# LASER CURSOR

by

Michael Liman

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2011

# Table of Contents

# Table of Figures

# Acknowledgements

I'd like to thank my advisor, Tina Smilkstein, for providing the necessary components for the project and constantly pushing me to reach a final fully functional product. I'd also like to thank my friends and family for supporting me and helping me with my project along the way.

## I. Abstract

A user controls a cursor on a computer screen using a laser pointer. A camera picks up the laser pointer's position and sends the data to the computer to move the cursor in the correct position.

## II. Introduction

Presentations have come a long way from the basic speeches to the more sophisticated use of PowerPoint. Wireless presenters were invented to complement PowerPoint in aiding presenters with the ability to control their presentations at a distance. The next step to improving this technology is to allow the presenter to control the cursor of a computer from a distance, allowing more unique presentations to evolve. The use of buttons in PowerPoint was useless to the presenter if they did not have the ability to click it. Wireless presenters now a day only have the basic abilities to scroll through slides or have very poor joystick control of the cursor. Presenters also sometimes used laser pointer devices to help point at specific objects on the screen. With the Laser Cursor project, the idea is to combine the use of a laser pointer device with the control of a wireless presenter. The applications of this project is not limited to just presentations. Gaming applications today utilize accelerometers and IR and cameras to detect the movement of a controller. The limit to that, however, is the user is unaware of the exact location that the controller is aimed at. With a laser sight that pin points the user's direction, the user can have a more accurate and enjoyable experience gaming. Other applications can range from digital art to regular home use.

## III. Background

The project involves being able to sense the location of a laser and accurately send that position's x and y coordinates to a computer. The method utilized in this project uses a basic camera with a color filter to remove any unwanted lighting in the room. The video data gets processed through an FPGA board to locate the said x and y coordinates. Once processed, the coordinates are sent to a computer that interprets the coordinates as a cursor movement to a specific location. The left and right clicks are controlled wirelessly with a microcontroller device and an RF link module. More details on the exact implementation are shown below.

## IV. Requirements

The cursor must be able to move to the laser dot created by the laser pointer.

The controller must be wireless to enable flexibility to the user.

The cursor must move within seconds as the laser pointer is moved.

The system must work in most lighting conditions.

# V. Design

## 1. *Overall Design*

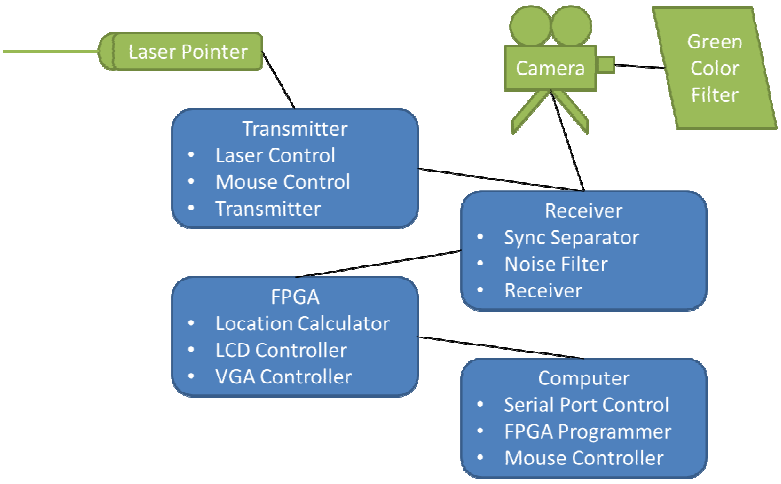


**Figure 1 – Overall Laser Cursor Design Block Diagram**

Figure 1 shows an overall prototype system design.

The transmitter circuit entails control of the laser pointer and the mouse clicks. This also includes a wireless transmitter to allow the user full flexibility to move around the area. Since the wireless transmission is using RF, there is no need to point the controller towards the screen or any other sensor. The radio frequency can be pointed in any direction and obstacles have less effect to the system compared to IR transmission.

The receiver circuit consists of a video sync separator and RF receiver. The RF receiver portion receives signals from the transmitter circuit. The video sync separator portion separates the horizontal and vertical signals from the camera. These signals are both sent to the FPGA device. Most of these components could theoretically be integrated in with the FPGA device. For a quick prototype design, the two are kept separate.

The FPGA device does the x and y position calculations. An ADC is used to read the video data while the horizontal and vertical syncs are used to keep track of the current position being analyzed. The FPGA device also has circuitry to properly send the data to the computer's serial port.

The computer has some processing involved. The program, Auto Hot Key, is used to send the mouse into specific locations along with providing the clicks when the clicks are detected. Since the ADC used is a low resolution, the computer uses a large amount of averaging to compensate for this issue. In a future design, a faster ADC will be used along with a higher resolution camera. This will improve accuracy and speed.

## 2. Controller/Transmitter Design

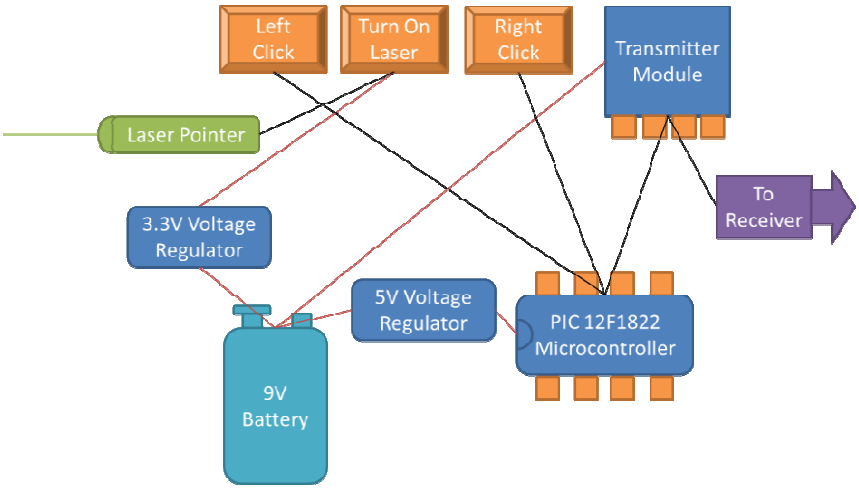


**Figure 2 – Controller/Transmitter Design Block Diagram**

Figure 2 shows the complete design block diagram for the prototype controller.

The buttons on the top indicate buttons that users can press on the controller. The buttons act as switches to activate specific functions on the controller.

The prototype design utilizes a 9V battery which will be replaced with a more powerful lithium ion battery in future designs. Theoretically, a charging circuit will be used to avoid swapping the battery.

Since the laser pointer is an off the shelf device with specific voltage requirements, a voltage regulator is used to bring the 9V from the battery down to 3.3V of the laser pointer. This is slightly higher than what is recommended on the laser pointer but was safe during testing. Future designs will involve building a laser diode driver circuit to integrate the laser into a more compact design.

A PIC12F1822 microcontroller was used in this design for its small form factor and built in capability to transmit and receive UART data. This microcontroller can run at 3.3V, though a higher voltage was used for testing purposes. In a future design, the microcontroller will share the voltage regulated supply with the laser diode circuit to cut on costs. The code for the microcontroller is located in

C code for the PIC12F1822 Transmitter and Receiver. The Pickit 3 programmer is used to program the microcontroller.

A transmitter module running at 315MHz was used in this project to simplify designing. It utilizes the voltage supply directly from the 9V battery to ensure the greatest transmission range.

### 3. *Receiver/Video Sync Design*

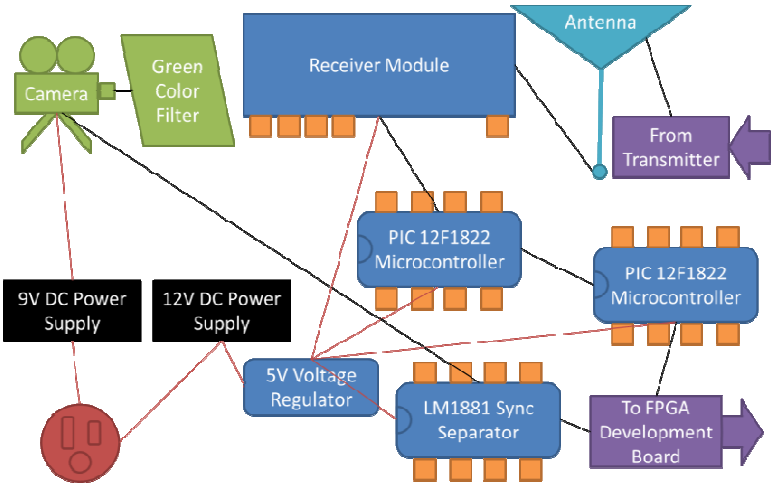


**Figure 3 – Receiver/Video Sync Design Block Diagram**

Figure 3 shows the complete design block diagram for the prototype receiver.

A green color filter is chosen for the front end of the camera to remove most of the ambient lighting in any environment. More specifically, the Rosco #90 Dark Yellow Green filter was used for its properties to pass 532nm wavelength while effectively blocking other wavelengths. This is specifically chosen because of the laser pointer wavelength used in this project. Other laser pointer colors can be used instead of the green laser pointer; however, different filter colors must be selected according to the laser pointer color selected.

A very basic surveillance camera was used for this project. The output signal provided by the camera follows conventional NTSC signals. This gives a maximum speed for the mouse movement to 60Hz. A 9V power adapter was chosen for the power supply to the camera.

The LM1881 Sync Separator chip is used to help separate the horizontal and vertical sync from the camera. A future design would integrate the chip's capability into the FPGA to cut down on manufacturing cost, chip count, and wire count.

All the integrated circuits utilize a 5V rail, so a spare 12V power supply is used. To improve efficiency, a lower value power supply would be used in the future.

Two PIC12F1822 microcontrollers are used in this design to receive the data from an RF receiver. One of the microcontrollers acts as a noise reduction to remove high frequency noise present from the receiver. This is used because the microcontroller doesn't have a built in noise filter with their built in UART transmission system. The second microcontroller utilizes a built in UART transmission system that connects to the RF receiver.

The RF receiver runs at 315MHz to match with the RF transmitter frequency. The antenna is a single wire approximately 15cm.

## 4. FPGA Design

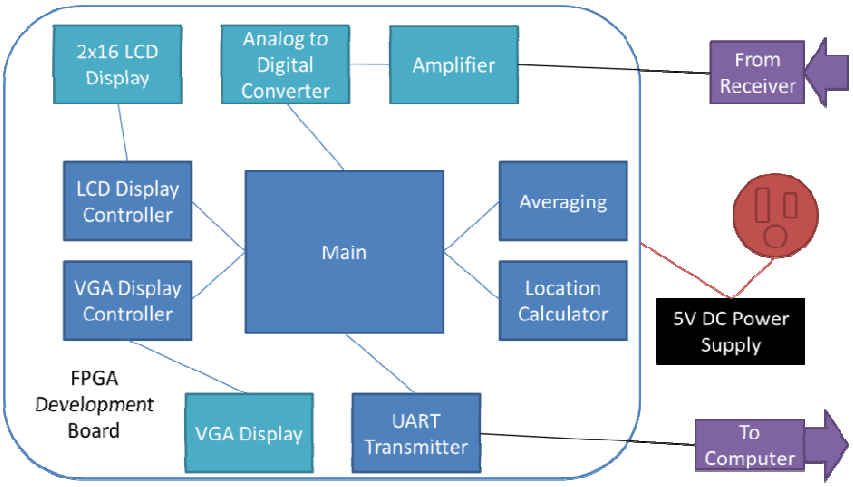


**Figure 4 – FPGA Design Block Diagram**

Figure 4 shows a full block diagram of the FPGA board used. Many of the features not used on the development board are excluded in this block diagram.

An inverting amplifier and ADC are used to digitally capture the data coming from the receiver circuit to the FPGA board. The ADC on the board is limited to 3MS/s so it cannot capture the entire resolution of the video signal. Also, the amplifier causes the digital reading to be a 2's complement. Fixing this required a simple invert on the most significant bit.

The LCD display is used as a debug tool to give accurate values for different parts of the circuit. The final prototype has this display disabled because it was not necessary to have.

The VGA display is also used as a debug tool to give a visual interface of the camera's action. The display is a 3 bit color interface due to the fact that the VGA port on the board does not

use DACs. This is used prior to utilizing the computer for debug. The final prototype has the VGA port disabled because it's no longer necessary to use.

The FPGA board is the main processor for locating and computing the laser position. There is a location calculation circuit that computes the location per frame received from the camera. Averaging is applied due to the slow speed of the ADC and low resolution of the camera.

A UART transmission block is used to convert the computed x and y coordinates along with the left and right clicks into a readable transmission through the RS232 port.

There are a few other minor blocks in the FPGA that does various processing for the video data. It is not necessary to go into detail about these basic blocks.

## 5. Computer Software Design

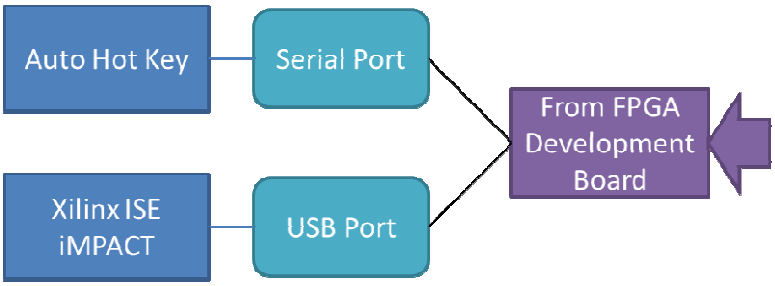


Figure 5 – Computer Software Block Diagram

Figure 5 shows a small block diagram of the computer connections and software used.

The Xilinx ISE iMPACT software is used to program the development board with the Laser Cursor bit file. Programming the board is required every time it's connected because this specific board does not have a specific memory space to hold the bit file.

Auto Hot Key is software that will allow a user to control a mouse or keyboard in unique ways. The way this software is used in this project is to read data from one of the com ports of the computer and move the cursor to specific locations on the screen based on the data received. The software is also used to apply averaging because of the limitations of the hardware used in the first prototype.

Future design will utilize a driver with a USB connection. This will also involve making the USB a plug and play device to make the user end much simpler.

# VI. Test Plans

Every component is tested in pieces to make sure every level is working. Each level is tested in a specific order to ensure a faster integration time. There may be mention of test methods in other sections, but this section covers the details of the test methods.

## 1. *ADC Test Plan*

The basic method to test the ADC is to applying a known voltage and view the digital representation of the voltage for accuracy. For a preliminary test, the 8 LEDs on the FPGA board were used to view the digital value from the ADC. A more advanced test was used with an LCD to display the binary value of the ADC. Since the ADC is a 14bit resolution, the values displayed on the LCD should range from 0 to 16383.

## 2. *LCD Display Test*

The method to test the LCD display is to send known characters to positions on the LCD screen and make sure that the characters are correct. For the first test, make sure the display fully initialize. Then write anything constantly to see if it would show up on the display. Finally write on specific points to make sure anything can be written anywhere.

## 3. *VGA Display Test*

The method to test the VGA display is to send colored squares with a known size to known coordinates on a VGA screen. Once the timing of the video data to the VGA screen is correct, send the video data from the camera to the VGA display. By moving the camera around, it should clearly show that the VGA display is working. If there are any flashes, check the timing of the video's vertical and horizontal syncs and make sure the front and back porch of the signals are long enough.

### 4. Sync Separator Test

Connect the camera video line to the LM1881 sync separator chip and use the recommended components in the LM1881 datasheet. Using an oscilloscope, look at the video signal, vertical sync signal, and horizontal sync signal to see if the syncs approximately matches with the video signal.

### 5. Camera Test

Check if the camera is powering on correctly by directly connecting the video cable into a TV that receives NTSC signals. Once that is verified, test the camera's capability to detect a laser pointer by utilizing that same TV and visually inspecting if the laser pointer is distinguishable from the environment.

### 6. Laser Pointer Test

Using an oscilloscope, connect the camera's video line. Point the camera in a dark environment at first and check to see the voltage level differences the laser pointer produces versus the environment's lighting. Also test how bright the environment can be while still being able to detect the laser pointer. Add color filters to the camera and continue the same testing.

### 7. Coordinate Test

Utilizing the VGA display, create a square or dot to symbolize a cursor and test to see how accurate the FPGA/camera can track the laser. Apply and adjust averaging to the system depending how inaccurate the cursor is.

### 8. Serial Port Data Transfer Test

Utilize the software called Hyperterminal to receive serial data. Send a constant stream of data from the FPGA and test to see if that constant data is received on Hyperterminal. Modify the COM port to match the serial port used. Also modify the baud rate to match the baud rate of the FPGA board. Once data is received in Hyperterminal, test the x, y and click data series sent by the FPGA board. Finally, utilize the Auto Hot Key program to test the data receive capabilities with the same settings as Hyperterminal.

### 9. Cursor Movement Test

While utilizing the Auto Hot Key software, test to see if the cursor movements match that of the movements seen in the VGA screen test. Apply further averaging through Auto Hot Key as necessary and test again. Apply a calibration method and test the cursor's accuracy to match the location of the laser.

### 10. Microcontroller Test

First, start with testing the microcontroller's capability to output highs and lows on each of the output pins. Use LEDs to help indicate the output's status. Place resistors in series with the LEDs to limit the voltage across the LEDs. Once the outputs are working, test the pins that will utilize the inputs by applying a high or low voltage to the pin and directly output the status to another pin. Use pull-up resistors to keep the pin high when nothing is connected. Using an oscilloscope, change the output of one of the pins frequently and measure the frequency. This will determine the operating frequency of the microcontroller.

### *11. UART Transmission Test*

Once the microcontroller is tested to work, test the UART data transfer by connecting two microcontrollers and see if the data sent is received on the other end. Use an input pin to change one of the UART's bits sent. Use an output pin to indicate that the same bit is changed as the input changes.

### *12. Wireless Transmission Test*

Taking the UART transmission test, connect the transmitter and receiver between transmit and receive of the respective pins on the microcontroller. Make sure that the signal is still being received correctly. Utilize an oscilloscope if necessary.

### *13. Noise Reduction Test*

Connect the receiver and do not transmit anything. Connect the noise reduction microcontroller and view the output. Measure the input and output of the microcontroller with an oscilloscope. Make sure the input shows a high frequency noise and the output shows a constant high. Turn on the transmitter circuit and verify that the data sent is correctly passing through the noise filter.

### *14. Buttons Test*

Make sure the metal piece used is flexible by bending it to the correct position and try to press it. Make sure the button used is conductive. Use a multimeter to test this. Make sure solder can stick onto the metal piece being used so that connections can be made.

### *15. Color Filter Test*

Start with a visual test by placing the filter between the eyes and the light source. Use the green laser to make sure it still passes through the color filter. Next, place the filter in front of the camera and test the system in a brighter environment, including using a projector. Apply multiple filters as necessary.

### *16. Polarized Lens Test*

Test a polarized lens capability to remove light from different sources. Using a visual test, place the polarized lens between the eyes and an LCD display. Rotate the polarized lens until the display is no longer visible. Point the laser pointer to the display and see if the laser point is visible through the polarized lens. Apply this to the camera and see if the laser still goes through while removing the light from the display.

### *17. Prototype Project Enclosure Receiver Test*

Once the enclosure is built, start with testing to see if the power supply and voltage regulation is working. Next, connect the remaining circuitry and look for any shorts that could cause issues. Finally connect the prototype to the remaining circuitry and test to see if it's still fully functional.

### *18. Prototype Project Enclosure Controller Test*

Once the enclosure is built, start with testing to see if the power supply and voltage regulation is working. Next, connect the remaining circuitry and look for any shorts that could cause issues. Test the buttons to see if there is a firm connection between the metal plates and the circuit. Test the laser pointer to make sure it's functional with the button. Finally connect the prototype to the remaining circuitry and test to see if it's still functional.

## VII. Development and Construction

This section covers the idea development and theory behind each component of the project.

### *1. ADC and Amplifier*

First the ADC and amplifier communication is set up. Both of these devices use an SPI interface. The bus is shared with several other components so the other components need to be shut off. The ADC is one of the most important components because the speed of the ADC determines the resolution seen in the camera. There are two ports of the ADC device; however, one of which is shut off in order to improve the speed of the other.

### *2. LCD Display*

The LCD display utilizes a semi parallel and SPI interface. It has a clock line with several data lines rather than the single line in SPI interfaces. The current design for this LCD uses a constant display refresh. This is a known issue that causes a heavier load on the clock. It does not cause any visual issues so it works as a prototype design, but on a performance standpoint, it is not efficient. Future designs will refresh the screen when there is a change needed to be made to the screen.

