/* End Group PAL General Functions */

/* Group PAL Init Functions */
VL53L0X_Error VL53L0X_SetDeviceAddress(VL53L0X_DEV Dev, uint8_t DeviceAddress)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    LOG_FUNCTION_START("");

    Status = VL53L0X_WrByte(Dev, VL53L0X_REG_I2C_SLAVE_DEVICE_ADDRESS,
        DeviceAddress / 2);

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_DataInit(VL53L0X_DEV Dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    VL53L0X_DeviceParameters_t CurrentParameters;
    int i;
    uint8_t StopVariable;

```

```

LOG_FUNCTION_START("");

/* by default the I2C is running at 1V8 if you want to change it you
 * need to include this define at compilation level. */
#ifdef USE_I2C_2V8
    Status = VL53L0X_UpdateByte(Dev,
                                VL53L0X_REG_VHV_CONFIG_PAD_SCL_SDA__EXTSUP_HV,
                                0xFE,
                                0x01);
#endif

/* Set I2C standard mode */
if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev, 0x88, 0x00);

/* read WHO_AM_I */
uint8_t b;
Status = VL53L0X_RdByte(Dev, 0xC0, &b);
//Serial.print("WHOAMI: 0x"); Serial.println(b, HEX);

/* read WHO_AM_I */

VL53L0X_SETDEVICESPECIFICPARAMETER(Dev, ReadDataFromDeviceDone, 0);

#ifdef USE_IQC_STATION
    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_apply_offset_adjustment(Dev);
#endif

```

```

/* Default value is 1000 for Linearity Corrective Gain */
PALDevDataSet(Dev, LinearityCorrectiveGain, 1000);

/* Dmax default Parameter */
PALDevDataSet(Dev, DmaxCalRangeMilliMeter, 400);
PALDevDataSet(Dev, DmaxCalSignalRateRtnMegaCps,
               (FixPoint1616_t)((0x00016B85))); /* 1.42 No Cover Glass*/

/* Set Default static parameters
 *set first temporary values 9.44MHz * 65536 = 618660 */
VL53L0X_SETDEVICESPECIFICPARAMETER(Dev, OscFrequencyMHz, 618660);

/* Set Default XTalkCompensationRateMegaCps to 0 */
VL53L0X_SETPARAMETERFIELD(Dev, XTalkCompensationRateMegaCps, 0);

/* Get default parameters */
Status = VL53L0X_GetDeviceParameters(Dev, &CurrentParameters);

if (Status == VL53L0X_ERROR_NONE) {
    /* initialize PAL values */
    CurrentParameters.DeviceMode = VL53L0X_DEVICEMODE_SINGLE_RANGING;
    CurrentParameters.HistogramMode = VL53L0X_HISTOGRAMMODE_DISABLED;
    PALDevDataSet(Dev, CurrentParameters, CurrentParameters);
}

/* Sigma estimator variable */
PALDevDataSet(Dev, SigmaEstRefArray, 100);
PALDevDataSet(Dev, SigmaEstEffPulseWidth, 900);
PALDevDataSet(Dev, SigmaEstEffAmbWidth, 500);

```

```
PALDevDataSet(Dev, targetRefRate, 0x0A00); /* 20 MCPS in 9:7 format */
```

```
/* Use internal default settings */
```

```
PALDevDataSet(Dev, UseInternalTuningSettings, 1);
```

```
Status |= VL53L0X_WrByte(Dev, 0x80, 0x01);
```

```
Status |= VL53L0X_WrByte(Dev, 0xFF, 0x01);
```

```
Status |= VL53L0X_WrByte(Dev, 0x00, 0x00);
```

```
Status |= VL53L0X_RdByte(Dev, 0x91, &StopVariable);
```

```
PALDevDataSet(Dev, StopVariable, StopVariable);
```

```
Status |= VL53L0X_WrByte(Dev, 0x00, 0x01);
```

```
Status |= VL53L0X_WrByte(Dev, 0xFF, 0x00);
```

```
Status |= VL53L0X_WrByte(Dev, 0x80, 0x00);
```

```
/* Enable all check */
```

```
for (i = 0; i < VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS; i++) {
```

```
    if (Status == VL53L0X_ERROR_NONE)
```

```
        Status |= VL53L0X_SetLimitCheckEnable(Dev, i, 1);
```

```
    else
```

```
        break;
```

```
}
```

```
/* Disable the following checks */
```

```
if (Status == VL53L0X_ERROR_NONE)
```

```
    Status = VL53L0X_SetLimitCheckEnable(Dev,
```

```
        VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP, 0);
```

```
if (Status == VL53L0X_ERROR_NONE)
```

```

        Status = VL53L0X_SetLimitCheckEnable(Dev,
            VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD, 0);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_SetLimitCheckEnable(Dev,
            VL53L0X_CHECKENABLE_SIGNAL_RATE_MSRC, 0);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_SetLimitCheckEnable(Dev,
            VL53L0X_CHECKENABLE_SIGNAL_RATE_PRE_RANGE, 0);

    /* Limit default values */
    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_SetLimitCheckValue(Dev,
            VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,
            (FixPoint1616_t)(18 * 65536));
    }

    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_SetLimitCheckValue(Dev,
            VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
            (FixPoint1616_t)(25 * 65536 / 100));
        /* 0.25 * 65536 */
    }

    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_SetLimitCheckValue(Dev,
            VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP,
            (FixPoint1616_t)(35 * 65536));
    }

```

```

if (Status == VL53L0X_ERROR_NONE) {
    Status = VL53L0X_SetLimitCheckValue(Dev,
        VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD,
        (FixPoint1616_t)(0 * 65536));
}

if (Status == VL53L0X_ERROR_NONE) {

    PALDevDataSet(Dev, SequenceConfig, 0xFF);
    Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG,
        0xFF);

    /* Set PAL state to tell that we are waiting for call to
     * VL53L0X_StaticInit */
    PALDevDataSet(Dev, PalState, VL53L0X_STATE_WAIT_STATICINIT);
}

if (Status == VL53L0X_ERROR_NONE)
    VL53L0X_SETDEVICESPECIFICPARAMETER(Dev, RefSpadsInitialised, 0);

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_SetTuningSettingBuffer(VL53L0X_DEV Dev,
    uint8_t *pTuningSettingBuffer, uint8_t UseInternalTuningSettings)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

```

```

LOG_FUNCTION_START("");

if (UseInternalTuningSettings == 1) {
    /* Force use internal settings */
    PALDevDataSet(Dev, UseInternalTuningSettings, 1);
} else {

    /* check that the first byte is not 0 */
    if (*pTuningSettingBuffer != 0) {
        PALDevDataSet(Dev, pTuningSettingsPointer,
                      pTuningSettingBuffer);
        PALDevDataSet(Dev, UseInternalTuningSettings, 0);

    } else {
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
}

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_GetTuningSettingBuffer(VL53L0X_DEV Dev,
uint8_t **ppTuningSettingBuffer, uint8_t *pUseInternalTuningSettings)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

```

```

    *ppTuningSettingBuffer = PALDevDataGet(Dev, pTuningSettingsPointer);

    *pUseInternalTuningSettings = PALDevDataGet(Dev,
        UseInternalTuningSettings);

    LOG_FUNCTION_END(Status);

    return Status;
}

```

```

VL53L0X_Error VL53L0X_StaticInit(VL53L0X_DEV Dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    VL53L0X_DeviceParameters_t CurrentParameters = {0};

    uint8_t *pTuningSettingBuffer;

    uint16_t tempword = 0;

    uint8_t tempbyte = 0;

    uint8_t UseInternalTuningSettings = 0;

    uint32_t count = 0;

    uint8_t isApertureSpads = 0;

    uint32_t refSpadCount = 0;

    uint8_t ApertureSpads = 0;

    uint8_t vcselPulsePeriodPCLK;

    FixPoint1616_t seqTimeoutMilliSecs;

    LOG_FUNCTION_START("");

    Status = VL53L0X_get_info_from_device(Dev, 1);

    /* set the ref spad from NVM */

```

```

count  = (uint32_t)VL53L0X_GETDEVICESPECIFICPARAMETER(Dev,
    ReferenceSpadCount);

ApertureSpads = VL53L0X_GETDEVICESPECIFICPARAMETER(Dev,
    ReferenceSpadType);

/* NVM value invalid */
if ((ApertureSpads > 1) ||
    ((ApertureSpads == 1) && (count > 32)) ||
    ((ApertureSpads == 0) && (count > 12)))
    Status = VL53L0X_perform_ref_spad_management(Dev, &refSpadCount,
        &isApertureSpads);
else
    Status = VL53L0X_set_reference_spads(Dev, count, ApertureSpads);

/* Initialize tuning settings buffer to prevent compiler warning. */
pTuningSettingBuffer = DefaultTuningSettings;

if (Status == VL53L0X_ERROR_NONE) {
    UseInternalTuningSettings = PALDevDataGet(Dev,
        UseInternalTuningSettings);

    if (UseInternalTuningSettings == 0)
        pTuningSettingBuffer = PALDevDataGet(Dev,
            pTuningSettingsPointer);
    else
        pTuningSettingBuffer = DefaultTuningSettings;
}

```

```

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_load_tuning_settings(Dev, pTuningSettingBuffer);

/* Set interrupt config to new sample ready */
if (Status == VL53L0X_ERROR_NONE) {
    Status = VL53L0X_SetGpioConfig(Dev, 0, 0,
    VL53L0X_REG_SYSTEM_INTERRUPT_GPIO_NEW_SAMPLE_READY,
    VL53L0X_INTERRUPTPOLARITY_LOW);
}

if (Status == VL53L0X_ERROR_NONE) {
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x01);
    Status |= VL53L0X_RdWord(Dev, 0x84, &tempword);
    Status |= VL53L0X_WrByte(Dev, 0xFF, 0x00);
}

if (Status == VL53L0X_ERROR_NONE) {
    VL53L0X_SETDEVICESPECIFICPARAMETER(Dev, OscFrequencyMHz,
    VL53L0X_FIXPOINT412TOFIXPOINT1616(tempword));
}

/* After static init, some device parameters may be changed,
 * so update them */
if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_GetDeviceParameters(Dev, &CurrentParameters);

```

```
if (Status == VL53L0X_ERROR_NONE) {  
    Status = VL53L0X_GetFractionEnable(Dev, &tempbyte);  
    if (Status == VL53L0X_ERROR_NONE)  
        PALDevDataSet(Dev, RangeFractionalEnable, tempbyte);  
}
```

```
if (Status == VL53L0X_ERROR_NONE)  
    PALDevDataSet(Dev, CurrentParameters, CurrentParameters);
```

```
/* read the sequence config and save it */
```

```
if (Status == VL53L0X_ERROR_NONE) {  
    Status = VL53L0X_RdByte(Dev,  
        VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG, &tempbyte);  
    if (Status == VL53L0X_ERROR_NONE)  
        PALDevDataSet(Dev, SequenceConfig, tempbyte);  
}
```

```
/* Disable MSRC and TCC by default */
```

```
if (Status == VL53L0X_ERROR_NONE)  
    Status = VL53L0X_SetSequenceStepEnable(Dev,  
        VL53L0X_SEQUENCESTEP_TCC, 0);
```

```
if (Status == VL53L0X_ERROR_NONE)  
    Status = VL53L0X_SetSequenceStepEnable(Dev,  
        VL53L0X_SEQUENCESTEP_MSRC, 0);
```

```
/* Set PAL State to standby */  
if (Status == VL53L0X_ERROR_NONE)  
    PALDevDataSet(Dev, PalState, VL53L0X_STATE_IDLE);
```

```
/* Store pre-range vcsel period */  
if (Status == VL53L0X_ERROR_NONE) {  
    Status = VL53L0X_GetVcselPulsePeriod(  
        Dev,  
        VL53L0X_VCSEL_PERIOD_PRE_RANGE,  
        &vcselPulsePeriodPCLK);  
}
```

```
if (Status == VL53L0X_ERROR_NONE) {  
    VL53L0X_SETDEVICESPECIFICPARAMETER(  
        Dev,  
        PreRangeVcselPulsePeriod,  
        vcselPulsePeriodPCLK);  
}
```

```
/* Store final-range vcsel period */  
if (Status == VL53L0X_ERROR_NONE) {  
    Status = VL53L0X_GetVcselPulsePeriod(  
        Dev,  
        VL53L0X_VCSEL_PERIOD_FINAL_RANGE,  
        &vcselPulsePeriodPCLK);
```

```
}
```

```
if (Status == VL53L0X_ERROR_NONE) {  
    VL53L0X_SETDEVICESPECIFICPARAMETER(  
        Dev,  
        FinalRangeVcselPulsePeriod,  
        vcselPulsePeriodPCLK);  
}
```

```
/* Store pre-range timeout */
```

```
if (Status == VL53L0X_ERROR_NONE) {  
    Status = VL53L0X_GetSequenceStepTimeout(  
        Dev,  
        VL53L0X_SEQUENCESTEP_PRE_RANGE,  
        &seqTimeoutMillisecs);  
}
```

```
if (Status == VL53L0X_ERROR_NONE) {  
    VL53L0X_SETDEVICESPECIFICPARAMETER(  
        Dev,  
        PreRangeTimeoutMicrosecs,  
        seqTimeoutMillisecs);  
}
```

```
/* Store final-range timeout */
```

```
if (Status == VL53L0X_ERROR_NONE) {  
    Status = VL53L0X_GetSequenceStepTimeout(  
        Dev,  
        VL53L0X_SEQUENCESTEP_FINAL_RANGE,
```

```

        &seqTimeoutMillisecs);
    }

    if (Status == VL53L0X_ERROR_NONE) {
        VL53L0X_SETDEVICESPECIFICPARAMETER(
            Dev,
            FinalRangeTimeoutMicroSecs,
            seqTimeoutMillisecs);
    }

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_WaitDeviceBooted(VL53L0X_DEV Dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NOT_IMPLEMENTED;
    LOG_FUNCTION_START("");

    /* not implemented on VL53L0X */

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_ResetDevice(VL53L0X_DEV Dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t Byte;

```

```

LOG_FUNCTION_START("");

/* Set reset bit */
Status = VL53L0X_WrByte(Drv, VL53L0X_REG_SOFT_RESET_GO2_SOFT_RESET_N,
                        0x00);

/* Wait for some time */
if (Status == VL53L0X_ERROR_NONE) {
    do {
        Status = VL53L0X_RdByte(Drv,
                                VL53L0X_REG_IDENTIFICATION_MODEL_ID, &Byte);
    } while (Byte != 0x00);
}

/* Release reset */
Status = VL53L0X_WrByte(Drv, VL53L0X_REG_SOFT_RESET_GO2_SOFT_RESET_N,
                        0x01);

/* Wait until correct boot-up of the device */
if (Status == VL53L0X_ERROR_NONE) {
    do {
        Status = VL53L0X_RdByte(Drv,
                                VL53L0X_REG_IDENTIFICATION_MODEL_ID, &Byte);
    } while (Byte == 0x00);
}

/* Set PAL State to VL53L0X_STATE_POWERDOWN */
if (Status == VL53L0X_ERROR_NONE)
    PALDevDataSet(Drv, PalState, VL53L0X_STATE_POWERDOWN);

```

```

        LOG_FUNCTION_END(Status);

        return Status;
    }

/* End Group PAL Init Functions */

/* Group PAL Parameters Functions */
VL53L0X_Error VL53L0X_SetDeviceParameters(VL53L0X_DEV Dev,
        const VL53L0X_DeviceParameters_t *pDeviceParameters)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    int i;

    LOG_FUNCTION_START("");

    Status = VL53L0X_SetDeviceMode(Dev, pDeviceParameters->DeviceMode);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_SetInterMeasurementPeriodMilliseconds(Dev,
            pDeviceParameters->InterMeasurementPeriodMilliseconds);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_SetXTalkCompensationRateMegaCps(Dev,
            pDeviceParameters->XTalkCompensationRateMegaCps);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_SetOffsetCalibrationDataMicroMeter(Dev,
            pDeviceParameters->RangeOffsetMicroMeters);

```

```

for (i = 0; i < VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS; i++) {
    if (Status == VL53L0X_ERROR_NONE)
        Status |= VL53L0X_SetLimitCheckEnable(Dev, i,
            pDeviceParameters->LimitChecksEnable[i]);
    else
        break;

    if (Status == VL53L0X_ERROR_NONE)
        Status |= VL53L0X_SetLimitCheckValue(Dev, i,
            pDeviceParameters->LimitChecksValue[i]);
    else
        break;
}

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_SetWrapAroundCheckEnable(Dev,
        pDeviceParameters->WrapAroundCheckEnable);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_SetMeasurementTimingBudgetMicroSeconds(Dev,
        pDeviceParameters->MeasurementTimingBudgetMicroSeconds);

LOG_FUNCTION_END(Status);

return Status;
}

```

```

VL53L0X_Error VL53L0X_GetDeviceParameters(VL53L0X_DEV Dev,
    VL53L0X_DeviceParameters_t *pDeviceParameters)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int i;

    LOG_FUNCTION_START("");

    Status = VL53L0X_GetDeviceMode(Dev, &(pDeviceParameters->DeviceMode));

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_GetInterMeasurementPeriodMilliseconds(Dev,
            &(pDeviceParameters->InterMeasurementPeriodMilliseconds));

    if (Status == VL53L0X_ERROR_NONE)
        pDeviceParameters->XTalkCompensationEnable = 0;

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_GetXTalkCompensationRateMegaCps(Dev,
            &(pDeviceParameters->XTalkCompensationRateMegaCps));

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_GetOffsetCalibrationDataMicroMeter(Dev,
            &(pDeviceParameters->RangeOffsetMicroMeters));

```

```

if (Status == VL53L0X_ERROR_NONE) {
    for (i = 0; i < VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS; i++) {
        /* get first the values, then the enables.
        * VL53L0X_GetLimitCheckValue will modify the enable
        * flags
        */
        if (Status == VL53L0X_ERROR_NONE) {
            Status |= VL53L0X_GetLimitCheckValue(Dev, i,
            &(pDeviceParameters->LimitChecksValue[i]));
        } else {
            break;
        }
        if (Status == VL53L0X_ERROR_NONE) {
            Status |= VL53L0X_GetLimitCheckEnable(Dev, i,
            &(pDeviceParameters->LimitChecksEnable[i]));
        } else {
            break;
        }
    }
}

if (Status == VL53L0X_ERROR_NONE) {
    Status = VL53L0X_GetWrapAroundCheckEnable(Dev,
    &(pDeviceParameters->WrapAroundCheckEnable));
}

/* Need to be done at the end as it uses VCSELPulsePeriod */
if (Status == VL53L0X_ERROR_NONE) {
    Status = VL53L0X_GetMeasurementTimingBudgetMicroSeconds(Dev,

```

```

        &(pDeviceParameters->MeasurementTimingBudgetMicroSeconds));
    }

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_SetDeviceMode(VL53L0X_DEV Dev, VL53L0X_DeviceModes DeviceMode)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("%d", (int)DeviceMode);

    switch (DeviceMode) {
    case VL53L0X_DEVICEMODE_SINGLE_RANGING:
    case VL53L0X_DEVICEMODE_CONTINUOUS_RANGING:
    case VL53L0X_DEVICEMODE_CONTINUOUS_TIMED_RANGING:
    case VL53L0X_DEVICEMODE_GPIO_DRIVE:
    case VL53L0X_DEVICEMODE_GPIO_OSC:
        /* Supported modes */
        VL53L0X_SETPARAMETERFIELD(Dev, DeviceMode, DeviceMode);
        break;
    default:
        /* Unsupported mode */
        Status = VL53L0X_ERROR_MODE_NOT_SUPPORTED;
    }

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```
}
```

```
VL53L0X_Error VL53L0X_GetDeviceMode(VL53L0X_DEV Dev,
```

```
    VL53L0X_DeviceModes *pDeviceMode)
```

```
{
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    LOG_FUNCTION_START("");
```

```
    VL53L0X_GETPARAMETERFIELD(Dev, DeviceMode, *pDeviceMode);
```

```
    LOG_FUNCTION_END(Status);
```

```
    return Status;
```

```
}
```

```
VL53L0X_Error VL53L0X_SetRangeFractionEnable(VL53L0X_DEV Dev,    uint8_t Enable)
```

```
{
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    LOG_FUNCTION_START("%d", (int)Enable);
```

```
    Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSTEM_RANGE_CONFIG, Enable);
```

```
    if (Status == VL53L0X_ERROR_NONE)
```

```
        PALDevDataSet(Dev, RangeFractionalEnable, Enable);
```

```
    LOG_FUNCTION_END(Status);
```

```
    return Status;
```

```
}
```

```

VL53L0X_Error VL53L0X_GetFractionEnable(VL53L0X_DEV Dev, uint8_t *pEnabled)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    LOG_FUNCTION_START("");

    Status = VL53L0X_RdByte(Dev, VL53L0X_REG_SYSTEM_RANGE_CONFIG, pEnabled);

    if (Status == VL53L0X_ERROR_NONE)
        *pEnabled = (*pEnabled & 1);

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_SetHistogramMode(VL53L0X_DEV Dev,
    VL53L0X_HistogramModes HistogramMode)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NOT_IMPLEMENTED;
    LOG_FUNCTION_START("");

    /* not implemented on VL53L0X */

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetHistogramMode(VL53L0X_DEV Dev,
    VL53L0X_HistogramModes *pHistogramMode)
{

```

```

VL53L0X_Error Status = VL53L0X_ERROR_NOT_IMPLEMENTED;

LOG_FUNCTION_START("");

/* not implemented on VL53L0X */

LOG_FUNCTION_END(Status);

return Status;
}

VL53L0X_Error VL53L0X_SetMeasurementTimingBudgetMicroSeconds(VL53L0X_DEV Dev,
    uint32_t MeasurementTimingBudgetMicroSeconds)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_set_measurement_timing_budget_micro_seconds(Dev,
        MeasurementTimingBudgetMicroSeconds);

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_GetMeasurementTimingBudgetMicroSeconds(VL53L0X_DEV Dev,
    uint32_t *pMeasurementTimingBudgetMicroSeconds)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

```

```

        Status = VL53L0X_get_measurement_timing_budget_micro_seconds(Dev,
                                pMeasurementTimingBudgetMicroSeconds);

    LOG_FUNCTION_END(Status);

    return Status;
}

```

```

VL53L0X_Error VL53L0X_SetVcselPulsePeriod(VL53L0X_DEV Dev,
        VL53L0X_VcselPeriod VcselPeriodType, uint8_t VCSELPulsePeriodPCLK)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_set_vcsel_pulse_period(Dev, VcselPeriodType,
        VCSELPulsePeriodPCLK);

    LOG_FUNCTION_END(Status);

    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetVcselPulsePeriod(VL53L0X_DEV Dev,
        VL53L0X_VcselPeriod VcselPeriodType, uint8_t *pVCSELPulsePeriodPCLK)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_get_vcsel_pulse_period(Dev, VcselPeriodType,
        pVCSELPulsePeriodPCLK);
}

```

```

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_SetSequenceStepEnable(VL53L0X_DEV Dev,
    VL53L0X_SequenceStepId SequenceStepId, uint8_t SequenceStepEnabled)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t SequenceConfig = 0;
    uint8_t SequenceConfigNew = 0;
    uint32_t MeasurementTimingBudgetMicroSeconds;

    LOG_FUNCTION_START("");

    Status = VL53L0X_RdByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG,
        &SequenceConfig);

    SequenceConfigNew = SequenceConfig;

    if (Status == VL53L0X_ERROR_NONE) {
        if (SequenceStepEnabled == 1) {

            /* Enable requested sequence step
             */
            switch (SequenceStepId) {
                case VL53L0X_SEQUENCESTEP_TCC:
                    SequenceConfigNew |= 0x10;
                    break;

                case VL53L0X_SEQUENCESTEP_DSS:
                    SequenceConfigNew |= 0x28;

```

```

        break;
    case VL53L0X_SEQUENCESTEP_MSRC:
        SequenceConfigNew |= 0x04;
        break;
    case VL53L0X_SEQUENCESTEP_PRE_RANGE:
        SequenceConfigNew |= 0x40;
        break;
    case VL53L0X_SEQUENCESTEP_FINAL_RANGE:
        SequenceConfigNew |= 0x80;
        break;
    default:
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
} else {
    /* Disable requested sequence step
    */
    switch (SequenceStepId) {
    case VL53L0X_SEQUENCESTEP_TCC:
        SequenceConfigNew &= 0xef;
        break;
    case VL53L0X_SEQUENCESTEP_DSS:
        SequenceConfigNew &= 0xd7;
        break;
    case VL53L0X_SEQUENCESTEP_MSRC:
        SequenceConfigNew &= 0xfb;
        break;
    case VL53L0X_SEQUENCESTEP_PRE_RANGE:
        SequenceConfigNew &= 0xbf;
        break;
    }
}

```

```

        case VL53L0X_SEQUENCESTEP_FINAL_RANGE:
            SequenceConfigNew &= 0x7f;
            break;
        default:
            Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
}

if (SequenceConfigNew != SequenceConfig) {
    /* Apply New Setting */
    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_WrByte(Dev,
            VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG, SequenceConfigNew);
    }
    if (Status == VL53L0X_ERROR_NONE)
        PALDevDataSet(Dev, SequenceConfig, SequenceConfigNew);

    /* Recalculate timing budget */
    if (Status == VL53L0X_ERROR_NONE) {
        VL53L0X_GETPARAMETERFIELD(Dev,
            MeasurementTimingBudgetMicroSeconds,
            MeasurementTimingBudgetMicroSeconds);

        VL53L0X_SetMeasurementTimingBudgetMicroSeconds(Dev,
            MeasurementTimingBudgetMicroSeconds);
    }
}

```

```

LOG_FUNCTION_END(Status);

return Status;
}

VL53L0X_Error sequence_step_enabled(VL53L0X_DEV Dev,
VL53L0X_SequenceStepId SequenceStepId, uint8_t SequenceConfig,
uint8_t *pSequenceStepEnabled)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    *pSequenceStepEnabled = 0;
    LOG_FUNCTION_START("");

    switch (SequenceStepId) {
    case VL53L0X_SEQUENCESTEP_TCC:
        *pSequenceStepEnabled = (SequenceConfig & 0x10) >> 4;
        break;

    case VL53L0X_SEQUENCESTEP_DSS:
        *pSequenceStepEnabled = (SequenceConfig & 0x08) >> 3;
        break;

    case VL53L0X_SEQUENCESTEP_MSRC:
        *pSequenceStepEnabled = (SequenceConfig & 0x04) >> 2;
        break;

    case VL53L0X_SEQUENCESTEP_PRE_RANGE:
        *pSequenceStepEnabled = (SequenceConfig & 0x40) >> 6;
        break;

    case VL53L0X_SEQUENCESTEP_FINAL_RANGE:
        *pSequenceStepEnabled = (SequenceConfig & 0x80) >> 7;

```

```

        break;

default:
    Status = VL53L0X_ERROR_INVALID_PARAMS;
}

LOG_FUNCTION_END(Status);

return Status;
}

VL53L0X_Error VL53L0X_GetSequenceStepEnable(VL53L0X_DEV Dev,
    VL53L0X_SequenceStepId SequenceStepId, uint8_t *pSequenceStepEnabled)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t SequenceConfig = 0;

    LOG_FUNCTION_START("");

    Status = VL53L0X_RdByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG,
        &SequenceConfig);

    if (Status == VL53L0X_ERROR_NONE) {
        Status = sequence_step_enabled(Dev, SequenceStepId,
            SequenceConfig, pSequenceStepEnabled);
    }

    LOG_FUNCTION_END(Status);

    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetSequenceStepEnables(VL53L0X_DEV Dev,

```

```

VL53L0X_SchedulerSequenceSteps_t *pSchedulerSequenceSteps)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t SequenceConfig = 0;
    LOG_FUNCTION_START("");

    Status = VL53L0X_RdByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG,
        &SequenceConfig);

    if (Status == VL53L0X_ERROR_NONE) {
        Status = sequence_step_enabled(Dev,
            VL53L0X_SEQUENCESTEP_TCC, SequenceConfig,
            &pSchedulerSequenceSteps->TccOn);
    }

    if (Status == VL53L0X_ERROR_NONE) {
        Status = sequence_step_enabled(Dev,
            VL53L0X_SEQUENCESTEP_DSS, SequenceConfig,
            &pSchedulerSequenceSteps->DssOn);
    }

    if (Status == VL53L0X_ERROR_NONE) {
        Status = sequence_step_enabled(Dev,
            VL53L0X_SEQUENCESTEP_MSRC, SequenceConfig,
            &pSchedulerSequenceSteps->MsrcOn);
    }

    if (Status == VL53L0X_ERROR_NONE) {
        Status = sequence_step_enabled(Dev,
            VL53L0X_SEQUENCESTEP_PRE_RANGE, SequenceConfig,
            &pSchedulerSequenceSteps->PreRangeOn);
    }
}

```

```

    if (Status == VL53L0X_ERROR_NONE) {

        Status = sequence_step_enabled(Dev,

            VL53L0X_SEQUENCESTEP_FINAL_RANGE, SequenceConfig,

            &pSchedulerSequenceSteps->FinalRangeOn);

    }

    LOG_FUNCTION_END(Status);

    return Status;

}

VL53L0X_Error VL53L0X_GetNumberOfSequenceSteps(VL53L0X_DEV Dev,

    uint8_t *pNumberOfSequenceSteps)

{

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    *pNumberOfSequenceSteps = VL53L0X_SEQUENCESTEP_NUMBER_OF_CHECKS;

    LOG_FUNCTION_END(Status);

    return Status;

}

VL53L0X_Error VL53L0X_GetSequenceStepsInfo(VL53L0X_SequenceStepId SequenceStepId,

    char *pSequenceStepsString)

{

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_get_sequence_steps_info(

```

```

        SequenceStepId,
        pSequenceStepsString);

LOG_FUNCTION_END(Status);

return Status;
}

VL53L0X_Error VL53L0X_SetSequenceStepTimeout(VL53L0X_DEV Dev,
        VL53L0X_SequenceStepId SequenceStepId, FixPoint1616_t TimeOutMilliSecs)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    VL53L0X_Error Status1 = VL53L0X_ERROR_NONE;
    uint32_t TimeoutMicroSeconds = ((TimeOutMilliSecs * 1000) + 0x8000)
        >> 16;
    uint32_t MeasurementTimingBudgetMicroSeconds;
    FixPoint1616_t OldTimeOutMicroSeconds;

    LOG_FUNCTION_START("");

    /* Read back the current value in case we need to revert back to this.
    */
    Status = get_sequence_step_timeout(Dev, SequenceStepId,
        &OldTimeOutMicroSeconds);

    if (Status == VL53L0X_ERROR_NONE) {
        Status = set_sequence_step_timeout(Dev, SequenceStepId,
            TimeoutMicroSeconds);
    }
}

```

```

if (Status == VL53L0X_ERROR_NONE) {
    VL53L0X_GETPARAMETERFIELD(Dev,
        MeasurementTimingBudgetMicroSeconds,
        MeasurementTimingBudgetMicroSeconds);

    /* At this point we don't know if the requested value is valid,
    therefore proceed to update the entire timing budget and
    if this fails, revert back to the previous value.
    */
    Status = VL53L0X_SetMeasurementTimingBudgetMicroSeconds(Dev,
        MeasurementTimingBudgetMicroSeconds);

    if (Status != VL53L0X_ERROR_NONE) {
        Status1 = set_sequence_step_timeout(Dev, SequenceStepId,
            OldTimeOutMicroSeconds);

        if (Status1 == VL53L0X_ERROR_NONE) {
            Status1 =
                VL53L0X_SetMeasurementTimingBudgetMicroSeconds(
                    Dev,
                    MeasurementTimingBudgetMicroSeconds);
        }

        Status = Status1;
    }
}

LOG_FUNCTION_END(Status);

```

```

        return Status;
    }

VL53L0X_Error VL53L0X_GetSequenceStepTimeout(VL53L0X_DEV Dev,
        VL53L0X_SequenceStepId SequenceStepId, FixPoint1616_t *pTimeOutMilliSecs)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint32_t TimeoutMicroSeconds;
    uint32_t WholeNumber_ms = 0;
    uint32_t Fraction_ms = 0;
    LOG_FUNCTION_START("");

    Status = get_sequence_step_timeout(Dev, SequenceStepId,
        &TimeoutMicroSeconds);
    if (Status == VL53L0X_ERROR_NONE) {
        WholeNumber_ms = TimeoutMicroSeconds / 1000;
        Fraction_ms = TimeoutMicroSeconds - (WholeNumber_ms * 1000);
        *pTimeOutMilliSecs = (WholeNumber_ms << 16)
            + (((Fraction_ms * 0xffff) + 500) / 1000);
    }

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_SetInterMeasurementPeriodMilliSeconds(VL53L0X_DEV Dev,
    uint32_t InterMeasurementPeriodMilliSeconds)
{

```

```

VL53L0X_Error Status = VL53L0X_ERROR_NONE;

uint16_t osc_calibrate_val;

uint32_t IMPeriodMilliseconds;

LOG_FUNCTION_START("");

Status = VL53L0X_RdWord(Dev, VL53L0X_REG_OSC_CALIBRATE_VAL,
                        &osc_calibrate_val);

if (Status == VL53L0X_ERROR_NONE) {
    if (osc_calibrate_val != 0) {
        IMPeriodMilliseconds =
            InterMeasurementPeriodMilliseconds
                * osc_calibrate_val;
    } else {
        IMPeriodMilliseconds =
            InterMeasurementPeriodMilliseconds;
    }
    Status = VL53L0X_WrDWord(Dev,
        VL53L0X_REG_SYSTEM_INTERMEASUREMENT_PERIOD,
        IMPeriodMilliseconds);
}

if (Status == VL53L0X_ERROR_NONE) {
    VL53L0X_SETPARAMETERFIELD(Dev,
        InterMeasurementPeriodMilliseconds,
        InterMeasurementPeriodMilliseconds);
}

```

```

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_GetInterMeasurementPeriodMilliseconds(VL53L0X_DEV Dev,
    uint32_t *pInterMeasurementPeriodMilliseconds)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint16_t osc_calibrate_val;
    uint32_t IMPeriodMilliseconds;

    LOG_FUNCTION_START("");

    Status = VL53L0X_RdWord(Dev, VL53L0X_REG_OSC_CALIBRATE_VAL,
        &osc_calibrate_val);

