Smart Eraser

Final Project Report

Project Manager: Heather Libecki
Team Members: Chris Quesada, Juan Colin
Technical Advisors: Dr. Hovannes Kulhandjian, Mr. Roger Moore

Thursday, April 25, 2019

ECE 186B - Senior Design II
Spring 2019 - Dr. Aaron Stillmaker

California State University, Fresno
Electrical and Computer Engineering Department

REVISION HISTORY

Version #	Date Finalized	Name of Editor	Changes Made
1	10/25/18	Heather Libecki Chris Quesada Juan Colin	The initial rough draft of the Project Charter was created, included proper sections and figures specified by Dr. Stillmaker.
2	12/9/18	Heather Libecki Chris Quesada	Revisions were made regarding system components: Ethernet and most of the planned wired connections were removed and replaced with wireless solutions. References were included in necessary figures and statements. Strengths and Weaknesses section was moved to a more appropriate location. Figures and tables were added and updated; Figure 2: not skipped anymore Figure 3: updated Figure 4: referenced correctly Figure 5: removed Figure 7: removed Figure 8: removed New figures were added. Updated GANTT chart to more clearly show dependencies. More research and background references were used. Test plan was added. Added links to items in budget and hyperlinks throughout document in LaTeX.
3	4/12/19	Heather Libecki Chris Quesada Juan Colin	Updated Final Budget Revisions were made regarding system components: Camera and main microcontroller were replaced with better options. Removed Raspberry Pi from design considerations. Removed wireless dongle from design considerations. Added Arduino Uno 3 microcontrollers to control stepper motor units Added sections pertaining to SPI and I^2C communications, updated information on image processing implementation. Modified components used to build prototype stand.

Version #	Date Finalized	Name of Editor	Changes Made
			stay in scope of final product.
4	4/12/19	Heather Libecki Chris Quesada	Changed the Arduino Uno 3 microcontrollers and the Kuman nRF24 wireless transceivers to 3 HUZZAH23 Feather Board microcontrollers for the wireless communications.
5	4/21/19	Heather Libecki Chris Quesada	Updated Milestone Schedule, GANTT Charts, and Test Plan.

Abstract

The Smart Eraser is an apparatus that scans, detects, and intelligently erases markings on a whiteboard. A camera at a fixed location facing the board records an image and sends it wirelessly to a microcontroller where image-processing is performed in order to locate the markings on the whiteboard. Another programmed algorithm then determines a path to erase all markings on the board, and creates instructions that are sent wirelessly to the x-y axis stepper motors. These motors are attached to the pulleys that move the linear tracking system, which allows the eraser to move across the whiteboard to the marking locations. The eraser then returns to its standby position.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. Introduction

The idea of the Smart Eraser originated from the need to allow class time to be utilized for learning. It aims to assist teachers who write lengthy, involved examples on a whiteboard while lecturing by erasing the board in between examples so the teacher can continue to lecture. This would allow students to learn in an environment with less interruptions and distractions from the material being taught, resulting in improved overall time efficiency of the amount of material being taught.

The Smart Eraser is an automatic whiteboard eraser. The main deliverable of this project is an eraser which can move left-to-right on a track, and up-and-down on another motion system attached to the track. This system is able to detect where markings are on a whiteboard through the use of a camera and an image-processing program. The camera is mounted across from the whiteboard, and sends the image of the whiteboard to a microcontroller which processes the image, detects where the markings are, and traverses a path to erase all of these markings via an algorithm. Once the path is created, its locations are converted to a coordinate system that the mechanical aspects of the eraser are able to read. This coordinate system converts the locations of the markings into rotations of the stepper motors attached to the tracks, allowing the eraser to move. Finally, the eraser is sent to its stand-by position once all of the markings have been erased.

The following figure is a simple pictorial representation of the final product.

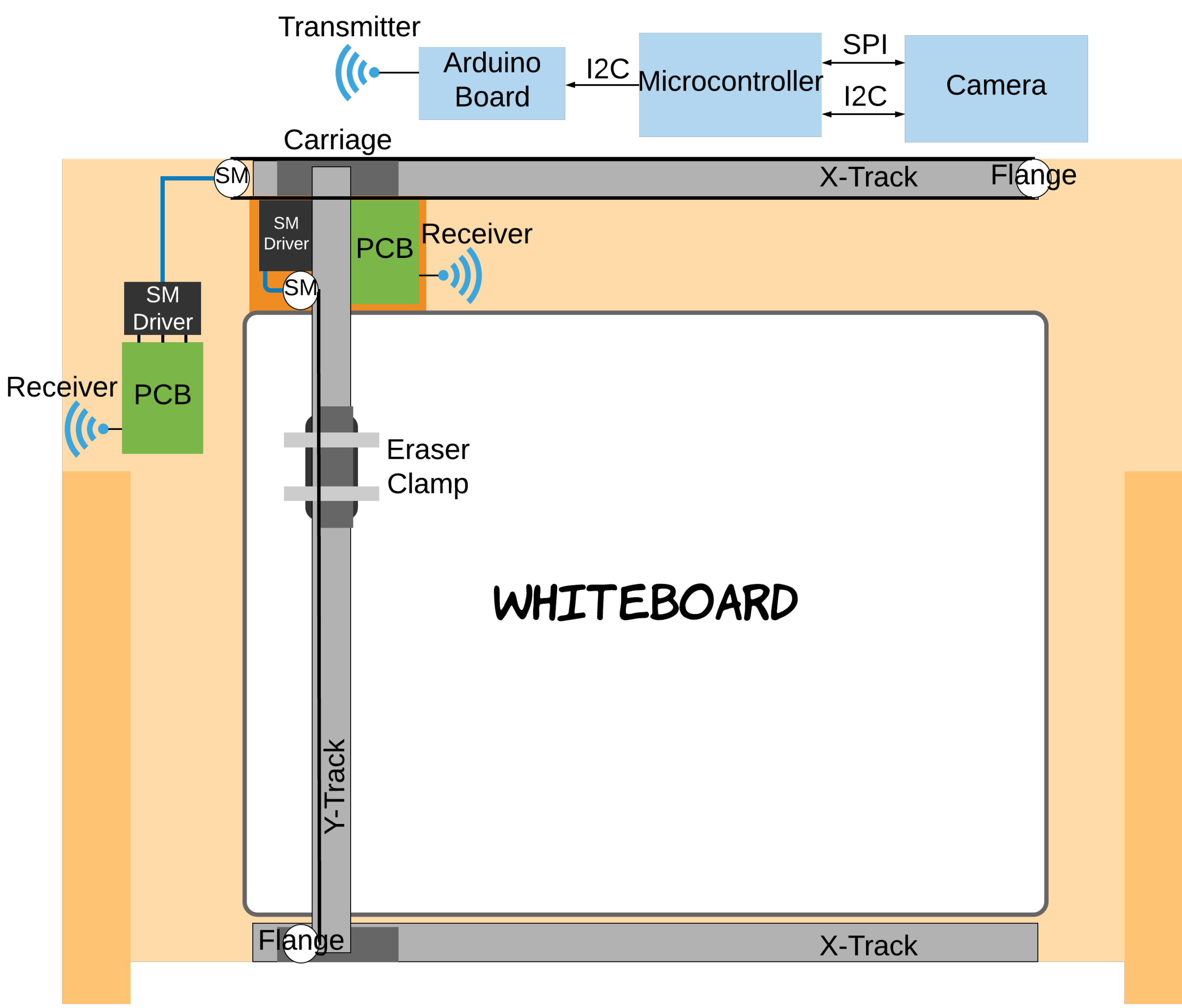


Fig. 1: Design overview of the final product

Figure 1 shows an overview of the final deliverable's mechanical and electrical components put together with names of each item. These components and a small description of their role are listed next. For a more detailed description of each component and its role, see the *Project Description and Boundaries* section of this report.

- Camera: the Arducam will take in an image of a predefined (fixed) area on the whiteboard.
- Microcontroller: the PSoC 6 microcontroller will take in the image from the camera and perform image processing to detect edges, indicating where there are markings on the whiteboard. It will then convert the array of edges that have been detected and find the top-most, left-most, bottom-most, and right-most locations of the markings. These four values are then put into an array. An array of instructions to be given to the stepper motors is then created, and this array is then sent to the Feather board.
- SPI: one of two communication protocols used to communicate with the camera through the micro-controller and trigger a capture of an image.
- I^2C: the other of two communication protocols used to initialize the resolution and compression type of the image that is being captured by the camera. It is also the same type of communication method that is used to send data from the microcontroller to the Feather board.
- Feather Board: acts as an in-between for the microcontroller and the stepper motors, and is connected to the wireless transceivers (a.k.a. Adafruit's HUZZAH32 microcontroller boards) that allow wireless communication between the board and the stepper motors.
- Transmitter: reads in the stepper motor instruction array from the microcontroller and sends the instructions to the stepper motors wirelessly, one at a time.
- Receiver: receives the instructions sent by the transmitter and uses the proper libraries to move the stepper motors accordingly.
- SM: stepper motors that will allow the eraser to move in the X and Y axis directions. The stepper motors are connected to flanges on the opposite side of their axes with a pulley belt to allow movement.
- SM Driver: regulates the voltage, current, and rotation instructions for the stepper motors, allowing them to move the way they should with little to no risk of burnout.
- PCB: connects the stepper motors, their drivers, the wireless transceivers, and the power supplies needed for all of these components.
- Flange: connected to the stepper motors via a pulley belt in order to allow movement.
- Carriage: component within the track that actually moves back and forth. The carriage connects to the pulley belt that runs between the stepper motors and the flanges via washers and nuts. This connects the devices that need to move to the parts that do the movement.
- X-Track: the two tracks that move the eraser in the X-axis direction.
- Y-Track: the track that moves the eraser in the Y-axis direction.
- Eraser Clamp: metal braces and brackets that clamp to the eraser and connect it to the Y-axis carriage in order to move it.
- Whiteboard: surface that will be written on and erased.

A. *Work Distribution*

Heather Libecki was responsible for being the project manager of the Smart Eraser project. She was in charge of the stepper motors and all of their various components, including their drivers, their power system, and the PCBs needed to connect them together. She also created the wireless connections between the receiver stepper motor Feather boards and the transmitter Feather board. She assisted Chris with creating the SPI communication between the transmitter Feather board and the PSoC 6. Finally, she was in charge of the algorithms used to find the instructions for the stepper motors needed to move the eraser to the markings found on the whiteboard, before moving back to its standby position. Therefore, she was in charge of the movement of the Smart Eraser.

Chris Quesada was in charge of the research into and understanding of the PSoC 6 microcontroller, including its comprehensive Integrated Development Environment (IDE). He was then responsible for developing a working SPI and I^2C communication between the PSoC 6 to configure the image sensor(I^2C) to output RGB565 with a 320x240 resolution and then read in image data (SPI). After an image is received, he was to develop an image processing program which takes in an image, converts it to grayscale, then uses a sobel edge detection algorithm to detect objects in the image. To know if this algorithm worked correctly, he had to set up the PSoC 6 to display images on the included TFT display. He headed the creation of the I^2C communication between the HUZZAH32 and the PSoC 6 with assistance from Heather. To allow for easier program management, he was in charge of adding virtual memory to the PSoC 6 to allow for an entire image to be processed rather than parts of the image. Therefore, he was in charge of the image processing brain behind the Smart Eraser, configuration of the PSoC 6 hardware, graphical display of image, addition of virtual memory, and any wired communications to and from the PSoC 6.

Juan Colin was in charge of the initial conception, design, and execution of the physical mechanical system and its interconnections. He was also in charge of drafting and making the wooden prototype stand and mounting all of the components together so as to ensure spacial efficiency and visual impact. Therefore, he was in charge of the entire physical prototype construction of the Smart Eraser. He also assisted in setting up the Arducam via an Arduino platform to analyze the SPI and I^2C being sent from the camera.

During the project's life cycle, there were a number of deliverables that needed to be turned in to Dr. Stillmaker which were worked on by all three members to ensure a consistent flow of information throughout all written documentation.

II. Project Objectives and Success Criteria

This section describes the objectives of the Smart Eraser. A more detailed description of the project can be found in the *Project Description and Boundaries* section of this report.

A. Main Objectives

The main objectives the Smart Eraser have are listed next.

- Create a functioning mechanical system that allows the eraser to move in the x and y plane in order to erase the entire whiteboard.
- Create a functioning Smart Eraser that erases detected markings in a timely manner.
- Create an image processing program to detect said markings on the whiteboard.
- Create a coordinate system based on processed image to move eraser to specific markings.
- Create an algorithm to sort the order in which the markings should be erased to ensure the shortest path is taken.
- Create a motion-detection program to check for people obstructing the whiteboard.

B. Simple success criteria

The specific criteria that need to be met in order to consider this project a success are listed next. These criteria help to measure the success of the final product. Simple success criteria are first listed which describe the tangible goals for the project to be considered a success. Then more ambitious success criteria that describe a more ideal version of the project are listed, which include additional features that would have been added if there was more time after the completion of the simple success criteria.

1) Completed:

- The tracking system moves the eraser to all parts of the whiteboard.
- Eraser erases the entire whiteboard with no smart processing.
- Location of markings in an image found through an edge-detection image processing program.
- Image captured can be processed to find markings on board.
- Array with locations of markings converted to stepper motor rotations to move eraser to proper destination.
- Image processing program on microcontroller works with the image taken to find the markings on the whiteboard.

2) Not Completed:

- Shortest path determined through Dijkstra's analysis of the array with the marking locations.
  - This was unable to be completed within the allotted time. An alternative algorithm that erases the markings within a boundary in a serpentine-like pattern was used instead.
- Camera connects wirelessly to the microcontroller then stores it to its SDRAM.
  - The camera wound up needing to be hardwired to the microcontroller. Because of format being output by the camera to the controller (JPEG rather than RGB565), the camera being hardwired had to be scrapped as well. The "camera" wound up being a smartphone image which was then cropped, converted to RGB565 format, then sent into the image processing program.

C. Ambitious success criteria

Ambitious success criteria detail functionalities that would have been completed if there was more time to work on the project.

1) Completed:

- Unfortunately, because not all of the simple success criteria were completed, no ambitious success criteria were able to be started.

2) Not Completed:

- Phone or tablet application that shows a live feed of the whiteboard from the camera
  - Application can send specific coordinates to the whiteboard to "pick and choose" what section of the board to erase.
- Attachable spray system applies whiteboard cleaning liquid solution to perform "full clean" of whiteboard.
  - Timer tells eraser to perform a "full clean" during the night when no one is using the classroom.
- Eraser can be raised off of whiteboard surface and subsequently re-pressed onto the board as needed.
- Store images each time the board gets erased to provide the teacher the class notes for the day.
- Add remote control capabilities to control eraser at-will.
- Smart Eraser patent.

III. High-Level Requirements

The high-level requirements associated with the completion of this project are outlined in the following list.

The project should:

- Be completed within the outlined budget.
- Be implementable within 2 semesters.
- Be complex enough to warrant the title "senior design project".

- Produce a complete Project Charter outlining the various project information, figures, and tables of the Smart Eraser.
- Have significant, roughly equal portions of the project be completed by each team member.
- Utilize material learned in core and technical elective classes throughout college careers.
- Produce a deliverable that can be presented in the Senior Project Presentation Day event held in the Satellite Student Union building.

IV. Assumptions, Constraints, and Standards

This section outlines the various assumptions that were made for the requirements of this project based on previously learned course information, the constraints the project had during its conception, and the standards that were used and followed throughout the completion of this project.

A. Assumptions

The Smart Eraser was the first large-scale project that the students involved had worked on. Therefore, there were many assumptions made about the components that would be used to complete it, the skill-set and course information needed to complete it, and the amount of time it would take to actually complete the project. Some of the major assumptions made that wound up not being followed included the budget outlined, using the DE1_SoC microcontroller for the main brain behind the Smart Eraser, using the Raspberry Pi microcontroller with wireless transceivers for the wireless communications, and the amount of time it would take to put together the wireless communication system, especially because the first attempt at creating the wireless system ended in failure, so a new system had to be conceptualized and created from scratch with 3 weeks left before the due date of the final working product.

B. Constraints

A few of the constraints foreseen at the beginning of the project were the additional power supplies needed to make the stepper motors and their drivers operate without having to constantly change batteries, and how the power is supplied to the system, especially the moving components.

C. Relevant Information Learned from Past Courses

The following list outlines the relevant courses and the material they contain that were used throughout the project's lifecycle.