### 3. VGA Display

The VGA Display utilizes the horizontal and vertical sync of the camera to help with the timing with the camera. This causes known issues with the display because the timing. The camera utilizes NTSC signals while the displays use standard VGA signals. These two types of signals have slightly different timings and as such will not properly display the camera data on the screen. Some compensation is made to fix this error, but without the use of memory, this issue will not be completely solved. As a debug tool, this method is more than enough to solve issues.

### 4. Sync Separator

The LM1881 sync separator has several useful outputs including the horizontal sync, vertical sync, and even/odd. The even/odd signal is excluded in this project to simulate a faster mouse reaction. In a future design, this should be included because it will cause the cursor to jitter if ignored. The horizontal and vertical syncs of the chip are used to help determine the x and y positions of the camera.

### 5. Camera

The camera chosen for this project is a mini surveillance camera that has both an NTSC video format and audio. For this project, the audio signal is ignored. Future designs could utilize the audio to implement voice commands. The video resolution of this camera is 628 pixels by 582 pixels. Due to the ADC speed and the exclusion of the even/odd signal, the resolution is closer to 75 pixels by 241 pixels.

### 6. Laser Pointer

A basic 5mW green laser pointer is used for this project. The reason why green was chosen is because of its greater visibility compared to other colors such as red. This will also prove useful when used on an LCD screen because the absorption of the green light is less compared to a red light thus making a green laser pointer actually functional when used with an LCD screen.

### 7. Coordinate

Since a memory chip was not used to save frames from the camera, another method using threshold and averaging is used to determine the pointer location. If the threshold voltage is surpassed, the coordinates will be averaged with other points that passed the threshold voltage. In doing so, the laser cursor position will be determined as the video streams in without the use of memory.

### 8. Serial Port Data Transfer

The serial port utilizes UART protocol to transfer data. For this project, the baud rate was set to a fixed value of 115200. This baud rate must match on the computer in order to properly receive data. More detail on the UART protocol can be found in the Universal Asynchronous Receiver/Transmitter section of the Appendices.

### 9. Cursor Movement

The cursor movement is done using the program called Auto Hot Key. This program has its own unique programming language that allows a user to create logic code to control the cursor on a screen and many other functions. In this project, the Auto Hot Key software is used to read from the serial port, sort the data from the serial port, apply averaging to the coordinates received, and finally move the cursor to the correct location on the screen. The software also has calibration code written so that the screen can be matched up with the camera. A known issue that will be solved in future designs is that the COM port settings is fixed to specific values and must be changed within the code to change the settings. Currently, the code is set to COM port 3 since the serial port was defaulted to port 3.

### 10. Microcontroller

The microcontroller selected for this project is the PIC12F1822. The reason for selecting this chip is because of its small form factor and built in UART communication block. The form factor chosen for this project is an 8 pin DIP packaging. There are other surface mount packaging available, however that would require designing a PCB, which is not necessary to get the project working. The software used to program the microcontroller is MPLAB IDE. The programmer used for the microcontroller is the Pickit 3 programmer. The software requires the use of either C or assembly language. C is chosen because it's faster to program in and easier to debug. Assembly language would have given a better performance; however, it was not required in this project.

### *11. UART Transmission*

UART transmission is used to transfer data between two microcontrollers. The reason why UART is chosen over another communication method is because the data transferred will go through a wireless transmission which has only one available frequency data line. UART can transmit using only one line so this is an ideal communication method for this situation.

### *12. Wireless Transmission*

This project utilized a 315MHz RF link. The reason why 315MHz was chosen is because it is legal to transmit data across the 315MHz band. The RF link has two components, a transmitter and a receiver. The transmitter's voltage supply ranges from 3V to 12V depending on the desired signal range. The limit to these modules is in the baud rate at which the signal is sent. If the baud rate is too fast, the rise and fall time requirements of the receiver will not be met and the signal will be lost. If the baud rate is too slow, the receiver will go into a high-Z state and output a high frequency noise causing data to be lost. The high frequency noise issue can also be seen when the transmitter is too far or the transmitter is off. This is why a noise reduction circuit is implemented.

### *13. Noise Reduction*

The noise reduction circuit is implemented because there is a high frequency signal present on the receiver line when the transmitter is either out of range or off. This causes false data to be received on the microcontroller because the built in UART receiver of the microcontroller does not implement a noise filter. Since the noise is of a high frequency, a quick fix in this project is used by placing an intermediate microcontroller to act as a filter for the high frequencies. The microcontroller does add a very slight delay to the system; however it also does get rid of the high frequency noise. In a future design, a proper high frequency filter would be implemented. Or a different receiver module will be used. Another suggestion made in the datasheet is to use decoders and encoders. The reason why this was ignored in this project is because the component count to add a decoder and encoder would mean the need to design a bulkier controller that's not as user friendly.

### *14. Buttons*

The buttons chosen for this project is simple metal strips that are flexible enough to bend while retaining its original shape.

### *15. Color Filter*

There are several filters that could be used in this project depending on the lighting conditions, the color of the laser, and the type of screen used. For lighting conditions, it is important to add darkening filters as the environment gets brighter. This is to keep the environment from tripping the threshold voltage set by the FPGA. For the type of color the laser is, the corresponding color filter should be used. If a green laser is used, a green filter is selected. The same idea goes with selecting a red laser filter or any other laser filter. If a projector is used, the brightness of the projector must be reduced with darkening filters. If an LCD screen is used, a polarized lens should be used for its unique properties in this application.

### *16. Polarized Lens*

A polarized lens has a very unique property that complements with LCD displays. LCD displays output display is polarized. So when a polarized filter is rotated in front of the screen, the display will be completely blocked at a certain rotation. When a laser pointer is pointed at the screen, the laser light will go through the filter. This unique property will allow the laser light to be detected easily amongst the bright light coming from the screen and the light absorption properties of screen.

### *17. Prototype Project Enclosure Receiver*

The receiver project enclosure is a box that contains the receiver and sync separator circuit. It is like a black box because the internal circuitry is hidden while exposing just the necessary pins. The wireless receiver is hidden from view in this project enclosure. There is a pin that directly connects to the header of the FPGA development board. There is a RCA connector for the video data coming from the camera. There is a 12V power adapter that connects directly to an AC outlet.

### *18. Prototype Project Enclosure Controller*

The controller project enclosure includes the laser pointer control and the mouse click control circuits. This includes a wireless transmitter to send the mouse click data to the receiver circuit. The laser pointer is a basic off the shelf 5mW green laser. The controller has a voltage control circuit internally that sets the voltage of the green laser pointer to a higher setting of 3.3V rather than 3V. This allows the laser to be easily detected in a very bright environment. In a future design, this light will be auto adjusted based on the environment it's in. The buttons are active ground. When the buttons are pressed, they make connections in particular areas in the circuit with ground. This causes some inputs to read ground or low, and some connections to be made to turn on the laser pointer. A 9V battery is used in the project enclosure because it's a much higher voltage than the regulators. A future design to reduce the size of the enclosure would be to use a much smaller batter such as lithium ion button cell batteries.

## VIII. Integration and Test Results

This section covers a little information on integration of the project as well as general results of some of the components.

### *1. ADC and Amplifier*

The ADC and amplifier both have 2 channels. One of the channels is disabled to speed up the sample rate to about twice the speed. The Amplifier was found to be an inverting amplifier through testing. The ADC will interpret the data into a 14-bit 2's complement digital data. When the ADC was displayed in real time to the LCD screen, it was found that there is a large noise of several hundred in decimal. This equates to an effective bit range of about 6 or 7 out of the 14 bits. For this reason, it may be possible to reduce the resolution even further for this ADC in order to get a faster sample rate.

### *2. Laser Brightness*

The green laser pointer brightness surpassed the red laser pointer with the same voltage. This is found to be true because the green wavelength is more visible to the human eye than a red wavelength. This is also why a green laser pointer is used for star pointing because even the beam is visible in dark environments. Next the voltages were varied for the green laser pointer to see if there is any regulation within the laser driver circuit. It was found that there is no regulation, so a slight increase in voltage would allow a much brighter laser. But this should not normally be done because this is going against the recommended specifications of the laser pointer.

### 3. Camera Resolution

With the ADC, the camera resolution was reduced to about 75 pixels by 241 pixels. It was also found that the vertical resolution was virtually unaffected by the poor ADC sample rate. This is because the ADC would cut off the horizontal resolution first before it would affect the vertical resolution.

### 4. Cursor resolution

Due to the limited resolution of the camera and ADC, the cursor could not move to some locations on the screen. This is because the screen resolution is much finer compared to even the camera resolution. In order to actually move to those specific location, a more fine averaging needs to be implemented or a faster ADC and greater resolution camera should be used.

### 5. Battery Life

The battery life of the 9V battery is very low due to the high power demand of the laser. Under test, the battery life was able to handle about 4 hours before the circuit stopped working. This can be improved with the use of a different battery such as a lithium ion battery.

### 6. Color Filters

One color filter was found to be not enough to reduce the ambient light in normal lighting conditions. Under tests, it was found that a combination of Rosco's #90 Dark Yellow Green, #375 Cerulean Blue, #12 Straw, and #3404 Rosco N.9 reduces most ambient light. In theory, multiple Rosco #90 Dark Yellow Green filters overlapped would be a better solution, however availability of the filter during the test prevented the use of it. This test is done for the green laser. A similar test method could be applied for the red laser or any other colored laser. In theory, all lasers can work for this project with the correct light filters.

## IX. Setup

1. Connect the power adapter of the FPGA board to the AC outlet.
2. Turn on the FPGA board.
3. Open Xilinx iMPACT
4. Select Main.bit and program bit file to the FPGA board.
5. Connect the receiver project enclosure to the J1 header of the FPGA board.
6. Place the camera upright facing the screen.
7. Connect an RCA cable to the camera and to the receiver project enclosure.
8. Connect the power adapter of the receiver project enclosure to the AC outlet.
9. Connect the power adapter of the camera into the AC outlet
10. Place the color filters in front of the camera
11. Connect the serial cable from the computer to the FPGA board
    - Note: Ground of the FPGA board must connect to ground of the computer and transmit of the FPGA board must connect to receive of the computer.
12. Check the COM port setting in Control Panel of the computer
    - Note: The AutoHotKey code is defaulted to COM3. Change the code according to the com port settings found in the Control Panel of the computer.
13. Open AutoHotKey.
14. Wait or press okay when the message box appears.
15. Check if there is some mouse movement on every corner by moving the laser pointer around each corner.

- If there is no mouse movement present on one of the corners, adjust the camera until the laser is in range.

16. Move the laser pointer to the top left of the screen and hold the position.
17. Press Start + F5.
18. Move the laser pointer to the right side of the screen and hold the position.
19. Press Start + F6.
20. Move the laser pointer to the bottom of the screen and hold the position.
21. Press Start + F7.
22. Press Start + F8.
23. Close the message box that appears.
24. Check for cursor accuracy. If it's still not accurate, readjust the camera accordingly.

## X. Improvements

This section talks about possible improvements to the design of the project though there are mentions of improvements throughout the report.

### *1. ADC*

The ADC used in this project had a maximum sampling rate of 3MS/s. This is nowhere near the required sampling rate of the camera to reach the maximum resolution. A more desired sampling rate would be 20MS/s or higher depending on the camera's resolution. If this ADC is still used, another work around would be to use two cameras and have each camera utilize the vertical resolution. One camera would be faced in the correct orientation while the other would be lying on its side. Both cameras would be able to utilize the two channels of this ADC.

### 2. Amplifier

The amplifier used in this project was built into the FPGA board. Theoretically, this amplifier could be removed because the only function it served was a voltage follower which is not required.

### 3. Sync Separator

The LM1881 sync separator served its purpose well in this project; however, a future design should integrate the chip's functions into the FPGA to save on component costs.

### 4. Camera

The camera used in this project is of a lower resolution compared to the computer. If speed and accuracy is desired, an HD camera would be used. The higher resolution would allow a more fine location calculation. The higher resolution would also allow less brightness from the laser pointer and faster cursor speeds due to the reduced need of averaging.

### 5. FPGA

The FPGA design in this prototype has many inefficient blocks. A future design would require cleaning up the code and applying calibration within the FPGA rather than utilizing the computer.

### 6. Tracking Method

Memory wasn't used to track the laser's movement over time. With the use of memory, noise can be reduced or eliminated, and environments can be used in the calculations.

### 7. Receiver

The receiver circuit can integrate the two microcontrollers by speeding up the clock speed of the microcontroller. This will allow operations to run faster so when the two components are integrated, it would be seamless.

### 8. Transmitter

The transmitter circuit has no encryption so if there are multiple pointer devices in the room, there will be interference issues. Part of this can be solved by using encoders and decoders. Another requirement to solve this is to find a better wireless transmission method to allow for faster data transfers and more encryption.

### 9. Wireless Transmission

The wireless RF links used in this project has a limited baud rate. This limited baud rate also limits the amount of data that can be transferred per second. In order to improve the overall speed of the system, a better wireless transmission scheme must be used. Improvements such as using higher frequencies can help solve this issue.

### 10. Laser

The laser currently used is bulky and not flexible to integrate with a clicker. To improve this in a future design, a laser driver circuit should be made and integrated with the transmitter circuit. This will allow for a smaller and lighter clicker while also giving more flexibility in designing the laser.

### *11. Averaging*

It is best to design a system that relies less on averaging to solve some issues. Currently there is a heavy amount of averaging done in both software and hardware. To improve performance, the averaging done in software on the computer should be moved entirely to hardware, or the FPGA device. This will allow for more seamless computer operations rather than straining the computer with simple tasks.

### *12. USB Driver*

Currently, the project requires either a serial port on the computer or a USB to serial port adapter. This also requires that special software is used to run this project. A future design to improve user experience would be to move all the calibration and averaging done on the software into the hardware. Another improvement to user friendliness is to use a USB connection instead of a serial port connection. This will improve the project to a more universal environment. Finally, plug and play should be implemented so that there is no special software to run or install on the computer.

### *13. Buttons*

The buttons currently implemented are the left and right clicks of a mouse and the laser pointer. To further improve this project for applications like power point, buttons like backwards and forwards should be implemented. For applications like basic web surfing, a scroll mouse or zoom function should be added. For entertainment applications, movie and music controls such as play or stop should be implemented. For gaming applications, integrating the laser pointer in some kind of controller device would increase the gaming experience.

### *14. Color Filter*

Currently, the color filters must be swapped based on the environment. A more automated method such as using 2 polarized filters would improve the setup experience for the user. As the environment gets brighter or darker, one of the polarized filters will rotate to dim or brighten what the camera sees.

## XI. Conclusion

This project has shown that it is capable to control a cursor on a screen accurately. The goal set from the start of the project has been made. The main applications to this project are presentations, gaming, and entertainment and have been proven to work. There are some improvements that could be made to reduce component count and make the control more accurately. Those will be left for a future design. A quick video of the project is available on the following link: http://www.youtube.com/watch?v=amig-_94Erw.

## XII. Bibliography

"Aardvark I2C/SPI Host Adapter Data Sheet V5.13 - Total Phase." *Total Phase - USB, I2C, SPI, EEPROM, Flash, and CAN Embedded Systems Development Tools for Windows, Linux, and Mac OS X*. Web. <http://www.totalphase.com/docs/aardvark_datasheet/sect002/>.

Ascii Table - ASCII Character Codes and Html, Octal, Hex and Decimal Chart Conversion. Web. <http://www.asciitable.com/>.

*COLOR TELEVISION, NTSC Tutorials*. Web. <http://ntsc-tv.com/index.html>.

Liman, Michael. "YouTube - Cal Poly Senior Project - Laser Cursor 2011." YouTube - Broadcast Yourself. Web. <http://www.youtube.com/watch?v=amig-_94Erw>.

"LM1881 - Video Sync Separator." National Semiconductor | High-performance Analog. Web. <http://www.national.com/mpf/LM/LM1881.html>.

"Look RS232 - RS 232 (serial Port) Programming." *Look RS232 - RS 232 (serial Port) Monitor*. Web. <http://www.lookrs232.com/rs232/waveforms.htm>.

"PIC12F1822." *Microchip Technology Inc.* Web. <http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en544839>.

"Rosco US : Film/Video : Strobist Kit." *Rosco International*. Web. <http://www.rosco.com/us/video/strobist.cfm>.

"Sam's Laser FAQ - SS Laser Testing, Adjustment, Repair." *Misty's Realm*. Web. <http://members.misty.com/don/laserstr.htm>.

"Video Basics - Maxim." *Analog, Linear, and Mixed-signal Devices from Maxim/Dallas Semiconductor*. Web. <http://www.maxim-ic.com/app-notes/index.mvp/id/734>.

# Appendices

## A. Bill of Materials

### *1. Transmitter*

| | | | |
|---|---|---|---|
| • | 1x | 9V Battery | $2.00 |
| • | 1x | 9V Battery Holder | $0.15 |
| • | 1x | Proto Board | $0.75 |
| • | 1x | Project Enclosure | $2.00 |
| • | 1x | LM7805 5V Regulator | $0.29 |
| • | 1x | Switch | $3.49 |
| • | 3x | Buttons | $0.60 |
| • | 1x | 6 Pin Header | $0.20 |
| • | 1x | LM317 Linear Adjustable Regulator | $0.35 |
| • | 1x | PIC12F1822 PIC Microcontroller | $1.10 |
| • | 1x | 315MHz RF Transmitter | $2.40 |
| • | 3x | 1kΩ Resistors | $0.05 |
| • | 1x | 220Ω Resistor | $0.05 |
| • | 1x | 330Ω Resistor | $0.05 |
| • | 1x | 5mW Green Laser Pointer | $5.25 |
| • | Xx | Wires | $X.XX |

### 2. Receiver

- 1x Proto Board $0.75
- 1x Project Enclosure $2.00
- 1x 12V Power Supply $10.00
- 1x LM7805 5V Regulator $0.29
- 1x 6 Pin Header $0.20
- 5x 5 Pin Header $0.20
- 2x PIC12F1822 PIC Microcontroller $1.10
- 1x LM1881 Sync Separator $2.59
- 1x 315MHz RF Receiver $2.40
- 1x 680kΩ Resistor $0.05
- 2x 100nF Capacitors $0.50
- 1x RCA Connector $0.75
- 1x NTSC Camera $11.83
- 1x Rosco Dark Yellow Green Filter $6.49
- Xx Wires $X.XX

### 3. Other

- 1x Spartan 3E Starter Board $159.00
- 1x Serial Cable $5.63
- 1x USB to Serial Adapter $10.64
- 1x Pickit 3 Programmer $31.00

## B. Information on Data Transmission

### *1. Universal Asynchronous Receiver/Transmitter*

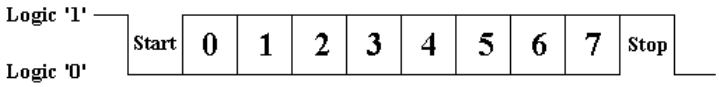


Figure 6 – UART Data Transmission General Waveform

UART allows the use of a single line for data transmission along with a ground line. Each end of the transmission must have the baud rate or frequency match. With wireless communications, UART can be used to send data through a single frequency. UART is also used to communicate between a computer through the serial port. Figure 6 shows the typical UART 8-bit waveform with a start and stop bit.

### *2. Serial Peripheral Interface*

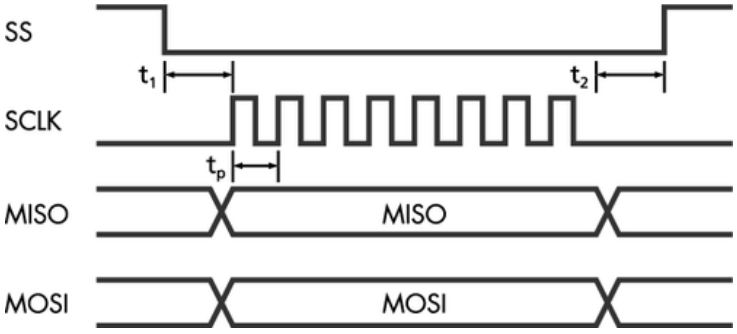


Figure 7 – SPI Data Transmission General Waveform

SPI allows for a faster transmit and receive compared to a UART transmission. This is because SPI utilizes a clock line to synchronize the two devices in communication. There is also an enable line to select certain devices to turn on and off. This allows the SPI to act as a single bus line for many different devices. Figure 7 shows the typical SPI waveform.