    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_RdDWord(Dev,
            VL53L0X_REG_SYSTEM_INTERMEASUREMENT_PERIOD,
            &IMPeriodMilliseconds);
    }

    if (Status == VL53L0X_ERROR_NONE) {
        if (osc_calibrate_val != 0) {
            *pInterMeasurementPeriodMilliseconds =
                IMPeriodMilliseconds / osc_calibrate_val;
        }

        VL53L0X_SETPARAMETERFIELD(Dev,
            InterMeasurementPeriodMilliseconds,

```

```

        *pInterMeasurementPeriodMilliseconds);

    }

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_SetXTalkCompensationEnable(VL53L0X_DEV Dev,
    uint8_t XTalkCompensationEnable)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    FixPoint1616_t TempFix1616;

    uint16_t LinearityCorrectiveGain;

    LOG_FUNCTION_START("");

    LinearityCorrectiveGain = PALDevDataGet(Dev, LinearityCorrectiveGain);

    if ((XTalkCompensationEnable == 0)
        || (LinearityCorrectiveGain != 1000)) {
        TempFix1616 = 0;
    } else {
        VL53L0X_GETPARAMETERFIELD(Dev, XTalkCompensationRateMegaCps,
            TempFix1616);
    }

    /* the following register has a format 3.13 */
    Status = VL53L0X_WrWord(Dev,
        VL53L0X_REG_CROSSTALK_COMPENSATION_PEAK_RATE_MCPS,

```

```

        VL53L0X_FIXPOINT1616TOFIXPOINT313(TempFix1616));

    if (Status == VL53L0X_ERROR_NONE) {
        if (XTalkCompensationEnable == 0) {
            VL53L0X_SETPARAMETERFIELD(Dev, XTalkCompensationEnable,
                                      0);
        } else {
            VL53L0X_SETPARAMETERFIELD(Dev, XTalkCompensationEnable,
                                      1);
        }
    }

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_GetXTalkCompensationEnable(VL53L0X_DEV Dev,
                                                  uint8_t *pXTalkCompensationEnable)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t Temp8;
    LOG_FUNCTION_START("");

    VL53L0X_GETPARAMETERFIELD(Dev, XTalkCompensationEnable, Temp8);
    *pXTalkCompensationEnable = Temp8;

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_SetXTalkCompensationRateMegaCps(VL53L0X_DEV Dev,
    FixPoint1616_t XTalkCompensationRateMegaCps)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t Temp8;
    uint16_t LinearityCorrectiveGain;
    uint16_t data;
    LOG_FUNCTION_START("");

    VL53L0X_GETPARAMETERFIELD(Dev, XTalkCompensationEnable, Temp8);
    LinearityCorrectiveGain = PALDevDataGet(Dev, LinearityCorrectiveGain);

    if (Temp8 == 0) { /* disabled write only internal value */
        VL53L0X_SETPARAMETERFIELD(Dev, XTalkCompensationRateMegaCps,
            XTalkCompensationRateMegaCps);
    } else {
        /* the following register has a format 3.13 */
        if (LinearityCorrectiveGain == 1000) {
            data = VL53L0X_FIXPOINT1616TOFIXPOINT313(
                XTalkCompensationRateMegaCps);
        } else {
            data = 0;
        }

        Status = VL53L0X_WrWord(Dev,
            VL53L0X_REG_CROSSTALK_COMPENSATION_PEAK_RATE_MCPS, data);

        if (Status == VL53L0X_ERROR_NONE) {

```

```

        VL53L0X_SETPARAMETERFIELD(Dev,
                                    XTalkCompensationRateMegaCps,
                                    XTalkCompensationRateMegaCps);
    }
}

LOG_FUNCTION_END(Status);

return Status;
}

VL53L0X_Error VL53L0X_GetXTalkCompensationRateMegaCps(VL53L0X_DEV Dev,
                                                         FixPoint1616_t *pXTalkCompensationRateMegaCps)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint16_t Value;
    FixPoint1616_t TempFix1616;

    LOG_FUNCTION_START("");

    Status = VL53L0X_RdWord(Dev,
                             VL53L0X_REG_CROSSTALK_COMPENSATION_PEAK_RATE_MCPS, (uint16_t *)&Value);
    if (Status == VL53L0X_ERROR_NONE) {
        if (Value == 0) {
            /* the Xtalk is disabled return value from memory */
            VL53L0X_GETPARAMETERFIELD(Dev,
                                       XTalkCompensationRateMegaCps, TempFix1616);
            *pXTalkCompensationRateMegaCps = TempFix1616;
            VL53L0X_SETPARAMETERFIELD(Dev, XTalkCompensationEnable,
                                       0);

```

```

        } else {
            TempFix1616 = VL53L0X_FIXPOINT313TOFIXPOINT1616(Value);
            *pXTalkCompensationRateMegaCps = TempFix1616;
            VL53L0X_SETPARAMETERFIELD(Dev,
                XTalkCompensationRateMegaCps, TempFix1616);
            VL53L0X_SETPARAMETERFIELD(Dev, XTalkCompensationEnable,
                1);
        }
    }

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_SetRefCalibration(VL53L0X_DEV Dev, uint8_t VhvSettings,
    uint8_t PhaseCal)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    LOG_FUNCTION_START("");

    Status = VL53L0X_set_ref_calibration(Dev, VhvSettings, PhaseCal);

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetRefCalibration(VL53L0X_DEV Dev, uint8_t *pVhvSettings,
    uint8_t *pPhaseCal)
{

```

```

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_get_ref_calibration(Dev, pVhvSettings, pPhaseCal);

    LOG_FUNCTION_END(Status);

    return Status;
}

/*
 * CHECK LIMIT FUNCTIONS
 */

VL53L0X_Error VL53L0X_GetNumberOfLimitCheck(uint16_t *pNumberOfLimitCheck)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    *pNumberOfLimitCheck = VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS;

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_GetLimitCheckInfo(VL53L0X_DEV Dev, uint16_t LimitCheckId,
    char *pLimitCheckString)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

```

```

LOG_FUNCTION_START("");

Status = VL53L0X_get_limit_check_info(Dev, LimitCheckId,
    pLimitCheckString);

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_GetLimitCheckStatus(VL53L0X_DEV Dev, uint16_t LimitCheckId,
    uint8_t *pLimitCheckStatus)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t Temp8;

    LOG_FUNCTION_START("");

    if (LimitCheckId >= VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS) {
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    } else {

        VL53L0X_GETARRAYPARAMETERFIELD(Dev, LimitChecksStatus,
            LimitCheckId, Temp8);

        *pLimitCheckStatus = Temp8;

    }

    LOG_FUNCTION_END(Status);

```

```

        return Status;
    }

VL53L0X_Error VL53L0X_SetLimitCheckEnable(VL53L0X_DEV Dev, uint16_t LimitCheckId,
    uint8_t LimitCheckEnable)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    FixPoint1616_t TempFix1616 = 0;
    uint8_t LimitCheckEnableInt = 0;
    uint8_t LimitCheckDisable = 0;
    uint8_t Temp8;

    LOG_FUNCTION_START("");

    if (LimitCheckId >= VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS) {
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    } else {
        if (LimitCheckEnable == 0) {
            TempFix1616 = 0;
            LimitCheckEnableInt = 0;
            LimitCheckDisable = 1;

        } else {
            VL53L0X_GETARRAYPARAMETERFIELD(Dev, LimitChecksValue,
                LimitCheckId, TempFix1616);

            LimitCheckDisable = 0;
            /* this to be sure to have either 0 or 1 */
            LimitCheckEnableInt = 1;
        }
    }
}

```

```

switch (LimitCheckId) {

case VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE:

    /* internal computation: */

    VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksEnable,

        VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,

        LimitCheckEnableInt);

    break;

case VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE:

    Status = VL53L0X_WrWord(Dev,

        VL53L0X_REG_FINAL_RANGE_CONFIG_MIN_COUNT_RATE_RTN_LIMIT,

        VL53L0X_FIXPOINT1616TOFIXPOINT97(TempFix1616));

    break;

case VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP:

    /* internal computation: */

    VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksEnable,

        VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP,

        LimitCheckEnableInt);

    break;

case VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD:

```

```
/* internal computation: */
```

```
VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksEnable,  
    VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD,  
    LimitCheckEnableInt);
```

```
break;
```

```
case VL53L0X_CHECKENABLE_SIGNAL_RATE_MSRC:
```

```
Temp8 = (uint8_t)(LimitCheckDisable << 1);  
Status = VL53L0X_UpdateByte(Dev,  
    VL53L0X_REG_MSRC_CONFIG_CONTROL,  
    0xFE, Temp8);
```

```
break;
```

```
case VL53L0X_CHECKENABLE_SIGNAL_RATE_PRE_RANGE:
```

```
Temp8 = (uint8_t)(LimitCheckDisable << 4);  
Status = VL53L0X_UpdateByte(Dev,  
    VL53L0X_REG_MSRC_CONFIG_CONTROL,  
    0xEF, Temp8);
```

```
break;
```

```
default:
```

```
Status = VL53L0X_ERROR_INVALID_PARAMS;
```

```

    }

}

if (Status == VL53L0X_ERROR_NONE) {
    if (LimitCheckEnable == 0) {
        VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksEnable,
            LimitCheckId, 0);
    } else {
        VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksEnable,
            LimitCheckId, 1);
    }
}

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_GetLimitCheckEnable(VL53L0X_DEV Dev, uint16_t LimitCheckId,
    uint8_t *pLimitCheckEnable)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t Temp8;

    LOG_FUNCTION_START("");

    if (LimitCheckId >= VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS) {
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
}

```

```

        *pLimitCheckEnable = 0;
    } else {
        VL53L0X_GETARRAYPARAMETERFIELD(Dev, LimitChecksEnable,
            LimitCheckId, Temp8);
        *pLimitCheckEnable = Temp8;
    }

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_SetLimitCheckValue(VL53L0X_DEV Dev, uint16_t LimitCheckId,
    FixPoint1616_t LimitCheckValue)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t Temp8;

    LOG_FUNCTION_START("");

    VL53L0X_GETARRAYPARAMETERFIELD(Dev, LimitChecksEnable, LimitCheckId,
        Temp8);

    if (Temp8 == 0) { /* disabled write only internal value */
        VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksValue,
            LimitCheckId, LimitCheckValue);
    } else {

        switch (LimitCheckId) {

```

```
case VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE:
```

```
    /* internal computation: */
```

```
    VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksValue,  
                                     VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,  
                                     LimitCheckValue);
```

```
    break;
```

```
case VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE:
```

```
    Status = VL53L0X_WrWord(Dev,  
                             VL53L0X_REG_FINAL_RANGE_CONFIG_MIN_COUNT_RATE_RTN_LIMIT,  
                             VL53L0X_FIXPOINT1616TOFIXPOINT97(  
                                 LimitCheckValue));
```

```
    break;
```

```
case VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP:
```

```
    /* internal computation: */
```

```
    VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksValue,  
                                     VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP,  
                                     LimitCheckValue);
```

```
    break;
```

```
case VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD:
```

```
    /* internal computation: */
```

```
    VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksValue,
```

```

        VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD,
        LimitCheckValue);

    break;

case VL53L0X_CHECKENABLE_SIGNAL_RATE_MSRC:
case VL53L0X_CHECKENABLE_SIGNAL_RATE_PRE_RANGE:

    Status = VL53L0X_WrWord(Dev,
        VL53L0X_REG_PRE_RANGE_MIN_COUNT_RATE_RTN_LIMIT,
        VL53L0X_FIXPOINT1616TOFIXPOINT97(
            LimitCheckValue));

    break;

default:
    Status = VL53L0X_ERROR_INVALID_PARAMS;

}

if (Status == VL53L0X_ERROR_NONE) {
    VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksValue,
        LimitCheckId, LimitCheckValue);
}

}

LOG_FUNCTION_END(Status);

return Status;

}

```

```

VL53L0X_Error VL53L0X_GetLimitCheckValue(VL53L0X_DEV Dev, uint16_t LimitCheckId,
    FixPoint1616_t *pLimitCheckValue)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t EnableZeroValue = 0;
    uint16_t Temp16;
    FixPoint1616_t TempFix1616;

    LOG_FUNCTION_START("");

    switch (LimitCheckId) {

    case VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE:
        /* internal computation: */
        VL53L0X_GETARRAYPARAMETERFIELD(Dev, LimitChecksValue,
            VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE, TempFix1616);
        EnableZeroValue = 0;
        break;

    case VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE:
        Status = VL53L0X_RdWord(Dev,
            VL53L0X_REG_FINAL_RANGE_CONFIG_MIN_COUNT_RATE_RTN_LIMIT,
            &Temp16);
        if (Status == VL53L0X_ERROR_NONE)
            TempFix1616 = VL53L0X_FIXPOINT97TOFIXPOINT1616(Temp16);

        EnableZeroValue = 1;
    }
}

```

```
break;
```

```
case VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP:
```

```
    /* internal computation: */
```

```
    VL53L0X_GETARRAYPARAMETERFIELD(Dev, LimitChecksValue,  
                                     VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP, TempFix1616);
```

```
    EnableZeroValue = 0;
```

```
    break;
```

```
case VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD:
```

```
    /* internal computation: */
```

```
    VL53L0X_GETARRAYPARAMETERFIELD(Dev, LimitChecksValue,  
                                     VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD, TempFix1616);
```

```
    EnableZeroValue = 0;
```

```
    break;
```

```
case VL53L0X_CHECKENABLE_SIGNAL_RATE_MSRC:
```

```
case VL53L0X_CHECKENABLE_SIGNAL_RATE_PRE_RANGE:
```

```
    Status = VL53L0X_RdWord(Dev,  
                             VL53L0X_REG_PRE_RANGE_MIN_COUNT_RATE_RTN_LIMIT,  
                             &Temp16);
```

```
    if (Status == VL53L0X_ERROR_NONE)
```

```
        TempFix1616 = VL53L0X_FIXPOINT97TOFIXPOINT1616(Temp16);
```

```
    EnableZeroValue = 0;
```

```
    break;
```

```
default:
```

```

        Status = VL53LOX_ERROR_INVALID_PARAMS;

    }

    if (Status == VL53LOX_ERROR_NONE) {

        if (EnableZeroValue == 1) {

            if (TempFix1616 == 0) {

                /* disabled: return value from memory */
                VL53LOX_GETARRAYPARAMETERFIELD(Dev,
                    LimitChecksValue, LimitCheckId,
                    TempFix1616);

                *pLimitCheckValue = TempFix1616;

                VL53LOX_SETARRAYPARAMETERFIELD(Dev,
                    LimitChecksEnable, LimitCheckId, 0);
            } else {

                *pLimitCheckValue = TempFix1616;

                VL53LOX_SETARRAYPARAMETERFIELD(Dev,
                    LimitChecksValue, LimitCheckId,
                    TempFix1616);

                VL53LOX_SETARRAYPARAMETERFIELD(Dev,
                    LimitChecksEnable, LimitCheckId, 1);
            }
        } else {

            *pLimitCheckValue = TempFix1616;
        }
    }
}

```

```

LOG_FUNCTION_END(Status);

return Status;

}

VL53L0X_Error VL53L0X_GetLimitCheckCurrent(VL53L0X_DEV Dev, uint16_t LimitCheckId,
FixPoint1616_t *pLimitCheckCurrent)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    VL53L0X_RangingMeasurementData_t LastRangeDataBuffer;

    LOG_FUNCTION_START("");

    if (LimitCheckId >= VL53L0X_CHECKENABLE_NUMBER_OF_CHECKS) {
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    } else {
        switch (LimitCheckId) {
            case VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE:
                /* Need to run a ranging to have the latest values */
                *pLimitCheckCurrent = PALDevDataGet(Dev, SigmaEstimate);

                break;

            case VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE:
                /* Need to run a ranging to have the latest values */
                LastRangeDataBuffer = PALDevDataGet(Dev,
                    LastRangeMeasure);
                *pLimitCheckCurrent =
                    LastRangeDataBuffer.SignalRateRtnMegaCps;

```

```
break;
```

```
case VL53LOX_CHECKENABLE_SIGNAL_REF_CLIP:
```

```
/* Need to run a ranging to have the latest values */
```

```
*pLimitCheckCurrent = PALDevDataGet(Dev,  
    LastSignalRefMcps);
```

```
break;
```

```
case VL53LOX_CHECKENABLE_RANGE_IGNORE_THRESHOLD:
```

```
/* Need to run a ranging to have the latest values */
```

```
LastRangeDataBuffer = PALDevDataGet(Dev,  
    LastRangeMeasure);
```

```
*pLimitCheckCurrent =  
    LastRangeDataBuffer.SignalRateRtnMegaCps;
```

```
break;
```

```
case VL53LOX_CHECKENABLE_SIGNAL_RATE_MSRC:
```

```
/* Need to run a ranging to have the latest values */
```

```
LastRangeDataBuffer = PALDevDataGet(Dev,  
    LastRangeMeasure);
```

```
*pLimitCheckCurrent =  
    LastRangeDataBuffer.SignalRateRtnMegaCps;
```

```
break;
```

```
case VL53LOX_CHECKENABLE_SIGNAL_RATE_PRE_RANGE:
```

```

        /* Need to run a ranging to have the latest values */
        LastRangeDataBuffer = PALDevDataGet(Dev,
            LastRangeMeasure);
        *pLimitCheckCurrent =
            LastRangeDataBuffer.SignalRateRtnMegaCps;

        break;

    default:
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
}

LOG_FUNCTION_END(Status);
return Status;

}

/*
 * WRAPAROUND Check
 */
VL53L0X_Error VL53L0X_SetWrapAroundCheckEnable(VL53L0X_DEV Dev,
    uint8_t WrapAroundCheckEnable)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t Byte;
    uint8_t WrapAroundCheckEnableInt;

    LOG_FUNCTION_START("");

```

```

    Status = VL53L0X_RdByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG, &Byte);

    if (WrapAroundCheckEnable == 0) {
        /* Disable wraparound */
        Byte = Byte & 0x7F;
        WrapAroundCheckEnableInt = 0;
    } else {
        /*Enable wraparound */
        Byte = Byte | 0x80;
        WrapAroundCheckEnableInt = 1;
    }

    Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG, Byte);

    if (Status == VL53L0X_ERROR_NONE) {
        PALDevDataSet(Dev, SequenceConfig, Byte);
        VL53L0X_SETPARAMETERFIELD(Dev, WrapAroundCheckEnable,
                                   WrapAroundCheckEnableInt);
    }

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_GetWrapAroundCheckEnable(VL53L0X_DEV Dev,
        uint8_t *pWrapAroundCheckEnable)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t data;

```

```

LOG_FUNCTION_START("");

Status = VL53L0X_RdByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG, &data);
if (Status == VL53L0X_ERROR_NONE) {
    PALDevDataSet(Dev, SequenceConfig, data);
    if (data & (0x01 << 7))
        *pWrapAroundCheckEnable = 0x01;
    else
        *pWrapAroundCheckEnable = 0x00;
}
if (Status == VL53L0X_ERROR_NONE) {
    VL53L0X_SETPARAMETERFIELD(Dev, WrapAroundCheckEnable,
        *pWrapAroundCheckEnable);
}

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_SetDmaxCalParameters(VL53L0X_DEV Dev,
    uint16_t RangeMilliMeter, FixPoint1616_t SignalRateRtnMegaCps)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    FixPoint1616_t SignalRateRtnMegaCpsTemp = 0;

    LOG_FUNCTION_START("");

    /* Check if one of input parameter is zero, in that case the

```

```

    * value are get from NVM */
if ((RangeMilliMeter == 0) || (SignalRateRtnMegaCps == 0)) {
    /* NVM parameters */
    /* Run VL53L0X_get_info_from_device wit option 4 to get
    * signal rate at 400 mm if the value have been already
    * get this function will return with no access to device */
    VL53L0X_get_info_from_device(Dev, 4);

    SignalRateRtnMegaCpsTemp = VL53L0X_GETDEVICESPECIFICPARAMETER(
        Dev, SignalRateMeasFixed400mm);

    PALDevDataSet(Dev, DmaxCalRangeMilliMeter, 400);
    PALDevDataSet(Dev, DmaxCalSignalRateRtnMegaCps,
        SignalRateRtnMegaCpsTemp);
} else {
    /* User parameters */
    PALDevDataSet(Dev, DmaxCalRangeMilliMeter, RangeMilliMeter);
    PALDevDataSet(Dev, DmaxCalSignalRateRtnMegaCps,
        SignalRateRtnMegaCps);
}

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_GetDmaxCalParameters(VL53L0X_DEV Dev,
    uint16_t *pRangeMilliMeter, FixPoint1616_t *pSignalRateRtnMegaCps)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

```

```

LOG_FUNCTION_START("");

*pRangeMilliMeter = PALDevDataGet(Dev, DmaxCalRangeMilliMeter);
*pSignalRateRtnMegaCps = PALDevDataGet(Dev,
    DmaxCalSignalRateRtnMegaCps);

LOG_FUNCTION_END(Status);
return Status;
}

/* End Group PAL Parameters Functions */

/* Group PAL Measurement Functions */
VL53L0X_Error VL53L0X_PerformSingleMeasurement(VL53L0X_DEV Dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    VL53L0X_DeviceModes DeviceMode;

    LOG_FUNCTION_START("");

    /* Get Current DeviceMode */
    Status = VL53L0X_GetDeviceMode(Dev, &DeviceMode);

    /* Start immediately to run a single ranging measurement in case of
    * single ranging or single histogram */
    if (Status == VL53L0X_ERROR_NONE
        && DeviceMode == VL53L0X_DEVICEMODE_SINGLE_RANGING)
        Status = VL53L0X_StartMeasurement(Dev);

```

```

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_measurement_poll_for_completion(Dev);

    /* Change PAL State in case of single ranging or single histogram */
    if (Status == VL53L0X_ERROR_NONE
        && DeviceMode == VL53L0X_DEVICEMODE_SINGLE_RANGING)
        PALDevDataSet(Dev, PalState, VL53L0X_STATE_IDLE);

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_PerformSingleHistogramMeasurement(VL53L0X_DEV Dev,
    VL53L0X_HistogramMeasurementData_t *pHistogramMeasurementData)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NOT_IMPLEMENTED;
    LOG_FUNCTION_START("");

    /* not implemented on VL53L0X */

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_PerformRefCalibration(VL53L0X_DEV Dev, uint8_t *pVhvSettings,

```

```

uint8_t *pPhaseCal)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_perform_ref_calibration(Dev, pVhvSettings,
        pPhaseCal, 1);

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_PerformXTalkMeasurement(VL53L0X_DEV Dev,
    uint32_t TimeoutMs, FixPoint1616_t *pXtalkPerSpad,
    uint8_t *pAmbientTooHigh)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NOT_IMPLEMENTED;

    LOG_FUNCTION_START("");

    /* not implemented on VL53L0X */

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_PerformXTalkCalibration(VL53L0X_DEV Dev,
    FixPoint1616_t XTalkCalDistance,
    FixPoint1616_t *pXTalkCompensationRateMegaCps)
{

```

```

VL53L0X_Error Status = VL53L0X_ERROR_NONE;

LOG_FUNCTION_START("");

Status = VL53L0X_perform_xtalk_calibration(Dev, XTalkCalDistance,
                                           pXTalkCompensationRateMegaCps);

LOG_FUNCTION_END(Status);

return Status;
}

VL53L0X_Error VL53L0X_PerformOffsetCalibration(VL53L0X_DEV Dev,
                                                FixPoint1616_t CalDistanceMilliMeter, int32_t *pOffsetMicroMeter)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_perform_offset_calibration(Dev, CalDistanceMilliMeter,
                                                pOffsetMicroMeter);

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_CheckAndLoadInterruptSettings(VL53L0X_DEV Dev,
                                                      uint8_t StartNotStopFlag)
{
    uint8_t InterruptConfig;
    FixPoint1616_t ThresholdLow;
    FixPoint1616_t ThresholdHigh;

```

```
VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
InterruptConfig = VL53L0X_GETDEVICESPECIFICPARAMETER(Dev,  
    Pin0GpioFunctionality);
```

```
if ((InterruptConfig ==  
    VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_LOW) ||  
    (InterruptConfig ==  
    VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_HIGH) ||  
    (InterruptConfig ==  
    VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_OUT)) {
```

```
    Status = VL53L0X_GetInterruptThresholds(Dev,  
        VL53L0X_DEVICEMODE_CONTINUOUS_RANGING,  
        &ThresholdLow, &ThresholdHigh);
```

```
    if (((ThresholdLow > 255*65536) ||  
        (ThresholdHigh > 255*65536)) &&  
        (Status == VL53L0X_ERROR_NONE)) {
```

```
        if (StartNotStopFlag != 0) {  
            Status = VL53L0X_load_tuning_settings(Dev,  
                InterruptThresholdSettings);
```

```
        } else {  
            Status |= VL53L0X_WrByte(Dev, 0xFF, 0x04);  
            Status |= VL53L0X_WrByte(Dev, 0x70, 0x00);  
            Status |= VL53L0X_WrByte(Dev, 0xFF, 0x00);  
            Status |= VL53L0X_WrByte(Dev, 0x80, 0x00);
```

```
        }
```

```

    }

}

return Status;

}

VL53L0X_Error VL53L0X_StartMeasurement(VL53L0X_DEV Dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    VL53L0X_DeviceModes DeviceMode;
    uint8_t Byte;
    uint8_t StartStopByte = VL53L0X_REG_SYSRANGE_MODE_START_STOP;
    uint32_t LoopNb;
    LOG_FUNCTION_START("");

    /* Get Current DeviceMode */
    VL53L0X_GetDeviceMode(Dev, &DeviceMode);

    Status = VL53L0X_WrByte(Dev, 0x80, 0x01);
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x01);
    Status = VL53L0X_WrByte(Dev, 0x00, 0x00);
    Status = VL53L0X_WrByte(Dev, 0x91, PALDevDataGet(Dev, StopVariable));
    Status = VL53L0X_WrByte(Dev, 0x00, 0x01);
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x00);

```

```

Status = VL53L0X_WrByte(Dev, 0x80, 0x00);

switch (DeviceMode) {
case VL53L0X_DEVICEMODE_SINGLE_RANGING:

    Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSRANGE_START, 0x01);

    Byte = StartStopByte;

    if (Status == VL53L0X_ERROR_NONE) {

        /* Wait until start bit has been cleared */

        LoopNb = 0;

        do {

            if (LoopNb > 0)

                Status = VL53L0X_RdByte(Dev,

                    VL53L0X_REG_SYSRANGE_START, &Byte);

            LoopNb = LoopNb + 1;

        } while (((Byte & StartStopByte) == StartStopByte)

            && (Status == VL53L0X_ERROR_NONE)

            && (LoopNb < VL53L0X_DEFAULT_MAX_LOOP));

        if (LoopNb >= VL53L0X_DEFAULT_MAX_LOOP)

            Status = VL53L0X_ERROR_TIME_OUT;

    }

    break;
case VL53L0X_DEVICEMODE_CONTINUOUS_RANGING:

    /* Back-to-back mode */

    /* Check if need to apply interrupt settings */

```

```

    if (Status == VL53L0X_ERROR_NONE)

        Status = VL53L0X_CheckAndLoadInterruptSettings(Dev, 1);

    Status = VL53L0X_WrByte(Dev,
    VL53L0X_REG_SYSRANGE_START,
    VL53L0X_REG_SYSRANGE_MODE_BACKTOBACK);
    if (Status == VL53L0X_ERROR_NONE) {
        /* Set PAL State to Running */
        PALDevDataSet(Dev, PalState, VL53L0X_STATE_RUNNING);
    }
    break;
case VL53L0X_DEVICEMODE_CONTINUOUS_TIMED_RANGING:
    /* Continuous mode */
    /* Check if need to apply interrupt settings */
    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_CheckAndLoadInterruptSettings(Dev, 1);

    Status = VL53L0X_WrByte(Dev,
    VL53L0X_REG_SYSRANGE_START,
    VL53L0X_REG_SYSRANGE_MODE_TIMED);

    if (Status == VL53L0X_ERROR_NONE) {
        /* Set PAL State to Running */
        PALDevDataSet(Dev, PalState, VL53L0X_STATE_RUNNING);
    }
    break;
default:
    /* Selected mode not supported */
    Status = VL53L0X_ERROR_MODE_NOT_SUPPORTED;

```

```
}
```

```
LOG_FUNCTION_END(Status);
```

```
return Status;
```

```
}
```

```
VL53L0X_Error VL53L0X_StopMeasurement(VL53L0X_DEV Dev)
```

```
{
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    LOG_FUNCTION_START("");
```

```
    Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSRANGE_START,  
VL53L0X_REG_SYSRANGE_MODE_SINGLESHOT);
```

```
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x01);
```

```
    Status = VL53L0X_WrByte(Dev, 0x00, 0x00);
```

```
    Status = VL53L0X_WrByte(Dev, 0x91, 0x00);
```

```
    Status = VL53L0X_WrByte(Dev, 0x00, 0x01);
```

```
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x00);
```

```
    if (Status == VL53L0X_ERROR_NONE) {
```

```
        /* Set PAL State to Idle */
```

```
        PALDevDataSet(Dev, PalState, VL53L0X_STATE_IDLE);
```

```
    }
```

```
    /* Check if need to apply interrupt settings */
```

```
    if (Status == VL53L0X_ERROR_NONE)
```

```
        Status = VL53L0X_CheckAndLoadInterruptSettings(Dev, 0);
```

```

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_GetMeasurementDataReady(VL53L0X_DEV Dev,
    uint8_t *pMeasurementDataReady)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t SysRangeStatusRegister;

    uint8_t InterruptConfig;

    uint32_t InterruptMask;

    LOG_FUNCTION_START("");

    InterruptConfig = VL53L0X_GETDEVICESPECIFICPARAMETER(Dev,
        Pin0GpioFunctionality);

    if (InterruptConfig ==
        VL53L0X_REG_SYSTEM_INTERRUPT_GPIO_NEW_SAMPLE_READY) {
        Status = VL53L0X_GetInterruptMaskStatus(Dev, &InterruptMask);

        if (InterruptMask ==
            VL53L0X_REG_SYSTEM_INTERRUPT_GPIO_NEW_SAMPLE_READY)
            *pMeasurementDataReady = 1;
        else
            *pMeasurementDataReady = 0;
    } else {
        Status = VL53L0X_RdByte(Dev, VL53L0X_REG_RESULT_RANGE_STATUS,
            &SysRangeStatusRegister);

        if (Status == VL53L0X_ERROR_NONE) {

```

```

        if (SysRangeStatusRegister & 0x01)
            *pMeasurementDataReady = 1;
        else
            *pMeasurementDataReady = 0;
    }
}

LOG_FUNCTION_END(Status);
return Status;
}

```

```

VL53L0X_Error VL53L0X_WaitDeviceReadyForNewMeasurement(VL53L0X_DEV Dev,
    uint32_t MaxLoop)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NOT_IMPLEMENTED;
    LOG_FUNCTION_START("");

    /* not implemented for VL53L0X */

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetRangingMeasurementData(VL53L0X_DEV Dev,
    VL53L0X_RangingMeasurementData_t *pRangingMeasurementData)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t DeviceRangeStatus;

```

```

uint8_t RangeFractionalEnable;

uint8_t PalRangeStatus;

uint8_t XTalkCompensationEnable;

uint16_t AmbientRate;

FixPoint1616_t SignalRate;

uint16_t XTalkCompensationRateMegaCps;

uint16_t EffectiveSpadRtnCount;

uint16_t tmpuint16;

uint16_t XtalkRangeMilliMeter;

uint16_t LinearityCorrectiveGain;

uint8_t localBuffer[12];

VL53L0X_RangingMeasurementData_t LastRangeDataBuffer;


LOG_FUNCTION_START("");


/*
 * use multi read even if some registers are not useful, result will
 * be more efficient
 * start reading at 0x14 dec20
 * end reading at 0x21 dec33 total 14 bytes to read
 */
Status = VL53L0X_ReadMulti(Dev, 0x14, localBuffer, 12);


if (Status == VL53L0X_ERROR_NONE) {

    pRangingMeasurementData->ZoneId = 0; /* Only one zone */

    pRangingMeasurementData->TimeStamp = 0; /* Not Implemented */


    tmpuint16 = VL53L0X_MAKEUINT16(localBuffer[11], localBuffer[10]);

```

```

/* cut1.1 if SYSTEM__RANGE_CONFIG if 1 range is 2bits fractional
*(format 11.2) else no fractional
*/

pRangingMeasurementData->MeasurementTimeUsec = 0;

SignalRate = VL53L0X_FIXPOINT97TOFIXPOINT1616(
    VL53L0X_MAKEUINT16(localBuffer[7], localBuffer[6]));
/* peak_signal_count_rate_rtn_mcps */
pRangingMeasurementData->SignalRateRtnMegaCps = SignalRate;

AmbientRate = VL53L0X_MAKEUINT16(localBuffer[9], localBuffer[8]);
pRangingMeasurementData->AmbientRateRtnMegaCps =
    VL53L0X_FIXPOINT97TOFIXPOINT1616(AmbientRate);

EffectiveSpadRtnCount = VL53L0X_MAKEUINT16(localBuffer[3],
    localBuffer[2]);
/* EffectiveSpadRtnCount is 8.8 format */
pRangingMeasurementData->EffectiveSpadRtnCount =
    EffectiveSpadRtnCount;

DeviceRangeStatus = localBuffer[0];

/* Get Linearity Corrective Gain */
LinearityCorrectiveGain = PALDevDataGet(Dev,
    LinearityCorrectiveGain);

/* Get ranging configuration */
RangeFractionalEnable = PALDevDataGet(Dev,

```

```
RangeFractionalEnable);
```

```
if (LinearityCorrectiveGain != 1000) {
```

```
    tmpuint16 = (uint16_t)((LinearityCorrectiveGain  
        * tmpuint16 + 500) / 1000);
```

```
/* Implement Xtalk */
```

```
VL53L0X_GETPARAMETERFIELD(Dev,  
    XTalkCompensationRateMegaCps,  
    XTalkCompensationRateMegaCps);  
VL53L0X_GETPARAMETERFIELD(Dev, XTalkCompensationEnable,  
    XTalkCompensationEnable);
```