- ECE 70 - Engineering Computations
  - C programming
- ECE 85 - Digital Logic Design
  - Developing PCB for stepper motor connections
  - Any state diagrams needed for logic between processes
- ECE 90 - Principles of Electrical Circuits
  - Developing the power scheme and parameters for the stepper motors, stepper motor drivers and track system
- ECE 106 - Switching Theory and Logical Design
  - Developing flow charts and block diagrams of the overall system
- ECE 118 - Microprocessor Architecture and Programming
  - Recursion programming and algorithms developed for shortest path
  - Determining how to store memory in a microcontroller
  - Working with GPIO connectors on a microcontroller
  - Using stepper motors, their drivers, and an additional power supply to operate the motors with a microcontroller

  - Programming push-buttons on a microcontroller
- ECE 122L - Micropython Lab
  - Setting up the wireless communication on the specific HUZZAH32 microcontroller board chosen to be the wireless transceivers for the stepper motors.
  - The Python language that is used to program the HUZZAH32 microcontroller board.
- ECE 146 - Computer Networks
  - Wireless connection between systems and how data is transferred over this connection
- ECE 174 - Computer Architecture and Organization
  - Knowing how memory is set up and the different methods of accessing it for the storage and access of the image taken from the camera.
- ECE 178 - Embedded Systems
  - Development of algorithms
  - Basic foundation of and general layout of hardware in microcontrollers, as well as the IDE that is specific to it
  - Interfacing with peripherals on the microcontroller
  - SPI and I^2C communication

Based on the courses provided in the previous list, the following information was extracted from the knowledge gained while taking these classes over the last 3-4 years. Because the following information is from notes taken during these courses, there are no sources provided for where it was learned. More in depth research obtained from external sources and their references are located in the *Project Description and Boundaries* section of this report.

1) Data Storage: The images taken from the camera need to be saved to the microcontroller in order to allow the image processing program to access them. Due to knowledge gained from the *Computer Architecture and Organization* class, attempts at adding memory were made in order to account for the minimal memory provided by the 1 MB of flash. However, after ample time was spent trying to get it to work, this was abandoned in order to focus on delivering a working prototype that met our simple success criteria. Because virtual memory was not able to be added, only the flash memory was used to run our entire system which led to design changes in our algorithms. For example, only 80 rows of pixels could be stored in array during a given processing cycle. This means that one-third of the image is processed at a time and then overwritten by the next one-third instead of being able to process all image data at once.

2) Translate Detected Markings into Coordinate System (Stepper Motor Rotations): Through team collaboration, it was decided that each stepper motor rotation would represent one pixel length. This is how a coordinate system can be developed. For example, if the image processing algorithm picks up a mark that is 56 pixels from the left of the image, and 178 pixels from the top of the image, this would result in 56 partial rotations (to a specific degree that has yet to be determined) to the right on the x-axis motor and 178 partial rotations (to a specific degree that has yet to be determined) down on the y-axis motor. As mentioned, further configuration at the time of testing needs to be done towards how many degrees of rotation the stepper motor would need to rotate in order to represent 1 pixel length. In order to line up the physical layout of the board with a 320x240 image window, indicators are placed on the board in order to know where the image area actually is.

3) Stepper Motor System Design: The stepper motors themselves have 6.35mm teeth bore flanges. These "teeth" are used to grip the pulleys that drive the movement of the linear motion system. In order to rotate the stepper motors. Specific stepper motor drivers (DM542T) are used in order to drive them. This stepper motor "system" is designed on a PCB in order to minimize odd connections and spaghetti wires in the final product. Also attached to these stepper motor system's PCBs are an Adafruit HUZZAH32 Feather Board microcontroller that receives the instructions needed to make the stepper motors rotate the desired distance.

4) Microcontroller Involvement: The process of learning the hardware and IDE of a microcontroller, which was gained from the Embedded Systems course, was implemented with the PSoC 6 microcontroller. Their IDE, Modus Toolbox, is an all-in-one hardware configurator, debugger, and program downloader that is based off Eclipse. The hardware configuration is very straightforward when compared to other programs like Quartus and makes developing and implementing a system much easier with everything in one place. The programs are written in C and any hardware designs that are generated create structures that can be used with provided API's(functions and macros) in order to interact with said hardware. Development of the C programs used in the system took experience gained from the Engineering computations class to employ modular programming and a more optimized flow.

5) Shortest Path Algorithm: The original plan to create a shortest path algorithm came from the knowledge of Dijkstra's algorithm gained from the *Computer Networks* course. This idea was scrapped, however, due to the time constraints on the project. An algorithm that finds the outer-most edges of the markings on the whiteboard, then erases the markings in a serpentine-like pattern was chosen instead.

6) Image Processing: One of the main components of the Smart Eraser is its image processing capabilities. The PSoC 6 is fed an image through a wired SPI and I^2C communication with the camera and store that image in memory. Once the image has completed its processing and the array of the image data needs to be sent to the HUZZAH32, it does so through a UART serial connection to the board. All of these methods of communication were learned in the *Embedded Systems* course. The rest of the image processing algorithm and other information pertaining to it had to be researched outside of the course information learned.

7) Wireless Connectivity: The instructions for the stepper motors need to be sent wirelessly from the PSoC 6. The chat room program that was assigned in the *Computer Networks* course was used as a template for the wireless transmission of these instructions between the transmitter HUZZAH32 and the two receiving HUZZAH32s. This program was made with the MicroPython programming language, and was also used as an assignment in the *Micropython Lab* course.

Additional research pertaining to these topics and how they were implemented in the project are in the *Project Description and Boundaries* section of this report.

D. Standards

Based on the components and parts that were used in this project, the following standards were found to apply, and were followed during the creation and implementation of the Smart Eraser.

- IEEE 802.15.3e-2017 - IEEE Standard for High Data Rate Wireless Multi-Media Networks–Amendment 1: High-Rate Close Proximity Point-to-Point Communications
  - Applies to the wireless communication and data transfer between components in the system [21]
- IEEE Editorial Style Manual
  - Standards to follow when writing this Project Charter and any other reports that were written for this project [22]
- NEMA Standards Publication ICS 16 Motion/Position Control Motors, Controls, and Feedback Devices
  - Applies to the stepper motors being used and their operation [23]
- SPI Communications
  - Defacto standard

- I^2C Communications
  - Defacto standard
- ITU-R Recommendation BT.709
  - Applies to the coefficients used in the grayscale algorithm conversion [24]
- RS232 Serial Communications standard
  - Applies to the UART communication between the PSoC 6 and HUZZAH32 [25]

V. Project Description and Boundaries

This section contains a list of the major and minor components that are used in this project, as well as a quick overview of each of these components and how they play a role in the Smart Eraser project, and some boundaries that were overcome in order to ensure the success of this project. For a more detailed description of the procedure followed and the actual implementation of each part, see the *Analysis and Design* section of this report.

The following lists contain the major and minor components to be used in this project. More details about each major component's specifications and model numbers can be found in the *Equipment and Budget* section of this report.

A. Major Components

- PSoC 6 Development Kit
- Arducam
- Stepper motors
- Stepper motor drivers
- HUZZAH32 boards
- PCBs
- Rechargeable 9V batteries
- Linear motion tracks (x & y direction)
- Linear motion carriages (x & y direction)
- Timing belt
- Timing belt pulley flange
- Eraser
- Whiteboard
- Logic analyzer

B. Minor Components

- Wire jumpers
- Stepper motor mounting brackets
- Various screws, nuts, and bolts
- Wooden plywood board (for mobile prototype)
- Wooden frame (for mobile prototype)
- Wheels (for mobile prototype)

C. Software Components

1) SPI Communication: SPI communication is used in order to talk to the SPI interface of the Arducam-5MP- mini- plus. The PSoC 6 acts as the master and the Arducam acts as the slave. In order for successful communication between the the two, the PSoC 6 needs to send a command byte where the MSB is either a 1 for 'write' or 0 for 'read' and the other 6 bits represent an address in the Arducam. If the command bytes sent is a write command, the byte sent directly after is the data to be written to the register address.

If the command byte sent is a read command, the byte sent directly after does not matter, often referred to as a dummy byte. The purpose of the SPI communication is to initiate captures from the image sensor and to then poll register 0x41 for a capture complete signal of 0x08. Once a capture complete signal has been read, transferring of the image data can ensue. These processes are shown in the following figures.

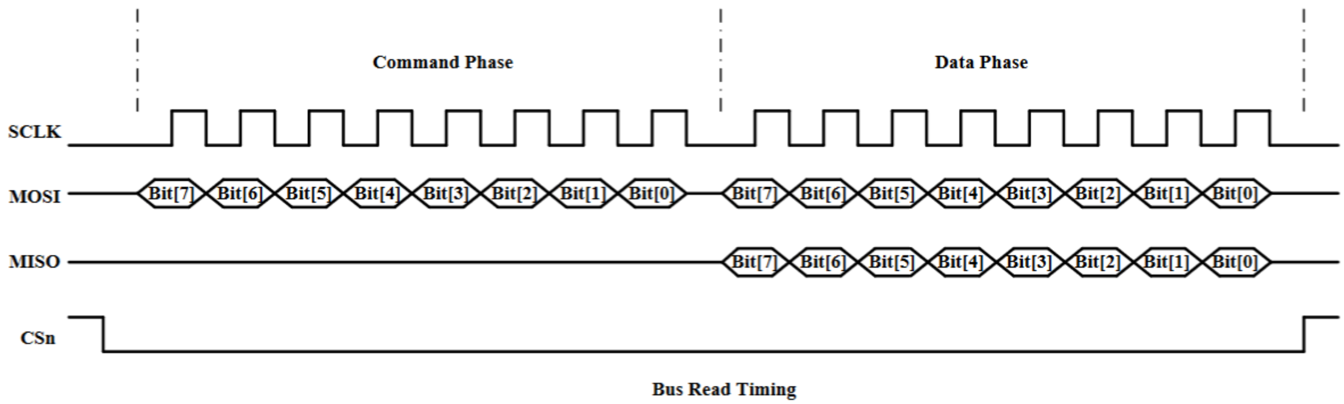


Fig. 2: Bits corresponding to a system SPI read [1]

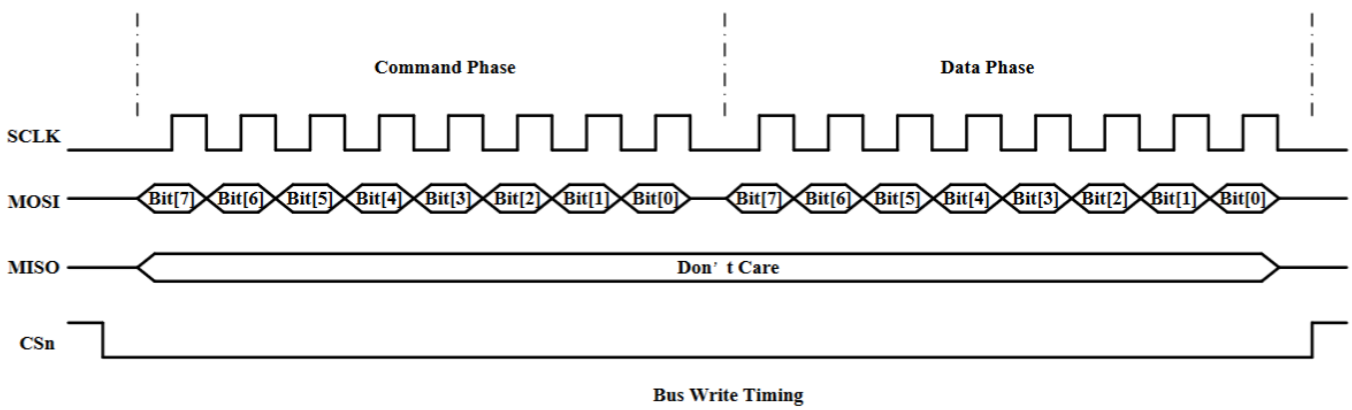


Fig. 3: Bits corresponding to a system SPI write [1]

2) I^2C Communication: The purpose of including I^2C communication is both to initialize and configure the image sensor of the Arducam-5MP-mini-plus as well as providing an interface between the PSoC 6 and HUZZAH32 microcontrollers. The figure below provides and example I^2C exchange between a master and slave where the master is the PSoC 6 and the slave is the Arducam. The information provided from Arducam is slightly incorrect, as most developers incorrectly provide an 8-bit slave address when it should be a 7-bit. The correct address is 0x3C where a read/write bit is tacked on by the microcontroller. So, if its a write operation the slave address would be 0x3C plus a write-bit(0), making the slave address 0x78 (0x3C with a zero bit tacked onto the front of the LSB equals 0x78) and the same applies to a read operation which adds a read-bit(1). This means that there is only one slave address and not separate addresses for read and write operations.

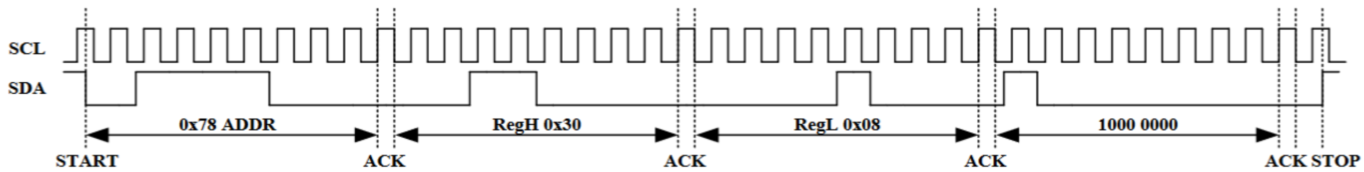


Fig. 4: Bits corresponding to a system I^{2}C write [1]

3) Grayscale Conversion from RGB565: In order to detect objects in an image, the image needs to be converted from RGB to grayscale. This is because the result of the sobel detection algorithm compares light intensity changes against a threshold value. Using RGB pixel information does not provide a way for the intensities to be correctly compared because the R, G, & B values are used to depict color and not to represent light intensity. By converting to grayscale you are creating an intensity value from the pixel information on a scale from 0-255. This can be done by using the grayscale luminance conversion algorithm which assigns weights to each value (R, G, & B) using coefficients from CITE BT.709 :

$$Gray = (Red \times 0.2126 + Green \times 0.7152 + Blue \times 0.0722)$$

However, before this algorithm can be used, the red, green, and blue intensities need to be separated from the 16 bit value and shifted to an 8 - bit scale, hence the name RGB565. Bits[15:11] represent red intensity, bits[10:5] represent green intensity, and bits[4:0] represent blue intensity.

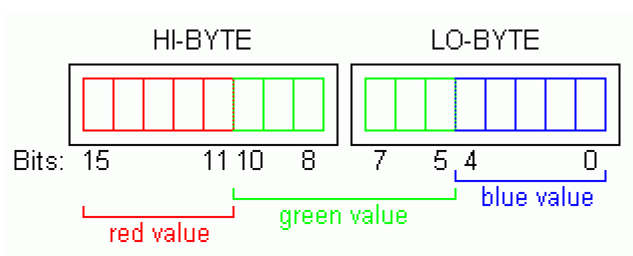


Fig. 5: How an RGB565 pixel value is stored in memory [2]

4) Sobel Edge Detection: After Grayscale pixels have been generated, a sobel edge detection algorithm is used in order to detect edges in an image. This is done by convolving sobel kernels, one vertical one horizontal, through the array of gray pixels. This is shown in the following figure. By doing this, you can find the gradients in both the Y and X direction which is eventually be used in PythagorasâĂŹ Theorem to determine the magnitude of the gradient at that pixel location. The magnitude value is then compared against a threshold value (this threshold value can be altered to fit one's needs).

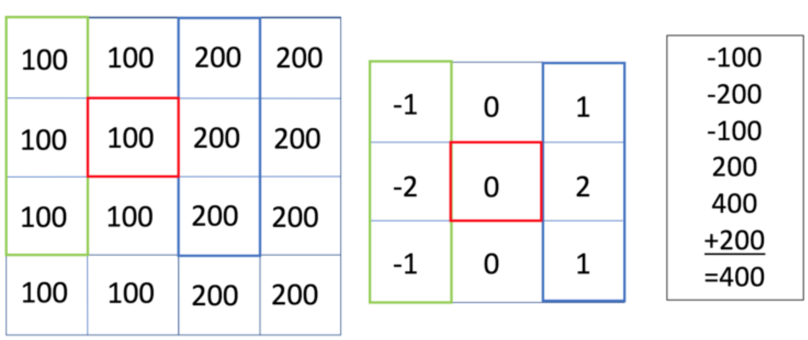


Fig. 6: Kernel convolution used in sobel edge detection [3]

5) Socket Programming for Wireless Communication: Due to malfunctioning hardware concerning the Arduinos and the Kuman wireless transceivers, the original idea for the wireless communication system had to be scrapped. The new concept is to use the HUZZAH32 Feather boards as the wireless transceivers that send and receive instructions for the stepper motors to execute. These boards can be programmed with either Micropython or Arduino programming languages, but in this project, they use Micropython to perform their tasks. One Feather board is initialized as a transmitter (server) that connect to the PSoC 6 via an I^2C connection and take in the stepper motor instructions, then send these instructions to the stepper motors. Two other Feather boards act as receivers (clients); one is attached to each stepper motor, and receive the instructions for the stepper motors. They then give them the directions to move according to the instructions received.