## C. Information on Video Data

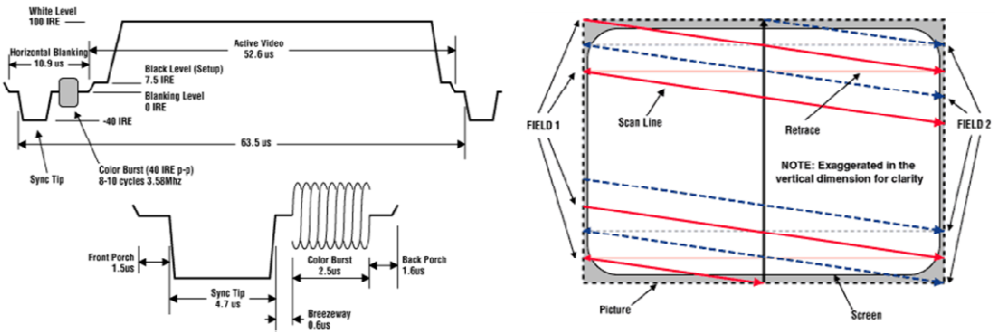


**Figure 8 – Typical Video Data Waveform (Left) Scan Lines on a Display (Right)**

Video data involves being able to have sync between the two devices such as a camera and display. The waveform in Figure 8 shows a typical video signal. There is a sync tip that indicates either the horizontal or vertical sync. There is a color burst signal that can be ignored in some applications and requires high frequency to read. The active video portion gives the exact intensity or color levels. The video signal normally scans through horizontally, so horizontal syncs are more frequent than vertical syncs. Each vertical sync signal indicates the next frame. Each horizontal sync signal indicates going back to the start of the next line.

## D. Information on the RF Link Module

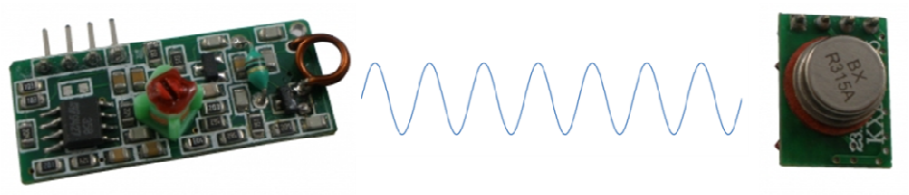


Figure 9 – 315MHz RF Link Module. Receiver (Left) Transmitter (Right)

RF link modules help simplify prototyping a wireless system. This RF link includes a transmitter module and receiver module running at 315MHz. The module design requires the use of microcontrollers to send and receive data. The limitation to these modules is that there is a specific range of baud rates that these modules run at. The receiver module requires a 5V supply and the transmitter runs up to 12V. 12V being the greatest transmission range. Figure 9 shows the actual RF modules used in this project.

## E. Information on Color Filters

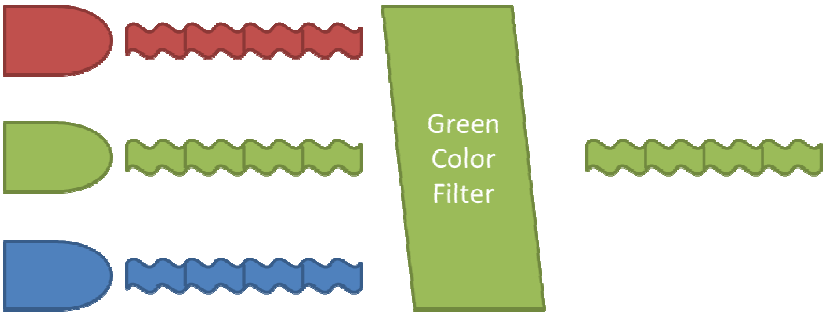


**Figure 10 – Color Filter Example with a Green Filter.**

Color filters are a great way to remove certain colors while passing other colors. The example in Figure 10 utilizes a green filter to remove the red and yellow lights. There are a variety of different filters in the market including light reduction filters and specific color filters.

## F. Information on Green Laser Pointers

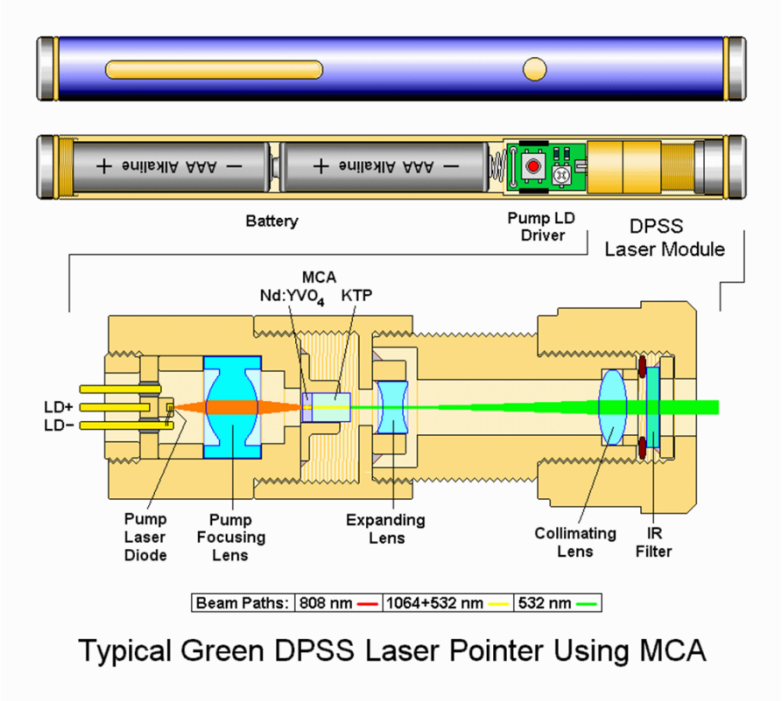


Figure 11 – Diode Pumped Solid State Green Laser Pointer

The green DPSS laser pointer outputs a green light, but not from a green light source. First, an infrared laser diode is used at 808nm. That wavelength is split to two different frequencies, 1064nm and 532nm. With an infrared filter, the 1064nm beam is removed leaving the 532nm wavelength light. 532nm equates to a green light. This method of producing a green light is inexpensive in the manufacturing process; however, it does waste a lot more energy because of the wasted infrared light being produced in the system. Figure 11 gives a full diagram on the process of outputting a green light from a DPSS laser pointer.

## G. Information on Polarized Lens

Polarized lens help passes light in only a specific orientation. This property allows the camera to remove light emitted from an LCD screen depending on the orientation of the lens since most LCD screens display in one direction. Also, with two polarized lens, brightness of the light passing through can be adjusted.

## H. Information on Microcontrollers

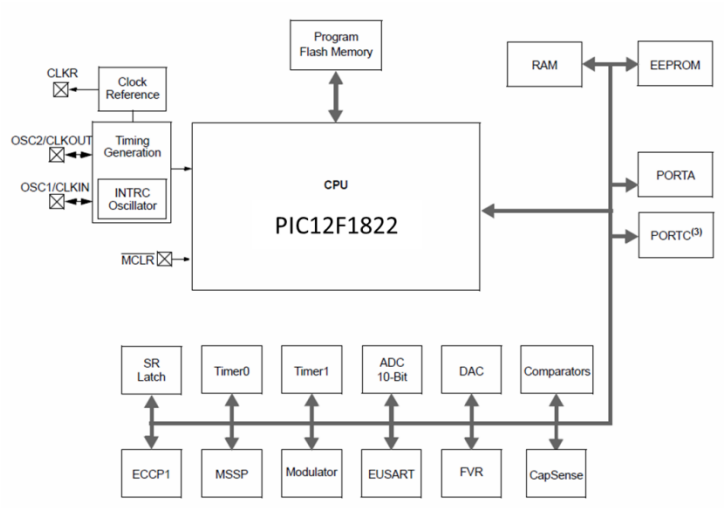


Figure 12 – PIC12F1822 Microcontroller Block Diagram

The PIC12F1822 is a very powerful small form microcontroller that has a built in UART transmit and receive block. The basic programmer for these PIC microcontrollers is to use the Pickit 3 programmer. The voltage requirement for this microcontroller is very flexible so it can be integrated into many different circuits.

## I. C code for the PIC12F1822 Transmitter and Receiver

```
#include <htc.h>

__CONFIG(FOSC_INTOSC & WDTE_OFF & PWRTE_OFF & MCLRE_OFF & CP_ON & CPD_ON &
BOREN_OFF & CLKOUTEN_OFF & IESO_OFF & FCMEN_OFF);
__CONFIG(WRT_OFF & PLLEN_OFF & STVREN_OFF & BORV_25 & LVP_OFF);

char datareceived1 = 0;
char datareceived2 = 0;
char datareceived3 = 0;
bit error;
bit invert;
bit input1;
bit input2;

interrupt receive(void)
{
	if (RCIF == 1)
	{
		datareceived3 = datareceived2;
		datareceived2 = datareceived1;
		datareceived1 = RCREG;
	}
	error = 1;
	if ((datareceived3 & 0b11011011) == 0b00000000)
	{
		if ((datareceived2 & 0b11011011) == 0b00000000)
		{
			if ((datareceived1 & 0b11011011) == 0b00000000)
			{
				error = 0;
			}
		}
	}
}

void main()
{
	/*Initialize*/
	OSCCON = 0b11110000;		//Oscillator Control
	APFCON = 0b10000000;			//Select pin location for several
components
	ANSELA = 0b00000000;		//PORTA Analog Select
	TRISA = 0b00111001;			//I/O  1-Input 0-Output
```

```
	LATA = 0b11111111;			//Set Output Value 1-High 0-Low
	BAUDCON = 0b00001000;			//Baud Rate Control
	RCSTA = 0b10010000;			//Receive Status and Control
	TXSTA = 0b00100000;			//Transmit Status and Control
	SPBRGH = 0b00011111;		//Baud Rate Generator
	SPBRGL = 0b11111111;		//2000bps
	INTCON = 0b11000000;		//Interrupt Control
	PIE1 = 0b00100000;			//Peripheral Interrupt Enable -
Enable Receiver Interrupt
	int i;
	char transmit = 0b00000000;

	while(1)
	{
		input1 = RA3 + 1;
		input2 = RA4 + 1;
		transmit = (input2 * 4) + (input1 * 32);
		while(TRMT == 0);
		for(i=0;i<14000;i++);
		TXREG = transmit;
		if (error == 0)
		{
			LATA2 = (datareceived1 & 0b00000100) / 4;
			LATA1 = (datareceived1 & 0b00100000) / 32;
		}
		else
		{
			LATA2 = 0;
			LATA1 = 0;
		}
	}
}
```

## J. C code for the PIC12F1822 Noise Reduction

```
#include <htc.h>

__CONFIG(FOSC_INTOSC & WDTE_OFF & PWRTE_OFF & MCLRE_OFF & CP_ON & CPD_ON &
BOREN_OFF & CLKOUTEN_OFF & IESO_OFF & FCMEN_OFF);
__CONFIG(WRT_OFF & PLLEN_OFF & STVREN_OFF & BORV_25 & LVP_OFF);

char datareceived1 = 0;
char datareceived2 = 0;
char datareceived3 = 0;
bit error;

void main()
{
        /*Initialize*/
        OSCCON = 0b01110000;                    //Oscillator Control
        APFCON = 0b10000000;                            //Select pin location for several
components
        ANSELA = 0b00000000;                    //PORTA Analog Select
        TRISA = 0b00111101;                             //I/O  1-Input 0-Output
        LATA = 0b11111111;                              //Set Output Value 1-High 0-Low
        int i;


        while(1)
        {
                        if (i <= 20)
                        {
                                if (RA0 == 1)
                                {
                                        i = 0;
                                }
                                else
                                {
                                        i = i + 1;
                                }
                                LATA1 = 1;
                        }
                        else
                        {
                                if (RA0 == 1)
                                {
                                        LATA1 = 0;
                                        for(i=0;i<20;i++);
```

```
                                        LATA1 = 1;
                                        i = 0;
                                }
                                else
                                {
                                        LATA1 = 0;
                                }
                        }
                }
        }
```

## K. VHDL code for the Spartain 3E Starter Board

### *1. Main Block*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Main is
   Port ( sw : in STD_LOGIC_VECTOR(3 downto 0);                    --4 Switches
                        btn_east : in STD_LOGIC;
                --East Button
                        btn_north : in STD_LOGIC;
        --North Button
                        btn_south : in STD_LOGIC;
        --South Button
                        btn_west : in STD_LOGIC;
                --West Button
                        rot_a : in STD_LOGIC;
                --Rotar A
                        rot_b : in STD_LOGIC;
                --Rotar B
                        rot_center : in STD_LOGIC;
        --Rotar Button
                        led : out STD_LOGIC_VECTOR(7 downto 0);                 --8 LEDs
                        clk_50mhz : in STD_LOGIC;
        --Internal Clock
                        lcd_e : out STD_LOGIC;
        --LCD Read/Write Enable Pulse
                        lcd_rs : out STD_LOGIC;
                --LCD Register Select
                        lcd_rw : out STD_LOGIC;
                --LCD Read/Write Control
                        sf_d : out STD_LOGIC_VECTOR(11 downto 8);   --LCD Data Line
                        spi_mosi : out STD_LOGIC;
        --Amp Set Gain
                        amp_cs : out STD_LOGIC;
                --Amp Chip Select
                        amp_shdn : out STD_LOGIC;
        --Amp Active High Shutdown
                        amp_dout : in STD_LOGIC;
                --Amp Gain Setting (Ignore)
```

```
                ad_conv : out STD_LOGIC;
        --ADC Start Conversion
                spi_clk : out STD_LOGIC;
        --ADC/Amp Clock
                spi_miso : in STD_LOGIC;
        --ADC Data
                vga_red : out STD_LOGIC;
        --VGA Red
                vga_green : out STD_LOGIC;
    --VGA Green
                vga_blue : out STD_LOGIC;
    --VGA Blue
                vga_hsync : out STD_LOGIC;
    --VGA HSync
                vga_vsync : out STD_LOGIC;
    --CGA VSync
                j1 : in STD_LOGIC_VECTOR(3 downto 0);          --Header 1
                rs232_dte_rxd : in STD_LOGIC;
                rs232_dte_txd : out STD_LOGIC;
                rs232_dce_rxd : in STD_LOGIC;
                rs232_dce_txd : out STD_LOGIC;
                spi_ss_b : out STD_LOGIC;
                dac_cs : out STD_LOGIC;
                dac_clr : out STD_LOGIC;
                sf_ce0 : out STD_LOGIC;
                sf_oe : out STD_LOGIC;
                sf_we : out STD_LOGIC;
                fpga_init_b : out STD_LOGIC
                );
end Main;

architecture Behavioral of Main is
    component debounce
        Port ( clk : in  STD_LOGIC;
                button : in  STD_LOGIC;
                pulse : out  STD_LOGIC);
    end component;

    component debouncehold
        Port ( clk : in  STD_LOGIC;
                button : in  STD_LOGIC;
                hold : out  STD_LOGIC);
    end component;

    component pulselong
```

```
        Port ( clk : in  STD_LOGIC;
                        signal_in : in  STD_LOGIC;
                        pulse : out  STD_LOGIC);
end component;

component rotaryshaft
        Port ( clk : in  STD_LOGIC;
                        rot_a : in  STD_LOGIC;
                        rot_b : in  STD_LOGIC;
                        up_down : out  STD_LOGIC;
                        count_pulse : out  STD_LOGIC);
end component;

component lcdmain
        Port ( clk : in  STD_LOGIC;
                        reset : in STD_LOGIC;
                        adc_in : in STD_LOGIC_VECTOR(13 downto 0);
                        lcd_e : out STD_LOGIC;
                        lcd_rs : out STD_LOGIC;
                        lcd_rw : out STD_LOGIC;
                        sf_d : out STD_LOGIC_VECTOR(11 downto 8);
                        h_limit : in integer range 0 to 640;
                        v_limit : in integer range 0 to 480;
                        precision : in integer range 0 to 46000;
                        calibration : in integer range 0 to 16383);
end component;

component ampsettings
        Port ( clk : in  STD_LOGIC;
                        reset : in STD_LOGIC;
                        spi_clk : out  STD_LOGIC;
                        spi_mosi : out  STD_LOGIC;
                        amp_cs : out  STD_LOGIC;
                        amp_shdn : out  STD_LOGIC;
                        amp_dout : in  STD_LOGIC;
                        amp_reading_out : out STD_LOGIC_VECTOR(7 downto 0);
                        done : out STD_LOGIC);
end component;

component adccontroller
        Port ( clk : in  STD_LOGIC;
                        reset : in STD_LOGIC;
                        ad_conv : out STD_LOGIC;
                        spi_clk : out STD_LOGIC;
                        spi_miso : in STD_LOGIC;
```

```
                                adc_reading_out_0 : out STD_LOGIC_VECTOR(13 downto
0);
                                adc_reading_out_1 : out STD_LOGIC_VECTOR(13 downto
0));
        end component;

        component vgadisplay
                Port ( clk : in  STD_LOGIC;
                                red : out  STD_LOGIC;
                                green : out  STD_LOGIC;
                                blue : out  STD_LOGIC;
                                hs : out  STD_LOGIC;
                                vs : out  STD_LOGIC;
                                adc_in : in STD_LOGIC_VECTOR(13 downto 0);
                                j1 : in STD_LOGIC_VECTOR(1 downto 0);
                                h_limit_out : out integer range 0 to 4800;
                                v_limit_out : out integer range 0 to 571;
                                btn_east : in STD_LOGIC;
                                btn_west : in STD_LOGIC;
                                up_down : in STD_LOGIC;
                                count_pulse : in STD_LOGIC;
                                x_position : in integer range 0 to 640;
                                y_position : in integer range 0 to 480);
        end component;

        component locationcalculator
                Port ( clk : in  STD_LOGIC;
                                adc_in : in  STD_LOGIC_VECTOR(13 downto 0);
                                j1 : in  STD_LOGIC_VECTOR(1 downto 0);
                                calibrate_laser : in STD_LOGIC;
                                x_position : out  integer range 0 to 92;
                                y_position : out  integer range 0 to 500;
                                precision_out : out integer range 0 to 46000;
                                calibrate_on : in STD_LOGIC;
                                calibration : out integer range 0 to 16383);
        end component;

        component averaging
                Port ( clk : in  STD_LOGIC;
                                x_input : in  integer range 0 to 640;
                                y_input : in  integer range 0 to 480;
                                x_output : out  integer range 0 to 640;
                                y_output : out  integer range 0 to 480;
                                v_sync : in STD_LOGIC);
        end component;
```

```
component transmit
        Port ( reset : in  STD_LOGIC;
                       clk : in  STD_LOGIC;
                       data : in  STD_LOGIC_VECTOR(7 downto 0);
                       enable : in  STD_LOGIC;
                       led : out STD_LOGIC_VECTOR(7 downto 0);
                       serial_out : out  STD_LOGIC);
end component;

component transmitsort
        Port ( reset : in  STD_LOGIC;
                       clk : in  STD_LOGIC;
                       x_position : in  STD_LOGIC_VECTOR (7 downto 0);
                       y_position : in  STD_LOGIC_VECTOR (7 downto 0);
                       left_click : in  STD_LOGIC;
                       right_click : in  STD_LOGIC;
                       enable : in  STD_LOGIC;
                       data_out : out  STD_LOGIC_VECTOR (7 downto 0);
                       enable_out : out  STD_LOGIC);
end component;

signal deb_btn_east : STD_LOGIC;
signal deb_btn_north : STD_LOGIC;
signal deb_btn_south : STD_LOGIC;
signal deb_btn_west : STD_LOGIC;
signal deb_v : STD_LOGIC;

signal up_down : STD_LOGIC;
signal count_pulse : STD_LOGIC;
signal led_count : STD_LOGIC_VECTOR(7 downto 0) := "00000000";

signal data : STD_LOGIC_VECTOR(7 downto 0);
signal command : STD_LOGIC;
signal write_e : STD_LOGIC;
signal reset : STD_LOGIC;
signal lcd_enable : STD_LOGIC;

signal amp_reading : STD_LOGIC_VECTOR(7 downto 0);
signal amp_done : STD_LOGIC;
signal amp_cs_int : STD_LOGIC;
signal reset_adc : STD_LOGIC;
signal adc_in : STD_LOGIC_VECTOR(13 downto 0);
signal ad_conv_int : STD_LOGIC;
signal adc_reading_0 : STD_LOGIC_VECTOR(13 downto 0);
```

