Fault Tolerant Sequencer

For the

Naval Postgraduate School Cubesat Launcher (NPSCUL)

By

Justin Jordan

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2013
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>iii</td>
</tr>
<tr>
<td>Abstract</td>
<td>iv</td>
</tr>
<tr>
<td>I. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>II. Background</td>
<td>4</td>
</tr>
<tr>
<td>III. Requirements</td>
<td>7</td>
</tr>
<tr>
<td>IV. Design</td>
<td></td>
</tr>
<tr>
<td>A. Hardware</td>
<td>8</td>
</tr>
<tr>
<td>B. Software</td>
<td>15</td>
</tr>
<tr>
<td>V. Testing</td>
<td>20</td>
</tr>
<tr>
<td>VI. Conclusions and Recommendation</td>
<td>27</td>
</tr>
<tr>
<td>VII. Bibliography</td>
<td>29</td>
</tr>
</tbody>
</table>

## Appendices

### A. Schematics
- A.1 Propeller A                                        | 1    |
- A.2 Propeller B                                         | 2    |
- A.3 Propeller C                                         | 3    |
- A.4 NEA Voting Logic 1-4                                | 4    |
- A.5 NEA Voting Logic 5-8                                | 5    |
- A.6 Voltage Regulation/Serial Data Voting Logic          | 6    |
- A.7 Programming Ports                                   | 7    |

### B. Bill Of Materials                                  | 8    |

### C. Sequencer_rev0.8 Flowchart                          | 9    |

### D. Source Code Listing                                 |      |
- C.1 Sequencer_rev0.8                                    | 11   |
- C.2 Sequencer_Telemetry_rev0.3                          | 16   |
- C.3 FullDuplexSerial                                     | 18   |
# LIST OF TABLES AND FIGURES

## Table

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Small Satellite Classification By Mass</td>
<td>1</td>
</tr>
<tr>
<td>2. List of Single Event Effects</td>
<td>5</td>
</tr>
<tr>
<td>3. List of approved requirements</td>
<td>7</td>
</tr>
<tr>
<td>4. Propeller chip input/output configuration</td>
<td>12</td>
</tr>
</tbody>
</table>

## Figures

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. P-POD next to 1U structural model of Cubesat</td>
<td>2</td>
</tr>
<tr>
<td>2. NPSCUL integrated with Atlas V launch vehicle</td>
<td>3</td>
</tr>
<tr>
<td>3. Block diagram of Propeller chip</td>
<td>9</td>
</tr>
<tr>
<td>4. Conceptual diagram of cog/hub interaction</td>
<td>9</td>
</tr>
<tr>
<td>5. Sequencer block diagram</td>
<td>10</td>
</tr>
<tr>
<td>6. Sequencer redundant voltage regulation</td>
<td>11</td>
</tr>
<tr>
<td>7. Schematic diagram of Propeller interface</td>
<td>14</td>
</tr>
<tr>
<td>8. Voting logic for sequencer outputs</td>
<td>14</td>
</tr>
<tr>
<td>9. Sequencer object structure</td>
<td>15</td>
</tr>
<tr>
<td>10. Breadboard prototype of sequencer</td>
<td>20</td>
</tr>
<tr>
<td>11. Time deviation of Propeller outputs without a reset</td>
<td>22</td>
</tr>
<tr>
<td>12. Serial data without reset</td>
<td>22</td>
</tr>
<tr>
<td>13. Sequenced output vs. voted result without reset</td>
<td>23</td>
</tr>
<tr>
<td>14. Serial data vs. voted result without reset</td>
<td>24</td>
</tr>
<tr>
<td>15. Time deviation of sequenced output with a reset</td>
<td>25</td>
</tr>
<tr>
<td>16. Serial data with reset</td>
<td>25</td>
</tr>
<tr>
<td>17. Sequenced output vs. voted result with reset</td>
<td>26</td>
</tr>
<tr>
<td>18. Serial data vs. voted result with reset</td>
<td>26</td>
</tr>
</tbody>
</table>
Acknowledgements

I would like to take a few moments to thank the exceptional people that have influenced my life over the last few years. Only through their help and support have I been able to reach this point in my life. First, I would like to thank Dr. James Newman and the staff of the Small Satellite Laboratory at the Naval Postgraduate School. While attending Hartnell College in Salinas, CA, and summers between these last few years at Cal Poly, I have been able to work with Dr. Newman in the laboratory and exercise the theory I have been learning in class to practical problems. Second, I would like to thank my Senior Project advisor Dr. John Oliver and the staff of the Cal Poly Electrical Engineering Department here in San Luis Obispo. The last few years have been filled with inspiring conversations with many of them, and I thank them all. Last, but definitely not least, I would like to thank my girlfriend Maria Barroso and my family. It was Maria’s encouragement and morale support which lead me to pursuing a higher education and obtaining a degree in Electrical Engineering. Over the last six and half years I have been exposed to a whole new world filled with opportunities limited only by myself and I would like to say thank you to you all.
Abstract

The purpose of this report is to describe the design of a low cost fault tolerant sequencer for the Naval Postgraduate School Cubesat Launcher (NPSCUL). Three different approaches are being researched for the design of the fault tolerant sequencer; discrete components consisting of timers, counters, etc., a radiation hardened FPGA and using replicated processors with external voting logic which is the focus of this senior project report. This report will introduce what Cubesats are and the current methods of integrating them to a launch vehicle. Then a brief background on space radiation and its effects on electronics will be given. Finally the design of a prototype for implementing a fault tolerant sequencer through replication to address the problems caused by space radiation will be discussed.
I. Introduction

Cubesats are defined as either pico, or nanosatellites with the purpose of enabling university students in the design, construction, testing and operation of a flight capable satellite. These miniature satellites have a default volume of one liter, or 10 cm cubed and a mass of approximately 1.33 kilograms. Cubesats of this size are known as 1U, or one unit, with 2U and 3U configurations being possible. In 1999, Professor Jordi Puig-Suari of Cal Poly and Bob Twiggs of Stanford University proposed the Cubesat standard which has been continuously updated since [4]. Table 1 shows the categorization of satellites based on their mass.

<table>
<thead>
<tr>
<th>Classification</th>
<th>Mass (kg) (wet mass including fuel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minisatellite</td>
<td>100 – 500</td>
</tr>
<tr>
<td>Microsatellite</td>
<td>10 - 100</td>
</tr>
<tr>
<td>Nanosatellite</td>
<td>1 - 10</td>
</tr>
<tr>
<td>Picosatellite</td>
<td>0.1 - 1</td>
</tr>
<tr>
<td>Femtosatellite</td>
<td>0.01 – 0.1</td>
</tr>
</tbody>
</table>

To bring down the cost of development for universities, several avenues have been taken in the development of Cubesats. Cubesats are normally designed with commercial off-the-shelf (COTS) electronics reducing development time and cost. Due to their small size, more than one Cubesat is normally integrated with a launch vehicle allowing universities to share the cost of the launch. Additionally, Cubesats are considered low priority customers on a launch vehicle and the majority of the cost for a launch is being funded by the primary customer/payload. These methods have potentially brought down the cost of developing and launching a Cubesat to approximately $65,000 - $80,000, which is considerably less than developing a commercial satellite (~$300,000,000) [6].
Currently there are two ways that have been successfully used to integrate Cubesats to a launch vehicle. The first is the Poly-Picosatellite Orbital Deployer (P-POD) developed by Cal Poly. The P-POD is a spring loaded canister capable of integrating up to three 1U Cubesats, or any combination of Cubesats resulting in a volume of 3U. After the primary payload has been successfully deployed and the launch vehicle reaches the designated orbit for deploying the Cubesats, a Non Explosive Actuator (NEA) which secures the door of the P-POD is enabled by the launch vehicle and the Cubesats are literally pushed out into space. Figure 1 shows a P-POD with a structural model of a 1U Cubesat.

