Embedded Vision Control System

CE3910 Embedded III - Prof. Barnekow

Curt Henrichs
Spring 2017

Submitted May 10, 2017

Abstract

The design laboratory project for the Embedded III course is an embedded computer vision system using a digital camera attached to a DE0 development board with a NIOS II softcore processor image. The target object is a black object on a white background that will move greater than one inch per second across the cameras field of view. The system tracks the target with a two axis, servo actuated turret to position the center of the target into the center of the camera's field of view. Additionally, the camera image is displayed onto a VGA monitor so the user has feedback onto what the camera detects. The purpose of this design project is to act as a capstone project for the previous embedded courses. Also this project exposes new hardware devices that are common in embedded systems such as I^2C and graphics buffers. Commands entered by the user makeup the functionality of the system and were used to mark progress milestones for the project. Commands supported follow either instant processing format such as read and writing from memory or they follow a polling loop pattern. When a user commands the system to track the object the command will continue running until user presses the exit button. All code was written in embedded C for the NIOS processor and made use of a data structure to simplify command creation. Testing of the system shows that all requirements are met for the tracking application. Future work could be performed to tune the pixel filtering but may not increase performance from the user's perspective.

Table of Contents

Introduction

Overview

The laboratory project for the Embedded III course is a computer vision tracking system using a NIOS II softcore processor on a DE0 development board. A digital camera is mounted to a two axis servo actuated turret providing pan and tilt control to the microcontroller. The microcontroller automatically tracks a dark object on a white background at a rate greater than one inch per second. Additionally, the user directs the behavior of the microcontroller through asynchronous serial communication connected between the controller and the user's terminal. Commands exposed to the user are read/write microcontroller memory, read/write camera registers, pan/tilt control, image capture, target tracking, and command help information. Each command will check for correct parameters and perform the appropriate action when requested. Finally, the hardware connection will take the form shown in Figure 1 and is supplied from EECS tech support or previous courses.

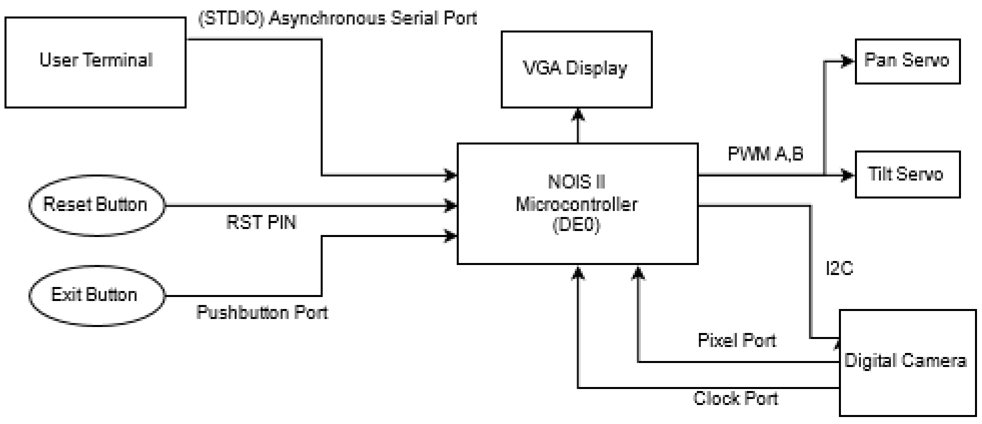

Figure 1: General diagram of system hardware showing the input user interface on the left and application hardware on the right. Note that digital camera I^{2}C bus is both input and output for the microcontroller.

Design

User Interface

The user interface for the digital camera project consists of an asynchronous serial connection between user's terminal and the cameras microcontroller. The specific connection used for this project is the built in standard IO stream provided by the NIOS II image. Commands are entered by the user as a single line string. Each line captured using fgets() function is compared against the list of valid commands, as shown in the appendix. On the match of command name, the program will invoke the correct behavior function for further parsing and state change.

All commands available are implemented using a data structure containing the name string, description string, and command function pointer. All functions are referenced in a list which is linearly searched when a new command is requested by the user. When a match occurs the search loop calls the matched command's function pointer. Each command function is required to have an end condition so that the processor can resume user input entry.

Read and write functions are implemented such that only a small time delay is between start of requested command and polling for next command. Pan and tilt commands operate with a similar time delay. The deviation in the instant processing approach is in the image and track commands. Both of these use a loop that operates until a physical button is pressed. When the button is pressed the function returns control to the user input loop.

Camera Interface

The digital camera is attached to the microcontroller through two 8-bit registers and an I^2C bus. The registers are for pixel data and camera clock data. Pixel data is sent in monochrome QCIF format to reduce processing overhead. Camera clock data is used to synchronize the pixel data to a specific frame as covered in progress milestones three and four. The I^2C bus is used to access the control registers for the camera as specified in its datasheet. I^2C communication is a two wire synchronous serial communication protocol that relies on sending a device address followed by the appropriate data. The benefit of using two wires and defining an address is that it reduces the total number of wires routed to communicate. Another benefit of I^2C, which is not utilized in this project, is the ability for multi-master operation. Thus the camera peripheral could be managed by multiple controllers if so desired.

Tracking

Tracking and imaging commands rely on valid frame capture as described in the camera interface and related milestones. Both commands make use of the VGA display to either display the filtered image or monochrome image respectively. Finally, both are implemented as a polling loop checking for a physical button press before returning to the UI main loop.

Image capture is written as a frame capture function that does not apply filtering to the incoming pixel data. Additionally, no feedback data is needed as no control is applied to the turret.

Tracking is written as a several step process with steps: pixel capture, filtering, object location detection, servo position update, and auto light level adjustment. Pixel capture and filtering is described with Figure 6 in milestone four. Each pixel is sent through a low-filter as described in milestone six which improves the accuracy of object detection. Object detection is applied through a bounding box which is based on the minimum and maximum coordinates of valid pixels. The center of the bounding box is then considered the center of the object being tracked. Given that the desired center of screen as the set point, current error can be computed in the x and y coordinates which are then feed into a PD controller. The controller has a derivative term in order to decrease the time to settle making the camera stable sooner. No integral term is used as the steady state error is within an acceptable range. Finally, the maximum and minimum light levels are recorded to adjust the automatic filters applied to the next frame.

Progress Summary

Weekly milestones were used to enforce a deadline to the project's features. Each milestone defines the overall design objectives in more detail. Each milestone's design section details the implementation of requirements. Finally, when additional work was performed that was unrelated to the current milestone it was noted in its own section.

Milestone 1

Requirements

Develop a prototype of the command entry system for user control of the product. Commands for reading and writing to memory are to be implemented. By developing this prototype, the project will have a definite starting point and thus establish a testing point for correct hardware operation. The NIOS image will be deployed to the DE0 development board which will prove the image to be functional. By deploying the project onto NIOS the toolchain used will be proven functional. Finally, the commands implemented will be useful for debugging the product as it develops.

A command for reading memory is to be implemented. The read command will follow the form of,

RR <address> and **RR <address> <count>**

Where the address will be entered in hexadecimal and the optional count value will be entered in hexadecimal. The count parameter is used to specify the number of bytes printed starting at the address specified. The count will default to one when not supplied. Reading from memory will bypass the cache to get the most recent IO data. The output will take the form of,

```
  Base    : +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +a +b +c +d +e +f
00000000 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Line one header will be printed only to +f and not exceed the count specified if less than sixteen. The following lines will start with the base address which will be either the supplied address or its offset by a multiple of sixteen. Each byte to be printed will be in hexadecimal along with the base address being printed in hexadecimal.

A command to write to memory is to be implemented. The write command will take the form of,

WR <address> <value>

Where the address specified is in hexadecimal and the value specified is in hexadecimal. The value must be specified and is expected to be in hexadecimal. The value is a byte of information and larger inputs will only use the first byte supplied. This command will always bypass the cache so that memory mapped IO will be updated with less delay.

A command to lookup information about commands is to be implemented. This command will be labeled the help command and will take the form of,

HELP or **HELP <command>** or **HELP ALL**

Where specifying just the command will pull up help information on this command. Specifying ALL will print the help information for all commands available to the user. Specifying a single command will print the information on the command given.

Design

Implementation of the commands follow a common data structure shown in the Figure 2. This structure contains a name of the command (used for string comparison against the user input), a string detailing the usage of the command for both documentation and for the help menu being developed, and also a function pointer which is invoked by the main loop on the structure with the name desired. The main reason for the function pointer is to invoke the behavior of the command without naming the command directly thus giving a polymorphic effect to the data structure and also to allow for list based approaches to valid commands.

```
/**
 * Defines a command object that has a name, help information, and commanded
 * function pointer.
 */
typedef struct command{
      char* name;
      char* helpInfo;
      void (*cmdFnt)(char*);
}command_t;

/**
 * Stores all commands in a list for polymorphic invocation.
 */
static command_t* commandList[NUMBER_OF_COMMANDS];