```
        signal adc_reading_1 : STD_LOGIC_VECTOR(13 downto 0);
        signal spi_clk_amp : STD_LOGIC;
        signal spi_clk_adc : STD_LOGIC;
        signal spi_mosi_int : STD_LOGIC;

        signal h_limit : integer range 0 to 4800;
        signal v_limit : integer range 0 to 571;

        signal calibrate_laser : STD_LOGIC;
        signal x_position : integer range 0 to 92;
        signal x_position_calc : integer range 0 to 640;
        signal y_position : integer range 0 to 500;
        signal y_position_calc : integer range 0 to 480;
        signal final_x_position : integer range 0 to 640;
        signal final_y_position : integer range 0 to 480;
        signal final_avg_x_position : integer range 0 to 640;
        signal final_avg_y_position : integer range 0 to 480;

        signal calibration : integer range 0 to 16383;
        signal precision : integer range 0 to 46000;

        signal fixed_data : STD_LOGIC_VECTOR(7 downto 0);
        signal transfer_data : STD_LOGIC_VECTOR(7 downto 0);
        signal enable_data_transfer : STD_LOGIC;
        signal fixed_on : STD_LOGIC;
        signal rs232transfer : STD_LOGIC;

        signal x_position_transfer : STD_LOGIC_VECTOR(7 downto 0);
        signal y_position_transfer : STD_LOGIC_VECTOR(7 downto 0);
begin

--Debounce
n1: debouncehold port map (clk_50mhz,btn_east,deb_btn_east);
n2: debouncehold port map (clk_50mhz,btn_north,deb_btn_north);
n3: debouncehold port map (clk_50mhz,btn_south,deb_btn_south);
n4: debouncehold port map (clk_50mhz,btn_west,deb_btn_west);
n5: pulselong port map (clk_50mhz,not(j1(1)),deb_v);


--Rotary Shaft Up Down Counter
n10: rotaryshaft port map (clk_50mhz,rot_a,rot_b,up_down,count_pulse);

--LCD
```

```
--n20: lcdmain port map
(clk_50mhz,reset,adc_in,lcd_e,lcd_rs,lcd_rw,sf_d,final_avg_x_position,final_avg_y_position,
precision,calibration);
lcd_e <= '0';
lcd_rs <= '0';
lcd_rw <= '0';
sf_d <= "0000";
--ADC and Pre-amplifier
n30: ampsettings port map
(clk_50mhz,reset,spi_clk_amp,spi_mosi_int,amp_cs_int,amp_shdn,amp_dout,amp_reading,
amp_done);
n31: adccontroller port map
(clk_50mhz,reset_adc,ad_conv_int,spi_clk_adc,spi_miso,adc_reading_0,adc_reading_1);

--VGA
--n40: vgadisplay port map
(clk_50mhz,vga_red,vga_green,vga_blue,vga_hsync,vga_vsync,adc_in,j1(1 downto
0),h_limit,v_limit,btn_east,btn_west,up_down,count_pulse,final_avg_x_position,final_avg_
y_position);
vga_red <= '0';
vga_green <= '0';
vga_blue <= '0';
vga_hsync <= '0';
vga_vsync <= '0';
--Laser location calculator
n50: locationcalculator port map (clk_50mhz,adc_in,j1(1 downto
0),calibrate_laser,x_position,y_position,precision,sw(0),calibration);

--Averaging
n60: averaging port map
(clk_50mhz,final_x_position,final_y_position,final_avg_x_position,final_avg_y_position,j1(1)
);

--RS232 Transfer
n70: transmit port
map(reset,clk_50mhz,transfer_data,enable_data_transfer,led,rs232transfer);
n71: transmitsort port
map(reset,clk_50mhz,x_position_transfer,y_position_transfer,j1(3),j1(2),deb_v,transfer_dat
a,enable_data_transfer);

fixed_data <= "01000001";
fixed_on <= '1';

x_position_transfer <= CONV_STD_LOGIC_VECTOR(final_avg_x_position,8);
y_position_transfer <= CONV_STD_LOGIC_VECTOR(final_avg_y_position,8);
```

```
x_position_calc <= x_position * 1; --6.96
y_position_calc <= y_position * 1; --1.92

process (clk_50mhz)
begin
	if (rising_edge(clk_50mhz)) then
		if (x_position_calc = 0) then
			final_x_position <= final_x_position;
		else
			final_x_position <= x_position_calc;
		end if;
		if (y_position_calc = 0) then
			final_y_position <= final_y_position;
		else
			final_y_position <= y_position_calc;
		end if;
	end if;
end process;

calibrate_laser <= sw(0);

reset_adc <= '1' when amp_done = '0' else
                                        '0';
spi_clk <= spi_clk_amp when amp_done = '0' else
                                  spi_clk_adc;
spi_mosi <= spi_mosi_int;
reset <= deb_btn_north;

spi_ss_b <= '1';
dac_cs <= '1';
dac_clr <= '1';
sf_ce0 <= '1';
sf_oe <= '1';
sf_we <= '1';
fpga_init_b <= '0';

--Rotary Shaft Test
process (up_down,count_pulse)
begin
	if (rising_edge(count_pulse)) then
		if (up_down = '1') then
			led_count <= led_count + 1;
		else
			led_count <= led_count - 1;
```

```
                    end if;
            end if;
end process;

adc_in <= adc_reading_0;
ad_conv <= ad_conv_int;
amp_cs <= amp_cs_int;

rs232_dte_txd <= rs232transfer;
rs232_dce_txd <= rs232transfer;

end Behavioral;
```

### 2. *Debounce Circuit with an Output Hold*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity debouncehold is
   Port ( clk : in  STD_LOGIC;
          button : in  STD_LOGIC;
          hold : out  STD_LOGIC);
end debouncehold;

architecture Behavioral of debouncehold is
           signal count : STD_LOGIC_VECTOR(15 downto 0);
begin

process (clk,button,count)
begin
           if button = '0' then
                      count <= "0000000000000000";
           elsif (rising_edge(clk)) then
                      if (count /= "1111111111111111") then
                                 count <= count + 1;
                      end if;
           end if;
           if (count = "1111111111111111" and button = '1') then
                      hold <= '1';
           else
                      hold <= '0';
           end if;
end process;


end Behavioral;
```

### *3. Pulse with a long high*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity pulselong is
   Port ( clk : in  STD_LOGIC;
          signal_in : in  STD_LOGIC;
          pulse : out  STD_LOGIC);
end pulselong;

architecture Behavioral of pulselong is
            signal count : integer range 0 to 434 := 0;
begin

process (clk,signal_in,count)
begin
            if signal_in = '0' then
                        count <= 0;
            elsif (rising_edge(clk)) then
                        if (count /= 434) then
                                    count <= count + 1;
                        end if;
            end if;
            if (count /= 0 and count /= 434 and signal_in = '1') then
                        pulse <= '1';
            else
                        pulse <= '0';
            end if;
end process;

end Behavioral;
```

### *4. Debounce Circuit with an Output Pulse*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity debounce is
   Port ( clk : in  STD_LOGIC;
          button : in  STD_LOGIC;
          pulse : out  STD_LOGIC);
end debounce;

architecture Behavioral of debounce is
          signal count : STD_LOGIC_VECTOR(15 downto 0);
begin

process (clk,button,count)
begin
          if button = '0' then
                    count <= "0000000000000000";
          elsif (rising_edge(clk)) then
                    if (count /= "1111111111111111") then
                              count <= count + 1;
                    end if;
          end if;
          if (count = "1111111111111110" and button = '1') then
                    pulse <= '1';
          else
                    pulse <= '0';
          end if;
end process;

end Behavioral;
```

### 5. *Rotary Shaft Control Circuit*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity rotaryshaft is
   Port ( clk : in  STD_LOGIC;
          rot_a : in  STD_LOGIC;
          rot_b : in  STD_LOGIC;
          up_down : out  STD_LOGIC;
          count_pulse : out  STD_LOGIC);
end rotaryshaft;

architecture Behavioral of rotaryshaft is

          component debounce
                    Port ( clk : in  STD_LOGIC;
                                        button : in  STD_LOGIC;
                                        pulse : out  STD_LOGIC);
          end component;

          component debouncehold
                    Port ( clk : in  STD_LOGIC;
                                        button : in  STD_LOGIC;
                                        hold : out  STD_LOGIC);
          end component;

          type state_type is (st0,sta1,sta2,sta3,sta4,stb1,stb2,stb3,stb4);
          signal PS,NS : state_type;

          signal deb_rot_a : STD_LOGIC;
          signal deb_rot_b : STD_LOGIC;
          signal deb_rot_a_pulse : STD_LOGIC;
          signal deb_rot_b_pulse : STD_LOGIC;
          signal a_and_b_pulse : STD_LOGIC;

begin

--Debounce
n1: debouncehold port map (clk,rot_a,deb_rot_a);
n2: debouncehold port map (clk,rot_b,deb_rot_b);
n3: debounce port map (clk,rot_a,deb_rot_a_pulse);
n4: debounce port map (clk,rot_b,deb_rot_b_pulse);

--State Change Clock
```

```
sync_proc: process(clk,NS)
begin
	if (rising_edge(clk)) then
		PS <= NS;
	end if;
end process sync_proc;

--State Change Conditions
comb_proc: process(PS,deb_rot_a,deb_rot_b,clk)
begin
	case PS is
	when st0 => -- when looking for rotation direction
		if (deb_rot_a = '0') then
			NS <= sta1;
		elsif (deb_rot_b = '0') then
			NS <= stb1;
		else
			NS <= st0;
		end if;
	when sta1 => -- when rotating right
		if (deb_rot_a = '0' and deb_rot_b = '0') then
			NS <= sta2;
		elsif (deb_rot_a = '0' and deb_rot_b = '1') then
			NS <= sta1;
		else
			NS <= st0;
		end if;
		up_down <= '1';
	when sta2 =>
		if (deb_rot_a = '1' and deb_rot_b = '0') then
			NS <= sta3;
		elsif (deb_rot_a = '0' and deb_rot_b = '0') then
			NS <= sta2;
		else
			NS <= st0;
		end if;
		up_down <= '1';
	when sta3 =>
		if (deb_rot_a = '1' and deb_rot_b = '1') then
			NS <= st0;
		elsif (deb_rot_a = '1' and deb_rot_b = '0') then
			NS <= sta3;
		else
			NS <= st0;
		end if;
```

```
                up_down <= '1';
        when stb1 => -- when rotating left
                if (deb_rot_a = '0' and deb_rot_b = '0') then
                        NS <= stb2;
                elsif (deb_rot_a = '1' and deb_rot_b = '0') then
                        NS <= stb1;
                else
                        NS <= st0;
                end if;
                up_down <= '0';
        when stb2 =>
                if (deb_rot_a = '0' and deb_rot_b = '1') then
                        NS <= stb3;
                elsif (deb_rot_a = '0' and deb_rot_b = '0') then
                        NS <= stb2;
                else
                        NS <= st0;
                end if;
                up_down <= '0';
        when stb3 =>
                if (deb_rot_a = '1' and deb_rot_b = '1') then
                        NS <= st0;
                elsif (deb_rot_a = '0' and deb_rot_b = '1') then
                        NS <= stb3;
                else
                        NS <= st0;
                end if;
                up_down <= '0';
        when others => -- should never go into this
                NS <= st0;
        end case;
end process comb_proc;

--Pulse for Rotating Changes
a_and_b_pulse <= deb_rot_a_pulse or deb_rot_b_pulse;

--Increment and Decrement Indicators
process(a_and_b_pulse,PS,deb_rot_a,deb_rot_b,clk)
begin
        if rising_edge(a_and_b_pulse) then
                if (PS = sta3 and deb_rot_a = '1') or (PS = stb3 and deb_rot_b = '1') then
                        count_pulse <= '1';
                else
                        count_pulse <= '0';
                end if;
```

```
            end if;
end process;

end Behavioral;
```

### 6. *Amplifier Control Circuit*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ampsettings is
   Port ( clk : in  STD_LOGIC;
                              reset : in STD_LOGIC;
                              spi_clk : out  STD_LOGIC;
          spi_mosi : out  STD_LOGIC;
          amp_cs : out  STD_LOGIC;
          amp_shdn : out  STD_LOGIC;
          amp_dout : in  STD_LOGIC;
                              amp_reading_out : out STD_LOGIC_VECTOR(7 downto 0);
                              done : out STD_LOGIC);
end ampsettings;

architecture Behavioral of ampsettings is
          signal next_read : integer range 0 to 8 := 0;
          signal next_indicator : STD_LOGIC;
          signal next_initializer : STD_LOGIC;
          signal next_write : integer range 0 to 8 := 0;
          signal write_indicator : STD_LOGIC;
          signal read_indicator : STD_LOGIC;
          signal read_twice_indicator : STD_LOGIC;
          signal read_twice : integer range 0 to 2 := 0;
          signal amp_reading : STD_LOGIC_VECTOR(7 downto 0);

          signal clk_delay : STD_LOGIC;

          type state_type is (rd0,rd1,rd2,rd3,rd4,rd5,rd6,rd7,rd8,rd9);
          signal PS,NS : state_type;
begin

process (clk)
begin
          if (rising_edge(clk)) then
                    if (clk_delay = '0') then
                              clk_delay <= '1';
                    else
                              clk_delay <= '0';
                    end if;
          end if;
```

```
end process;

sync_proc: process(clk_delay,NS,reset)
begin
	if (reset = '1') then
		PS <= rd0;
		read_twice <= 0;
	elsif (rising_edge(clk_delay)) then
		PS <= NS;
		if (next_initializer = '1') then
			next_read <= 0;
			next_write <= 0;
		elsif (next_indicator = '1') then
			next_read <= next_read + 1;
			next_write <= next_write;
		elsif (write_indicator = '1') then
			next_read <= next_read;
			next_write <= next_write + 1;
		else
			next_read <= next_read;
			next_write <= next_write;
		end if;
		if (read_twice_indicator = '1') then
			read_twice <= read_twice + 1;
		else
			read_twice <= read_twice;
		end if;
	end if;
end process sync_proc;

comb_proc: process(clk_delay,PS,next_read,read_twice)
begin
	case PS is
	when rd0 =>
		amp_cs <= '1';
		spi_clk <= '0';
		write_indicator <= '0';
		read_indicator <= '0';
		read_twice_indicator <= '0';
		amp_shdn <= '0';
		next_initializer <= '1';
		next_indicator <= '0';
		done <= '0';
		NS <= rd1;
	when rd1 =>
```

```
            amp_cs <= '0';
            spi_clk <= '0';
            write_indicator <= '0';
            read_indicator <= '0';
            read_twice_indicator <= '0';
            amp_shdn <= '0';
            next_initializer <= '0';
            next_indicator <= '0';
            done <= '0';
            NS <= rd2;
        when rd2 =>
            amp_cs <= '0';
            spi_clk <= '0';
            write_indicator <= '0';
            read_indicator <= '0';
            read_twice_indicator <= '0';
            amp_shdn <= '0';
            next_initializer <= '0';
            next_indicator <= '0';
            done <= '0';
            NS <= rd3;
        when rd3 =>
            amp_cs <= '0';
            spi_clk <= '1';
            write_indicator <= '0';
            read_indicator <= '1';
            read_twice_indicator <= '0';
            amp_shdn <= '0';
            next_initializer <= '0';
            next_indicator <= '0';
            done <= '0';
            NS <= rd4;
        when rd4 =>
            amp_cs <= '0';
            spi_clk <= '1';
            write_indicator <= '0';
            read_indicator <= '0';
            read_twice_indicator <= '0';
            amp_shdn <= '0';
            next_initializer <= '0';
            next_indicator <= '0';
            done <= '0';
            NS <= rd5;
        when rd5 =>
            amp_cs <= '0';
```

```
		spi_clk <= '1';
		write_indicator <= '1';
		read_indicator <= '0';
		read_twice_indicator <= '0';
		amp_shdn <= '0';
		next_initializer <= '0';
		next_indicator <= '0';
		done <= '0';
		NS <= rd6;
	when rd6 =>
		amp_cs <= '0';
		spi_clk <= '0';
		write_indicator <= '0';
		read_indicator <= '0';
		read_twice_indicator <= '0';
		amp_shdn <= '0';
		next_initializer <= '0';
		next_indicator <= '0';
		done <= '0';
		NS <= rd7;
	when rd7 =>
		amp_cs <= '0';
		spi_clk <= '0';
		write_indicator <= '0';
		read_indicator <= '0';
		read_twice_indicator <= '0';
		amp_shdn <= '0';
		next_initializer <= '0';
		next_indicator <= '0';
		done <= '0';
		NS <= rd8;
	when rd8 =>
		amp_cs <= '0';
		spi_clk <= '0';
		write_indicator <= '0';
		read_indicator <= '0';
		amp_shdn <= '0';
		next_initializer <= '0';
		next_indicator <= '1';
		done <= '0';
		if (next_read = 8) then
			read_twice_indicator <= '1';
			if (read_twice = 2) then
				NS <= rd9;
			else
```

```
                        NS <= rd0;
                end if;
        else
                read_twice_indicator <= '0';
                NS <= rd3;
        end if;
    when rd9 =>
        amp_cs <= '1';
        spi_clk <= '0';
        write_indicator <= '0';
        read_indicator <= '0';
        read_twice_indicator <= '0';
        amp_shdn <= '0';
        next_initializer <= '0';
        next_indicator <= '0';
        done <= '1';
        NS <= rd9;
    when others => -- should never go into this
        NS <= rd0;
    end case;
end process;

spi_mosi <= '1' when (next_write = 4) or (next_write = 8) else
                        '0';
process (clk_delay)
begin
    if (rising_edge(clk_delay)) then
        if (read_indicator = '1') then
            if (next_write = 0) then
                amp_reading(6 downto 0) <= amp_reading(6 downto 0);
                amp_reading(7) <= amp_dout;
            elsif (next_write = 1) then
                amp_reading(5 downto 0) <= amp_reading(5 downto 0);
                amp_reading(6) <= amp_dout;
                amp_reading(7) <= amp_reading(7);
            elsif (next_write = 2) then
                amp_reading(4 downto 0) <= amp_reading(4 downto 0);
                amp_reading(5) <= amp_dout;
                amp_reading(7 downto 6) <= amp_reading(7 downto 6);
            elsif (next_write = 3) then
                amp_reading(3 downto 0) <= amp_reading(3 downto 0);
                amp_reading(4) <= amp_dout;
                amp_reading(7 downto 5) <= amp_reading(7 downto 5);
            elsif (next_write = 4) then
                amp_reading(2 downto 0) <= amp_reading(2 downto 0);
```

```
                        amp_reading(3) <= amp_dout;
                        amp_reading(7 downto 4) <= amp_reading(7 downto 4);
                    elsif (next_write = 5) then
                        amp_reading(1 downto 0) <= amp_reading(1 downto 0);
                        amp_reading(2) <= amp_dout;
                        amp_reading(7 downto 3) <= amp_reading(7 downto 3);
                    elsif (next_write = 6) then
                        amp_reading(0) <= amp_reading(0);
                        amp_reading(1) <= amp_dout;
                        amp_reading(7 downto 2) <= amp_reading(7 downto 2);
                    elsif (next_write = 7) then
                        amp_reading(0) <= amp_dout;
                        amp_reading(7 downto 1) <= amp_reading(7 downto 1);
                    else
                        amp_reading <= amp_reading;
                    end if;
                end if;
            end if;
    end process;

    amp_reading_out <= amp_reading;

    end Behavioral;
```