```
if (XTalkCompensationEnable) {
```

```
    if ((SignalRate  
        - ((XTalkCompensationRateMegaCps  
            * EffectiveSpadRtnCount) >> 8))  
        <= 0) {  
        if (RangeFractionalEnable)  
            XtalkRangeMilliMeter = 8888;  
        else  
            XtalkRangeMilliMeter = 8888  
                << 2;  
    } else {  
        XtalkRangeMilliMeter =  
            (tmpuint16 * SignalRate)  
                / (SignalRate
```

```

        - ((XTalkCompensationRateMegaCps
        * EffectiveSpadRtnCount)
        >> 8));
    }

    tmpuint16 = XtalkRangeMilliMeter;
}

}

if (RangeFractionalEnable) {
    pRangingMeasurementData->RangeMilliMeter =
        (uint16_t)((tmpuint16) >> 2);
    pRangingMeasurementData->RangeFractionalPart =
        (uint8_t)((tmpuint16 & 0x03) << 6);
} else {
    pRangingMeasurementData->RangeMilliMeter = tmpuint16;
    pRangingMeasurementData->RangeFractionalPart = 0;
}

/*
 * For a standard definition of RangeStatus, this should
 * return 0 in case of good result after a ranging
 * The range status depends on the device so call a device
 * specific function to obtain the right Status.
 */
Status |= VL53LOX_get_pal_range_status(Dev, DeviceRangeStatus,
    SignalRate, EffectiveSpadRtnCount,
    pRangingMeasurementData, &PalRangeStatus);

```

```

        if (Status == VL53LOX_ERROR_NONE)
            pRangingMeasurementData->RangeStatus = PalRangeStatus;

    }

```

```

if (Status == VL53LOX_ERROR_NONE) {
    /* Copy last read data into Dev buffer */
    LastRangeDataBuffer = PALDevDataGet(Dev, LastRangeMeasure);

    LastRangeDataBuffer.RangeMilliMeter =
        pRangingMeasurementData->RangeMilliMeter;
    LastRangeDataBuffer.RangeFractionalPart =
        pRangingMeasurementData->RangeFractionalPart;
    LastRangeDataBuffer.RangeDMaxMilliMeter =
        pRangingMeasurementData->RangeDMaxMilliMeter;
    LastRangeDataBuffer.MeasurementTimeUsec =
        pRangingMeasurementData->MeasurementTimeUsec;
    LastRangeDataBuffer.SignalRateRtnMegaCps =
        pRangingMeasurementData->SignalRateRtnMegaCps;
    LastRangeDataBuffer.AmbientRateRtnMegaCps =
        pRangingMeasurementData->AmbientRateRtnMegaCps;
    LastRangeDataBuffer.EffectiveSpadRtnCount =
        pRangingMeasurementData->EffectiveSpadRtnCount;
    LastRangeDataBuffer.RangeStatus =
        pRangingMeasurementData->RangeStatus;

    PALDevDataSet(Dev, LastRangeMeasure, LastRangeDataBuffer);
}

```

```
LOG_FUNCTION_END(Status);  
return Status;  
}
```

```
VL53L0X_Error VL53L0X_GetMeasurementRefSignal(VL53L0X_DEV Dev,  
        FixPoint1616_t *pMeasurementRefSignal)  
{  
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;  
    LOG_FUNCTION_START("");  
  
    *pMeasurementRefSignal = PALDevDataGet(Dev, LastSignalRefMcps);  
  
    LOG_FUNCTION_END(Status);  
    return Status;  
}
```

```
VL53L0X_Error VL53L0X_GetHistogramMeasurementData(VL53L0X_DEV Dev,  
        VL53L0X_HistogramMeasurementData_t *pHistogramMeasurementData)  
{  
    VL53L0X_Error Status = VL53L0X_ERROR_NOT_IMPLEMENTED;  
    LOG_FUNCTION_START("");  
  
    LOG_FUNCTION_END(Status);  
    return Status;  
}
```

```
VL53L0X_Error VL53L0X_PerformSingleRangingMeasurement(VL53L0X_DEV Dev,
```

```

VL53L0X_RangingMeasurementData_t *pRangingMeasurementData)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    /* This function will do a complete single ranging
    * Here we fix the mode! */
    Status = VL53L0X_SetDeviceMode(Dev, VL53L0X_DEVICEMODE_SINGLE_RANGING);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_PerformSingleMeasurement(Dev);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_GetRangingMeasurementData(Dev,
            pRangingMeasurementData);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_ClearInterruptMask(Dev, 0);

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_SetNumberOfROIzones(VL53L0X_DEV Dev,
    uint8_t NumberOfROIzones)
{

```

```

VL53L0X_Error Status = VL53L0X_ERROR_NONE;

LOG_FUNCTION_START("");

if (NumberOfROIzones != 1)
    Status = VL53L0X_ERROR_INVALID_PARAMS;

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_GetNumberOfROIzones(VL53L0X_DEV Dev,
    uint8_t *pNumberOfROIzones)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    *pNumberOfROIzones = 1;

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_GetMaxNumberOfROIzones(VL53L0X_DEV Dev,
    uint8_t *pMaxNumberOfROIzones)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

```

```

LOG_FUNCTION_START("");

    *pMaxNumberOfROIzones = 1;

LOG_FUNCTION_END(Status);

return Status;

}

/* End Group PAL Measurement Functions */

VL53L0X_Error VL53L0X_SetGpioConfig(VL53L0X_DEV Dev, uint8_t Pin,
    VL53L0X_DeviceModes DeviceMode, VL53L0X_GpioFunctionality Functionality,
    VL53L0X_InterruptPolarity Polarity)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t data;

    LOG_FUNCTION_START("");

    if (Pin != 0) {
        Status = VL53L0X_ERROR_GPIO_NOT_EXISTING;
    } else if (DeviceMode == VL53L0X_DEVICEMODE_GPIO_DRIVE) {
        if (Polarity == VL53L0X_INTERRUPTPOLARITY_LOW)
            data = 0x10;
        else
            data = 1;

        Status = VL53L0X_WrByte(Dev,

```

```

        VL53L0X_REG_GPIO_HV_MUX_ACTIVE_HIGH, data);

    } else if (DeviceMode == VL53L0X_DEVICEMODE_GPIO_OSC) {

        Status |= VL53L0X_WrByte(Dev, 0xff, 0x01);
        Status |= VL53L0X_WrByte(Dev, 0x00, 0x00);

        Status |= VL53L0X_WrByte(Dev, 0xff, 0x00);
        Status |= VL53L0X_WrByte(Dev, 0x80, 0x01);
        Status |= VL53L0X_WrByte(Dev, 0x85, 0x02);

        Status |= VL53L0X_WrByte(Dev, 0xff, 0x04);
        Status |= VL53L0X_WrByte(Dev, 0xcd, 0x00);
        Status |= VL53L0X_WrByte(Dev, 0xcc, 0x11);

        Status |= VL53L0X_WrByte(Dev, 0xff, 0x07);
        Status |= VL53L0X_WrByte(Dev, 0xbe, 0x00);

        Status |= VL53L0X_WrByte(Dev, 0xff, 0x06);
        Status |= VL53L0X_WrByte(Dev, 0xcc, 0x09);

        Status |= VL53L0X_WrByte(Dev, 0xff, 0x00);
        Status |= VL53L0X_WrByte(Dev, 0xff, 0x01);
        Status |= VL53L0X_WrByte(Dev, 0x00, 0x00);

    } else {

        if (Status == VL53L0X_ERROR_NONE) {
            switch (Functionality) {

```

```

case VL53L0X_GPIOFUNCTIONALITY_OFF:

    data = 0x00;

    break;

case VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_LOW:

    data = 0x01;

    break;

case VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_HIGH:

    data = 0x02;

    break;

case VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_OUT:

    data = 0x03;

    break;

case VL53L0X_GPIOFUNCTIONALITY_NEW_MEASURE_READY:

    data = 0x04;

    break;

default:

    Status =

        VL53L0X_ERROR_GPIO_FUNCTIONALITY_NOT_SUPPORTED;

}

}

```

```

if (Status == VL53L0X_ERROR_NONE)

    Status = VL53L0X_WrByte(Dev,

        VL53L0X_REG_SYSTEM_INTERRUPT_CONFIG_GPIO, data);

```

```

if (Status == VL53L0X_ERROR_NONE) {

    if (Polarity == VL53L0X_INTERRUPTPOLARITY_LOW)

        data = 0;

    else

```

```

        data = (uint8_t)(1 << 4);

        Status = VL53L0X_UpdateByte(Dev,
        VL53L0X_REG_GPIO_HV_MUX_ACTIVE_HIGH, 0xEF, data);
    }

    if (Status == VL53L0X_ERROR_NONE)
        VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
        Pin0GpioFunctionality, Functionality);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_ClearInterruptMask(Dev, 0);

}

LOG_FUNCTION_END(Status);
return Status;
}

VL53L0X_Error VL53L0X_GetGpioConfig(VL53L0X_DEV Dev, uint8_t Pin,
    VL53L0X_DeviceModes *pDeviceMode,
    VL53L0X_GpioFunctionality *pFunctionality,
    VL53L0X_InterruptPolarity *pPolarity)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    VL53L0X_GpioFunctionality GpioFunctionality;
    uint8_t data;

    LOG_FUNCTION_START("");

```

```
/* pDeviceMode not managed by Ewok it return the current mode */
```

```
Status = VL53L0X_GetDeviceMode(Dev, pDeviceMode);
```

```
if (Status == VL53L0X_ERROR_NONE) {
```

```
    if (Pin != 0) {
```

```
        Status = VL53L0X_ERROR_GPIO_NOT_EXISTING;
```

```
    } else {
```

```
        Status = VL53L0X_RdByte(Dev,
```

```
        VL53L0X_REG_SYSTEM_INTERRUPT_CONFIG_GPIO, &data);
```

```
    }
```

```
}
```

```
if (Status == VL53L0X_ERROR_NONE) {
```

```
    switch (data & 0x07) {
```

```
    case 0x00:
```

```
        GpioFunctionality = VL53L0X_GPIOFUNCTIONALITY_OFF;
```

```
        break;
```

```
    case 0x01:
```

```
        GpioFunctionality =
```

```
        VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_LOW;
```

```
        break;
```

```
    case 0x02:
```

```
        GpioFunctionality =
```

```
        VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_HIGH;
```

```
        break;
```

```
    case 0x03:
```

```
        GpioFunctionality =
```

```

        VL53L0X_GPIOFUNCTIONALITY_THRESHOLD_CROSSED_OUT;

        break;

    case 0x04:

        GpioFunctionality =

        VL53L0X_GPIOFUNCTIONALITY_NEW_MEASURE_READY;

        break;

    default:

        Status = VL53L0X_ERROR_GPIO_FUNCTIONALITY_NOT_SUPPORTED;

    }

}

```

```

if (Status == VL53L0X_ERROR_NONE)

    Status = VL53L0X_RdByte(Dev, VL53L0X_REG_GPIO_HV_MUX_ACTIVE_HIGH,

        &data);

```

```

if (Status == VL53L0X_ERROR_NONE) {

    if ((data & (uint8_t)(1 << 4)) == 0)

        *pPolarity = VL53L0X_INTERRUPTPOLARITY_LOW;

    else

        *pPolarity = VL53L0X_INTERRUPTPOLARITY_HIGH;

}

```

```

if (Status == VL53L0X_ERROR_NONE) {

    *pFunctionality = GpioFunctionality;

    VL53L0X_SETDEVICESPECIFICPARAMETER(Dev, Pin0GpioFunctionality,

        GpioFunctionality);

}

```

```

LOG_FUNCTION_END(Status);

```

```

        return Status;
    }

VL53L0X_Error VL53L0X_SetInterruptThresholds(VL53L0X_DEV Dev,
        VL53L0X_DeviceModes DeviceMode, FixPoint1616_t ThresholdLow,
        FixPoint1616_t ThresholdHigh)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint16_t Threshold16;
    LOG_FUNCTION_START("");

    /* no dependency on DeviceMode for Ewok */
    /* Need to divide by 2 because the FW will apply a x2 */
    Threshold16 = (uint16_t)((ThresholdLow >> 17) & 0x00fff);
    Status = VL53L0X_WrWord(Dev, VL53L0X_REG_SYSTEM_THRESH_LOW, Threshold16);

    if (Status == VL53L0X_ERROR_NONE) {
        /* Need to divide by 2 because the FW will apply a x2 */
        Threshold16 = (uint16_t)((ThresholdHigh >> 17) & 0x00fff);
        Status = VL53L0X_WrWord(Dev, VL53L0X_REG_SYSTEM_THRESH_HIGH,
            Threshold16);
    }

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_GetInterruptThresholds(VL53L0X_DEV Dev,
        VL53L0X_DeviceModes DeviceMode, FixPoint1616_t *pThresholdLow,

```

```

    FixPoint1616_t *pThresholdHigh)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint16_t Threshold16;

    LOG_FUNCTION_START("");

    /* no dependency on DeviceMode for Ewok */

    Status = VL53L0X_RdWord(Dev, VL53L0X_REG_SYSTEM_THRESH_LOW, &Threshold16);

    /* Need to multiply by 2 because the FW will apply a x2 */
    *pThresholdLow = (FixPoint1616_t)((0x00fff & Threshold16) << 17);

    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_RdWord(Dev, VL53L0X_REG_SYSTEM_THRESH_HIGH,
                                &Threshold16);

        /* Need to multiply by 2 because the FW will apply a x2 */
        *pThresholdHigh =
            (FixPoint1616_t)((0x00fff & Threshold16) << 17);
    }

    LOG_FUNCTION_END(Status);

    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetStopCompletedStatus(VL53L0X_DEV Dev,
    uint32_t *pStopStatus)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t Byte = 0;

```

```

LOG_FUNCTION_START("");

Status = VL53L0X_WrByte(Drv, 0xFF, 0x01);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_RdByte(Drv, 0x04, &Byte);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Drv, 0xFF, 0x0);

*StopStatus = Byte;

if (Byte == 0) {
    Status = VL53L0X_WrByte(Drv, 0x80, 0x01);
    Status = VL53L0X_WrByte(Drv, 0xFF, 0x01);
    Status = VL53L0X_WrByte(Drv, 0x00, 0x00);
    Status = VL53L0X_WrByte(Drv, 0x91,
        PALDevDataGet(Drv, StopVariable));
    Status = VL53L0X_WrByte(Drv, 0x00, 0x01);
    Status = VL53L0X_WrByte(Drv, 0xFF, 0x00);
    Status = VL53L0X_WrByte(Drv, 0x80, 0x00);
}

LOG_FUNCTION_END(Status);

return Status;
}

/* Group PAL Interrupt Functions */
VL53L0X_Error VL53L0X_ClearInterruptMask(VL53L0X_DEV Drv, uint32_t InterruptMask)

```

```

{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t LoopCount;

    uint8_t Byte;

    LOG_FUNCTION_START("");

    /* clear bit 0 range interrupt, bit 1 error interrupt */
    LoopCount = 0;
    do {
        Status = VL53L0X_WrByte(Dev,
                                VL53L0X_REG_SYSTEM_INTERRUPT_CLEAR, 0x01);
        Status |= VL53L0X_WrByte(Dev,
                                VL53L0X_REG_SYSTEM_INTERRUPT_CLEAR, 0x00);
        Status |= VL53L0X_RdByte(Dev,
                                VL53L0X_REG_RESULT_INTERRUPT_STATUS, &Byte);
        LoopCount++;
    } while (((Byte & 0x07) != 0x00)
            && (LoopCount < 3)
            && (Status == VL53L0X_ERROR_NONE));

    if (LoopCount >= 3)
        Status = VL53L0X_ERROR_INTERRUPT_NOT_CLEARED;

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetInterruptMaskStatus(VL53L0X_DEV Dev,

```

```

uint32_t *pInterruptMaskStatus)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t Byte;

    LOG_FUNCTION_START("");

    Status = VL53L0X_RdByte(Dev, VL53L0X_REG_RESULT_INTERRUPT_STATUS, &Byte);

    *pInterruptMaskStatus = Byte & 0x07;

    if (Byte & 0x18)
        Status = VL53L0X_ERROR_RANGE_ERROR;

    LOG_FUNCTION_END(Status);

    return Status;
}

```

```

VL53L0X_Error VL53L0X_EnableInterruptMask(VL53L0X_DEV Dev, uint32_t InterruptMask)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NOT_IMPLEMENTED;

    LOG_FUNCTION_START("");

    /* not implemented for VL53L0X */

    LOG_FUNCTION_END(Status);

    return Status;
}

```

```

/* End Group PAL Interrupt Functions */

```

```
/* Group SPAD functions */
```

```
VL53L0X_Error VL53L0X_SetSpadAmbientDamperThreshold(VL53L0X_DEV Dev,  
    uint16_t SpadAmbientDamperThreshold)  
{  
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;  
    LOG_FUNCTION_START("");  
  
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x01);  
    Status |= VL53L0X_WrWord(Dev, 0x40, SpadAmbientDamperThreshold);  
    Status |= VL53L0X_WrByte(Dev, 0xFF, 0x00);  
  
    LOG_FUNCTION_END(Status);  
    return Status;  
}
```

```
VL53L0X_Error VL53L0X_GetSpadAmbientDamperThreshold(VL53L0X_DEV Dev,  
    uint16_t *pSpadAmbientDamperThreshold)  
{  
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;  
    LOG_FUNCTION_START("");  
  
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x01);  
    Status |= VL53L0X_RdWord(Dev, 0x40, pSpadAmbientDamperThreshold);  
    Status |= VL53L0X_WrByte(Dev, 0xFF, 0x00);  
  
    LOG_FUNCTION_END(Status);  
    return Status;  
}
```

```

VL53L0X_Error VL53L0X_SetSpadAmbientDamperFactor(VL53L0X_DEV Dev,
    uint16_t SpadAmbientDamperFactor)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t Byte;
    LOG_FUNCTION_START("");

    Byte = (uint8_t)(SpadAmbientDamperFactor & 0x00FF);

    Status = VL53L0X_WrByte(Dev, 0xFF, 0x01);
    Status |= VL53L0X_WrByte(Dev, 0x42, Byte);
    Status |= VL53L0X_WrByte(Dev, 0xFF, 0x00);

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_GetSpadAmbientDamperFactor(VL53L0X_DEV Dev,
    uint16_t *pSpadAmbientDamperFactor)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t Byte;
    LOG_FUNCTION_START("");

    Status = VL53L0X_WrByte(Dev, 0xFF, 0x01);
    Status |= VL53L0X_RdByte(Dev, 0x42, &Byte);
    Status |= VL53L0X_WrByte(Dev, 0xFF, 0x00);
    *pSpadAmbientDamperFactor = (uint16_t)Byte;
}

```

```

        LOG_FUNCTION_END(Status);

        return Status;
    }

/* END Group SPAD functions */

/*****
 * Internal functions
 *****/

VL53L0X_Error VL53L0X_SetReferenceSpads(VL53L0X_DEV Dev, uint32_t count,
    uint8_t isApertureSpads)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    Status = VL53L0X_set_reference_spads(Dev, count, isApertureSpads);

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_GetReferenceSpads(VL53L0X_DEV Dev, uint32_t *pSpadCount,
    uint8_t *pIsApertureSpads)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

```

```

        Status = VL53L0X_get_reference_spads(Dev, pSpadCount, plsApertureSpads);

        LOG_FUNCTION_END(Status);

        return Status;
    }

```

```

VL53L0X_Error VL53L0X_PerformRefSpadManagement(VL53L0X_DEV Dev,
        uint32_t *refSpadCount, uint8_t *isApertureSpads)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    LOG_FUNCTION_START("");

    Status = VL53L0X_perform_ref_spad_management(Dev, refSpadCount,
        isApertureSpads);

    LOG_FUNCTION_END(Status);

    return Status;
}

```

1.1.34 vl53l0x_api_strings.h

Below:

```

/*****

```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#ifndef VL53L0X_API_STRINGS_H_
```

```
#define VL53L0X_API_STRINGS_H_
```

```
#include "vl53l0x_def.h"
```

```
#include "vl53l0x_platform.h"
```

```
#ifdef __cplusplus
```

```
extern "C" {  
#endif
```

```
VL53L0X_Error VL53L0X_get_device_info(VL53L0X_DEV Dev,  
                                       VL53L0X_DeviceInfo_t *pVL53L0X_DeviceInfo);
```

```
VL53L0X_Error VL53L0X_get_device_error_string(VL53L0X_DeviceError ErrorCode,  
                                              char *pDeviceErrorString);
```

```
VL53L0X_Error VL53L0X_get_range_status_string(uint8_t RangeStatus,  
                                              char *pRangeStatusString);
```

```
VL53L0X_Error VL53L0X_get_pal_error_string(VL53L0X_Error PalErrorCode,  
                                           char *pPalErrorString);
```

```
VL53L0X_Error VL53L0X_get_pal_state_string(VL53L0X_State PalStateCode,  
                                           char *pPalStateString);
```

```
VL53L0X_Error VL53L0X_get_sequence_steps_info(  
    VL53L0X_SequenceStepId SequenceStepId,  
    char *pSequenceStepsString);
```

```
VL53L0X_Error VL53L0X_get_limit_check_info(VL53L0X_DEV Dev, uint16_t LimitCheckId,  
                                           char *pLimitCheckString);
```

```
#ifdef USE_EMPTY_STRING
```

```
    #define VL53L0X_STRING_DEVICE_INFO_NAME        ""
```

```

#define VL53L0X_STRING_DEVICE_INFO_NAME_TS0      ""
#define VL53L0X_STRING_DEVICE_INFO_NAME_TS1      ""
#define VL53L0X_STRING_DEVICE_INFO_NAME_TS2      ""
#define VL53L0X_STRING_DEVICE_INFO_NAME_ES1      ""
#define VL53L0X_STRING_DEVICE_INFO_TYPE          ""

/* PAL ERROR strings */
#define VL53L0X_STRING_ERROR_NONE                 ""
#define VL53L0X_STRING_ERROR_CALIBRATION_WARNING  ""
#define VL53L0X_STRING_ERROR_MIN_CLIPPED          ""
#define VL53L0X_STRING_ERROR_UNDEFINED            ""
#define VL53L0X_STRING_ERROR_INVALID_PARAMS       ""
#define VL53L0X_STRING_ERROR_NOT_SUPPORTED        ""
#define VL53L0X_STRING_ERROR_RANGE_ERROR          ""
#define VL53L0X_STRING_ERROR_TIME_OUT             ""
#define VL53L0X_STRING_ERROR_MODE_NOT_SUPPORTED   ""
#define VL53L0X_STRING_ERROR_BUFFER_TOO_SMALL     ""
#define VL53L0X_STRING_ERROR_GPIO_NOT_EXISTING    ""
#define VL53L0X_STRING_ERROR_GPIO_FUNCTIONALITY_NOT_SUPPORTED ""
#define VL53L0X_STRING_ERROR_CONTROL_INTERFACE    ""
#define VL53L0X_STRING_ERROR_INVALID_COMMAND      ""
#define VL53L0X_STRING_ERROR_DIVISION_BY_ZERO     ""
#define VL53L0X_STRING_ERROR_REF_SPAD_INIT        ""
#define VL53L0X_STRING_ERROR_NOT_IMPLEMENTED      ""

#define VL53L0X_STRING_UNKNOW_ERROR_CODE          ""

```

/* Range Status */

```
#define VL53L0X_STRING_RANGESTATUS_NONE      ""
#define VL53L0X_STRING_RANGESTATUS_RANGEVALID  ""
#define VL53L0X_STRING_RANGESTATUS_SIGMA      ""
#define VL53L0X_STRING_RANGESTATUS_SIGNAL      ""
#define VL53L0X_STRING_RANGESTATUS_MINRANGE    ""
#define VL53L0X_STRING_RANGESTATUS_PHASE      ""
#define VL53L0X_STRING_RANGESTATUS_HW         ""
```

/* Range Status */

```
#define VL53L0X_STRING_STATE_POWERDOWN      ""
#define VL53L0X_STRING_STATE_WAIT_STATICINIT  ""
#define VL53L0X_STRING_STATE_STANDBY         ""
#define VL53L0X_STRING_STATE_IDLE            ""
#define VL53L0X_STRING_STATE_RUNNING          ""
#define VL53L0X_STRING_STATE_UNKNOWN          ""
#define VL53L0X_STRING_STATE_ERROR            ""
```

/* Device Specific */

```
#define VL53L0X_STRING_DEVICEERROR_NONE      ""
#define VL53L0X_STRING_DEVICEERROR_VCSELCONTINUITYTESTFAILURE  ""
#define VL53L0X_STRING_DEVICEERROR_VCSELWATCHDOGTESTFAILURE    ""
#define VL53L0X_STRING_DEVICEERROR_NOVHVVALUEFOUND             ""
#define VL53L0X_STRING_DEVICEERROR_MSRCNOTARGET                 ""
#define VL53L0X_STRING_DEVICEERROR_SNRCHECK                     ""
#define VL53L0X_STRING_DEVICEERROR_RANGEPHASECHECK              ""
#define VL53L0X_STRING_DEVICEERROR_SIGMATHRESHOLDCHECK          ""
```

```

#define VL53L0X_STRING_DEVICEERROR_TCC          ""
#define VL53L0X_STRING_DEVICEERROR_PHASECONSISTENCY  ""
#define VL53L0X_STRING_DEVICEERROR_MINCLIP      ""
#define VL53L0X_STRING_DEVICEERROR_RANGECOMPLETE  ""
#define VL53L0X_STRING_DEVICEERROR_ALGOUNDERFLOW  ""
#define VL53L0X_STRING_DEVICEERROR_ALGOOVERFLOW  ""
#define VL53L0X_STRING_DEVICEERROR_RANGEIGNORETHRESHOLD  ""
#define VL53L0X_STRING_DEVICEERROR_UNKNOWN      ""

/* Check Enable */
#define VL53L0X_STRING_CHECKENABLE_SIGMA_FINAL_RANGE  ""
#define VL53L0X_STRING_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE  ""
#define VL53L0X_STRING_CHECKENABLE_SIGNAL_REF_CLIP  ""
#define VL53L0X_STRING_CHECKENABLE_RANGE_IGNORE_THRESHOLD  ""

/* Sequence Step */
#define VL53L0X_STRING_SEQUENCESTEP_TCC          ""
#define VL53L0X_STRING_SEQUENCESTEP_DSS          ""
#define VL53L0X_STRING_SEQUENCESTEP_MSRC         ""
#define VL53L0X_STRING_SEQUENCESTEP_PRE_RANGE    ""
#define VL53L0X_STRING_SEQUENCESTEP_FINAL_RANGE  ""

#else

#define VL53L0X_STRING_DEVICE_INFO_NAME          "VL53L0X cut1.0"
#define VL53L0X_STRING_DEVICE_INFO_NAME_TS0      "VL53L0X TS0"
#define VL53L0X_STRING_DEVICE_INFO_NAME_TS1      "VL53L0X TS1"
#define VL53L0X_STRING_DEVICE_INFO_NAME_TS2      "VL53L0X TS2"
#define VL53L0X_STRING_DEVICE_INFO_NAME_ES1      "VL53L0X ES1 or later"
#define VL53L0X_STRING_DEVICE_INFO_TYPE          "VL53L0X"

```

```
/* PAL ERROR strings */

#define VL53L0X_STRING_ERROR_NONE \
    "No Error"

#define VL53L0X_STRING_ERROR_CALIBRATION_WARNING \
    "Calibration Warning Error"

#define VL53L0X_STRING_ERROR_MIN_CLIPPED \
    "Min clipped error"

#define VL53L0X_STRING_ERROR_UNDEFINED \
    "Undefined error"

#define VL53L0X_STRING_ERROR_INVALID_PARAMS \
    "Invalid parameters error"

#define VL53L0X_STRING_ERROR_NOT_SUPPORTED \
    "Not supported error"

#define VL53L0X_STRING_ERROR_RANGE_ERROR \
    "Range error"

#define VL53L0X_STRING_ERROR_TIME_OUT \
    "Time out error"

#define VL53L0X_STRING_ERROR_MODE_NOT_SUPPORTED \
    "Mode not supported error"

#define VL53L0X_STRING_ERROR_BUFFER_TOO_SMALL \
    "Buffer too small"

#define VL53L0X_STRING_ERROR_GPIO_NOT_EXISTING \
    "GPIO not existing"

#define VL53L0X_STRING_ERROR_GPIO_FUNCTIONALITY_NOT_SUPPORTED \
    "GPIO funct not supported"

#define VL53L0X_STRING_ERROR_INTERRUPT_NOT_CLEARED \
    "Interrupt not Cleared"

#define VL53L0X_STRING_ERROR_CONTROL_INTERFACE \
    "Control Interface Error"
```

```
#define VL53LOX_STRING_ERROR_INVALID_COMMAND \
```

```
    "Invalid Command Error"
```

```
#define VL53LOX_STRING_ERROR_DIVISION_BY_ZERO \
```

```
    "Division by zero Error"
```

```
#define VL53LOX_STRING_ERROR_REF_SPAD_INIT \
```

```
    "Reference Spad Init Error"
```

```
#define VL53LOX_STRING_ERROR_NOT_IMPLEMENTED \
```

```
    "Not implemented error"
```

```
#define VL53LOX_STRING_UNKNOW_ERROR_CODE \
```

```
    "Unknown Error Code"
```

```
/* Range Status */
```

```
#define VL53LOX_STRING_RANGESTATUS_NONE        "No Update"
```

```
#define VL53LOX_STRING_RANGESTATUS_RANGEVALID  "Range Valid"
```

```
#define VL53LOX_STRING_RANGESTATUS_SIGMA      "Sigma Fail"
```

```
#define VL53LOX_STRING_RANGESTATUS_SIGNAL     "Signal Fail"
```

```
#define VL53LOX_STRING_RANGESTATUS_MINRANGE   "Min Range Fail"
```

```
#define VL53LOX_STRING_RANGESTATUS_PHASE      "Phase Fail"
```

```
#define VL53LOX_STRING_RANGESTATUS_HW         "Hardware Fail"
```

```
/* Range Status */
```

```
#define VL53LOX_STRING_STATE_POWERDOWN        "POWERDOWN State"
```

```
#define VL53LOX_STRING_STATE_WAIT_STATICINIT \
```

```
    "Wait for staticinit State"
```

```
#define VL53LOX_STRING_STATE_STANDBY          "STANDBY State"
```

```

#define VL53L0X_STRING_STATE_IDLE          "IDLE State"

#define VL53L0X_STRING_STATE_RUNNING       "RUNNING State"

#define VL53L0X_STRING_STATE_UNKNOWN      "UNKNOWN State"

#define VL53L0X_STRING_STATE_ERROR        "ERROR State"


/* Device Specific */

#define VL53L0X_STRING_DEVICEERROR_NONE    "No Update"

#define VL53L0X_STRING_DEVICEERROR_VCSELCONTINUITYTESTFAILURE \
    "VCSEL Continuity Test Failure"

#define VL53L0X_STRING_DEVICEERROR_VCSELWATCHDOGTESTFAILURE \
    "VCSEL Watchdog Test Failure"

#define VL53L0X_STRING_DEVICEERROR_NOVHVVALUEFOUND \
    "No VHV Value found"

#define VL53L0X_STRING_DEVICEERROR_MSRCNOTARGET \
    "MSRC No Target Error"

#define VL53L0X_STRING_DEVICEERROR_SNRCHECK \
    "SNR Check Exit"

#define VL53L0X_STRING_DEVICEERROR_RANGEPHASECHECK \
    "Range Phase Check Error"

#define VL53L0X_STRING_DEVICEERROR_SIGMATHRESHOLDCHECK \
    "Sigma Threshold Check Error"

#define VL53L0X_STRING_DEVICEERROR_TCC \
    "TCC Error"

#define VL53L0X_STRING_DEVICEERROR_PHASECONSISTENCY \
    "Phase Consistency Error"

#define VL53L0X_STRING_DEVICEERROR_MINCLIP \
    "Min Clip Error"

#define VL53L0X_STRING_DEVICEERROR_RANGECOMPLETE \

```

```

        "Range Complete"

#define VL53L0X_STRING_DEVICEERROR_ALGOUNDERFLOW \
        "Range Algo Underflow Error"

#define VL53L0X_STRING_DEVICEERROR_ALGOOVERFLOW \
        "Range Algo Overflow Error"

#define VL53L0X_STRING_DEVICEERROR_RANGEIGNORETHRESHOLD \
        "Range Ignore Threshold Error"

#define VL53L0X_STRING_DEVICEERROR_UNKNOWN \
        "Unknown error code"

/* Check Enable */

#define VL53L0X_STRING_CHECKENABLE_SIGMA_FINAL_RANGE \
        "SIGMA FINAL RANGE"

#define VL53L0X_STRING_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE \
        "SIGNAL RATE FINAL RANGE"

#define VL53L0X_STRING_CHECKENABLE_SIGNAL_REF_CLIP \
        "SIGNAL REF CLIP"

#define VL53L0X_STRING_CHECKENABLE_RANGE_IGNORE_THRESHOLD \
        "RANGE IGNORE THRESHOLD"

#define VL53L0X_STRING_CHECKENABLE_SIGNAL_RATE_MSRC \
        "SIGNAL RATE MSRC"

#define VL53L0X_STRING_CHECKENABLE_SIGNAL_RATE_PRE_RANGE \
        "SIGNAL RATE PRE RANGE"

/* Sequence Step */

#define VL53L0X_STRING_SEQUENCESTEP_TCC          "TCC"
#define VL53L0X_STRING_SEQUENCESTEP_DSS          "DSS"
#define VL53L0X_STRING_SEQUENCESTEP_MSRC          "MSRC"
#define VL53L0X_STRING_SEQUENCESTEP_PRE_RANGE    "PRE RANGE"

```

```
#define VL53L0X_STRING_SEQUENCESTEP_FINAL_RANGE    "FINAL RANGE"
#endif /* USE_EMPTY_STRING */
```

```
#ifdef __cplusplus
}
#endif
```

```
#endif
```

1.1.35 vl53l0x_api_strings.cpp

Below:

```
/******
```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the
names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED

WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#include "vl53l0x_api.h"
```