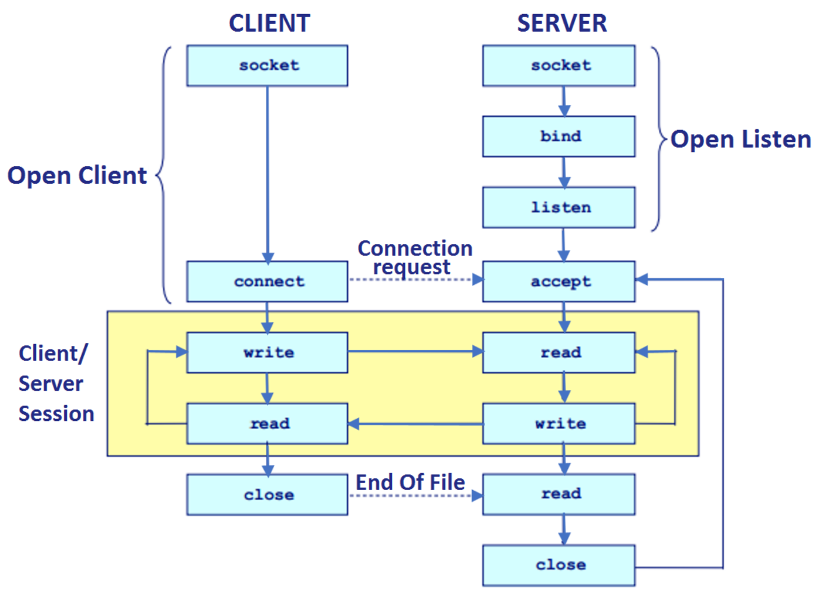


Fig. 7: Socket communication protocol between server and client [4]

6) Stepper Motor Operations: In order for the stepper motors to move, they must be programmed to receive the right signals at the right time to move the magnetic coils within the motor properly. They were programmed in Python and connected to the Feather boards, which provides the necessary signals to allow the motors to move. Figure 8 shows the different steps and signals that need to be sent along the 4 leads connected to the internal magnetic coils in order to rotate them CW (clockwise) and CCW (counter-clockwise).

FULL STEP 2 PHASE-Ex.,
WHEN FACING MOUNTING END (X)

STEP	A	B	A\	B\		CCW
1	+	+	-	-	↓	↑
2	-	+	+	-		
3	-	-	+	+		
4	+	-	-	+	CW	

Fig. 8: Rotation instructions for the stepper motor [5]

7) Moving the Eraser to the Markings: In order to move the stepper motors to the markings on the whiteboard, there are two algorithms: one detects the outermost edges of the markings, which essentially creates a “square” surrounding the markings, and the other creates a serpentine-like path within the square in order to eraser the markings. Figure 9 shows the concept behind these two algorithms, if the whiteboard was divided into an 8x8 matrix. Figure 10 shows the flowchart of the *square_detection.c* program that detects the square around the markings, and Figure 11 shows the flowchart of the *serpentine.c* program that moves the eraser to erase the markings.

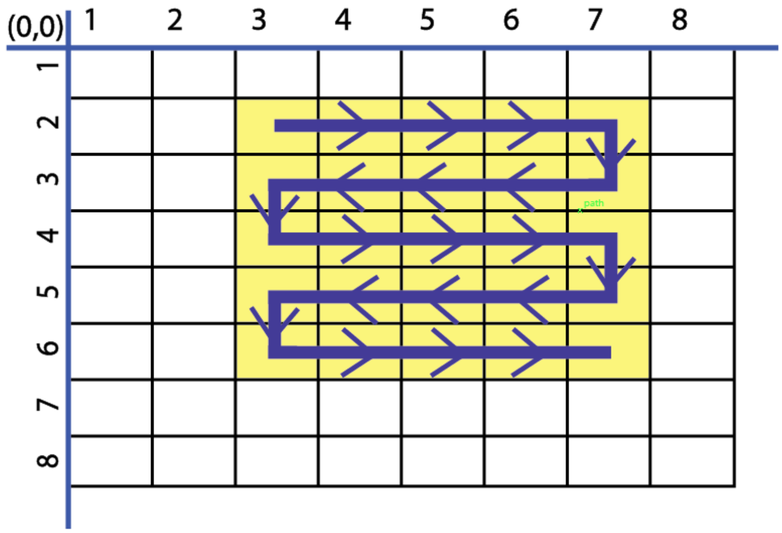


Fig. 9: Concept behind the two algorithms that move the eraser to the markings

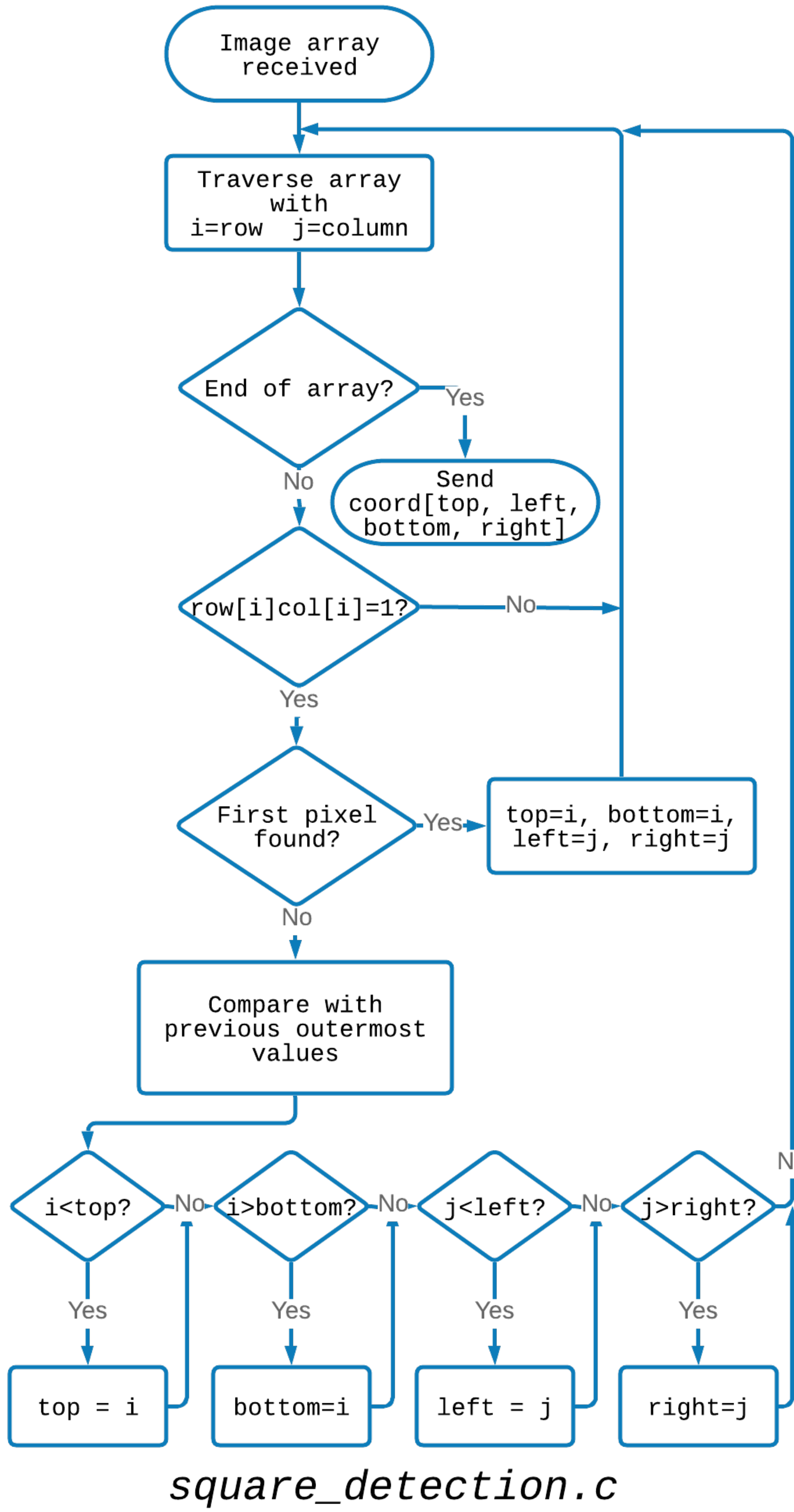


Fig. 10: Flowchart of the logic behind the *square_detection.c* program

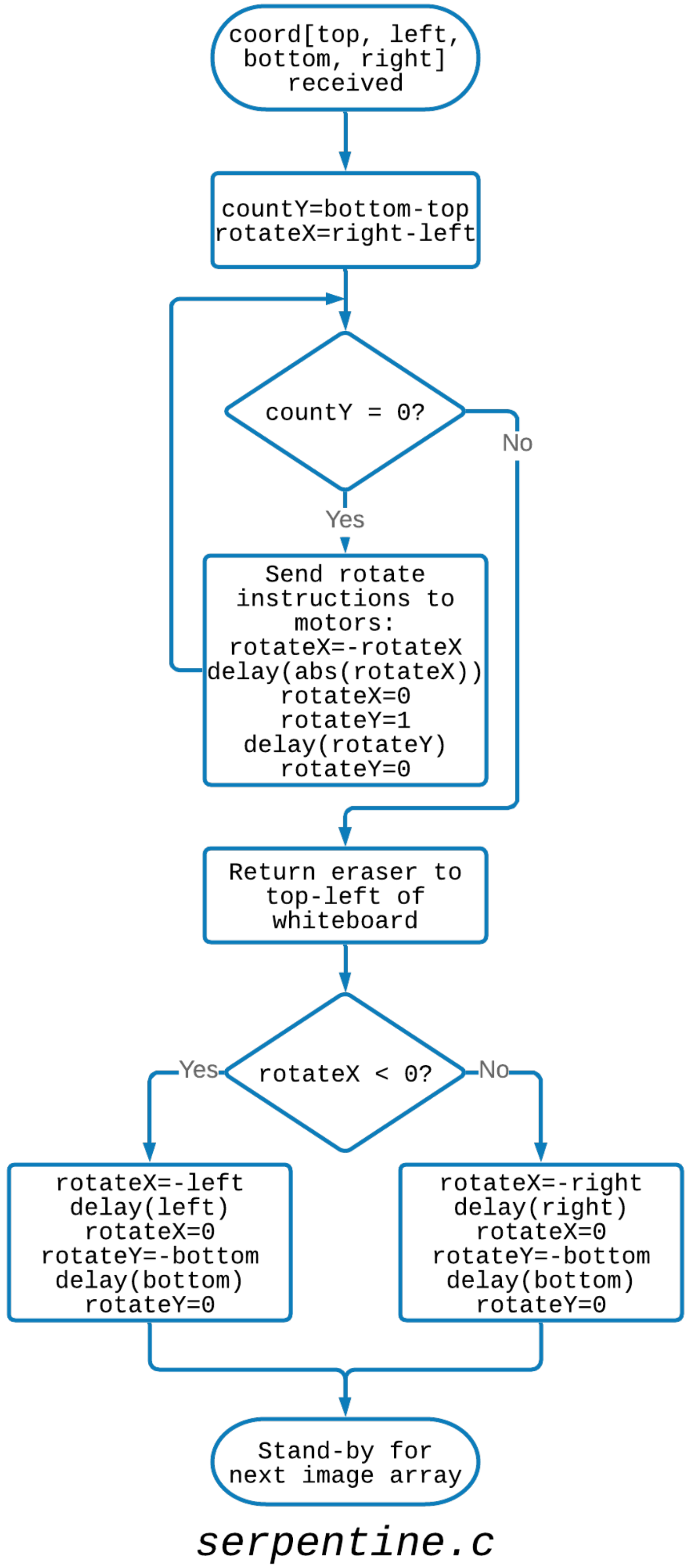


Fig. 11: Flowchart of the logic behind the *serpentine.c* program

D. Physical Technological Components

1) Arducam: The camera is the input that drives the functionality of the rest of the system. It provides high enough quality for future improvement, but for the purposes of this project, only a 320x240 resolution is needed. The current design has the camera directly connected to the PSoC 6 microcontroller through SPI and I^2C communication to both configure the image sensor (I^2C) and initiate data transfer (SPI). Although it is referred to as a camera it is more accurately defined as a module which has an I^2C interface to configure the image sensor and a SPI interface to handle data transfer.

Fig. 12: The Arducam Mini Module 5MP OV5642 Camera [5]

2) PSoC 6 Microcontroller: Due to the processing capabilities that the Smart Eraser needs to be able to handle, and due to the peripheral devices that need to be attached, the PSoC 6 was found to be the best microcontroller to use for the brains behind the mechanism. The PSoC 6 not only allows for the straightforward programming of peripheral devices attached to it, but it also has a dual -core processor built into it which can handle the programming that needs to be done for the Smart Eraser to work as intended. Within the PSoC 6 are Serial Control Blocks(SCB) which can be configured to suite the projects needs, whether it be I^2C, SPI, or UART. The SCB is the gateway to both the Arducam and HUZZAH32. There is also a TFT display attached to the microcontroller that allows for viewing of any image captured and any image processing results. Finally, SW2 is used to initiate the process from capture to erasing.

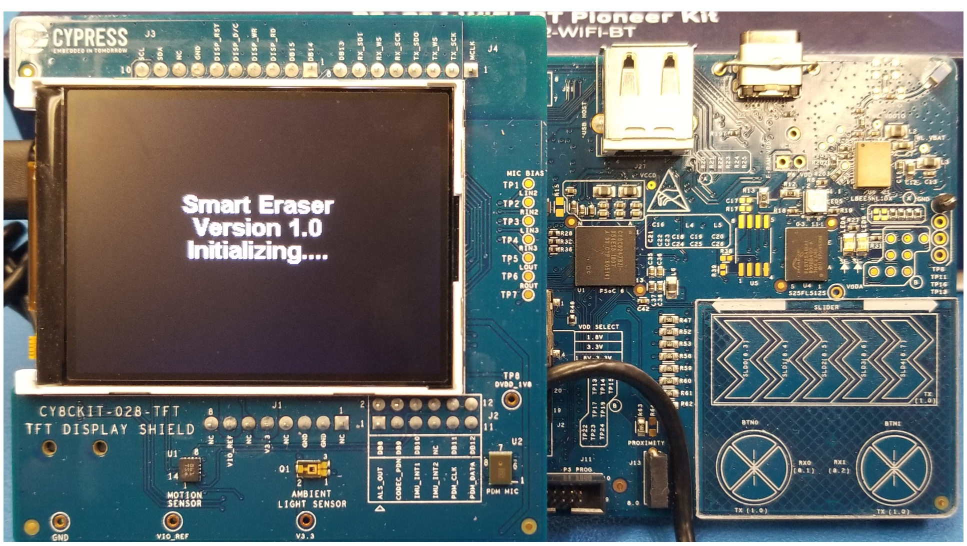


Fig. 13: The PSoC 6 microcontroller by Cypress

3) Stepper Motors: The stepper motors allow the eraser to move along the track and linear motion systems that are mounted to the wall above and below the whiteboard. Because of the weight of the components chosen to move along the track, a stepper motor with enough power and torque to move said components is needed. With this in mind, the NEMA 23 CNC Stepper Motor (with 1.26Nm holding torque, 1.8 degree step angle, 2.8A rated current per phase, 0.9âĎę resistance, and 24-48V driving voltage) was chosen for the Smart Eraser. It is connected via jumper wires to the stepper motor drivers that allow their movement. These require additional power, and this power is supplied via three 9V rechargeable lithium batteries, which is connected through the PCB that was created for the stepper motor connections.

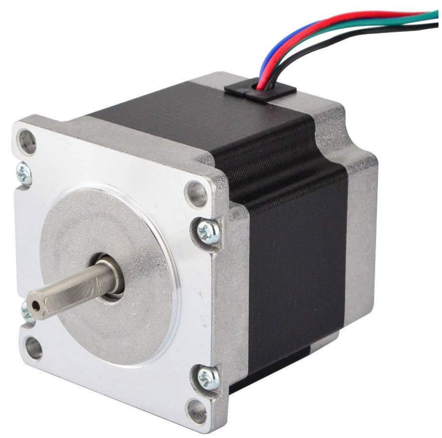


Fig. 14: 4-lead bipolar NEMA23 stepper motor [6]

4) Stepper Motor Drivers: The stepper motor drivers are an intermediary device between the stepper motors and the PCB. The drivers need to specifically work with the stepper motors that were chosen. Therefore, the stepper drivers that are used for the NEMA 23 stepper motors is the Full Digital Stepper Driver, model number DM542T. This model was not only chosen due to its compatibility with the NEMA 23 model stepper motors, but also because of its capabilities to drive the motors at lower noise, with lower heating, and with smoother movement. Like the stepper motors, the drivers need additional power, and they receive it through the PCB via three 9V rechargeable lithium batteries.

The table in Figure 15 shows the correct parameters for the drivers that need to be kept in mind when the additional power is connected to them via the PCB, and the table in Figure 8 shows the specific ports that need to be driven by a high voltage value in order to allow the motor to actually rotate clockwise or counterclockwise.

Parameters	DM542T			
	Min	Typical	Max	Unit
Output Peak Current	1.0	-	4.2 (3.0 RMS)	A
Input Voltage Logic	+20	+36	+50	VDC
Signal Current Pulse	7	10	16	mA
input frequency Pulse	0	-	200	kHz
Width	2.5	-	-	uS
Isolation resistance	500			MΩ

Fig. 15: Limited parameters of the DM542T stepper motor drivers [7]

5) HUZZAH32 for Stepper Motor Wireless Communication: The HUZZAH32 by Adafruit is the main communication hardware for the stepper motors. The transmitter board is physically connected to the PSoC 6, and it contain the code to send the instructions received from the PSoC 6 to the receivers. The receivers contain the necessary code to receive the instructions wirelessly, and they also act as the controller for the stepper motors to move once they have their instructions.