### 7. ADC Control Circuit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity adccontroller is
   Port ( clk : in  STD_LOGIC;
                              reset : in STD_LOGIC;
                              ad_conv : out STD_LOGIC;
                              spi_clk : out STD_LOGIC;
                              spi_miso : in STD_LOGIC;
                              adc_reading_out_0 : out STD_LOGIC_VECTOR(13 downto 0);
                              adc_reading_out_1 : out STD_LOGIC_VECTOR(13 downto 0));
end adccontroller;

architecture Behavioral of adccontroller is
        signal adc_reading_0 : STD_LOGIC_VECTOR(13 downto 0);
        signal adc_reading_1 : STD_LOGIC_VECTOR(13 downto 0);

        signal next_read : integer range 0 to 36 := 0;
        signal next_indicator : STD_LOGIC;
        signal next_initializer : STD_LOGIC;
        signal clk_on : STD_LOGIC;
        signal clk_25mhz : STD_LOGIC;

        type state_type is (rd0,rd1,rd2);
        signal PS,NS : state_type;
begin

process (clk)
begin
        if (rising_edge(clk)) then
                if clk_25mhz = '0' then
                        clk_25mhz <= '1';
                else
                        clk_25mhz <= '0';
                end if;
        end if;
end process;

sync_proc: process(clk_25mhz,NS,reset)
begin
        if (reset = '1') then
```

```
                PS <= rd0;
        elsif (rising_edge(clk_25mhz)) then
                PS <= NS;
                if (next_initializer = '1') then
                        next_read <= 0;
                elsif (next_indicator = '1') then
                        next_read <= next_read + 1;
                else
                        next_read <= next_read;
                end if;
        end if;
end process sync_proc;

comb_proc: process(clk_25mhz,PS,next_read)
begin
        case PS is
        when rd0 =>
                ad_conv <= '0';
                next_initializer <= '1';
                next_indicator <= '0';
                clk_on <= '0';
                NS <= rd1;
        when rd1 =>
                ad_conv <= '1';
                next_initializer <= '1';
                next_indicator <= '0';
                clk_on <= '0';
                NS <= rd2;
        when rd2 =>
                ad_conv <= '0';
                next_initializer <= '0';
                next_indicator <= '1';
                clk_on <= '1';
                if (next_read = 34) then
                        NS <= rd1;
                else
                        NS <= rd2;
                end if;
        when others => -- should never go into this
                NS <= rd0;
        end case;
end process;

process (clk_25mhz)
begin
```

```
if (falling_edge(clk_25mhz)) then
        if (next_read = 3) then
                adc_reading_0(13) <= not(spi_miso);
        elsif (next_read = 4) then
                adc_reading_0(12) <= spi_miso;
        elsif (next_read = 5) then
                adc_reading_0(11) <= spi_miso;
        elsif (next_read = 6) then
                adc_reading_0(10) <= spi_miso;
        elsif (next_read = 7) then
                adc_reading_0(9) <= spi_miso;
        elsif (next_read = 8) then
                adc_reading_0(8) <= spi_miso;
        elsif (next_read = 9) then
                adc_reading_0(7) <= spi_miso;
        elsif (next_read = 10) then
                adc_reading_0(6) <= spi_miso;
        elsif (next_read = 11) then
                adc_reading_0(5) <= spi_miso;
        elsif (next_read = 12) then
                adc_reading_0(4) <= spi_miso;
        elsif (next_read = 13) then
                adc_reading_0(3) <= spi_miso;
        elsif (next_read = 14) then
                adc_reading_0(2) <= spi_miso;
        elsif (next_read = 15) then
                adc_reading_0(1) <= spi_miso;
        elsif (next_read = 16) then
                adc_reading_0(0) <= spi_miso;
        elsif (next_read = 19) then
                adc_reading_1(13) <= not(spi_miso);
        elsif (next_read = 20) then
                adc_reading_1(12) <= spi_miso;
        elsif (next_read = 21) then
                adc_reading_1(11) <= spi_miso;
        elsif (next_read = 22) then
                adc_reading_1(10) <= spi_miso;
        elsif (next_read = 23) then
                adc_reading_1(9) <= spi_miso;
        elsif (next_read = 24) then
                adc_reading_1(8) <= spi_miso;
        elsif (next_read = 25) then
                adc_reading_1(7) <= spi_miso;
        elsif (next_read = 26) then
                adc_reading_1(6) <= spi_miso;
```

```
                elsif (next_read = 27) then
                        adc_reading_1(5) <= spi_miso;
                elsif (next_read = 28) then
                        adc_reading_1(4) <= spi_miso;
                elsif (next_read = 29) then
                        adc_reading_1(3) <= spi_miso;
                elsif (next_read = 30) then
                        adc_reading_1(2) <= spi_miso;
                elsif (next_read = 31) then
                        adc_reading_1(1) <= spi_miso;
                elsif (next_read = 32) then
                        adc_reading_1(0) <= spi_miso;
                end if;
        end if;
end process;

adc_reading_out_0 <= "11111111111111" - adc_reading_0;
adc_reading_out_1 <= "11111111111111" - adc_reading_1;

spi_clk <= clk_25mhz when clk_on = '1' else
                        '0';

end Behavioral;
```

### 8. Location Calculator Circuit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity locationcalculator is
   Port ( clk : in  STD_LOGIC;
        adc_in : in  STD_LOGIC_VECTOR(13 downto 0);
        j1 : in  STD_LOGIC_VECTOR(1 downto 0);
                        calibrate_laser : in STD_LOGIC;
        x_position : out  integer range 0 to 92;
        y_position : out  integer range 0 to 500;
                        precision_out : out integer range 0 to 46000;
                        calibrate_on : in STD_LOGIC;
                        calibration : out integer range 0 to 16383);
end locationcalculator;

architecture Behavioral of locationcalculator is
          component pulse
                    Port ( clk : in  STD_LOGIC;
                                    signal_in : in  STD_LOGIC;
                                    pulse : out  STD_LOGIC);
          end component;

          signal deb_h : STD_LOGIC;
          signal deb_v : STD_LOGIC;

          signal max_reading : STD_LOGIC_VECTOR(13 downto 0);
          signal max_reading_laser : STD_LOGIC_VECTOR(13 downto 0);
          signal current_x : integer range 0 to 92;
          signal current_y : integer range 0 to 500;
          signal sum_x : integer range 0 to 2140000 := 0;
          signal sum_y : integer range 0 to 11523000 := 0;

          signal sum_x_calc : integer range 0 to 729740000 := 0;
          signal sum_y_calc : integer range 0 to integer'high := 0;

          signal screen_init : STD_LOGIC;
          signal screen_count : integer range 0 to 833600 := 833600;
          signal line_init : STD_LOGIC;
          signal line_count : integer range 0 to 4800 := 3175;

          signal x_control : integer range 0 to 2; --0 is reset to 0; 1 is incrememnt; 2 is hold;
```

```
	signal y_control : integer range 0 to 2;
	signal p_control : integer range 0 to 2;
	signal sum_x_control : integer range 0 to 2;
	signal sum_y_control : integer range 0 to 2;

	signal delay_init : STD_LOGIC;
	signal delay : integer range 0 to 34 := 0;
	signal count : integer range 0 to 1;
	signal divide : integer range 0 to 1024 := 0;

	signal cal_done : STD_LOGIC;
	signal cal_display : integer range 0 to 16383 := 8000;
	signal cal_change : integer range 0 to 16383 := 8000;
	signal greater_true : STD_LOGIC;

	signal precision : integer range 0 to 46000 := 0;

	type state_type is (loc0,loc1,loc2,loc3);
	signal PS,NS : state_type;
begin

n1: pulse port map (clk,not(j1(0)),deb_h);
n2: pulse port map (clk,not(j1(1)),deb_v);

greater_true <= '1' when (adc_in > max_reading_laser) else
                                                '0';

sync_proc: process(clk,NS)
begin
	if (rising_edge(clk)) then
		PS <= NS;
		calibration <= cal_display;
		if (screen_init = '1') then
			screen_count <= 0;
		else
			screen_count <= screen_count + 1;
		end if;
		if (line_init = '1') then
			line_count <= 0;
		else
			line_count <= line_count + 1;
		end if;
		if (delay_init = '1') then
			delay <= 0;
		else
```

```
                delay <= delay + 1;
            end if;
            if (x_control = 0) then
                current_x <= 0;
            elsif (x_control = 1) then
                current_x <= current_x + 1;
            else
                current_x <= current_x;
            end if;
            if (y_control = 0) then
                current_y <= 0;
            elsif (y_control = 1) then
                current_y <= current_y + 1;
            else
                current_y <= current_y;
            end if;
            if (p_control = 0) then
                precision <= 0;
            elsif (p_control = 1) then
                precision <= precision + 1;
            else
                precision <= precision;
            end if;
            if (sum_x_control = 0) then
                sum_x <= 0;
            elsif (sum_x_control = 1) then
                sum_x <= sum_x + current_x;
            else
                sum_x <= sum_x;
            end if;
            if (sum_y_control = 0) then
                sum_y <= 0;
            elsif (sum_y_control = 1) then
                sum_y <= sum_y + current_y;
            else
                sum_y <= sum_y;
            end if;
        end if;
end process sync_proc;

max_reading_laser <= CONV_STD_LOGIC_VECTOR(cal_change,14);

comb_proc:
process(clk,PS,deb_v,deb_h,screen_count,line_count,delay,greater_true,precision)
begin
```

```
case PS is
when loc0 => --Wait for vertical sync first
        screen_init <= '1';
        line_init <= '1';
        y_control <= 0;
        x_control <= 0;
        p_control <= 2;
        if (deb_v = '1') then
                NS <= loc1;
        else
                NS <= loc0;
        end if;
when loc1 => --Start screen count.  Wait for vertical blanking to finish
        screen_init <= '0';
        line_init <= '1';
        sum_x_control <= 0;
        sum_y_control <= 0;
        y_control <= 0;
        x_control <= 0;
        p_control <= 0;
        if (screen_count < 50800) then
                NS <= loc1;
        else
                NS <= loc2;
        end if;
when loc2 => --Main state for capturing data
        screen_init <= '0';
        if (screen_count < 824075) then
                if (deb_h = '1') then
                        y_control <= 1;
                        x_control <= 0;
                        p_control <= 2;
                        line_init <= '1';
                elsif (line_count < 470) then
                        y_control <= 2;
                        x_control <= 0;
                        p_control <= 2;
                        line_init <= '0';
                        delay_init <= '1';
                elsif (line_count < 3100) then -- Collect laser position data here.
                        y_control <= 2;
                        line_init <= '0';
                        if (delay = 10) then
                                x_control <= 1;
                                delay_init <= '0';
```

```
                                if (greater_true = '1') then
                                        p_control <= 1;
                                        sum_x_control <= 1;
                                        sum_y_control <= 1;
                                else
                                        p_control <= 2;
                                        sum_x_control <= 2;
                                        sum_y_control <= 2;
                                end if;
                        elsif (delay = 34) then
                                p_control <= 2;
                                x_control <= 2;
                                delay_init <= '1';
                        else
                                p_control <= 2;
                                x_control <= 2;
                                delay_init <= '0';
                        end if;
                else
                        y_control <= 2;
                        x_control <= 2;
                        p_control <= 2;
                        line_init <= '0';
                end if;
                NS <= loc2;
        else
                NS <= loc3;
        end if;
when loc3 => --Do calculations here
        y_control <= 2;
        x_control <= 2;
        p_control <= 2;
        if (precision < 65) then
                sum_x_calc <= ((sum_x * divide) / 1024);
                sum_y_calc <= ((sum_y * divide) / 1024);
        else
                sum_x_calc <= 0;
                sum_y_calc <= 0;
        end if;
        screen_init <= '0';
        line_init <= '1';
        if (screen_count < 833000) then
                NS <= loc3;
        else
                x_position <= sum_x_calc;
```

```
                        y_position <= sum_y_calc;
                        NS <= loc0;
                end if;
        when others => -- should never go into this
                NS <= loc0;
        end case;
end process;

divide <= 1024 when precision = 1 else
                        512 when precision = 2 else
                        341 when precision = 3 else
                        256 when precision = 4 else
                        205 when precision = 5 else
                        171 when precision = 6 else
                        146 when precision = 7 else
                        128 when precision = 8 else
                        114 when precision = 9 else
                        102 when precision = 10 else
                        93 when precision = 11 else
                        85 when precision = 12 else
                        79 when precision = 13 else
                        73 when precision = 14 else
                        68 when precision = 15 else
                        64 when precision = 16 else
                        60 when precision = 17 else
                        57 when precision = 18 else
                        54 when precision = 19 else
                        51 when precision = 20 else
                        49 when precision = 21 else
                        47 when precision = 22 else
                        45 when precision = 23 else
                        43 when precision = 24 else
                        41 when precision = 25 else
                        39 when precision = 26 else
                        38 when precision = 27 else
                        37 when precision = 28 else
                        35 when precision = 29 else
                        34 when precision = 30 else
                        33 when precision = 31 else
                        32 when precision = 32 else
                        31 when precision = 33 else
                        30 when precision = 34 else
                        29 when precision = 35 else
                        28 when precision < 38 else
                        27 when precision = 38 else
```

```
                        26 when precision < 41 else
                        25 when precision = 41 else
                        24 when precision < 44 else
                        23 when precision < 46 else
                        22 when precision < 48 else
                        21 when precision < 50 else
                        20 when precision < 53 else
                        19 when precision < 56 else
                        18 when precision < 59 else
                        17 when precision < 63 else
                        16;

process (clk)
begin
        if (rising_edge(clk)) then
                if (calibrate_on = '1') then
                        cal_done <= '1';
                        if (deb_v = '1') then
                                if (precision > 64) then
                                        cal_change <= cal_change + 128;
                                        cal_display <= cal_display + 128;
                                elsif (precision > 1) then
                                        cal_change <= cal_change + 8;
                                        cal_display <= cal_display + 8;
                                elsif (precision = 0) then
                                        cal_change <= cal_change - 64;
                                        cal_display <= cal_display - 64;
                                else
                                        cal_change <= cal_change;
                                        cal_display <= cal_display;
                                end if;
                        end if;
                elsif (cal_done = '1') then
                        cal_change <= cal_change + 256;
                        cal_display <= cal_display + 256;
                        cal_done <= '0';
                end if;
        end if;
end process;

precision_out <= precision;

end Behavioral;
```

### 9. *Averaging Circuit*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity averaging is
   Port ( clk : in  STD_LOGIC;
          x_input : in  integer range 0 to 640;
          y_input : in  integer range 0 to 480;
                                x_output : out  integer range 0 to 640;
          y_output : out  integer range 0 to 480;
                                v_sync : in STD_LOGIC);
end averaging;

architecture Behavioral of averaging is
          component pulse
                    Port ( clk : in  STD_LOGIC;
                                        signal_in : in  STD_LOGIC;
                                        pulse : out  STD_LOGIC);
          end component;

          signal deb_v : STD_LOGIC;

          signal avg_x_1 : integer range 0 to 640;
          signal avg_x_2 : integer range 0 to 640;
          signal avg_x_3 : integer range 0 to 640;
          signal avg_x_4 : integer range 0 to 640;

          signal avg_y_1 : integer range 0 to 480;
          signal avg_y_2 : integer range 0 to 480;
          signal avg_y_3 : integer range 0 to 480;
          signal avg_y_4 : integer range 0 to 480;

          type state_type is (avg_0,avg_1,avg_2,avg_3);
          signal PS,NS : state_type;
begin

n1: pulse port map (clk,not(v_sync),deb_v);

sync_proc: process(clk,NS)
begin
          if (rising_edge(clk)) then
                    PS <= NS;
```

```
		end if;
end process sync_proc;

comb_proc: process(clk,PS,deb_v,x_input,y_input)
begin
		case PS is
		when avg_0 =>
			if (deb_v = '1') then
				avg_x_1 <= x_input;
				avg_y_1 <= y_input;
				NS <= avg_1;
			else
				avg_x_1 <= avg_x_1;
				avg_y_1 <= avg_y_1;
				NS <= avg_0;
			end if;
		when avg_1 =>
			if (deb_v = '1') then
				avg_x_2 <= x_input;
				avg_y_2 <= y_input;
				NS <= avg_2;
			else
				avg_x_2 <= avg_x_1;
				avg_y_2 <= avg_y_1;
				NS <= avg_1;
			end if;
		when avg_2 =>
			if (deb_v = '1') then
				avg_x_3 <= x_input;
				avg_y_3 <= y_input;
				NS <= avg_3;
			else
				avg_x_3 <= avg_x_1;
				avg_y_3 <= avg_y_1;
				NS <= avg_2;
			end if;
		when avg_3 =>
			if (deb_v = '1') then
				avg_x_4 <= x_input;
				avg_y_4 <= y_input;
				NS <= avg_0;
			else
				avg_x_4 <= avg_x_1;
				avg_y_4 <= avg_y_1;
				NS <= avg_3;
```

```
                end if;
        when others => -- should never go into this
                NS <= avg_0;
        end case;
end process;

x_output <= (avg_x_1 + avg_x_2 + avg_x_3 + avg_x_4) / 4;
y_output <= (avg_y_1 + avg_y_2 + avg_y_3 + avg_y_4) / 4;

end Behavioral;
```

### *10. Transmit through RS232 Circuit*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity transmit is
   Port ( reset : in  STD_LOGIC;
          clk : in  STD_LOGIC;
          data : in  STD_LOGIC_VECTOR(7 downto 0);
          enable : in  STD_LOGIC;
                                led : out STD_LOGIC_VECTOR(7 downto 0);
          serial_out : out  STD_LOGIC);
end transmit;

architecture Behavioral of transmit is
            signal baud_count : integer range 0 to 433;
            signal baud_clk : STD_LOGIC;
            signal data_store : STD_LOGIC_VECTOR(9 downto 0);
            signal count : integer range 0 to 10;
            signal count_control : integer range 0 to 2; --0 is reset to 0; 1 is incrememnt; 2 is
hold;

            type state_type is (s0,s1);
            signal PS,NS : state_type;
begin

process (clk)
begin
            if (reset = '1') then
                        baud_count <= 0;
            elsif (rising_edge(clk)) then
                        if (baud_count = 433) then
                                    baud_count <= 0;
                                    baud_clk <= '1';
                        else
                                    baud_count <= baud_count + 1;
                                    baud_clk <= '0';
                        end if;
            end if;
end process;

sync_proc: process(baud_clk,NS)
begin
```

```
        if (reset = '1') then
                count <= 10;
        elsif (rising_edge(baud_clk)) then
                PS <= NS;
                if (count_control = 0) then
                        count <= 0;
                elsif (count_control = 1) then
                        count <= count + 1;
                else
                        count <= count;
                end if;
        end if;
end process sync_proc;

comb_proc: process(baud_clk,PS)
begin
        case PS is
        when s0 => -- wait for an enable pulse
                if (enable = '1') then
                        data_store(9) <= '1';
                        data_store(8 downto 1) <= data(7 downto 0);
                        data_store(0) <= '0';
                        count_control <= 0;
                        NS <= s1;
                else
                        data_store <= data_store;
                        count_control <= 2;
                        NS <= s0;
                end if;
        when s1 => -- send data state
                data_store <= data_store;
                if (count = 10) then
                        count_control <= 2;
                        NS <= s0;
                else
                        count_control <= 1;
                        NS <= s1;
                end if;
        when others => -- should never go into this
                NS <= s0;
        end case;
end process;

serial_out <= data_store(0) when count = 0 else
                                data_store(1) when count = 1 else
```

```
                                    data_store(2) when count = 2 else
                                    data_store(3) when count = 3 else
                                    data_store(4) when count = 4 else
                                    data_store(5) when count = 5 else
                                    data_store(6) when count = 6 else
                                    data_store(7) when count = 7 else
                                    data_store(8) when count = 8 else
                                    data_store(9) when count = 9 else
                                    '1';

led <= "10000000" when count = 0 else
                    "01000000" when count = 1 else
                    "00100000" when count = 2 else
                    "00010000" when count = 3 else
                    "00001000" when count = 4 else
                    "00000100" when count = 5 else
                    "00000010" when count = 6 else
                    "00000001";

end Behavioral;
```

### *11. Transmit Sorting Circuit*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity transmitsort is
   Port ( reset : in  STD_LOGIC;
                              clk : in  STD_LOGIC;
          x_position : in  STD_LOGIC_VECTOR (7 downto 0);
          y_position : in  STD_LOGIC_VECTOR (7 downto 0);
          left_click : in  STD_LOGIC;
          right_click : in  STD_LOGIC;
          enable : in  STD_LOGIC;
          data_out : out  STD_LOGIC_VECTOR (7 downto 0);
          enable_out : out  STD_LOGIC);
end transmitsort;

architecture Behavioral of transmitsort is
          signal baud_count : integer range 0 to 433;
          signal baud_clk : STD_LOGIC;
          signal count : integer range 0 to 10;
          signal d_count : integer range 0 to 7;
          signal count_control : integer range 0 to 2; --0 is reset to 0; 1 is incrememnt; 2 is
hold;
          signal d_count_control : integer range 0 to 2;

          signal combination : STD_LOGIC_VECTOR(7 downto 0);

          type state_type is (s0,s1);
          signal PS,NS : state_type;
begin

process (clk)
begin
          if (reset = '1') then
                    baud_count <= 0;
          elsif (rising_edge(clk)) then
                    if (baud_count = 433) then
                              baud_count <= 0;
                              baud_clk <= '1';
                    else
                              baud_count <= baud_count + 1;
                              baud_clk <= '0';
```