![Figure 1: P-POD next to 1U structural model of Cubesat](image)

NPSCUL is the second method known to the author that has been successfully used to integrate Cubesats to a launch vehicle. NPSCUL is best described by a box that actually integrates up to eight P-PODs to the launch vehicle allowing for any combination of 24U

2
Cubesats to be deployed on a given mission. NPSCUL has greatly advanced launch possibilities for universities through availability and cost reduction. NPSCUL had its maiden flight on September 14, 2012 on an Atlas V launch vehicle operated by United Launch Alliance (ULA) from Space Launch Complex 3E at Vandenberg Air Force Base, California. The author had the opportunity of experiencing the launch which was an exceptional experience and would like to say thank you to the people who made it possible. Figure 2 shows NPSCUL integrated with the launch vehicle.

![NPSCUL integrated with Atlas V launch vehicle](image)

Figure 2: NPSCUL integrated with Atlas V launch vehicle

The sequence of events for deploying Cubesats when using NPSCUL is the same as when using P-PODs, because NPSCUL integrates P-PODs to the launch vehicle. The only difference is the possibility of the launch vehicle having to sequence up to eight P-PODs instead of one. This could possibly be a problem for NPSCUL, because in some missions the electrical signals needed to enable the NEAs on the P-PODS might not be available from the launch vehicle to NPSCUL. Therefore, NPSCUL needs its own sequencer capable of sequencing all eight P-PODS.
II. Background

There are many challenges engineers face when designing electronic systems for space operation. Dealing with space radiation and its effects on electronics is one of those challenges. There are many sources of radiation in space including cosmic rays which consist of protons, alpha particles, and heavy ions mixed with x-ray and gamma-ray radiation. Our atmosphere filters most of these particles so they are primarily a concern for spacecraft. Solar events are another source of radiation affecting spacecraft. The Van Allen radiation belts contain electrons and protons trapped in the geomagnetic field creating another source of radiation for spacecraft [7].

Determining exactly what kind of radiation and the dose that this project will be exposed to is beyond the scope of this report. To determine the type and level of radiation the sequencer would be exposed to the author would need to know the expected orbit of the mission, the duration of the mission, and the date of the mission at a minimum. The purpose of this discussion is to make the reader aware of the different sources of radiation for spacecraft electronics and the resulting problems caused by space radiation at an introductory level.

Lattice displacement and ionization are the two fundamental effects of radiation on electronics. Lattice displacement is the rearrangement of the atoms in the crystalline structure of the silicon component. It is caused by neutrons, protons, alpha particles, heavy ions, and very high energy gamma photons. Ionization effects are caused by charged particles. These effects are more commonly known as single event effects (SEE) and are usually transient, glitches.
However, they can lead to destructive problems if the charged particle caused a single event latchup (SEL). Table 2 lists different types of SEE and their description [8].

Table 2: List of Single Event Effects

<table>
<thead>
<tr>
<th>Effect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-event upset (SEU)</td>
<td>State change of bit in memory</td>
</tr>
<tr>
<td>Single-event latchup (SEL)</td>
<td>Short in devices with parasitic PNPN structure</td>
</tr>
<tr>
<td></td>
<td>High levels of current can cause device failure if the short is between the power rails</td>
</tr>
<tr>
<td>Single-event transient (SET)</td>
<td>Charge from ionization discharges through circuit as transient noise</td>
</tr>
<tr>
<td>Single-event induced burnout (SEB)</td>
<td>May occur in power MOSFETS</td>
</tr>
<tr>
<td></td>
<td>Substrate under source is forward biased resulting in $V_{DS}$ higher than breakdown voltage</td>
</tr>
<tr>
<td>Single-event gate rupture (SEGR)</td>
<td>Heavy ion hits gate region of device while a high voltage is applied to the gate. Looks like tiny explosion on device.</td>
</tr>
</tbody>
</table>

To address the problem of SEE in spacecraft fault-tolerant systems are used. A fault-tolerant system has the property of being able to recover from one or more faults. The requirements of a fault tolerant system are; no single point of failure, fault isolation to the failing component, fault containment, and availability of reversion modes. No single point of failure means that the system must continue to operate when it experiences a fault. No single component or code statement should be able to cause the system to fail. Fault isolation and containment allows the system to determine the offending component of the system and prevents the fault from
propagating through the rest of the system. Reversion allows the system to either revert to a previous state (roll-back), or move forward to the next state in a corrected manner.

Three different methods of implementing a fault-tolerant system are through replication, redundancy, and diversity. Replication would be the implementation of identical systems whose results are voted upon. Redundancy is the implementation of identical systems and switching to a spare in the event of a failure. Diversity is the implementation of different systems designed to perform the same task. A fault-tolerant system could be implemented in any combination of these three ideas in attempt to make it more reliable at the expense of complication. A lock-step fault-tolerant system uses replication with each instance of the system operating in parallel and in the same state at the same time. This is the type of system that was attempted for this project [5].
III. Requirements

Along with the actual sequencers being developed, a Power/Relay board was also developed by a Cal Poly senior, Billy Beecher, to facilitate power management from the launch vehicle, and contain the relays that would be switching the 28 VDC required by the NEAs. Each sequencer design was expected to interface to the Power/Relay board in order to sequence the relays housed on the Power/Relay board, and receive their power requirements. The requirements of each sequencer design, and the Power/Relay board, were developed by Beecher and the author based on information given through “word of mouth” throughout the design cycle. The requirements developed are listed in Table 3.

Table 3: List of approved requirements (Splitter Auxiliary Device SAD)

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Power/Relay Board</th>
<th>Sequencer</th>
</tr>
</thead>
</table>
| Interface    | Interfaced with Launch Vehicle, NEAs  
Primary Power 28VDC, 18A  
Secondary Power 28VDC, 18A  
+5V data signal and return (sampled at 100 Hz by launch vehicle) | Interfaces with Power/Relay board  
Supplied redundant 5VDC, 1A  
Provides 8 gate/base drive signals  
Data signal at 20 bps, return |
| Relay        | Load - 1.7 Ω NEA  
Switch 28VDC, 10A  
Tested at 28VDC, 18A for 250 ms | NA                                                                         |
| Capabilities |                                                                                  |                                                                             |
| Sequencer    | NA                                                                               | 8 programmable/configurable delays  
Maximum delay of 5 minutes | events                                                                 |
| Redundancy   | Redundant wherever possible                                                      | Same                                                                      |
| PCB          | Conforms to SAD PCB layout  
Bolts to SAD enclosure  
One six pin power input connector  
Eight six pin output connectors | Conforms to SAD PCB layout  
Interfaces to Power/Relay board |
IV. Design

A. Hardware

The basic concept of a sequencer for NPSCUL is simple; a system that when powered up will sequence through eight or less events and provide serial data back to the launch vehicle. Making the system a lock step fault-tolerant system is not so simple. The first decision that needed to be made was which processor would be used for the design. The Propeller chip from Parallax Inc. was chosen due to its multi-core topology, available development tools, and the author’s familiarity with the device.