```
#include "vl53l0x_api_core.h"
```

```
#include "vl53l0x_api_strings.h"
```

```
#ifndef __KERNEL__
```

```
#include <stdlib.h>
```

```
#endif
```

```
#define LOG_FUNCTION_START(fmt, ...) \
```

```
    _LOG_FUNCTION_START	TRACE_MODULE_API, fmt, ##__VA_ARGS__
```

```
#define LOG_FUNCTION_END(status, ...) \
```

```
    _LOG_FUNCTION_END	TRACE_MODULE_API, status, ##__VA_ARGS__
```

```
#define LOG_FUNCTION_END_FMT(status, fmt, ...) \
```

```
    _LOG_FUNCTION_END_FMT	TRACE_MODULE_API, status, fmt, ##__VA_ARGS__
```

```
VL53L0X_Error VL53L0X_check_part_used(VL53L0X_DEV Dev,
```

```
    uint8_t *Revision,
```

```

        VL53L0X_DeviceInfo_t *pVL53L0X_DeviceInfo)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t ModuleIdInt;
    char *ProductId_tmp;

    LOG_FUNCTION_START("");

    Status = VL53L0X_get_info_from_device(Dev, 2);

    if (Status == VL53L0X_ERROR_NONE) {
        ModuleIdInt = VL53L0X_GETDEVICESPECIFICPARAMETER(Dev, ModuleId);

        if (ModuleIdInt == 0) {
            *Revision = 0;
            VL53L0X_COPYSTRING(pVL53L0X_DeviceInfo->ProductId, "");
        } else {
            *Revision = VL53L0X_GETDEVICESPECIFICPARAMETER(Dev, Revision);
            ProductId_tmp = VL53L0X_GETDEVICESPECIFICPARAMETER(Dev,
                ProductId);
            VL53L0X_COPYSTRING(pVL53L0X_DeviceInfo->ProductId, ProductId_tmp);
        }
    }

    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

VL53L0X_Error VL53L0X_get_device_info(VL53L0X_DEV Dev,
                                       VL53L0X_DeviceInfo_t *pVL53L0X_DeviceInfo)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t revision_id;

    uint8_t Revision;

    Status = VL53L0X_check_part_used(Dev, &Revision, pVL53L0X_DeviceInfo);
    if (Status == VL53L0X_ERROR_NONE) {
        if (Revision == 0) {
            VL53L0X_COPYSTRING(pVL53L0X_DeviceInfo->Name,
                               VL53L0X_STRING_DEVICE_INFO_NAME_TS0);
        } else if ((Revision <= 34) && (Revision != 32)) {
            VL53L0X_COPYSTRING(pVL53L0X_DeviceInfo->Name,
                               VL53L0X_STRING_DEVICE_INFO_NAME_TS1);
        } else if (Revision < 39) {
            VL53L0X_COPYSTRING(pVL53L0X_DeviceInfo->Name,
                               VL53L0X_STRING_DEVICE_INFO_NAME_TS2);
        } else {
            VL53L0X_COPYSTRING(pVL53L0X_DeviceInfo->Name,
                               VL53L0X_STRING_DEVICE_INFO_NAME_ES1);
        }

        VL53L0X_COPYSTRING(pVL53L0X_DeviceInfo->Type,
                           VL53L0X_STRING_DEVICE_INFO_TYPE);
    }

    if (Status == VL53L0X_ERROR_NONE) {

```

```

        Status = VL53L0X_RdByte(Dev, VL53L0X_REG_IDENTIFICATION_MODEL_ID,
                                &pVL53L0X_DeviceInfo->ProductType);
    }

    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_RdByte(Dev,
                                VL53L0X_REG_IDENTIFICATION_REVISION_ID,
                                &revision_id);

        pVL53L0X_DeviceInfo->ProductRevisionMajor = 1;
        pVL53L0X_DeviceInfo->ProductRevisionMinor =
            (revision_id & 0xF0) >> 4;
    }

    return Status;
}

```

```

VL53L0X_Error VL53L0X_get_device_error_string(VL53L0X_DeviceError ErrorCode,
                                              char *pDeviceErrorString)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    switch (ErrorCode) {
    case VL53L0X_DEVICEERROR_NONE:
        VL53L0X_COPYSTRING(pDeviceErrorString,
                            VL53L0X_STRING_DEVICEERROR_NONE);
        break;

    case VL53L0X_DEVICEERROR_VCSELCONTINUITYTESTFAILURE:

```

```
        VL53L0X_COPYSTRING(pDeviceErrorString,
                            VL53L0X_STRING_DEVICEERROR_VCSELCONTINUITYTESTFAILURE);
break;
case VL53L0X_DEVICEERROR_VCSELWATCHDOGTESTFAILURE:
    VL53L0X_COPYSTRING(pDeviceErrorString,
                        VL53L0X_STRING_DEVICEERROR_VCSELWATCHDOGTESTFAILURE);
break;
case VL53L0X_DEVICEERROR_NOVHVVALUEFOUND:
    VL53L0X_COPYSTRING(pDeviceErrorString,
                        VL53L0X_STRING_DEVICEERROR_NOVHVVALUEFOUND);
break;
case VL53L0X_DEVICEERROR_MSRCNOTARGET:
    VL53L0X_COPYSTRING(pDeviceErrorString,
                        VL53L0X_STRING_DEVICEERROR_MSRCNOTARGET);
break;
case VL53L0X_DEVICEERROR_SNRCHECK:
    VL53L0X_COPYSTRING(pDeviceErrorString,
                        VL53L0X_STRING_DEVICEERROR_SNRCHECK);
break;
case VL53L0X_DEVICEERROR_RANGEPHASECHECK:
    VL53L0X_COPYSTRING(pDeviceErrorString,
                        VL53L0X_STRING_DEVICEERROR_RANGEPHASECHECK);
break;
case VL53L0X_DEVICEERROR_SIGMATHRESHOLDCHECK:
    VL53L0X_COPYSTRING(pDeviceErrorString,
                        VL53L0X_STRING_DEVICEERROR_SIGMATHRESHOLDCHECK);
break;
case VL53L0X_DEVICEERROR_TCC:
    VL53L0X_COPYSTRING(pDeviceErrorString,
```

```
        VL53L0X_STRING_DEVICEERROR_TCC);

break;

case VL53L0X_DEVICEERROR_PHASECONSISTENCY:

    VL53L0X_COPYSTRING(pDeviceErrorString,

        VL53L0X_STRING_DEVICEERROR_PHASECONSISTENCY);

break;

case VL53L0X_DEVICEERROR_MINCLIP:

    VL53L0X_COPYSTRING(pDeviceErrorString,

        VL53L0X_STRING_DEVICEERROR_MINCLIP);

break;

case VL53L0X_DEVICEERROR_RANGECOMPLETE:

    VL53L0X_COPYSTRING(pDeviceErrorString,

        VL53L0X_STRING_DEVICEERROR_RANGECOMPLETE);

break;

case VL53L0X_DEVICEERROR_ALGOUNDERFLOW:

    VL53L0X_COPYSTRING(pDeviceErrorString,

        VL53L0X_STRING_DEVICEERROR_ALGOUNDERFLOW);

break;

case VL53L0X_DEVICEERROR_ALGOOVERFLOW:

    VL53L0X_COPYSTRING(pDeviceErrorString,

        VL53L0X_STRING_DEVICEERROR_ALGOOVERFLOW);

break;

case VL53L0X_DEVICEERROR_RANGEIGNORETHRESHOLD:

    VL53L0X_COPYSTRING(pDeviceErrorString,

        VL53L0X_STRING_DEVICEERROR_RANGEIGNORETHRESHOLD);

break;

default:

    VL53L0X_COPYSTRING(pDeviceErrorString,
```

```

        VL53L0X_STRING_UNKNOW_ERROR_CODE);

    }

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_get_range_status_string(uint8_t RangeStatus,
        char *pRangeStatusString)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    switch (RangeStatus) {
    case 0:
        VL53L0X_COPYSTRING(pRangeStatusString,
            VL53L0X_STRING_RANGESTATUS_RANGEVALID);
        break;
    case 1:
        VL53L0X_COPYSTRING(pRangeStatusString,
            VL53L0X_STRING_RANGESTATUS_SIGMA);
        break;
    case 2:
        VL53L0X_COPYSTRING(pRangeStatusString,
            VL53L0X_STRING_RANGESTATUS_SIGNAL);
        break;
    case 3:

```

```

        VL53L0X_COPYSTRING(pRangeStatusString,
                            VL53L0X_STRING_RANGESTATUS_MINRANGE);

    break;

    case 4:

        VL53L0X_COPYSTRING(pRangeStatusString,
                            VL53L0X_STRING_RANGESTATUS_PHASE);

    break;

    case 5:

        VL53L0X_COPYSTRING(pRangeStatusString,
                            VL53L0X_STRING_RANGESTATUS_HW);

    break;


    default: /**/

        VL53L0X_COPYSTRING(pRangeStatusString,
                            VL53L0X_STRING_RANGESTATUS_NONE);

    }


    LOG_FUNCTION_END(Status);

    return Status;

}


VL53L0X_Error VL53L0X_get_pal_error_string(VL53L0X_Error PalErrorCode,
                                           char *pPalErrorString)

{

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;


    LOG_FUNCTION_START("");


    switch (PalErrorCode) {

```

```
case VL53L0X_ERROR_NONE:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_NONE);

break;

case VL53L0X_ERROR_CALIBRATION_WARNING:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_CALIBRATION_WARNING);

break;

case VL53L0X_ERROR_MIN_CLIPPED:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_MIN_CLIPPED);

break;

case VL53L0X_ERROR_UNDEFINED:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_UNDEFINED);

break;

case VL53L0X_ERROR_INVALID_PARAMS:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_INVALID_PARAMS);

break;

case VL53L0X_ERROR_NOT_SUPPORTED:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_NOT_SUPPORTED);

break;

case VL53L0X_ERROR_INTERRUPT_NOT_CLEARED:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_INTERRUPT_NOT_CLEARED);

break;

case VL53L0X_ERROR_RANGE_ERROR:
```

```
        VL53L0X_COPYSTRING(pPalErrorString,
                            VL53L0X_STRING_ERROR_RANGE_ERROR);

break;

case VL53L0X_ERROR_TIME_OUT:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_TIME_OUT);

break;

case VL53L0X_ERROR_MODE_NOT_SUPPORTED:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_MODE_NOT_SUPPORTED);

break;

case VL53L0X_ERROR_BUFFER_TOO_SMALL:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_BUFFER_TOO_SMALL);

break;

case VL53L0X_ERROR_GPIO_NOT_EXISTING:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_GPIO_NOT_EXISTING);

break;

case VL53L0X_ERROR_GPIO_FUNCTIONALITY_NOT_SUPPORTED:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_GPIO_FUNCTIONALITY_NOT_SUPPORTED);

break;

case VL53L0X_ERROR_CONTROL_INTERFACE:

    VL53L0X_COPYSTRING(pPalErrorString,
                        VL53L0X_STRING_ERROR_CONTROL_INTERFACE);

break;

case VL53L0X_ERROR_INVALID_COMMAND:

    VL53L0X_COPYSTRING(pPalErrorString,
```

```

        VL53L0X_STRING_ERROR_INVALID_COMMAND));

break;

case VL53L0X_ERROR_DIVISION_BY_ZERO:

    VL53L0X_COPYSTRING(pPalErrorString,

        VL53L0X_STRING_ERROR_DIVISION_BY_ZERO);

break;

case VL53L0X_ERROR_REF_SPAD_INIT:

    VL53L0X_COPYSTRING(pPalErrorString,

        VL53L0X_STRING_ERROR_REF_SPAD_INIT);

break;

case VL53L0X_ERROR_NOT_IMPLEMENTED:

    VL53L0X_COPYSTRING(pPalErrorString,

        VL53L0X_STRING_ERROR_NOT_IMPLEMENTED);

break;


default:

    VL53L0X_COPYSTRING(pPalErrorString,

        VL53L0X_STRING_UNKNOW_ERROR_CODE);

}


LOG_FUNCTION_END(Status);

return Status;

}


VL53L0X_Error VL53L0X_get_pal_state_string(VL53L0X_State PalStateCode,

    char *pPalStateString)

{

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

```

```
LOG_FUNCTION_START("");
```

```
switch (PalStateCode) {
```

```
case VL53L0X_STATE_POWERDOWN:
```

```
    VL53L0X_COPYSTRING(pPalStateString,  
                        VL53L0X_STRING_STATE_POWERDOWN);
```

```
break;
```

```
case VL53L0X_STATE_WAIT_STATICINIT:
```

```
    VL53L0X_COPYSTRING(pPalStateString,  
                        VL53L0X_STRING_STATE_WAIT_STATICINIT);
```

```
break;
```

```
case VL53L0X_STATE_STANDBY:
```

```
    VL53L0X_COPYSTRING(pPalStateString,  
                        VL53L0X_STRING_STATE_STANDBY);
```

```
break;
```

```
case VL53L0X_STATE_IDLE:
```

```
    VL53L0X_COPYSTRING(pPalStateString,  
                        VL53L0X_STRING_STATE_IDLE);
```

```
break;
```

```
case VL53L0X_STATE_RUNNING:
```

```
    VL53L0X_COPYSTRING(pPalStateString,  
                        VL53L0X_STRING_STATE_RUNNING);
```

```
break;
```

```
case VL53L0X_STATE_UNKNOWN:
```

```
    VL53L0X_COPYSTRING(pPalStateString,  
                        VL53L0X_STRING_STATE_UNKNOWN);
```

```
break;
```

```
case VL53L0X_STATE_ERROR:
```

```
    VL53L0X_COPYSTRING(pPalStateString,
```

```

        VL53L0X_STRING_STATE_ERROR);

    break;

    default:

        VL53L0X_COPYSTRING(pPalStateString,
            VL53L0X_STRING_STATE_UNKNOWN);

    }

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_get_sequence_steps_info(
    VL53L0X_SequenceStepId SequenceStepId,
    char *pSequenceStepsString)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

    switch (SequenceStepId) {
    case VL53L0X_SEQUENCESTEP_TCC:
        VL53L0X_COPYSTRING(pSequenceStepsString,
            VL53L0X_STRING_SEQUENCESTEP_TCC);

        break;

    case VL53L0X_SEQUENCESTEP_DSS:
        VL53L0X_COPYSTRING(pSequenceStepsString,
            VL53L0X_STRING_SEQUENCESTEP_DSS);

        break;

    case VL53L0X_SEQUENCESTEP_MSRC:

```

```

        VL53L0X_COPYSTRING(pSequenceStepsString,
                            VL53L0X_STRING_SEQUENCESTEP_MSRC);

    break;

    case VL53L0X_SEQUENCESTEP_PRE_RANGE:

        VL53L0X_COPYSTRING(pSequenceStepsString,
                            VL53L0X_STRING_SEQUENCESTEP_PRE_RANGE);

    break;

    case VL53L0X_SEQUENCESTEP_FINAL_RANGE:

        VL53L0X_COPYSTRING(pSequenceStepsString,
                            VL53L0X_STRING_SEQUENCESTEP_FINAL_RANGE);

    break;

    default:

        Status = VL53L0X_ERROR_INVALID_PARAMS;

    }

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_get_limit_check_info(VL53L0X_DEV Dev, uint16_t LimitCheckId,
char *pLimitCheckString)
{

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    LOG_FUNCTION_START("");

```

```

switch (LimitCheckId) {
case VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE:
    VL53L0X_COPYSTRING(pLimitCheckString,
        VL53L0X_STRING_CHECKENABLE_SIGMA_FINAL_RANGE);
    break;
case VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE:
    VL53L0X_COPYSTRING(pLimitCheckString,
        VL53L0X_STRING_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE);
    break;
case VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP:
    VL53L0X_COPYSTRING(pLimitCheckString,
        VL53L0X_STRING_CHECKENABLE_SIGNAL_REF_CLIP);
    break;
case VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD:
    VL53L0X_COPYSTRING(pLimitCheckString,
        VL53L0X_STRING_CHECKENABLE_RANGE_IGNORE_THRESHOLD);
    break;

case VL53L0X_CHECKENABLE_SIGNAL_RATE_MSRC:
    VL53L0X_COPYSTRING(pLimitCheckString,
        VL53L0X_STRING_CHECKENABLE_SIGNAL_RATE_MSRC);
    break;

case VL53L0X_CHECKENABLE_SIGNAL_RATE_PRE_RANGE:
    VL53L0X_COPYSTRING(pLimitCheckString,
        VL53L0X_STRING_CHECKENABLE_SIGNAL_RATE_PRE_RANGE);
    break;

default:

```

```

        VL53L0X_COPYSTRING(pLimitCheckString,
                            VL53L0X_STRING_UNKNOW_ERROR_CODE);

    }

    LOG_FUNCTION_END(Status);

    return Status;

}

```

1.1.36 vl53l0x_api_ranging.h

Below:

```

/*****

```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#ifndef _VL53L0X_API_RANGING_H_
```

```
#define _VL53L0X_API_RANGING_H_
```

```
#include "vl53l0x_def.h"
```

```
#include "vl53l0x_platform.h"
```

```
#ifdef __cplusplus
```

```
extern "C" {
```

```
#endif
```

```
#ifdef __cplusplus
```

```
}
```

```
#endif
```

```
#endif /* _VL53L0X_API_RANGING_H_ */
```

1.1.37 vl53l0x_api_ranging.cpp

Below:

/******

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the
names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```

#include "vl53l0x_api.h"

#include "vl53l0x_api_core.h"


#ifndef __KERNEL__

#include <stdlib.h>

#endif

#define LOG_FUNCTION_START(fmt, ...) \
    _LOG_FUNCTION_START	TRACE_MODULE_API, fmt, ##__VA_ARGS__

#define LOG_FUNCTION_END(status, ...) \
    _LOG_FUNCTION_END	TRACE_MODULE_API, status, ##__VA_ARGS__

#define LOG_FUNCTION_END_FMT(status, fmt, ...) \
    _LOG_FUNCTION_END_FMT	TRACE_MODULE_API, status, fmt, ##__VA_ARGS__

```

1.1.38 vl53l0x_api_core.h

Below:

```

/*****

```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the

names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#ifndef _VL53L0X_API_CORE_H_  
#define _VL53L0X_API_CORE_H_
```

```
#include "vl53l0x_def.h"  
#include "vl53l0x_platform.h"
```

```
#ifdef __cplusplus  
extern "C" {  
#endif
```

```
VL53L0X_Error VL53L0X_reverse_bytes(uint8_t *data, uint32_t size);
```

VL53L0X_Error VL53L0X_measurement_poll_for_completion(VL53L0X_DEV Dev);

uint8_t VL53L0X_encode_vcsel_period(uint8_t vcsel_period_pclks);

uint8_t VL53L0X_decode_vcsel_period(uint8_t vcsel_period_reg);

uint32_t VL53L0X_isqrt(uint32_t num);

uint32_t VL53L0X_quadrature_sum(uint32_t a, uint32_t b);

VL53L0X_Error VL53L0X_get_info_from_device(VL53L0X_DEV Dev, uint8_t option);

VL53L0X_Error VL53L0X_set_vcsel_pulse_period(VL53L0X_DEV Dev,
VL53L0X_VcselPeriod VcselPeriodType, uint8_t VCSELPulsePeriodPCLK);

VL53L0X_Error VL53L0X_get_vcsel_pulse_period(VL53L0X_DEV Dev,
VL53L0X_VcselPeriod VcselPeriodType, uint8_t *pVCSELPulsePeriodPCLK);

uint32_t VL53L0X_decode_timeout(uint16_t encoded_timeout);

VL53L0X_Error get_sequence_step_timeout(VL53L0X_DEV Dev,
VL53L0X_SequenceStepId SequenceStepId,
uint32_t *pTimeOutMicroSecs);

VL53L0X_Error set_sequence_step_timeout(VL53L0X_DEV Dev,
VL53L0X_SequenceStepId SequenceStepId,
uint32_t TimeOutMicroSecs);

```
VL53L0X_Error VL53L0X_set_measurement_timing_budget_micro_seconds(VL53L0X_DEV Dev,  
    uint32_t MeasurementTimingBudgetMicroSeconds);
```

```
VL53L0X_Error VL53L0X_get_measurement_timing_budget_micro_seconds(VL53L0X_DEV Dev,  
    uint32_t *pMeasurementTimingBudgetMicroSeconds);
```

```
VL53L0X_Error VL53L0X_load_tuning_settings(VL53L0X_DEV Dev,  
    uint8_t *pTuningSettingBuffer);
```

```
VL53L0X_Error VL53L0X_calc_sigma_estimate(VL53L0X_DEV Dev,  
    VL53L0X_RangingMeasurementData_t *pRangingMeasurementData,  
    FixPoint1616_t *pSigmaEstimate, uint32_t *pDmax_mm);
```

```
VL53L0X_Error VL53L0X_get_total_xtalk_rate(VL53L0X_DEV Dev,  
    VL53L0X_RangingMeasurementData_t *pRangingMeasurementData,  
    FixPoint1616_t *ptotal_xtalk_rate_mcps);
```

```
VL53L0X_Error VL53L0X_get_total_signal_rate(VL53L0X_DEV Dev,  
    VL53L0X_RangingMeasurementData_t *pRangingMeasurementData,  
    FixPoint1616_t *ptotal_signal_rate_mcps);
```

```
VL53L0X_Error VL53L0X_get_pal_range_status(VL53L0X_DEV Dev,  
    uint8_t DeviceRangeStatus,  
    FixPoint1616_t SignalRate,  
    uint16_t EffectiveSpadRtnCount,  
    VL53L0X_RangingMeasurementData_t *pRangingMeasurementData,  
    uint8_t *pPalRangeStatus);
```

```
uint32_t VL53L0X_calc_timeout_mclks(VL53L0X_DEV Dev,
```

```
uint32_t timeout_period_us, uint8_t vtsel_period_pclks);

uint16_t VL53L0X_encode_timeout(uint32_t timeout_macro_clks);
```

```
#ifdef __cplusplus
}
#endif
```

```
#endif /* _VL53L0X_API_CORE_H_ */
```

[1.1.39 vl53l0x_api_core.cpp](#)

Below:

```
/******
```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the
names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND

NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#include "vl53l0x_api.h"
```

```
#include "vl53l0x_api_core.h"
```

```
#include "vl53l0x_api_calibration.h"
```

```
#ifndef __KERNEL__
```

```
#include <stdlib.h>
```

```
#endif
```

```
#define LOG_FUNCTION_START(fmt, ...) \
```

```
    _LOG_FUNCTION_START	TRACE_MODULE_API, fmt, ##__VA_ARGS__
```

```
#define LOG_FUNCTION_END(status, ...) \
```

```
    _LOG_FUNCTION_END	TRACE_MODULE_API, status, ##__VA_ARGS__
```

```
#define LOG_FUNCTION_END_FMT(status, fmt, ...) \
```

```
    _LOG_FUNCTION_END_FMT	TRACE_MODULE_API, status, fmt, ##__VA_ARGS__
```

```
VL53L0X_Error VL53L0X_reverse_bytes(uint8_t *data, uint32_t size)
```

```
{
```

```
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
    uint8_t tempData;
```

```

uint32_t mirrorIndex;

uint32_t middle = size/2;

uint32_t index;

for (index = 0; index < middle; index++) {
    mirrorIndex          = size - index - 1;
    tempData             = data[index];
    data[index]          = data[mirrorIndex];
    data[mirrorIndex] = tempData;
}

return Status;
}

VL53L0X_Error VL53L0X_measurement_poll_for_completion(VL53L0X_DEV Dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t NewDataReady = 0;

    uint32_t LoopNb;

    LOG_FUNCTION_START("");

    LoopNb = 0;

    do {
        Status = VL53L0X_GetMeasurementDataReady(Dev, &NewDataReady);

        if (Status != 0)
            break; /* the error is set */

        if (NewDataReady == 1)

```

```

        break; /* done note that status == 0 */

    LoopNb++;

    if (LoopNb >= VL53L0X_DEFAULT_MAX_LOOP) {
        Status = VL53L0X_ERROR_TIME_OUT;
        break;
    }

    VL53L0X_PollingDelay(Dev);
} while (1);

LOG_FUNCTION_END(Status);

return Status;
}

uint8_t VL53L0X_decode_vcsel_period(uint8_t vcsel_period_reg)
{
    /*!
    * Converts the encoded VCSEL period register value into the real
    * period in PLL clocks
    */

    uint8_t vcsel_period_pclks = 0;

    vcsel_period_pclks = (vcsel_period_reg + 1) << 1;

    return vcsel_period_pclks;
}

```

```
}
```

```
uint8_t VL53L0X_encode_vcsel_period(uint8_t vcsel_period_pclks)
```

```
{
```

```
    /*!
```

```
     * Converts the encoded VCSEL period register value into the real period
```

```
     * in PLL clocks
```

```
    */
```

```
    uint8_t vcsel_period_reg = 0;
```

```
    vcsel_period_reg = (vcsel_period_pclks >> 1) - 1;
```

```
    return vcsel_period_reg;
```

```
}
```

```
uint32_t VL53L0X_isqrt(uint32_t num)
```

```
{
```

```
    /*
```

```
     * Implements an integer square root
```

```
     *
```

```
     * From: http://en.wikipedia.org/wiki/Methods\_of\_computing\_square\_roots
```

```
    */
```

```
    uint32_t res = 0;
```

```
    uint32_t bit = 1 << 30;
```

```
    /* The second-to-top bit is set:
```

```
     *      1 << 14 for 16-bits, 1 << 30 for 32 bits */
```

```

/* "bit" starts at the highest power of four <= the argument. */
while (bit > num)
    bit >>= 2;

while (bit != 0) {
    if (num >= res + bit) {
        num -= res + bit;
        res = (res >> 1) + bit;
    } else
        res >>= 1;

    bit >>= 2;
}

return res;
}

```

```

uint32_t VL53L0X_quadrature_sum(uint32_t a, uint32_t b)
{
    /*
     * Implements a quadrature sum
     *
     *  $rea = \sqrt{a^2 + b^2}$ 
     *
     * Trap overflow case max input value is 65535 (16-bit value)
     * as internal calc are 32-bit wide
    */
}

```

```

*

* If overflow then set output to maximum
*/
uint32_t res = 0;

if (a > 65535 || b > 65535)
    res = 65535;
else
    res = VL53L0X_isqrt(a * a + b * b);

return res;
}

VL53L0X_Error VL53L0X_device_read_strobe(VL53L0X_DEV Dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t strobe;
    uint32_t LoopNb;
    LOG_FUNCTION_START("");

    Status |= VL53L0X_WrByte(Dev, 0x83, 0x00);

    /* polling
    * use timeout to avoid deadlock*/
    if (Status == VL53L0X_ERROR_NONE) {
        LoopNb = 0;
        do {
            Status = VL53L0X_RdByte(Dev, 0x83, &strobe);

```

```

        if ((strobe != 0x00) || Status != VL53L0X_ERROR_NONE)
            break;

        LoopNb = LoopNb + 1;
    } while (LoopNb < VL53L0X_DEFAULT_MAX_LOOP);

    if (LoopNb >= VL53L0X_DEFAULT_MAX_LOOP)
        Status = VL53L0X_ERROR_TIME_OUT;

}

Status |= VL53L0X_WrByte(Dev, 0x83, 0x01);

LOG_FUNCTION_END(Status);

return Status;

}

VL53L0X_Error VL53L0X_get_info_from_device(VL53L0X_DEV Dev, uint8_t option)
{

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t byte;

    uint32_t TmpDWord;

    uint8_t ModuleId;

    uint8_t Revision;

    uint8_t ReferenceSpadCount = 0;

    uint8_t ReferenceSpadType = 0;

    uint32_t PartUIDUpper = 0;

```

```

uint32_t PartUIDLower = 0;
uint32_t OffsetFixed1104_mm = 0;
int16_t OffsetMicroMeters = 0;
uint32_t DistMeasTgtFixed1104_mm = 400 << 4;
uint32_t DistMeasFixed1104_400_mm = 0;
uint32_t SignalRateMeasFixed1104_400_mm = 0;
char ProductId[19];
char *ProductId_tmp;
uint8_t ReadDataFromDeviceDone;
FixPoint1616_t SignalRateMeasFixed400mmFix = 0;
uint8_t NvmRefGoodSpadMap[VL53L0X_REF_SPAD_BUFFER_SIZE];
int i;

```

```

LOG_FUNCTION_START("");

```

```

ReadDataFromDeviceDone = VL53L0X_GETDEVICESPECIFICPARAMETER(Dev,
    ReadDataFromDeviceDone);

```

```

/* This access is done only once after that a GetDeviceInfo or
 * datainit is done*/

```

```

if (ReadDataFromDeviceDone != 7) {

```

```

    Status |= VL53L0X_WrByte(Dev, 0x80, 0x01);
    Status |= VL53L0X_WrByte(Dev, 0xFF, 0x01);
    Status |= VL53L0X_WrByte(Dev, 0x00, 0x00);

```

```

    Status |= VL53L0X_WrByte(Dev, 0xFF, 0x06);
    Status |= VL53L0X_RdByte(Dev, 0x83, &byte);

```

```
Status |= VL53L0X_WrByte(Dev, 0x83, byte|4);
```

```
Status |= VL53L0X_WrByte(Dev, 0xFF, 0x07);
```

```
Status |= VL53L0X_WrByte(Dev, 0x81, 0x01);
```

```
Status |= VL53L0X_PollingDelay(Dev);
```

```
Status |= VL53L0X_WrByte(Dev, 0x80, 0x01);
```

```
if (((option & 1) == 1) &&
```

```
    ((ReadDataFromDeviceDone & 1) == 0)) {
```

```
    Status |= VL53L0X_WrByte(Dev, 0x94, 0x6b);
```

```
    Status |= VL53L0X_device_read_strobe(Dev);
```

```
    Status |= VL53L0X_RdDWord(Dev, 0x90, &TmpDWord);
```

```
    ReferenceSpadCount = (uint8_t)((TmpDWord >> 8) & 0x07f);
```

```
    ReferenceSpadType = (uint8_t)((TmpDWord >> 15) & 0x01);
```

```
    Status |= VL53L0X_WrByte(Dev, 0x94, 0x24);
```

```
    Status |= VL53L0X_device_read_strobe(Dev);
```

```
    Status |= VL53L0X_RdDWord(Dev, 0x90, &TmpDWord);
```

```
    NvmRefGoodSpadMap[0] = (uint8_t)((TmpDWord >> 24)
```

```
        & 0xff);
```

```
    NvmRefGoodSpadMap[1] = (uint8_t)((TmpDWord >> 16)
```

```
        & 0xff);
```

```
    NvmRefGoodSpadMap[2] = (uint8_t)((TmpDWord >> 8)
```

```
        & 0xff);
```

```
    NvmRefGoodSpadMap[3] = (uint8_t)(TmpDWord & 0xff);
```

```

        Status |= VL53L0X_WrByte(Dev, 0x94, 0x25);

        Status |= VL53L0X_device_read_strobe(Dev);

        Status |= VL53L0X_RdDWord(Dev, 0x90, &TmpDWord);


        NvmRefGoodSpadMap[4] = (uint8_t)((TmpDWord >> 24)
                                           & 0xff);

        NvmRefGoodSpadMap[5] = (uint8_t)((TmpDWord >> 16)
                                           & 0xff);
    }

    if (((option & 2) == 2) &&
        ((ReadDataFromDeviceDone & 2) == 0)) {

        Status |= VL53L0X_WrByte(Dev, 0x94, 0x02);

        Status |= VL53L0X_device_read_strobe(Dev);

        Status |= VL53L0X_RdByte(Dev, 0x90, &ModuleId);


        Status |= VL53L0X_WrByte(Dev, 0x94, 0x7B);

        Status |= VL53L0X_device_read_strobe(Dev);

        Status |= VL53L0X_RdByte(Dev, 0x90, &Revision);


        Status |= VL53L0X_WrByte(Dev, 0x94, 0x77);

        Status |= VL53L0X_device_read_strobe(Dev);

        Status |= VL53L0X_RdDWord(Dev, 0x90, &TmpDWord);


        ProductId[0] = (char)((TmpDWord >> 25) & 0x07f);

        ProductId[1] = (char)((TmpDWord >> 18) & 0x07f);

        ProductId[2] = (char)((TmpDWord >> 11) & 0x07f);

```

```
ProductId[3] = (char)((TmpDWord >> 4) & 0x07f);
```

```
byte = (uint8_t)((TmpDWord & 0x00f) << 3);
```

```
Status |= VL53L0X_WrByte(Drv, 0x94, 0x78);
```

```
Status |= VL53L0X_device_read_strobe(Drv);
```

```
Status |= VL53L0X_RdDWord(Drv, 0x90, &TmpDWord);
```

```
ProductId[4] = (char)(byte +  
((TmpDWord >> 29) & 0x07f));
```

```
ProductId[5] = (char)((TmpDWord >> 22) & 0x07f);
```

```
ProductId[6] = (char)((TmpDWord >> 15) & 0x07f);
```

```
ProductId[7] = (char)((TmpDWord >> 8) & 0x07f);
```

```
ProductId[8] = (char)((TmpDWord >> 1) & 0x07f);
```

```
byte = (uint8_t)((TmpDWord & 0x001) << 6);
```

```
Status |= VL53L0X_WrByte(Drv, 0x94, 0x79);
```

```
Status |= VL53L0X_device_read_strobe(Drv);
```

```
Status |= VL53L0X_RdDWord(Drv, 0x90, &TmpDWord);
```

```
ProductId[9] = (char)(byte +  
((TmpDWord >> 26) & 0x07f));
```

```
ProductId[10] = (char)((TmpDWord >> 19) & 0x07f);
```

```
ProductId[11] = (char)((TmpDWord >> 12) & 0x07f);
```

```
ProductId[12] = (char)((TmpDWord >> 5) & 0x07f);
```

```
byte = (uint8_t)((TmpDWord & 0x01f) << 2);
```

```
Status |= VL53L0X_WrByte(Dev, 0x94, 0x7A);
```

```
Status |= VL53L0X_device_read_strobe(Dev);
```

```
Status |= VL53L0X_RdDWord(Dev, 0x90, &TmpDWord);
```

```
ProductId[13] = (char)(byte +  
                        ((TmpDWord >> 30) & 0x07f));
```