```
                end if;
        end if;
end process;

sync_proc: process(baud_clk,NS)
begin
        if (reset = '1') then
                count <= 10;
                d_count <= 7;
        elsif (rising_edge(baud_clk)) then
                PS <= NS;
                if (count_control = 0) then
                        count <= 0;
                elsif (count_control = 1) then
                        count <= count + 1;
                else
                        count <= count;
                end if;
                if (d_count_control = 0) then
                        d_count <= 0;
                elsif (d_count_control = 1) then
                        d_count <= d_count + 1;
                else
                        d_count <= d_count;
                end if;
        end if;
end process sync_proc;

comb_proc: process(baud_clk,PS)
begin
        case PS is
        when s0 => -- wait for an enable pulse
                if (enable = '1') then
                        count_control <= 0;
                        d_count_control <= 0;
                        NS <= s1;
                else
                        count_control <= 2;
                        d_count_control <= 2;
                        NS <= s0;
                end if;
        when s1 =>
                if (d_count = 7) then
                        d_count_control <= 2;
                        if (count = 10) then
```

```
                        count_control <= 2;
                        NS <= s0;
                else
                        count_control <= 1;
                        NS <= s1;
                end if;
        else
                if (count = 10) then
                        count_control <= 0;
                        d_count_control <= 1;
                else
                        count_control <= 1;
                        d_count_control <= 2;
                end if;
                NS <= s1;
        end if;
    when others => -- should never go into this
        NS <= s0;
    end case;
end process;

combination(0) <= left_click;
combination(1) <= right_click;
combination(7 downto 2) <= "000000";

data_out <= "11111111" when d_count = 0 else
                        "00000000" when d_count = 1 else
                        x_position when d_count = 2 else
                        y_position when d_count = 3 else
                        combination when d_count = 4 else
                        "00000000" when d_count = 5 else
                        "00000000" when d_count = 6 else
                        "11111111";
enable_out <= '0' when d_count = 7 else
                         '1';

end Behavioral;
```

### *12. Clock Divider Circuit (Disabled)*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clockdivider is
   Port ( clk_in : in  STD_LOGIC;
          clk_out : out  STD_LOGIC);
end clockdivider;

architecture Behavioral of clockdivider is

          signal clk_count : integer range 0 to 1000;

begin

process(clk_in)
begin
          if (rising_edge(clk_in)) then
                    clk_count <= clk_count + 1;
                    if (clk_count < 500) then
                              clk_out <= '1';
                    else
                              clk_out <= '0';
                    end if;
          end if;
end process;

end Behavioral;
```

### *13. LCD Main Controller (Disabled)*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lcdmain is
   Port ( clk : in  STD_LOGIC;
                        reset : in STD_LOGIC;
                        adc_in : in STD_LOGIC_VECTOR(13 downto 0);
                        lcd_e : out STD_LOGIC;
                        lcd_rs : out STD_LOGIC;
                        lcd_rw : out STD_LOGIC;
                        sf_d : out STD_LOGIC_VECTOR(11 downto 8);
                        h_limit : in integer range 0 to 640;
                        v_limit : in integer range 0 to 480;
                        precision : in integer range 0 to 46000;
                        calibration : in integer range 0 to 16383);
end lcdmain;

architecture Behavioral of lcdmain is
        component lcdcontroller
                Port ( clk : in  STD_LOGIC;
                                write_e : in STD_LOGIC;
                                data : in  STD_LOGIC_VECTOR(7 downto 0);
                                command : in  STD_LOGIC;
                                reset : in STD_LOGIC;
                                lcd_e : out  STD_LOGIC;
                                lcd_rs : out  STD_LOGIC;
                                lcd_rw : out  STD_LOGIC;
                                sf_d : out  STD_LOGIC_VECTOR(11 downto 8));
        end component;

        component lcdwrite
                Port ( clk : in  STD_LOGIC;
                                reset : in STD_LOGIC;
                                lcd_e : in STD_LOGIC;
                                a1 : in STD_LOGIC_VECTOR(7 downto 0);
                                a2 : in STD_LOGIC_VECTOR(7 downto 0);
                                a3 : in STD_LOGIC_VECTOR(7 downto 0);
                                a4 : in STD_LOGIC_VECTOR(7 downto 0);
                                a5 : in STD_LOGIC_VECTOR(7 downto 0);
                                a6 : in STD_LOGIC_VECTOR(7 downto 0);
                                a7 : in STD_LOGIC_VECTOR(7 downto 0);
```

```
                a8 : in STD_LOGIC_VECTOR(7 downto 0);
                a9 : in STD_LOGIC_VECTOR(7 downto 0);
                a10 : in STD_LOGIC_VECTOR(7 downto 0);
                a11 : in STD_LOGIC_VECTOR(7 downto 0);
                a12 : in STD_LOGIC_VECTOR(7 downto 0);
                a13 : in STD_LOGIC_VECTOR(7 downto 0);
                a14 : in STD_LOGIC_VECTOR(7 downto 0);
                a15 : in STD_LOGIC_VECTOR(7 downto 0);
                a16 : in STD_LOGIC_VECTOR(7 downto 0);
                b1 : in STD_LOGIC_VECTOR(7 downto 0);
                b2 : in STD_LOGIC_VECTOR(7 downto 0);
                b3 : in STD_LOGIC_VECTOR(7 downto 0);
                b4 : in STD_LOGIC_VECTOR(7 downto 0);
                b5 : in STD_LOGIC_VECTOR(7 downto 0);
                b6 : in STD_LOGIC_VECTOR(7 downto 0);
                b7 : in STD_LOGIC_VECTOR(7 downto 0);
                b8 : in STD_LOGIC_VECTOR(7 downto 0);
                b9 : in STD_LOGIC_VECTOR(7 downto 0);
                b10 : in STD_LOGIC_VECTOR(7 downto 0);
                b11 : in STD_LOGIC_VECTOR(7 downto 0);
                b12 : in STD_LOGIC_VECTOR(7 downto 0);
                b13 : in STD_LOGIC_VECTOR(7 downto 0);
                b14 : in STD_LOGIC_VECTOR(7 downto 0);
                b15 : in STD_LOGIC_VECTOR(7 downto 0);
                b16 : in STD_LOGIC_VECTOR(7 downto 0);
                data : out STD_LOGIC_VECTOR(7 downto 0);
                command : out STD_LOGIC;
                write_e : out  STD_LOGIC);
end component;

component binarytodigits
        Port ( binary : in  STD_LOGIC_VECTOR(13 downto 0);
                one : out  integer range 0 to 9;
                ten : out  integer range 0 to 9;
                hundred : out  integer range 0 to 9;
                thousand : out  integer range 0 to 9;
                tenthousand : out  integer range 0 to 9);
end component;

component digittolcdcode
        Port ( digit : in  integer range 0 to 9;
                code : out  STD_LOGIC_VECTOR(7 downto 0));
end component;

signal lcd_enable : STD_LOGIC;
```

```
        signal write_e : STD_LOGIC;
        signal data : STD_LOGIC_VECTOR(7 downto 0);
        signal command : STD_LOGIC;
        signal a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,a16,
                        b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14,b15,b16 :
STD_LOGIC_VECTOR(7 downto 0);
        signal in_one : integer range 0 to 9;
        signal in_ten : integer range 0 to 9;
        signal in_hundred : integer range 0 to 9;
        signal in_thousand : integer range 0 to 9;
        signal in_tenthousand : integer range 0 to 9;
        signal write_one : STD_LOGIC_VECTOR(7 downto 0);
        signal write_ten : STD_LOGIC_VECTOR(7 downto 0);
        signal write_hundred : STD_LOGIC_VECTOR(7 downto 0);
        signal write_thousand : STD_LOGIC_VECTOR(7 downto 0);
        signal write_tenthousand : STD_LOGIC_VECTOR(7 downto 0);

        signal h_int : STD_LOGIC_VECTOR(13 downto 0);
        signal h_one : integer range 0 to 9;
        signal h_ten : integer range 0 to 9;
        signal h_hundred : integer range 0 to 9;
        signal h_thousand : integer range 0 to 9;
        signal h_tenthousand : integer range 0 to 9;
        signal h_write_one : STD_LOGIC_VECTOR(7 downto 0);
        signal h_write_ten : STD_LOGIC_VECTOR(7 downto 0);
        signal h_write_hundred : STD_LOGIC_VECTOR(7 downto 0);
        signal h_write_thousand : STD_LOGIC_VECTOR(7 downto 0);

        signal v_int : STD_LOGIC_VECTOR(13 downto 0);
        signal v_one : integer range 0 to 9;
        signal v_ten : integer range 0 to 9;
        signal v_hundred : integer range 0 to 9;
        signal v_thousand : integer range 0 to 9;
        signal v_tenthousand : integer range 0 to 9;
        signal v_write_one : STD_LOGIC_VECTOR(7 downto 0);
        signal v_write_ten : STD_LOGIC_VECTOR(7 downto 0);
        signal v_write_hundred : STD_LOGIC_VECTOR(7 downto 0);

        signal c_int : STD_LOGIC_VECTOR(13 downto 0);
        signal c_one : integer range 0 to 9;
        signal c_ten : integer range 0 to 9;
        signal c_hundred : integer range 0 to 9;
        signal c_thousand : integer range 0 to 9;
        signal c_tenthousand : integer range 0 to 9;
        signal c_write_one : STD_LOGIC_VECTOR(7 downto 0);
```

```
        signal c_write_ten : STD_LOGIC_VECTOR(7 downto 0);
        signal c_write_hundred : STD_LOGIC_VECTOR(7 downto 0);
        signal c_write_thousand : STD_LOGIC_VECTOR(7 downto 0);
        signal c_write_tenthousand : STD_LOGIC_VECTOR(7 downto 0);

        signal p_int : STD_LOGIC_VECTOR(13 downto 0);
        signal p_one : integer range 0 to 9;
        signal p_ten : integer range 0 to 9;
        signal p_hundred : integer range 0 to 9;
        signal p_thousand : integer range 0 to 9;
        signal p_tenthousand : integer range 0 to 9;
        signal p_write_one : STD_LOGIC_VECTOR(7 downto 0);
        signal p_write_ten : STD_LOGIC_VECTOR(7 downto 0);
        signal p_write_hundred : STD_LOGIC_VECTOR(7 downto 0);
        signal p_write_thousand : STD_LOGIC_VECTOR(7 downto 0);
        signal p_write_tenthousand : STD_LOGIC_VECTOR(7 downto 0);
begin

n1: lcdcontroller port map (clk,write_e,data,command,reset,lcd_enable,lcd_rs,lcd_rw,sf_d);
n2: lcdwrite port map (clk,reset,lcd_enable,
                                                    a1,a2,a3,a4,a5,a6,a7,a8,
                                                    a9,a10,a11,a12,a13,a14,a15,a16,
                                                    b1,b2,b3,b4,b5,b6,b7,b8,
                                                    b9,b10,b11,b12,b13,b14,b15,b16,
                                                    data,command,write_e);
n3: binarytodigits port map
(adc_in,in_one,in_ten,in_hundred,in_thousand,in_tenthousand);
n10: digittolcdcode port map (in_one,write_one);
n11: digittolcdcode port map (in_ten,write_ten);
n12: digittolcdcode port map (in_hundred,write_hundred);
n13: digittolcdcode port map (in_thousand,write_thousand);
n14: digittolcdcode port map (in_tenthousand,write_tenthousand);

h_int <= CONV_STD_LOGIC_VECTOR(h_limit,14);
n20: binarytodigits port map (h_int,h_one,h_ten,h_hundred,h_thousand,h_tenthousand);
n21: digittolcdcode port map (h_one,h_write_one);
n22: digittolcdcode port map (h_ten,h_write_ten);
n23: digittolcdcode port map (h_hundred,h_write_hundred);
n24: digittolcdcode port map (h_thousand,h_write_thousand);

v_int <= CONV_STD_LOGIC_VECTOR(v_limit,14);
n30: binarytodigits port map (v_int,v_one,v_ten,v_hundred,v_thousand,v_tenthousand);
n31: digittolcdcode port map (v_one,v_write_one);
n32: digittolcdcode port map (v_ten,v_write_ten);
n33: digittolcdcode port map (v_hundred,v_write_hundred);
```

```
c_int <= CONV_STD_LOGIC_VECTOR(calibration,14);
n40: binarytodigits port map (c_int,c_one,c_ten,c_hundred,c_thousand,c_tenthousand);
n41: digittolcdcode port map (c_one,c_write_one);
n42: digittolcdcode port map (c_ten,c_write_ten);
n43: digittolcdcode port map (c_hundred,c_write_hundred);
n44: digittolcdcode port map (c_thousand,c_write_thousand);
n45: digittolcdcode port map (c_tenthousand,c_write_tenthousand);

p_int <= CONV_STD_LOGIC_VECTOR(precision,14);
n50: binarytodigits port map (p_int,p_one,p_ten,p_hundred,p_thousand,p_tenthousand);
n51: digittolcdcode port map (p_one,p_write_one);
n52: digittolcdcode port map (p_ten,p_write_ten);
n53: digittolcdcode port map (p_hundred,p_write_hundred);
n54: digittolcdcode port map (p_thousand,p_write_thousand);
n55: digittolcdcode port map (p_tenthousand,p_write_tenthousand);

lcd_e <= lcd_enable;

a1 <= "01111111";
a2 <= "00100000";
a3 <= c_write_tenthousand;
a4 <= c_write_thousand;
a5 <= c_write_hundred;
a6 <= c_write_ten;
a7 <= c_write_one;
a8 <= "00100000";
a9 <= "00100000";
a10 <= "00100000";
a11 <= p_write_tenthousand;
a12 <= p_write_thousand;
a13 <= p_write_hundred;
a14 <= p_write_ten;
a15 <= p_write_one;
a16 <= "01111110";
b1 <= h_write_thousand;
b2 <= h_write_hundred;
b3 <= h_write_ten;
b4 <= h_write_one;
b5 <= "00100000";
b6 <= v_write_hundred;
b7 <= v_write_ten;
b8 <= v_write_one;
b9 <= "00100000";
b10 <= "00100000";
```

```
b11 <= "00100000";
b12 <= write_tenthousand;
b13 <= write_thousand;
b14 <= write_hundred;
b15 <= write_ten;
b16 <= write_one;



end Behavioral;
```

### *14. LCD Controller Circuit (Disabled)*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lcdcontroller is
   Port ( clk : in  STD_LOGIC;
                              write_e : in  STD_LOGIC;
          data : in  STD_LOGIC_VECTOR(7 downto 0);
          command : in  STD_LOGIC;
                              reset : in  STD_LOGIC;
          lcd_e : out  STD_LOGIC;
          lcd_rs : out  STD_LOGIC;
          lcd_rw : out  STD_LOGIC;
          sf_d : out  STD_LOGIC_VECTOR(11 downto 8));
end lcdcontroller;

architecture Behavioral of lcdcontroller is
          signal delay : integer range 0 to 750000 := 0;
          signal delay_active : STD_LOGIC;

          type state_type is (init0,init1,init2,init3,init4,init5,init6,init7,init8,
                                                                fs0,fs1,fs2,fs3,
                                                                ems0,ems1,ems2,ems3,
                                                                doo0,doo1,doo2,doo3,
                                                                cd0,cd1,cd2,cd3,cd4,
                                                                wr0,wr1,wr2,wr3,wr4);
          signal PS,NS : state_type;
begin

--LCD Control
--State Change Clock
sync_proc: process(clk,NS,reset)
begin
          if (reset = '1') then
                    PS <= init0;
          elsif (rising_edge(clk)) then
                    PS <= NS;
                    if (delay_active <= '0') then
                              delay <= 0;
                    else
                              delay <= delay + 1;
                    end if;
```

```
		end if;
end process sync_proc;

--State Change Condition
comb_proc: process(clk,PS,delay,write_e,command,data)
begin
		case PS is
-- Power-On Initialization --
		when init0 => -- Wait 15ms
			if (delay < 750000) then
				delay_active <= '1';
				NS <= init0;
			else
				delay_active <= '0';
				NS <= init1;
			end if;
			lcd_rs <= '0';
			lcd_e <= '0';
			sf_d <= "0000";
		when init1 => -- Write 0x3
			if (delay < 12) then
				delay_active <= '1';
				NS <= init1;
			else
				delay_active <= '0';
				NS <= init2;
			end if;
			lcd_rs <= '0';
			lcd_e <= '1';
			sf_d <= "0011";
		when init2 => -- Wait 4.1ms
			if (delay < 205000) then
				delay_active <= '1';
				NS <= init2;
			else
				delay_active <= '0';
				NS <= init3;
			end if;
			lcd_rs <= '0';
			lcd_e <= '0';
			sf_d <= "0000";
		when init3 => -- Write 0x3
			if (delay < 12) then
				delay_active <= '1';
				NS <= init3;
```

```
		else
			delay_active <= '0';
			NS <= init4;
		end if;
		lcd_rs <= '0';
		lcd_e <= '1';
		sf_d <= "0011";
when init4 => -- Wait 100us
		if (delay < 5000) then
			delay_active <= '1';
			NS <= init4;
		else
			delay_active <= '0';
			NS <= init5;
		end if;
		lcd_rs <= '0';
		lcd_e <= '0';
		sf_d <= "0000";
when init5 => -- Write 0x3
		if (delay < 12) then
			delay_active <= '1';
			NS <= init5;
		else
			delay_active <= '0';
			NS <= init6;
		end if;
		lcd_rs <= '0';
		lcd_e <= '1';
		sf_d <= "0011";
when init6 => -- Wait 40us
		if (delay < 2000) then
			delay_active <= '1';
			NS <= init6;
		else
			delay_active <= '0';
			NS <= init7;
		end if;
		lcd_rs <= '0';
		lcd_e <= '0';
		sf_d <= "0000";
when init7 => -- Write 0x2
		if (delay < 12) then
			delay_active <= '1';
			NS <= init7;
		else
```

```
                        delay_active <= '0';
                        NS <= init8;
                end if;
                lcd_rs <= '0';
                lcd_e <= '1';
                sf_d <= "0010";
        when init8 => -- Wait 40us
                if (delay < 2000) then
                        delay_active <= '1';
                        NS <= init8;
                else
                        delay_active <= '0';
                        NS <= fs0;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
-- Issue Function Set --
        when fs0 => -- Wait 40us
                if (delay < 2000) then
                        delay_active <= '1';
                        NS <= fs0;
                else
                        delay_active <= '0';
                        NS <= fs1;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
        when fs1 => -- Write 0x2
                if (delay < 12) then
                        delay_active <= '1';
                        NS <= fs1;
                else
                        delay_active <= '0';
                        NS <= fs2;
                end if;
                lcd_rs <= '0';
                lcd_e <= '1';
                sf_d <= "0010";
        when fs2 => -- Wait 1us
                if (delay < 50) then
                        delay_active <= '1';
                        NS <= fs2;
                else
```

```
                        delay_active <= '0';
                        NS <= fs3;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
        when fs3 => -- Write 0x8
                if (delay < 12) then
                        delay_active <= '1';
                        NS <= fs3;
                else
                        delay_active <= '0';
                        NS <= ems0;
                end if;
                lcd_rs <= '0';
                lcd_e <= '1';
                sf_d <= "1000";
-- Issue Entry Mode Set --
        when ems0 => -- Wait 40us
                if (delay < 2000) then
                        delay_active <= '1';
                        NS <= ems0;
                else
                        delay_active <= '0';
                        NS <= ems1;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
        when ems1 => -- Write 0x0
                if (delay < 12) then
                        delay_active <= '1';
                        NS <= ems1;
                else
                        delay_active <= '0';
                        NS <= ems2;
                end if;
                lcd_rs <= '0';
                lcd_e <= '1';
                sf_d <= "0000";
        when ems2 => -- Wait 1us
                if (delay < 50) then
                        delay_active <= '1';
                        NS <= ems2;
                else
```

```
                        delay_active <= '0';
                        NS <= ems3;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
        when ems3 => -- Write 0x6
                if (delay < 12) then
                        delay_active <= '1';
                        NS <= ems3;
                else
                        delay_active <= '0';
                        NS <= doo0;
                end if;
                lcd_rs <= '0';
                lcd_e <= '1';
                sf_d <= "0110";
-- Issue Display On/Off --
        when doo0 => -- Wait 40us
                if (delay < 2000) then
                        delay_active <= '1';
                        NS <= doo0;
                else
                        delay_active <= '0';
                        NS <= doo1;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
        when doo1 => -- Write 0x0
                if (delay < 12) then
                        delay_active <= '1';
                        NS <= doo1;
                else
                        delay_active <= '0';
                        NS <= doo2;
                end if;
                lcd_rs <= '0';
                lcd_e <= '1';
                sf_d <= "0000";
        when doo2 => -- Wait 1us
                if (delay < 50) then
                        delay_active <= '1';
                        NS <= doo2;
                else
```

```
                        delay_active <= '0';
                        NS <= doo3;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
        when doo3 => -- Write 0xC
                if (delay < 12) then
                        delay_active <= '1';
                        NS <= doo3;
                else
                        delay_active <= '0';
                        NS <= cd0;
                end if;
                lcd_rs <= '0';
                lcd_e <= '1';
                sf_d <= "1100";
-- Issue Clear Display --
        when cd0 => -- Wait 40us
                if (delay < 2000) then
                        delay_active <= '1';
                        NS <= cd0;
                else
                        delay_active <= '0';
                        NS <= cd1;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
        when cd1 => -- Write 0x0
                if (delay < 12) then
                        delay_active <= '1';
                        NS <= cd1;
                else
                        delay_active <= '0';
                        NS <= cd2;
                end if;
                lcd_rs <= '0';
                lcd_e <= '1';
                sf_d <= "0000";
        when cd2 => -- Wait 1us
                if (delay < 50) then
                        delay_active <= '1';
                        NS <= cd2;
                else
```

```
                        delay_active <= '0';
                        NS <= cd3;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
        when cd3 => -- Write 0x1
                if (delay < 12) then
                        delay_active <= '1';
                        NS <= cd3;
                else
                        delay_active <= '0';
                        NS <= cd4;
                end if;
                lcd_rs <= '0';
                lcd_e <= '1';
                sf_d <= "0001";
-- Long Delay after Clear Display
        when cd4 => -- Wait 1.64ms
                if (delay < 82000) then
                        delay_active <= '1';
                        NS <= cd4;
                else
                        delay_active <= '0';
                        NS <= wr0;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
-- Write Sequence
        when wr0 => -- Check for write enable
                if      (write_e = '0') then
                        NS <= wr0;
                elsif (write_e = '1') then
                        NS <= wr1;
                end if;
                lcd_rs <= '0';
                lcd_e <= '0';
                sf_d <= "0000";
        when wr1 => -- Wait 40us
                if (delay < 2000) then
                        delay_active <= '1';
                        NS <= wr1;
                else
                        delay_active <= '0';
```

```
                        NS <= wr2;
                end if;
                lcd_rs <= not(command);
                lcd_e <= '0';
                sf_d <= "0000";
        when wr2 => -- Write Upper 4 Bits
                if (delay < 12) then
                        delay_active <= '1';
                        NS <= wr2;
                else
                        delay_active <= '0';
                        NS <= wr3;
                end if;
                lcd_rs <= not(command);
                lcd_e <= '1';
                sf_d <= data(7 downto 4);
        when wr3 => -- Wait 1us
                if (delay < 50) then
                        delay_active <= '1';
                        NS <= wr3;
                else
                        delay_active <= '0';
                        NS <= wr4;
                end if;
                lcd_rs <= not(command);
                lcd_e <= '0';
                sf_d <= "0000";
        when wr4 => -- Write Lower 4 Bits
                if (delay < 12) then
                        delay_active <= '1';
                        NS <= wr4;
                else
                        delay_active <= '0';
                        NS <= wr0;
                end if;
                lcd_rs <= not(command);
                lcd_e <= '1';
                sf_d <= data(3 downto 0);
        when others => -- should never go into this
                NS <= init0;
        end case;
end process;

lcd_rw <= '0';
```

```
end Behavioral;
```