The Propeller chip is not like traditional microcontrollers in two very fundamental ways. Firstly, the Propeller chip has eight cores that Parallax calls cogs. Each cog contains the same components: a Processor block, local 2KB Ram, two counter modules with PLLs, a video generator, I/O output register, and I/O direction register. The Propeller chip has two types of shared resources; 1) common, and 2) mutually exclusive. Common resources can be accessed by any number of cogs at any time. Mutually exclusive resources can also be accessed by any cog, but only one cog at a time. Common resources are the system counter and the I/O pins. All other shared resources are mutually exclusive resources and access to them is controlled by the Hub. The Hub is responsible for system integrity and contains 32KB of global RAM, 32KB of ROM, and other system registers. The Hub maintains system integrity by allowing each cog access to mutually exclusive resources one at a time in a round robin type fashion [2].

The second way that the propeller chip is unlike traditional microcontrollers is the lack of interrupts, and built in peripherals. Peripherals such as a universal-asynchronous-receiver-transmitter (UART) or analog to digital converter (ADC) are not found in the device.
Banging must be employed by a developer if any form of serial communication is needed. This is not as bad as it seems though. The Propeller’s native programming language “Spin” is object based allowing a developer to take advantage of objects that have been developed by Parallax and other users. This also allows for any pin on the device to be used by the developer for whatever purpose he or she desires. A block diagram of the Propeller chip is given in Figure 3. Figure 4 illustrates cog/Hub interaction through round-robin architecture.

![Figure 3: Block diagram of Propeller chip](image1)

![Figure 4: Conceptual diagram of cog/hub interaction](image2)
Initially the author planned on taking advantage of the Propeller’s multi-core topology to implement identical copies of the same application. Three cogs running identical code could have been synchronized, and their results voted upon. It was determined that this was a single point failure due to each cog residing in the same chip, even though each cog resides in its own silicon on that chip. Therefore, it was decided to use three Propeller chips to implement replication and have their results voted upon by external voting logic. A block diagram of the system is shown below.

![Figure 5: Sequencer block diagram](image-url)
Figure 6 is a schematic level diagram of the voltage regulation block shown at the top of Figure 5. Two Texas Instruments TPS7A80xx Low-dropout linear regulators have their outputs “OR” tied together through diodes D34 and D35, so that if one fails the other will continue to provide a regulated supply to the system. The input voltage range of the regulators is 0.5VDC greater than $V_{\text{out}}$ to 6VDC. The device is fully specified over the temperature range -40°C to +125°C. The input and output capacitors were chosen based on the recommended application circuit from the datasheet. The output voltage sense resistors ($R_{31}/R_{32}$) and ($R_{33}/R_{34}$) were sized based on the formula given in the datasheet [3].

$$V_{\text{out}} = \frac{(R_1 + R_2)}{R_2} \times 0.8$$

Using this formula the output voltage of each regulator should be 3.76VDC. With the typical forward voltage drop of diodes D34 and D35 being 0.39VDC at a junction temperature of 25°C and a forward current of 700mA, VCC will be approximately 3.37VDC.

Figure 6: Sequencer redundant voltage regulation
Figure 7 is a schematic level diagram of the Propeller A block shown in Figure 5. Starting with P0-P2 and P4-P7 these pins are used for synchronizing the three Propeller chips, and inter-processor communications between them. The explanation of how this is done will be discussed later in the software section in detail. P3 on each Propeller chip is used to shift out serial data to the launch vehicle after being voted upon by the external voting logic. The result is then passed to the launch vehicle through the Power/Relay board.

The outputs P9-P15 on each Propeller chip are also voted upon with the results being passed to the Power/Relay board for biasing the Relay drive circuits on that board. Each output has a pull down resistor to prevent any glitches during the unknown state of the pin while the device is initializing. The inputs P16-P23 on each Propeller chip allow the device to sense the state of the door switch on each P-POD. No decisions are made on the state of these inputs, they are simply indicators of the door status only. Pins P24-P27 are unused.
Pins P28 and 29 are used during startup of the Propeller chip to load the user program stored in the external EEPROM into the main RAM of the Propeller chip. After the user program is loaded, these pins can be used by the developer if he or she desires. Pins P30 and P31 are used to communicate with the Propeller when programming the device. Similar to P28 and P29, these pins can also be used by the developer after start up. The author chose not to use these pins, and to leave them dedicated to programming and startup. The interface for Propeller B and Propeller C are identical, and can be seen in the appendices.

There are two circuit topologies being used for the voting logic. The first topology consists of “AND” gates and diodes connected in an “OR” gate configuration. This topology is being used by the eight outputs from each Propeller chip for enabling the NEAs. The second topology uses diode logic only to implement the logical functions. This topology is being used by the serial data from each Propeller chip to the launch vehicle. The logical statement for each implementation is \((AB + AC + BC) = Y\). An example of the voting logic for the outputs from each Propeller chip can be seen in Figure 8.

![Figure 8: Voting logic for sequencer outputs](image-url)
Table 4 lists the input/output configuration of each pin and their function, while using Propeller A as an example. The input/output configuration and functionality of each Propeller chip is identical, but might be labeled with respect to that device. For example, P3 is listed as SEROUT_A in Table 4 with the function of providing serial data from the device to the voting logic. On all three Propeller chips P3 has this function, but might be labeled SEROUT_B, or SEROUT_C depending on which schematic sheet is being examined. The same labeling scheme applies to P0-P2. P0 and P1 are both inputs, and P2 is an output for all three devices, but might be labeled SYNC_A, SYNC_B, or SYNC_C depending on which schematic sheet is being examined. I hope this explanation prevents any future confusion.

<table>
<thead>
<tr>
<th>Label/Pin</th>
<th>Input/output</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYNC_A, SYNC_B, SYNC_C</td>
<td>Input, Input, Output</td>
<td>Synchronization of devices</td>
</tr>
<tr>
<td>P0, P1, P2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEROUT_A</td>
<td>Output</td>
<td>Serial data to voting logic</td>
</tr>
<tr>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inter_Comms</td>
<td>Output, Input, Output</td>
<td>Inter-Processor Communications for synchronization, and possibly voting of future data</td>
</tr>
<tr>
<td>P4, P5, P6, P7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUTA 0-7</td>
<td>Outputs</td>
<td>Relay drive signals to voting logic</td>
</tr>
<tr>
<td>P9-P15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOOR_SWITCH 1-8</td>
<td>Inputs</td>
<td>P-POD door switch sense</td>
</tr>
<tr>
<td>P16-P23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UA 0-3</td>
<td>Unused</td>
<td>NA</td>
</tr>
<tr>
<td>P24-P27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCL, SDA</td>
<td>Output, Bi-directional</td>
<td>External EEPROM I'C Buss</td>
</tr>
<tr>
<td>P28, P29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TX_A, RX_A</td>
<td>Output, Input</td>
<td>Programming interface</td>
</tr>
<tr>
<td>P30, P31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
B. Software

There are several tools available for developing source code for the Propeller chip. The Propeller Tool from Parallax is heavily supported and uses the native programming language of the Propeller chip called Spin. Spin is a high level object-based programming language developed by Parallax specifically for the Propeller chip. A Spin application may consist of many .spin files called objects, or just one. Every Spin application has what is called a top object which is loaded into cog 0 at startup. The top object will initialize any other objects used in the application. These objects can, and mostly are, run in another cog; however some objects are just a collection of useful methods. Methods are equivalent to a function in more familiar programming languages like C [1]. Figure 9 shows the object structure for this project.