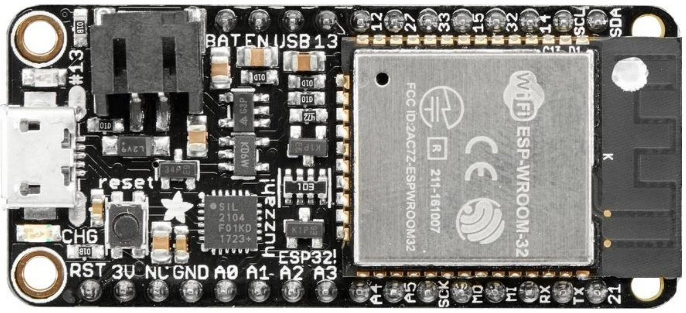


Fig. 16: Adafruit's HUZZAH32 Feather Board with ESP32 chip [8]

6) PCBs: The PCB (Printed Circuit Board) is an intermediary device between the stepper motor drivers and the Feather board, as well as the additional power that is needed to power the motors and their drivers. This PCB was created by the PCB design software DipTrace, and contains connecting pins to ensure spatial efficiency, as well as to make sure all components are connected properly. *Because the wireless system has to be redone, there is no image to place here at the moment*

7) Additional Power: The chosen stepper motor drivers can handle a voltage input ranging from 20 - 50 V, and the stepper motors themselves can handle a voltage input from 24 - 48V. Therefore, the power source needs to be able to drive enough current to generate that range for the PCB, which the stepper motors are connected to. With this in mind, the 9V rechargeable lithium batteries were chosen for the additional power source. Three of these batteries are connected in series on the PCB to provide the additional power needed to driver the motors and their drivers. The batteries can be recharged when they run out of charge, which allow them to be reusable many times for the prototype as well.

Fig. 17: 9V lithium rechargeable batteries [9]

E. Physical Mechanical Components

1) Linear Motion Track System: The model 115RC linear motion track system allows the eraser to move in the x-axis direction of the whiteboard. There are two tracks mounted above and below the whiteboard, and attached to the cassettes with washers and nuts is a third track that allows the eraser to move in the y-axis direction of the whiteboard. The y-axis of the tracking system was pre-drilled at both ends with a 7/32 inch drill bit in order to be placed on the m5 screw of the x-axis carriages. The y-axis was also pre-drilled with a 11/32 inch drill bit in order to attach the housing unit of the stepper motor system. The dimensions of the track systems and the carriages in them are shown in the following figure.

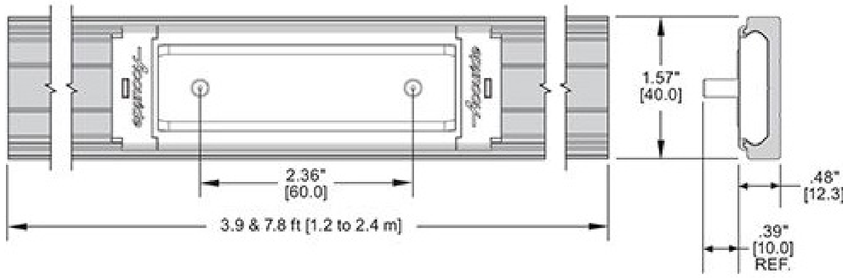


Fig. 18: Linear track system specifications [10]

2) Stepper Motor Mounts: The stepper motor steel mounting brackets are used to actually mount the stepper motors onto the track system, as well as to lift the motors off of the wood backing enough so they are above the track. This allows the pulley system to be attached correctly.

Fig. 19: Mounting brackets for the stepper motors [11]

3) Pulley System: The pulley system consists of two components: a GT2 40-teeth 6.35 mm bore timing belt pulley flange synchronous wheel, and a 5 meter long, 6mm wide GT2 timing belt. Two pulley flanges connect to either side of the track on the x and y axes, and the timing belt wraps around them. One of the flanges on each of the axes connects to the stepper motors in order to allow the belt to actually move. The belt in Figure 21 is black in order to show the details of the teeth, but the actual belt that was used is white.

Fig. 20: Pulley flange [12]

Fig. 21: Pulley belt [13]

F. Prototype Mounting Components

The following figure shows a rough view of what the final prototype will look like when it is mounted on the mobile station. The specifics on what used for this prototype and how it was built are explained after it.

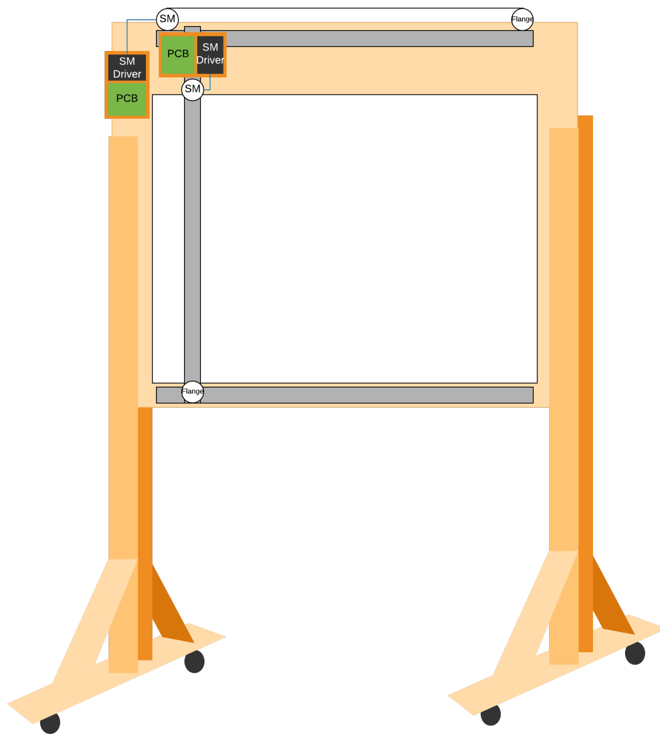


Fig. 22: Rough image of the prototype stand

The main screws used were 3 inch T-25 star-headed screws. These were used to attach the horizontal tracks to the 2x3 posts, which were then attached to the plywood backing of the whiteboard. These screws were also used to mount the whiteboard to the wood backing. Two 2x3 posts were placed behind the tracks in order to give them more stability, as well as lift them off of the board enough to allow the tracks to travel in front of the whiteboard. Three more 2x3 posts were mounted on the back of the wooden backing as well so the plywood that the whiteboard is attached to would not bow or warp due to any changes in the environment. In order to stand the board, two 2x4 pieces of wood were placed at the bottom of the posts holding the stand up. Wheels were then attached to the bottom of this post to make the prototype stand mobile for Projects Day.

Washers and nuts were used to connect the pulley belt to the carriages on the x and y axes, which allow the eraser to move when the stepper motors rotate. Nuts and bolts were then used to connect the stepper motor system to the y axis. A metal L brace was used to mount the other flange to the opposite side of each axis from the stepper motors. Finally, braces were used to clamp the eraser to the carriage that moves across the y axis.

The dimensions of the whiteboard prototype are shown in the next two figures. More information and details about how the system was created, built, and complete can be found in the *Project Prototype* section of this report.

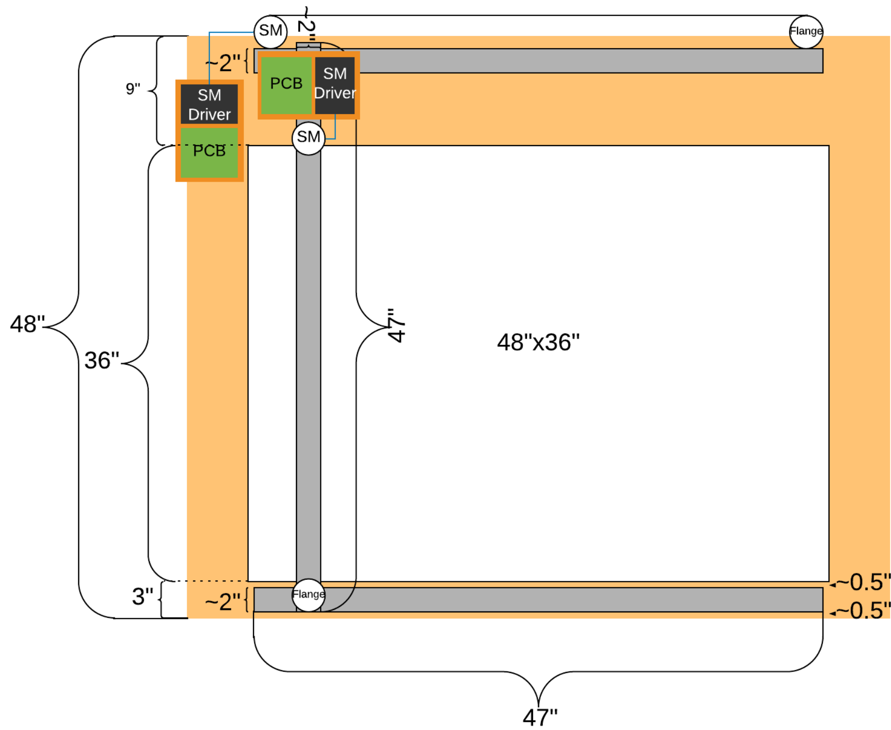


Fig. 23: Front view of the Smart Eraser system with dimensions

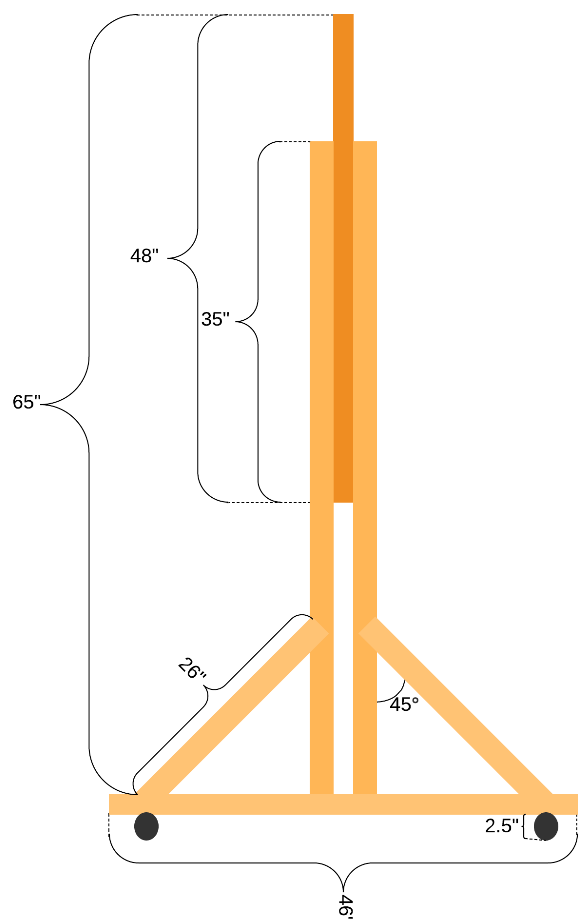


Fig. 24: Side view of the Smart Eraser system with dimensions

G. Boundaries

Because of the problems that were had with the wireless communication via the Arduino and the Kuman wireless transceivers, the wireless communication system has to be completely redesigned using the HUZZAH32 Feather boards. This is a boundary, as this part essentially has to be completely redone with Python, and the PCBs that were designed for the other system are not usable, and must be redesigned as well.

Unfortunately, configuration of the camera sensor proved to be a project all in itself. It involves writing a driver from scratch that needs to configure hundreds of different registers on the Arducam-5MP-Mini-Plus to output an RGB565 image. In order to show the system working, it is possible to take a picture with a phone, run it through software to convert it to the desired resolution and output format and pre loaded onto the PSoC 6. It should be stated that the camera does work, and based off examples from the Arducam company, JPEG image data can be received from the camera. However, due to the complexity of JPEG image data and the fact that the image processing program written is for RGB565, there is nothing that can be done with this data in the scope of this project.

H. Technical Advisor Backgrounds

Dr. Hovannes Kulhandjian, who specializes in wireless communications and networking as well as digital signal processing. He has contributed advice and information pertaining to the wireless connectivity of the camera to the image processing microcontroller. Dr. Kulhandjian was the original mind behind the idea of this project as well. Because of this, he also contributed more specifications and features to implement in the project as its completion progressed.

Roger Moore has many specialties, ranging from tesla coil winding and electrical engineering to microcontroller usage, all stemming from his various research interests. He has contributed input on components that would likely lead to the most success in this project, as well as advice on how to perform the image processing.

VI. High-Level Risks

The following list details the high-level risks of this project and what was done as a precaution against these risks in order to keep the development of the Smart Eraser safe.

- Power system malfunction causing shorts:
  - Ensure a secure connection of the power to the components from the batteries
  - Have all connections of wires complete before the power is applied
  - If connections cannot be completed before power is applied, double-check then triple-check that the pin will be going into the right port
- Moving parts attached to whiteboard:
  - Always stand away from whiteboard when in motion
  - Always callout when it is about to start moving
  - Identify the failsafe wire to unplug in order to emergency shutdown the moving parts
- Use of power tools to create prototype:
  - Have an experienced advisor watching over the construction of the prototype stand
  - Wear safety glasses and gloves when needed
  - Work in a well ventilated area with plenty of room

VII. Milestone Schedule

The following tables show the milestone schedule for the Smart Eraser project over the next two semesters.

Member Assign.	Start-End Date	Description
All	10/12-10/19/18	Complete Smart Eraser Project Proposal to be submitted to DPS Telecom for review.
All	10/12-10/19/18	Finalize the specifics of the budget.
All	10/15-10/19/18	Create the Project Charter rough draft to be turned in.
All	10/16-10/26/18	Draft a more detailed blueprint of the physical Smart Eraser deliverable.
All	10/16-10/26/18	Revise the Project Description; complete for future reference.
All	10/16-10/26/18	Draft the flowchart to show the logical relationships between all connected devices within the project.
All	10/18-10/19/18	Complete bi-monthly update presentation for Senior Design class.
Juan C.	10/26-11/10/18	Complete a block diagram detailing the specific connections between the devices within the project.
Heather L.	10/26-11/15/18	Research wireless communication and protocols to be used.
Heather L.	10/26-11/15/18	Research the camera and how it will send data over WiFi connection.
All	11/1-11/2/18	Complete bi-monthly update presentation for Senior Design class.
Heather L. & Chris Q.	11/1-11/21/18	Research the microcontroller to be used (DE1_SoC).
Chris Q.	11/1-11/21/18	Research the image processing program and what programming language to use.
Juan C.	11/1-12/1/18	Research the mechanical system and the power connection it requires.
All	11/15-11/16/18	Complete bi-monthly update presentation for Senior Design class.
Chris Q.	11/15-11/25/18	Test initial information found on image processing program.
Heather L.	11/15-11/25/18	Test the microcontroller after researching the ports needed for the project.
Heather L.	11/20-12/1/18	Research the coordinate system; converts pixels to stepper motor rotations in the mechanical system.
All	11/29-11/30/18	Complete bi-monthly update presentation for Senior Design class.
All	12/1-12/17/18	Complete the final draft of the Project Charter.
All	12/13-12/14/18	Present Project Charter to Senior Design class, professor, and academic advisor.