/*
 * Use Case: Searching for a command given user input.
 */
int i;
for(i = 0; i < NUMBER_OF_COMMANDS;++i){
      if(strcmp(cmdStr,commandList[i]->name)==0){
            commandList[i]->cmdFnt(inputStr);
            commandFoundFlag = 1;
      }
}
```

Figure 2: Command data structure developed to define relevant features of all commands for more polymorphic command invocation.

Additional Work

In addition to the work required for this milestone, a port of applicable code modules from the embedded two course was performed. Code modules that were ported are, INTERUPT_TABLE, GENRIC_IO, LCD, LED, PUSHBUTTON, PWM, TIMER, and UART modules. All modules were tested in a driver program on the NIOS image for embedded three also comments were modified to fit the Doxygen format. Modules that were not ported include ADC, DC_MOTOR_CONTROL, KEYPAD, and SLIDER_SWITCHES. By porting these modules, the code development should faster as basic interfacing is already complete on various hardware devices. The goal of these modules is to construct higher level functions that are not dependent on specific hardware. Note that not all ported modules must be used but rather are supplied for possible use.

Test Plan

The following is the test plan used to prove correct operation of this prototype command system.

1. Test write command at address 0x00802030 (LEDs) with byte value of 0xFF.
2. Test write command at NULL (0x00000000) with any value. Program should ideally not crash but may allow write to occur.
3. Test read command at NULL check for invalid behavior. Expecting to see change from write command however this is an odd case where result is a don’t care condition.

4. Test read command at 0x00802030. Expecting to get result of 0xFF. Write to 0x00802030 with 0x00 and read expecting 0x00.
5. Test read with multiple bytes at 0x00802030, specifically 0x20 bytes need to read.
6. Test read with count of zero inputted, expect error message back to user.
7. Test read with negative number of bytes. Expecting to print entire memory out. Acceptable to quit early as real use would not expect this. Should not crash the program.
8. Test write command with no value, expecting error message back to user.
9. Test write command with no address specified, expecting error message back to user.
10. Test read command with no address specified, expecting error message back to user.
11. Test read command at address 0xFFFFFFFF and count of both 1 and 2. Expect addressing to roll over. Expect also a warning for overflow of address.
12. Test help command with no parameters. Expect to see help commands help information.
13. Test help command with ALL as parameter. Expect to see all commands listed with their help information.
14. Test help command with WR, RR, HELP as parameter separately. Expect to see each respective help information.
15. Test help command with too many arguments. Expect help command to give error message back to user. (Note: Does not give back error message as it ignores other stuff in the string past parsing. This is determined to be acceptable behavior.)
16. Test help command with invalid command. Expect an error and the help information of help command to be shown.
17. Test program by requesting a nonexistent command. Expect error message back to user.

Milestone 2

Requirements

Develop modularized servo motor control using the previously built PWM module. In addition, two commands will be developed to control the absolute positon of each servo motor from the command line. Additional functions will be stubbed out for later camera tracking application.

A command for controlling the absolute position of the pan servo is to be implemented as shown,

PAN <column>

Where the column parameter is based off of the QCIF resolution of 144 x 176. Thus the range of the servo will extend from 0 to 175 for angles between 0° and 180° of rotation.

A command for controlling the absolute position of the tilt servo is to be implemented as shown,

TILT <row>

Where the row parameter ranges between 0 and 143 for the restricted rotation range of the tilt servo. This restriction is due to physical construction of the turret. Iterative testing of angle range was developed using OCR values, see design section.

Final requirement, when the firmware starts, it should orient the servo motors to their center position and then wait for further instruction from the user.

Design

Implementation of the commands use the command data structure developed in the previous milestone making the high-level code trivial to implement. However, the hardware interface was the main focus of this milestone which meant using the PWM driver developed in embedded two as a template for the servo motor control. The first step taken in development was to identify the hardware limits in relation to the OCR (Output Compare Register) value inputted. Using the standard 5% to 10% duty cycle for servo motor control achieved most of the range expected but was not precise enough for full range. Therefore, through incremental testing of OCR values the maximum, center, and minimum positions are determined. By mapping the maximum, center, and minimum positions to the QCIF resolution of 176 x 144 the input can be plotted to desired output as shown in Figure 3 for pan servo and Figure 4 for tilt servo. Note that slight nonlinearity exists around the center position commanded but since it has minimal effect on the final position each plot can be linearly approximated. Theses linear equations translate from user input into OCR output on the PWM module. Issues that arise by naively truncating the floating point values is reduced range and imprecision. While it may seem that floating point operations are more desirable than integer math it comes at the cost of computation speed. Thus the solution is to save two decimal points of precision, by scaling the coefficient by one hundred, for all calculations until the end where the value is then divided by one hundred and decimal points are truncated.

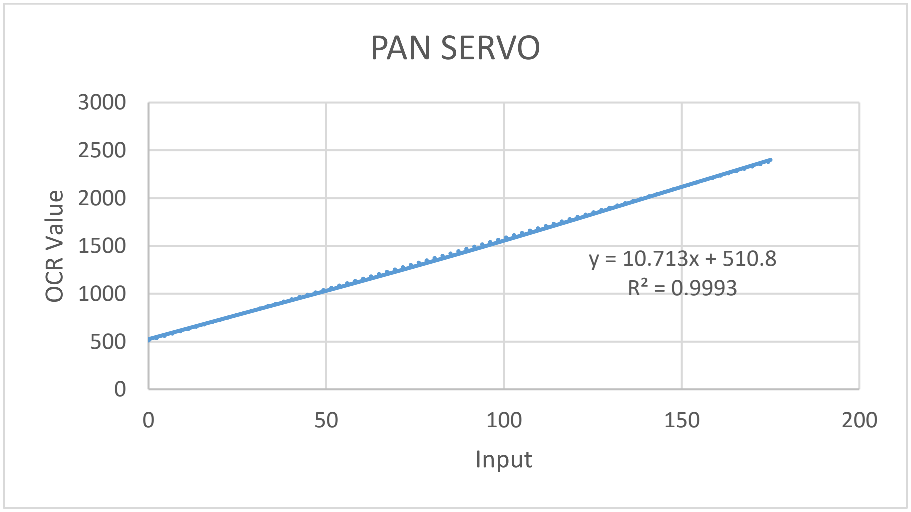

Figure 3: OCR Value versus Input for full range of pan servo motor.

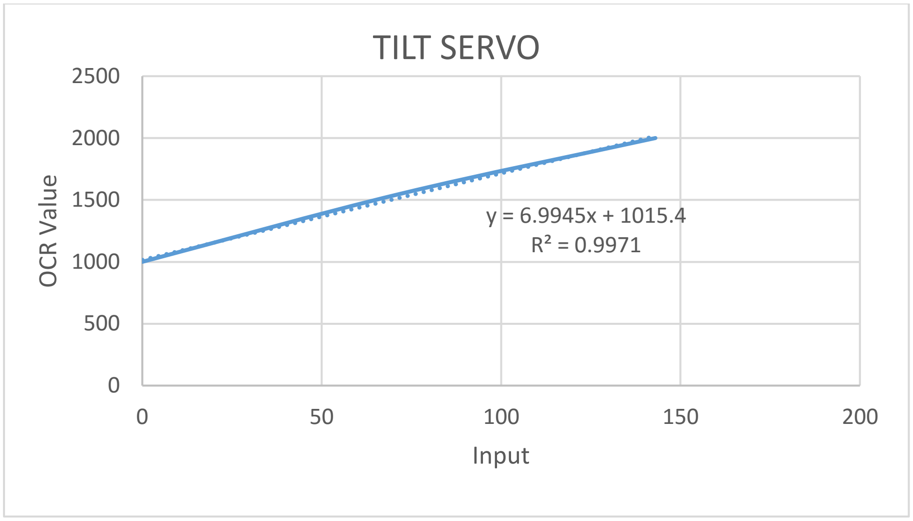

Figure 4: OCR Value versus Input for full range of tilt servo motor.

Test Plan

The following is the test plan used to prove correct operation of servo turret system commands.

1. Test pan command with parameter of 0, expect turret to pan to minimum position.
2. Test pan command with parameter of 175, expect turret to pan to maximum position.
3. Test pan command with parameter of 88, expect turret to pan to center position.
4. Test pan command with parameter of -1, expect error message back to user for invalid input.
5. Test pan command with parameter of 176, expect error message back to user for invalid input.
6. Test tilt command with parameter of 0, expect turret to tilt to minimum position.
7. Test tilt command with parameter of 143, expect turret to tilt to maximum position.
8. Test tilt command with parameter of 72, expect turret to tilt to center position.
9. Test tilt command with parameter of -1, expect error message back to user for invalid input.
10. Test tilt command with parameter of 144, expect error message back to user for invalid input.
11. Test help command with PAN as parameter, expect to see pan help information.
12. Test help command with TILT as parameter, expect to see tilt help information.
13. Test help command with ALL as parameter, expect tilt and pan commands to show up in listed commands.

Milestone 3

Requirements

Requirement for this milestone is to develop an I^{2}C software module to provide read and write capability to slave devices on the I^{2}C network. The only device physically connected is the camera controller which will be used for verification of the software module's behavior. Two commands will be implemented that makes use of this modules as shown below.

An instruction to read from the I^{2}C network is to be developed with the following command structure,

RDCAMREG <register number>

The command will use the camera's address and will use the parameter as the 8-bit register address for the slave device. The value inputted from the user and the value returned to the user will be in hexadecimal.

An instruction to write to the I^2C network is to developed with the following command structure,

WRCAMREG <register number> <value>

Where register number is the internal register address of the slave device and the value is the 8-bit value to be written to that register. All values inputted will be in hexadecimal.

Design

Lecture material covered the basic operations of the hardware I^2C module supplied in the NIOS image which lead to initial development that followed the required write address, write register pattern for both writes and reads followed by writing data for a write command and performing a stop and restart of a read from the device for the read command. The camera hardware documentation is used to determine registers for testing of the device along with the data expected by camera.

Several registers were selected for either to further develop the digital camera system. The manufacturer ID is the first set of registers used by the project to prove correct read operation. The register numbers are 0x1C and 0x1D. When read, the returned values are 0x7F and 0xA2 respectively. Second, a control register, numbered 0x11, allows for manipulation of the pixel clock speed to slow it down. The initial problem with the camera that makes this register useful is that the camera's pixel clock frequency is too high for the CPU frequency of the NIOS image to reliably capture all pixels. So by using the provide formula in eq. 1 a valid speed can be calculated for the pixel clock. A final value of 0x0F is chosen to provide some time for the algorithm to do filtering.

$$Freq = \left(\frac{17.7MHz}{(value + 1) * 4}\right) \tag{1}$$

Control register 0x39 was selected as it has a control bit that toggles on a pixel clock, Pclk, feature where horizontal reference, Href, must be high to clock out data. This means that the pixel clock will not be free running but rather only will clock when actual data is being sent by the camera. By doing this it will simplify the pixel data collection as a check of high state on pixel clock is all that is needed to know that data is valid. Figure 5 shows the oscilloscope recording both the Href and Pclk signals. Note that the Pclk is shown to only go high when Href is high thereby verifying the I^2C write process. Finally, control register 0x14 is written to, in order to convert the image from CIF to QCIF format which reduces the number of pixels sent out thereby reducing the time to find the target.

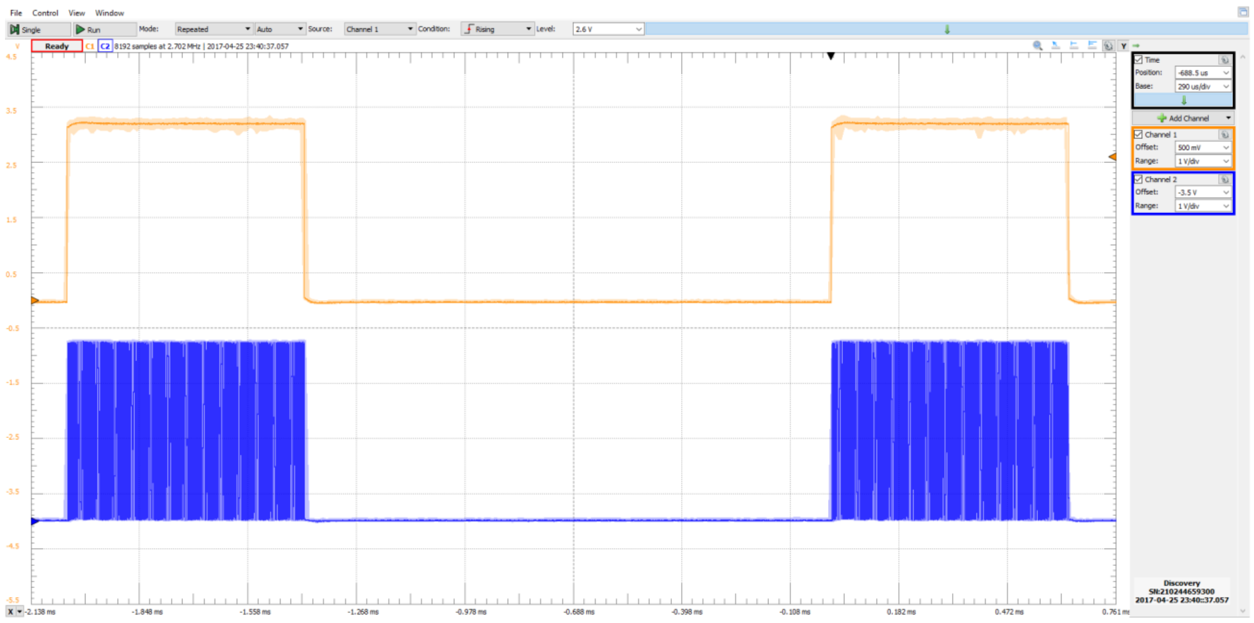
Figure 5: Oscilloscope showing both Href and Pclk signals. This plot proves the correct operation of the I²C write function with expected result of Pclk only being active when Href is high.

Test Plan

The following is the test plan used to prove correct operation of the I²C module.

1. Test RDCAMREG command with register 0x1C and 0x1D which expects constant data 0x7F and 0xA2 back to the user as the manufacturer ID.
2. Test WRCAMREG command with register 0x11 to slow down the pixel clock. Supply a value that follows eq.1 to get a desired clock frequency. Tested by using oscilloscope to detect frequency change.
3. Test WRCAMREG command with register 0x39 to make the Pclk generate only when Href is high. Tested by using oscilloscope to show no generation when Href is low. Write value 0x40 to turn on this feature.
4. Test HELP command with RDCAMREG, expecting to see help information.
5. Test HELP command with WRCAMREG, expecting to see help information.
6. Test HELP command with ALL, expecting to see RDCAMREG and WRCAMREG to show up in list.
7. Test RDCAMREG without register specified, expecting to get an error message from console to warn user.
8. Test WRCAMREG without data specified, expecting to see error message.
9. Test WECAMREG without register number specified, expecting to see error message.

Milestone 4

Requirements

Develop an algorithm to read pixel data from the camera and then display the resulting image onto a display connected through VGA port. Additionally, a command should be implemented to display a video feed of the camera until a pushbutton is pressed. The form of this command is,

IMAGE

Where no parameters are needed to operate and the end condition is when the user presses button two on the DE0 board. When command ends the screen should be cleared.

Also basic tracking should be prototyped by using a minimum brightness threshold to determine the target of interest. A closed loop controller will be developed to take the target position and move the image to the center of the screen. The command that a user will input to cause this behavior will be of form,

TRACK

Where no parameters are supplied in this prototype command and the end condition is when pushbutton two on the DE0 is pressed. Finally, when command ends the screen should be cleared and the servo motors repositioned to home.

Design

The algorithm to implement the pixel data collection is shown in Figure 6. This flowchart shows the steps necessary to get a frame of pixels into memory, specifically the VGA memory which acts as a buffer for further processing and as video buffer for the DMA controller. The data acquisition is performed using polling as the time cost of switching for interrupts is too high. Note, the frame rate is significantly reduced to provide time to sample and process data but still high enough to be performant for tracking, thus some error is expected in pixel data. Figure 7 shows a sample image generated by the camera when using the IMAGE command. The horizontal distortion is caused by vibration of the servo motors as they keep positon. This should not be a problem as shown in the second section of this design where the turret still tracks correctly. The object shown is the author's hand at approximately one foot away from the camera to show a sense of scale with an easily identifiable object given the resolution.

A major concern of this algorithm is the frame rate of the camera being too low for adequate tracking of the object. Testing is needed to determine if lower quality data is acceptable when the frame rate is increased. Another concern is speed of the pixel collection when more complex filters are used to determine the object. Again more testing is needed to determine the limits of the system.

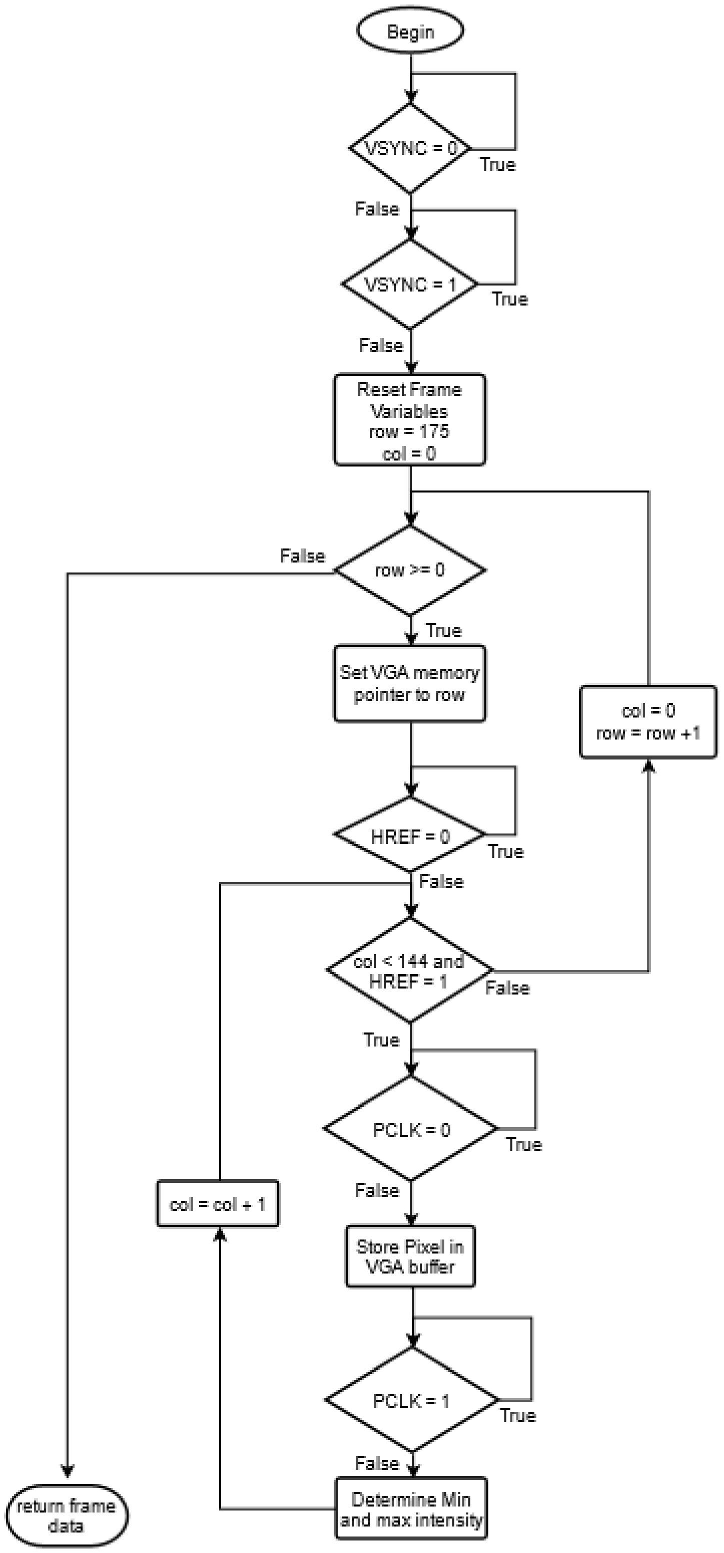

Figure 6: Flowchart for pixel data input collection. First step is to wait until VSYNC pulse passes, signaling that a new frame starts. For each row, wait for HREF to equal one, then clock in the pixel data using PCLK. Returns data structure for frame metadata and VGA memory as pixel data.

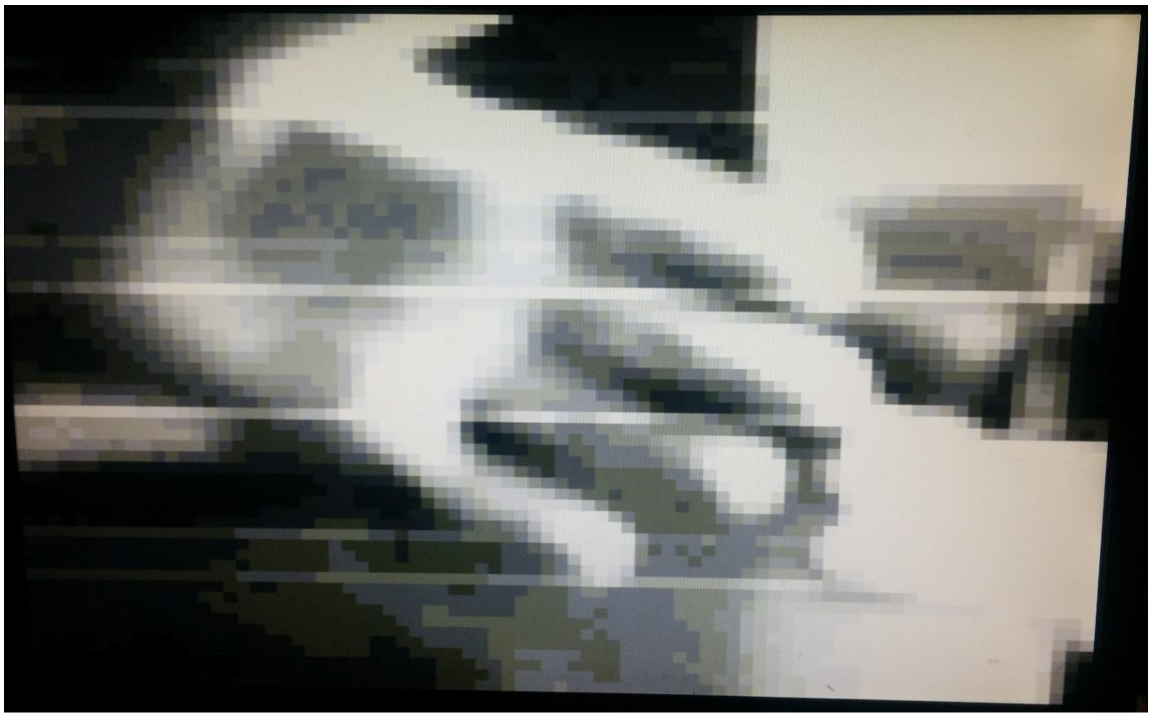

Figure 7: Sample image taken by camera. Note that the author's hand is the object being shown approximately a foot away from the camera. The bright horizontal lines appear to be due to the servo motor vibrating.

A prototype tracking system is implemented by following the block diagram shown in Figure 8. The controller design uses filtered data from the image capture algorithm discussed above. Where the filtering is a brightness threshold such that only pixels above a certain value are stored as 0xFF and less than that value is 0x00. This image is displayed on the VGA display similar to the camera video stream. This image is then subject to a bounding box calculation to determine the size of the object being tracked. Using the bounding box values to calculate the center of the target which is then compared to the desired center of the screen (40,30). The error is then sent through a PD controller, note the derivative term is just the slope of previous to current error. The generated signal is then sent to the servo motor for each axis of rotation.

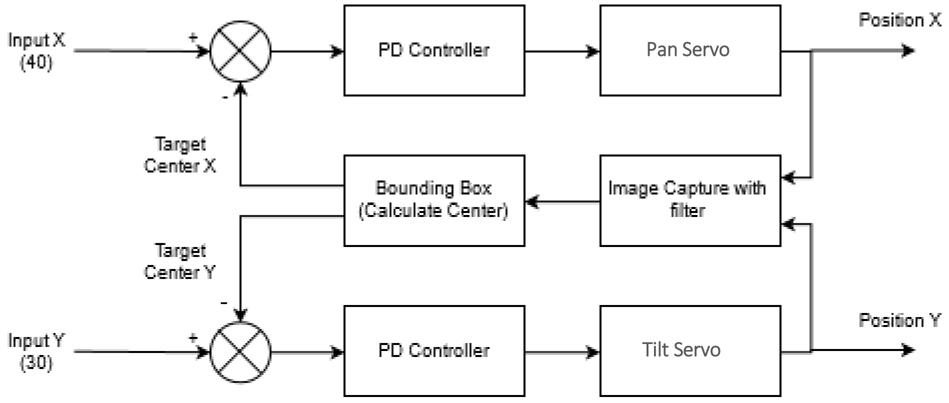

Figure 8: Block diagram for tracking control system. Note that each servo has its own controller values.

The current PD controller has P = 1/4 and D = 0 for panning and P = 3/8 and D = 1/8 for tilting. The response of the servo motor is a slightly underdamped system which can be improved with later iterations. Note that there is not a derivative term in the x direction due to difference in error scaling. Figure 9 shows a target with desired threshold shown on the VGA display.

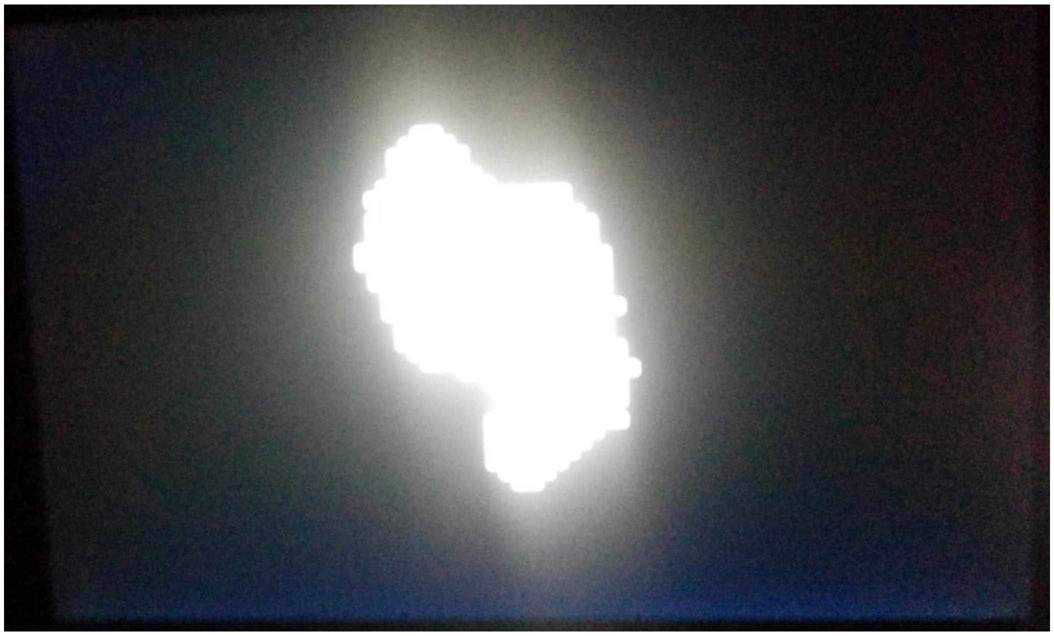

Figure 9: Sample image of target being tracked by system. Note this is taken in the dark and thus saturation from the camera taking the picture of the display is causing the distortion.

Additional Work

Additional work was performed on the command line interface to improve code readability. The command line code existed in a single source and header file which became unmanageable for the larger set of commands. Thus commands were broken up into memory, servo, and camera commands where each type has its own header and source file. Additionally, the command object definition was given its own header file. Polymorphic behavior is still maintained and the command line source code still generates the list correctly. Note that the help function will be maintained in the command line file as help is depended on the list generated of all commands.

Test Plan

The following is the test plan for the image command.

1. Entering IMAGE produces a video stream on the display connected through the VGA port.
2. Image displayed is correctly oriented and scaled for QCIF with every other pixel used.
3. Loss of pixel data due to sample errors should be minimal.
4. Pressing pushbutton two causes the video loop to end and the command line to start again.
5. End of command causes the VGA display to be cleared.
6. Help command shows that image is successfully in the list and has accurate information.

The following is the test plan for the tracking command.

1. Entering TRACK produces a video stream on the display connected through the VGA port.

2. Image displayed is either black or white and the white pixels are approximately the location of the object being tracked. (Relative brightness above a certain value is being tracked.)
3. Moving the target object causes the turret to move in the correct direction to track the object and the position of target on screen approaches the center of the screen.
4. Servo motors should stay in positon when no object is detected.
5. Pressing pushbutton two causes the tracking to cease and servo motors return to home position.
6. End of command causes the VGA display to be cleared.
7. Help command will show that track is a valid command with accurate information.

Milestone 5

Requirements

In a previous milestone prototype tracking was developed that would target an object above an arbitrary light threshold. In this milestone the goal is to further develop the tracking algorithm to track a dark object, light object, or a specific valued object. Furthermore, tracking dark and light objects should have both a manual threshold adjustment mechanism and an automatic threshold mechanism. Automatic threshold adjustment should be based on the light levels detected in the image and account for a small variance of light levels between images.

The new command structure that will replace the previous TRACK command is as follows,

TRACK <type> <value>

Where type is a required field with values of LIGHT, DARK, and TARGET such that LIGHT tracks objects above a threshold value, DARK tracks objects below a certain threshold, and TARGET tracks objects around some threshold. The LIGHT and DARK parameter have an optional value parameter that when given, the threshold is manually set to the value else value is automatically determined from the image. TARGET requires the value parameter to specify the value being targeted.

Design

Implementation of the requirements meant changing the threshold check applied to each pixel in the frame capture function. Additionally, the tracking command needed a parsing structure and a mechanism to control filter behavior. An enum was used for communication between the parsing function and the tracking loop to switch on the correct behavior of each filter. Filtering is applied to each pixel as a bound check. Thus from the capture frame function's perspective only the minimum and maximum values of a range is accepted to adequately determine the valid pixels. Filtering is handled by determining the range of acceptable pixel values which is related to the enum value selected. Automatic filtering has an additional step of recording the max light and dark values of an image and using those to dynamically set the tracking range.

Test Plan

The following is the test plan used to verify correct operation of the tracking command.

1. Entering TRACK LIGHT 0xE0 should track light values above 0xE0 in the frame. No adjustment of the threshold should occur. 0xE0 is close to saturation and thus best tested in a dark environment with a single light source to be tracked.

2. Entering TRACK DARK 0x20 should track dark values below 0x20 in the frame. No adjustment of the threshold should occur. 0x20 is close to the darkest value of the environment thus a well saturated environment is best to track a dark object.
3. Entering TRACK TARGET 0xAA should track the value of 0xAA in the environment. 0xAA is lighter value and thus a well light environment is needed to test this command.
4. Entering TRACK LIGHT should track light values in the environment using automatic adjustment. Thus various light levels in the environment are needed to test automatic adjustment.
5. Entering TRACK DARK should track dark values in the environment using automatic adjustment. Thus various light levels in the environment are needed to test automatic adjustment.
6. Entering TRACK TARGET should return an error message back to the user for invalid command structure.
7. Entering TRACK should return an error message back to the user for invalid command structure.
8. Entering any other value as the type parameter should return an error message back to the user for invalid command structure.
9. Check HELP command with both the TRACK input and the ALL input to verify correct information in the internal documentation.
10. Tracking control system functions as well or better than previous milestone implementations.

Milestone 6

Requirements

During testing of the tracking algorithm it was determined that several minor adjustments were needed. First a need to attenuate small command signals to the servo motors so as to stabilize the camera's positon. Second a need to filter the frame being processed so that the object's center would not be as affected by noise. Note that no change to the track command structure is made.

Design

The attenuation of command signals to the servo motors was implemented as a conditional statement that required the commanded relative position to be greater than some tuned threshold value. This simple change prevents the servo system from hunting for the exact position thereby making the camera stable.

The filter applied to each frame is a row based low-pass filter where the pixel intensity must meet certain criteria to be considered valid. Figure 10 shows the filter algorithm as described. The first requirement for a valid pixel is that it must be within the valid value range. When a pixel is valid it gets a minimum confidence value plus the value of the previous pixel. So after several pixels the object will accumulate towards high confidence. However, when a pixel does not meet minimum threshold values it does not decay back down but rather has an instantaneous attenuation to zero. This lack of symmetry prevents the filter from shifting the object past its original boundary. In short, the filter implemented is dependent on ordering of the columns as they are processed for each row.