![Figure 9: Sequencer object structure](image)

15
The source code developed for the sequencer prototype uses three objects, 1) the top object ‘Sequencer_rev0.8.spin’, 2) a child object ‘Sequencer_Telemetry_rev0.3.spin’, and 3) another child object ‘FullDuplexSerial.spin’. The object ‘FullDuplexSerial.spin’ is a standard object found in the Propeller Tool library, while the other two were developed by the author for this project. Figure 9 also illustrates that ‘FullDuplexSerial.spin’ is used twice by this application demonstrating the usefulness of object based programming. A complete listing of all the source code can be seen in the appendices along with a flowchart for the top object ‘Sequencer_rev0.8.spin’.

The object ‘Sequencer_rev0.8.spin’ contains five methods along with declarations in the CON, VAR, OBJ, and DAT sections. The CON section of the listing declares the frequency of the external crystal through the instruction _xinfreq, the system clock frequency through the instruction _clkmode, aliases preventing ‘magic numbers’, and constants providing aliases to I/O pins. The VAR section of the listing declares variables global to the object. Variables can be a long (32 bits), word (16 bits), or a byte (8 bits). The OBJ section of the listing declares aliases for objects used by this object. The DAT section of the listing is used for declaring pre-defined data used during run time. The DAT section of a listing can also be used for Propeller Assembly code. This object does not use Propeller Assembly.

The five methods of ‘Sequencer_rev0.8.spin’ are; 1) init, 2) main, 3) syncProps, 4) inter_processor_comms, and 5) majorityVote. The five methods are all public methods, instead of private, meaning they could be used by other projects if implemented for that use. Since this is the top object of the application, the declaration of public or private is trivial. The ‘init’ method initializes the I/O used by the object, both counter A and B of the cog running the object, and calls the start/init methods of the other objects used by this object.
The ‘main’ method can be broken down into six steps; 1) construct the variable ‘temp’, 2) enable output pins based on ‘temp’, 3) disable output pins after they have been enabled for 250 milliseconds, 4) update the variable ‘mSecA’, 5) update the array ‘dataBuffer’, and 6) call the method ‘syncProps’. The variable ‘temp’ is used as an indicator of what NEAs should be enabled through each iteration of the main loop. The variable ‘temp’ is constructed by a repeat loop that compares the current time stored in the variable ‘mSecA’ to the pre-defined data stored in the array ‘NEA_DELAYS’. If the comparison evaluates to true, a one is written to the bit corresponding to the pin which will enable the appropriate NEA.

If the variable ‘temp’ does not equal zero, the appropriate output pins are enabled and the hardware counter, ctrb, is enabled. Both counters, ctra and ctrb, are configured to keep track of system clock ticks. The counters do this by adding the value stored in frqa and frqb to their respective accumulators, phsa and phsb, every rising edge of the system clock frequency. In the ‘init’ method both frqa and frqb were set to one. The difference between the two counters is the way they are being used. The counter, ctra, is constantly enabled and providing a constant loop interval of 50 milliseconds, while ctrb is enabled and disabled and being used to control how long the outputs enabling the NEAs are on. Both counters are hardware counters, and continue to count while the main loop is executing, therefore the accuracy of the counters depends on the external crystal being used, and the PLL of the Propeller chip. It is worth noting again that each cog has two counters, ctra and ctrb, one of the few peripherals that the Propeller chip has.

After any output pins have been enabled/disabled the main loop enters a repeat while loop in line 150. This is how ctra is being used to provide a consistent 50 millisecond loop interval. The repeat while loop repeats until the count in phsa is greater than or equal to 4,000,000, which is 50 milliseconds with a clock frequency of 80MHz. The repeat while loop
actually uses the statement `clkfreq/20` instead of a constant to account for different system clock frequencies being used. As long as the instructions between instances of this statement take less time than 50 milliseconds, the main loop will have a constant interval of 50 milliseconds. The only other way for this not to be true is if one of the other Propeller chips reaches the `syncProps` method before 50 milliseconds has been reached. The method has been implemented this way purposely as an attempt to keep the three Propeller chips synced.

After the previously mentioned `repeat while` loop, `phsa` is cleared for the next iteration of the loop, and the `syncProps` method is called. This is the authors attempt to synchronize three Propeller chips being clocked by independent external crystals while running the same application. The `syncProps` method does five things; 1) sets an output high indicating to the other Propeller chips that this Propeller chip is ready to sync, 2) waits for only one of the other Propeller chips to do the same, 3) calls the `inter_processor_coms` method, 4) calls the `majorityVote` method, 5) clears the output from step one. In step two, only one other Propeller chip is waited on to prevent a continuous loop causing a fatal error. If one of the other Propeller chips had failed, and this method waited on both to reply, this Propeller chip would be stuck in a continuous loop.

The `inter_processor_comms` method uses both instances of the `FullDuplexSerial.spin` object to transmit the current value of the variable `mSecA` to the other two Propeller chips, and receive what they think the current value of `mSecA` is. The `majorityVote` method performs the same logical operation as the external voting logic mentioned previously. The voted upon result for `mSecA` is then written to `mSecA`. This allows a reset processor to roll forward to the current time that the other two processors are at. After the execution of these methods have
completed, execution returns to the ‘main’ method were the array ‘dataBuffer’ used by the object ‘Sequencer_Telemetry_rev0.3.spin’ is updated. Then the ‘main’ method starts over.

The object ‘Sequencer_Telemetry_rev0.3.spin’ is responsible for shifting data out to the launch vehicle in parallel with the operation of the object ‘Sequencer_rev0.8.spin’. The data being shifted out to the launch vehicle is byte sized data with a start and stop bit similar to RS-232. It is assumed that the launch vehicle will be sampling this data at a rate of 100 Hz, so a data rate of 20 bits-per-second was chosen to meet the Nyquist-Shannon criterion of sampling. At this rate two bytes of data can be transmitted each second. Currently, the only data being transmitted is the current value of ‘mSecA’, and the status of the P-POD door switches. Provisions have been made for the possibility of transmitting more information.