```
ProductId[14] = (char)((TmpDWord >> 23) & 0x07f);
```

```
ProductId[15] = (char)((TmpDWord >> 16) & 0x07f);
```

```
ProductId[16] = (char)((TmpDWord >> 9) & 0x07f);
```

```
ProductId[17] = (char)((TmpDWord >> 2) & 0x07f);
```

```
ProductId[18] = '\0';
```

```
}
```

```
if (((option & 4) == 4) &&
```

```
    ((ReadDataFromDeviceDone & 4) == 0)) {
```

```
Status |= VL53L0X_WrByte(Dev, 0x94, 0x7B);
```

```
Status |= VL53L0X_device_read_strobe(Dev);
```

```
Status |= VL53L0X_RdDWord(Dev, 0x90, &PartUIDUpper);
```

```
Status |= VL53L0X_WrByte(Dev, 0x94, 0x7C);
```

```
Status |= VL53L0X_device_read_strobe(Dev);
```

```
Status |= VL53L0X_RdDWord(Dev, 0x90, &PartUIDLower);
```

```
Status |= VL53L0X_WrByte(Dev, 0x94, 0x73);  
Status |= VL53L0X_device_read_strobe(Dev);  
Status |= VL53L0X_RdDWord(Dev, 0x90, &TmpDWord);
```

```
SignalRateMeasFixed1104_400_mm = (TmpDWord &  
0x0000000ff) << 8;
```

```
Status |= VL53L0X_WrByte(Dev, 0x94, 0x74);  
Status |= VL53L0X_device_read_strobe(Dev);  
Status |= VL53L0X_RdDWord(Dev, 0x90, &TmpDWord);
```

```
SignalRateMeasFixed1104_400_mm |= ((TmpDWord &  
0xff000000) >> 24);
```

```
Status |= VL53L0X_WrByte(Dev, 0x94, 0x75);  
Status |= VL53L0X_device_read_strobe(Dev);  
Status |= VL53L0X_RdDWord(Dev, 0x90, &TmpDWord);
```

```
DistMeasFixed1104_400_mm = (TmpDWord & 0x0000000ff)  
    << 8;
```

```
Status |= VL53L0X_WrByte(Dev, 0x94, 0x76);  
Status |= VL53L0X_device_read_strobe(Dev);  
Status |= VL53L0X_RdDWord(Dev, 0x90, &TmpDWord);
```

```
DistMeasFixed1104_400_mm |= ((TmpDWord & 0xff000000)  
    >> 24);
```

```
}
```

```

        Status |= VL53L0X_WrByte(Dev, 0x81, 0x00);
        Status |= VL53L0X_WrByte(Dev, 0xFF, 0x06);
        Status |= VL53L0X_RdByte(Dev, 0x83, &byte);
        Status |= VL53L0X_WrByte(Dev, 0x83, byte&0xfb);
        Status |= VL53L0X_WrByte(Dev, 0xFF, 0x01);
        Status |= VL53L0X_WrByte(Dev, 0x00, 0x01);

        Status |= VL53L0X_WrByte(Dev, 0xFF, 0x00);
        Status |= VL53L0X_WrByte(Dev, 0x80, 0x00);
    }

    if ((Status == VL53L0X_ERROR_NONE) &&
        (ReadDataFromDeviceDone != 7)) {
        /* Assign to variable if status is ok */
        if (((option & 1) == 1) &&
            ((ReadDataFromDeviceDone & 1) == 0)) {
            VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
                ReferenceSpadCount, ReferenceSpadCount);

            VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
                ReferenceSpadType, ReferenceSpadType);

            for (i = 0; i < VL53L0X_REF_SPAD_BUFFER_SIZE; i++) {
                Dev->Data.Spadata.RefGoodSpadMap[i] =
                    NvmRefGoodSpadMap[i];
            }
        }
    }

    if (((option & 2) == 2) &&

```

```

((ReadDataFromDeviceDone & 2) == 0)) {

VL53L0X_SETDEVICESTANDARDPARAMETER(Dev,

                                ModuleId, ModuleId);

VL53L0X_SETDEVICESTANDARDPARAMETER(Dev,

                                Revision, Revision);

ProductId_tmp = VL53L0X_GETDEVICESTANDARDPARAMETER(Dev,

                                ProductId);

VL53L0X_COPYSTRING(ProductId_tmp, ProductId);

}

if (((option & 4) == 4) &&

    ((ReadDataFromDeviceDone & 4) == 0)) {

VL53L0X_SETDEVICESTANDARDPARAMETER(Dev,

                                PartUIDUpper, PartUIDUpper);

VL53L0X_SETDEVICESTANDARDPARAMETER(Dev,

                                PartUIDLower, PartUIDLower);

SignalRateMeasFixed400mmFix =

VL53L0X_FIXPOINT97TOFIXPOINT1616(

SignalRateMeasFixed1104_400_mm);

VL53L0X_SETDEVICESTANDARDPARAMETER(Dev,

SignalRateMeasFixed400mm,

SignalRateMeasFixed400mmFix);

```

```

        OffsetMicroMeters = 0;

        if (DistMeasFixed1104_400_mm != 0) {

            OffsetFixed1104_mm =

                DistMeasFixed1104_400_mm -

                DistMeasTgtFixed1104_mm;

            OffsetMicroMeters = (OffsetFixed1104_mm

                * 1000) >> 4;

            OffsetMicroMeters *= -1;

        }

        PALDevDataSet(Dev,

            Part2PartOffsetAdjustmentNVMMicroMeter,

            OffsetMicroMeters);

    }

    byte = (uint8_t)(ReadDataFromDeviceDone | option);

    VL53L0X_SETDEVICESPECIFICPARAMETER(Dev, ReadDataFromDeviceDone,

        byte);

}

LOG_FUNCTION_END(Status);

return Status;

}

```

```

uint32_t VL53L0X_calc_macro_period_ps(VL53L0X_DEV Dev, uint8_t vcsel_period_pclks)
{

    uint64_t PLL_period_ps;

    uint32_t macro_period_vclks;

    uint32_t macro_period_ps;

```

```

LOG_FUNCTION_START("");

/* The above calculation will produce rounding errors,
   therefore set fixed value
*/
PLL_period_ps = 1655;

macro_period_vclks = 2304;
macro_period_ps = (uint32_t)(macro_period_vclks
                             * vcsel_period_pclks * PLL_period_ps);

LOG_FUNCTION_END("");
return macro_period_ps;
}

uint16_t VL53L0X_encode_timeout(uint32_t timeout_macro_clks)
{
    /*!
     * Encode timeout in macro periods in (LSByte * 2^MSByte) + 1 format
     */

    uint16_t encoded_timeout = 0;
    uint32_t ls_byte = 0;
    uint16_t ms_byte = 0;

    if (timeout_macro_clks > 0) {
        ls_byte = timeout_macro_clks - 1;
    }

```

```

        while ((ls_byte & 0xFFFFF00) > 0) {
            ls_byte = ls_byte >> 1;
            ms_byte++;
        }

        encoded_timeout = (ms_byte << 8)
                        + (uint16_t) (ls_byte & 0x000000FF);
    }

    return encoded_timeout;
}

uint32_t VL53L0X_decode_timeout(uint16_t encoded_timeout)
{
    /*!
     * Decode 16-bit timeout register value - format (LSByte * 2^MSByte) + 1
     */

    uint32_t timeout_macro_clks = 0;

    timeout_macro_clks = ((uint32_t) (encoded_timeout & 0x00FF)
                        << (uint32_t) ((encoded_timeout & 0xFF00) >> 8)) + 1;

    return timeout_macro_clks;
}

/* To convert ms into register value */

```

```

uint32_t VL53L0X_calc_timeout_mclks(VL53L0X_DEV Dev,
    uint32_t timeout_period_us,
    uint8_t vcsel_period_pclks)
{
    uint32_t macro_period_ps;
    uint32_t macro_period_ns;
    uint32_t timeout_period_mclks = 0;

    macro_period_ps = VL53L0X_calc_macro_period_ps(Dev, vcsel_period_pclks);
    macro_period_ns = (macro_period_ps + 500) / 1000;

    timeout_period_mclks =
        (uint32_t) (((timeout_period_us * 1000)
            + (macro_period_ns / 2)) / macro_period_ns);

    return timeout_period_mclks;
}

```

/* To convert register value into us */

```

uint32_t VL53L0X_calc_timeout_us(VL53L0X_DEV Dev,
    uint16_t timeout_period_mclks,
    uint8_t vcsel_period_pclks)
{
    uint32_t macro_period_ps;
    uint32_t macro_period_ns;
    uint32_t actual_timeout_period_us = 0;

    macro_period_ps = VL53L0X_calc_macro_period_ps(Dev, vcsel_period_pclks);
    macro_period_ns = (macro_period_ps + 500) / 1000;

```

```

actual_timeout_period_us =
    ((timeout_period_mclks * macro_period_ns)
    + (macro_period_ns / 2)) / 1000;

return actual_timeout_period_us;
}

VL53L0X_Error get_sequence_step_timeout(VL53L0X_DEV Dev,
                                         VL53L0X_SequenceStepId SequenceStepId,
                                         uint32_t *pTimeOutMicroSecs)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t CurrentVCSELPulsePeriodPClk;
    uint8_t EncodedTimeOutByte = 0;
    uint32_t TimeoutMicroSeconds = 0;
    uint16_t PreRangeEncodedTimeOut = 0;
    uint16_t MsrcTimeOutMClks;
    uint16_t PreRangeTimeOutMClks;
    uint16_t FinalRangeTimeOutMClks = 0;
    uint16_t FinalRangeEncodedTimeOut;
    VL53L0X_SchedulerSequenceSteps_t SchedulerSequenceSteps;

    if ((SequenceStepId == VL53L0X_SEQUENCESTEP_TCC) ||
        (SequenceStepId == VL53L0X_SEQUENCESTEP_DSS) ||
        (SequenceStepId == VL53L0X_SEQUENCESTEP_MSRC)) {

        Status = VL53L0X_GetVcselPulsePeriod(Dev,

```

```

        VL53L0X_VCSEL_PERIOD_PRE_RANGE,
        &CurrentVCSELPulsePeriodPClk);

if (Status == VL53L0X_ERROR_NONE) {
    Status = VL53L0X_RdByte(Dev,
        VL53L0X_REG_MSRC_CONFIG_TIMEOUT_MACROP,
        &EncodedTimeOutByte);
}

MsrcTimeOutMClks = VL53L0X_decode_timeout(EncodedTimeOutByte);

TimeoutMicroSeconds = VL53L0X_calc_timeout_us(Dev,
    MsrcTimeOutMClks,
    CurrentVCSELPulsePeriodPClk);
} else if (SequenceStepId == VL53L0X_SEQUENCESTEP_PRE_RANGE) {
    /* Retrieve PRE-RANGE VCSEL Period */
    Status = VL53L0X_GetVcselPulsePeriod(Dev,
        VL53L0X_VCSEL_PERIOD_PRE_RANGE,
        &CurrentVCSELPulsePeriodPClk);

    /* Retrieve PRE-RANGE Timeout in Macro periods (MCLKS) */
    if (Status == VL53L0X_ERROR_NONE) {

        /* Retrieve PRE-RANGE VCSEL Period */
        Status = VL53L0X_GetVcselPulsePeriod(Dev,
            VL53L0X_VCSEL_PERIOD_PRE_RANGE,
            &CurrentVCSELPulsePeriodPClk);

        if (Status == VL53L0X_ERROR_NONE) {
            Status = VL53L0X_RdWord(Dev,
                VL53L0X_REG_PRE_RANGE_CONFIG_TIMEOUT_MACROP_HI,

```

```

        &PreRangeEncodedTimeOut);
    }

    PreRangeTimeOutMClks = VL53L0X_decode_timeout(
        PreRangeEncodedTimeOut);

    TimeoutMicroSeconds = VL53L0X_calc_timeout_us(Dev,
        PreRangeTimeOutMClks,
        CurrentVCSELPulsePeriodPClk);
}

} else if (SequenceStepId == VL53L0X_SEQUENCESTEP_FINAL_RANGE) {

    VL53L0X_GetSequenceStepEnables(Dev, &SchedulerSequenceSteps);
    PreRangeTimeOutMClks = 0;

    if (SchedulerSequenceSteps.PreRangeOn) {
        /* Retrieve PRE-RANGE VCSEL Period */
        Status = VL53L0X_GetVcselPulsePeriod(Dev,
            VL53L0X_VCSEL_PERIOD_PRE_RANGE,
            &CurrentVCSELPulsePeriodPClk);

        /* Retrieve PRE-RANGE Timeout in Macro periods
        * (MCLKS) */
        if (Status == VL53L0X_ERROR_NONE) {
            Status = VL53L0X_RdWord(Dev,
                VL53L0X_REG_PRE_RANGE_CONFIG_TIMEOUT_MACROP_HI,
                &PreRangeEncodedTimeOut);
            PreRangeTimeOutMClks = VL53L0X_decode_timeout(
                PreRangeEncodedTimeOut);

```

```

    }

}

if (Status == VL53L0X_ERROR_NONE) {

    /* Retrieve FINAL-RANGE VCSEL Period */

    Status = VL53L0X_GetVcselPulsePeriod(Dev,

        VL53L0X_VCSEL_PERIOD_FINAL_RANGE,

        &CurrentVCSELPulsePeriodPClk);

}

/* Retrieve FINAL-RANGE Timeout in Macro periods (MCLKS) */
if (Status == VL53L0X_ERROR_NONE) {

    Status = VL53L0X_RdWord(Dev,

        VL53L0X_REG_FINAL_RANGE_CONFIG_TIMEOUT_MACROP_HI,

        &FinalRangeEncodedTimeout);

    FinalRangeTimeoutMClks = VL53L0X_decode_timeout(

        FinalRangeEncodedTimeout);

}

FinalRangeTimeoutMClks -= PreRangeTimeoutMClks;

TimeoutMicroSeconds = VL53L0X_calc_timeout_us(Dev,

    FinalRangeTimeoutMClks,

    CurrentVCSELPulsePeriodPClk);

}

*pTimeOutMicroSecs = TimeoutMicroSeconds;

return Status;

}

```

```

VL53L0X_Error set_sequence_step_timeout(VL53L0X_DEV Dev,
                                         VL53L0X_SequenceStepId SequenceStepId,
                                         uint32_t TimeOutMicroSecs)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t CurrentVCSELPulsePeriodPClk;
    uint8_t MsrcEncodedTimeOut;
    uint16_t PreRangeEncodedTimeOut;
    uint16_t PreRangeTimeOutMClks;
    uint16_t MsrcRangeTimeOutMClks;
    uint16_t FinalRangeTimeOutMClks;
    uint16_t FinalRangeEncodedTimeOut;
    VL53L0X_SchedulerSequenceSteps_t SchedulerSequenceSteps;

    if ((SequenceStepId == VL53L0X_SEQUENCESTEP_TCC) ||
        (SequenceStepId == VL53L0X_SEQUENCESTEP_DSS) ||
        (SequenceStepId == VL53L0X_SEQUENCESTEP_MSRC)) {

        Status = VL53L0X_GetVcselPulsePeriod(Dev,
                                              VL53L0X_VCSEL_PERIOD_PRE_RANGE,
                                              &CurrentVCSELPulsePeriodPClk);

        if (Status == VL53L0X_ERROR_NONE) {
            MsrcRangeTimeOutMClks = VL53L0X_calc_timeout_mclks(Dev,
                                                                TimeOutMicroSecs,
                                                                (uint8_t)CurrentVCSELPulsePeriodPClk);

```

```

        if (MsrcRangeTimeOutMClks > 256)
            MsrcEncodedTimeOut = 255;
        else
            MsrcEncodedTimeOut =
                (uint8_t)MsrcRangeTimeOutMClks - 1;

        VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
            LastEncodedTimeout,
            MsrcEncodedTimeOut);
    }

    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_WrByte(Dev,
            VL53L0X_REG_MSRC_CONFIG_TIMEOUT_MACROP,
            MsrcEncodedTimeOut);
    }
} else {

    if (SequenceStepId == VL53L0X_SEQUENCESTEP_PRE_RANGE) {

        if (Status == VL53L0X_ERROR_NONE) {
            Status = VL53L0X_GetVcselPulsePeriod(Dev,
                VL53L0X_VCSEL_PERIOD_PRE_RANGE,
                &CurrentVCSELPulsePeriodPClk);

            PreRangeTimeOutMClks =
                VL53L0X_calc_timeout_mclks(Dev,
                    TimeOutMicroSecs,
                    (uint8_t)CurrentVCSELPulsePeriodPClk);

            PreRangeEncodedTimeOut = VL53L0X_encode_timeout(

```

```

        PreRangeTimeOutMClks);

VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
        LastEncodedTimeout,
        PreRangeEncodedTimeOut);
}

if (Status == VL53L0X_ERROR_NONE) {
    Status = VL53L0X_WrWord(Dev,
        VL53L0X_REG_PRE_RANGE_CONFIG_TIMEOUT_MACROP_HI,
        PreRangeEncodedTimeOut);
}

if (Status == VL53L0X_ERROR_NONE) {
    VL53L0X_SETDEVICESPECIFICPARAMETER(
        Dev,
        PreRangeTimeoutMicroSecs,
        TimeOutMicroSecs);
}
} else if (SequenceStepId == VL53L0X_SEQUENCESTEP_FINAL_RANGE) {

    /* For the final range timeout, the pre-range timeout
    * must be added. To do this both final and pre-range
    * timeouts must be expressed in macro periods MClks
    * because they have different vcsel periods.
    */

    VL53L0X_GetSequenceStepEnables(Dev,
        &SchedulerSequenceSteps);

```

```

PreRangeTimeOutMClks = 0;

if (SchedulerSequenceSteps.PreRangeOn) {

    /* Retrieve PRE-RANGE VCSEL Period */
    Status = VL53L0X_GetVcselPulsePeriod(Dev,
        VL53L0X_VCSEL_PERIOD_PRE_RANGE,
        &CurrentVCSELPulsePeriodPClk);

    /* Retrieve PRE-RANGE Timeout in Macro periods
    * (MCLKS) */
    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_RdWord(Dev, 0x51,
            &PreRangeEncodedTimeOut);
        PreRangeTimeOutMClks =
            VL53L0X_decode_timeout(
                PreRangeEncodedTimeOut);
    }
}

/* Calculate FINAL RANGE Timeout in Macro Periods
* (MCLKS) and add PRE-RANGE value
*/
if (Status == VL53L0X_ERROR_NONE) {

    Status = VL53L0X_GetVcselPulsePeriod(Dev,
        VL53L0X_VCSEL_PERIOD_FINAL_RANGE,
        &CurrentVCSELPulsePeriodPClk);
}

if (Status == VL53L0X_ERROR_NONE) {

```

```

        FinalRangeTimeOutMClks =
            VL53L0X_calc_timeout_mclks(Dev,
            TimeOutMicroSecs,
            (uint8_t) CurrentVCSELPulsePeriodPClk);

        FinalRangeTimeOutMClks += PreRangeTimeOutMClks;

        FinalRangeEncodedTimeOut =
            VL53L0X_encode_timeout(FinalRangeTimeOutMClks);

        if (Status == VL53L0X_ERROR_NONE) {
            Status = VL53L0X_WrWord(Dev, 0x71,
            FinalRangeEncodedTimeOut);
        }

        if (Status == VL53L0X_ERROR_NONE) {
            VL53L0X_SETDEVICESPECIFICPARAMETER(
                Dev,
                FinalRangeTimeoutMicroSecs,
                TimeOutMicroSecs);
        }
    }
    } else
        Status = VL53L0X_ERROR_INVALID_PARAMS;

    }

    return Status;
}

```

```

VL53L0X_Error VL53L0X_set_vcsel_pulse_period(VL53L0X_DEV Dev,
        VL53L0X_VcselPeriod VcselPeriodType, uint8_t VCSELPulsePeriodPCLK)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t vcsel_period_reg;

    uint8_t MinPreVcselPeriodPCLK = 12;
    uint8_t MaxPreVcselPeriodPCLK = 18;
    uint8_t MinFinalVcselPeriodPCLK = 8;
    uint8_t MaxFinalVcselPeriodPCLK = 14;
    uint32_t MeasurementTimingBudgetMicroSeconds;
    uint32_t FinalRangeTimeoutMicroSeconds;
    uint32_t PreRangeTimeoutMicroSeconds;
    uint32_t MsrcTimeoutMicroSeconds;
    uint8_t PhaseCalInt = 0;

    /* Check if valid clock period requested */

    if ((VCSELPulsePeriodPCLK % 2) != 0) {
        /* Value must be an even number */
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    } else if (VcselPeriodType == VL53L0X_VCSEL_PERIOD_PRE_RANGE &&
        (VCSELPulsePeriodPCLK < MinPreVcselPeriodPCLK ||
        VCSELPulsePeriodPCLK > MaxPreVcselPeriodPCLK)) {
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    } else if (VcselPeriodType == VL53L0X_VCSEL_PERIOD_FINAL_RANGE &&
        (VCSELPulsePeriodPCLK < MinFinalVcselPeriodPCLK ||
        VCSELPulsePeriodPCLK > MaxFinalVcselPeriodPCLK)) {

```

```

        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }

    /* Apply specific settings for the requested clock period */

    if (Status != VL53L0X_ERROR_NONE)
        return Status;

    if (VcselPeriodType == VL53L0X_VCSEL_PERIOD_PRE_RANGE) {

        /* Set phase check limits */
        if (VCSELPulsePeriodPCLK == 12) {

            Status = VL53L0X_WrByte(Dev,
                                    VL53L0X_REG_PRE_RANGE_CONFIG_VALID_PHASE_HIGH,
                                    0x18);

            Status = VL53L0X_WrByte(Dev,
                                    VL53L0X_REG_PRE_RANGE_CONFIG_VALID_PHASE_LOW,
                                    0x08);
        } else if (VCSELPulsePeriodPCLK == 14) {

            Status = VL53L0X_WrByte(Dev,
                                    VL53L0X_REG_PRE_RANGE_CONFIG_VALID_PHASE_HIGH,
                                    0x30);

            Status = VL53L0X_WrByte(Dev,
                                    VL53L0X_REG_PRE_RANGE_CONFIG_VALID_PHASE_LOW,
                                    0x08);
        } else if (VCSELPulsePeriodPCLK == 16) {

```

```

        Status = VL53L0X_WrByte(Dev,
                                VL53L0X_REG_PRE_RANGE_CONFIG_VALID_PHASE_HIGH,
                                0x40);

        Status = VL53L0X_WrByte(Dev,
                                VL53L0X_REG_PRE_RANGE_CONFIG_VALID_PHASE_LOW,
                                0x08);
    } else if (VCSELPulsePeriodPCLK == 18) {

        Status = VL53L0X_WrByte(Dev,
                                VL53L0X_REG_PRE_RANGE_CONFIG_VALID_PHASE_HIGH,
                                0x50);

        Status = VL53L0X_WrByte(Dev,
                                VL53L0X_REG_PRE_RANGE_CONFIG_VALID_PHASE_LOW,
                                0x08);

    }

} else if (VcselPeriodType == VL53L0X_VCSEL_PERIOD_FINAL_RANGE) {

    if (VCSELPulsePeriodPCLK == 8) {

        Status = VL53L0X_WrByte(Dev,
                                VL53L0X_REG_FINAL_RANGE_CONFIG_VALID_PHASE_HIGH,
                                0x10);

        Status = VL53L0X_WrByte(Dev,
                                VL53L0X_REG_FINAL_RANGE_CONFIG_VALID_PHASE_LOW,
                                0x08);

        Status |= VL53L0X_WrByte(Dev,
                                VL53L0X_REG_GLOBAL_CONFIG_VCSEL_WIDTH, 0x02);
    }
}

```

```

        Status |= VL53L0X_WrByte(Dev,
                                   VL53L0X_REG_ALGO_PHASECAL_CONFIG_TIMEOUT, 0x0C);

        Status |= VL53L0X_WrByte(Dev, 0xff, 0x01);
        Status |= VL53L0X_WrByte(Dev,
                                   VL53L0X_REG_ALGO_PHASECAL_LIM,
                                   0x30);
        Status |= VL53L0X_WrByte(Dev, 0xff, 0x00);
    } else if (VCSELPulsePeriodPCLK == 10) {

        Status = VL53L0X_WrByte(Dev,
                                   VL53L0X_REG_FINAL_RANGE_CONFIG_VALID_PHASE_HIGH,
                                   0x28);
        Status = VL53L0X_WrByte(Dev,
                                   VL53L0X_REG_FINAL_RANGE_CONFIG_VALID_PHASE_LOW,
                                   0x08);

        Status |= VL53L0X_WrByte(Dev,
                                   VL53L0X_REG_GLOBAL_CONFIG_VCSEL_WIDTH, 0x03);
        Status |= VL53L0X_WrByte(Dev,
                                   VL53L0X_REG_ALGO_PHASECAL_CONFIG_TIMEOUT, 0x09);

        Status |= VL53L0X_WrByte(Dev, 0xff, 0x01);
        Status |= VL53L0X_WrByte(Dev,
                                   VL53L0X_REG_ALGO_PHASECAL_LIM,
                                   0x20);
        Status |= VL53L0X_WrByte(Dev, 0xff, 0x00);
    } else if (VCSELPulsePeriodPCLK == 12) {

```

```

Status = VL53L0X_WrByte(Dev,
    VL53L0X_REG_FINAL_RANGE_CONFIG_VALID_PHASE_HIGH,
    0x38);

Status = VL53L0X_WrByte(Dev,
    VL53L0X_REG_FINAL_RANGE_CONFIG_VALID_PHASE_LOW,
    0x08);

Status |= VL53L0X_WrByte(Dev,
    VL53L0X_REG_GLOBAL_CONFIG_VCSEL_WIDTH, 0x03);

Status |= VL53L0X_WrByte(Dev,
    VL53L0X_REG_ALGO_PHASECAL_CONFIG_TIMEOUT, 0x08);

Status |= VL53L0X_WrByte(Dev, 0xff, 0x01);

Status |= VL53L0X_WrByte(Dev,
    VL53L0X_REG_ALGO_PHASECAL_LIM,
    0x20);

Status |= VL53L0X_WrByte(Dev, 0xff, 0x00);
} else if (VCSELPulsePeriodPCLK == 14) {

    Status = VL53L0X_WrByte(Dev,
        VL53L0X_REG_FINAL_RANGE_CONFIG_VALID_PHASE_HIGH,
        0x048);

    Status = VL53L0X_WrByte(Dev,
        VL53L0X_REG_FINAL_RANGE_CONFIG_VALID_PHASE_LOW,
        0x08);

    Status |= VL53L0X_WrByte(Dev,
        VL53L0X_REG_GLOBAL_CONFIG_VCSEL_WIDTH, 0x03);

    Status |= VL53L0X_WrByte(Dev,

```

```

        VL53L0X_REG_ALGO_PHASECAL_CONFIG_TIMEOUT, 0x07);

    Status |= VL53L0X_WrByte(Dev, 0xff, 0x01);

    Status |= VL53L0X_WrByte(Dev,
        VL53L0X_REG_ALGO_PHASECAL_LIM,
        0x20);

    Status |= VL53L0X_WrByte(Dev, 0xff, 0x00);

}

}

```

/* Re-calculate and apply timeouts, in macro periods */

```

if (Status == VL53L0X_ERROR_NONE) {
    vcsel_period_reg = VL53L0X_encode_vcsel_period((uint8_t)
        VCSELPulsePeriodPCLK);

    /* When the VCSEL period for the pre or final range is changed,
     * the corresponding timeout must be read from the device using
     * the current VCSEL period, then the new VCSEL period can be
     * applied. The timeout then must be written back to the device
     * using the new VCSEL period.
     *
     * For the MSRC timeout, the same applies - this timeout being
     * dependant on the pre-range vcsel period.
     */
    switch (VcselPeriodType) {
    case VL53L0X_VCSEL_PERIOD_PRE_RANGE:
        Status = get_sequence_step_timeout(Dev,

```

```
VL53L0X_SEQUENCESTEP_PRE_RANGE,  
&PreRangeTimeoutMicroSeconds);
```

```
if (Status == VL53L0X_ERROR_NONE)  
    Status = get_sequence_step_timeout(Dev,  
                                       VL53L0X_SEQUENCESTEP_MSRC,  
                                       &MsrcTimeoutMicroSeconds);
```

```
if (Status == VL53L0X_ERROR_NONE)  
    Status = VL53L0X_WrByte(Dev,  
                           VL53L0X_REG_PRE_RANGE_CONFIG_VCSEL_PERIOD,  
                           vcsel_period_reg);
```

```
if (Status == VL53L0X_ERROR_NONE)  
    Status = set_sequence_step_timeout(Dev,  
                                       VL53L0X_SEQUENCESTEP_PRE_RANGE,  
                                       PreRangeTimeoutMicroSeconds);
```

```
if (Status == VL53L0X_ERROR_NONE)  
    Status = set_sequence_step_timeout(Dev,  
                                       VL53L0X_SEQUENCESTEP_MSRC,  
                                       MsrcTimeoutMicroSeconds);
```

```
VL53L0X_SETDEVICESPECIFICPARAMETER(  
    Dev,  
    PreRangeVcselPulsePeriod,  
    VCSELPulsePeriodPCLK);
```

```

        break;
case VL53L0X_VCSEL_PERIOD_FINAL_RANGE:
    Status = get_sequence_step_timeout(Dev,
        VL53L0X_SEQUENCESTEP_FINAL_RANGE,
        &FinalRangeTimeoutMicroSeconds);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_WrByte(Dev,
            VL53L0X_REG_FINAL_RANGE_CONFIG_VCSEL_PERIOD,
            vcsel_period_reg);

    if (Status == VL53L0X_ERROR_NONE)
        Status = set_sequence_step_timeout(Dev,
            VL53L0X_SEQUENCESTEP_FINAL_RANGE,
            FinalRangeTimeoutMicroSeconds);

    VL53L0X_SETDEVICESPECIFICPARAMETER(
        Dev,
        FinalRangeVcselPulsePeriod,
        VCSELPulsePeriodPCLK);
    break;
default:
    Status = VL53L0X_ERROR_INVALID_PARAMS;
}
}

/* Finally, the timing budget must be re-applied */
if (Status == VL53L0X_ERROR_NONE) {

```



```

        break;

    case VL53L0X_VCSEL_PERIOD_FINAL_RANGE:

        Status = VL53L0X_RdByte(Dev,

                                VL53L0X_REG_FINAL_RANGE_CONFIG_VCSEL_PERIOD,

                                &vcsel_period_reg);

        break;

    default:

        Status = VL53L0X_ERROR_INVALID_PARAMS;

    }

    if (Status == VL53L0X_ERROR_NONE)

        *pVCSELPulsePeriodPCLK =

            VL53L0X_decode_vcsel_period(vcsel_period_reg);

    return Status;

}

VL53L0X_Error VL53L0X_set_measurement_timing_budget_micro_seconds(VL53L0X_DEV Dev,

                                                                    uint32_t MeasurementTimingBudgetMicroSeconds)

{

    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint32_t FinalRangeTimingBudgetMicroSeconds;

    VL53L0X_SchedulerSequenceSteps_t SchedulerSequenceSteps;

    uint32_t MsrcDccTccTimeoutMicroSeconds    = 2000;

    uint32_t StartOverheadMicroSeconds         = 1320;

    uint32_t EndOverheadMicroSeconds           = 960;

    uint32_t MsrcOverheadMicroSeconds          = 660;

```

```

uint32_t TccOverheadMicroSeconds      = 590;
uint32_t DssOverheadMicroSeconds      = 690;
uint32_t PreRangeOverheadMicroSeconds = 660;
uint32_t FinalRangeOverheadMicroSeconds = 550;
uint32_t PreRangeTimeoutMicroSeconds  = 0;
uint32_t cMinTimingBudgetMicroSeconds = 20000;
uint32_t SubTimeout = 0;

```

```

LOG_FUNCTION_START("");

```

```

if (MeasurementTimingBudgetMicroSeconds
    < cMinTimingBudgetMicroSeconds) {
    Status = VL53L0X_ERROR_INVALID_PARAMS;
    return Status;
}

```

```

FinalRangeTimingBudgetMicroSeconds =
    MeasurementTimingBudgetMicroSeconds -
    (StartOverheadMicroSeconds + EndOverheadMicroSeconds);

```

```

Status = VL53L0X_GetSequenceStepEnables(Dev, &SchedulerSequenceSteps);

```

```

if (Status == VL53L0X_ERROR_NONE &&
    (SchedulerSequenceSteps.TccOn ||
     SchedulerSequenceSteps.MsrcOn ||
     SchedulerSequenceSteps.DssOn)) {

    /* TCC, MSRC and DSS all share the same timeout */
    Status = get_sequence_step_timeout(Dev,

```

```

        VL53L0X_SEQUENCESTEP_MSRC,
        &MsrcDccTccTimeoutMicroSeconds);

/* Subtract the TCC, MSRC and DSS timeouts if they are
 * enabled. */

if (Status != VL53L0X_ERROR_NONE)
    return Status;

/* TCC */
if (SchedulerSequenceSteps.TccOn) {

    SubTimeout = MsrcDccTccTimeoutMicroSeconds
        + TccOverheadMicroSeconds;

    if (SubTimeout <
        FinalRangeTimingBudgetMicroSeconds) {
        FinalRangeTimingBudgetMicroSeconds -=
            SubTimeout;
    } else {
        /* Requested timeout too big. */
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
}

if (Status != VL53L0X_ERROR_NONE) {
    LOG_FUNCTION_END(Status);
    return Status;
}

```

```

/* DSS */
if (SchedulerSequenceSteps.DssOn) {

    SubTimeout = 2 * (MsrcDccTccTimeoutMicroSeconds +
        DssOverheadMicroSeconds);

    if (SubTimeout < FinalRangeTimingBudgetMicroSeconds) {
        FinalRangeTimingBudgetMicroSeconds
            -= SubTimeout;
    } else {
        /* Requested timeout too big. */
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
} else if (SchedulerSequenceSteps.MsrcOn) {
    /* MSRC */
    SubTimeout = MsrcDccTccTimeoutMicroSeconds +
        MsrcOverheadMicroSeconds;

    if (SubTimeout < FinalRangeTimingBudgetMicroSeconds) {
        FinalRangeTimingBudgetMicroSeconds
            -= SubTimeout;
    } else {
        /* Requested timeout too big. */
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
}
}

```