```
if(pixel >= thresMin && pixel <= thresMax)
{
      vpixel = MIN_CONFIDENCE_MATCH + prevPixel;
      prevPixel = *vgaPtr = (vpixel > 0x00FF)? 0xFF : (uint8_t)(vpixel);
}
else
{
      vpixel = (prevPixel >= MIN_CONFIDENCE_MATCH)?
               (prevPixel - MIN_CONFIDENCE_MATCH) : 0;
      prevPixel = *vgaPtr = (uint8_t)(vpixel);
}
```

Figure 10: Code implemented for the low-pass row filter with integer confidence accumulation.

Test Plan

The following is the test plan to prove correct operation of the code developed for this milestone.

1. Test the tracking of an object with the new threshold by observing a dead band between object movement and servo movement.
2. Test threshold by holding the object still, servo should come to complete stop as opposed to hunting for a closer center positon.
3. Test filtering by displaying filtered data on VGA monitor and show that the servo still hits center positon.
4. Test filtering by showing a lack of noise in the object when tracking relative to the camera image.

Conclusion

For the embedded computer vision, digital camera project a test plan was developed to prove compliance with stated requirements. The test plan can be found in the appendix and is broken down by user command. The results of the test plan show compliance with the general requirements of developing a tracking turret system using a NIOS II softcore processor along with required camera hardware. Additionally, testing shows correct operation of each command for valid input and invalid input. Thus the design is validated and meets the overall requirements. The design for this project did encounter some roadblocks and improvements are still recommended.

The main problem encountered while implementing the project was managing the camera data. Several milestones were dedicated to improving the frame data collection and filtering so that the object being tracked was more likely the correct object. Specifically, attempting to filter noise out of the object data was needed and took a milestone of progress to complete. The filter chosen was a low-pass column filter that looked at the previous column in a given row. Using this filter decreased the size of the bounding box as noisy pixel data was not assumed to be part of the target object. However, when applying the filter, it forced a reduction in frame rate as the image would lose more pixel data to time delay.

Future improvements would include an attempt at further optimization of the frame collection code to the point of writing it in assembly. This could increase the speed of processing allowing for an increase in frame rate. A second optimization could be on further tuning of the low-pass filter to increase detection of the object. Another approach would be to develop a column and row filter to further cut out light noise. Finally tuning the automatic light threshold could prevent some errors in tracking when moving in non-uniform lighting. All of these suggestions would involve further tuning and refinement of the current system which as noted already performs as expected. Thus further improvements may not necessarily increase the experience of the system from the user's perspective.

Appendix

Command List

Command	Description
WR <address> <value>	Writes to selected memory at the given start address. Writes a single byte of data to specified location.
RD <address> <count>	Reads from selected memory at the given start address. Outputs number of bytes specified else if not specified then a single byte printed.
PAN <position>	Absolute position control of the pan servo. Valid input is between 0 and 175.
TILT <position>	Absolute position control of the tilt servo. Valid input is between 0 and 144.
WRCAMREG <register> <value>	Writes to camera I^2C register specified as parameters.
RDCAMREG <register>	Reads from camera I^2C register specified as parameter.
IMAGE	Displays image from camera onto VGA display.
TRACK <filter type> <value>	Tracks an object in view of camera with servo turret. Displays filtered image onto the VGA display. The type of filter is specified to be DARK, LIGHT, or TARGET. Value sets threshold for manual mode, else automatic adjustment is used.
HELP <command>	Displays information about a command on the serial terminal. Can specify name of command or ALL to get more information.

Table 1: List of all commands for the camera system along with a brief description on the commands behavior.

Test Plan

The following is the complete test plan for all command of the digital camera system. Each test plan has been verified in the milestone that it was developed and a final test was performed for the final build of the project.

General

18. Test program by requesting a nonexistent command. Expect error message back to user.

WR Command

19. Test write command at address 0x00802030 (LEDs) with byte value of 0xFF.
20. Test write command with no value, expecting error message back to user.
21. Test write command with no address specified, expecting error message back to user.

RD Command

22. Test read command at NULL check for invalid behavior. Expecting to see change from write command however this is an odd case where result is a don't care.
23. Test read command at 0x00802030. Expecting to get result of 0xFF. Write to 0x00802030 with 0x00 and read expecting 0x00.
24. Test read with multiple bytes at 0x00802030, specifically 0x20 bytes need to read.
25. Test read with count of zero inputted, expect error message back to user.
26. Test read with negative number of bytes. Expecting to print entire memory out. Acceptable to quit early as real use would not expect this. Should not crash the program.
27. Test read command with no address specified, expecting error message back to user.

28. Test read command at address 0xFFFFFFFF and count of both 1 and 2. Expect addressing to roll over. Expect also a warning for overflow of address.

PAN Command

14. Test pan command with parameter of 0, expect turret to pan to minimum position.
15. Test pan command with parameter of 175, expect turret to pan to maximum position.
16. Test pan command with parameter of 88, expect turret to pan to center position.
17. Test pan command with parameter of -1, expect error message back to user for invalid input.
18. Test pan command with parameter of 176, expect error message back to user for invalid input.

TILT Command

1. Test tilt command with parameter of 0, expect turret to tilt to minimum position.
2. Test tilt command with parameter of 143, expect turret to tilt to maximum position.
3. Test tilt command with parameter of 72, expect turret to tilt to center position.
4. Test tilt command with parameter of -1, expect error message back to user for invalid input.
5. Test tilt command with parameter of 144, expect error message back to user for invalid input.

WRCAMREG Command

10. Test WRCAMREG command with register 0x11 to slow down the pixel clock. Supply a value that follows eq. 1 to get a desired clock frequency. Tested by using oscilloscope to detect frequency change.
11. Test WRCAMREG command with register 0x39 to make the Pclk generate only when Href is high. Tested by using oscilloscope to show no generation when Href is low. Write value 0x40 to turn on this feature.
12. Test WRCAMREG without data specified, expecting to see error message.
13. Test WECAMREG without register number specified, expecting to see error message.

RGCAMREG Command

1. Test RDCAMREG command with register 0x1C and 0x1D which expects constant data 0x7F and 0xA2 back to the user as the manufacturer ID.
2. Test RDCAMREG without register specified, expecting to get an error message from console to warn user.

IMAGE Command

7. Entering IMAGE produces a video stream on the display connected through the VGA port.
8. Image displayed is correctly oriented and scaled for QCIF with every other pixel used.
9. Loss of pixel data due to sample errors should be minimal.
10. Pressing pushbutton two causes the video loop to end and the command line to start again.
11. End of command causes the VGA display to be cleared.

TRACK Command

8. Entering TRACK command produces a video stream on the display connected through the VGA port.
9. Image displayed is either black or white and the white pixels are approximately the location of the object being tracked.

10. Moving the target object causes the turret to move in the correct direction to track the object and the position of target on screen approaches the center of the screen.
11. Servo motors should stay in positon when no object is detected.
12. Pressing pushbutton two causes the tracking to cease and servo motors return to home position.
13. End of command causes the VGA display to be cleared.
14. Entering TRACK LIGHT 0xE0 should track light values above 0xE0 in the frame. No adjustment of the threshold should occur. 0xE0 is close to saturation and thus best tested in a dark environment with a single light source to be tracked.
15. Entering TRACK DARK 0x20 should track dark values below 0x20 in the frame. No adjustment of the threshold should occur. 0x20 is close to the darkest value of the environment thus a well saturated environment is best to track a dark object.
16. Entering TRACK TARGET 0xAA should track the value of 0xAA in the environment. 0xAA is lighter value and thus a well light environment is needed to test this command.
17. Entering TRACK LIGHT should track light values in the environment using automatic adjustment. Thus various light levels in the environment are needed to test automatic adjustment.
18. Entering TRACK DARK should track dark values in the environment using automatic adjustment. Thus various light levels in the environment are needed to test automatic adjustment.
19. Entering TRACK TARGET should return an error message back to the user for invalid command structure.
20. Entering TRACK should return an error message back to the user for invalid command structure.
21. Entering any other value as the type parameter should return an error message back to the user for invalid command structure.
22. Test the tracking of an object with the new threshold by observing a dead band between object movement and servo movement.
23. Test threshold by holding the object still, servo should come to complete stop as opposed to hunting for a closer center positon.
24. Test filtering by displaying filtered data on VGA monitor and show that the servo still hits center positon.
25. Test filtering by showing a lack of noise in the object when tracking relative to the camera image.

HELP Command

1. Test help command with no parameters. Expect to see help commands help information.
2. Test help command with ALL as parameter. Expect to see all commands listed with their help information.
3. Test help command with each named command as listed above.

Code Listing

Main

main.c