The object ‘Sequencer_Telemetry_rev0.3.spin’ has three methods; 1) init, 2) stop, and 3) txData. The ‘init’ method first stops the object if it is running in another cog which is a recommended practice by Parallax. The method then starts the ‘txData’ method in another cog. The ‘txData’ method has three arguments; 1) pin, 2) buffLength, and 3) dataBuffAdrs. The ‘pin’ argument is the I/O pin to be used by the Propeller chip for shifting out the data. The ‘buffLength’ argument is the length of the ‘dataBuffer’ array declared in ‘Sequencer_rev0.8.spin’, and the ‘dataBuffAdrs’ argument is the address of the first element of the ‘dataBuffer’ array. By passing the address of the array to the method, it now has access to the array through the hub.
V. Testing

To test the design of the sequencer a breadboard prototype was built, and can be seen in Figure 10. The only aspect of the project not implemented on the breadboard was the redundant voltage regulators. A CS13005X5 bench top power supply set to 3.3VDC was used to power the prototype. All oscilloscope measurements shown were captured with a Rigol DS1052E two channel oscilloscope. Light emitting diodes (LED) were connected at the outputs of the voting logic simulating an NEA being enabled. DIP switches were used to simulate the door switches on the P-PODS, and connected to the appropriate inputs.

Figure 10: Breadboard prototype of sequencer
Two scenarios were looked at for testing the prototype. The first scenario was the assumption that no faults had occurred, while the second scenario was the insertion of a fault by resetting one of the processors to see if the system would continue to operate properly. The purpose of the system is to reliably sequence through eight delays while transmitting data to the launch vehicle at a rate of 20bps. Each processor should be able to trigger an event at the specified time, with the difference between each processor’s outputs being no greater than 1% of the shortest delay which was specified to be one second. Therefore, the outputs from each processor driving a specific NEA should occur within 10 milliseconds of each other.

To test for the time difference between each processor indicating that an NEA should be enabled, a delay of five minutes was set in the array ‘NEA_DELAYS’, and the outputs of two Propeller chips were measured at a time with a total of three measurements taken; 1) A and B, 2) A and C, 3) B and C. The o-scope was configured to trigger from a rising edge on channel 1 with each channel set to 2V/div, and a time base of 5 microseconds. This test was performed several times in order to obtain an adequate sample. A difference of no greater than 20 microseconds was measured between each output. The results of this test can be seen in Figure 11.

Along with the sequenced outputs, the serial data being transmitted to the launch vehicle from each Propeller chip was also sampled. Samples were taken as closely as possible to the five minute marker, and the DIP switches simulating the P-POD door switches were arbitrarily set to 0xAA_{16}. The o-scope was configured to trigger from a falling edge on channel 1 with a hold-off of one second. Each channel was set to 2V/div, and a time base of 100 milliseconds. The results of this test can be seen in Figure 12.
Figure 11: Time deviation of sequenced output without a reset

Figure 12: Serial data without reset
Along with comparing the serial data and outputs of each Propeller chip to each other, the voted result of the serial data and outputs were also compared to the signals deriving them. The o-scope was configured to trigger from a rising edge on channel 1 with each channel set to 2V/div, and a time base of 50 milliseconds for comparing the voted result driving an NEA, and each output deriving it. A time base of 50 milliseconds was chosen for this test to show the duration of the output to be 250 milliseconds. The o-scope configuration for capturing the voted serial data was the same as the previous serial data test. The results of these tests can be seen in Figures 13 and 14.

**Figure 13**: Sequenced output vs. voted result without reset
The four previously mentioned tests were then re-conducted with the insertion of a fault. The inserted fault was the reset of Propeller chip A. The o-scope configuration for each test was the same as the tests conducted without a fault. The reset Propeller chip was reset after ten seconds of normal operation for each test. The results of these tests can be seen in Figures 15, 16, 17, and 18.

Figure 14: Serial data vs. voted result without reset
Figure 15: Time deviation of sequenced output with a reset

Figure 16: Serial data with reset
Figure 17: Sequenced output vs. voted result with reset

Figure 18: Serial data vs. voted result with reset
VI. Conclusions and Recommendations

Earlier in this report the requirements of a fault-tolerant system were defined as; no single point of failure, fault isolation to the failing component, fault containment, and availability of reversion modes. The prototype developed in this project meets two of those four requirements. Firstly, through replication the requirement of no single point of failure has been met. Secondly, the availability of reversion modes has been met through the ability of a non-fatal faulted processor being able to ‘roll-forward’ and synchronize with the rest of the system. Additionally, through majority voting no single fault will cause the system to fail.

One possible solution in identifying a failing processor is to use the information obtained in the ‘majorityVote’ method of the top object ‘Sequencer_rev0.8’. Possibly by identifying which Propeller is the minority could be used as an indication of a faulted device? Methods of isolating the faulted device would require further thought and testing.

While testing the system, the average current draw of the system was approximately 150 milli-amps. This differs from the previous assumption of a forward current through diodes D34, and D35 of 700 milli-amps. Further design needs to be done in selecting a diode, for ‘OR’ tying the outputs of each voltage regulator together, with an appropriate forward voltage drop. Due to the low power consumption of the system, a possible redesign of the voltage regulation of the system is in order. One solution could be the use of zener diodes as voltage regulators.

I would also like to recommend the redesign of the voting logic for the serial data being transmitted by each Propeller chip. Instead of using diodes for the ‘AND’ gates of the voter, a two input/single output ‘AND’ gate with an open drain/collector on the output could be used to regenerate the signal. Currently, the voted result of the serial data has a peak value of 2VDC,
instead of 5VDC, due to the diode drops. The outputs of each ‘AND’ gate could be pulled high to the 5VDC supply from the Power/Relay board allowing for regeneration of the signal. Possibly, the 5VDC supply should be changed to 5.5VDC to account for the voltage drop of the ‘ORing’ diodes. Diodes should still be used instead of a three input/single output ‘OR’ gate to prevent a single point of failure. One example of such an ‘AND’ gate is the SN74AHC1G09 from Texas Instruments.

The author’s intent at the start of this project was to design, and build, a fault-tolerant sequencer for NPSCUL. Unfortunately this was not accomplished. What was accomplished was the development of a breadboard prototype that could be used to conduct further testing and development of a fault-tolerant sequencer for NPSCUL. At this point, the prototype doesn’t meet all the requirements of a fault-tolerant system, and there are still further system requirements that need to be defined. Currently the only telemetry being provided to the launch vehicle is the status of the P-POD door switches, what other telemetry should be collected; system status and health, temperature, faulted components, etc.? Answering this question and developing a method of determining, and isolating, a faulted component could be accomplished through the use of this prototype.
VII. Bibliography


A. Schematics

A.1 Propeller A
A.2 Propeller B
A.3 Propeller C
A.4 NEA Voting Logic 1-4
A.5 NEA Voting Logic 1-5
A.6 Voltage Regulation/Serial Data Voting Logic
A.7 Programming Ports
B. Schedule, Bill Of Materials