TABLE I: Senior Design Semester 1 - Research Phase

Assignee	Start-End Date	Description
All	1/5/19 - 1/6/19	Decide what order to buy the parts needed for the project.
All	1/6/19 - 1/9/19	Complete budget increase justification and send to Dr. Stillmaker.
Juan C.	1/6/19-1/16/19	Help research how to get Linux onto DE1_SoC Board; research the micro SD card needed and the how to implement the QSYS system to run with Linux.
Heather L.	1/15/19 - 1/20/19	Figure out how stepper motors programmed with Arduino.
Chris Q.	1/15/19 - 1/30/19	Research and implement Linux onto the DE1_SoC.
Juan C.	1/16/19 - 1/31/19	Create schematic for wooden frame of prototype stand.
Heather L.	1/20/19 - 2/5/19	Figure out wireless communication with Arduino and wireless transceivers.
Chris Q.	1/20/19 - 1/30/19	Develop system in QSYS that allows the HPS (Linux) to communicate with its peripherals (FPGA components).
Chris Q.	1/31/19	Decided to change microcontroller from DE1_SoC to PSoC 6 under advisement from Roger Moore.
Juan C.	1/31/19 - 2/19/19	Develop more in-depth blueprint of wooden stand measurements and dimensions.
Chris Q.	2/2/19 - 2/7/19	Research PSoC 6 IDE Modus Toolbox.
Heather L.	2/5/19 - 2/18/19	Combine stepper motor code with wireless communication for wireless stepper motor system.
Chris Q.	2/5/19 - 2/15/19	Create system within Modus Toolbox for PSoC 6 to connect to needed peripherals.
Juan C.	2/14/19 - 2/19/19	Determine all different types of screws, nuts, and bolts needed for each individual part of the stand.
Chris Q.	2/15/19 - 2/18/19	Research camera modules suitable for connection with the PSoC 6.
Heather L.	2/18/19 - 3/11/19	Create and assemble PCBs with stepper motors, drivers, the Arduino, and the power needed for all.
Chris Q.	2/18/19 - 3/15/19	Develop SPI communication between the Arducam camera and the PSoC 6 microcontroller.
Juan C.	2/20/19 - 3/15/19	Design mechanical system involving pulleys and linear motion system.
Chris Q.	2/25/19 - 3/2/19	Research image processing program in C with raw data.
Chris Q.	3/2/19 - 3/9/19	Create program that translates a colored RGB565 image to grayscale.
Heather L.	3/4/19 - 3/13/19	Create algorithm to find outermost edges of markings on whiteboard with Chris's processed image data.
All	3/4/19 - 3/14/19	Prepare for Midterm Project Presentations.
Chris Q.	3/9/19 - 3/16/19	Create program that detects edges in the converted grayscale image using sobel edge detection.
All	3/14/19 - 4/12/19	Write rough draft of final project report.
Juan C.	3/15/19 - 3/31/19	Put entire wooden prototype stand together.
Chris Q.	3/16/19 - 4/14/19	Continue trying to configure Arducam camera with PSoC 6.
Juan C.	3/31/19 - 4/12/19	Mount all mechanical parts including tracks, pulley system, and stepper motors to wooden stand.
Heather L.	4/1/19 - 4/3/19	Wireless communication with Arduinos stopped working - come up with alternative.
Juan C.	4/12/19 - 4/15/19	Design power system to connect to stepper motors, their drivers, and the HUZZAH32 boards on the system.

TABLE II: Senior Design Semester 2 - Implementation Phase - Part 1

Assignee	Start-End Date	Description
Heather L.	4/14/19 - 4/16/19	Learn basics of Micropython for new microcontrollers.
Chris Q.	4/14/19 - 4/17/19	Help Heather with learning Python language.
Heather L.	4/14/19 - 4/17/19	Figure out code for stepper motors on new HUZZAH32 microcontrollers.
Heather L.	4/14/19 - 4/21/19	Create algorithm to move stepper motors in a serpentine-like way to erase the markings using the outermost edges found in the previously made algorithm.
Heather L.	4/14/19 - 4/17/19	Create simple messaging program between two HUZZAH32s to test wireless communication.
All	4/17/19	Create and finalize Projects Day poster.
Juan C.	4/15/19 - 4/21/19	Create UART connection between HUZZAH32 and PSoC 6 to send instructions for stepper motors.
Heather L.	4/17/19 - 4/20/19	Combine wireless communication with stepper motor instructions to create wireless stepper motor system.
Chris Q.	4/17/19 - 4/28/19	Help Heather integrate stepper motor movement algorithms and image processing program.
Juan C.	4/17/19 - 4/26/19	Research motion detection system with ultrasonic sensor and HUZZAH32 board.
Juan C.	4/20/19 - 4/22/19	Test power system on breadboard and order necessary parts.
Heather L.	4/20/19 - 4/22/19	Figure out angle of stepper motor rotations needed to move one “pixel” length of the image taken of the whiteboard for coordinates.
Juan C.	4/22/19	Mount power system to wooden stand with HUZZAH32 boards and stepper motor drivers.
Heather L.	4/22/19 - 4/28/19	Combine standalone algorithms, written in C with a dummy matrix, with the actual image data that will be processed in order to give correct instructions to stepper motors for whiteboard.
All	4/22/19 - 4/30/19	Think of Projects Day, what to talk about, and how to present to the audience.
Juan C.	4/25/19 - 4/27/19	Decide how to connect to system as a failsafe for someone standing too close to moving parts.
Juan C.	4/28/19	Mount ultrasonic sensor to prototype with indicator LEDs.
All	5/2/19 - 5/9/19	Create Final Project report presentation and practice!
All	4/30/19 - 5/9/19	Write final draft of Final Project report.
All	5/7/19	Projects Day!
All	5/9/19	Final Project report due!
All	5/(?)/19	Final Project Presentation day!

TABLE III: Senior Design Semester 2 - Implementation Phase - Part 2

VIII. Test Plan

The following Test Plan was made to test the features of the Smart Eraser as they are completed. The test plan includes what feature is being tested, who is in charge of creating that feature, and who is in charge of testing that feature. It then lists the success criteria, which is what determines if the test was a success, and the stopping criteria, which is what determines if a test needs to be stopped.

Smart Eraser *Test Plan*										
	Responsible Person	Date of Test	2/1/2019	2/1/2019	3/5/2019					
PSoC 6 Hardware development	**Chris**		**v1.0**	**v1.2**						
If the hardware is unresponsive or begins to smoke										
Configuration of GUI interface	**Chris**									
The TFT display shows what is expected from the program										
Image is correctly displayed on TFT display	Chris		Image was correctly display							
screen is responsive to commands made in program (adding text, changing image, etc.)	Chris		Program hung	Stepping through program resulted in changes on TFT display						
Configuration of SPI, UART, and I2C Serical Control Blocks (SCB)	**Chris**									
Building of application results in no errors										
I2C: clock rate, TX/RX length, mode, pin configs	Chris		No configuration errors							
SPI: clock rate, TX/RX length, mode, pin configs	Chris		clock speed/data transfer rate incompatible	No configuration errors	changes RX/TX length, no erros in new config					
UART: baud rate, parity, stop bits, pin configs	Chris		baud rate and clock rate mismatch	No configuration errors						
	Responsible Person	Date of Test	2/8/2019	3/1/2019	3/24/2019	3/28/2019	4/3/2019			
SM.py	**Heather**		**v1.0**							
If the stepper motors fail to rotate										
Stepper motor operation	**Heather**									
rotate motors	Heather		success							
Configured to step 320 times horizontally	Heather		too short	too far	success					
Configured to step 240 times vertically	Heather				success					
run motors with prototype fully assembled	Heather				y moves, x doesn't	x moves but struggles	success			

Fig. 25: Test Plan for the Smart Eraser - Part 1

	Responsible Person	Date of Test	3/2/2019	3/10/2019	3/14/2019	3/14/2019	3/18/2019			
server.py & client.py	**Heather**		**v1.0**							
If the HUZZAH32 or PSoC 6 crashes										
Wireless transmission of stepper motor instructions	**Heather**									
The data transfered from the PSoC 6 to the HUZZAH32 is accurately										
Establish socket connection	Heather		program hangs	bind failed	bind failed	successful connection				
Send/receive one byte of data	Heather					data is garbled	works			
send/receive multiple bytes of data	Heather						works			
	Responsible Person	Date of Test	2/19/2019	2/21/2019	2/24/2019	2/26/2019	2/26/2019	2/27/2019	2/28/2019	
conv2gray()	**Chris**		**v1.0**	**v1.1**	**v1.2**	**v1.3**	**v1.4**	**v1.5**	**v1.6**	
If the program crashes										
Conversion of RGB565 to grayscale	**Chris**									
If grayscale image is displayed on TFT display										
Run pixels through conv2gray(), store correctly back to array in main.c	Chris		program hangs	program hangs	pixel values updated in function, not in main	array in main was updated correctly				
Display image on TFT display.	Chris					no image	image is purpleish	image is purpleish	grayscale image displayed	
	Responsible Person	Date of Test	3/2/2019	3/4/2019	3/8/2019	3/9/2019	3/9/2019	3/10/2019	3/11/2019	3/11/2019
sobel()	**Chris**		**v1.0**	**v1.1**	**v1.2**	**v1.3**	**v1.4**	**v1.5**	**v1.6**	**v1.7**
If the program crashes										
Detection of objects in grayscale image	**Chris**									
If white and black image with detected edges displays on TFT display										
Integrate conv2gray() within sobel()	Chris		gray pixel array not updating	returned pointer from conver2gray() inconsistent	Working as expected					
run pixels through sobel(), update image array in main.c with only 0x00 or 0xFF based off threshold	Chris				program hangs	array in main.c not updating correctly	array in main was updated correctly			
Display image on TFT display.	Chris						no image	image is distorted	threshold is off	black and white image with detected edges displayed

Fig. 26: Test Plan for the Smart Eraser - Part 2

	Responsible Person	Date of Test	3/1/2019	3/2/2019	3/2/2019	3/2/2019				12/12/2018
I2C Communication	**Chris**		**v1.0**	**v1.1**	**v1.2**	**v1.3**				
If the hardware is unresponsive or begins to smoke or program crashes										
PSoC 6 and image sensor communication/configuration	**Chris**									
Logic analyzer shows succesful transfer of data										
Use API's to intialize and interact with I2C SCB	Chris		program hangs	program makes it through initialization						
Analyze signal being transmitted with logic analyzer	Chris			Signal is distorted	Signal is structured correctly but receiving NACKs	succesful transfer of data				
Send configurations	Chris					successful transfer of configurations				
	Responsible Person	Date of Test	3/5/2019	3/19/2019	3/24/2019	3/28/2019	4/1/2019	4/5/2019	4/6/2019	4/10/2019
SPI Communication	**Chris**		**v1.0**	**v1.1**	**v1.2**	**v1.3**	**v1.4**	**v1.5**	**v1.6**	**v1.7**
If the hardware is unresponsive or begins to smoke or program crashes										
PSoC 6 and Arducam module communication and data transfer	**Chris**									
Logic analyzer shows succesful transfer and receiving of data										
Use API's to intialize and interact with SPI SCB	Chris		program makes it through initialization							
Analyze MOSI and MISO signals with logic analyzer	Chris		Signal is sent, deassertion occurs at each byte sent, this is wrong	Reconfigured SPI, signal sent correctly						
Send trigger command and wait for response	Chris			no response	no response	no response	no response	no response	no response	New configuration, got response

Fig. 27: Test Plan for the Smart Eraser - Part 3

	Responsible Person	Date of Test	2/28/2019	3/2/2019	3/5/2019					
UART	**Juan**		**v1.0**	**v1.1**	**v1.2**					
If the hardware is unresponsive or begins to smoke or program crashes										
PSoC 6 and HUZZAH32 communication and data transfer	**Juan**									
Logic analyzer shows succesful transfer of data										
Send/receive one byte of data	Juan		no response	no response	changed baud rate, added delay, works now					
send/receive multiple bytes of data	Juan				works					
	Responsible Person	Date of Test	3/20/2019	3/25/2019	3/26/2019					
SquareDetection.C	**Heather**		**v1.0**	**v1.1**	**v1.2**					
If the program crashes										
Locating area around detected objects	**Heather**									
If the top-left most, top-right most, bottom-left most and bottom-right most values are correctly found										
test program (small scale) to test accurateness of program	Heather		Incorrect values returnd	correct values returned						
Intergrate into PSoC 6	Heather			program hung	PSoC 6 succesfully finds window are objects					
	Responsible Person	Date of Test	3/28/2019	4/1/2019	4/5/2019	4/6/2019	4/10/2019			
Serpentine.C	**Heather**		**v1.0**	**v1.1**						
If the eraser fails to erase any marks, then stop all tests										
Correctness of Path that Eraser Follows	**Heather**									
If Eraser moves in serpentine fashion within window determined from SquareDetection.C										
test program (small scale) to test accurateness of program with stepper motor rotations	Heather		incorrect path	incorrect path	incorrect path	scale off	success!			
Eraser movement should be smooth	Heather						movement is smooth			

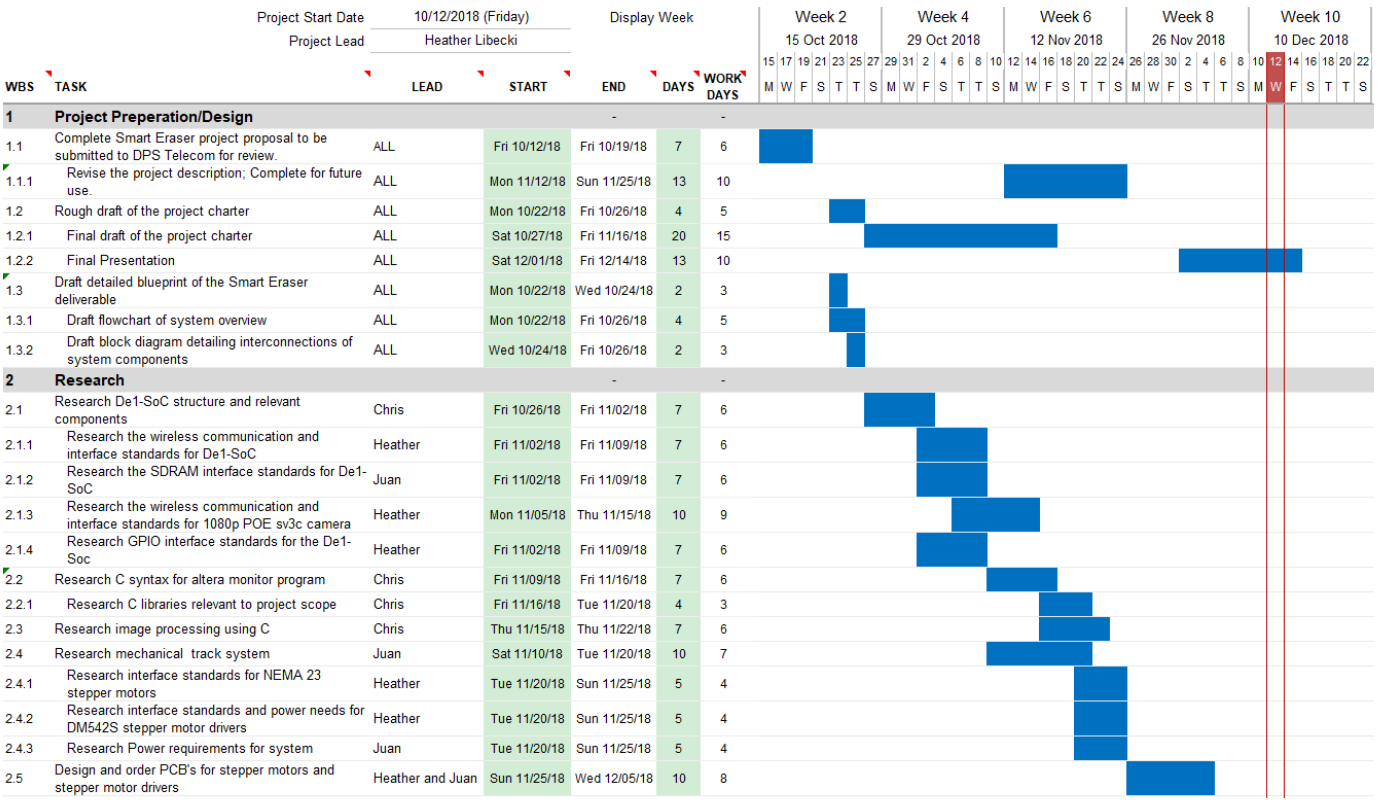
Fig. 28: Test Plan for the Smart Eraser - Part 4

	Responsible Person	Date of Test	4/5/2019							
Movement Detection	**Juan**		**v1.0**							
If the hardware is unresponsive or begins to smoke or program crashes										
System Stops with Detection of Person	**Juan**									
All markings on the board being erased										
A signal from the sensor should be received by the x-axis HUZZAH32 when movement is detected	Juan		success							
A signal from the x-axis HUZZAH32 should be sent to server when movement is detected	Juan									
A signal from the server HUZZAH32 should send an immediate stop command to stepper motor systems	Juan									

Fig. 29: Test Plan for the Smart Eraser - Part 5

IX. Gantt Charts

The following figures show the GANTT chart schedules over the next two semesters. These list the tasks to be completed, who is in charge of what task, and the time duration the task is expected to take.