```
/**
 * Milestone 3
 * @author Curt Henrichs
 * @version 0.2
 * @date 3-22-17
 * @file main.c
 *
 * Milestone 1: Command Entry System. Commands entered through stdio device and
 * commands are parsed to carry out various operations. Read and write to
 * memory is required.
 *
 * Milestone 2: Develop servo motor module and related commands for the command
 * structure developed in milestone 1.
 *
 * Milestone 3: Develop I2C camera control module. Develop camera control
 * module and VGA module.
 *
 * Commands:
 *    RD <address> reads contest of single location. Returns byte
 *    RD <address> <count> Reads from address to address + count.
 *    WR <address> <value> Writes data to location address. Only a byte used.
 *    PAN <position> moves pan servo to desired absolute position.
 *    TILT <position> moves tilt servo to desired absolute position.
 *    RDCAMREG <Register> reads byte from the camera I2C device.
 *    WRCAMREG <Register> <Value> writes byte to camera I2C device.
 *    IMAGE displays the camera data onto the VGA display
 *    TRACK <type> <value> tracks object matching filter type and threshold
 */

//==============================================================================
//                                  Libraries
//==============================================================================

#include "CommandLine.h"
#include "Hardware/LCD.h"
#include "Servo.h"
#include "Camera.h"

//==============================================================================
//                                    MAIN
//==============================================================================

/**
 * main will run the serial command line input to get information from user
 * and enact the commands.
 * @return does not actually return (infinite loop)
 */
int main(){
	//initialization
	lcd_init();
	servo_init();
	cam_init();
	cmd_init();

	//run input loop from serial input to command the controller
	while(1){
		lcd_clr();
		lcd_printString("Waiting For\nUser Input");
		cmd_getCommand();
	}

	//should never get here
	return 0;
}
```

CommandType

CommandType.h

```
/**
 * CommandType Interface
 * @author Curt Henrichs
 * @date 3-31-17
 * Object definition for all commands used in the command line program.
 * Each object has a string name that will be used to polymorphically
 * match when present in the user input stream. The function pointer will be
 * invoked with the assumption that this call will return back to the
 * invocation point after some condition. Additionally, help information
 * is present in the object for internal documentation and for the menu
 * system.
 */

#ifndef COMMANDTYPE_H_
#define COMMANDTYPE_H_

//=============================================================================
//                                Data Structures
//=============================================================================

/**
 * Defines a command object that has a name, help information, and commanded
 * function pointer.
 */
typedef struct command{
	char* name;
	char* helpInfo;
	void (*cmdFnt)(char*);
}command_t;

#endif /* COMMANDTYPE_H_ */
```

CommandLine

CommandLine.h

```
/**
 * CommandLine Module
 * @author Curt Henrichs
 * @date 3-16-17
 *
 * Serial command line parse and function module. Called from main loop to
 * execute all commands given by the user.
 */

#ifndef COMMANDLINE_H_
#define COMMANDLINE_H_

//=============================================================================
//                          Public Function Declaration
```

```
//==============================================================================

/**
 * Initializes the command line module. Specifically the command list is
 * generated so that command parsing can occur.
 */
void cmd_init();

/**
 * Gets the command from the user and will perform that action. Blocking
 */
void cmd_getCommand();

#endif /* COMMANDLINE_H_ */
```

CommandLine.c

```
/**
 * CommandLine Module
 * @author Curt Henrichs
 * @date 3-16-17
 *
 * Serial command line parse and function module. Called from main loop to
 * execute all commands given by the user.
 */

//==============================================================================
//                                  Libraries
//==============================================================================

#include "CommandLine.h"

#include "Hardware/LCD.h"

#include "Commands/CommandType.h"
#include "Commands/CameraCMD.h"
#include "Commands/MemoryCMD.h"
#include "Commands/ServoCMD.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

//==============================================================================
//                        Private Function Declaration
//==============================================================================

/**
 * Generates the list of commands in an array of pointers so that commands
 * can be looked up based on their name when parsed.
 */
static void _generateList();

/**
 * Matches the command given by the user with the predefined list. Will run
 * any command that is matched with given.
 * @param inputStr original string from the user.
 * @param cmdStr parsed out command for matching and further parsing.
 */
static void cmd_parseInput(char* inputStr, char* cmdStr);
```

```
/**
 * Help Command Function
 * @param arg string to parse
 */
static void _help(char* arg);

//==============================================================================
//                        Constant and Macro Declaration
//==============================================================================

/**
 * Defines the number of commands that this device decodes from user input.
 */
#define NUMBER_OF_COMMANDS (9)

//==============================================================================
//                              Private Data Members
//==============================================================================

/**
 * Stores all commands in a list for polymorphic invocation.
 */
static command_t* commandList[NUMBER_OF_COMMANDS];

//input string buffers
static char inputStr[80];
static char cmdStr[80]; //must be same size for worst case parsing

//==============================================================================
//                         Command Objects (Use Like Constant)
//==============================================================================

/**
 * Help Command Object
 */
static command_t HELP = {
       .name = "HELP",
       .helpInfo = "Displays information about a command.\n\tForm HELP or HELP "
                        "<command> or HELP ALL\n",
       .cmdFnt = _help
};

//==============================================================================
//                           Public Function Implementation
//==============================================================================

/**
 * Initializes the command line module. Specifically the command list is
 * generated so that command parsing can occur.
 */
void cmd_init(){
       _generateList();
}

/**
 * Gets the command from the user and will perform that action. Blocking
 */
void cmd_getCommand(){
       //print prompt to the user for next command
       printf("\nEnter Command:");

       //block until string entered
       fgets(inputStr, sizeof inputStr,stdin);
	char* str = inputStr;
	while(*str){
		*str = toupper(*str);
		str++;
	}

	//parse for command
	int matched = sscanf(inputStr,"%s",cmdStr);
	if(matched != EOF){
		cmd_parseInput(inputStr,cmdStr);
	}
}

//================================================================================
//                             Private Function Implementation
//================================================================================

/**
 * Generates the list of commands in an array of pointers so that commands
 * can be looked up based on their name when parsed.
 */
static void _generateList(){
	//setup command list
	commandList[0] = &RR;
	commandList[1] = &WR;
	commandList[2] = &HELP;
	commandList[3] = &PAN;
	commandList[4] = &TILT;
	commandList[5] = &WRCAMREG;
	commandList[6] = &RDCAMREG;
	commandList[7] = &IMAGE;
	commandList[8] = &TRACK;
}

/**
 * Matches the command given by the user with the predefined list. Will run
 * any command that is matched with given.
 * @param inputStr original string from the user.
 * @param cmdStr parsed out command for matching and further parsing.
 */
static void cmd_parseInput(char* inputStr, char* cmdStr){
		//search command list for match
		int commandFoundFlag = 0;
		int i; for(i = 0; i < NUMBER_OF_COMMANDS;++i){
			if(strcmp(cmdStr,commandList[i]->name)==0){
				lcd_clr();
				commandList[i]->cmdFnt(inputStr);
				commandFoundFlag = 1;
			}
		}

		//if command not found then print error message
		if(!commandFoundFlag){
			printf("Invalid Command.\n");
		}
}
```

MemoryCMD

MemoryCMD.h

```
/**
 * MemoryCMD Objects
 * @author Curt Henrichs
 * @date 3-31-17
 * Objects used by the command line to access memory for reading and writing.
 */

#ifndef MEMORYCMD_H_
#define MEMORYCMD_H_

//=============================================================================
//                                  Libraries
//=============================================================================

#include "CommandType.h"

//=============================================================================
//                       Command Objects (Use Like Constant)
//=============================================================================

/**
 * Read Command Object
 */
extern command_t RR;

/**
 * Write Command Object
 */
extern command_t WR;

#endif /* MEMORYCMD_H_ */
```

MemoryCMD.c

```
/**
 * MemoryCMD Objects
 * @author Curt Henrichs
 * @date 3-31-17
 * Objects used by the command line to access memory for reading and writing.
 */

//=============================================================================
//                                  Libraries
//=============================================================================

#include "MemoryCMD.h"

#include <stdio.h>
#include <string.h>
#include "../Hardware/GenericIO.h"
#include "../Hardware/LCD.h"

//=============================================================================
//                        Private Function Declaration
//=============================================================================

/**
 * Read Command Function
 * @param arg string to parse
 */
static void _read(char* arg);
/**
 * Write Command Function
 * @param arg string to parse
 */
static void _write(char* arg);
/**
 * Prints out memory starting at the address for the count amount of bytes
 * @param address the address of memory to start printing
 * @param count the number of bytes to read and print
 */
static void _printMem(unsigned int address, unsigned int count);

//==============================================================================
//                        Command Object Instantiation
//==============================================================================

/**
 * Read Command Object
 */
command_t RR = {
       .name = "RR",
       .helpInfo = "Reads from selected memory.\n\tForm RR <address> or RR "
                   "<address> <count>.\n",
       .cmdFnt = _read
};
/**
 * Write Command Object
 */
command_t WR = {
       .name = "WR",
       .helpInfo = "Writes to selected memory.\n\tForm WR <address> <value>.\n",
       .cmdFnt = _write
};

//==============================================================================
//                           Private Function Implementation
//==============================================================================

/**
 * Read Command Function
 * @param arg string to parse
 */
static void _read(char* arg){
       lcd_printString("Reading Memory");
       char cmd[3];
       unsigned int address, count;
       int matched = sscanf(arg,"%s %x %x",cmd,&address,&count);
       if(matched == 3){
              //Read multiple
              _printMem(address,count);
       }else if(matched == 2){
              //Read single
              _printMem(address,1);
       }else{
              printf("Invalid RR Command Structure.\n");
       }
}

/**
 * Write Command Function
 * @param arg string to parse
 */
static void _write(char* arg){
       lcd_printString("Writing Memory");
       char cmd[3];
       unsigned int address, value;
       int matched = sscanf(arg,"%s %x %x",cmd,&address,&value);
       if(matched == 3){
              //always apply the store IO version of the instruction for immediate
              // change and store in original for memory required versions
              *((volatile unsigned char*)(address | BYPASS_CACHE_FLAG)) = value;
       }else{
              printf("Invalid WR Command Structure.\n");
       }
}


/**
 * Prints out memory starting at the address for the count amount of bytes
 * @param address the address of memory to start printing
 * @param count the number of bytes to read and print
 */
static void _printMem(unsigned int address, unsigned int count){

       //sanity check count for valid specification
       if(count < 1){
              printf("Error - Invalid Number of Addresses Specified.\n");
              return; //error case so leave early
       }else if(address + count < address){
              printf("Warning - Address Overflow.\n");
       }

       //buffer allocation
       char buffer[80];
       char temp[12];

       //print header
       memset(buffer,'\0',sizeof buffer);
       strcat(buffer,"  Base    :");
       int i; for(i = 0; i < count && i < 16; ++i){
              *(temp+3) = '\0';
              sprintf(temp," +%x",i);
              strcat(buffer,temp);
       }
       strcat(buffer,"\n");
       printf(buffer);

       //print bytes as chunks of buffer
       int j; for(j = 0; j < (count - 1) / 16 + 1; ++j){
              memset(buffer,'\0',sizeof buffer);

              //print address
              *(temp+10) = '\0';
              sprintf(temp,"%08x :",address + j*16);
              strcat(buffer,temp);