---

**Breadboard Prototype**

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Manufacturer</th>
<th>Vendor #</th>
<th>Manufacturer #</th>
<th>Description</th>
<th>Quantity</th>
<th>Unit Cost</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dig-Key</td>
<td>Parallax</td>
<td>P8X32A-D40</td>
<td>P8X32A-D40</td>
<td>IC PROPELLER 8 32BIT COGS 40-DIP</td>
<td>3</td>
<td>5.59</td>
<td>22.77</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>Microchip Technology</td>
<td>24LC256-IP-ND</td>
<td>24LC256-UP</td>
<td>IC EEPROM 256KBIT 400KHZ 8DIP</td>
<td>3</td>
<td>0.88</td>
<td>2.64</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>ECS Inc</td>
<td>XC1738-ND</td>
<td>ECS-50-16-4XEN</td>
<td>CRYSTAL 4.000 MHZ 18PF 49US</td>
<td>3</td>
<td>0.66</td>
<td>1.98</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>TE Connectivity</td>
<td>450-1551-ND</td>
<td>FSM5JH</td>
<td>SWITCH TACTILE SPST-NO 0.05A 12V</td>
<td>100</td>
<td>0.12</td>
<td>11.73</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>Parallax Inc</td>
<td>32201-ND</td>
<td>32201</td>
<td>PLUG USB-SRL FOR PROPELLER CI</td>
<td>1</td>
<td>14.99</td>
<td>14.99</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>Texas Instruments</td>
<td>296-1570-5-ND</td>
<td>SN74HC08N</td>
<td>IC QUAD 2-INPUT AND GATE 14-DIP</td>
<td>6</td>
<td>0.28</td>
<td>1.68</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>Texas Instruments</td>
<td>296-1589-5-ND</td>
<td>SN74HC32N</td>
<td>IC QUAD 2-INPUT OR GATE 14-DIP</td>
<td>3</td>
<td>0.28</td>
<td>0.83</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>Parallax Inc</td>
<td>700-52305-ND</td>
<td>700-52305</td>
<td>BREADBOARD SET 3 PROPELLER/EXP</td>
<td>3</td>
<td>19.99</td>
<td>59.97</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Miscellaneous, resistors, capacitors, etc.</td>
<td></td>
<td></td>
<td>25.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>TOTAL</strong></td>
<td></td>
<td></td>
<td>141.56</td>
</tr>
</tbody>
</table>

**PCB Prototype**

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Manufacturer</th>
<th>Vendor #</th>
<th>Manufacturer #</th>
<th>Description</th>
<th>Quantity</th>
<th>Unit Cost</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dig-Key</td>
<td>Parallax</td>
<td>P8X32A-Q44</td>
<td>P8X32A-Q44</td>
<td>IC PROPELLER 8 32BIT COGS 44LQFP</td>
<td>9</td>
<td>5.59</td>
<td>68.31</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>Microchip Technology</td>
<td>24LC256-1MS-ND</td>
<td>24LC256-1MS</td>
<td>IC EEPROM 256KBIT 400KHZ 8MSOP</td>
<td>9</td>
<td>0.86</td>
<td>7.74</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>ECS Inc</td>
<td>XC1738-ND</td>
<td>ECS-50-18-4XEN</td>
<td>CRYSTAL 4.000 MHZ 18PF 49US</td>
<td>9</td>
<td>0.66</td>
<td>5.94</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>Parallax Inc</td>
<td>32201-ND</td>
<td>32201</td>
<td>PLUG USB-SRL FOR PROPELLER CI</td>
<td>3</td>
<td>14.99</td>
<td>44.97</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>Texas Instruments</td>
<td>296-12884-1-ND</td>
<td>SN74HC08N</td>
<td>IC QUAD 2IN POS AND GATE 14-SOP</td>
<td>18</td>
<td>0.28</td>
<td>5.10</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>Texas Instruments</td>
<td>296-27630-1-ND</td>
<td>TP57AB001DRBR</td>
<td>IC REG LDO ADJ 1A LN 8SON</td>
<td>6</td>
<td>2.58</td>
<td>15.48</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>Diodes Inc</td>
<td>SBR130S3-7</td>
<td>SBR130S3-7</td>
<td>DIODE SBR 1A 30V SOD-323</td>
<td>6</td>
<td>0.40</td>
<td>2.40</td>
</tr>
<tr>
<td>Dig-Key</td>
<td>ON Semiconductor</td>
<td>NSR0140P2T5GOSCT-ND</td>
<td>NSR0140P2T5G</td>
<td>DIODE SCHOTTKY 40V 70MA SOD-923</td>
<td>90</td>
<td>0.24</td>
<td>21.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Miscellaneous, resistors, capacitors, etc.</td>
<td></td>
<td></td>
<td>25.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PCB (Approximate cost for 3 boards)</td>
<td></td>
<td></td>
<td>500.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>TOTAL</strong></td>
<td></td>
<td></td>
<td>696.41</td>
</tr>
</tbody>
</table>
C. Sequencer_rev0.8 Flowchart

Start

Initialize GPIO
Start FullDuplexSerial (1)
Start FullDuplexSerial (2)
Start Sequencer_Telemetry

Set SYNC pin

Set phase to 0

Read SYNC_B
SYNC_C

Props B and C ready?
Or 1 second has passed

Yes
Clear SYNC pin

NO

Does temp = 0?

Yes

Build "temp" var for determining which NEAs should be enabled if any

No
Enable appropriate outputs based on temp
Clear and enable counter B (phSB)

Has 250 ms passed since counter B was enabled?

Yes
Disable outputs
Clear, disable counter B

No

Read
SYNC_B
SYNC_C

Has 50 ms passed based on counter A (phSA)

Yes

Set phase to 0 and increment mSecA by 50

Is Prop B or C trying to sync

Yes

Call syncProps

No

Update "dataBuffer" for Telemetry (read door switches etc.)

From syncProps

To syncProps

FullDuplexSerial
FullDuplexSerial
Sequencer_Telemetry
From Main loop

Set SYNC pin

Read SYNC_B SYNC_C

Is Prop B, or Prop C ready?

Yes

Call Inter_processor_comms

Transmit mSecA to Prop B and Prop C

Receive mSecB and mSecC from Prop B and Prop C

Call majorityVote

mSecA = mSecA&mSecB + mSecA&mSecC + mSecB&mSecC

Clear SYNC pin

To Main Loop
D.1 Sequencer_rev0.8

{{****************************************************************************
File: Sequencer_rev0.8
Programmer: Justin Jordan
Date: 06AUG12
Updated: 08MAY13

Description: Sycnronizes 3 propeller chips running this code. Objective is
to sequence 8 PPODs with the delay between them set by NEA_DELAYS
in the DAT section.
*****************************************************************************}}

CON

'declaration specifying clock mode for low speed external crystal and clock
'pll (phased lock loop) multiplier of 16
_clkmode = xtal1 + pll16x
'declaration specifying the external crystal frequency
_xinfreq = 5_000_000

'Aliases for gpio pins. Anywhere these aliases are seen in the code they
'refer to the given gpio.
'Example
'Alias = px

'p0 and p1 are inputs from other 2 props SYNC pins. Aliases not need in code
SYNC = 2
DOUT = 3
TX1 = 4
RX1 = 5
RX2 = 6
TX2 = 7
DOOR_SW1 = 8
DOOR_SW8 = 15
NEA_0 = 16

'code constants instead of magic numbers
NUM_of_NEA = 8
MODE = 0
BAUD = 115200
BUFF_LENGTH = 2
VAR

'Global variables to this object only
long idx, temp, mSecA, mSecB, mSecC, mSecAB, mSecAC, mSecBC

'This array is accessible to 'Sequencer_Telemetry_rev0.2' through the HUB
'because its base address is passed to that object
'Byte:          0,             1
'Data: time (sec), door switches

byte dataBuffer[BUFF_LENGTH]