```

if (Status != VL53L0X_ERROR_NONE) {
    LOG_FUNCTION_END(Status);
    return Status;
}

if (SchedulerSequenceSteps.PreRangeOn) {

    /* Subtract the Pre-range timeout if enabled. */

    Status = get_sequence_step_timeout(Dev,
                                       VL53L0X_SEQUENCESTEP_PRE_RANGE,
                                       &PreRangeTimeoutMicroSeconds);

    SubTimeout = PreRangeTimeoutMicroSeconds +
                 PreRangeOverheadMicroSeconds;

    if (SubTimeout < FinalRangeTimingBudgetMicroSeconds) {
        FinalRangeTimingBudgetMicroSeconds -= SubTimeout;
    } else {
        /* Requested timeout too big. */
        Status = VL53L0X_ERROR_INVALID_PARAMS;
    }
}

if (Status == VL53L0X_ERROR_NONE &&
    SchedulerSequenceSteps.FinalRangeOn) {

```

```

        FinalRangeTimingBudgetMicroSeconds -=
            FinalRangeOverheadMicroSeconds;

    /* Final Range Timeout
    * Note that the final range timeout is determined by the timing
    * budget and the sum of all other timeouts within the sequence.
    * If there is no room for the final range timeout, then an error
    * will be set. Otherwise the remaining time will be applied to
    * the final range.
    */
    Status = set_sequence_step_timeout(Dev,
        VL53L0X_SEQUENCESTEP_FINAL_RANGE,
        FinalRangeTimingBudgetMicroSeconds);

    VL53L0X_SETPARAMETERFIELD(Dev,
        MeasurementTimingBudgetMicroSeconds,
        MeasurementTimingBudgetMicroSeconds);
}

LOG_FUNCTION_END(Status);

return Status;
}

VL53L0X_Error VL53L0X_get_measurement_timing_budget_micro_seconds(VL53L0X_DEV Dev,
    uint32_t *pMeasurementTimingBudgetMicroSeconds)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    VL53L0X_SchedulerSequenceSteps_t SchedulerSequenceSteps;

```

```

uint32_t FinalRangeTimeoutMicroSeconds;

uint32_t MsrcDccTccTimeoutMicroSeconds    = 2000;

uint32_t StartOverheadMicroSeconds        = 1910;

uint32_t EndOverheadMicroSeconds           = 960;

uint32_t MsrcOverheadMicroSeconds          = 660;

uint32_t TccOverheadMicroSeconds           = 590;

uint32_t DssOverheadMicroSeconds           = 690;

uint32_t PreRangeOverheadMicroSeconds      = 660;

uint32_t FinalRangeOverheadMicroSeconds = 550;

uint32_t PreRangeTimeoutMicroSeconds       = 0;


LOG_FUNCTION_START("");


/* Start and end overhead times always present */
*pMeasurementTimingBudgetMicroSeconds
    = StartOverheadMicroSeconds + EndOverheadMicroSeconds;


Status = VL53L0X_GetSequenceStepEnables(Dev, &SchedulerSequenceSteps);


if (Status != VL53L0X_ERROR_NONE) {
    LOG_FUNCTION_END(Status);
    return Status;
}


if (SchedulerSequenceSteps.TccOn ||
    SchedulerSequenceSteps.MsrcOn ||
    SchedulerSequenceSteps.DssOn) {

```

```

Status = get_sequence_step_timeout(Dev,
                                    VL53L0X_SEQUENCESTEP_MSRC,
                                    &MsrcDccTccTimeoutMicroSeconds);

if (Status == VL53L0X_ERROR_NONE) {
    if (SchedulerSequenceSteps.TccOn) {
        *pMeasurementTimingBudgetMicroSeconds +=
            MsrcDccTccTimeoutMicroSeconds +
            TccOverheadMicroSeconds;
    }

    if (SchedulerSequenceSteps.DssOn) {
        *pMeasurementTimingBudgetMicroSeconds +=
            2 * (MsrcDccTccTimeoutMicroSeconds +
                DssOverheadMicroSeconds);
    } else if (SchedulerSequenceSteps.MsrcOn) {
        *pMeasurementTimingBudgetMicroSeconds +=
            MsrcDccTccTimeoutMicroSeconds +
            MsrcOverheadMicroSeconds;
    }
}
}

```

```

if (Status == VL53L0X_ERROR_NONE) {
    if (SchedulerSequenceSteps.PreRangeOn) {
        Status = get_sequence_step_timeout(Dev,
                                            VL53L0X_SEQUENCESTEP_PRE_RANGE,
                                            &PreRangeTimeoutMicroSeconds);
        *pMeasurementTimingBudgetMicroSeconds +=

```

```

        PreRangeTimeoutMicroSeconds +
        PreRangeOverheadMicroSeconds;
    }
}

if (Status == VL53L0X_ERROR_NONE) {
    if (SchedulerSequenceSteps.FinalRangeOn) {
        Status = get_sequence_step_timeout(Drv,
            VL53L0X_SEQUENCESTEP_FINAL_RANGE,
            &FinalRangeTimeoutMicroSeconds);

        *pMeasurementTimingBudgetMicroSeconds +=
            (FinalRangeTimeoutMicroSeconds +
            FinalRangeOverheadMicroSeconds);
    }
}

if (Status == VL53L0X_ERROR_NONE) {
    VL53L0X_SETPARAMETERFIELD(Drv,
        MeasurementTimingBudgetMicroSeconds,
        *pMeasurementTimingBudgetMicroSeconds);
}

LOG_FUNCTION_END(Status);
return Status;
}

```

```

VL53L0X_Error VL53L0X_load_tuning_settings(VL53L0X_DEV Dev,

```

```

        uint8_t *pTuningSettingBuffer)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    int i;

    int Index;

    uint8_t msb;

    uint8_t lsb;

    uint8_t SelectParam;

    uint8_t NumberOfWrites;

    uint8_t Address;

    uint8_t localBuffer[4]; /* max */

    uint16_t Temp16;

    LOG_FUNCTION_START("");

    Index = 0;

    while ((*pTuningSettingBuffer + Index) != 0) &&
        (Status == VL53L0X_ERROR_NONE) {
        NumberOfWrites = *(pTuningSettingBuffer + Index);
        Index++;
        if (NumberOfWrites == 0xFF) {
            /* internal parameters */
            SelectParam = *(pTuningSettingBuffer + Index);
            Index++;
            switch (SelectParam) {
            case 0: /* uint16_t SigmaEstRefArray -> 2 bytes */
                msb = *(pTuningSettingBuffer + Index);
                Index++;

```

```

        lsb = *(pTuningSettingBuffer + Index);

        Index++;

        Temp16 = VL53L0X_MAKEUINT16(lsb, msb);

        PALDevDataSet(Dev, SigmaEstRefArray, Temp16);

        break;
case 1: /* uint16_t SigmaEstEffPulseWidth -> 2 bytes */
        msb = *(pTuningSettingBuffer + Index);

        Index++;

        lsb = *(pTuningSettingBuffer + Index);

        Index++;

        Temp16 = VL53L0X_MAKEUINT16(lsb, msb);

        PALDevDataSet(Dev, SigmaEstEffPulseWidth,
                        Temp16);

        break;
case 2: /* uint16_t SigmaEstEffAmbWidth -> 2 bytes */
        msb = *(pTuningSettingBuffer + Index);

        Index++;

        lsb = *(pTuningSettingBuffer + Index);

        Index++;

        Temp16 = VL53L0X_MAKEUINT16(lsb, msb);

        PALDevDataSet(Dev, SigmaEstEffAmbWidth, Temp16);

        break;
case 3: /* uint16_t targetRefRate -> 2 bytes */
        msb = *(pTuningSettingBuffer + Index);

        Index++;

        lsb = *(pTuningSettingBuffer + Index);

        Index++;

        Temp16 = VL53L0X_MAKEUINT16(lsb, msb);

        PALDevDataSet(Dev, targetRefRate, Temp16);

```

```

        break;

default: /* invalid parameter */
    Status = VL53L0X_ERROR_INVALID_PARAMS;
}

} else if (NumberOfWrites <= 4) {
    Address = *(pTuningSettingBuffer + Index);
    Index++;

    for (i = 0; i < NumberOfWrites; i++) {
        localBuffer[i] = *(pTuningSettingBuffer +
                           Index);
        Index++;
    }

    Status = VL53L0X_WriteMulti(Dev, Address, localBuffer,
                                NumberOfWrites);

} else {
    Status = VL53L0X_ERROR_INVALID_PARAMS;
}

}

LOG_FUNCTION_END(Status);

return Status;
}

VL53L0X_Error VL53L0X_get_total_xtalk_rate(VL53L0X_DEV Dev,
      VL53L0X_RangingMeasurementData_t *pRangingMeasurementData,

```

```

FixPoint1616_t *ptotal_xtalk_rate_mcps)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t xtalkCompEnable;
    FixPoint1616_t totalXtalkMegaCps;
    FixPoint1616_t xtalkPerSpadMegaCps;

    *ptotal_xtalk_rate_mcps = 0;

    Status = VL53L0X_GetXTalkCompensationEnable(Dev, &xtalkCompEnable);
    if (Status == VL53L0X_ERROR_NONE) {

        if (xtalkCompEnable) {

            VL53L0X_GETPARAMETERFIELD(
                Dev,
                XTalkCompensationRateMegaCps,
                xtalkPerSpadMegaCps);

            /* FixPoint1616 * FixPoint 8:8 = FixPoint0824 */
            totalXtalkMegaCps =
                pRangingMeasurementData->EffectiveSpadRtnCount *
                xtalkPerSpadMegaCps;

            /* FixPoint0824 >> 8 = FixPoint1616 */
            *ptotal_xtalk_rate_mcps =
                (totalXtalkMegaCps + 0x80) >> 8;
        }
    }
}

```

```

    }

    return Status;
}

VL53L0X_Error VL53L0X_get_total_signal_rate(VL53L0X_DEV Dev,
    VL53L0X_RangingMeasurementData_t *pRangingMeasurementData,
    FixPoint1616_t *ptotal_signal_rate_mcps)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    FixPoint1616_t totalXtalkMegaCps;

    LOG_FUNCTION_START("");

    *ptotal_signal_rate_mcps =
        pRangingMeasurementData->SignalRateRtnMegaCps;

    Status = VL53L0X_get_total_xtalk_rate(
        Dev, pRangingMeasurementData, &totalXtalkMegaCps);

    if (Status == VL53L0X_ERROR_NONE)
        *ptotal_signal_rate_mcps += totalXtalkMegaCps;

    return Status;
}

VL53L0X_Error VL53L0X_calc_dmax(
    VL53L0X_DEV Dev,
    FixPoint1616_t totalSignalRate_mcps,

```

```

FixPoint1616_t totalCorrSignalRate_mcps,

FixPoint1616_t pwMult,

uint32_t sigmaEstimateP1,

FixPoint1616_t sigmaEstimateP2,

uint32_t peakVcseIDuration_us,

uint32_t *pdmax_mm)

{

    const uint32_t cSigmaLimit          = 18;

    const FixPoint1616_t cSignalLimit    = 0x4000; /* 0.25 */

    const FixPoint1616_t cSigmaEstRef    = 0x00000042; /* 0.001 */

    const uint32_t cAmbEffWidthSigmaEst_ns = 6;

    const uint32_t cAmbEffWidthDMax_ns   = 7;

    uint32_t dmaxCalRange_mm;

    FixPoint1616_t dmaxCalSignalRateRtn_mcps;

    FixPoint1616_t minSignalNeeded;

    FixPoint1616_t minSignalNeeded_p1;

    FixPoint1616_t minSignalNeeded_p2;

    FixPoint1616_t minSignalNeeded_p3;

    FixPoint1616_t minSignalNeeded_p4;

    FixPoint1616_t sigmaLimitTmp;

    FixPoint1616_t sigmaEstSqTmp;

    FixPoint1616_t signalLimitTmp;

    FixPoint1616_t SignalAt0mm;

    FixPoint1616_t dmaxDark;

    FixPoint1616_t dmaxAmbient;

    FixPoint1616_t dmaxDarkTmp;

    FixPoint1616_t sigmaEstP2Tmp;

    uint32_t signalRateTemp_mcps;

```

```
VL53L0X_Error Status = VL53L0X_ERROR_NONE;
```

```
LOG_FUNCTION_START("");
```

```
dmaxCalRange_mm =
```

```
    PALDevDataGet(Dev, DmaxCalRangeMilliMeter);
```

```
dmaxCalSignalRateRtn_mcps =
```

```
    PALDevDataGet(Dev, DmaxCalSignalRateRtnMegaCps);
```

```
/* uint32 * FixPoint1616 = FixPoint1616 */
```

```
SignalAt0mm = dmaxCalRange_mm * dmaxCalSignalRateRtn_mcps;
```

```
/* FixPoint1616 >> 8 = FixPoint2408 */
```

```
SignalAt0mm = (SignalAt0mm + 0x80) >> 8;
```

```
SignalAt0mm *= dmaxCalRange_mm;
```

```
minSignalNeeded_p1 = 0;
```

```
if (totalCorrSignalRate_mcps > 0) {
```

```
    /* Shift by 10 bits to increase resolution prior to the
```

```
    * division */
```

```
    signalRateTemp_mcps = totalSignalRate_mcps << 10;
```

```
    /* Add rounding value prior to division */
```

```
    minSignalNeeded_p1 = signalRateTemp_mcps +
```

```
        (totalCorrSignalRate_mcps/2);
```

```
    /* FixPoint0626/FixPoint1616 = FixPoint2210 */
```

```

minSignalNeeded_p1 /= totalCorrSignalRate_mcps;

/* Apply a factored version of the speed of light.
Correction to be applied at the end */
minSignalNeeded_p1 *= 3;

/* FixPoint2210 * FixPoint2210 = FixPoint1220 */
minSignalNeeded_p1 *= minSignalNeeded_p1;

/* FixPoint1220 >> 16 = FixPoint2804 */
minSignalNeeded_p1 = (minSignalNeeded_p1 + 0x8000) >> 16;
}

```

```

minSignalNeeded_p2 = pwMult * sigmaEstimateP1;

```

```

/* FixPoint1616 >> 16 = uint32 */
minSignalNeeded_p2 = (minSignalNeeded_p2 + 0x8000) >> 16;

```

```

/* uint32 * uint32 = uint32 */
minSignalNeeded_p2 *= minSignalNeeded_p2;

```

```

/* Check sigmaEstimateP2
* If this value is too high there is not enough signal rate
* to calculate dmax value so set a suitable value to ensure
* a very small dmax.
*/
sigmaEstP2Tmp = (sigmaEstimateP2 + 0x8000) >> 16;
sigmaEstP2Tmp = (sigmaEstP2Tmp + cAmbEffWidthSigmaEst_ns/2)/
cAmbEffWidthSigmaEst_ns;

```

```

sigmaEstP2Tmp *= cAmbEffWidthDMax_ns;

if (sigmaEstP2Tmp > 0xffff) {
    minSignalNeeded_p3 = 0xffff00000;
} else {

    /* DMAX uses a different ambient width from sigma, so apply
    * correction.
    * Perform division before multiplication to prevent overflow.
    */
    sigmaEstimateP2 = (sigmaEstimateP2 + cAmbEffWidthSigmaEst_ns/2)/
        cAmbEffWidthSigmaEst_ns;
    sigmaEstimateP2 *= cAmbEffWidthDMax_ns;

    /* FixPoint1616 >> 16 = uint32 */
    minSignalNeeded_p3 = (sigmaEstimateP2 + 0x8000) >> 16;

    minSignalNeeded_p3 *= minSignalNeeded_p3;

}

/* FixPoint1814 / uint32 = FixPoint1814 */
sigmaLimitTmp = ((cSigmaLimit << 14) + 500) / 1000;

/* FixPoint1814 * FixPoint1814 = FixPoint3628 := FixPoint0428 */
sigmaLimitTmp *= sigmaLimitTmp;

/* FixPoint1616 * FixPoint1616 = FixPoint3232 */
sigmaEstSqTmp = cSigmaEstRef * cSigmaEstRef;

```

```

/* FixPoint3232 >> 4 = FixPoint0428 */
sigmaEstSqTmp = (sigmaEstSqTmp + 0x08) >> 4;

/* FixPoint0428 - FixPoint0428 = FixPoint0428 */
sigmaLimitTmp -= sigmaEstSqTmp;

/* uint32_t * FixPoint0428 = FixPoint0428 */
minSignalNeeded_p4 = 4 * 12 * sigmaLimitTmp;

/* FixPoint0428 >> 14 = FixPoint1814 */
minSignalNeeded_p4 = (minSignalNeeded_p4 + 0x2000) >> 14;

/* uint32 + uint32 = uint32 */
minSignalNeeded = (minSignalNeeded_p2 + minSignalNeeded_p3);

/* uint32 / uint32 = uint32 */
minSignalNeeded += (peakVcseIDuration_us/2);
minSignalNeeded /= peakVcseIDuration_us;

/* uint32 << 14 = FixPoint1814 */
minSignalNeeded <<= 14;

/* FixPoint1814 / FixPoint1814 = uint32 */
minSignalNeeded += (minSignalNeeded_p4/2);
minSignalNeeded /= minSignalNeeded_p4;

/* FixPoint3200 * FixPoint2804 := FixPoint2804*/
minSignalNeeded *= minSignalNeeded_p1;

```

```

/* Apply correction by dividing by 1000000.
 * This assumes 10E16 on the numerator of the equation
 * and 10E-22 on the denominator.
 * We do this because 32bit fix point calculation can't
 * handle the larger and smaller elements of this equation,
 * i.e. speed of light and pulse widths.
 */
minSignalNeeded = (minSignalNeeded + 500) / 1000;
minSignalNeeded <<= 4;

minSignalNeeded = (minSignalNeeded + 500) / 1000;

/* FixPoint1616 >> 8 = FixPoint2408 */
signalLimitTmp = (cSignalLimit + 0x80) >> 8;

/* FixPoint2408/FixPoint2408 = uint32 */
if (signalLimitTmp != 0)
    dmaxDarkTmp = (SignalAt0mm + (signalLimitTmp / 2))
        / signalLimitTmp;
else
    dmaxDarkTmp = 0;

dmaxDark = VL53L0X_isqrt(dmaxDarkTmp);

/* FixPoint2408/FixPoint2408 = uint32 */
if (minSignalNeeded != 0)
    dmaxAmbient = (SignalAt0mm + minSignalNeeded/2)
        / minSignalNeeded;

```

```

else

    dmaxAmbient = 0;

dmaxAmbient = VL53L0X_isqrt(dmaxAmbient);

*pdmax_mm = dmaxDark;
if (dmaxDark > dmaxAmbient)
    *pdmax_mm = dmaxAmbient;

LOG_FUNCTION_END(Status);

return Status;
}

VL53L0X_Error VL53L0X_calc_sigma_estimate(VL53L0X_DEV Dev,
    VL53L0X_RangingMeasurementData_t *pRangingMeasurementData,
    FixPoint1616_t *pSigmaEstimate,
    uint32_t *pDmax_mm)
{
    /* Expressed in 100ths of a ns, i.e. centi-ns */
    const uint32_t cPulseEffectiveWidth_cent_ns = 800;
    /* Expressed in 100ths of a ns, i.e. centi-ns */
    const uint32_t cAmbientEffectiveWidth_cent_ns = 600;
    const FixPoint1616_t cSigmaEstRef = 0x00000042; /* 0.001 */
    const uint32_t cVcslPulseWidth_ps = 4700; /* pico secs */
    const FixPoint1616_t cSigmaEstMax = 0x028F87AE;
    const FixPoint1616_t cSigmaEstRtnMax = 0xF000;
    const FixPoint1616_t cAmbToSignalRatioMax = 0xF0000000/

```

```

        cAmbientEffectiveWidth_centims;

/* Time Of Flight per mm (6.6 pico secs) */

const FixPoint1616_t cTOF_per_mm_ps      = 0x0006999A;
const uint32_t c16BitRoundingParam        = 0x00008000;
const FixPoint1616_t cMaxXTalk_kcps       = 0x00320000;
const uint32_t cPIIPeriod_ps              = 1655;


uint32_t vcSelTotalEventsRtn;
uint32_t finalRangeTimeoutMicroSecs;
uint32_t preRangeTimeoutMicroSecs;
FixPoint1616_t sigmaEstimateP1;
FixPoint1616_t sigmaEstimateP2;
FixPoint1616_t sigmaEstimateP3;
FixPoint1616_t deltaT_ps;
FixPoint1616_t pwMult;
FixPoint1616_t sigmaEstRtn;
FixPoint1616_t sigmaEstimate;
FixPoint1616_t xTalkCorrection;
FixPoint1616_t ambientRate_kcps;
FixPoint1616_t peakSignalRate_kcps;
FixPoint1616_t xTalkCompRate_mcps;
uint32_t xTalkCompRate_kcps;
VL53L0X_Error Status = VL53L0X_ERROR_NONE;
FixPoint1616_t diff1_mcps;
FixPoint1616_t diff2_mcps;
FixPoint1616_t sqr1;
FixPoint1616_t sqr2;
FixPoint1616_t sqrSum;
FixPoint1616_t sqrtResult_centims;

```

```

FixPoint1616_t sqrtResult;

FixPoint1616_t totalSignalRate_mcps;

FixPoint1616_t correctedSignalRate_mcps;

uint32_t vcselWidth;

uint32_t finalRangeMacroPCLKS;

uint32_t preRangeMacroPCLKS;

uint32_t peakVcselDuration_us;

uint8_t finalRangeVcselPCLKS;

uint8_t preRangeVcselPCLKS;

/*! \addtogroup calc_sigma_estimate
* @{
*
* Estimates the range sigma based on the
*
* - vcsel_rate_kcps
* - ambient_rate_kcps
* - signal_total_events
* - xtalk_rate
*
* and the following parameters
*
* - SigmaEstRefArray
* - SigmaEstEffPulseWidth
* - SigmaEstEffAmbWidth
*/

LOG_FUNCTION_START("");

VL53L0X_GETPARAMETERFIELD(Dev, XTalkCompensationRateMegaCps,

```

```

        xTalkCompRate_mcps);

/*
 * We work in kcps rather than mcps as this helps keep within the
 * confines of the 32 Fix1616 type.
 */

ambientRate_kcps =
    (pRangingMeasurementData->AmbientRateRtnMegaCps * 1000) >> 16;

correctedSignalRate_mcps =
    pRangingMeasurementData->SignalRateRtnMegaCps;

Status = VL53L0X_get_total_signal_rate(
    Dev, pRangingMeasurementData, &totalSignalRate_mcps);
Status = VL53L0X_get_total_xtalk_rate(
    Dev, pRangingMeasurementData, &xTalkCompRate_mcps);

/* Signal rate measurement provided by device is the
 * peak signal rate, not average.
 */
peakSignalRate_kcps = (totalSignalRate_mcps * 1000);
peakSignalRate_kcps = (peakSignalRate_kcps + 0x8000) >> 16;

xTalkCompRate_kcps = xTalkCompRate_mcps * 1000;

if (xTalkCompRate_kcps > cMaxXTalk_kcps)

```

```

xTalkCompRate_kcps = cMaxXTalk_kcps;

if (Status == VL53L0X_ERROR_NONE) {

    /* Calculate final range macro periods */
    finalRangeTimeoutMicroSecs = VL53L0X_GETDEVICESPECIFICPARAMETER(
        Dev, FinalRangeTimeoutMicroSecs);

    finalRangeVcselPCLKS = VL53L0X_GETDEVICESPECIFICPARAMETER(
        Dev, FinalRangeVcselPulsePeriod);

    finalRangeMacroPCLKS = VL53L0X_calc_timeout_mclks(
        Dev, finalRangeTimeoutMicroSecs, finalRangeVcselPCLKS);

    /* Calculate pre-range macro periods */
    preRangeTimeoutMicroSecs = VL53L0X_GETDEVICESPECIFICPARAMETER(
        Dev, PreRangeTimeoutMicroSecs);

    preRangeVcselPCLKS = VL53L0X_GETDEVICESPECIFICPARAMETER(
        Dev, PreRangeVcselPulsePeriod);

    preRangeMacroPCLKS = VL53L0X_calc_timeout_mclks(
        Dev, preRangeTimeoutMicroSecs, preRangeVcselPCLKS);

    vcselWidth = 3;
    if (finalRangeVcselPCLKS == 8)
        vcselWidth = 2;

```

```

peakVcseIDuration_us = vcseIDWidth * 2048 *
    (preRangeMacroPCLKS + finalRangeMacroPCLKS);
peakVcseIDuration_us = (peakVcseIDuration_us + 500)/1000;
peakVcseIDuration_us *= cPIIPeriod_ps;
peakVcseIDuration_us = (peakVcseIDuration_us + 500)/1000;

/* Fix1616 >> 8 = Fix2408 */
totalSignalRate_mcps = (totalSignalRate_mcps + 0x80) >> 8;

/* Fix2408 * uint32 = Fix2408 */
vcseIDTotalEventsRtn = totalSignalRate_mcps *
    peakVcseIDuration_us;

/* Fix2408 >> 8 = uint32 */
vcseIDTotalEventsRtn = (vcseIDTotalEventsRtn + 0x80) >> 8;

/* Fix2408 << 8 = Fix1616 = */
totalSignalRate_mcps <<= 8;
}

if (Status != VL53L0X_ERROR_NONE) {
    LOG_FUNCTION_END(Status);
    return Status;
}

if (peakSignalRate_kcps == 0) {
    *pSigmaEstimate = cSigmaEstMax;
    PALDevDataSet(Dev, SigmaEstimate, cSigmaEstMax);
    *pDmax_mm = 0;
}

```

```

} else {
    if (vcseTotalEventsRtn < 1)
        vcseTotalEventsRtn = 1;

    /*
    * Calculate individual components of the main equation -
    * replicating the equation implemented in the script
    * OpenAll_Ewok_ranging_data.jsl.
    *
    * sigmaEstimateP1 represents the effective pulse width, which
    * is a tuning parameter, rather than a real value.
    *
    * sigmaEstimateP2 represents the ambient/signal rate ratio
    * expressed as a multiple of the effective ambient width
    * (tuning parameter).
    *
    * sigmaEstimateP3 provides the signal event component, with the
    * knowledge that
    *
    *   - Noise of a square pulse is 1/sqrt(12) of the pulse
    *     width.
    *
    *   - at 0Lux, sigma is proportional to
    *
    *     effectiveVcsePulseWidth/sqrt(12 * signalTotalEvents)
    *
    * deltaT_ps represents the time of flight in pico secs for the
    * current range measurement, using the "TOF per mm" constant
    * (in ps).
    */

    sigmaEstimateP1 = cPulseEffectiveWidth_centis;

```

```

/* ((FixPoint1616 << 16)* uint32)/uint32 = FixPoint1616 */
sigmaEstimateP2 = (ambientRate_kcps << 16)/peakSignalRate_kcps;
if (sigmaEstimateP2 > cAmbToSignalRatioMax) {
    /* Clip to prevent overflow. Will ensure safe
    * max result. */
    sigmaEstimateP2 = cAmbToSignalRatioMax;
}
sigmaEstimateP2 *= cAmbientEffectiveWidth_centi_ns;

sigmaEstimateP3 = 2 * VL53LOX_isqrt(vcselTotalEventsRtn * 12);

/* uint32 * FixPoint1616 = FixPoint1616 */
deltaT_ps = pRangingMeasurementData->RangeMilliMeter *
            cTOF_per_mm_ps;

/*
* vcselRate - xtalkCompRate
* (uint32 << 16) - FixPoint1616 = FixPoint1616.
* Divide result by 1000 to convert to mcps.
* 500 is added to ensure rounding when integer division
* truncates.
*/
diff1_mcps = (((peakSignalRate_kcps << 16) -
                xTalkCompRate_kcps) + 500)/1000;

/* vcselRate + xtalkCompRate */
diff2_mcps = (((peakSignalRate_kcps << 16) +
                xTalkCompRate_kcps) + 500)/1000;

```

```

/* Shift by 8 bits to increase resolution prior to the
 * division */
diff1_mcps <<= 8;

/* FixPoint0824/FixPoint1616 = FixPoint2408 */
xTalkCorrection = abs(diff1_mcps/diff2_mcps);

/* FixPoint2408 << 8 = FixPoint1616 */
xTalkCorrection <<= 8;

/* FixPoint1616/uint32 = FixPoint1616 */
pwMult = deltaT_ps/cVcslPulseWidth_ps; /* smaller than 1.0f */

/*
 * FixPoint1616 * FixPoint1616 = FixPoint3232, however both
 * values are small enough such that 32 bits will not be
 * exceeded.
 */
pwMult *= ((1 << 16) - xTalkCorrection);

/* (FixPoint3232 >> 16) = FixPoint1616 */
pwMult = (pwMult + c16BitRoundingParam) >> 16;

/* FixPoint1616 + FixPoint1616 = FixPoint1616 */
pwMult += (1 << 16);

/*
 * At this point the value will be 1.xx, therefore if we square

```

* the value this will exceed 32 bits. To address this perform

* a single shift to the right before the multiplication.

*/

pwMult >>= 1;

/* FixPoint1715 * FixPoint1715 = FixPoint3430 */

pwMult = pwMult * pwMult;

/* (FixPoint3430 >> 14) = Fix1616 */

pwMult >>= 14;

/* FixPoint1616 * uint32 = FixPoint1616 */

sqr1 = pwMult * sigmaEstimateP1;

/* (FixPoint1616 >> 16) = FixPoint3200 */

sqr1 = (sqr1 + 0x8000) >> 16;

/* FixPoint3200 * FixPoint3200 = FixPoint6400 */

sqr1 *= sqr1;

sqr2 = sigmaEstimateP2;

/* (FixPoint1616 >> 16) = FixPoint3200 */

sqr2 = (sqr2 + 0x8000) >> 16;

/* FixPoint3200 * FixPoint3200 = FixPoint6400 */

sqr2 *= sqr2;

/* FixPoint64000 + FixPoint6400 = FixPoint6400 */

sqrSum = sqr1 + sqr2;

```

/* SQRT(FixPoin6400) = FixPoint3200 */
sqrtResult_centi_ns = VL53LOX_isqrt(sqrSum);

/* (FixPoint3200 << 16) = FixPoint1616 */
sqrtResult_centi_ns <<= 16;

/*
 * Note that the Speed Of Light is expressed in um per 1E-10
 * seconds (2997) Therefore to get mm/ns we have to divide by
 * 10000
 */
sigmaEstRtn = (((sqrtResult_centi_ns+50)/100) /
                sigmaEstimateP3);
sigmaEstRtn      *= VL53LOX_SPEED_OF_LIGHT_IN_AIR;

/* Add 5000 before dividing by 10000 to ensure rounding. */
sigmaEstRtn      += 5000;
sigmaEstRtn      /= 10000;

if (sigmaEstRtn > cSigmaEstRtnMax) {
    /* Clip to prevent overflow. Will ensure safe
     * max result. */
    sigmaEstRtn = cSigmaEstRtnMax;
}

/* FixPoint1616 * FixPoint1616 = FixPoint3232 */
sqr1 = sigmaEstRtn * sigmaEstRtn;

/* FixPoint1616 * FixPoint1616 = FixPoint3232 */

```

```

sqr2 = cSigmaEstRef * cSigmaEstRef;

/* sqrt(FixPoint3232) = FixPoint1616 */
sqrtResult = VL53L0X_isqrt((sqr1 + sqr2));
/*
 * Note that the Shift by 4 bits increases resolution prior to
 * the sqrt, therefore the result must be shifted by 2 bits to
 * the right to revert back to the FixPoint1616 format.
 */

sigmaEstimate  = 1000 * sqrtResult;

if ((peakSignalRate_kcps < 1) || (vcSelTotalEventsRtn < 1) ||
    (sigmaEstimate > cSigmaEstMax)) {
    sigmaEstimate = cSigmaEstMax;
}

*pSigmaEstimate = (uint32_t)(sigmaEstimate);
PALDevDataSet(Dev, SigmaEstimate, *pSigmaEstimate);
Status = VL53L0X_calc_dmax(
    Dev,
    totalSignalRate_mcps,
    correctedSignalRate_mcps,
    pwMult,
    sigmaEstimateP1,
    sigmaEstimateP2,
    peakVcSelDuration_us,
    pDmax_mm);
}

```

```

    LOG_FUNCTION_END(Status);

    return Status;
}

VL53L0X_Error VL53L0X_get_pal_range_status(VL53L0X_DEV Dev,
    uint8_t DeviceRangeStatus,
    FixPoint1616_t SignalRate,
    uint16_t EffectiveSpadRtnCount,
    VL53L0X_RangingMeasurementData_t *pRangingMeasurementData,
    uint8_t *pPalRangeStatus)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t NoneFlag;

    uint8_t SigmaLimitflag = 0;

    uint8_t SignalRefClipflag = 0;

    uint8_t RangeIgnoreThresholdflag = 0;

    uint8_t SigmaLimitCheckEnable = 0;

    uint8_t SignalRateFinalRangeLimitCheckEnable = 0;

    uint8_t SignalRefClipLimitCheckEnable = 0;

    uint8_t RangeIgnoreThresholdLimitCheckEnable = 0;

    FixPoint1616_t SigmaEstimate;

    FixPoint1616_t SigmaLimitValue;

    FixPoint1616_t SignalRefClipValue;

    FixPoint1616_t RangeIgnoreThresholdValue;

    FixPoint1616_t SignalRatePerSpad;

    uint8_t DeviceRangeStatusInternal = 0;

    uint16_t tmpWord = 0;

    uint8_t Temp8;

```

```
uint32_t Dmax_mm = 0;
```

```
FixPoint1616_t LastSignalRefMcps;
```

```
LOG_FUNCTION_START("");
```

```
/*
```

```
 * VL53L0X has a good ranging when the value of the
```

```
 * DeviceRangeStatus = 11. This function will replace the value 0 with
```

```
 * the value 11 in the DeviceRangeStatus.
```

```
 * In addition, the SigmaEstimator is not included in the VL53L0X
```

```
 * DeviceRangeStatus, this will be added in the PalRangeStatus.
```

```
*/
```

```
DeviceRangeStatusInternal = ((DeviceRangeStatus & 0x78) >> 3);
```

```
if (DeviceRangeStatusInternal == 0 ||
```

```
    DeviceRangeStatusInternal == 5 ||
```

```
    DeviceRangeStatusInternal == 7 ||
```

```
    DeviceRangeStatusInternal == 12 ||
```

```
    DeviceRangeStatusInternal == 13 ||
```

```
    DeviceRangeStatusInternal == 14 ||
```

```
    DeviceRangeStatusInternal == 15
```

```
    ) {
```

```
    NoneFlag = 1;
```

```
} else {
```

```
    NoneFlag = 0;
```

```
}
```

```

/* LastSignalRefMcps */
if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x01);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_RdWord(Dev,
        VL53L0X_REG_RESULT_PEAK_SIGNAL_RATE_REF,
        &tmpWord);