              //print row or less if count
              int a; for(a = 0; a < 16 && a < (count - j * 16); ++a){
                     unsigned int addr = address + j*16+a;
                     unsigned char val = *((unsigned char*)
                                          (addr | BYPASS_CACHE_FLAG));
                     *(temp+3) = '\0';
                     sprintf(temp," %02x",val);
```

```
                    strcat(buffer,temp);
             }

             //send to serial device
             strcat(buffer,"\n");
             printf(buffer);
      }

      //send final character to clean up display
      printf("\n");
}
```

ServoCMD

ServoCMD.h

```
/**
 * ServoCMD Objects
 * @author Curt Henrichs
 * @date 3-31-17
 * Implementation of the servo control commands used in the command line.
 * Pan and tilt functionality exposed for the user to control the servos.
 */

#ifndef SERVOCMD_H_
#define SERVOCMD_H_

//===============================================================================
//                                  Libraries
//===============================================================================

#include "CommandType.h"

//===============================================================================
//                       Command Objects (Use Like Constant)
//===============================================================================

/**
 * Pan Servo Command Object
 */
extern command_t PAN;

/**
 * Tilt Servo Command Object
 */
extern command_t TILT;

#endif /* SERVOCMD_H_ */
```

ServoCMD.c

```
/**
 * ServoCMD Objects
 * @author Curt Henrichs
 * @date 3-31-17
 * Implementation of the servo control commands used in the command line.
 * Pan and tilt functionality exposed for the user to control the servos.
 */

//===============================================================================
//                                  Libraries
```

```
//==============================================================================

#include "ServoCMD.h"

#include <stdio.h>
#include "../Servo.h"
#include "../Hardware/LCD.h"

//==============================================================================
//                          Private Function Declaration
//==============================================================================

/**
 * Pans the servo by the input value provided
 * @param arg string to parse
 */
static void _pan(char* arg);
/**
 * Tilts the servo by the input value provided
 * @param arg string to parse
 */
static void _tilt(char* arg);

//==============================================================================
//                         Command Object Instantiation
//==============================================================================

/**
 * Pan Servo Command Object
 */
command_t PAN = {
       .name = "PAN",
       .helpInfo = "Absolute position control of the pan servo. Valid input is "
                  "between 0 and 175.\n\tForm PAN <position>\n",
       .cmdFnt = _pan
};
/**
 * Tilt Servo Command Object
 */
command_t TILT = {
       .name = "TILT",
       .helpInfo = "Absolute position control of the tilt servo. Valid input is "
                  "between 0 and 144.\n\tForm TILT <position>\n",
       .cmdFnt = _tilt
};

//==============================================================================
//                            Private Function Implementation
//==============================================================================

/**
 * Pans the servo by the input value provided
 * @param arg string to parse
 */
static void _pan(char* arg){
       lcd_printString("Panning");
       char cmd[5];
       int pos;
       int matched = sscanf(arg,"%s %d",cmd,&pos);
       if(matched > 1){
              if(servo_pan(pos) == -1){
                     printf("Position ");
                     char temp[11]; *(temp+10) = '\0';
```

```
                        sprintf(temp,"%d",pos);
                        printf(temp);
                        printf(" is invalid. Enter between 0 and 175.\n");
                }
        }else{
                printf("Invalid command structure.\n");
        }
}

/**
 * Tilts the servo by the input value provided
 * @param arg string to parse
 */
static void _tilt(char* arg){
        lcd_printString("Tilting");
        char cmd[5];
        int pos;
        int matched = sscanf(arg,"%s %d",cmd,&pos);
        if(matched > 1){
                if(servo_tilt(pos) == -1){
                        printf("Position ");
                        char temp[11]; *(temp+10) = '\0';
                        sprintf(temp,"%d",pos);
                        printf(temp);
                        printf(" is invalid. Enter between 0 and 143.\n");
                }
        }else{
                printf("Invalid command structure.\n");
        }
}
```

CameraCMD

CameraCMD.h

```
/**
 * CameraCMD Object
 * @author Curt Henrichs
 * @date 3-31-17
 * Command Line objects for the camera commands (read register, write register,
 * display image, and tracking target) Commands exposed are to be treated as
 * constants.
 */

#ifndef CAMERACMD_H_
#define CAMERACMD_H_

//==============================================================================
//                                   Libraries
//==============================================================================

#include "CommandType.h"

//==============================================================================
//                         Command Objects (Use Like Constant)
//==============================================================================

/**
 * Reads from the camera over I2C
 */
extern command_t RDCAMREG;
/**
```

```
 * Writes to the camera over I2C
 */
extern command_t WRCAMREG;
/**
 * Displays video feed from camera onto VGA display.
 */
extern command_t IMAGE;
/**
 * Tracks an object in view of camera with servo turret.
 */
extern command_t TRACK;

#endif /* CAMERACMD_H_ */
```

CameraCMD.c

```
/**
 * CameraCMD Object
 * @author Curt Henrichs
 * @date 3-31-17
 * Command Line objects for the camera commands (read register, write register,
 * display image, and tracking target) Commands exposed are to be treated as
 * constants.
 */

//==============================================================================
//                                  Libraries
//==============================================================================

#include "CameraCMD.h"

#include <stdio.h>
#include <string.h>
#include <stdint.h>
#include "../Servo.h"
#include "../Camera.h"
#include "../Hardware/GenericIO.h"
#include "../Hardware/Pushbutton.h"
#include "../Hardware/LCD.h"

//==============================================================================
//                         Constant & Macro Declaration
//==============================================================================

//Filtering Constants
/**
 * Filter types for tracking algorithms
 */
typedef enum FILTER_TYPE{
       DARK_AUTO,    //!< DARK_AUTO automatic filtering for dark objects
       LIGHT_AUTO,   //!< LIGHT_AUTO automatic filtering for light objects
       DARK_MANUAL,  //!< DARK_MANUAL manual value for dark objects
       LIGHT_MANUAL, //!< LIGHT_MANUAL manual value for light objects
       TARGET_VALUE, //!< TARGET_VALUE manual target value filtering
       ERROR          //!< ERROR parse failed
} FILTER_TYPE_t;

/**
 * Filter tolerance from desired value
 */
#define TOLERANCE 0x08
/**
```

```
 * Minimum pixel confidence value before pixel is counted in the boxing method
 */
#define CONFIDENCE_THRESHOLD 0xF0

//Position Control Constants
/**
 * Minimum command value sent to X servo before attenuated to prevent hunting
 */
#define MIN_SERVO_COMMAND_X 1
/**
 * Minimum command value sent to Y servo before attenuated to prevent hunting
 */
#define MIN_SERVO_COMMAND_Y 4

//PID Controller Constants
/**
 * Proportional Term for X servo divided by PID_DEM
 */
#define PX 2
/**
 * Proportional Term for Y servo divided by PID_DEM
 */
#define PY 3
/**
 * Derivative term for X servo divided by PID_DEM
 */
#define DX 0
/**
 * Derivative term for Y servo divided by PID_DEM
 */
#define DY 1
/**
 * Common division term for PID controller used for integer math
 */
#define PID_DEM 8

//==============================================================================
//                             I2C Register Structure
//==============================================================================

/**
 * Bounding box of the shape being tracked
 */
struct boundingBox{
	uint8_t minr;
	uint8_t minc;
	uint8_t maxr;
	uint8_t maxc;
};

//==============================================================================
//                        Private Function Declaration
//==============================================================================

/**
 * Reads from the I2C camera.
 * @param arg string to parse
 */
static void _readCam(char* arg);
/**
 * Writes to the I2C camera.
 * @param arg string to parse.
 */
```

```
static void _writeCam(char* arg);
/**
 * Displays the camera image onto the VGA hardware. Runs in a loop until
 * pushbutton two is pressed to exit.
 * @param arg ignored
 */
static void _displayImage(char* arg);
/**
 * Track command looks for target that meets threshold requirement. Manipulates
 * servo turret to track target under closed loop control.
 * @param arg ignored
 */
static void _track(char* arg);

/**
 * Calculates the bounding box of the target and updates through the bounding
 * box pointer.
 * @param box pointer to object that will be updated with new bounding info.
 */
static inline void _track_boundingBox(struct boundingBox* box);
/**
 * Parses user input for the tracking command such that a filtering type and
 * possible manual value is selected.
 * @param arg String to parse
 * @param min minimum value of range
 * @param max maximum value of range
 * @return filter type
 */
static inline FILTER_TYPE_t _track_inputParsing(char* arg, uint8_t* min,
    uint8_t* max);

//=============================================================================
//                       Command Object Instantiation
//=============================================================================

/**
 * Reads from the camera over I2C
 */
command_t RDCAMREG = {
       .name = "RDCAMREG",
       .helpInfo = "Reads from Camera I2C register specified as parameter.\n\t"
                   "Form RDCAMREG RegisterNumber\n",
       .cmdFnt = _readCam
};
/**
 * Writes to the camera over I2C
 */
command_t WRCAMREG = {
       .name = "WRCAMREG",
       .helpInfo = "Writes to Camera I2C register specified as parameter.\n\t"
                   "Form WRCAMREG RegisterNumber Value\n",
       .cmdFnt = _writeCam
};
/**
 * Displays video feed from camera onto VGA display.
 */
command_t IMAGE = {
              .name = "IMAGE",
              .helpInfo = "Displays image from camera onto the VGA display. \n\t"
                          "Form IMAGE\n",
              .cmdFnt = _displayImage
};
/**
```

```
 * Tracks an object in view of camera with servo turret.
 */
command_t TRACK = {
              .name = "TRACK",
              .helpInfo = "Tracks object in view of camera with servo turret.\n"
                          "Displays filtered image onto the VGA display.\n"
                          "Type of filter specified to be DARK, LIGHT, TARGET.\n"
                         "Value sets threshold to manual, else automatic adjustment."
                          "\n\tForm TRACK <type> <value>\n",
              .cmdFnt = _track
};

//==============================================================================
//                            Private Function Implementation
//==============================================================================

/**
 * Reads from the I2C camera.
 * @param arg string to parse
 */
static void _readCam(char* arg){
       lcd_printString("Reading I2C");
       char cmd[10];
       unsigned int regNumber;
       int matched = sscanf(arg,"%s %x",cmd,&regNumber);
       if(matched > 1){
              unsigned char data = I2C_read(CAMERA_I2C_ADDRESS,regNumber);
              printf("%02x: %02x\n",regNumber,data);
       }else{
              printf("Invalid read command structure\n");
       }
}

/**
 * Writes to the I2C camera.
 * @param arg string to parse.
 */
static void _writeCam(char* arg){
       lcd_printString("Writing I2C");
       char cmd[10];
       unsigned int regNumber, value;
       int matched = sscanf(arg,"%s %x %x",cmd,&regNumber,&value);
       if(matched > 2){
              I2C_write(CAMERA_I2C_ADDRESS,regNumber,value);
       }else{
              printf("Invalid write command structure\n");
       }
}

/**
 * Displays the camera image onto the VGA hardware. Runs in a loop until
 * pushbutton two is pressed to exit.
 * @param arg ignored
 */
static void _displayImage(char* arg){
       lcd_printString("Displaying\nCamera Image");
       printf("Press Button 2 to exit.\n");

       while(pushbutton_read(PUSHBUTTON_1_MASK)){
              cam_imageCapture();
       }

       //set to default
	vga_clearDisplay();
}

/**
 * Track command looks for target that meets threshold requirement. Manipulates
 * servo turret to track target under closed loop control.
 * @param arg ignored
 */
static void _track(char* arg){

	//data members
	struct frameData* fd;
	struct boundingBox box;
	int16_t errorX, prevErrorX = 0;
	int16_t errorY, prevErrorY = 0;
	uint8_t targetMin = 0, targetMax = 0;

	//parse user input
	FILTER_TYPE_t type = _track_inputParsing(arg,&targetMin,&targetMax);
	if(type == ERROR){
		return; //return since parse failed
	}

	//notify user UI of exit process
	lcd_printString("Tracking Target");
	printf("Press Button 2 to exit.\n");

	//loop the tracking until button pressed
	int16_t cogx,cogy;
	int32_t xcommand, ycommand;
	while(pushbutton_read(PUSHBUTTON_1_MASK)){
		//get frame data
		fd = cam_track(targetMin,targetMax);

		//determine location of object
		box.minr = VGA_ROW_MAX;
		box.minc = VGA_COL_MAX;
		box.maxr = 0;
		box.maxc = 0;
		_track_boundingBox(&box);

		//calculate cog of the object
		cogy = (box.maxr + box.minr)/2;
		cogx = (box.maxc + box.minc)/2;
		if(cogx < 0 || cogy < 0){
			continue; //no pixels to track
		}

		//calculate servo command
		errorX = (40 - cogx);
		errorY = (30 - cogy);
		xcommand = ((errorX*PX) + (errorX - prevErrorX)*DX)/PID_DEM;
		if(xcommand < MIN_SERVO_COMMAND_X && xcommand > -MIN_SERVO_COMMAND_X){
			xcommand = 0;
		}
		ycommand = ((errorY*PY) + (errorY - prevErrorY)*DY)/PID_DEM;
		if(ycommand < MIN_SERVO_COMMAND_Y && ycommand > -MIN_SERVO_COMMAND_Y){
			ycommand = 0;
		}
		prevErrorX = errorX;
		prevErrorY = errorY;

		//set servo motors
```

```
            setServoX(xcommand);
            setServoY(ycommand);

            //adjust target value in automatic mode
            switch(type){
                  case DARK_AUTO:
                        if(fd->minBrightness > (0xFF - TOLERANCE)){
                              targetMax = fd->minBrightness;
                        }else{
                              targetMax = fd->minBrightness + TOLERANCE;
                        }
                        break;
                  case LIGHT_AUTO:
                        if(fd->maxBrightness < (TOLERANCE)){
                              targetMin = fd->maxBrightness;
                        }else{
                              targetMin = fd->maxBrightness - TOLERANCE;
                        }
                        break;
                  default:
                        break;
            }
      }

      //set system to default
      servo_init();
      vga_clearDisplay();
}

/**
 * Calculates the bounding box of the target and updates through the bounding
 * box pointer.
 * @param box pointer to object that will be updated with new bounding info.
 */
static inline void _track_boundingBox(struct boundingBox* box){
      //store data statically for reduced stack frame time
      static uint8_t row,col,pixel;
      static uint8_t* ptr;

      //iterate through buffer to determine location of object
      for(row=0;row<VGA_ROW_MAX;row++){
            ptr = (uint8_t*)(VGA_MEM_START + (row << 7));
            for(col=0;col<VGA_COL_MAX;col++){
                  pixel = *(ptr + col);
                  if(pixel > CONFIDENCE_THRESHOLD){
                        //check within current bounding box
                        if(row < box->minr)
                              box->minr = row;
                        if(col < box->minc)
                              box->minc = col;
                        if(col > box->maxc)
                              box->maxc = col;
                        if(row > box->maxr)
                              box->maxr = row;
                  }
            }
      }
}