### *15. LCD Write Circuit (Disabled)*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity lcdwrite is
   Port ( clk : in  STD_LOGIC;
                              reset : in STD_LOGIC;
                              lcd_e : in STD_LOGIC;
                              a1 : in STD_LOGIC_VECTOR(7 downto 0);
                              a2 : in STD_LOGIC_VECTOR(7 downto 0);
                              a3 : in STD_LOGIC_VECTOR(7 downto 0);
                              a4 : in STD_LOGIC_VECTOR(7 downto 0);
                              a5 : in STD_LOGIC_VECTOR(7 downto 0);
                              a6 : in STD_LOGIC_VECTOR(7 downto 0);
                              a7 : in STD_LOGIC_VECTOR(7 downto 0);
                              a8 : in STD_LOGIC_VECTOR(7 downto 0);
                              a9 : in STD_LOGIC_VECTOR(7 downto 0);
                              a10 : in STD_LOGIC_VECTOR(7 downto 0);
                              a11 : in STD_LOGIC_VECTOR(7 downto 0);
                              a12 : in STD_LOGIC_VECTOR(7 downto 0);
                              a13 : in STD_LOGIC_VECTOR(7 downto 0);
                              a14 : in STD_LOGIC_VECTOR(7 downto 0);
                              a15 : in STD_LOGIC_VECTOR(7 downto 0);
                              a16 : in STD_LOGIC_VECTOR(7 downto 0);
                              b1 : in STD_LOGIC_VECTOR(7 downto 0);
                              b2 : in STD_LOGIC_VECTOR(7 downto 0);
                              b3 : in STD_LOGIC_VECTOR(7 downto 0);
                              b4 : in STD_LOGIC_VECTOR(7 downto 0);
                              b5 : in STD_LOGIC_VECTOR(7 downto 0);
                              b6 : in STD_LOGIC_VECTOR(7 downto 0);
                              b7 : in STD_LOGIC_VECTOR(7 downto 0);
                              b8 : in STD_LOGIC_VECTOR(7 downto 0);
                              b9 : in STD_LOGIC_VECTOR(7 downto 0);
                              b10 : in STD_LOGIC_VECTOR(7 downto 0);
                              b11 : in STD_LOGIC_VECTOR(7 downto 0);
                              b12 : in STD_LOGIC_VECTOR(7 downto 0);
                              b13 : in STD_LOGIC_VECTOR(7 downto 0);
                              b14 : in STD_LOGIC_VECTOR(7 downto 0);
                              b15 : in STD_LOGIC_VECTOR(7 downto 0);
                              b16 : in STD_LOGIC_VECTOR(7 downto 0);
                              data : out STD_LOGIC_VECTOR(7 downto 0);
                              command : out STD_LOGIC;
```

```
                                write_e : out  STD_LOGIC);
end lcdwrite;

architecture Behavioral of lcdwrite is
        signal next_write : integer range 0 to 33 := 0;
        signal next_indicator : STD_LOGIC;
        signal write_done : STD_LOGIC;
        signal delay : integer range 0 to 750000 := 0;
        signal delay_active : STD_LOGIC;

        type state_type is (wr0,wr1,wr2);
        signal PS,NS : state_type;
begin

--State Change Clock
sync_proc: process(clk,NS,reset)
begin
        if (reset = '1') then
                PS <= wr0;
        elsif (rising_edge(clk)) then
                PS <= NS;
                if (next_indicator = '1') then
                        next_write <= next_write + 1;
                end if;
                if (delay_active <= '0') then
                        delay <= 0;
                else
                        delay <= delay + 1;
                end if;
        end if;
end process sync_proc;

--State Change Condition
comb_proc: process(clk,PS,delay)
begin
        case PS is
        when wr0 => -- Prepare for next write
                NS <= wr1;
                delay_active <= '0';
                next_indicator <= '1';
        when wr1 => -- Write Values
                NS <= wr2;
                delay_active <= '0';
                next_indicator <= '0';
        when wr2 => -- Wait for write to finish
```

```
                if (delay < 2100) then
                        delay_active <= '1';
                        NS <= wr2;
                else
                        delay_active <= '0';
                        NS <= wr0;
                end if;
                next_indicator <= '0';
        when others => -- should never go into this
                NS <= wr0;
        end case;
end process;

write_e <= next_indicator;

data <= "10000000" when next_write = 0 else
                a1 when next_write = 1 else
                a2 when next_write = 2 else
                a3 when next_write = 3 else
                a4 when next_write = 4 else
                a5 when next_write = 5 else
                a6 when next_write = 6 else
                a7 when next_write = 7 else
                a8 when next_write = 8 else
                a9 when next_write = 9 else
                a10 when next_write = 10 else
                a11 when next_write = 11 else
                a12 when next_write = 12 else
                a13 when next_write = 13 else
                a14 when next_write = 14 else
                a15 when next_write = 15 else
                a16 when next_write = 16 else
                "11000000" when next_write = 17 else
                b1 when next_write = 18 else
                b2 when next_write = 19 else
                b3 when next_write = 20 else
                b4 when next_write = 21 else
                b5 when next_write = 22 else
                b6 when next_write = 23 else
                b7 when next_write = 24 else
                b8 when next_write = 25 else
                b9 when next_write = 26 else
                b10 when next_write = 27 else
                b11 when next_write = 28 else
                b12 when next_write = 29 else
```

```
                    b13 when next_write = 30 else
                    b14 when next_write = 31 else
                    b15 when next_write = 32 else
                    b16 when next_write = 33 else
                    "00000001";

command <= '1' when next_write = 0 or next_write = 17 else
                                '0';

end Behavioral;
```

### *16. Binary to Decimal Converter 14-Bit (Disabled)*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity binarytodigits is
   Port ( binary : in  STD_LOGIC_VECTOR(13 downto 0);
          one : out  integer range 0 to 9;
          ten : out  integer range 0 to 9;
          hundred : out  integer range 0 to 9;
          thousand : out  integer range 0 to 9;
                                tenthousand : out  integer range 0 to 9);
end binarytodigits;

architecture Behavioral of binarytodigits is
          signal hex_thousand : STD_LOGIC_VECTOR(13 downto 0);
          signal hex_hundred : STD_LOGIC_VECTOR(13 downto 0);
          signal hex_ten : STD_LOGIC_VECTOR(13 downto 0);

          signal bin_tenthousand : integer range 0 to 9;
          signal bin_thousand : integer range 0 to 9;
          signal bin_hundred : integer range 0 to 9;
          signal bin_ten : integer range 0 to 9;
begin

tenthousand <= bin_tenthousand;
thousand <= bin_thousand;
hundred <= bin_hundred;
ten <= bin_ten;

comb_proc: process (binary,hex_thousand,hex_hundred,hex_ten)
          begin
                    if (binary >= 10000) then bin_tenthousand <= 1;
                    else bin_tenthousand <= 0;
                    end if;

                    hex_thousand <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER(binary) -
CONV_INTEGER(bin_tenthousand)*10000, 14);

                    if (hex_thousand >= 9000) then bin_thousand <= 9;
                    elsif (hex_thousand >= 8000) then bin_thousand <= 8;
                    elsif (hex_thousand >= 7000) then bin_thousand <= 7;
                    elsif (hex_thousand >= 6000) then bin_thousand <= 6;
```

```
                elsif (hex_thousand >= 5000) then bin_thousand <= 5;
                elsif (hex_thousand >= 4000) then bin_thousand <= 4;
                elsif (hex_thousand >= 3000) then bin_thousand <= 3;
                elsif (hex_thousand >= 2000) then bin_thousand <= 2;
                elsif (hex_thousand >= 1000) then bin_thousand <= 1;
                else bin_thousand <= 0;
                end if;

                hex_hundred <=
CONV_STD_LOGIC_VECTOR(CONV_INTEGER(hex_thousand) -
CONV_INTEGER(bin_thousand)*1000, 14);

                if (hex_hundred >= 900) then bin_hundred <= 9;
                elsif (hex_hundred >= 800) then bin_hundred <= 8;
                elsif (hex_hundred >= 700) then bin_hundred <= 7;
                elsif (hex_hundred >= 600) then bin_hundred <= 6;
                elsif (hex_hundred >= 500) then bin_hundred <= 5;
                elsif (hex_hundred >= 400) then bin_hundred <= 4;
                elsif (hex_hundred >= 300) then bin_hundred <= 3;
                elsif (hex_hundred >= 200) then bin_hundred <= 2;
                elsif (hex_hundred >= 100) then bin_hundred <= 1;
                else bin_hundred <= 0;
                end if;

                hex_ten <= CONV_STD_LOGIC_VECTOR(CONV_INTEGER(hex_hundred) -
CONV_INTEGER(bin_hundred)*100, 14);

                if (hex_ten >= 90) then bin_ten <= 9;
                elsif (hex_ten >= 80) then bin_ten <= 8;
                elsif (hex_ten >= 70) then bin_ten <= 7;
                elsif (hex_ten >= 60) then bin_ten <= 6;
                elsif (hex_ten >= 50) then bin_ten <= 5;
                elsif (hex_ten >= 40) then bin_ten <= 4;
                elsif (hex_ten >= 30) then bin_ten <= 3;
                elsif (hex_ten >= 20) then bin_ten <= 2;
                elsif (hex_ten >= 10) then bin_ten <= 1;
                else bin_ten <= 0;
                end if;

                one <= CONV_INTEGER(hex_ten) - CONV_INTEGER(bin_ten)*10;
        end process;

end Behavioral;
```

### *17. Digit to ASCII code (Disabled)*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity digittolcdcode is
   Port ( digit : in  integer range 0 to 9;
          code : out  STD_LOGIC_VECTOR(7 downto 0));
end digittolcdcode;

architecture Behavioral of digittolcdcode is

begin

code <= "00110000" when digit = 0 else
                        "00110001" when digit = 1 else
                        "00110010" when digit = 2 else
                        "00110011" when digit = 3 else
                        "00110100" when digit = 4 else
                        "00110101" when digit = 5 else
                        "00110110" when digit = 6 else
                        "00110111" when digit = 7 else
                        "00111000" when digit = 8 else
                        "00111001" when digit = 9 else
                        "00100000";

end Behavioral;
```

### *18. VGA Display Control Circuit (Disabled)*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vgadisplay is
   Port ( clk : in  STD_LOGIC;
          red : out  STD_LOGIC;
          green : out  STD_LOGIC;
          blue : out  STD_LOGIC;
          hs : out  STD_LOGIC;
          vs : out  STD_LOGIC;
                              adc_in : in STD_LOGIC_VECTOR(13 downto 0);
                              j1 : in STD_LOGIC_VECTOR(1 downto 0);
                              h_limit_out : out integer range 0 to 4800;
                              v_limit_out : out integer range 0 to 571;
                              btn_east : in STD_LOGIC;
                              btn_west : in STD_LOGIC;
                              up_down : in STD_LOGIC;
                              count_pulse : in STD_LOGIC;
                              x_position : in integer range 0 to 640;
                              y_position : in integer range 0 to 480);
end vgadisplay;

architecture Behavioral of vgadisplay is
          component pulse
                    Port ( clk : in  STD_LOGIC;
                                        signal_in : in  STD_LOGIC;
                                        pulse : out  STD_LOGIC);
          end component;

          signal clock_25mhz : STD_LOGIC;
          signal h_counter : integer range 0 to 850 := 0;
          signal v_counter : integer range 0 to 571 := 0;
          signal flash : integer range 0 to 2 := 0;
          signal deb_v : STD_LOGIC;
          signal deb_h : STD_LOGIC;
          signal h_limit : integer range 0 to 4800 := 109;
          signal v_limit : integer range 471 to 571 := 521;

          signal delay_count_v : integer range 0 to 833600 := 833600;
          signal delay_count_h : integer range 0 to 4800 := 4800;
```

```
	signal adc_adjust : STD_LOGIC_VECTOR(13 downto 0);

	signal display_x1 : integer range 0 to 4800;
	signal display_x2 : integer range 0 to 4800;
	signal display_y : integer range 0 to 833600;
begin

display_x1 <= (x_position * 2) + 288;
display_x2 <= (x_position * 2) + 1888;
display_y <= (y_position * 1600) + 62400;

n1: pulse port map (clk,not(j1(0)),deb_h);
n2: pulse port map (clk,not(j1(1)),deb_v);

h_limit_out <= h_limit;
v_limit_out <= v_limit;

adc_adjust <= CONV_STD_LOGIC_VECTOR((((CONV_INTEGER(adc_in) - 3277) / 64) * 133),
14) when CONV_INTEGER(adc_in) > 3277 else
                                        "00000000000000";

process (clk)
begin
	if (rising_edge(clk)) then
		if (deb_v = '1') then
			delay_count_v <= 0;
			delay_count_h <= 0;
			vs <= '0';
			hs <= '0';
			green <= '0';
			red <= '0';
			blue <= '0';
		elsif (delay_count_v < 3200) then
			delay_count_v <= delay_count_v + 1;
			vs <= '0';
			green <= '0';
			red <= '0';
			blue <= '0';
			if (delay_count_h < 192) then
				delay_count_h <= delay_count_h + 1;
				hs <= '0';
			elsif (delay_count_h < 1600) then
				delay_count_h <= delay_count_h + 1;
				hs <= '1';
			else
```

```
                        delay_count_h <= 0;
                        hs <= '1';
                end if;
        elsif (delay_count_v < 62400) then
                delay_count_v <= delay_count_v + 1;
                vs <= '1';
                green <= '0';
                red <= '0';
                blue <= '0';
                if (delay_count_h < 192) then
                        delay_count_h <= delay_count_h + 1;
                        hs <= '0';
                elsif (delay_count_h < 1600) then
                        delay_count_h <= delay_count_h + 1;
                        hs <= '1';
                else
                        delay_count_h <= 0;
                        hs <= '1';
                end if;
        elsif (delay_count_v < 824640) then  --Display Stuff here
                delay_count_v <= delay_count_v + 1;
                vs <= '1';
                if (delay_count_h < 192) then
                        delay_count_h <= delay_count_h + 1;
                        hs <= '0';
                        green <= '0';
                        red <= '0';
                        blue <= '0';
                elsif (delay_count_h < 288) then
                        delay_count_h <= delay_count_h + 1;
                        hs <= '1';
                        green <= '0';
                        red <= '0';
                        blue <= '0';
                elsif (delay_count_h < 1556) then
                        delay_count_h <= delay_count_h + 1;
                        hs <= '1';
                        if (((delay_count_v - 12800) < display_y) and (display_y <
(delay_count_v + 12800))) and
                                (((delay_count_h - 8) < display_x1) and (display_x1
< (delay_count_h + 8))) then
                                green <= '1';
                                red <= '1';
                                blue <= '1';
                        else
```

```
                                green <= adc_adjust(13);
                                red <= adc_adjust(12);
                                blue <= adc_adjust(11);
                        end if;
                elsif (delay_count_h < 1588) then
                        delay_count_h <= delay_count_h + 1;
                        hs <= '1';
                        green <= '0';
                        red <= '0';
                        blue <= '0';
--              elsif (delay_count_h < 1780) then
--                      delay_count_h <= delay_count_h + 1;
--                      hs <= '0';
--                      green <= '0';
--                      red <= '0';
--                      blue <= '0';
--              elsif (delay_count_h < 1888) then
--                      delay_count_h <= delay_count_h + 1;
--                      hs <= '1';
--                      green <= '0';
--                      red <= '0';
--                      blue <= '0';
--              elsif (delay_count_h < 3168) then
--                      delay_count_h <= delay_count_h + 1;
--                      hs <= '1';
--                      green <= adc_adjust(13);
--                      red <= adc_adjust(12);
--                      blue <= adc_adjust(11);
--              elsif (delay_count_h < 3177) then
--                      delay_count_h <= delay_count_h + 1;
--                      hs <= '1';
--                      green <= '0';
--                      red <= '0';
--                      blue <= '0';
                else
                        delay_count_h <= 0;
                        hs <= '1';
                        green <= '0';
                        red <= '0';
                        blue <= '0';
                end if;
        elsif (delay_count_v < 833600) then
                delay_count_v <= delay_count_v + 1;
                vs <= '1';
                green <= '0';
```

```
                        red <= '0';
                        blue <= '0';
                        if (delay_count_h < 192) then
                                delay_count_h <= delay_count_h + 1;
                                hs <= '0';
                        elsif (delay_count_h < 1588) then
                                delay_count_h <= delay_count_h + 1;
                                hs <= '1';
                        else
                                delay_count_h <= 0;
                                hs <= '1';
                        end if;
                else
                        delay_count_v <= 0;
                        vs <= '1';
                        hs <= '1';
                        green <= '0';
                        red <= '0';
                        blue <= '0';
                end if;
        end if;
end process;

process (up_down,count_pulse)
begin
        if (rising_edge(count_pulse)) then
                if (up_down = '1' and btn_east = '1') then
                        h_limit <= h_limit + 1;
                elsif (up_down = '1' and btn_west = '1') then
                        v_limit <= v_limit + 1;
                elsif (up_down = '0' and btn_east = '1') then
                        h_limit <= h_limit - 1;
                elsif (up_down = '0' and btn_west = '1') then
                        v_limit <= v_limit - 1;
                end if;
        end if;
end process;

--process (clk)
--begin
--      if (rising_edge(clk)) then
--              if (clock_25mhz = '0') then
--                      clock_25mhz <= '1';
--              else
--                      clock_25mhz <= '0';
```

```
--                      end if;
--          end if;
--end process;

--process (clock_25mhz)
--begin
--          if (rising_edge(clock_25mhz)) then
--                      if (h_counter >= 144)
--                      and (h_counter < 784)
--                      and (v_counter >= 39)
--                      and (v_counter < 519) then
--                                  green <= adc_in(13);
--                                  red <= adc_in(12);
--                                  blue <= adc_in(11);
--                      end if;
--                      if (h_counter > 0)
--                      and (h_counter < 97) then
--                                  hs <= '0';
--                      else
--                                  hs <= '1';
--                      end if;
--                      if (v_counter >= 0)
--                      and (v_counter < 3) then
--                                  vs <= '0';
--                      else
--                                  vs <= '1';
--                      end if;
--                      if (h_counter = h_limit) then
--                                  h_counter <= 0;
--                                  if (v_counter = v_limit) then
--                                              v_counter <= 0;
--                                  elsif (deb_v <= '1') then
--                                              v_counter <= 0;
--                                  else
--                                              v_counter <= v_counter + 1;
--                                  end if;
--                      elsif (j1(0) = '0') then
--                                  h_counter <= 0;
--                      else
--                                  h_counter <= h_counter + 1;
--                      end if;
--          end if;
--end process;

end Behavioral;
```