OBJ

'Create aliases for objects
uart1: "FullDuplexSerial"
uart2: "FullDuplexSerial"
telem: "Sequencer_Telemetry_rev0.3"

PUB init

'init gpio
outa := 0
dira := (($FF << NEA_0) | (1 << SYNC))

'Enable counter A, adds frqa to phsa for every system clock tick.
ctra := 1 << 27
frqa := 1

counter B is not enabled here because it used to keep track of how long an
'NEA has been enabled, not system clock ticks

'add 1 to phsb for every system clock tick.
frqb := 1

'Starts cogs for inter-processor comms
uart1.start(RX1, TX1, MODE, BAUD)
uart2.start(RX2, TX2, MODE, BAUD)

'Starts cog for launch vehicle telemetry
telem.init(DOUT, BUFF_LENGTH, @dataBuffer)

'Hard sync of props. Waits for all three props to come up or one second
phsa := 0
outa[SYNC] := 1
repeate while((ina[1..0] <> %11) or (phsa == clkfreq)) 'Sync props
waitcnt(clkfreq/2000 + cnt)
outa[SYNC] := 0

'initialize timing vars
mSecA := 0
mSecB := 0
mSecC := 0

'method call
main
PUB main

repeat

  'clear temp variable
  temp := 0

  'iterate through delays
  repeat idx from 0 to (NUM_of_NEA - 1)
    'test for delay and if 1 second has passed
    if((long[@NEA_DELAYS][idx] == mSecA/1000) and (mSecA//1000 == 0))
      'build temporary var for enabling outputs
      temp |= ((1 << idx) << NEA_0)

  'enable outputs if any
  if temp <> 0
    'set bits in gpio output register
    outa |= temp
    'clear counter B for start of 250ms count
    phsb := 0
    'enable counter B
    ctrb := 1 << 27

    'disable outputs after counter B has been enabled for 250ms
    if phsb => clkfreq/4
      'clear bits in gpio output register
      outa &= !($FF << NEA_0)
      'disable counter B
      ctrb := 0
      'clear counter B for good practice
      phsb := 0

    'wait for 50ms, or if at least one other prop has tried to sync
    repeat while((phsa < clkfreq/20) and (ina[1..0] == %00))

      'clear counter A, counter A is still enabled in order to count system ticks
      phsa := 0

      'accumulate milli seconds
      mSecA+= 50

      'method call
      syncProps

      'update dataBuffer for 'Sequencer_Telemetry_rev0.3' object
      if mSecA//1000 == 0
        dataBuffer[0] := mSecA/1000
        dataBuffer[1] := ina[DOOR_SW8..DOOR_SW1]

  'end main

**************************************************************************
PUB syncProps

'Set sync bit
outa[SYNC] := 1

'wait for at least one other prop
repeat while(ina[1..0] == %00)

'method call
inter_processor_comms

'method call
majorityVote

'clear sync bit
outa[SYNC] := 0

'end syncProps

****************************************************************************

PUB inter_processor_comms

repeat idx from 0 to 3

'tx seconds LSB first, seconds is a long sized var = 4 bytes
uart1.tx(byte[@mSecA] [idx])

'rx seconds from prop B
byte[@mSecB] [idx] := uart1.rxtime(1)

'rx seconds from prop C
byte[@mSecC] [idx] := uart2.rxtime(1)

uart1.rxflush
uart2.rxflush

'end inter_processor_comms

****************************************************************************

PUB majorityVote

'Vote on correct time
mSecAB := (mSecA & mSecB)
mSecAC := (mSecA & mSecC)
mSecBC := (mSecB & mSecC)
mSecA := (mSecAB | mSecAC | mSecBC)

'end majorityVote

******************************************************************************
DAT
{{If NUM_NEA is less than 8, remove unnecessary delays from end of list. If enabling more than one NEA at the same time, store same value to respective delay. For example, if NEA0 and NEA1 are to be enabled after 100 seconds from initialization, make delay0 and delay1 100.}}

'Enter as seconds: delay0, delay1, delay2, delay3, delay4, delay5, delay6, delay7
NEA_DELAYS long 0, 10, 15, 20, 25, 30, 35, 300
D.2 Sequencer_Telemetry_rev0.3

{*********************************************************************
File: Sequencer_Telemetry_rev0.3
Programmer: Justin Jordan
Date: 12SEP12
Updated: 13MAY13

Description: Object for transmitting sequencer telemetry

*********************************************************************}

CON

VAR

  long cog, stack[100]

PUB init(pin, buffLength, dataBuffAdrs)

  stop
  result := cog := cognew(txData(pin, buffLength, dataBuffAdrs), @stack) + 1

PUB stop

  ' Stop Sequencer_Telemetry - frees a cog

  if cog
    cogstop(cog~ - 1)

PRI txData(dpin, buffLength, dataBuffAdrs) | idx, idy, data, packetTime

{*********************************************************************
Method: txData
Parameters: pin used to transmit data (dpin)
Returns: NA

Description: Shifts out data on dpin at ~20bps LSB first. dpin is high any
other time. Start of transmission is indicated by dpin
transitioning from high to low, and staying low for ~50ms
followed by the lsb of the LSB of dataBuff. Method is ment to be
run continuosly in another cog from the sequencing.
*********************************************************************}

  outa[dpin] := 1
  dira[dpin] := 1

  bytefill(dataBuffAdrs, 0, buffLength)

  packetTime := (1 + (buffLength/2))
repeat

{Synchronizes transmissions from each Prop by transmitting on intervals of packetTime which is derived from mSecA. Recall that mSecA is the voted result of what time all Props think it is}

repeat while((byte[dataBuffAdrs]//packetTime) <> 0 )

'shift out data
repeat idx from 0 to buffLength - 1
  data := byte[dataBuffAdrs] [idx]

'create start condition
outa[dpin] := 0
waitcnt(clkfreq/20 + cnt)

'rotate byte left 1, pre-aligns byte
data <= 1
repeat idy from 0 to 7
  'rotates byte right 1, then AND with one
  outa[dpin] := ((data >= 1) & 1)
  waitcnt(clkfreq/20 + cnt)

'create stop condition
outa[dpin] := 1
waitcnt(clkfreq/20 + cnt)

waitcnt(clkfreq/2 + cnt)    'forces 0.5 seconds of idle time for clarity

'end txData

******************************************************************************
D.3 FullDuplexSerial

'* Full-Duplex Serial Driver v1.2
'* Author: Chip Gracey, Jeff Martin
'* Copyright (c) 2006-2009 Parallax, Inc.
'* See end of file for terms of use.
'*

{--------------------REVISION HISTORY--------------------
 v1.2 - 5/7/2009 fixed bug in dec method causing largest negative value (-2,147,483,648) to be output as -0.
}

VAR

long  cog                     'cog flag/id
long  rx_head                 '9 contiguous longs
long  rx_tail
long  tx_head
long  tx_tail
long  rx_pin
long  tx_pin
long  rxtx_mode
long  bit_ticks
long  buffer_ptr

byte  rx_buffer[16]           'transmit and receive buffers
byte  tx_buffer[16]