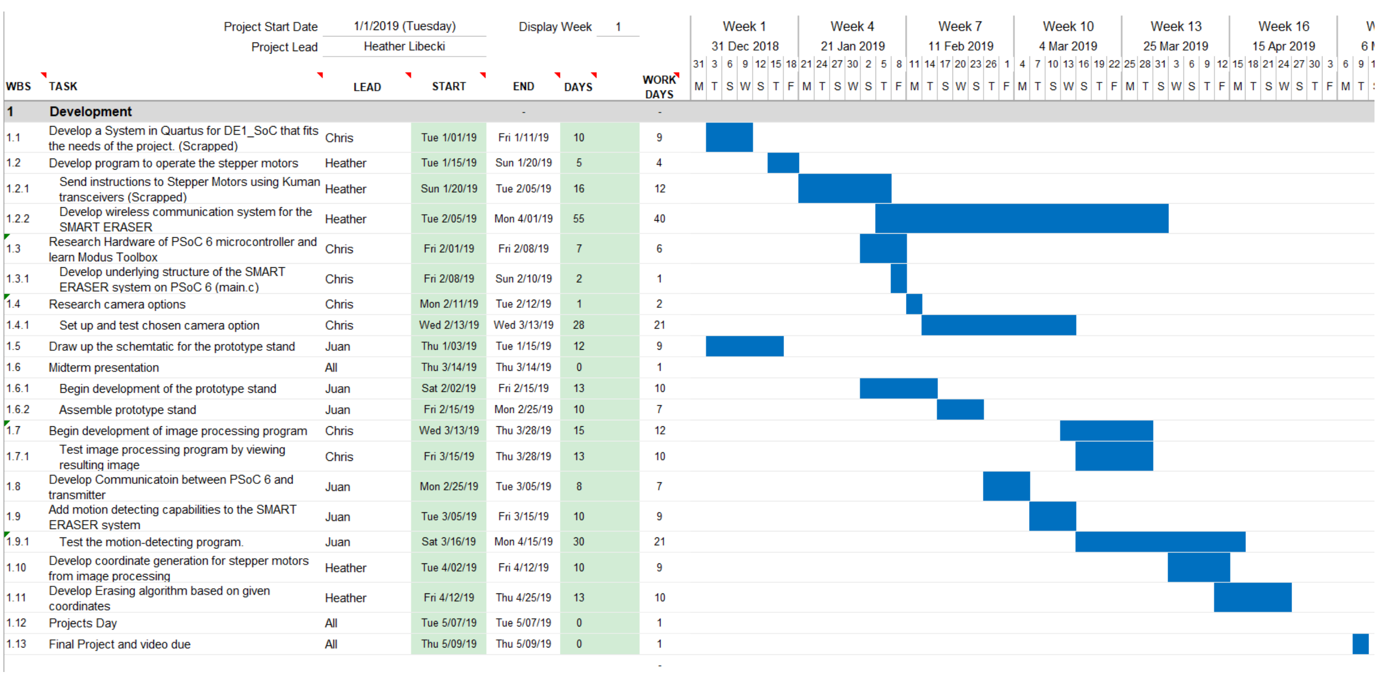


Project Start Date: 10/12/2018 (Friday) — Display Week

Project Lead: Heather Libecki

WBS	TASK	LEAD	START	END	DAYS	WORK DAYS
1	**Project Preperation/Design**			-		-
1.1	Complete Smart Eraser project proposal to be submitted to DPS Telecom for review.	ALL	Fri 10/12/18	Fri 10/19/18	7	6
1.1.1	Revise the project description; Complete for future use.	ALL	Mon 11/12/18	Sun 11/25/18	13	10
1.2	Rough draft of the project charter	ALL	Mon 10/22/18	Fri 10/26/18	4	5
1.2.1	Final draft of the project charter	ALL	Sat 10/27/18	Fri 11/16/18	20	15
1.2.2	Final Presentation	ALL	Sat 12/01/18	Fri 12/14/18	13	10
1.3	Draft detailed blueprint of the Smart Eraser deliverable	ALL	Mon 10/22/18	Wed 10/24/18	2	3
1.3.1	Draft flowchart of system overview	ALL	Mon 10/22/18	Fri 10/26/18	4	5
1.3.2	Draft block diagram detailing interconnections of system components	ALL	Wed 10/24/18	Fri 10/26/18	2	3
2	**Research**			-		-
2.1	Research De1-SoC structure and relevant components	Chris	Fri 10/26/18	Fri 11/02/18	7	6
2.1.1	Research the wireless communication and interface standards for De1-SoC	Heather	Fri 11/02/18	Fri 11/09/18	7	6
2.1.2	Research the SDRAM interface standards for De1-SoC	Juan	Fri 11/02/18	Fri 11/09/18	7	6
2.1.3	Research the wireless communication and interface standards for 1080p POE sv3c camera	Heather	Mon 11/05/18	Thu 11/15/18	10	9
2.1.4	Research GPIO interface standards for the De1-Soc	Heather	Fri 11/02/18	Fri 11/09/18	7	6
2.2	Research C syntax for altera monitor program	Chris	Fri 11/09/18	Fri 11/16/18	7	6
2.2.1	Research C libraries relevant to project scope	Chris	Fri 11/16/18	Tue 11/20/18	4	3
2.3	Research image processing using C	Chris	Thu 11/15/18	Thu 11/22/18	7	6
2.4	Research mechanical track system	Juan	Sat 11/10/18	Tue 11/20/18	10	7
2.4.1	Research interface standards for NEMA 23 stepper motors	Heather	Tue 11/20/18	Sun 11/25/18	5	4
2.4.2	Research interface standards and power needs for DM542S stepper motor drivers	Heather	Tue 11/20/18	Sun 11/25/18	5	4
2.4.3	Research Power requirements for system	Juan	Tue 11/20/18	Sun 11/25/18	5	4
2.5	Design and order PCB's for stepper motors and stepper motor drivers	Heather and Juan	Sun 11/25/18	Wed 12/05/18	10	8

Fig. 30: GANTT chart for Senior Design Semester 1 - Research Phase



Project Start Date: 1/1/2019 (Tuesday) — Display Week: 1

Project Lead: Heather Libecki

WBS	TASK	LEAD	START	END	DAYS	WORK DAYS
1	**Development**			-		-
1.1	Develop a System in Quartus for DE1_SoC that fits the needs of the project. (Scrapped)	Chris	Tue 1/01/19	Fri 1/11/19	10	9
1.2	Develop program to operate the stepper motors	Heather	Tue 1/15/19	Sun 1/20/19	5	4
1.2.1	Send instructions to Stepper Motors using Kuman transceivers (Scrapped)	Heather	Sun 1/20/19	Tue 2/05/19	16	12
1.2.2	Develop wireless communication system for the SMART ERASER	Heather	Tue 2/05/19	Mon 4/01/19	55	40
1.3	Research Hardware of PSoC 6 microcontroller and learn Modus Toolbox	Chris	Fri 2/01/19	Fri 2/08/19	7	6
1.3.1	Develop underlying structure of the SMART ERASER system on PSoC 6 (main.c)	Chris	Fri 2/08/19	Sun 2/10/19	2	1
1.4	Research camera options	Chris	Mon 2/11/19	Tue 2/12/19	1	2
1.4.1	Set up and test chosen camera option	Chris	Wed 2/13/19	Wed 3/13/19	28	21
1.5	Draw up the schemtatic for the prototype stand	Juan	Thu 1/03/19	Tue 1/15/19	12	9
1.6	Midterm presentation	All	Thu 3/14/19	Thu 3/14/19	0	1
1.6.1	Begin development of the prototype stand	Juan	Sat 2/02/19	Fri 2/15/19	13	10
1.6.2	Assemble prototype stand	Juan	Fri 2/15/19	Mon 2/25/19	10	7
1.7	Begin development of image processing program	Chris	Wed 3/13/19	Thu 3/28/19	15	12
1.7.1	Test image processing program by viewing resulting image	Chris	Fri 3/15/19	Thu 3/28/19	13	10
1.8	Develop Communicatoin between PSoC 6 and transmitter	Juan	Mon 2/25/19	Tue 3/05/19	8	7
1.9	Add motion detecting capabilities to the SMART ERASER system	Juan	Tue 3/05/19	Fri 3/15/19	10	9
1.9.1	Test the motion-detecting program.	Juan	Sat 3/16/19	Mon 4/15/19	30	21
1.10	Develop coordinate generation for stepper motors from image processing	Heather	Tue 4/02/19	Fri 4/12/19	10	9
1.11	Develop Erasing algorithm based on given coordinates	Heather	Fri 4/12/19	Thu 4/25/19	13	10
1.12	Projects Day	All	Tue 5/07/19	Tue 5/07/19	0	1
1.13	Final Project and video due	All	Thu 5/09/19	Thu 5/09/19	0	1

Fig. 31: GANTT chart for Senior Design Semester 2 - Implementation Phase

X. Equipment and Budget

The following table lists the components that were bought in order to complete the Smart Eraser, as well as the company that made them and their cost. The costs are listed without taxes and shipping costs taken into account, but the total of the budget at the bottom of the table includes these. The components are listed in alphanumeric order.

Component Name and Model	Production Company	Cost
Aluminum GT2 40 Teeth 6.35mm Bore Timing Belt Pulley Flange	Uxcell	$7.19 x4
Antenna Wireless Transceiver nRF24L01+PA+LNA	Kuman	$13.99
Arducam Mini Module Camera 5 Megapixel OV5647	Arducam	$39.99
Arduino MKR1000 WiFi with Headers	Arduino	$33.95
Assembled HUZZAH32-ESP32 Feather Board with Stacking Headers	Adafruit	$21.95 x3
CNC Stepper Motor Driver DM542T	STEPPERONLINE	$38.99 x2
Li-ion 9V Battery Charger Bay	EBL Official	$12.99
Magnetic Whiteboard 48"x36"	Lockways	$49.99
NEMA 23 CNC Stepper Motors	STEPPERONLINE	$19.99 x2
NEMA 23 Stepper Motor Mounting Bracket 4 pack	Jiuwu	$18.99
PSoC 6 WiFi-BT Pioneer Kit	Cypress	$99.00
Stepper Motor PCBs	PCBWay	$147.00
Various Braces and Glue	Home Depot	$27.46
Various Screws, Nuts, and Bolts	Home Depot	$24.64
Wood Plywood Backing 96"x48"	Home Depot	$39.82
Wood 2"x4"	Home Depot	$12.27 x4
10 Meter GT2 6mm Steel Core White Open Ended Timing Belt	Nineone	$15.89
115RC Cassette with Stainless Steel Bearings	Accuride	$45.00 x3
115RC 47" Aluminum Track	Accuride	$36.22 x3
2" Caster Rubber Wheels with Swivel and Brake	Home Depot	$3.98 x4
9V Battery Clips	BBTO US	$6.25
9V Battery Holders for PCBs	Mouser	$1.90 x10
9V Rechargeable Batteries 6 pack	EBL Official	$25.99 x2
TOTAL COST		**$1,202.13**
Allotted Budget		**$930.00**
Over Budget Cost		**$272.13**

TABLE IV: Cost of components for project

As shown in the table for the budget, this project has exceeded the original expected budget by almost $300. The original estimate was not taking into account the problems that were run into during the course of the project's life cycle. For example, the Arduino and Kuman wireless transceivers had to be changed to the HUZZAH32 Feather boards for the wireless communications because of their inoperability. This and a few other unforeseen circumstances pushed the budget over the limit.

Along with the components listed in the budget, the following resources were also used to complete the project.

Name of Equipment	Type	Description
DipTrace	Software	Designing PCBs
Arduino IDE	Software	For programming and running the Arduinos when they were being used for the wireless communications.
Modus ToolBox IDE	Software	Programming and setting up the system for the PSoC 6.
uPyCraft IDE	Software	Programming the HUZZAH32 Feather Board.
Various connectors and jumper cables	Hardware	For connections between components
Digital Multimeter	Hardware	For testing and recording voltages, currents, and resistances across components.
Solderless Breadboard	Hardware	For testing purposes.
Logic Analyzer	Hardware	For testing purposes.

TABLE V: Equipment used besides the components bought

XI. Roles of Team Members

This section contains a more in depth look at what the specific roles of each team member were throughout this project. Included in this section is each memberâĂŹs areas of expertise, areas they were not experienced in, and what they had to work on the most throughout the completion of the Smart Eraser.

A. Heather Libecki

She had the responsibility of being the project manager for the Smart Eraser project. She also wanted to take charge of the stepper motors and their operations. She has had previous experience with stepper motors and how they work, so she was ready to take on this task. The technical advisor Dr. Kulhandjian wanted the project to be as wireless as possible, so wireless transceivers were decided upon to communicate with the stepper motors from a master controller that would send instructions. She decided to take on the task of creating the wireless communications as well, as she has a personal interest in wireless connections and networking programming. Although she had no experience in wireless communications, she wanted to learn this for the project. Finally, because she was in charge of the connections and movement of the stepper motors, she needed to create the algorithms that would actually allow the stepper motors to move the eraser to where they needed to go based on the image processing that Chris did. She was responsible for the algorithm that would find the outer-most edges of the markings on the whiteboard, then for the algorithm that took those coordinates and created a serpentine-like path, out of stepper motor instructions, that the eraser would need to follow. The following list details her strengths and weaknesses that would come in handy and needed to be addressed for this project.

- Strengths: programming (Verilog, C programming), PCB design, mathematics, debugging, circuit implementation, problem solving, technical writing, public speaking
- Weaknesses: circuitry design, power systems, socket programming with Micropython

B. Chris Quesada

Has experience in working with embedded systems and developing code for different applications. This project will be heavy on the software side, using both *C* and *Python* to take in data, analyze it, and then send an output. He also developed the UART, SPI, and I^2C communications needed for the Smart Eraser system. A solid understanding of arrays, pointers, and how information is stored in memory was applied towards implementing the image processing techniques used to find pixels that represent markings in the digital image. His work in ECE 70, concepts and experience gained in CSCI 41, and research into image processing allowed for the succesful completion of the Smart Eraser image processing capabilities.

Researching SPI and I^2C in Embedded systems along with further research into theses communications along with UART during the Smart Eraser developement led to successful implementations of all three. He also assisted Heather with the developement of the wireless communicaions between the HUZZAH32 microcontrollers.

- Strengths: programming (C, C++, Python), programming concepts (arrays, pointers, structures, data types), embedded systems, developing algorithms, communications(SPI, UART, I^2C)
- Weaknesses: circuitry design, mathematics, public speaking, power

C. Juan Colin

Has experience in working with electrical systems and physical circuit design. He is proficient in the use of problem solving techniques to create a functioning system with given design specifications. His part of the project was dependent on learning the physical mechanical aspects of the design, and how the connected parts will be powered. Therefore, he was in charge of the main mechanical system as well as the wooden prototype stand, and how the power can be supplied to the technological components in order to allow all parts of the system to work properly and move the way they need to. He also used knowledge from previous classes to program a microcontroller to detect the presence of a person in front of the whiteboard in order to provide a safety measure because there are moving parts in this project. He needed to research Micropython from scratch and use that knowledge to accomplish this.

- Strengths: electrical systems, circuitry design, problem solving, power systems, public relations
- Weaknesses: programming (Assembly, Verilog), technical writing and spelling

XII. Analysis and Design

In this section, the implementation of the components that came together to make the Smart Eraser functional will be described.

A. I^2C Communication - PSoC 6 to Arducam

Inter-Integrated-Circuit(I^2C) is a popular method for interactions between processors and slower ICs. For optimal efficiency, short distances should be used so it is mainly for intra-board connections. The naming scheme used to define the components is referred to as a master/slave relationship where there is one master and one or more slaves. Communication between the master and slave is through messages, with the messages being structured in a specific way. A start and stop bit are used to indicate the start and end of a message. After the start bit, an address is given, usually 7-10 bits, followed by the read/write bit. As was the case with Arducam, the data sheets for slave devices often incorrectly list the slave address with the read/write bit included. Using the PSoC 6 APIs for the I^2C Serial Control Block(SCB) added a start and stop bit to the given slave address based on whether it was a read or write. So, after omitting the LSB of the slave address provided from Arducam, communications worked perfectly(0x78 - LSB = 0x3C). This is known as the address frame. There are also 2 data frames that ensue, each a byte wide. Between each frame either an ACK is received from the slave or the communication failed and the ACK bit is set to NACK. The flow of how a message is sent can be seen in figure 32.