/**
 * Parses user input for the tracking command such that a filtering type and
 * possible manual value is selected.
 * @param arg String to parse
 * @param min minimum value of range
 * @param max maximum value of range
 * @return filter type
 */
static inline FILTER_TYPE_t _track_inputParsing(char* arg, uint8_t* min, uint8_t*
max){
	// parse input for filtering method
	char cmd[10];
	char typeStr[80]; //large to prevent failure of size
	unsigned int value;
	FILTER_TYPE_t retVal;
	int matched = sscanf(arg,"%s %s %x",cmd,typeStr,&value);
	if(matched == 3){
		if(strcmp(typeStr,"TARGET") == 0){
			retVal = TARGET_VALUE;
			*min = (value < TOLERANCE)? value : value - TOLERANCE;
			*max = (value > 0xFF - TOLERANCE)? value : value + TOLERANCE;
		}else if(strcmp(typeStr,"LIGHT") == 0){
			retVal = LIGHT_MANUAL;
			*min = value;
			*max = 0xFF;
		}else if(strcmp(typeStr,"DARK") == 0){
			retVal = DARK_MANUAL;
			*min = 0x00;
			*max = value;
		}else{
			//error
			printf("Invalid TRACK Command Structure.\n");
			retVal = ERROR;
		}
	}else if(matched == 2){
		if(strcmp(typeStr,"LIGHT") == 0){
			retVal = LIGHT_AUTO;
			*min = 0x55; //default value
			*max = 0xFF;
		}else if(strcmp(typeStr,"DARK") == 0){
			retVal = DARK_AUTO;
			*min = 0x00;
			*max = 0x55;
		}else{
			//error
			printf("Invalid TRACK Command Structure.\n");
			retVal = ERROR;
		}
	}else{
		printf("Invalid TRACK Command Structure.\n");
		retVal = ERROR;
	}
	return retVal;
}
```

Camera

Camera.h

```
/**
 * Camera Module
 * @author Curt Henrichs
 * @date 3-26-17
 * Defines the entirety of the camera module from the hardware perspective.
 * This device will implement the control loop for tracking and provide VGA
 * output.
 */

#ifndef CAMERA_H_
#define CAMERA_H_

//================================================================================
//                                  Libraries
//================================================================================

#include "Hardware/I2C.h"
#include "Hardware/VGA.h"

//================================================================================
//                              I2C Register Structure
//================================================================================

/**
 * Information returned regarding meta-data of the image frame
 */
struct frameData{
       uint8_t maxBrightness;
       uint8_t minBrightness;
};

//================================================================================
//                                        Constant and Macro Declaration
//================================================================================

/**
 * I2C address of the camera as 7-bit value shift left 1 and append R/W bit.
 */
#define CAMERA_I2C_ADDRESS 0x60

// Registers of interest from camera
#define CAMERA_I2C_CLKRC 0x11
#define CAMERA_I2C_COMC 0x14
#define CAMERA_I2C_COML 0x39
#define CAMERA_I2C_MIDH 0x1C
#define CAMERA_I2C_MIDL 0x1D

//================================================================================
//                           Public Function Declaration
//================================================================================

/**
 * Initializes the camera module over I2C for slow rate and desired format.
 * Sets any necessary flags and controls for the camera and prepares all
 * data structures.
 */
void cam_init();

/**
 * Captures a frame from the camera and returns it unaltered.
 */
void cam_imageCapture();
/**
 * Captures a frame from the camera and returns it filtered with the given value
 * @param thresMin value used as lower bound for filtering
 * @param thresMax value used as upper bound for filtering
 * @return meta-date of the frame as pointer to the internal structure.
 */
struct frameData* cam_track(uint8_t thresMin, uint8_t thresMax);
```

```
#endif /* CAMERA_H_ */
```

Camera.c

```
/**
 * Camera Module
 * @author Curt Henrichs
 * @date 3-26-17
 * Defines the entirety of the camera module from the hardware perspective.
 * This device will implement the control loop for tracking and provide VGA
 * output.
 */

//==============================================================================
//                                  Libraries
//==============================================================================

#include "Camera.h"
#include "Hardware/Timer.h"

//==============================================================================
//                                        Constant and Macro Declaration
//==============================================================================

// I2C register values
#define CAMERA_CLKRC_SCALER 0x0F //(17.7MHz)/((Scaler+1)*4) 6-bit value
#define CAMERA_COMC_QCIF_MASK 0x20
#define CAMERA_COML_PCLK_VALID_MASK 0x40

// Video format size
#define QCIF_COL_MAX 176
#define QCIF_ROW_MAX 144

// Camera clock signals for pixel synchronization
#define CAM_CNTRL_VSYNC_MASK (0x04)
#define CAM_CNTRL_HREF_MASK (0x02)
#define CAM_CNTRL_PCLK_MASK (0x01)

// Camera Control Register and Camera Pixel Register pointers
#define CAM_CNTRL_REGISTER (uint8_t volatile*)(CAM_CONTROL_BASE | BYPASS_CACHE_FLAG)
#define CAM_PIXEL_REGISTER (uint8_t volatile*)(PIXEL_PORT_BASE | BYPASS_CACHE_FLAG)

//confidence of a pixel that meets minimum required for object
#define MIN_CONFIDENCE_MATCH 0x7F

//==============================================================================
//                              Private Data Members
//==============================================================================

/**
 * Data for each frame. While private it is exposed through the captureImage
 * function for further processing.
 */
static struct frameData fd;

//==============================================================================
//                          Public Function Implementation
//==============================================================================

/**
 * Initializes the camera module over I2C for slow rate and desired format.
 * Sets any necessary flags and controls for the camera and prepares all
 * data structures.
 */
void cam_init(){

	//initialize members
	fd.maxBrightness = 0x00;
	fd.minBrightness = 0xFF;

	//initialize I2C camera hardware
	I2C_init();
	timer_delay(100000,TIMER_0);
	I2C_write(CAMERA_I2C_ADDRESS,CAMERA_I2C_CLKRC,CAMERA_CLKRC_SCALER);
	timer_delay(100000,TIMER_0);
	I2C_write(CAMERA_I2C_ADDRESS,CAMERA_I2C_COML,CAMERA_COML_PCLK_VALID_MASK);
	timer_delay(100000,TIMER_0);
	I2C_write(CAMERA_I2C_ADDRESS,CAMERA_I2C_COMC,CAMERA_COMC_QCIF_MASK);
	timer_delay(100000,TIMER_0);

	//clear VGA display
	vga_clearDisplay();
}

/**
 * Captures a frame from the camera and returns it unaltered.
 */
void cam_imageCapture(){
	static int row, col;
	static volatile uint8_t* vgaPtr;
	static uint8_t pixel;

	//start of image
	while(!(*CAM_CNTRL_REGISTER & CAM_CNTRL_VSYNC_MASK));
	while((*CAM_CNTRL_REGISTER & CAM_CNTRL_VSYNC_MASK));
	vgaPtr = VGA_MEM_START;

	//gather by row
	for(row = QCIF_ROW_MAX-1;row >= 0; row--){
		//set next VGA row
		if(row > 11 && row < 132 && row%2){
			vgaPtr = VGA_MEM_START + ((row - 12)/2 << 7);
		}
		//gather pixels in each column
		while(!(*CAM_CNTRL_REGISTER & CAM_CNTRL_HREF_MASK));
		for(col = 0; (col < QCIF_COL_MAX) && (*CAM_CNTRL_REGISTER
		              & CAM_CNTRL_HREF_MASK); col++){
			//get single pixel
			while(!(*CAM_CNTRL_REGISTER & CAM_CNTRL_PCLK_MASK));
			pixel = *CAM_PIXEL_REGISTER;
			while((*CAM_CNTRL_REGISTER & CAM_CNTRL_PCLK_MASK));

			//display pixels to VGA
			if(row > 11 && row < 132 && row%2 && col > 7 && col < 168
			   && col%2){
				//write to VGA
				*vgaPtr = pixel;
				vgaPtr++;
			}
		}
		while((*CAM_CNTRL_REGISTER & CAM_CNTRL_HREF_MASK));
	}
}

/**
```

```
 * Captures a frame from the camera and returns it filtered with the given value
 * @param thresMin value used as lower bound for filtering
 * @param thresMax value used as upper bound for filtering
 * @return meta-date of the frame as pointer to the internal structure.
 */
struct frameData* cam_track(uint8_t thresMin, uint8_t thresMax){
	static int row, col;
	static volatile uint8_t* vgaPtr;
	static uint16_t pixel, vpixel, prevPixel;

	//start of image
	while(!(*CAM_CNTRL_REGISTER & CAM_CNTRL_VSYNC_MASK));
	while((*CAM_CNTRL_REGISTER & CAM_CNTRL_VSYNC_MASK));
	fd.maxBrightness = 0x00;
	fd.minBrightness = 0xFF;
	vgaPtr = VGA_MEM_START;

	//gather by row
	for(row = QCIF_ROW_MAX-1;row >= 0; row--){
		//reset confidence
		prevPixel = 0;

		//set next VGA row
		if(row > 11 && row < 132 && row%2){
			vgaPtr = VGA_MEM_START + ((row - 12)/2 << 7);
		}
		//gather pixels in each column
		while(!(*CAM_CNTRL_REGISTER & CAM_CNTRL_HREF_MASK)); //while 0
		for(col = 0; (col < QCIF_COL_MAX) && (*CAM_CNTRL_REGISTER
			      & CAM_CNTRL_HREF_MASK); col++){
			//get single pixel
			while(!(*CAM_CNTRL_REGISTER & CAM_CNTRL_PCLK_MASK));
			pixel = *CAM_PIXEL_REGISTER;
			while((*CAM_CNTRL_REGISTER & CAM_CNTRL_PCLK_MASK));

			//display pixels to VGA
			if(row > 11 && row < 132 && row%2 && col > 7 && col < 168
			   && col%2){
				//write to VGA
				if(pixel >= thresMin && pixel <= thresMax){
					vpixel = MIN_CONFIDENCE_MATCH + prevPixel;
					prevPixel = *vgaPtr = (vpixel > 0x00FF)?
					               0xFF : (uint8_t)(vpixel);
				}else{
					vpixel = (prevPixel >= MIN_CONFIDENCE_MATCH)?
					               (prevPixel - MIN_CONFIDENCE_MATCH) :
					                    0;
					prevPixel = *vgaPtr = (uint8_t)(vpixel);
				}
				vgaPtr++;

				//update pixel information
				if(fd.maxBrightness < pixel)
					fd.maxBrightness = pixel;
				if(fd.minBrightness > pixel)
					fd.minBrightness = pixel;
			}
		}
		while((*CAM_CNTRL_REGISTER & CAM_CNTRL_HREF_MASK));
	}
	return &fd;
}
```

Servo

Servo.h

```c
/**
 * Servo Module
 * @author Curt Henrichs
 * @date 3-15-17
 *
 * Servo module is a pan/tilt abstraction for the camera base. The module will
 * rotate x and y by the limits specified. Absolute position control is used
 * with respect of the column and row position of the camera.
 *
 * Servo Motor Tuning
 *     PAN SERVO:
 *             MIN OCR 525
 *             MAX OCR 2400
 *             CTR OCR 1425
 *
 *     TILT SERVO:
 *             MIN OCR 1100
 *             MAX OCR 2100
 *             CTR OCR 1775
 *
 * Tuned approximate equations
 *  PAN_OCR = 107 * pos / 10 + 510
 *  TILT_OCR = 69 * pos / 10 + 1015
 */

#ifndef SERVO_H_
#define SERVO_H_

//=============================================================================
//                        Constants and Macro Declaration
//=============================================================================

#define PAN_MAX (175)      /** Maximum Pan in pixel cols */
#define PAN_MIN (0)        /** Minimum Pan in pixel cols */
#define TILT_MAX (143)     /** Maximum Tilt in pixel rows */
#define TILT_MIN (0)       /** Minimum Tilt in pixel rows */

//=============================================================================
//                          Public Function Declaration
//=============================================================================

/**
 * Sets up the PWM to move the servos to center position
 */
void servo_init();
/**
 * Used to provide an absolute x-axis servo position. The number provided is to
 * be a number between 0 and 175. This corresponds to the horizontal resolution
 * of the camera. These values should cause a full range of travel from far
 * left to far right.
 * @param col the x tracking position
 * @return -1 if invalid input, else 0
 */
int servo_pan(int col);
/**
 * Used to provide absolute y-axis servo position. the number provided is to be
 * a number between 0 and 144. This corresponds to the vertical resolution for
 * the camera. These values should cause a full range of travel from top to
 * bottom.
 * @param row the y tracking position
 * @return -1 if invalid input, else 0
 */
int servo_tilt(int row);
/**
 * Gets the current pan location
 * @return absolute position of servo.
 */
int getServoX();
/**
 * Sets the relative pan location. Will stop at limit of range.
 * @param xpos position to add to current position.
 */
void setServoX(int xpos);
/**
 * Gets the current tilt location.
 * @return absolute position of servo.
 */
int getServoY();
/**
 * Sets the relative tilt location. Will stop at limit of range.
 * @param ypos position to add to current position.
 */
void setServoY(int ypos);

#endif /* SERVO_H_ */
```

Servo.c

```
/**
 * Servo Module
 * @author Curt Henrichs
 * @date 3-15-17
 *
 * Servo module is a pan/tilt abstraction for the camera base. The module will
 * rotate x and y by the limits specified. Absolute position control is used
 * with respect of the column and row position of the camera.
 *
 * Servo Motor Tuning
 *     PAN SERVO:
 *            MIN OCR 525
 *            MAX OCR 2400
 *            CTR OCR 1425
 *
 *     TILT SERVO:
 *            MIN OCR 1100
 *            MAX OCR 2100
 *            CTR OCR 1775
 *
 * Tuned approximate equations
 *  PAN_OCR = 107 * pos / 10 + 510
 *  TILT_OCR = 69 * pos / 10 + 1015
 */

//============================================================================
//                                  Libraries
//============================================================================

#include "Servo.h"
#include "Hardware/PWM.h"

//============================================================================
```

```c
//                         Constants and Macro Declaration
//==============================================================================

/**
 * Converts between pixel position to OCRA register
 * @param column position of pixel
 * @return OCR PWM value
 */
#define PAN_POS_TO_OCR_VALUE(pos) ((107 * pos)/10 + 510)
/**
 * Converts between pixel position to OCRB register
 * @param column position of pixel
 * @return OCR PWM value
 */
#define TILT_POS_TO_OCR_VALUE(pos) ((69 * pos)/10 + 1015)
/**
 * Converts between OCRA register value to pixel position
 * @param value is OCR register value
 * @return pixel position column
 */
#define OCR_VALUE_TO_PAN_POS(value) (((value - 510)*10)/107)
/**
 * Converts between OCRB register value to pixel position
 * @param value in OCR register value
 * @return pixel position row
 */
#define OCR_VALUE_TO_TILT_POS(value) (((value - 1015)*10)/69)

//==============================================================================
//                           Public Function Implementation
//==============================================================================

/**
 * Sets up the PWM to move the servos to center position
 */
void servo_init(){
       servo_pan(PAN_MAX/2);
       servo_tilt(TILT_MAX/2+5);
}

/**
 * Used to provide an absolute x-axis servo position. The number provided is to
 * be a number between 0 and 175. This corresponds to the horizontal resolution
 * of the camera. These values should cause a full range of travel from far
 * left to far right.
 * @param col the x tracking position
 * @return -1 if invalid input, else 0
 */
int servo_pan(int col){
       if(col < PAN_MIN || col > PAN_MAX){
              return -1;
       }
       PWM_writeOCRA(PAN_POS_TO_OCR_VALUE(col));
       return 0;
}