LastSignalRefMcps = VL53L0X_FIXPOINT97TOFIXPOINT1616(tmpWord);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x00);

PALDevDataSet(Dev, LastSignalRefMcps, LastSignalRefMcps);

/*
 * Check if Sigma limit is enabled, if yes then do comparison with limit
 * value and put the result back into pPalRangeStatus.
 */
if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_GetLimitCheckEnable(Dev,
        VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,
        &SigmaLimitCheckEnable);

if ((SigmaLimitCheckEnable != 0) && (Status == VL53L0X_ERROR_NONE)) {
    /*
     * compute the Sigma and check with limit
     */

```

```

        Status = VL53L0X_calc_sigma_estimate(
            Dev,
            pRangingMeasurementData,
            &SigmaEstimate,
            &Dmax_mm);
    if (Status == VL53L0X_ERROR_NONE)
        pRangingMeasurementData->RangeDMaxMilliMeter = Dmax_mm;

    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_GetLimitCheckValue(Dev,
            VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,
            &SigmaLimitValue);

        if ((SigmaLimitValue > 0) &&
            (SigmaEstimate > SigmaLimitValue))
            /* Limit Fail */
            SigmaLimitflag = 1;
    }
}

/*
 * Check if Signal ref clip limit is enabled, if yes then do comparison
 * with limit value and put the result back into pPalRangeStatus.
 */
if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_GetLimitCheckEnable(Dev,
        VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP,
        &SignalRefClipLimitCheckEnable);

```

```

if ((SignalRefClipLimitCheckEnable != 0) &&
    (Status == VL53L0X_ERROR_NONE)) {

    Status = VL53L0X_GetLimitCheckValue(Dev,
        VL53L0X_CHECKENABLE_SIGNAL_REF_CLIP,
        &SignalRefClipValue);

    if ((SignalRefClipValue > 0) &&
        (LastSignalRefMcps > SignalRefClipValue)) {
        /* Limit Fail */
        SignalRefClipflag = 1;
    }
}

/*
 * Check if Signal ref clip limit is enabled, if yes then do comparison
 * with limit value and put the result back into pPalRangeStatus.
 * EffectiveSpadRtnCount has a format 8.8
 * If (Return signal rate < (1.5 x Xtalk x number of Spads)) : FAIL
 */
if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_GetLimitCheckEnable(Dev,
        VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD,
        &RangeIgnoreThresholdLimitCheckEnable);

if ((RangeIgnoreThresholdLimitCheckEnable != 0) &&
    (Status == VL53L0X_ERROR_NONE)) {

    /* Compute the signal rate per spad */

```

```

if (EffectiveSpadRtnCount == 0) {
    SignalRatePerSpad = 0;
} else {
    SignalRatePerSpad = (FixPoint1616_t)((256 * SignalRate)
        / EffectiveSpadRtnCount);
}

Status = VL53L0X_GetLimitCheckValue(Dev,
    VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD,
    &RangeIgnoreThresholdValue);

if ((RangeIgnoreThresholdValue > 0) &&
    (SignalRatePerSpad < RangeIgnoreThresholdValue)) {
    /* Limit Fail add 2^6 to range status */
    RangeIgnoreThresholdflag = 1;
}
}

if (Status == VL53L0X_ERROR_NONE) {
    if (NoneFlag == 1) {
        *pPalRangeStatus = 255;      /* NONE */
    } else if (DeviceRangeStatusInternal == 1 ||
        DeviceRangeStatusInternal == 2 ||
        DeviceRangeStatusInternal == 3) {
        *pPalRangeStatus = 5; /* HW fail */
    } else if (DeviceRangeStatusInternal == 6 ||
        DeviceRangeStatusInternal == 9) {
        *pPalRangeStatus = 4; /* Phase fail */
    } else if (DeviceRangeStatusInternal == 8 ||

```

```

        DeviceRangeStatusInternal == 10 ||
        SignalRefClipflag == 1) {
            *pPalRangeStatus = 3; /* Min range */
        } else if (DeviceRangeStatusInternal == 4 ||
            RangeIgnoreThresholdflag == 1) {
            *pPalRangeStatus = 2; /* Signal Fail */
        } else if (SigmaLimitflag == 1) {
            *pPalRangeStatus = 1; /* Sigma Fail */
        } else {
            *pPalRangeStatus = 0; /* Range Valid */
        }
    }

    /* DMAX only relevant during range error */
    if (*pPalRangeStatus == 0)
        pRangingMeasurementData->RangeDMaxMilliMeter = 0;

    /* fill the Limit Check Status */

    Status = VL53L0X_GetLimitCheckEnable(Dev,
        VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
        &SignalRateFinalRangeLimitCheckEnable);

    if (Status == VL53L0X_ERROR_NONE) {
        if ((SignalLimitCheckEnable == 0) || (SigmaLimitflag == 1))
            Temp8 = 1;
        else
            Temp8 = 0;
        VL53L0X_SETARRAYPARAMETERFIELD(Dev, LimitChecksStatus,

```

```
VL53LOX_CHECKENABLE_SIGMA_FINAL_RANGE, Temp8);
```

```
if ((DeviceRangeStatusInternal == 4) ||
```

```
    (SignalRateFinalRangeLimitCheckEnable == 0))
```

```
    Temp8 = 1;
```

```
else
```

```
    Temp8 = 0;
```

```
VL53LOX_SETARRAYPARAMETERFIELD(Dev, LimitChecksStatus,
```

```
    VL53LOX_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE,
```

```
    Temp8);
```

```
if ((SignalRefClipLimitCheckEnable == 0) ||
```

```
    (SignalRefClipflag == 1))
```

```
    Temp8 = 1;
```

```
else
```

```
    Temp8 = 0;
```

```
VL53LOX_SETARRAYPARAMETERFIELD(Dev, LimitChecksStatus,
```

```
    VL53LOX_CHECKENABLE_SIGNAL_REF_CLIP, Temp8);
```

```
if ((RangeIgnoreThresholdLimitCheckEnable == 0) ||
```

```
    (RangeIgnoreThresholdflag == 1))
```

```
    Temp8 = 1;
```

```
else
```

```
    Temp8 = 0;
```

```
VL53LOX_SETARRAYPARAMETERFIELD(Dev, LimitChecksStatus,
```

```
    VL53LOX_CHECKENABLE_RANGE_IGNORE_THRESHOLD,
```

```
    Temp8);
```

```
}
```

```
LOG_FUNCTION_END(Status);
```

```
return Status;
```

```
}
```

1.1.40 vl53l0x_api_calibration.h

Below:

```
/******
```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the
names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.
IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#ifndef _VL53L0X_API_CALIBRATION_H_
```

```
#define _VL53L0X_API_CALIBRATION_H_
```

```
#include "vl53l0x_def.h"
```

```
#include "vl53l0x_platform.h"
```

```
#ifdef __cplusplus
```

```
extern "C" {
```

```
#endif
```

```
VL53L0X_Error VL53L0X_perform_xtalk_calibration(VL53L0X_DEV Dev,  
        FixPoint1616_t XTalkCalDistance,  
        FixPoint1616_t *pXTalkCompensationRateMegaCps);
```

```
VL53L0X_Error VL53L0X_perform_offset_calibration(VL53L0X_DEV Dev,  
        FixPoint1616_t CalDistanceMilliMeter,  
        int32_t *pOffsetMicroMeter);
```

```
VL53L0X_Error VL53L0X_set_offset_calibration_data_micro_meter(VL53L0X_DEV Dev,  
        int32_t OffsetCalibrationDataMicroMeter);
```

```
VL53L0X_Error VL53L0X_get_offset_calibration_data_micro_meter(VL53L0X_DEV Dev,  
    int32_t *pOffsetCalibrationDataMicroMeter);
```

```
VL53L0X_Error VL53L0X_apply_offset_adjustment(VL53L0X_DEV Dev);
```

```
VL53L0X_Error VL53L0X_perform_ref_spad_management(VL53L0X_DEV Dev,  
    uint32_t *refSpadCount, uint8_t *isApertureSpads);
```

```
VL53L0X_Error VL53L0X_set_reference_spads(VL53L0X_DEV Dev,  
    uint32_t count, uint8_t isApertureSpads);
```

```
VL53L0X_Error VL53L0X_get_reference_spads(VL53L0X_DEV Dev,  
    uint32_t *pSpadCount, uint8_t *plsApertureSpads);
```

```
VL53L0X_Error VL53L0X_perform_phase_calibration(VL53L0X_DEV Dev,  
    uint8_t *pPhaseCal, const uint8_t get_data_enable,  
    const uint8_t restore_config);
```

```
VL53L0X_Error VL53L0X_perform_ref_calibration(VL53L0X_DEV Dev,  
    uint8_t *pVhvSettings, uint8_t *pPhaseCal, uint8_t get_data_enable);
```

```
VL53L0X_Error VL53L0X_set_ref_calibration(VL53L0X_DEV Dev,  
    uint8_t VhvSettings, uint8_t PhaseCal);
```

```
VL53L0X_Error VL53L0X_get_ref_calibration(VL53L0X_DEV Dev,  
    uint8_t *pVhvSettings, uint8_t *pPhaseCal);
```

```
#ifdef __cplusplus
```

```
}
```

```
#endif
```

```
#endif /* _VL53L0X_API_CALIBRATION_H_ */
```

1.1.41 vl53l0x_api_calibration.cpp

Below:

```
/******
```

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the
names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.
IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#include "vl53l0x_api.h"
```

```
#include "vl53l0x_api_core.h"
```

```
#include "vl53l0x_api_calibration.h"
```

```
#ifndef __KERNEL__
```

```
#include <stdlib.h>
```

```
#endif
```

```
#define LOG_FUNCTION_START(fmt, ...) \
```

```
    _LOG_FUNCTION_START	TRACE_MODULE_API, fmt, ##__VA_ARGS__
```

```
#define LOG_FUNCTION_END(status, ...) \
```

```
    _LOG_FUNCTION_END	TRACE_MODULE_API, status, ##__VA_ARGS__
```

```
#define LOG_FUNCTION_END_FMT(status, fmt, ...) \
```

```
    _LOG_FUNCTION_END_FMT	TRACE_MODULE_API, status, fmt, ##__VA_ARGS__
```

```
#define REF_ARRAY_SPAD_0 0
```

```
#define REF_ARRAY_SPAD_5 5
```

```
#define REF_ARRAY_SPAD_10 10
```

```
uint32_t refArrayQuadrants[4] = {REF_ARRAY_SPAD_10, REF_ARRAY_SPAD_5,  
    REF_ARRAY_SPAD_0, REF_ARRAY_SPAD_5};
```

```

VL53L0X_Error VL53L0X_perform_xtalk_calibration(VL53L0X_DEV Dev,
                                                FixPoint1616_t XTalkCalDistance,
                                                FixPoint1616_t *pXTalkCompensationRateMegaCps)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint16_t sum_ranging = 0;
    uint16_t sum_spads = 0;
    FixPoint1616_t sum_signalRate = 0;
    FixPoint1616_t total_count = 0;
    uint8_t xtalk_meas = 0;

    VL53L0X_RangingMeasurementData_t RangingMeasurementData;
    FixPoint1616_t xTalkStoredMeanSignalRate;
    FixPoint1616_t xTalkStoredMeanRange;
    FixPoint1616_t xTalkStoredMeanRtnSpads;
    uint32_t signalXTalkTotalPerSpad;
    uint32_t xTalkStoredMeanRtnSpadsAsInt;
    uint32_t xTalkCalDistanceAsInt;
    FixPoint1616_t XTalkCompensationRateMegaCps;

    if (XTalkCalDistance <= 0)
        Status = VL53L0X_ERROR_INVALID_PARAMS;

    /* Disable the XTalk compensation */
    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_SetXTalkCompensationEnable(Dev, 0);

    /* Disable the RIT */
    if (Status == VL53L0X_ERROR_NONE) {
        Status = VL53L0X_SetLimitCheckEnable(Dev,

```

```

VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD, 0);
}

/* Perform 50 measurements and compute the averages */
if (Status == VL53L0X_ERROR_NONE) {
    sum_ranging = 0;
    sum_spads = 0;
    sum_signalRate = 0;
    total_count = 0;
    for (xtalk_meas = 0; xtalk_meas < 50; xtalk_meas++) {
        Status = VL53L0X_PerformSingleRangingMeasurement(Dev,
            &RangingMeasurementData);

        if (Status != VL53L0X_ERROR_NONE)
            break;

        /* The range is valid when RangeStatus = 0 */
        if (RangingMeasurementData.RangeStatus == 0) {
            sum_ranging = sum_ranging +
                RangingMeasurementData.RangeMilliMeter;
            sum_signalRate = sum_signalRate +
                RangingMeasurementData.SignalRateRtnMegaCps;
            sum_spads = sum_spads +
                RangingMeasurementData.EffectiveSpadRtnCount
                / 256;
            total_count = total_count + 1;
        }
    }
}

```

```

/* no valid values found */
if (total_count == 0)
    Status = VL53L0X_ERROR_RANGE_ERROR;

}

if (Status == VL53L0X_ERROR_NONE) {
    /* FixPoint1616_t / uint16_t = FixPoint1616_t */
    xTalkStoredMeanSignalRate = sum_signalRate / total_count;
    xTalkStoredMeanRange = (FixPoint1616_t)((uint32_t)(
        sum_ranging << 16) / total_count);
    xTalkStoredMeanRtnSpads = (FixPoint1616_t)((uint32_t)(
        sum_spads << 16) / total_count);

    /* Round Mean Spads to Whole Number.
    * Typically the calculated mean SPAD count is a whole number
    * or very close to a whole
    * number, therefore any truncation will not result in a
    * significant loss in accuracy.
    * Also, for a grey target at a typical distance of around
    * 400mm, around 220 SPADs will
    * be enabled, therefore, any truncation will result in a loss
    * of accuracy of less than
    * 0.5%.
    */
    xTalkStoredMeanRtnSpadsAsInt = (xTalkStoredMeanRtnSpads +
        0x8000) >> 16;

```

```

/* Round Cal Distance to Whole Number.

* Note that the cal distance is in mm, therefore no resolution
* is lost.*/

xTalkCalDistanceAsInt = (XTalkCalDistance + 0x8000) >> 16;

if (xTalkStoredMeanRtnSpadsAsInt == 0 ||
    xTalkCalDistanceAsInt == 0 ||
    xTalkStoredMeanRange >= XTalkCalDistance) {
    XTalkCompensationRateMegaCps = 0;
} else {
    /* Round Cal Distance to Whole Number.

    Note that the cal distance is in mm, therefore no
    resolution is lost.*/

    xTalkCalDistanceAsInt = (XTalkCalDistance +
        0x8000) >> 16;

    /* Apply division by mean spad count early in the
    * calculation to keep the numbers small.
    * This ensures we can maintain a 32bit calculation.
    * Fixed1616 / int := Fixed1616 */
    signalXTalkTotalPerSpad = (xTalkStoredMeanSignalRate) /
        xTalkStoredMeanRtnSpadsAsInt;

    /* Complete the calculation for total Signal XTalk per
    * SPAD
    * Fixed1616 * (Fixed1616 - Fixed1616/int) :=
    * (2^16 * Fixed1616)
    */
    signalXTalkTotalPerSpad *= ((1 << 16) -

```

```

        (xTalkStoredMeanRange / xTalkCalDistanceAsInt));

/* Round from 2^16 * Fixed1616, to Fixed1616. */
XTalkCompensationRateMegaCps = (signalXTalkTotalPerSpad
    + 0x8000) >> 16;
}

*pXTalkCompensationRateMegaCps = XTalkCompensationRateMegaCps;

/* Enable the XTalk compensation */
if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_SetXTalkCompensationEnable(Dev, 1);

/* Enable the XTalk compensation */
if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_SetXTalkCompensationRateMegaCps(Dev,
        XTalkCompensationRateMegaCps);

}

return Status;
}

VL53L0X_Error VL53L0X_perform_offset_calibration(VL53L0X_DEV Dev,
    FixPoint1616_t CalDistanceMilliMeter,
    int32_t *pOffsetMicroMeter)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint16_t sum_ranging = 0;

```

```

FixPoint1616_t total_count = 0;

VL53L0X_RangingMeasurementData_t RangingMeasurementData;

FixPoint1616_t StoredMeanRange;

uint32_t StoredMeanRangeAsInt;

uint32_t CalDistanceAsInt_mm;

uint8_t SequenceStepEnabled;

int meas = 0;


if (CalDistanceMilliMeter <= 0)

    Status = VL53L0X_ERROR_INVALID_PARAMS;


if (Status == VL53L0X_ERROR_NONE)

    Status = VL53L0X_SetOffsetCalibrationDataMicroMeter(Dev, 0);


/* Get the value of the TCC */

if (Status == VL53L0X_ERROR_NONE)

    Status = VL53L0X_GetSequenceStepEnable(Dev,

        VL53L0X_SEQUENCESTEP_TCC, &SequenceStepEnabled);


/* Disable the TCC */

if (Status == VL53L0X_ERROR_NONE)

    Status = VL53L0X_SetSequenceStepEnable(Dev,

        VL53L0X_SEQUENCESTEP_TCC, 0);


/* Disable the RIT */

if (Status == VL53L0X_ERROR_NONE)

```

```

        Status = VL53L0X_SetLimitCheckEnable(Dev,
                                              VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD, 0);

/* Perform 50 measurements and compute the averages */
if (Status == VL53L0X_ERROR_NONE) {
    sum_ranging = 0;
    total_count = 0;
    for (meas = 0; meas < 50; meas++) {
        Status = VL53L0X_PerformSingleRangingMeasurement(Dev,
                                                         &RangingMeasurementData);

        if (Status != VL53L0X_ERROR_NONE)
            break;

        /* The range is valid when RangeStatus = 0 */
        if (RangingMeasurementData.RangeStatus == 0) {
            sum_ranging = sum_ranging +
                RangingMeasurementData.RangeMilliMeter;
            total_count = total_count + 1;
        }
    }

    /* no valid values found */
    if (total_count == 0)
        Status = VL53L0X_ERROR_RANGE_ERROR;
}

if (Status == VL53L0X_ERROR_NONE) {

```

```

/* FixPoint1616_t / uint16_t = FixPoint1616_t */
StoredMeanRange = (FixPoint1616_t)((uint32_t)(sum_ranging << 16)
    / total_count);

StoredMeanRangeAsInt = (StoredMeanRange + 0x8000) >> 16;

/* Round Cal Distance to Whole Number.
 * Note that the cal distance is in mm, therefore no resolution
 * is lost.*/
CalDistanceAsInt_mm = (CalDistanceMilliMeter + 0x8000) >> 16;

*pOffsetMicroMeter = (CalDistanceAsInt_mm -
    StoredMeanRangeAsInt) * 1000;

/* Apply the calculated offset */
if (Status == VL53L0X_ERROR_NONE) {
    VL53L0X_SETPARAMETERFIELD(Dev, RangeOffsetMicroMeters,
        *pOffsetMicroMeter);
    Status = VL53L0X_SetOffsetCalibrationDataMicroMeter(Dev,
        *pOffsetMicroMeter);
}

}

/* Restore the TCC */
if (Status == VL53L0X_ERROR_NONE) {
    if (SequenceStepEnabled != 0)
        Status = VL53L0X_SetSequenceStepEnable(Dev,
            VL53L0X_SEQUENCESTEP_TCC, 1);
}

```

```

    }

    return Status;
}

VL53L0X_Error VL53L0X_set_offset_calibration_data_micro_meter(VL53L0X_DEV Dev,
    int32_t OffsetCalibrationDataMicroMeter)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    int32_t cMaxOffsetMicroMeter = 511000;
    int32_t cMinOffsetMicroMeter = -512000;
    int16_t cOffsetRange = 4096;
    uint32_t encodedOffsetVal;

    LOG_FUNCTION_START("");

    if (OffsetCalibrationDataMicroMeter > cMaxOffsetMicroMeter)
        OffsetCalibrationDataMicroMeter = cMaxOffsetMicroMeter;
    else if (OffsetCalibrationDataMicroMeter < cMinOffsetMicroMeter)
        OffsetCalibrationDataMicroMeter = cMinOffsetMicroMeter;

    /* The offset register is 10.2 format and units are mm
     * therefore conversion is applied by a division of
     * 250.
     */
    if (OffsetCalibrationDataMicroMeter >= 0) {
        encodedOffsetVal =
            OffsetCalibrationDataMicroMeter/250;
    }
}

```

```

    } else {
        encodedOffsetVal =
            cOffsetRange +
            OffsetCalibrationDataMicroMeter/250;
    }

    Status = VL53L0X_WrWord(Dev,
        VL53L0X_REG_ALGO_PART_TO_PART_RANGE_OFFSET_MM,
        encodedOffsetVal);

    LOG_FUNCTION_END(Status);
    return Status;
}

VL53L0X_Error VL53L0X_get_offset_calibration_data_micro_meter(VL53L0X_DEV Dev,
    int32_t *pOffsetCalibrationDataMicroMeter)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint16_t RangeOffsetRegister;
    int16_t cMaxOffset = 2047;
    int16_t cOffsetRange = 4096;

    /* Note that offset has 10.2 format */

    Status = VL53L0X_RdWord(Dev,
        VL53L0X_REG_ALGO_PART_TO_PART_RANGE_OFFSET_MM,
        &RangeOffsetRegister);

    if (Status == VL53L0X_ERROR_NONE) {

```

```

RangeOffsetRegister = (RangeOffsetRegister & 0x0fff);

/* Apply 12 bit 2's compliment conversion */
if (RangeOffsetRegister > cMaxOffset)
    *pOffsetCalibrationDataMicroMeter =
        (int16_t)(RangeOffsetRegister - cOffsetRange)
            * 250;
else
    *pOffsetCalibrationDataMicroMeter =
        (int16_t)RangeOffsetRegister * 250;
}

return Status;
}

```

```

VL53L0X_Error VL53L0X_apply_offset_adjustment(VL53L0X_DEV Dev)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    int32_t CorrectedOffsetMicroMeters;
    int32_t CurrentOffsetMicroMeters;

    /* if we run on this function we can read all the NVM info
     * used by the API */
    Status = VL53L0X_get_info_from_device(Dev, 7);

    /* Read back current device offset */
    if (Status == VL53L0X_ERROR_NONE) {

```

```

        Status = VL53L0X_GetOffsetCalibrationDataMicroMeter(Dev,
                                                                &CurrentOffsetMicroMeters);
    }

    /* Apply Offset Adjustment derived from 400mm measurements */
    if (Status == VL53L0X_ERROR_NONE) {

        /* Store initial device offset */
        PALDevDataSet(Dev, Part2PartOffsetNVMMicroMeter,
                      CurrentOffsetMicroMeters);

        CorrectedOffsetMicroMeters = CurrentOffsetMicroMeters +
            (int32_t)PALDevDataGet(Dev,
                                   Part2PartOffsetAdjustmentNVMMicroMeter);

        Status = VL53L0X_SetOffsetCalibrationDataMicroMeter(Dev,
                                                                CorrectedOffsetMicroMeters);

        /* store current, adjusted offset */
        if (Status == VL53L0X_ERROR_NONE) {
            VL53L0X_SETPARAMETERFIELD(Dev, RangeOffsetMicroMeters,
                                       CorrectedOffsetMicroMeters);
        }
    }

    return Status;
}

```

```

void get_next_good_spad(uint8_t goodSpadArray[], uint32_t size,

```

```

        uint32_t curr, int32_t *next)

{
    uint32_t startIndex;
    uint32_t fineOffset;
    uint32_t cSpadsPerByte = 8;
    uint32_t coarseIndex;
    uint32_t fineIndex;
    uint8_t dataByte;
    uint8_t success = 0;

    /*
     * Starting with the current good spad, loop through the array to find
     * the next. i.e. the next bit set in the sequence.
     *
     * The coarse index is the byte index of the array and the fine index is
     * the index of the bit within each byte.
     */

    *next = -1;

    startIndex = curr / cSpadsPerByte;
    fineOffset = curr % cSpadsPerByte;

    for (coarseIndex = startIndex; ((coarseIndex < size) && !success);
        coarseIndex++) {
        fineIndex = 0;
        dataByte = goodSpadArray[coarseIndex];

        if (coarseIndex == startIndex) {

```

```

        /* locate the bit position of the provided current
        * spad bit before iterating */
        dataByte >>= fineOffset;
        fineIndex = fineOffset;
    }

    while (fineIndex < cSpadsPerByte) {
        if ((dataByte & 0x1) == 1) {
            success = 1;

            *next = coarseIndex * cSpadsPerByte + fineIndex;
            break;
        }
        dataByte >>= 1;
        fineIndex++;
    }
}

```

```

uint8_t is_aperture(uint32_t spadIndex)
{
    /*
    * This function reports if a given spad index is an aperture SPAD by
    * deriving the quadrant.
    */
    uint32_t quadrant;
    uint8_t isAperture = 1;
    quadrant = spadIndex >> 6;
    if (refArrayQuadrants[quadrant] == REF_ARRAY_SPAD_0)

```

```

        isAperture = 0;

    return isAperture;
}

VL53L0X_Error enable_spad_bit(uint8_t spadArray[], uint32_t size,
    uint32_t spadIndex)
{
    VL53L0X_Error status = VL53L0X_ERROR_NONE;

    uint32_t cSpadsPerByte = 8;

    uint32_t coarseIndex;
    uint32_t fineIndex;

    coarseIndex = spadIndex / cSpadsPerByte;
    fineIndex = spadIndex % cSpadsPerByte;

    if (coarseIndex >= size)
        status = VL53L0X_ERROR_REF_SPAD_INIT;
    else
        spadArray[coarseIndex] |= (1 << fineIndex);

    return status;
}

VL53L0X_Error count_enabled_spads(uint8_t spadArray[],
    uint32_t byteCount, uint32_t maxSpads,
    uint32_t *pTotalSpadsEnabled, uint8_t *pIsAperture)
{
    VL53L0X_Error status = VL53L0X_ERROR_NONE;

```

```

uint32_t cSpadsPerByte = 8;

uint32_t lastByte;

uint32_t lastBit;

uint32_t byteIndex = 0;

uint32_t bitIndex = 0;

uint8_t tempByte;

uint8_t spadTypeIdentified = 0;

/* The entire array will not be used for spads, therefore the last
 * byte and last bit is determined from the max spads value.
 */

lastByte = maxSpads / cSpadsPerByte;

lastBit = maxSpads % cSpadsPerByte;

/* Check that the max spads value does not exceed the array bounds. */
if (lastByte >= byteCount)
    status = VL53LOX_ERROR_REF_SPAD_INIT;

*pTotalSpadsEnabled = 0;

/* Count the bits enabled in the whole bytes */
for (byteIndex = 0; byteIndex <= (lastByte - 1); byteIndex++) {
    tempByte = spadArray[byteIndex];

    for (bitIndex = 0; bitIndex <= cSpadsPerByte; bitIndex++) {
        if ((tempByte & 0x01) == 1) {
            (*pTotalSpadsEnabled)++;
        }
    }
}

```

```

        if (!spadTypeIdentified) {
            *plsAperture = 1;
            if ((byteIndex < 2) && (bitIndex < 4))
                *plsAperture = 0;
            spadTypeIdentified = 1;
        }
    }
    tempByte >>= 1;
}

/* Count the number of bits enabled in the last byte accounting
 * for the fact that not all bits in the byte may be used.
 */
tempByte = spadArray[lastByte];

for (bitIndex = 0; bitIndex <= lastBit; bitIndex++) {
    if ((tempByte & 0x01) == 1)
        (*pTotalSpadsEnabled)++;
}

return status;
}

VL53L0X_Error set_ref_spad_map(VL53L0X_DEV Dev, uint8_t *refSpadArray)
{
    VL53L0X_Error status = VL53L0X_WriteMulti(Dev,
        VL53L0X_REG_GLOBAL_CONFIG_SPAD_ENABLES_REF_0,
        refSpadArray, 6);
}

```

```
        return status;
    }
}
```

```
VL53L0X_Error get_ref_spad_map(VL53L0X_DEV Dev, uint8_t *refSpadArray)
{
    VL53L0X_Error status = VL53L0X_ReadMulti(Dev,
                                                VL53L0X_REG_GLOBAL_CONFIG_SPAD_ENABLES_REF_0,
                                                refSpadArray,
                                                6);

    return status;
}
```

```
VL53L0X_Error enable_ref_spads(VL53L0X_DEV Dev,
                                uint8_t apertureSpads,
                                uint8_t goodSpadArray[],
                                uint8_t spadArray[],
                                uint32_t size,
                                uint32_t start,
                                uint32_t offset,
                                uint32_t spadCount,
                                uint32_t *lastSpad)
{
    VL53L0X_Error status = VL53L0X_ERROR_NONE;
    uint32_t index;
    uint32_t i;
    int32_t nextGoodSpad = offset;
    uint32_t currentSpad;
    uint8_t checkSpadArray[6];
```

```

/*
 * This function takes in a spad array which may or may not have SPADS
 * already enabled and appends from a given offset a requested number
 * of new SPAD enables. The 'good spad map' is applied to
 * determine the next SPADs to enable.
 *
 * This function applies to only aperture or only non-aperture spads.
 * Checks are performed to ensure this.
 */

```

```

currentSpad = offset;
for (index = 0; index < spadCount; index++) {
    get_next_good_spad(goodSpadArray, size, currentSpad,
                       &nextGoodSpad);

    if (nextGoodSpad == -1) {
        status = VL53LOX_ERROR_REF_SPAD_INIT;
        break;
    }

    /* Confirm that the next good SPAD is non-aperture */
    if (is_aperture(start + nextGoodSpad) != apertureSpads) {
        /* if we can't get the required number of good aperture
         * spads from the current quadrant then this is an error
         */
        status = VL53LOX_ERROR_REF_SPAD_INIT;
        break;
    }

    currentSpad = (uint32_t)nextGoodSpad;
}

```

```

        enable_spad_bit(spadArray, size, currentSpad);
        currentSpad++;
    }
    *lastSpad = currentSpad;

    if (status == VL53L0X_ERROR_NONE)
        status = set_ref_spad_map(Dev, spadArray);

    if (status == VL53L0X_ERROR_NONE) {
        status = get_ref_spad_map(Dev, checkSpadArray);

        i = 0;

        /* Compare spad maps. If not equal report error. */
        while (i < size) {
            if (spadArray[i] != checkSpadArray[i]) {
                status = VL53L0X_ERROR_REF_SPAD_INIT;
                break;
            }
            i++;
        }
    }
    return status;
}

```

```

VL53L0X_Error perform_ref_signal_measurement(VL53L0X_DEV Dev,
        uint16_t *refSignalRate)

```

```

{

VL53L0X_Error status = VL53L0X_ERROR_NONE;

VL53L0X_RangingMeasurementData_t rangingMeasurementData;


uint8_t SequenceConfig = 0;


/* store the value of the sequence config,
 * this will be reset before the end of the function
 */

SequenceConfig = PALDevDataGet(Dev, SequenceConfig);


/*
 * This function performs a reference signal rate measurement.
 */
if (status == VL53L0X_ERROR_NONE)
    status = VL53L0X_WrByte(Dev,
        VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG, 0xC0);

if (status == VL53L0X_ERROR_NONE)
    status = VL53L0X_PerformSingleRangingMeasurement(Dev,
        &rangingMeasurementData);

if (status == VL53L0X_ERROR_NONE)
    status = VL53L0X_WrByte(Dev, 0xFF, 0x01);

if (status == VL53L0X_ERROR_NONE)
    status = VL53L0X_RdWord(Dev,
        VL53L0X_REG_RESULT_PEAK_SIGNAL_RATE_REF,

```

```

        refSignalRate);

if (status == VL53L0X_ERROR_NONE)
    status = VL53L0X_WrByte(Dev, 0xFF, 0x00);

if (status == VL53L0X_ERROR_NONE) {
    /* restore the previous Sequence Config */
    status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG,
        SequenceConfig);
    if (status == VL53L0X_ERROR_NONE)
        PALDevDataSet(Dev, SequenceConfig, SequenceConfig);
}

return status;
}

```

```

VL53L0X_Error VL53L0X_perform_ref_spad_management(VL53L0X_DEV Dev,
    uint32_t *refSpadCount,
    uint8_t *isApertureSpads)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t lastSpadArray[6];
    uint8_t startSelect = 0xB4;
    uint32_t minimumSpadCount = 3;
    uint32_t maxSpadCount = 44;
    uint32_t currentSpadIndex = 0;
    uint32_t lastSpadIndex = 0;
    int32_t nextGoodSpad = 0;
    uint16_t targetRefRate = 0x0A00; /* 20 MCPS in 9:7 format */

```

```
uint16_t peakSignalRateRef;
uint32_t needAptSpads = 0;
uint32_t index = 0;
uint32_t spadArraySize = 6;
uint32_t signalRateDiff = 0;
uint32_t lastSignalRateDiff = 0;
uint8_t complete = 0;
uint8_t VhvSettings = 0;
uint8_t PhaseCal = 0;
uint32_t refSpadCount_int = 0;
uint8_t isApertureSpads_int = 0;
```

```
/*
```

```
* The reference SPAD initialization procedure determines the minimum
* amount of reference spads to be enables to achieve a target reference
* signal rate and should be performed once during initialization.
*
* Either aperture or non-aperture spads are applied but never both.
* Firstly non-aperture spads are set, begining with 5 spads, and
* increased one spad at a time until the closest measurement to the
* target rate is achieved.
*
* If the target rate is exceeded when 5 non-aperture spads are enabled,
* initialization is performed instead with aperture spads.
*
* When setting spads, a 'Good Spad Map' is applied.
*
* This procedure operates within a SPAD window of interest of a maximum
* 44 spads.
```

```
* The start point is currently fixed to 180, which lies towards the end
* of the non-aperture quadrant and runs in to the adjacent aperture
* quadrant.
*/
```