PUB start(rxpin, txpin, mode, baudrate) : okay

' Start serial driver - starts a cog
' returns false if no cog available
''
' mode bit 0 = invert rx
' mode bit 1 = invert tx
' mode bit 2 = open-drain/source tx
' mode bit 3 = ignore tx echo on rx

stop
longfill(@rx_head, 0, 4)
longmove(@rx_pin, @rxpin, 3)
bit_ticks := clkfreq / baudrate
buffer_ptr := @rx_buffer
okay := cog := cognew(@entry, @rx_head) + 1

PUB stop

' Stop serial driver - frees a cog
if cog
cogstop(cog - 1)
longfill(@rx_head, 0, 9)

PUB rxflush

'' Flush receive buffer
   repeat while rxcheck => 0

PUB rxcheck : rxbyte

'' Check if byte received (never waits)
'' returns -1 if no byte received, $00..$FF if byte

   rxbyte--
   if rx_tail <> rx_head
      rxbyte := rx_buffer[rx_tail]
      rx_tail := (rx_tail + 1) & $F

PUB rxtime(ms) : rxbyte | t

'' Wait ms milliseconds for a byte to be received
'' returns -1 if no byte received, $00..$FF if byte

   t := cnt
   repeat until (rxbyte := rxcheck) => 0 or (cnt - t) / (clkfreq / 1000) > ms

PUB rx : rxbyte

'' Receive byte (may wait for byte)
'' returns $00..$FF

   repeat while (rxbyte := rxcheck) < 0

PUB tx(txbyte)

'' Send byte (may wait for room in buffer)

   repeat until (tx_tail <> (tx_head + 1) & $F)
   tx_buffer[tx_head] := txbyte
   tx_head := (tx_head + 1) & $F

   if rxtx_mode & %1000
      rx

PUB str(stringptr)

'' Send string

   repeat strsize(stringptr)
      tx(byte[stringptr++])

PUB dec(value) | i, x

' Print a decimal number

x := value == NEGX
'Check for max negative
if value < 0
    value := ||(value+x)
'If negative, make positive; adjust for max negative
    tx("-")
'and output sign

i := 1_000_000_000
'Initialize divisor

repeat 10
'Loop for 10 digits
    if value => i
        tx(value / i + "0" + x*(i == 1))
'If non-zero digit, output digit; adjust for max negative
    value //= i
'and digit from value
    result~~
'flag non-zero found
elseif result or i == 1
    tx("0")
'If zero digit (or only digit) output it
    i /= 10
'Update divisor

PUB hex(value, digits)

' Print a hexadecimal number

value <<= (8 - digits) << 2
repeat digits
    tx(lookupz((value <= 4) & $F : "0".."9", "A".."F"))

PUB bin(value, digits)

' Print a binary number

value <<= 32 - digits
repeat digits
    tx((value <= 1) & 1 + "0")

DAT

'***********************************
'* Assembly language serial driver *
'***********************************

org
Entry                   mov    t1,par        'get structure address
add   t1,#4 << 2    'skip past heads and tails
rdlong t2,t1        'get rx_pin
mov   rxmask,#1
shl   rxmask,t2
add   t1,#4        'get tx_pin
rdlong t2,t1
mov   txmask,#1
shl   txmask,t2
add   t1,#4        'get rxtx_mode
rdlong rxtxmode,t1
add   t1,#4        'get bit_ticks
rdlong bitticks,t1
add   t1,#4        'get buffer_ptr
rdlong rxbuff,t1
mov   txbuff,rxbuff
add   txbuff,#16

add   t1,#4        'get rxtx_mode
rdlong rxbuff,t1
mov   txbuff,rxbuff
add   txbuff,#16

test  rxtxmode,#%100 wz    'init tx pin according to
test  rxtxmode,#%010 wc
  if_z_ne_c       or      outa,txmask
  if_z            or      dira,txmask

  mov   txcode,#transmit      'initialize ping-pong

mode
  test  rxtxmode,#%001 wz    'wait for start bit on rx
pin
  test  rxmask,ina      wc
  if_z_eq_c jmp     #receive

  mov   rxbits,#9        'ready to receive byte
  mov   rxcnt,bitticks
  shr   rxcnt,#1
  add   rxcnt,cnt

/bit
  add   rxcnt,bitticks       'ready next bit period

:wait
  jmpret  rxcode,txcode     'run a chuck of transmit

Receive                 jmpret  rxc ode,txcode    'run a chunk of transmit
code, then return

:test  rxtxmode,#%001 wz    'wait for start bit on rx
pin
  test  rxmask,ina      wc
  if_z_eq_c jmp     #receive

  mov   rxbits,#9        'ready to receive byte
  mov   rxcnt,bitticks
  shr   rxcnt,#1
  add   rxcnt,cnt

:bit
  add   rxcnt,bitticks       'ready next bit period

:wait
  jmpret  rxcode,txcode    'run a chuck of transmit
code, then return
mov     t1,rxcnt              ;'check if bit receive
sub     t1,cnt
cmps    t1,#0               ;'check if bit receive
jc      :wait

if nc:
    test    rxmask,ina        ;'receive bit on rx pin
    rcr     rxdata,#1
    djnz    rxbits,:bit
    shr     rxdata,#32-9      ;'justify and trim received
    and     rxdata,#$FF
    test    rxtxmode,#%001   ;'if rx inverted, invert
    jnz
    xor     rxdata,#$FF
rdlong  t2,par               ;'save received byte and inc
head:
    add     t2,rxbuff
    wrbyte  rxdata,t2
    sub     t2,rxbuff
    add     t2,#1
    and     t2,#$0F
    wrlong  t2,par
    jmp     #receive         ;'byte done, receive next

byte
    '
    ' Transmit
    transmit                ;'run a chunk of receive
code, then return
mov     t1,par               ;'check for head <> tail
add     t1,#2 << 2
rdlong  t2,t1
add     t1,#1 << 2
rdlong  t3,t1
cmp     t2,t3              ;'check if bit receive
jc      :wait

if z:
    jmp     #transmit

add     t3,txbuff
rdbyte  txdata,t3
sub     t3,txbuff
add     t3,#1
and     t3,#$0F
wrlong  t3,t1
or      txdata,#$100       ;'ready byte to transmit
shl     txdata,#2
or      txdata,#1
mov     txbits,#11
mov     txcnt,cnt

:bit:
    test    rxtxmode,#%100   ;'output bit on tx pin
    according to mode
test rxtxmode,#%010 wc
if_z_and_c xor txdata,#1 shr txdata,#1 wc
if_z muxc outa,txmask
if_nz muxnc dira,txmask
add txcnt,bitticks 'ready next cnt
:wait jmpret txcode,rxcode 'run a chunk of receive code, then return
mov t1,txcnt 'check if bit transmit period done
sub t1,cnt
cmps t1,#0 wc
if_nc jmp #:wait
djnz txbits,#:bit 'another bit to transmit?
byte
byte
' Uninitialized data
' t1 res 1
res 1
t3 res 1
rxtxmode res 1
bitticks res 1
rxmask res 1
rxbuff res 1
rxdata res 1
rxbits res 1
rxcnt res 1
rxcode res 1
txmask res 1
txbuff res 1
txdata res 1
txbits res 1
txcnt res 1
txcode res 1

{|

TERMS OF USE: MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy,
modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.