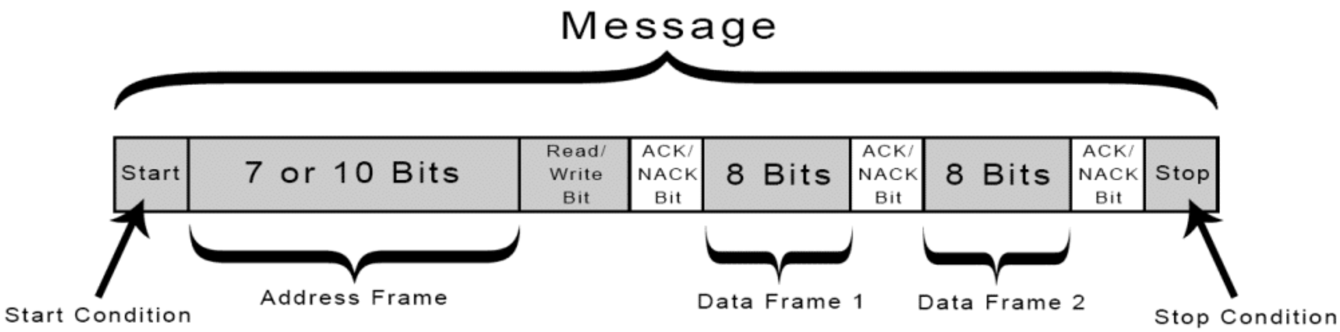


Fig. 32: Simple overview of an I2C message [14]

Some confusion may arise when it is stated that in order to communicate with the Arducam module both I^{2}C and SPI are needed to do so. This is because the Arducam is just that, a module, that has different components. The difference between these two types of communication is that I^{2}C is needed to initialize the image sensor on the module, determining what is transferred to the FIFO storage, located on the module as well. For example, with I^{2}C, the resolution, format, and size of the image can be configured. With SPI, the actual capture and transfer of the image data from that FIFO storage is done. Basically, the I^{2}C can only interact with the image sensor itself and the SPI communication is for operating the entire module. The layout of the different components is shown in 33

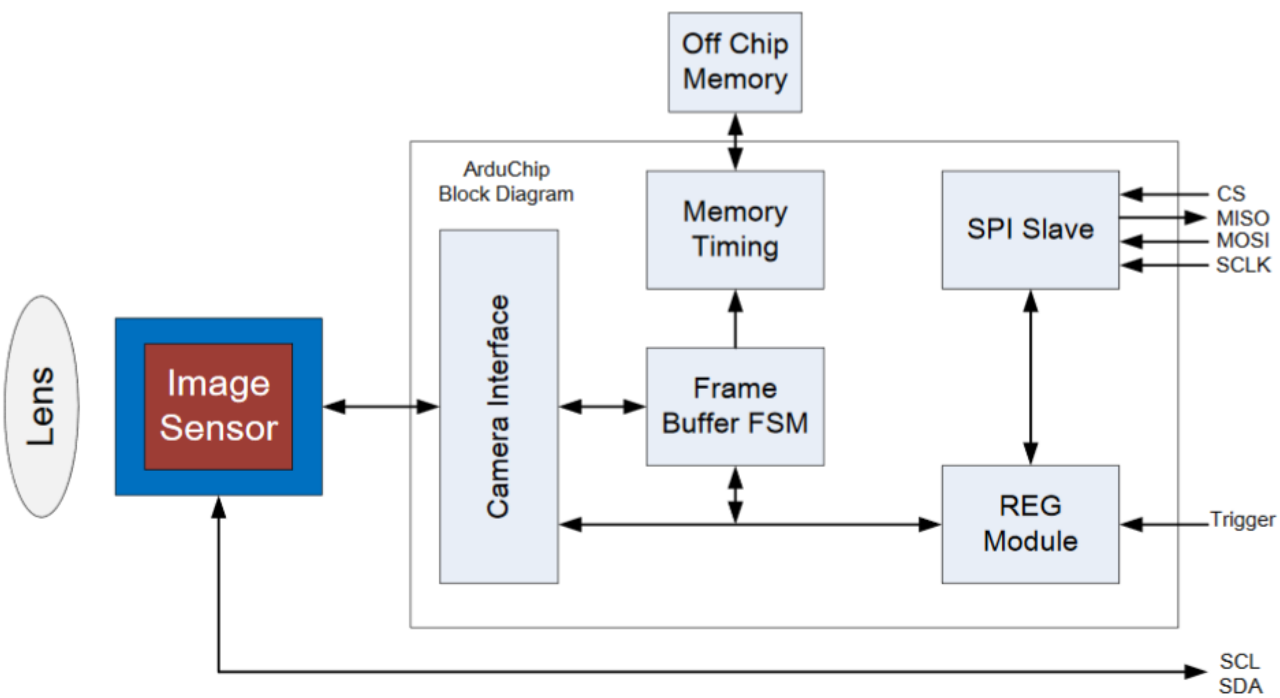


Fig. 33: Arducam module block diagram [1]

Unfortunately due to time constraints, although the I^{2}C communication works correctly, attempts at configuring the image sensor to output RGB565 with a resolution of 320x240 were unsuccessful. This is due to the hundreds of control registers that can be found in the sensor and not knowing which ones need to be set and what values to set in them. This can be a project all on its own. However, the analysis and design will still be explained.

In the file Arducam.C, found in XVI-H, MyCam_Init() uses the Write_I2C() function to send configuration signals to the image sensor of the Arducam module. These configurations are stored in structures in the beginning of the code file. As can be seen, just in the first configuration structure alone, there are over 250 configurations that need to be sent to the image sensor to get a RAW 1280x960 image stored to the FIFO buffer of the Arducam module. This is followed by another, much smaller set of configurations that is meant to resize the image. Due to the inability to set the output to RGB565, an attempt to receive RAW data was attempted and is the reasoning for sending theses configurations. Receiving RAW data requires another component to be added to the image processing, demosaicing, and is currently still in development. The Write_I2C() function uses a combination of APIs provided by Cypress in order to interact with Serial Control Block 3 (SCB3) configured to be a I^{2}C master. It can be found in the section XVI-F.

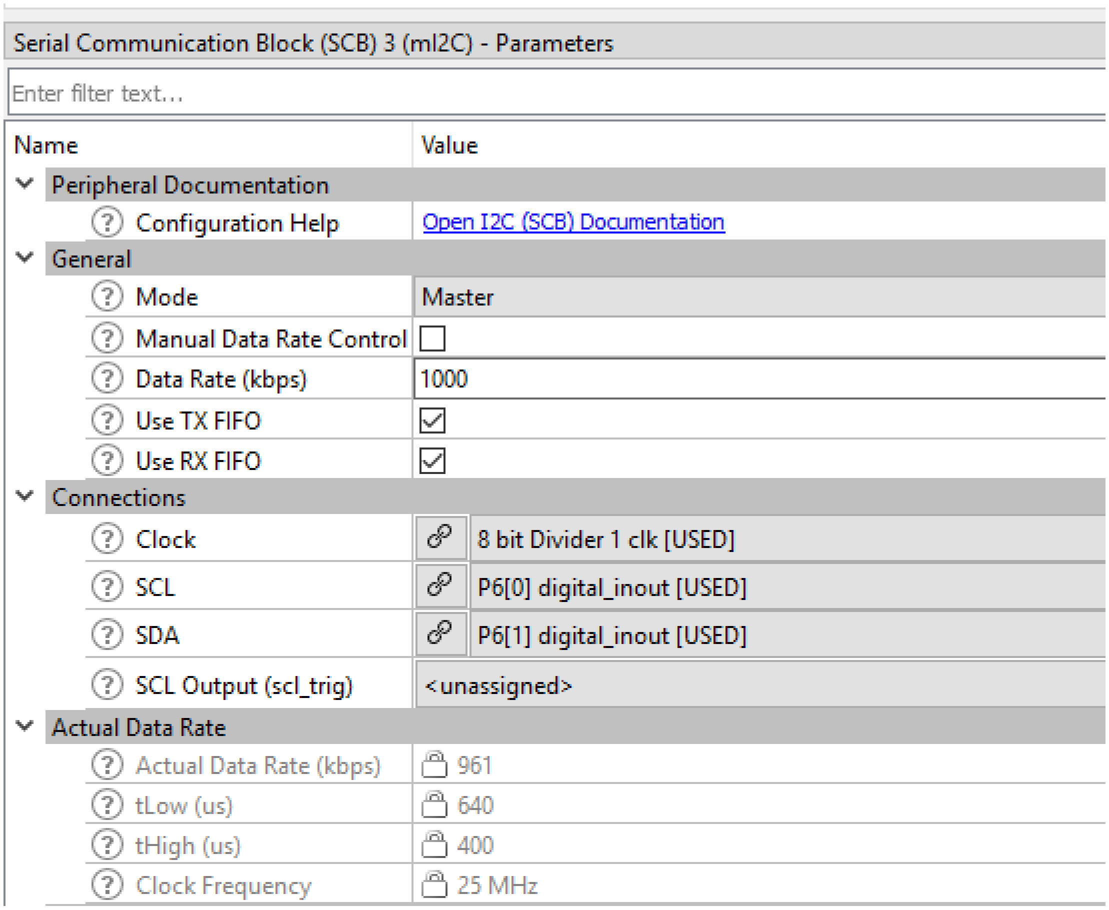


Fig. 34: Hardware Configuration of I2C

B. SPI communication - PSoC 6 to Arducam

Serial Peripheral Communication (SPI) is a common form of wired communication between different systems. The master/slave relationship used in I^{2}C is also used in SPI. There is no official standard for SPI however it is so widely used that it can be referred to as a De facto standard due to everyone operating along the same assumptions. These assumptions include the 4 modes SPI operate in and each mode is a different combination of the clock phase, when data is evaluated, and clock polarity, whether the clock is high or low when not in use. Once the mode has been decided, the slave clock dictates the rate of transaction between master and slave and therefore, the master must be configured accordingly. A Master-Out-Slave-In (MOSI) line feeds data, either MSB or LSB first, to the shift register in the SPI interface of the slave device. For every bit received, the slave will send a bit back to the master along the Master-In-Slave-Out (MISO) line. During a SPI exchange, every bit sent is a bit received and depending if you want to read or write data, the bits received are sometimes irrelevant (writing to slave).

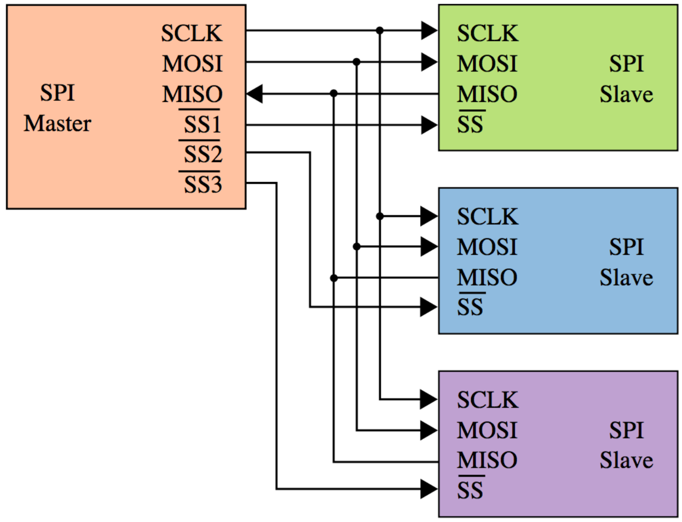


Fig. 35: Simple overview of SPI communication [15]

For the Smart Eraser system, the PSoC 6 is the master and the Arducam module is the slave. According to Arducam's data sheet, the Arducam SPI interface operates in SPI mode 0, meaning CPHA = 0 and CPOL = 0. Therefore the PSoC 6 is configured to operate in that mode as well. Also according the Arducam's data sheet the max frequency of its SPI interface is 8 MHz and therefore the PSoC 6 was configured to send a 7.142857 MHz signal to meet the requirement. Both the RX and TX of the PSoC 6 was set to be 16 bits wide in order to properly communicate with the Arducam SPI interface. An example of what the Arducam needs in order for proper communication can be found in figures 2 and 3. The command byte, sent first, is structured so that the MSB is designated as the read/write byte, leaving the other 6 bits to represent an address in the Arducam SPI interface. The following byte is the data byte which contains specific data if the action is a write. If it is a read SPI transaction then this byte is known as a dummy byte because its just sent to push what needs to be read to master. The reason the TX and RX were set to 16 bits instead of 8 is because when this command and data byte is sent to the Arducam, the CS line needs to be asserted while both bytes are sent. When the PSoC 6 was configured with an 8-bit TX and RX, it would de-assert the CS line between each byte. This caused improper communication between the devices because from the point of view of the Arducam, it was only receiving command bytes.

This can be observed in the Arducam.C file, in any of the functions that use the Write_SPI() function. A packet of 2 bytes is built and then placed into the TX of the PSoC 6 for transfer, which is handled by Write_SPI(). These functions are:

- MyCam_Test(): Tests whether or not the SPI is working correctly
- MyCam_Trigger(): Function that initiates a capture
- MyCam_Check_Capture_Status(): Function that polls the capture ready flag in register 0x41 of the Arducam module.

Code Implementation (Packet assembling):
txBuffer = (((uint16_t)WRITE + (uint16_t)TEST_SPI) « 8) + (uint16_t)TEST_VALUE;

Code is from MyCam_Test() in Arducam.C located in section XVI-H

The Write_SPI() function uses a combination of the APIs provided by Cypress in order to interact with the Serial Control Block 1(SCB1) that has been configured as aa SPI master. It is located in the I2Cmaster.C file found in XVI-D.

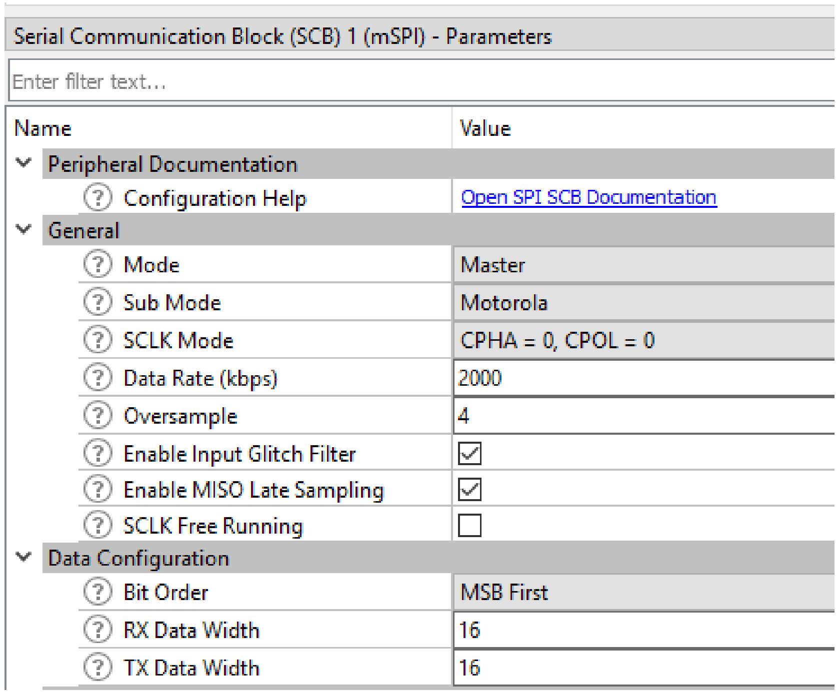


Fig. 36: Hardware configuration of SPI (1)

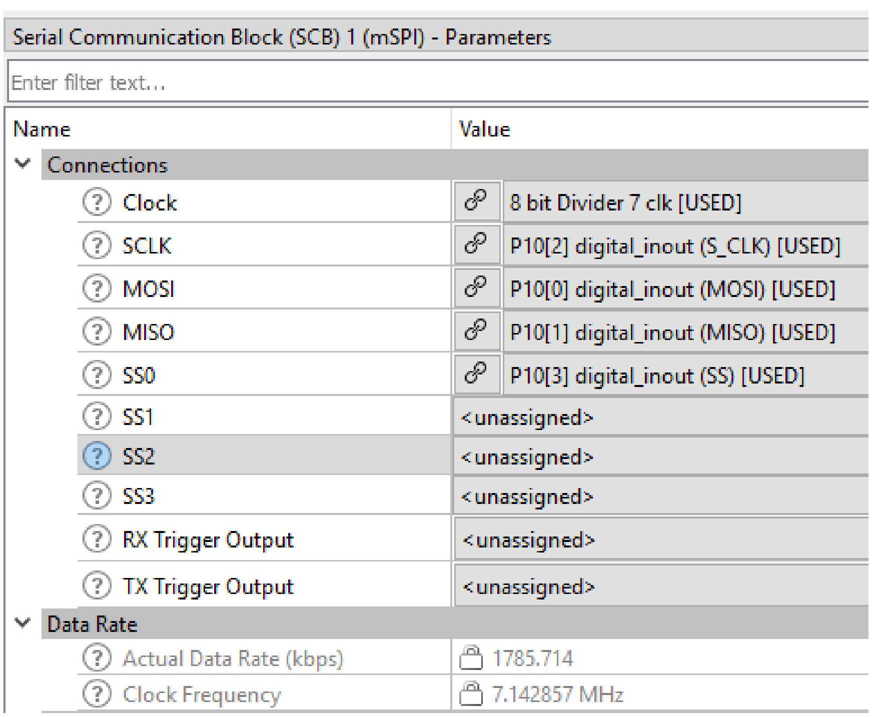


Fig. 37: Hardware configuration of SPI (2)

C. Image Processing

1) Resolution: The original idea was to use a HD image to process because it would be able to capture more information, and therefore more accurately detect objects in a picture. This was assumed without taking into account the capabilities of the microcontroller. As it has been stated, attempts to add more physical memory to the system proved to more difficult than expected, leaving the program to only operate with a limited amount of memory. Due to this limitation, as well as matching the resolution of the TFT screen, a resolution of 320x240 was chosen to implement the image processing aspects of the system. It should be stated that the program was structured in a way that allows for easy changing of resolutions. Another impact of memory limitation is that only 80 rows of the image can be evaluated at a time. The resolution parameters are set in ImProc.H which can be found in section XVI-K.

2) Output Format: There are a multitude of image formats than can be used and each one has its own specific way to read pixel information. JPEG was immediately decided against due to its lossy nature and high level of complexity to read pixel data. That left RGB, raw RGB, and YUV formats. Going a level down, each of the three formats have different arrangements and sizes in which they can be formed. For example, RGB can be either 888, 565, 555, 444, and even still further the ordering of the R,G, and B can vary.

Cypress provides an example project that interacts with the TFT display and uses an example image with a format of RGB565 and resolution of 320x240. Since the Arducam can, if configured correctly (a whole project in itself), output this format with that resolution, pre loaded images with this formatting were used to test and develop the image processing aspects of the system. Without being able to see the results on the TFT display of what was being applied to the pixel data, the image processing program would have never worked correctly.