```
targetRefRate = PALDevDataGet(Dev, targetRefRate);
```

```
/*
* Initialize Spad arrays.
* Currently the good spad map is initialised to 'All good'.
* This is a short term implementation. The good spad map will be
* provided as an input.
* Note that there are 6 bytes. Only the first 44 bits will be used to
* represent spads.
*/
```

```
for (index = 0; index < spadArraySize; index++)
```

```
    Dev->Data.SpadData.RefSpadEnables[index] = 0;
```

```
Status = VL53L0X_WrByte(Dev, 0xFF, 0x01);
```

```
if (Status == VL53L0X_ERROR_NONE)
```

```
    Status = VL53L0X_WrByte(Dev,
```

```
        VL53L0X_REG_DYNAMIC_SPAD_REF_EN_START_OFFSET, 0x00);
```

```
if (Status == VL53L0X_ERROR_NONE)
```

```
    Status = VL53L0X_WrByte(Dev,
```

```
        VL53L0X_REG_DYNAMIC_SPAD_NUM_REQUESTED_REF_SPAD, 0x2C);
```

```

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x00);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev,
        VL53L0X_REG_GLOBAL_CONFIG_REF_EN_START_SELECT,
        startSelect);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev,
        VL53L0X_REG_POWER_MANAGEMENT_GO1_POWER_FORCE, 0);

/* Perform ref calibration */
if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_perform_ref_calibration(Dev, &VhvSettings,
        &PhaseCal, 0);

if (Status == VL53L0X_ERROR_NONE) {
    /* Enable Minimum NON-APERTURE Spads */
    currentSpadIndex = 0;
    lastSpadIndex = currentSpadIndex;
    needAptSpads = 0;
    Status = enable_ref_spads(Dev,
        needAptSpads,
        Dev->Data.SpadData.RefGoodSpadMap,
        Dev->Data.SpadData.RefSpadEnables,
        spadArraySize,

```

```

        startSelect,
        currentSpadIndex,
        minimumSpadCount,
        &lastSpadIndex);
    }

    if (Status == VL53LOX_ERROR_NONE) {
        currentSpadIndex = lastSpadIndex;

        Status = perform_ref_signal_measurement(Dev,
            &peakSignalRateRef);
        if ((Status == VL53LOX_ERROR_NONE) &&
            (peakSignalRateRef > targetRefRate)) {
            /* Signal rate measurement too high,
             * switch to APERTURE SPADs */

            for (index = 0; index < spadArraySize; index++)
                Dev->Data.SpadData.RefSpadEnables[index] = 0;

            /* Increment to the first APERTURE spad */
            while ((is_aperture(startSelect + currentSpadIndex)
                == 0) && (currentSpadIndex < maxSpadCount)) {
                currentSpadIndex++;
            }

            needAptSpads = 1;

            Status = enable_ref_spads(Dev,

```

```

        needAptSpads,
        Dev->Data.SpadData.RefGoodSpadMap,
        Dev->Data.SpadData.RefSpadEnables,
        spadArraySize,
        startSelect,
        currentSpadIndex,
        minimumSpadCount,
        &lastSpadIndex);

    if (Status == VL53L0X_ERROR_NONE) {
        currentSpadIndex = lastSpadIndex;
        Status = perform_ref_signal_measurement(Dev,
                                                &peakSignalRateRef);

        if ((Status == VL53L0X_ERROR_NONE) &&
            (peakSignalRateRef > targetRefRate)) {
            /* Signal rate still too high after
             * setting the minimum number of
             * APERTURE spads. Can do no more
             * therefore set the min number of
             * aperture spads as the result.
             */
            isApertureSpads_int = 1;
            refSpadCount_int = minimumSpadCount;
        }
    }
} else {
    needAptSpads = 0;
}

```

```
}
```

```
if ((Status == VL53L0X_ERROR_NONE) &&  
    (peakSignalRateRef < targetRefRate)) {  
    /* At this point, the minimum number of either aperture  
     * or non-aperture spads have been set. Proceed to add  
     * spads and perform measurements until the target  
     * reference is reached.  
     */  
    isApertureSpads_int = needAptSpads;  
    refSpadCount_int     = minimumSpadCount;  
  
    memcpy(lastSpadArray, Dev->Data.SpadData.RefSpadEnables,  
           spadArraySize);  
    lastSignalRateDiff = abs(peakSignalRateRef -  
                             targetRefRate);  
    complete = 0;  
  
    while (!complete) {  
        get_next_good_spad(  
            Dev->Data.SpadData.RefGoodSpadMap,  
            spadArraySize, currentSpadIndex,  
            &nextGoodSpad);  
  
        if (nextGoodSpad == -1) {  
            Status = VL53L0X_ERROR_REF_SPAD_INIT;  
            break;  
        }  
    }  
}
```

```

(refSpadCount_int)++;

/* Cannot combine Aperture and Non-Aperture spads, so
 * ensure the current spad is of the correct type.
 */
if (is_aperture((uint32_t)startSelect + nextGoodSpad) !=
    needAptSpads) {
    Status = VL53L0X_ERROR_REF_SPAD_INIT;
    break;
}

currentSpadIndex = nextGoodSpad;
Status = enable_spad_bit(
    Dev->Data.SpadData.RefSpadEnables,
    spadArraySize, currentSpadIndex);

if (Status == VL53L0X_ERROR_NONE) {
    currentSpadIndex++;
    /* Proceed to apply the additional spad and
     * perform measurement. */
    Status = set_ref_spad_map(Dev,
        Dev->Data.SpadData.RefSpadEnables);
}

if (Status != VL53L0X_ERROR_NONE)
    break;

Status = perform_ref_signal_measurement(Dev,
    &peakSignalRateRef);

```

```

if (Status != VL53L0X_ERROR_NONE)

    break;

signalRateDiff = abs(peakSignalRateRef - targetRefRate);

if (peakSignalRateRef > targetRefRate) {

    /* Select the spad map that provides the
     * measurement closest to the target rate,
     * either above or below it.
     */

    if (signalRateDiff > lastSignalRateDiff) {

        /* Previous spad map produced a closer
         * measurement, so choose this. */

        Status = set_ref_spad_map(Dev,
                                lastSpadArray);

        memcpy(
            Dev->Data.SpadData.RefSpadEnables,
            lastSpadArray, spadArraySize);

        (refSpadCount_int)--;

    }

    complete = 1;
} else {

    /* Continue to add spads */

    lastSignalRateDiff = signalRateDiff;

    memcpy(lastSpadArray,
           Dev->Data.SpadData.RefSpadEnables,
           spadArraySize);

```

```

        }

        } /* while */
    }

    if (Status == VL53L0X_ERROR_NONE) {
        *refSpadCount = refSpadCount_int;
        *isApertureSpads = isApertureSpads_int;

        VL53L0X_SETDEVICESPECIFICPARAMETER(Dev, RefSpadsInitialised, 1);
        VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
            ReferenceSpadCount, (uint8_t)(*refSpadCount));
        VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
            ReferenceSpadType, *isApertureSpads);
    }

    return Status;
}

VL53L0X_Error VL53L0X_set_reference_spads(VL53L0X_DEV Dev,
    uint32_t count, uint8_t isApertureSpads)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint32_t currentSpadIndex = 0;
    uint8_t startSelect = 0xB4;
    uint32_t spadArraySize = 6;
    uint32_t maxSpadCount = 44;
    uint32_t lastSpadIndex;
    uint32_t index;

```

```

/*
 * This function applies a requested number of reference spads, either
 * aperture or
 * non-aperture, as requested.
 * The good spad map will be applied.
 */

Status = VL53L0X_WrByte(Dev, 0xFF, 0x01);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev,
        VL53L0X_REG_DYNAMIC_SPAD_REF_EN_START_OFFSET, 0x00);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev,
        VL53L0X_REG_DYNAMIC_SPAD_NUM_REQUESTED_REF_SPAD, 0x2C);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev, 0xFF, 0x00);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev,
        VL53L0X_REG_GLOBAL_CONFIG_REF_EN_START_SELECT,
        startSelect);

for (index = 0; index < spadArraySize; index++)
    Dev->Data.SpadData.RefSpadEnables[index] = 0;

```

```

if (isApertureSpads) {
    /* Increment to the first APERTURE spad */
    while ((is_aperture(startSelect + currentSpadIndex) == 0) &&
           (currentSpadIndex < maxSpadCount)) {
        currentSpadIndex++;
    }
}

Status = enable_ref_spads(Dev,
                           isApertureSpads,
                           Dev->Data.SpadData.RefGoodSpadMap,
                           Dev->Data.SpadData.RefSpadEnables,
                           spadArraySize,
                           startSelect,
                           currentSpadIndex,
                           count,
                           &lastSpadIndex);

if (Status == VL53L0X_ERROR_NONE) {
    VL53L0X_SETDEVICESPECIFICPARAMETER(Dev, RefSpadsInitialised, 1);
    VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
        ReferenceSpadCount, (uint8_t)(count));
    VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
        ReferenceSpadType, isApertureSpads);
}

return Status;
}

```

```

VL53L0X_Error VL53L0X_get_reference_spads(VL53L0X_DEV Dev,

```

```

        uint32_t *pSpadCount, uint8_t *pIsApertureSpads)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    uint8_t refSpadsInitialised;

    uint8_t refSpadArray[6];

    uint32_t cMaxSpadCount = 44;

    uint32_t cSpadArraySize = 6;

    uint32_t spadsEnabled;

    uint8_t isApertureSpads = 0;

    refSpadsInitialised = VL53L0X_GETDEVICESPECIFICPARAMETER(Dev,
                                                                RefSpadsInitialised);

    if (refSpadsInitialised == 1) {

        *pSpadCount = (uint32_t)VL53L0X_GETDEVICESPECIFICPARAMETER(Dev,
                                                                    ReferenceSpadCount);

        *pIsApertureSpads = VL53L0X_GETDEVICESPECIFICPARAMETER(Dev,
                                                                ReferenceSpadType);
    } else {

        /* obtain spad info from device.*/

        Status = get_ref_spad_map(Dev, refSpadArray);

        if (Status == VL53L0X_ERROR_NONE) {

            /* count enabled spads within spad map array and
            * determine if Aperture or Non-Aperture.
            */

            Status = count_enabled_spads(refSpadArray,

```

```

        cSpadArraySize,
        cMaxSpadCount,
        &spadsEnabled,
        &isApertureSpads);

    if (Status == VL53L0X_ERROR_NONE) {

        *pSpadCount = spadsEnabled;
        *pIsApertureSpads = isApertureSpads;

        VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
            RefSpadsInitialised, 1);
        VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
            ReferenceSpadCount,
            (uint8_t)spadsEnabled);
        VL53L0X_SETDEVICESPECIFICPARAMETER(Dev,
            ReferenceSpadType, isApertureSpads);
    }
}

return Status;
}

```

```

VL53L0X_Error VL53L0X_perform_single_ref_calibration(VL53L0X_DEV Dev,
    uint8_t vhw_init_byte)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

```

```

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSRANGE_START,
        VL53L0X_REG_SYSRANGE_MODE_START_STOP |
        vhw_init_byte);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_measurement_poll_for_completion(Dev);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_ClearInterruptMask(Dev, 0);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSRANGE_START, 0x00);

return Status;
}

```

```

VL53L0X_Error VL53L0X_ref_calibration_io(VL53L0X_DEV Dev, uint8_t read_not_write,
    uint8_t VhwSettings, uint8_t PhaseCal,
    uint8_t *pVhwSettings, uint8_t *pPhaseCal,
    const uint8_t vhw_enable, const uint8_t phase_enable)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t PhaseCalint = 0;

    /* Read VHV from device */
    Status |= VL53L0X_WrByte(Dev, 0xFF, 0x01);

```

```

Status |= VL53L0X_WrByte(Dev, 0x00, 0x00);
Status |= VL53L0X_WrByte(Dev, 0xFF, 0x00);

if (read_not_write) {
    if (vhv_enable)
        Status |= VL53L0X_RdByte(Dev, 0xCB, pVhvSettings);
    if (phase_enable)
        Status |= VL53L0X_RdByte(Dev, 0xEE, &PhaseCalint);
} else {
    if (vhv_enable)
        Status |= VL53L0X_WrByte(Dev, 0xCB, VhvSettings);
    if (phase_enable)
        Status |= VL53L0X_UpdateByte(Dev, 0xEE, 0x80, PhaseCal);
}

Status |= VL53L0X_WrByte(Dev, 0xFF, 0x01);
Status |= VL53L0X_WrByte(Dev, 0x00, 0x01);
Status |= VL53L0X_WrByte(Dev, 0xFF, 0x00);

*pPhaseCal = (uint8_t)(PhaseCalint&0xEF);

return Status;
}

VL53L0X_Error VL53L0X_perform_vhv_calibration(VL53L0X_DEV Dev,
    uint8_t *pVhvSettings, const uint8_t get_data_enable,
    const uint8_t restore_config)
{

```

```

VL53L0X_Error Status = VL53L0X_ERROR_NONE;

uint8_t SequenceConfig = 0;

uint8_t VhvSettings = 0;

uint8_t PhaseCal = 0;

uint8_t PhaseCalInt = 0;


/* store the value of the sequence config,
 * this will be reset before the end of the function
 */

if (restore_config)
    SequenceConfig = PALDevDataGet(Dev, SequenceConfig);


/* Run VHV */

Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG, 0x01);


if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_perform_single_ref_calibration(Dev, 0x40);


/* Read VHV from device */
if ((Status == VL53L0X_ERROR_NONE) && (get_data_enable == 1)) {
    Status = VL53L0X_ref_calibration_io(Dev, 1,
        VhvSettings, PhaseCal, /* Not used here */
        pVhvSettings, &PhaseCalInt,
        1, 0);
} else
    *pVhvSettings = 0;

```

```

if ((Status == VL53L0X_ERROR_NONE) && restore_config) {
    /* restore the previous Sequence Config */
    Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG,
                           SequenceConfig);
    if (Status == VL53L0X_ERROR_NONE)
        PALDevDataSet(Dev, SequenceConfig, SequenceConfig);

}

return Status;
}

VL53L0X_Error VL53L0X_perform_phase_calibration(VL53L0X_DEV Dev,
        uint8_t *pPhaseCal, const uint8_t get_data_enable,
        const uint8_t restore_config)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t SequenceConfig = 0;
    uint8_t VhvSettings = 0;
    uint8_t PhaseCal = 0;
    uint8_t VhvSettingsint;

    /* store the value of the sequence config,
     * this will be reset before the end of the function
     */

    if (restore_config)
        SequenceConfig = PALDevDataGet(Dev, SequenceConfig);

```

```

/* Run PhaseCal */
Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG, 0x02);

if (Status == VL53L0X_ERROR_NONE)
    Status = VL53L0X_perform_single_ref_calibration(Dev, 0x0);

/* Read PhaseCal from device */
if ((Status == VL53L0X_ERROR_NONE) && (get_data_enable == 1)) {
    Status = VL53L0X_ref_calibration_io(Dev, 1,
        VhvSettings, PhaseCal, /* Not used here */
        &VhvSettingsint, pPhaseCal,
        0, 1);
} else
    *pPhaseCal = 0;

if ((Status == VL53L0X_ERROR_NONE) && restore_config) {
    /* restore the previous Sequence Config */
    Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG,
        SequenceConfig);
    if (Status == VL53L0X_ERROR_NONE)
        PALDevDataSet(Dev, SequenceConfig, SequenceConfig);
}

return Status;
}

```

VL53L0X_Error VL53L0X_perform_ref_calibration(VL53L0X_DEV Dev,

```

uint8_t *pVhvSettings, uint8_t *pPhaseCal, uint8_t get_data_enable)
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;
    uint8_t SequenceConfig = 0;

    /* store the value of the sequence config,
     * this will be reset before the end of the function
     */

    SequenceConfig = PALDevDataGet(Dev, SequenceConfig);

    /* In the following function we don't save the config to optimize
     * writes on device. Config is saved and restored only once. */
    Status = VL53L0X_perform_vhv_calibration(
        Dev, pVhvSettings, get_data_enable, 0);

    if (Status == VL53L0X_ERROR_NONE)
        Status = VL53L0X_perform_phase_calibration(
            Dev, pPhaseCal, get_data_enable, 0);

    if (Status == VL53L0X_ERROR_NONE) {
        /* restore the previous Sequence Config */
        Status = VL53L0X_WrByte(Dev, VL53L0X_REG_SYSTEM_SEQUENCE_CONFIG,
            SequenceConfig);
        if (Status == VL53L0X_ERROR_NONE)
            PALDevDataSet(Dev, SequenceConfig, SequenceConfig);
    }
}

```

```
return Status;
```

[illegible]

```

        pVhvSettings, pPhaseCal,
        1, 1);

    return Status;
}

```

1.1.42 Adafruit_vl53l0x.h

Below:

```

/*****

```

This is a library for the Adafruit VL53L0X Sensor Breakout

Designed specifically to work with the VL53L0X sensor from Adafruit

----> <https://www.adafruit.com/products/3317>

These sensors use I2C to communicate, 2 pins are required to
interface

Adafruit invests time and resources providing this open source code,
please support Adafruit and open-source hardware by purchasing
products from Adafruit!

Written by Limor Fried/Ladyada for Adafruit Industries.

BSD license, all text above must be included in any redistribution

```

*****/

```

```

#if ( ARDUINO >= 100 )
    #include "Arduino.h"
#else
    #include "WProgram.h"
#endif

```

```

#include "Wire.h"

#include "vl53l0x_api.h"

#define VL53L0X_I2C_ADDR 0x29

class Adafruit_VL53L0X
{
public:
    boolean    begin( boolean debug = false );

    VL53L0X_Error

    rangingTest(VL53L0X_RangingMeasurementData_t* pRangingMeasurementData,
                boolean debug = false)

    { getSingleRangingMeasurement(pRangingMeasurementData, debug); };

    VL53L0X_Error getSingleRangingMeasurement( VL53L0X_RangingMeasurementData_t*
pRangingMeasurementData, boolean debug = false );

    void    printRangeStatus( VL53L0X_RangingMeasurementData_t* pRangingMeasurementData );

    VL53L0X_Error    Status    = VL53L0X_ERROR_NONE;

private:
    VL53L0X_Dev_t    MyDevice;
    VL53L0X_Dev_t    *pMyDevice = &MyDevice;
    VL53L0X_Version_t    Version;
    VL53L0X_Version_t    *pVersion = &Version;
    VL53L0X_DeviceInfo_t    DeviceInfo;
};

```

[1.1.43 Adafruit_vl53l0x.cpp](#)

Below:

```
#include "Adafruit_VL53L0X.h"
```

```
#define VERSION_REQUIRED_MAJOR 1
```

```
#define VERSION_REQUIRED_MINOR 0
```

```
#define VERSION_REQUIRED_BUILD 1
```

```
#define STR_HELPER( x ) #x
```

```
#define STR( x )      STR_HELPER(x)
```

```
boolean Adafruit_VL53L0X::begin( boolean debug ) {
```

```
    int32_t status_int;
```

```
    int32_t init_done      = 0;
```

```
    uint32_t refSpadCount;
```

```
    uint8_t  isApertureSpads;
```

```
    uint8_t  VhvSettings;
```

```
    uint8_t  PhaseCal;
```

```
    // Initialize Comms
```

```
    pMyDevice->I2cDevAddr  = VL53L0X_I2C_ADDR; // 7 bit addr
```

```
    pMyDevice->comms_type   = 1;
```

```
    pMyDevice->comms_speed_khz = 400;
```

```
    Wire.begin(); // VL53L0X_i2c_init();
```

```
    // unclear if this is even needed:
```

```
    if( VL53L0X_IMPLEMENTATION_VER_MAJOR != VERSION_REQUIRED_MAJOR ||
```

```
        VL53L0X_IMPLEMENTATION_VER_MINOR != VERSION_REQUIRED_MINOR ||
```

```

VL53L0X_IMPLEMENTATION_VER_SUB != VERSION_REQUIRED_BUILD ) {

    if( debug ) {

        Serial.println( F( "Found " STR(VL53L0X_IMPLEMENTATION_VER_MAJOR) "."
STR(VL53L0X_IMPLEMENTATION_VER_MINOR) "." STR(VL53L0X_IMPLEMENTATION_VER_SUB) " rev "
STR(VL53L0X_IMPLEMENTATION_VER_REVISION) ) );

        Serial.println( F( "Requires " STR(VERSION_REQUIRED_MAJOR) "."
STR(VERSION_REQUIRED_MINOR) "." STR(VERSION_REQUIRED_BUILD) ) );

    }

    Status = VL53L0X_ERROR_NOT_SUPPORTED;

    return false;
}

```

```

Status = VL53L0X_DataInit( &MyDevice );    // Data initialization

```

```

Status = VL53L0X_GetDeviceInfo( &MyDevice, &DeviceInfo );

```

```

if( Status == VL53L0X_ERROR_NONE ) {

    if( debug ) {

        Serial.println( F( "VL53L0X Info:" ) );

        Serial.print( F( "Device Name: " ) ); Serial.print( DeviceInfo.Name );

        Serial.print( F( ", Type: " ) ); Serial.print( DeviceInfo.Type );

        Serial.print( F( ", ID: " ) ); Serial.println( DeviceInfo.ProductId );

        Serial.print( F( "Rev Major: " ) ); Serial.print( DeviceInfo.ProductRevisionMajor );

        Serial.print( F( ", Minor: " ) ); Serial.println( DeviceInfo.ProductRevisionMinor );

    }

}

```

```

if( ( DeviceInfo.ProductRevisionMinor != 1 ) && ( DeviceInfo.ProductRevisionMinor != 1 ) ) {

```

```

    if( debug ) {
        Serial.print( F( "Error expected cut 1.1 but found " ) );
        Serial.print( DeviceInfo.ProductRevisionMajor );
        Serial.print( ',' );
        Serial.println( DeviceInfo.ProductRevisionMinor );
    }

    Status = VL53L0X_ERROR_NOT_SUPPORTED;
}

if( Status == VL53L0X_ERROR_NONE ) {
    if( debug ) {
        Serial.println( F( "VL53L0X: StaticInit" ) );
    }

    Status = VL53L0X_StaticInit( pMyDevice ); // Device Initialization
}

if( Status == VL53L0X_ERROR_NONE ) {
    if( debug ) {
        Serial.println( F( "VL53L0X: PerformRefSpadManagement" ) );
    }

    Status = VL53L0X_PerformRefSpadManagement( pMyDevice, &refSpadCount, &isApertureSpads ); //
    Device Initialization

    if( debug ) {
        Serial.print( F( "refSpadCount = " ) );

```

```

        Serial.print( refSpadCount );

        Serial.print( F( " , isApertureSpads = " ) );

        Serial.println( isApertureSpads );
    }
}

if( Status == VL53L0X_ERROR_NONE ) {
    if( debug ) {
        Serial.println( F( "VL53L0X: PerformRefCalibration" ) );
    }

    Status = VL53L0X_PerformRefCalibration( pMyDevice, &VhvSettings, &PhaseCal );    // Device
Initialization
}

if( Status == VL53L0X_ERROR_NONE ) {
    // no need to do this when we use VL53L0X_PerformSingleRangingMeasurement
    if( debug ) {
        Serial.println( F( "VL53L0X: SetDeviceMode" ) );
    }

    Status = VL53L0X_SetDeviceMode( pMyDevice, VL53L0X_DEVICEMODE_SINGLE_RANGING );    //
Setup in single ranging mode
}

// Enable/Disable Sigma and Signal check
if( Status == VL53L0X_ERROR_NONE ) {
    Status = VL53L0X_SetLimitCheckEnable( pMyDevice, VL53L0X_CHECKENABLE_SIGMA_FINAL_RANGE,
1 );
}

```

```
if( Status == VL53L0X_ERROR_NONE ) {  
    Status = VL53L0X_SetLimitCheckEnable( pMyDevice,  
VL53L0X_CHECKENABLE_SIGNAL_RATE_FINAL_RANGE, 1 );  
}
```

```
if( Status == VL53L0X_ERROR_NONE ) {  
    Status = VL53L0X_SetLimitCheckEnable( pMyDevice,  
VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD, 1 );  
}
```

```
if( Status == VL53L0X_ERROR_NONE ) {  
    Status = VL53L0X_SetLimitCheckValue( pMyDevice,  
VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD, (FixPoint1616_t)( 1.5 * 0.023 * 65536 ) );  
}
```

```
if( Status == VL53L0X_ERROR_NONE ) {  
    return true;  
} else {  
    if( debug ) {  
        Serial.print( F( "VL53L0X Error: " ) );  
        Serial.println( Status );  
    }  
}
```

```
    return false;  
}  
}
```

```

VL53L0X_Error Adafruit_VL53L0X::getSingleRangingMeasurement(
VL53L0X_RangingMeasurementData_t *RangingMeasurementData, boolean debug )
{
    VL53L0X_Error Status = VL53L0X_ERROR_NONE;

    FixPoint1616_t LimitCheckCurrent;

    /*
    * Step 4 : Test ranging mode
    */

    if( Status == VL53L0X_ERROR_NONE ) {
        if( debug ) {
            Serial.println( F( "sVL53L0X: PerformSingleRangingMeasurement" ) );
        }
        Status = VL53L0X_PerformSingleRangingMeasurement( pMyDevice, RangingMeasurementData );

        if( debug ) {
            printRangeStatus( RangingMeasurementData );
        }

        if( debug ) {
            VL53L0X_GetLimitCheckCurrent( pMyDevice,
            VL53L0X_CHECKENABLE_RANGE_IGNORE_THRESHOLD, &LimitCheckCurrent );

            Serial.print( F( "RANGE IGNORE THRESHOLD: " ) );
            Serial.println( (float)LimitCheckCurrent / 65536.0 );

            Serial.print( F( "Measured distance: " ) );

```

```

        Serial.println( RangingMeasurementData->RangeMilliMeter );
    }
}

return Status;
}

```

```

void Adafruit_VL53L0X::printRangeStatus( VL53L0X_RangingMeasurementData_t*
pRangingMeasurementData )
{
    char buf[ VL53L0X_MAX_STRING_LENGTH ];
    uint8_t RangeStatus;

    /*
     * New Range Status: data is valid when pRangingMeasurementData->RangeStatus = 0
     */

    RangeStatus = pRangingMeasurementData->RangeStatus;

    VL53L0X_GetRangeStatusString( RangeStatus, buf );

    Serial.print( F("Range Status: " ) );
    Serial.print( RangeStatus );
    Serial.print( F(" : " ) );
    Serial.println( buf );
}

```

}

1.1.44 vl53l0x_interrupt_threshold_settings.h

Below:

/*****

Copyright © 2016, STMicroelectronics International N.V.

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- * Neither the name of STMicroelectronics nor the
names of its contributors may be used to endorse or promote products
derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND
NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED.

IN NO EVENT SHALL STMICROELECTRONICS INTERNATIONAL N.V. BE LIABLE FOR ANY
DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

*****/

```
#ifndef _VL53L0X_INTERRUPT_THRESHOLD_SETTINGS_H_
```

```
#define _VL53L0X_INTERRUPT_THRESHOLD_SETTINGS_H_
```

```
#ifdef __cplusplus
```

```
extern "C" {
```

```
#endif
```

```
uint8_t InterruptThresholdSettings[] = {
```

```
    /* Start of Interrupt Threshold Settings */
```

```
    0x1, 0xff, 0x00,
```

```
    0x1, 0x80, 0x01,
```

```
    0x1, 0xff, 0x01,
```

```
    0x1, 0x00, 0x00,
```

```
    0x1, 0xff, 0x01,
```

```
    0x1, 0x4f, 0x02,
```

```
    0x1, 0xFF, 0x0E,
```

```
    0x1, 0x00, 0x03,
```

```
    0x1, 0x01, 0x84,
```

```
    0x1, 0x02, 0x0A,
```

```
    0x1, 0x03, 0x03,
```

```
    0x1, 0x04, 0x08,
```

0x1, 0x05, 0xC8,
0x1, 0x06, 0x03,
0x1, 0x07, 0x8D,
0x1, 0x08, 0x08,
0x1, 0x09, 0xC6,
0x1, 0x0A, 0x01,
0x1, 0x0B, 0x02,
0x1, 0x0C, 0x00,
0x1, 0x0D, 0xD5,
0x1, 0x0E, 0x18,
0x1, 0x0F, 0x12,
0x1, 0x10, 0x01,
0x1, 0x11, 0x82,
0x1, 0x12, 0x00,
0x1, 0x13, 0xD5,
0x1, 0x14, 0x18,
0x1, 0x15, 0x13,
0x1, 0x16, 0x03,
0x1, 0x17, 0x86,
0x1, 0x18, 0x0A,
0x1, 0x19, 0x09,
0x1, 0x1A, 0x08,
0x1, 0x1B, 0xC2,
0x1, 0x1C, 0x03,
0x1, 0x1D, 0x8F,
0x1, 0x1E, 0x0A,
0x1, 0x1F, 0x06,
0x1, 0x20, 0x01,
0x1, 0x21, 0x02,

0x1, 0x22, 0x00,
0x1, 0x23, 0xD5,
0x1, 0x24, 0x18,
0x1, 0x25, 0x22,
0x1, 0x26, 0x01,
0x1, 0x27, 0x82,
0x1, 0x28, 0x00,
0x1, 0x29, 0xD5,
0x1, 0x2A, 0x18,
0x1, 0x2B, 0x0B,
0x1, 0x2C, 0x28,
0x1, 0x2D, 0x78,
0x1, 0x2E, 0x28,
0x1, 0x2F, 0x91,
0x1, 0x30, 0x00,
0x1, 0x31, 0x0B,
0x1, 0x32, 0x00,
0x1, 0x33, 0x0B,
0x1, 0x34, 0x00,
0x1, 0x35, 0xA1,
0x1, 0x36, 0x00,
0x1, 0x37, 0xA0,
0x1, 0x38, 0x00,
0x1, 0x39, 0x04,
0x1, 0x3A, 0x28,
0x1, 0x3B, 0x30,
0x1, 0x3C, 0x0C,
0x1, 0x3D, 0x04,
0x1, 0x3E, 0x0F,

0x1, 0x3F, 0x79,
0x1, 0x40, 0x28,
0x1, 0x41, 0x1E,
0x1, 0x42, 0x2F,
0x1, 0x43, 0x87,
0x1, 0x44, 0x00,
0x1, 0x45, 0x0B,
0x1, 0x46, 0x00,
0x1, 0x47, 0x0B,
0x1, 0x48, 0x00,
0x1, 0x49, 0xA7,
0x1, 0x4A, 0x00,
0x1, 0x4B, 0xA6,
0x1, 0x4C, 0x00,
0x1, 0x4D, 0x04,
0x1, 0x4E, 0x01,
0x1, 0x4F, 0x00,
0x1, 0x50, 0x00,
0x1, 0x51, 0x80,
0x1, 0x52, 0x09,
0x1, 0x53, 0x08,
0x1, 0x54, 0x01,
0x1, 0x55, 0x00,
0x1, 0x56, 0x0F,
0x1, 0x57, 0x79,
0x1, 0x58, 0x09,
0x1, 0x59, 0x05,
0x1, 0x5A, 0x00,
0x1, 0x5B, 0x60,

0x1, 0x5C, 0x05,
0x1, 0x5D, 0xD1,
0x1, 0x5E, 0x0C,
0x1, 0x5F, 0x3C,
0x1, 0x60, 0x00,
0x1, 0x61, 0xD0,
0x1, 0x62, 0x0B,
0x1, 0x63, 0x03,
0x1, 0x64, 0x28,
0x1, 0x65, 0x10,
0x1, 0x66, 0x2A,
0x1, 0x67, 0x39,
0x1, 0x68, 0x0B,
0x1, 0x69, 0x02,
0x1, 0x6A, 0x28,
0x1, 0x6B, 0x10,
0x1, 0x6C, 0x2A,
0x1, 0x6D, 0x61,
0x1, 0x6E, 0x0C,
0x1, 0x6F, 0x00,
0x1, 0x70, 0x0F,
0x1, 0x71, 0x79,
0x1, 0x72, 0x00,
0x1, 0x73, 0x0B,
0x1, 0x74, 0x00,
0x1, 0x75, 0x0B,
0x1, 0x76, 0x00,
0x1, 0x77, 0xA1,
0x1, 0x78, 0x00,

```
    0x1, 0x79, 0xA0,  
    0x1, 0x7A, 0x00,  
    0x1, 0x7B, 0x04,  
    0x1, 0xFF, 0x04,  
    0x1, 0x79, 0x1D,  
    0x1, 0x7B, 0x27,  
    0x1, 0x96, 0x0E,  
    0x1, 0x97, 0xFE,  
    0x1, 0x98, 0x03,  
    0x1, 0x99, 0xEF,  
    0x1, 0x9A, 0x02,  
    0x1, 0x9B, 0x44,  
    0x1, 0x73, 0x07,  
    0x1, 0x70, 0x01,  
    0x1, 0xff, 0x01,  
    0x1, 0x00, 0x01,  
    0x1, 0xff, 0x00,  
    0x00, 0x00, 0x00  
};
```

```
#ifdef __cplusplus
```

```
}
```

```
#endif
```

```
#endif /* _VL53L0X_INTERRUPT_THRESHOLD_SETTINGS_H_ */
```

1.1.45 vl53l0x_i2c_platform.h

Below:

```
#include "Arduino.h"
```

```
#include "Wire.h"
```

```
// initialize I2C
```

```
int VL53L0X_i2c_init(void);
```

```
int VL53L0X_write_multi(uint8_t deviceAddress, uint8_t index, uint8_t *pdata, uint32_t count);
```

```
int VL53L0X_read_multi(uint8_t deviceAddress, uint8_t index, uint8_t *pdata, uint32_t count);
```

```
int VL53L0X_write_byte(uint8_t deviceAddress, uint8_t index, uint8_t data);
```

```
int VL53L0X_write_word(uint8_t deviceAddress, uint8_t index, uint16_t data);
```

```
int VL53L0X_write_dword(uint8_t deviceAddress, uint8_t index, uint32_t data);
```

```
int VL53L0X_read_byte(uint8_t deviceAddress, uint8_t index, uint8_t *data);
```

```
int VL53L0X_read_word(uint8_t deviceAddress, uint8_t index, uint16_t *data);
```

```
int VL53L0X_read_dword(uint8_t deviceAddress, uint8_t index, uint32_t *data);
```

1.1.46