## L. Auto Hot Key Code for the Auto Hot Key Software

```
MsgBox,,Laser Cursor,Begin Pointer Controller,2

;#########################################################################
;###### User Variables ###################################################
;#########################################################################
RS232_Port     = COM3
RS232_Baud     = 115200
RS232_Parity   = N
RS232_Data     = 8
RS232_Stop     = 1

;#########################################################################
;###### Script Variables #################################################
;#########################################################################
RS232_Settings   = %RS232_Port%:baud=%RS232_Baud% parity=%RS232_Parity%
data=%RS232_Data% stop=%RS232_Stop% dtr=Off

;#########################################################################
;###### Serial Port Receive ##############################################
;#########################################################################
;Quit_var is used to exit the RS232 COM port receive loop
;  0=Don't Exit; 1=Exit; CTRL-F1 to set to 1 and exit script.
Quit_var = 0

RightClick := 0
LeftClick := 0
xcount := 0
ycount := 0
RS232_FileHandle:=RS232_Initialize(RS232_Settings)

InCalibration := 0
VerticalMove := 0
HorizontalMove := 0
VerticalStretch := 190
HorizontalStretch := 65

Calibrate1 := 0
Calibrate2 := 0
Calibrate3 := 0
Calibrate4 := 0
```

```
;##############################################################################
##############################################################################
##########

;RS232 COM port receive loop
Loop
{
 ;Start Location Coding

 ;Code to locate start value split the data into their respective representations.
 ;Code format FF;00;Xcoord;Ycoord;Click;00;FF
 InitialValue = AA
 Loop
 {
  InitialValue := RS232_Read(RS232_FileHandle,"0x01",RS232_Bytes_Received)
  IfInString, InitialValue, FF
  {
   Loop
   {
    SecondValue := RS232_Read(RS232_FileHandle,"0x01",RS232_Bytes_Received)
    IfInString, SecondValue, 00
    {
     break
    }
    else
    {
     IfInString, SecondValue, FF
      InitialValue = %SecondValue%
    }
   }
   IfInString, InitialValue, FF
    IfInString, SecondValue, 00
     break
  }
 }
 xcoords := RS232_Read(RS232_FileHandle,"0x01",RS232_Bytes_Received)
 xcoords := HexConvert(xcoords)
 ycoords := RS232_Read(RS232_FileHandle,"0x01",RS232_Bytes_Received)
 ycoords := HexConvert(ycoords)
 clicks := RS232_Read(RS232_FileHandle,"0x01",RS232_Bytes_Received)
 clicks := HexConvert(clicks)

 ;Convert clicks into a hex
 SetFormat, Integer, HEX
 clicks += 0
```

```
  ;store click data into memory
  PreviousRightClick := RightClick
  PreviousLeftClick := LeftClick
  RightClick := clicks >> 1
  LeftClick := clicks - (RightClick << 1)

  ;averaging for pointer that is within a certain proximity
  if (xcount < 10)
  {
   xcount += 1
  }
  if (xcount = 1)
  {
   xstorage10 = 0
   xstorage9 = 0
   xstorage8 = 0
   xstorage7 = 0
   xstorage6 = 0
   xstorage5 = 0
   xstorage4 = 0
   xstorage3 = 0
   xstorage2 = 0
   xstorage1 = %xcoords%
  }
  else if (((xcoords + 5) > xstorage1) and ((xcoords - 5) < xstorage1))
  {
   xstorage10 = %xstorage9%
   xstorage9 = %xstorage8%
   xstorage8 = %xstorage7%
   xstorage7 = %xstorage6%
   xstorage6 = %xstorage5%
   xstorage5 = %xstorage4%
   xstorage4 = %xstorage3%
   xstorage3 = %xstorage2%
   xstorage2 = %xstorage1%
   xstorage1 = %xcoords%
   xcoords := (xstorage10 + xstorage9 + xstorage8 + xstorage7 + xstorage6 + xstorage5 +
xstorage4 + xstorage3 + xstorage2 + xstorage1) / xcount
  }
  else
  {
   xcount := 0
  }
  if (ycount < 10)
```

```
  {
   ycount += 1
  }
  if (ycount = 1)
  {
   ystorage10 = 0
   ystorage9 = 0
   ystorage8 = 0
   ystorage7 = 0
   ystorage6 = 0
   ystorage5 = 0
   ystorage4 = 0
   ystorage3 = 0
   ystorage2 = 0
   ystorage1 = %ycoords%
  }
  else if (((ycoords + 5) > ystorage1) and ((ycoords - 5) < ystorage1))
  {
   ystorage10 = %ystorage9%
   ystorage9 = %ystorage8%
   ystorage8 = %ystorage7%
   ystorage7 = %ystorage6%
   ystorage6 = %ystorage5%
   ystorage5 = %ystorage4%
   ystorage4 = %ystorage3%
   ystorage3 = %ystorage2%
   ystorage2 = %ystorage1%
   ystorage1 = %ycoords%
   ycoords := (ystorage10 + ystorage9 + ystorage8 + ystorage7 + ystorage6 + ystorage5 +
ystorage4 + ystorage3 + ystorage2 + ystorage1) / ycount
  }
  else
  {
   ycount := 0
  }

  if (Calibrate1 = 1)
  {
   xTLstorage := xcoords
   yTLstorage := ycoords
   Calibrate1 := 0
  }
  if (Calibrate2 = 1)
  {
   xTRstorage := xcoords
```

```
  yTRstorage := ycoords
  Calibrate2 := 0
 }
 if (Calibrate3 = 1)
 {
  xBstorage := xcoords
  yBstorage := ycoords
  Calibrate3 := 0
 }
 if (Calibrate4 = 1)
 {
  MsgBox, %xTRstorage% `n%xTLstorage% `n%yBstorage% `n%yTLstorage%
  HorizontalStretch := xTRstorage - xTLstorage
  VerticalStretch := yBstorage - yTLstorage
  HorizontalMove := xTLstorage
  VerticalMove := yTLstorage
  Calibrate4 := 0
 }

 xcoords := ((xcoords - HorizontalMove) / HorizontalStretch) * A_ScreenWidth
 ycoords := ((ycoords - VerticalMove) / VerticalStretch) * A_ScreenHeight
 if (RightClick = 1 and PreviousRightClick = 0)
  MouseClick, Right,,,,,D
 else if (RightClick = 0 and PreviousRightClick = 1)
  MouseClick, Right,,,,,U
 if (LeftClick = 1 and PreviousLeftClick = 0)
  MouseClick, Left,,,,,D
 else if (LeftClick = 0 and PreviousLeftClick = 1)
  MouseClick, Left,,,,,U
 DllCall("SetCursorPos", int, xcoords, int, ycoords)

 if (quit_var = 1)
 {
  break
 }
}

MsgBox, Closing Pointer Control
RS232_Close(RS232_FileHandle)
ExitApp ;Exit Script
Return

;##############################################################################
##############################################################################
##########
```

```
;###############################################################################
;###### Convert Hex to a Number ################################################
;###############################################################################
HexConvert(StringValue)
{
 StringLeft, LeftSide, StringValue, 1
 StringRight, RightSide, StringValue, 1

 IfInString, LeftSide, F
  LeftSide = 15
 else IfInString, LeftSide, E
  LeftSide = 14
 else IfInString, LeftSide, D
  LeftSide = 13
 else IfInString, LeftSide, C
  LeftSide = 12
 else IfInString, LeftSide, B
  LeftSide = 11
 else IfInString, LeftSide, A
  LeftSide = 10
 else IfInString, LeftSide, 9
  LeftSide = 9
 else IfInString, LeftSide, 8
  LeftSide = 8
 else IfInString, LeftSide, 7
  LeftSide = 7
 else IfInString, LeftSide, 6
  LeftSide = 6
 else IfInString, LeftSide, 5
  LeftSide = 5
 else IfInString, LeftSide, 4
  LeftSide = 4
 else IfInString, LeftSide, 3
  LeftSide = 3
 else IfInString, LeftSide, 2
  LeftSide = 2
 else IfInString, LeftSide, 1
  LeftSide = 1
 else IfInString, LeftSide, 0
  LeftSide = 0

 IfInString, RightSide, F
  RightSide = 15
 else IfInString, RightSide, E
```

```
   RightSide = 14
  else IfInString, RightSide, D
   RightSide = 13
  else IfInString, RightSide, C
   RightSide = 12
  else IfInString, RightSide, B
   RightSide = 11
  else IfInString, RightSide, A
   RightSide = 10
  else IfInString, RightSide, 9
   RightSide = 9
  else IfInString, RightSide, 8
   RightSide = 8
  else IfInString, RightSide, 7
   RightSide = 7
  else IfInString, RightSide, 6
   RightSide = 6
  else IfInString, RightSide, 5
   RightSide = 5
  else IfInString, RightSide, 4
   RightSide = 4
  else IfInString, RightSide, 3
   RightSide = 3
  else IfInString, RightSide, 2
   RightSide = 2
  else IfInString, RightSide, 1
   RightSide = 1
  else IfInString, RightSide, 0
   RightSide = 0

  Total := LeftSide * 16 + RightSide

  return %Total%
}

;###############################################################################
;###### Initialize RS232 COM Subroutine ########################################
;###############################################################################
RS232_Initialize(RS232_Settings)
{
  ;###### Extract/Format the RS232 COM Port Number ######
  ;7/23/08 Thanks krisky68 for finding/solving the bug in which RS232 COM Ports greater
than 9 didn't work.
  StringSplit, RS232_Temp, RS232_Settings, `:
```

```
 RS232_Temp1_Len := StrLen(RS232_Temp1)  ;For COM Ports > 9 \\.\ needs to prepended to the COM Port name.
 If (RS232_Temp1_Len > 4)               ;So the valid names are
  RS232_COM = \\.\%RS232_Temp1%          ; ... COM8  COM9   \\.\COM10  \\.\COM11 \\.\COM12 and so on...
 Else                                   ;
  RS232_COM = %RS232_Temp1%

 ;8/10/09 A BIG Thanks to trenton_xavier for figuring out how to make COM Ports greater than 9 work for USB-Serial Dongles.
 StringTrimLeft, RS232_Settings, RS232_Settings, RS232_Temp1_Len+1 ;Remove the COM number (+1 for the semicolon) for BuildCommDCB.
 ;MsgBox, RS232_COM=%RS232_COM% `nRS232_Settings=%RS232_Settings%

 ;###### Build RS232 COM DCB ######
 ;Creates the structure that contains the RS232 COM Port number, baud rate,...
 VarSetCapacity(DCB, 28)
 BCD_Result := DllCall("BuildCommDCB"
     ,"str" , RS232_Settings ;lpDef
     ,"UInt", &DCB)       ;lpDCB
 If (BCD_Result <> 1)
 {
  MsgBox, There is a problem with Serial Port communication. `nFailed Dll BuildCommDCB, BCD_Result=%BCD_Result% `nThe Script Will Now Exit.
  Exit
 }

 ;###### Create RS232 COM File ######
 ;Creates the RS232 COM Port File Handle
 RS232_FileHandle := DllCall("CreateFile"
     ,"Str" , RS232_COM     ;File Name
     ,"UInt", 0xC0000000   ;Desired Access
     ,"UInt", 3          ;Safe Mode
     ,"UInt", 0          ;Security Attributes
     ,"UInt", 3          ;Creation Disposition
     ,"UInt", 0          ;Flags And Attributes
     ,"UInt", 0          ;Template File
     ,"Cdecl Int")

 If (RS232_FileHandle < 1)
 {
  MsgBox, There is a problem with Serial Port communication. `nFailed Dll CreateFile, RS232_FileHandle=%RS232_FileHandle% `nThe Script Will Now Exit.
  Exit
 }
```

```
  ;###### Set COM State ######
  ;Sets the RS232 COM Port number, baud rate,...
  SCS_Result := DllCall("SetCommState"
      ,"UInt", RS232_FileHandle ;File Handle
      ,"UInt", &DCB)         ;Pointer to DCB structure
  If (SCS_Result <> 1)
  {
    MsgBox, There is a problem with Serial Port communication. `nFailed Dll SetCommState,
SCS_Result=%SCS_Result% `nThe Script Will Now Exit.
    RS232_Close(RS232_FileHandle)
    Exit
  }

  ;###### Create the SetCommTimeouts Structure ######
  ReadIntervalTimeout       = 0x0000000f
  ReadTotalTimeoutMultiplier = 0x00000000
  ReadTotalTimeoutConstant   = 0x00000000
  WriteTotalTimeoutMultiplier= 0x00000000
  WriteTotalTimeoutConstant  = 0x00000000

  VarSetCapacity(Data, 20, 0) ; 5 * sizeof(DWORD)
  NumPut(ReadIntervalTimeout,        Data,  0, "UInt")
  NumPut(ReadTotalTimeoutMultiplier,  Data,  4, "UInt")
  NumPut(ReadTotalTimeoutConstant,    Data,  8, "UInt")
  NumPut(WriteTotalTimeoutMultiplier, Data, 12, "UInt")
  NumPut(WriteTotalTimeoutConstant,   Data, 16, "UInt")

  ;###### Set the RS232 COM Timeouts ######
  SCT_result := DllCall("SetCommTimeouts"
    ,"UInt", RS232_FileHandle ;File Handle
    ,"UInt", &Data)         ;Pointer to the data structure
  If (SCT_result <> 1)
  {
    MsgBox, There is a problem with Serial Port communication. `nFailed Dll SetCommState,
SCT_result=%SCT_result% `nThe Script Will Now Exit.
    RS232_Close(RS232_FileHandle)
    Exit
  }

  Return %RS232_FileHandle%
}

;#############################################################################
;###### Close RS23 COM Subroutine ########################################
```

```
;#####################################################################
RS232_Close(RS232_FileHandle)
{
 ;###### Close the COM File ######
 CH_result := DllCall("CloseHandle", "UInt", RS232_FileHandle)
 If (CH_result <> 1)
  MsgBox, Failed Dll CloseHandle CH_result=%CH_result%

 Return
}

;#####################################################################
;###### Read from RS232 COM Subroutines ##############################
;#####################################################################
RS232_Read(RS232_FileHandle,Num_Bytes,ByRef RS232_Bytes_Received)
{
 SetFormat, Integer, HEX

 ;Set the Data buffer size, prefill with 0x55 = ASCII character "U"
 ;VarSetCapacity won't assign anything less than 3 bytes. Meaning: If you
 ;  tell it you want 1 or 2 byte size variable it will give you 3.
 Data_Length  := VarSetCapacity(Data, Num_Bytes, 0x55)
 ;MsgBox, Data_Length=%Data_Length%

 ;###### Read the data from the RS232 COM Port ######
 ;MsgBox, RS232_FileHandle=%RS232_FileHandle% `nNum_Bytes=%Num_Bytes%
 Read_Result := DllCall("ReadFile"
    ,"UInt" , RS232_FileHandle   ; hFile
    ,"Str"  , Data           ; lpBuffer
    ,"Int"  , Num_Bytes      ; nNumberOfBytesToRead
    ,"UInt*", RS232_Bytes_Received   ; lpNumberOfBytesReceived
    ,"Int"  , 0)             ; lpOverlapped

 ;MsgBox, RS232_FileHandle=%RS232_FileHandle% `nRead_Result=%Read_Result%
`nBR=%RS232_Bytes_Received% ,`nData=%Data%
 If (Read_Result <> 1)
 {
  MsgBox, There is a problem with Serial Port communication. `nFailed Dll ReadFile on
RS232 COM, result=%Read_Result% - The Script Will Now Exit.
  RS232_Close(RS232_FileHandle)
  Exit
 }

 ;###### Format the received data ######
 ;This loop is necessary because AHK doesn't handle NULL (0x00) characters very nicely.
```

```
  ;Quote from AHK documentation under DllCall:
  ;    "Any binary zero stored in a variable by a function will hide all data to the right
  ;    of the zero; that is, such data cannot be accessed or changed by most commands and
  ;    functions. However, such data can be manipulated by the address and dereference
operators
  ;    (& and *), as well as DllCall itself."
  i = 0
  Data_HEX =
  Loop %RS232_Bytes_Received%
  {
    ;First byte into the Rx FIFO ends up at position 0

    Data_HEX_Temp := NumGet(Data, i, "UChar") ;Convert to HEX byte-by-byte
    StringTrimLeft, Data_HEX_Temp, Data_HEX_Temp, 2 ;Remove the 0x (added by the above
line) from the front

    ;If there is only 1 character then add the leading "0'
    Length := StrLen(Data_HEX_Temp)
    If (Length =1)
      Data_HEX_Temp = 0%Data_HEX_Temp%

    ;i++

    ;Put it all together
    Data_HEX := Data_HEX . Data_HEX_Temp
  }
  ;MsgBox, Read_Result=%Read_Result%
`nRS232_Bytes_Received=%RS232_Bytes_Received% ,`nData_HEX=%Data_HEX%

  SetFormat, Integer, DEC
  Data := Data_HEX

  Return Data
}

;################################################################################
;###### Exit Console Receive Loop ###############################################
;################################################################################
^F1::
{
  Quit_var = 1
}
return

#F2::
```

```
{
 InCalibration = 1
 IfWinNotExist, Untitled - Notepad
 {
  Run, Notepad
  WinWait, Untitled - Notepad
 }
}
return

#F3::
{
 InCalibration = 0
 IfWinExist, Untitled - Notepad
 {
  WinActivate, Untitled - Notepad
  WinClose, Untitled - Notepad
  Sleep,1
  IfWinExist, Notepad
  {
   Send, n
   Send, Enter
  }
 }
}
return

#F5::Calibrate1 := 1
#F6::Calibrate2 := 1
#F7::Calibrate3 := 1
#F8::Calibrate4 := 1

#IfWinActive, Untitled - Notepad
if(InCalibration = 1)
{
 Up::VerticalMove := VerticalMove + 1
 Down::VerticalMove := VerticalMove - 1
 Left::HorizontalMove := HorizontalMove + 1
 Right::HorizontalMove := HorizontalMove - 1
 Home::HorizontalStretch := HorizontalStretch + 1
 End::HorizontalStretch := HorizontalStretch -ter 1
 PgUp::VerticalStretch := VerticalStretch + 1
 PgDn::VerticalStretch := VerticalStretch - 1
 Up Up::
 Down Up::
```

```
  Left Up::
  Right Up::
  Home Up::
  End Up::
  PgUp Up::
  PgDn Up::
  {
    Clipboard = VerticalMove = %VerticalMove%`nHorizontalMove =
%HorizontalMove%`nVerticalStretch = %VerticalStretch%`nHorizontalStretch =
%HorizontalStretch%`n
    Send, ^a
    Send, ^v
  }
  return
}
```