/**
 * Used to provide absolute y-axis servo position. the number provided is to be
 * a number between 0 and 144. This corresponds to the vertical resolution for
 * the camera. These values should cause a full range of travel from top to
 * bottom.
 * @param row the y tracking position
 * @return -1 if invalid input, else 0
 */
int servo_tilt(int row){
	if(row < TILT_MIN || row > TILT_MAX){
		return -1;
	}
	PWM_writeOCRB(TILT_POS_TO_OCR_VALUE(row));
	return 0;
}

/**
 * Gets the current pan location
 * @return absolute position of servo.
 */
int getServoX(){
	return OCR_VALUE_TO_PAN_POS(PWM_readOCRA());
}

/**
 * Sets the relative pan location. Will stop at limit of range.
 * @param xpos position to add to current position.
 */
void setServoX(int xpos){
	int pos = OCR_VALUE_TO_PAN_POS(PWM_readOCRA()) + xpos;
	pos = (pos > PAN_MAX) ? (PAN_MAX) : ((pos < PAN_MIN) ? PAN_MIN : pos);
	PWM_writeOCRA(PAN_POS_TO_OCR_VALUE(pos));
}

/**
 * Gets the current tilt location.
 * @return absolute position of servo.
 */
int getServoY(){
	return OCR_VALUE_TO_TILT_POS(PWM_readOCRB());
}

/**
 * Sets the relative tilt location. Will stop at limit of range.
 * @param ypos position to add to current position.
 */
void setServoY(int ypos){
	int pos = OCR_VALUE_TO_TILT_POS(PWM_readOCRB()) + ypos;
	pos = (pos > TILT_MAX) ? (TILT_MAX) : ((pos < TILT_MIN) ? TILT_MIN : pos);
	PWM_writeOCRB(TILT_POS_TO_OCR_VALUE(pos));
}
```

GenericIO

GenericIO.h

```
/**
 * GENERIC_IO Module
 * @author Curt Henrichs
 * @date 1-24-16, revised 3-10-17
 *
 * Defines a structure that is the general purpose parallel IO device. Should
 * be used for all specific devices that fit the register pattern for a
 * parallel IO device.
 */

#ifndef GEN_IO_LIB_H_
#define GEN_IO_LIB_H_
```

```
//==============================================================================
//                              Libraries Needed
//==============================================================================

#include <stdint.h>

//==============================================================================
//                            Constants and Macros
//==============================================================================

/**
 * Defines the Flag placed on an address to notify the program to bypass
 * the cache and directly update memory.
 */
#define BYPASS_CACHE_FLAG 0x80000000

//==============================================================================
//                              Structure Declaration
//==============================================================================

/**
 * Generic Structure of an IO device for NIOS.
 */
struct GEN_IO{
       uint32_t DATA;
       uint32_t DIRECTION;
       uint32_t INTERRUPT;
       uint32_t EDGE_CAPTURE;
};

#endif
```

I2C

I2C.h

```
/**
 *I2C Hardware Module
 * @author Curt Henrichs
 * @date 3-23-17
 * I2C module is used as a network connection to the Camera's settings.
 */

#ifndef I2C_H_
#define I2C_H_

//==============================================================================
//                                    Libraries
//==============================================================================

#include "GenericIO.h"
#include <stdint.h>
#include <system.h>

//==============================================================================
//                              I2C Register Structure
//==============================================================================

/**
 * Hardware Register set for I2C device
 */
struct I2C_HARDWARE{
      uint8_t PRESCALER_LO;
      uint8_t PRESCALER_HI;
      uint8_t CNTRL;
      uint8_t RX_TX;
      uint8_t STATUS_CMD;
};

//============================================================================
//                                        Constant and Macro Declaration
//============================================================================
/**
 * Hardware register pointer for NIOS image
 */
#define I2C ((struct I2C_HARDWARE volatile *)(OC_I2C_MASTER_TOP_0_BASE |
BYPASS_CACHE_FLAG))
/**
 * Calculates the desired bit-rate from the NIOS clock frequency
 * @param clock NIOS base clock
 * @param desiredRate rate of the I2C bus
 * @return bit-value for prescaler
 */
#define I2C_PRESCALER_FORMULA(clock,desiredRate) ((clock)/(5*desiredRate)-1)

// I2C Hardware Enable Commands
#define I2C_CNTRL_ENABLE_MASK (1<<7)
#define I2C_CNTRL_IENABLE_MASK (1<<6)

// I2C Hardware Commands
#define I2C_CMD_STA_MASK (1<<7)
#define I2C_CMD_STO_MASK (1<<6)
#define I2C_CMD_RD_MASK (1<<5)
#define I2C_CMD_WR_MASK (1<<4)
#define I2C_CMD_ACK_MASK (1<<3)
#define I2C_CMD_IACK_MASK (1<<0)

// I2C Hardware Flags
#define I2C_STATUS_RXACK_FLAG (1<<7)
#define I2C_STATUS_BUSY_FLAG (1<<6)
#define I2C_STATUS_TIP_FLAG (1<<1)
#define I2C_STATUS_IF_FLAG (1<<0)

//============================================================================
//                          Public Function Declaration
//============================================================================

/**
 * This function will initialize the pre-scaler register and enable the I2C
 * peripheral
 */
void I2C_init();
/**
 * This function is used to read the contents of any of the camera's registers.
 * It returns the 8-bit contents of the specified register.
 * @param addr slave address of device 7-bit
 * @param regNumber register address to read on slave device.
 * @return byte read from the device.
 */
uint8_t I2C_read(uint8_t addr, uint8_t regNumber);
/**
 * This function takes the value passed in and writes it to the specified
 * register. It returns nothing.
 * @param addr slave address of device 7-bit
 * @param regNumber register address to write on slave device.
 * @param value data byte to send to slave register.
 * @return 1 if nacked, 0 if acked
 */
uint8_t I2C_write(uint8_t addr, uint8_t regNumber,uint8_t value);


#endif /* I2C_H_ */
```

I2C.c

```
/**
 *I2C Hardware Module
 * @author Curt Henrichs
 * @date 3-23-17
 * I2C module is used as a network connection to the Camera's settings.
 */

//===============================================================================
//                                  Libraries
//===============================================================================

#include "I2C.h"

//===============================================================================
//                           Private Function Declaration
//===============================================================================

/**
 * Needed by every I2C transfer operation. It sets the command needed by the
 * particular I2C transfer, polls TIP for end of transfer and returns the
 * RxACK status (0 for success).
 * @param command bit flags specifying the commands to apply to I2C control
 * @return success/fail of command (ACK = 1, NACK = 0)
 */
uint8_t I2C_action(uint8_t command);

//===============================================================================
//                            Public Function Implementation
//===============================================================================

/**
 * This function will initialize the pre-scaler register and enable the I2C
 * peripheral
 */
void I2C_init(){
       uint16_t scaler = I2C_PRESCALER_FORMULA(50000000,100000);
       I2C->PRESCALER_LO = (scaler)&0xFF;
       I2C->PRESCALER_HI = (scaler>>8)&0xFF;
       I2C->CNTRL = I2C_CNTRL_ENABLE_MASK;
}

/**
 * This function is used to read the contents of any of the camera's registers.
 * It returns the 8-bit contents of the specified register.
 * @param addr slave address of device 7-bit
 * @param regNumber register address to read on slave device.
 * @return byte read from the device.
 */
uint8_t I2C_read(uint8_t addr, uint8_t regNumber){
       uint8_t nacked;

       //write slave address
	I2C->RX_TX = addr << 1;
	nacked = I2C_action(I2C_CMD_STA_MASK | I2C_CMD_WR_MASK);
	if(nacked){
		return -1; //error occurred
	}

	//write register address and end transmit
	I2C->RX_TX = regNumber;
	nacked = I2C_action(I2C_CMD_WR_MASK| I2C_CMD_STO_MASK);
	if(nacked){
		return -1; //error occurred
	}

	//write slave read address, data received is from device
	I2C->RX_TX = (addr << 1) | 1;
	nacked = I2C_action(I2C_CMD_STA_MASK | I2C_CMD_WR_MASK);
	if(nacked){
		return -1; //error occurred
	}
	nacked = I2C_action(I2C_CMD_RD_MASK | I2C_CMD_STO_MASK | I2C_CMD_ACK_MASK);
	return I2C->RX_TX;

}

/**
 * This function takes the value passed in and writes it to the specified
 * register. It returns nothing.
 * @param addr slave address of device 7-bit
 * @param regNumber register address to write on slave device.
 * @param value data byte to send to slave register.
 * @return 1 if nacked, 0 if acked
 */
uint8_t I2C_write(uint8_t addr, uint8_t regNumber, uint8_t value){
	uint8_t nacked;

	//write slave address
	I2C->RX_TX = addr << 1;
	nacked = I2C_action(I2C_CMD_STA_MASK | I2C_CMD_WR_MASK);
	if(nacked){
		return nacked; //error occurred
	}

	//write register address
	I2C->RX_TX = regNumber;
	nacked = I2C_action(I2C_CMD_WR_MASK);
	if(nacked){
		return nacked; //error occurred
	}

	//write data
	I2C->RX_TX = value;
	nacked = I2C_action(I2C_CMD_WR_MASK | I2C_CMD_STO_MASK);
	return nacked;
}

//==============================================================================
//                          Private Function Implementation
//==============================================================================

/**
 * Needed by every I2C transfer operation. It sets the command needed by the
 * particular I2C transfer, polls TIP for end of transfer and returns the
 * RxACK status (0 for success).
 * @param command bit flags specifying the commands to apply to I2C control
 * @return success/fail of command (NACK != 0, ACK = 0)
 */
uint8_t I2C_action(uint8_t command){
	I2C->STATUS_CMD = command;
	while(I2C->STATUS_CMD&I2C_STATUS_TIP_FLAG); //wait for TIP
	return I2C->STATUS_CMD&I2C_STATUS_RXACK_FLAG;
}
```

LCD

LCD.h

```
/**
 * LCD Module
 * @author Curt Henrichs
 * @date 1-4-16, revised 3-10-17
 *
 * Defines the API to control the LCD attached to the DE0. Note that errors
 * will occur in timing and missing characters if interrupts are active during
 * operation. Note that timer 1 is used for delay during output to the LCD.
 */

#ifndef LCD_LIB_H
#define LCD_LIB_H

//==============================================================================
//                                 Libraries
//==============================================================================

#include "Timer.h"
#include "GenericIO.h"

#include <system.h>
#include <stdint.h>

//==============================================================================
//                          Data Structure Declaration
//==============================================================================

/**
 * Defines the control register for the LCD.
 */
struct LCD_CONTROL{
	uint8_t L_CTRL;
};

//==============================================================================
//                         Constant and Macro Declaration
//==============================================================================

// LCD low level interface and addresses
#define LCD_DATA_DIRECTION_MASK 0xff
#define LCD_ENABLE_ON_MASK 0x4
#define LCD_ENABLE_OFF_MASK 0xfb
#define LCD_DATA_CMD_SELECT_DATA_MASK 0x2
#define LCD_DATA_CMD_SELECT_CMD_MASK 0x0

// LCD commands
#define LCD_CMD_DATALENGTH_8BIT 0x38
#define LCD_CMD_DISPLAY_CURSOR_BLINK_ON 0x0f
#define LCD_CMD_CLEAR 0x01
```

```
#define LCD_CMD_INCREMENT_CURSOR_1 0x06
#define LCD_CMD_CURSOR_HOME 0x02
#define LCD_CMD_CURSOR_SHIFT_RIGHT1 0x14
#define LCD_SECONDLINE_CMD 0xc0

//Register Locations
#define LCD_DATA ((struct GEN_IO volatile *)(LCDDATA_BASE | BYPASS_CACHE_FLAG))
#define LCD_CTRL ((struct LCD_CONTROL volatile *)(LCDCONTROL_BASE |
BYPASS_CACHE_FLAG))

//=============================================================================
//                              Function Declaration
//=============================================================================

/**
 * Writes a command to the LCD.
 * @param cmd command to be sent to LCD.
 */
void lcd_cmd(uint8_t cmd);

/**
 * Writes a character to the LCD.
 * @param c character that is to be sent to LCD.
 */
void lcd_prt(char c);

/**
 * Writes the clear command to the LCD.
 */
void lcd_clr(void);

/**
 * Moves the cursor back to home on the LCD.
 */
void lcd_home(void);

/**
 * Initializes the LCD. Sends out the command stream to setup the display. Must
 * call LCDportInit first for direction setting.
 */
void lcd_init(void);

/**
 * Sets the direction of the LCD data port.
 * @param dir Port direction mask for the LCD data port
 */
void lcd_portInit(uint8_t dir);

/**
 * Prints a string of characters to the LCD.
 * @param ptr pointer to the start of a C-string ended with a nul character.
 */
void lcd_printString(char* ptr);

#endif
```

LCD.c

```
/**
 * LCD Module
 * @author Curt Henrichs
 * @date 1-4-16, revised 3-10-17
 *
 * Implements the API to control the LCD attached to the DE0. Note that errors
 * will occur in timing and missing characters if interrupts are active during
 * operation. Note that timer 1 is used for delay during output to the LCD.
 */

//==============================================================================
//                                  Libraries
//==============================================================================

#include "LCD.h"

//==============================================================================
//                          Private Function Declaration
//==============================================================================

/**
 * Polls the busy flag of the LCD to synchronize the program to the LCD
 * instruction execution.
 */
static void lcd_util_pollBusyFlag();

//==============================================================================
//                            Function Implementation
//==============================================================================

/**
 * Writes a command to the LCD.
 * @param cmd command to be sent to LCD.
 */
void lcd_cmd(uint8_t cmd){
	//temporary value for data pin manipulation
	uint8_t ctrlVal;

	//Wait till LCD is ready to accept new instruction
	lcd_util_pollBusyFlag();

	// write E=RS=RW=0
	(LCD_CTRL->L_CTRL) = LCD_DATA_CMD_SELECT_CMD_MASK;

	// write command to data port
	(LCD_DATA->DATA) = cmd;

	// set enable ON
	ctrlVal = (LCD_CTRL->L_CTRL) | LCD_ENABLE_ON_MASK;
	(LCD_CTRL->L_CTRL) = ctrlVal;

	// delay for 500ns
	timer_delay(500, TIMER_1);

	// set enable OFF
	ctrlVal = (LCD_CTRL->L_CTRL) & LCD_ENABLE_OFF_MASK;
	(LCD_CTRL->L_CTRL) = ctrlVal;

}

/**
```

```
 * Writes a character to the LCD.
 * @param c character that is to be sent to LCD.
 */
void lcd_prt(char c){

       //temporary value for data pin manipulation
       uint8_t ctrlVal;

       //Wait till LCD is ready to accept new instruction
       lcd_util_pollBusyFlag();

       // write RW=E=0 RS=1  "0b010"
       (LCD_CTRL->L_CTRL) = LCD_DATA_CMD_SELECT_DATA_MASK;

       // write character to data port
       (LCD_DATA->DATA) = c;

       // set enable ON
       ctrlVal = (LCD_CTRL->L_CTRL) | LCD_ENABLE_ON_MASK;
       (LCD_CTRL->L_CTRL) = ctrlVal;

       // delay for 500ns
       timer_delay(500, TIMER_1);

       // set enable OFF
       ctrlVal = (LCD_CTRL->L_CTRL) & LCD_ENABLE_OFF_MASK;
       (LCD_CTRL->L_CTRL) = ctrlVal;
}

/**
 * Writes the clear command to the LCD.
 */
void lcd_clr(void){
       lcd_cmd(LCD_CMD_CLEAR);
}

/**
 * Moves the cursor back to home on the LCD.
 */
void lcd_home(void){
       lcd_cmd(LCD_CMD_INCREMENT_CURSOR_1);
       lcd_cmd(LCD_CMD_CURSOR_HOME);
}

/**
 * Initializes the LCD. Sends out the command stream to setup the display. Must
 * call LCDportInit first for direction setting.
 */
void lcd_init(void){
       lcd_portInit(0xFF);
       lcd_cmd(LCD_CMD_DATALENGTH_8BIT);
       lcd_cmd(LCD_CMD_DISPLAY_CURSOR_BLINK_ON);
       lcd_cmd(LCD_CMD_CLEAR);
       lcd_cmd(LCD_CMD_INCREMENT_CURSOR_1);
       lcd_cmd(LCD_CMD_CURSOR_HOME);
}

/**
 * Sets the direction of the LCD data port.
 * @param dir Port direction mask for the LCD data port
 */
void lcd_portInit(uint8_t dir){
       (LCD_DATA->DIRECTION) = dir;
```

```
}

/**
 * Prints a string of characters to the LCD.
 * @param ptr pointer to the start of a C-string ended with a nul character.
 */
void lcd_printString(char* ptr){
	int counter;
	for(counter = 0; (*ptr != '\0')&&(counter != 32); counter++){

		// If enter character
		if(*ptr == '\n'){
			//If on first line move to second
			if(counter < 16){
				lcd_cmd(LCD_SECONDLINE_CMD);
				counter = 16;
			}
			// If on second line already end print
			else {
				return;
			}

			//index pointer
			ptr++;
		}
		//Roll over to second line when first line fills
		else{
			if(counter == 16)
				lcd_cmd(LCD_SECONDLINE_CMD);

			//print the character and index pointer
			lcd_prt(*ptr);
			ptr++;
		}
	}
}

//==============================================================================
//                         Private Function Implementation
//==============================================================================

/**
 * Polls the busy flag of the LCD to synchronize the program to the LCD
 * instruction execution.
 */
static void lcd_util_pollBusyFlag(){

	//Set data register to input
	uint32_t tempDataDir = (LCD_DATA->DIRECTION);
	(LCD_DATA->DIRECTION) = 0;

	// Set RW=E=1, RS=0 to get visible busy flag
	(LCD_CTRL->L_CTRL) = 0x5;

	// Needs small delay since processor is faster than LCD. Thus this noop
	// is ran.
	int i = 0; i++;

	// Poll busy flag while it is active, since old operation is still running
	while((LCD_DATA->DATA) & 0x80);

	// Restore the initial direction of the data ports
	(LCD_DATA->DIRECTION) = tempDataDir;
```

```
        (LCD_CTRL->L_CTRL) = 0;
}
```

Timer

Timer.h

```
/**
 * TIMER Module
 * @author Curt Henrichs
 * @date 1-24-16, revised 3-10-17
 *
 * Defines the API for the hardware timers for both interrupt and non-interrupt
 * based timing. Note: Timer works in nanoseconds.
 */

#ifndef TIMER_LIB_H
#define TIMER_LIB_H

//=============================================================================
//                                Libraries
//=============================================================================

#include "GenericIO.h"

#include <stdint.h>
#include <system.h>

//=============================================================================
//                           Structure Declaration
//=============================================================================

/**
 * Defines the timer register hardware interface.
 */
struct TIMER_HARDWARE{
       uint32_t STATUS;
       uint32_t CTRL;
       uint32_t PERIODL;
       uint32_t PERIODH;
};

//=============================================================================
//                           Constant Declaration
//=============================================================================

// Timer setup constants
#define START_TIMER_MASK 0x04
#define CONT_TIMER_MASK 0x02
#define TIMER_STATUS_TO_MASK 0x01
#define INTERRUPT_TIMER_MASK 0x01
#define TIME_PER_CYCLE 20

// Time constant
#define SECOND_CONSTANT 1000000000
#define SCREEN_DELAY_TIME 2000000000
#define MICROSECOND_DELAY_50 50000

// Register Pointers
#define TIMER_0 ((struct TIMER_HARDWARE volatile *)(TIMER_0_BASE |
BYPASS_CACHE_FLAG))
```

```
#define TIMER_1 ((struct TIMER_HARDWARE volatile *)(TIMER_1_BASE |
BYPASS_CACHE_FLAG))

//==============================================================================
//                           Function Declaration
//==============================================================================

/**
 * Delay for the time passed into the timer as a polling loop.
 * @param timeToDelay nanoseconds to delay
 * @param timer Pointer to TIMER hardware.
 */
void timer_delay(uint32_t timeToDelay, struct TIMER_HARDWARE volatile * timer);

/**
 * Sets the timer to be interrupt enabled
 * @param isInterruptable 0 is disabling interrupts, 1 is enabling interrupts
 * @param timeToDelay Time to set in timer before interrupt once started.
 * @param timer Pointer to Timer hardware
 */
void timer_interrupt_init(uint8_t isInterruptable, uint32_t timeToDelay,
             struct TIMER_HARDWARE volatile * timer);

/**
 * Sets the timer to be started or stopped for interrupt use.
 * @param isStart boolean value whether to start of stop timer
 * @param timer Pointer to Timer hardware
 */
void timer_interrupt_start_stop(uint8_t isStart,
             struct TIMER_HARDWARE volatile * timer);

/**
 * Handles the interrupt for the given timer in name. Note that this clears the
 * interrupt but does not perform a callback. Instead template this function or
 * call from actual ISR.
 */
void timer_ISR_TIMER_0();

/**
 * Handles the interrupt for the given timer in name. Note that this clears the
 * interrupt but does not perform a callback. Instead template this function or
 * call from actual ISR.
 */
void timer_ISR_TIMER_1();

#endif
```

Timer.c

```
/**
 * TIMER Module
 * @author Curt Henrichs
 * @date 1-24-16, revised 3-10-17
 *
 * Implements the API for the hardware timers for both interrupt and
 * non-interrupt based timing. Note: Timer works in nanoseconds.
 */

//==============================================================================
//                                  Libraries
//==============================================================================
```

```c
#include "Timer.h"

#include <stdlib.h>

//================================================================================
//                              Function Implementation
//================================================================================

/**
 * Delay for the time passed into the timer as a polling loop.
 * @param timeToDelay nanoseconds to delay
 * @param timer Pointer to TIMER hardware.
 */
void timer_delay(uint32_t timeToDelay, struct TIMER_HARDWARE volatile * timer){

       //Convert from seconds to clock cycles
       timeToDelay = timeToDelay / TIME_PER_CYCLE;

       //Set initial state of the timer needs both since they are 16 bit registers
       (timer -> PERIODL) = timeToDelay;
       (timer -> PERIODH) = timeToDelay >> 16;

       // Clear t0 flag to reset timer if it was set before for whatever reason
       (timer -> STATUS) = TIMER_STATUS_T0_MASK;

       //Start timer count down by setting start bit
       uint8_t timerStatus = START_TIMER_MASK;
       (timer -> CTRL) = timerStatus;

       //Loop while t0 flag is a zero
       while(((timer -> STATUS) & TIMER_STATUS_T0_MASK) == 0)
              ;

       // Clear t0 flag to reset timer
       (timer -> STATUS) = TIMER_STATUS_T0_MASK;
}

/**
 * Sets the timer to be interrupt enabled
 * @param isInterruptable 0 is disabling interrupts, 1 is enabling interrupts
 * @param timeToDelay Time to set in timer before interrupt once started.
 * @param timer Pointer to Timer hardware
 */
void timer_interrupt_init(uint8_t isInterruptable, uint32_t timeToDelay,
              struct TIMER_HARDWARE volatile * timer){

       //check to see if enabling or disabling interrupts
       if(isInterruptable){

              // set IRQ0
              if(timer == TIMER_0){
                     // set interrupt ienable
                     asm volatile(
                            ".equ irq0, 0\n\t"               // interrupt mask
                            "movi r7, (1<<irq0)\n\t"
                            "rdctl  r6, ienable\n\t"   // combine previous
                            "or          r7, r7, r6\n\t"     // ienable and irq0
                            "wrctl ienable, r7\n\t"
                            ::
                     );
              }
              // set IRQ2
              else if (timer == TIMER_1){
                    // set interrupt ienable
                    asm volatile(
                            ".equ irq2, 2\n\t"                  // interrupt mask
                            "movi r7, (1<<irq2)\n\t"
                            "rdctl  r6, ienable\n\t"    // combine previous
                            "or          r7, r7, r6\n\t"       // ienable and irq2
                            "wrctl ienable, r7\n\t"
                            ::
                    );
            }

            //Clear t0 flag to reset timer if it was
            // set before for whatever reason
            (timer -> STATUS) = TIMER_STATUS_T0_MASK;

            //Convert from seconds to clock cycles
            timeToDelay = timeToDelay / TIME_PER_CYCLE;

            //Set initial state of the timer needs both since
            // they are 16 bit registers
            (timer -> PERIODL) = timeToDelay;
            (timer -> PERIODH) = timeToDelay >> 16;

            // set interrupt on control register
            (timer -> CTRL) = (timer -> CTRL) | (INTERRUPT_TIMER_MASK
                 | CONT_TIMER_MASK);

    } else {
            // set IRQ to off
            if(timer == TIMER_0){
                    asm volatile(
                            ".equ irq0, 0\n\t"                  // interrupt mask
                            "movi  r7, (~(1<<irq0))\n\t"
                            "rdctl r6, ienable\n\t"     // combine previous
                            "and   r7, r7, r6\n\t"      // ienable and irq0
                            "wrctl ienable, r7"
                            ::
                    );
            }else if(timer == TIMER_1){
                    asm volatile(
                            ".equ irq0, 0\n\t"                  // interrupt mask
                            "movi  r7, (~(1<<irq2))\n\t"
                            "rdctl r6, ienable\n\t"     // combine previous
                            "and   r7, r7, r6\n\t"      // ienable and irq2
                            "wrctl ienable, r7"
                            ::
                    );
            }

            // disable interrupt
            (timer -> CTRL) = (timer -> CTRL) & !(INTERRUPT_TIMER_MASK
                 | CONT_TIMER_MASK);
    }
}

/**
 * Sets the timer to be started or stopped for interrupt use.
 * @param isStart boolean value whether to start of stop timer
 * @param timer Pointer to Timer hardware
 */
void timer_interrupt_start_stop(uint8_t isStart,
            struct TIMER_HARDWARE volatile * timer){
    //Start the timer
	if(isStart){
		(timer -> CTRL) = (timer -> CTRL) | START_TIMER_MASK;
	}
	//Stop the timer
	else {
		(timer -> CTRL) = (timer -> CTRL) & !START_TIMER_MASK;
	}
}

//===============================================================================
//                                   Generic ISR
//===============================================================================

/**
 * Handles the interrupt for the given timer in name. Note that this clears the
 * interrupt but does not perform a callback. Instead template this function or
 * call from actual ISR.
 */
void timer_ISR_TIMER_0(){
	//write T0 flag to clear interrupt
	(TIMER_0 -> STATUS) = TIMER_STATUS_T0_MASK;
}

/**
 * Handles the interrupt for the given timer in name. Note that this clears the
 * interrupt but does not perform a callback. Instead template this function or
 * call from actual ISR.
 */
void timer_ISR_TIMER_1(){
	//write T0 flag to clear interrupt
	(TIMER_1 -> STATUS) = TIMER_STATUS_T0_MASK;
}
```

Pushbutton

Pushbutton.h

```
/**
 * PUSHBUTTON Module
 * @author Curt Henrichs
 * @date 1-24-16, revised 2-10-16, revised 3-10-17
 *
 * Defines the pushbutton interface. The object is input only, therefore no
 * direction setup is necessary. Interrupts are disabled on this version of
 * NIOS.
 */

#ifndef PUSHBUTTON_LIB_H_
#define PUSHBUTTON_LIB_H_

//===============================================================================
//                                 Libraries
//===============================================================================

#include "GenericIO.h"
#include <stdint.h>
#include <system.h>

//===============================================================================
//                         Constant and Macro Declaration
//===============================================================================
// Button Masks
#define PUSHBUTTON_1_MASK 0x1
#define PUSHBUTTON_2_MASK 0x2

//Registers
#define PUSHBUTTON ((struct GEN_IO volatile *)(PUSHBUTTONS_BASE |
BYPASS_CACHE_FLAG))

//=============================================================================
//                            Function Declaration
//=============================================================================

/**
 * Reads the raw data from the pushbutton. Note that the pushbutton logic is
 * inverted.
 * @param selectedButton number for the correct pushbutton. 1 or 2
 * @return  masked value returning value for register
 */
uint8_t pushbutton_read(uint8_t selectedButton);

#endif
```

Pushbutton.c

```
/**
 * PUSHBUTTON Module
 * @author Curt Henrichs
 * @date 1-24-16, revised 2-10-16, revised 3-10-17
 *
 * Implements the pushbutton interface. The object is input only, therefore no
 * direction setup is necessary. Interrupts are disabled on this version of
 * NIOS.
 */

//=============================================================================
//                                  Libraries
//=============================================================================

#include "PUSHBUTTON.h"

//=============================================================================
//                           Function Implementation
//=============================================================================

/**
 * Reads the raw data from the pushbutton. Note that the pushbutton logic is
 * inverted.
 * @param selectedButton number for the correct pushbutton. 1 or 2
 * @return  masked value returning value for register
 */
uint8_t pushbutton_read(uint8_t selectedButton){
       return (PUSHBUTTON -> DATA) & selectedButton;
}
```

VGA

VGA.h

```c
/**
 * VGA Module
 * @author Curt Henrichs
 * @date 3-31-17
 * VGA display hardware interface, implements clear functionality and the
 * address pointers. All application code must manipulate directly for fast
 * response by the system.
 */

#ifndef VGA_H_
#define VGA_H_

//==============================================================================
//                                    Libraries
//==============================================================================

#include <stdint.h>
#include <system.h>
#include "GenericIO.h"

//==============================================================================
//                                         Constant and Macro Declaration
//==============================================================================

#define VGA_ROW_MAX 60
#define VGA_COL_MAX 80

#define VGA_MEM_START (uint8_t volatile*)(ONCHIP_MEMORY2_0_BASE | BYPASS_CACHE_FLAG)
#define VGA_MEM_END  (VGA_MEM_START + ((VGA_ROW_MAX-1) << 7) + (VGA_COL_MAX-1))

//==============================================================================
//                           Public Function Declaration
//==============================================================================

/**
 * Clears the VGA device by making the screen black on all pixels.
 */
void vga_clearDisplay();

#endif /* VGA_H_ */
```

VGA.c

```c
/**
 * VGA Module
 * @author Curt Henrichs
 * @date 3-31-17
 * VGA display hardware interface, implements clear functionality and the
 * address pointers. All application code must manipulate directly for fast
 * response by the system.
 */

//==============================================================================
//                                    Libraries
//==============================================================================

#include "VGA.h"
```

```
//=============================================================================
//                          Public Function Implementation
//=============================================================================

/**
 * Clears the VGA device by making the screen black on all pixels.
 */
void vga_clearDisplay(){
	uint8_t row,col;
	for(row=0;row<VGA_ROW_MAX;row++){
		for(col=0;col<VGA_COL_MAX;col++){
			*(VGA_MEM_START + (row << 7) + col) = 0x00;
		}
	}
}
```