For the Smart Eraser system, as mentioned above, a format of RGB565 was chosen. Not only would images captured be able to be displayed on the TFT display, but the way in which the image data would be received from SPI communication would be relatively straightforward. Pixel information is broken up between 2 consecutive bytes, a high and low byte. Knowing how RGB565 is arranged, every two bytes of information received over SPI would be used to constructed one pixel of information and due to the nature of the SPI hardware configuration of the PSoC 6, each SPI transfer is 2 bytes. Therefore each SPI transfer consists of one complete pixel.

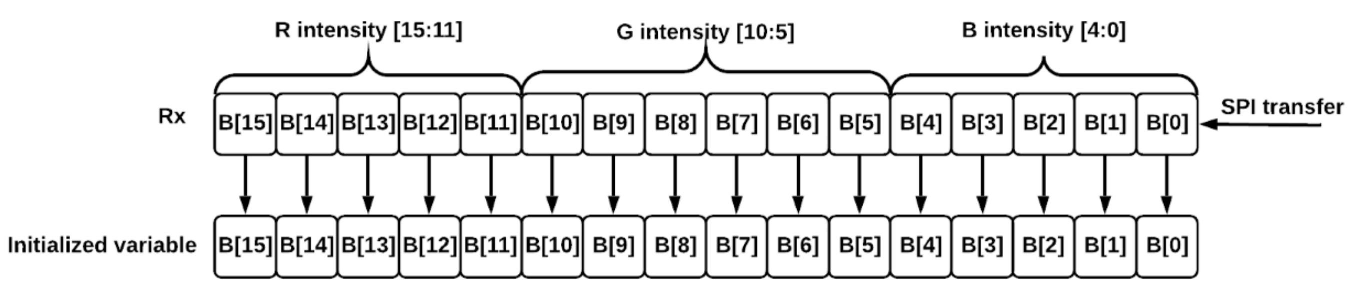


Fig. 38: Pixel Information being received through SPI from Arducam

3) Resolution: There are many different ways to do grayscale conversion from RGB. All are capable of achieving what was needed for Smart Eraser System, and the only difference between the methods are how true to gray the image looks and whether it has a darker or lighter tone to it. Because it is more accurate to the human eye, the Luminance method was chosen for the Smart Eraser system to convert to grayscale, mainly for the fact that the images would be viewed and not just processed behind the scenes.

Here is quick description of some different methods, including Luminance:

- Averaging Method

  - $Gray = (Red + Green + Blue)$ [26]
- Luminance Method
  - $Gray = (Red \times 0.2126 + Green \times 0.7152 + Blue \times 0.0722)$ [26]
- Desaturation
  - $Gray = (Max(Red, Green, Blue) + Min(Red, Green, Blue))/2$ [26]
- Decomposition (Max or Min)
  - $Gray = Max(Red, Green, Blue)$ [26]
  - $Gray = Max(Red, Green, Blue)$ [26]
- Single Color Channel
  - $Gray = Red$ [26]
  - $Gray = Green$ [26]
  - $Gray = Blue$ [26]

Once again, which method that is used is not important, it is separating the intensities from each other and then comparing them on an equivalent scale. An output format of RGB565 uses 16 bits to represent a single pixel, as shown in (figure with rgb565 output in previous section). In order to separate these intensities, the variable that contains the pixel information (type uint16_t) is masked accordingly to pull out the R,G, and B intensities, storing them into new variables (type uint8). Masking is the process of performing a logical AND operation on a variable to isolate bit values contained in that variable. As demonstrated in figure 39 if the pixel data variable is ANDed with the Red Mask variable, only bits[15:11] will be retained.

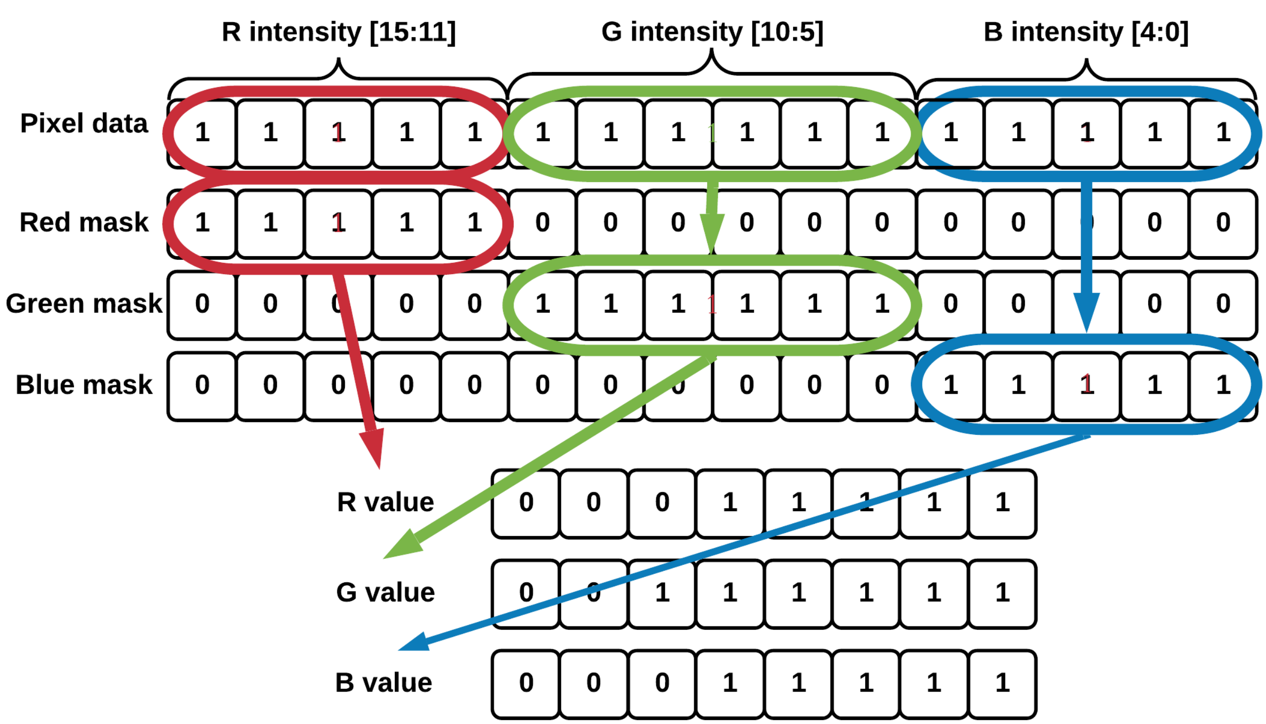


Fig. 39: How red, green, and blue intensities are seperated from one another

With the newly separated values as is, R and B represent shades of gray on a scale of 0 to 32 whereas G represents shades of gray on a scale of 0 to 64. If these values were to be tossed into one of the grayscale conversion algorithms, an image in shades of purple would be generated. This is known because it isnâĂŹt clearly stated anywhere that when using these conversion formulas, the intensities have to be represented by the same number of bits and therefore initial trials did not take this into account. Therefore some simple shifting is required before the formulas can be used. All intensities were upscaled to 8 bits so that values are compared on the complete range of grays (0 - 255) rather than descaling the green intensity down to 5 bits so that values are compared on a smaller, incomplete range of grays (0 - 32).

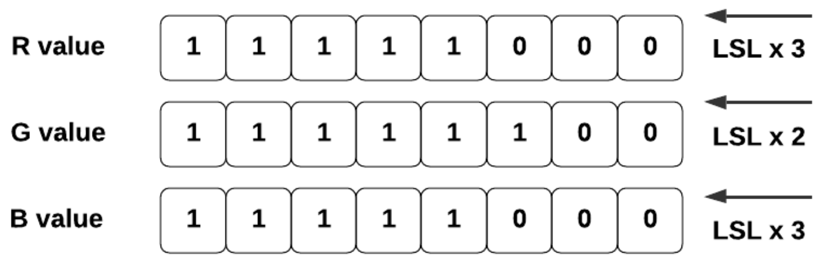


Fig. 40: Scaling seperated red green and blue intensities up to 8 bit numbers

Code Implementation (Separation and scaling):
R = ((pixel & 0b1111100000000000) » 11 « 3);
G = ((pixel & 0b0000011111100000) » 5 « 2);
B = (pixel & 0b0000000000011111) « 3;

Code is from conv2gray() in ImProc.C located in section XVI-J

At this point the RGB values are ready to be run through the conversion formula to generate a gray pixel value, in 8 bits. This gray value will then be used as the new R, G and B intensities of the original pixel, creating a shade of gray in the RGB565 format. To build this new RGB565 pixel, the process up until now is done in reverse. New variables are created to hold âĂIJprocessedâĂİ values of RGB that are of type uint16_t, to match the original pixel data length.

Gray = (Red * 0.2126 + Green * 0.7152 + Blue * 0.0722)

Fig. 41: Luminance method for converting RGB to grayscale

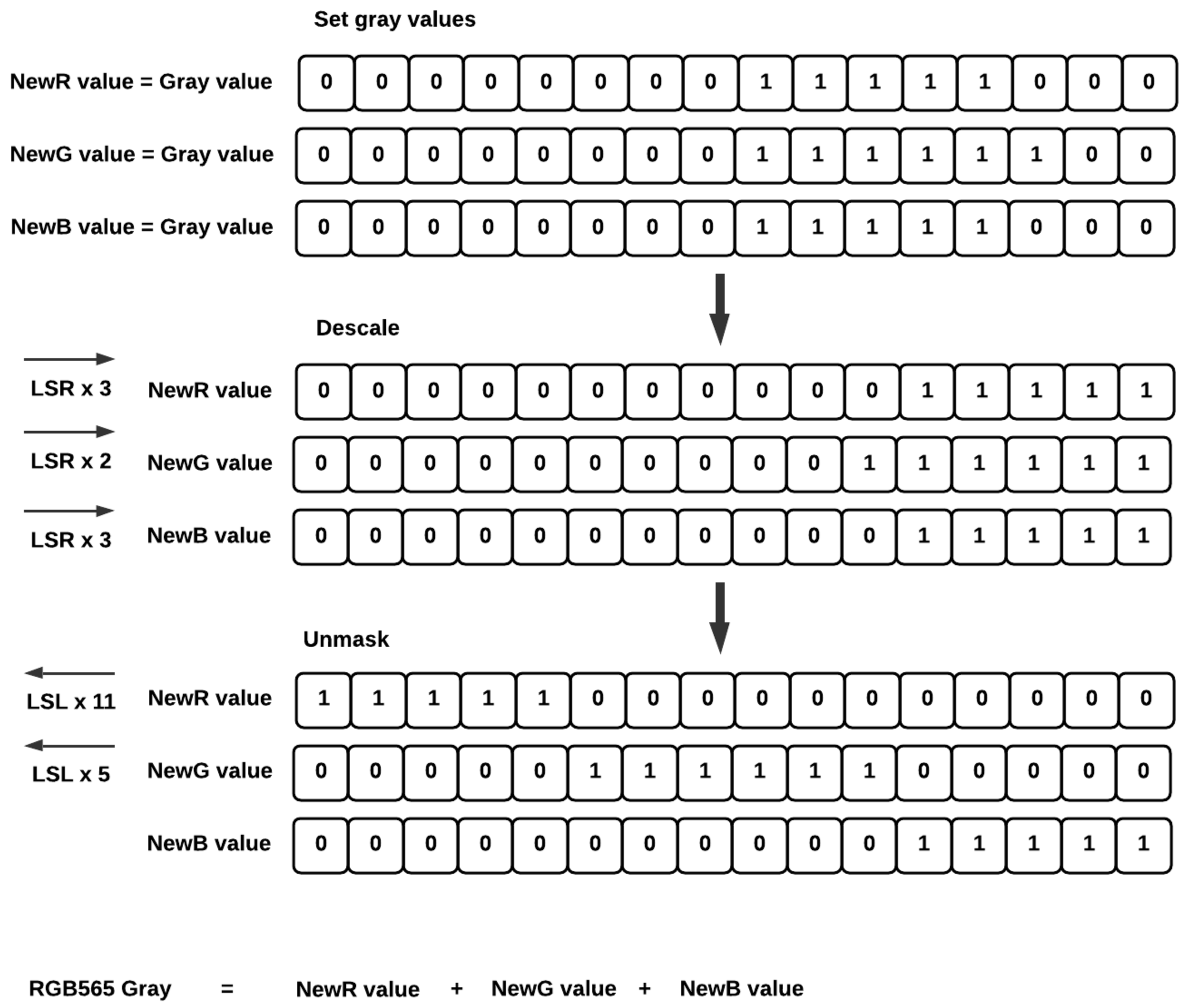


Fig. 42: Process in reverse: In order to create gray RGB pixel

Code Implementation (conversion, descaling, reassigning):
Gray_pixel = (R * 0.2126) + (G * 0.7152) + (B * 0.0722);
R_processed = ((uint16_t)Gray_pixel)» 3 « 11;
G_processed = ((uint16_t)Gray_pixel)» 2 « 5;
B_processed = (uint16_t)Gray_pixel » 3;
pixel = (uint16_t)(R_processed + G_processed + B_processed);

Code is from conv2gray() in ImProc.C located in section XVI-J

4) Grayscale Results:

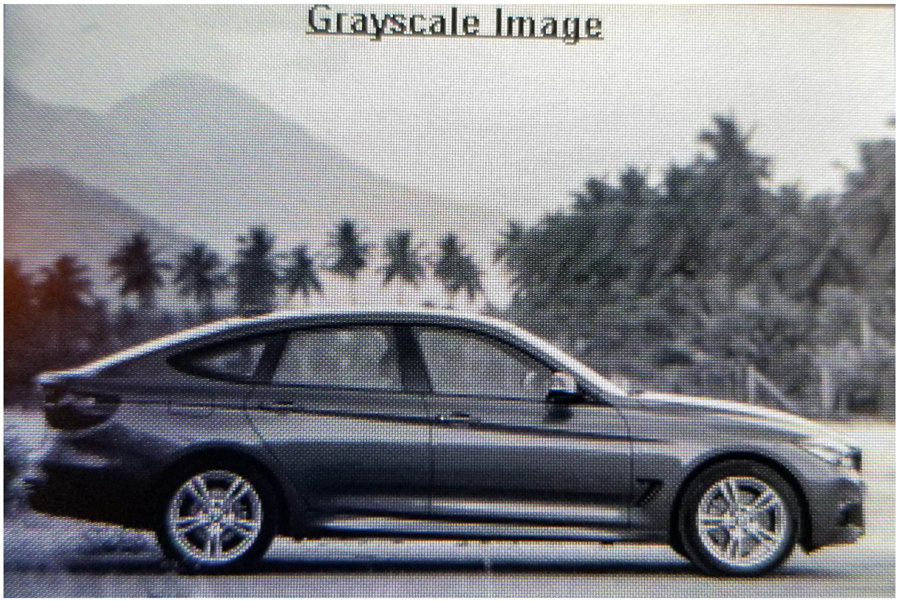


Fig. 43: Reuslts of grayscale algorithm from PSoC 6 (1)

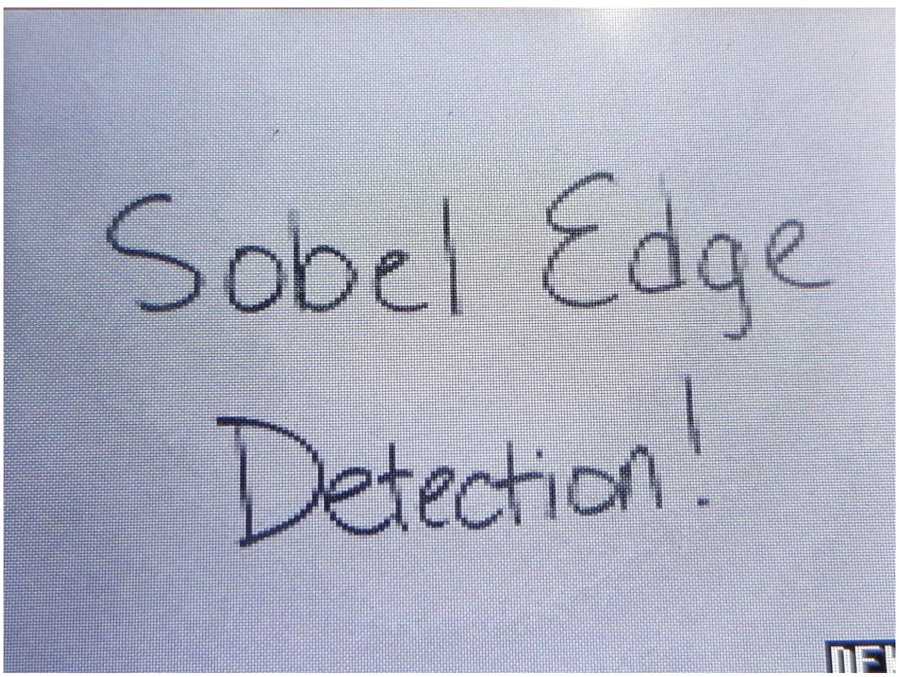


Fig. 44: Reuslts of grayscale algorithm from PSoC 6 (2)

5) Sobel Edge Detection: Sobel edge detection is an edge detection algorithm that uses a sobel kernel to detect differences in intensities at each pixel location. This is done by convolving a sobel kernel through each pixel in an image array. The pixel information must be grayscale in order for changes in intensity to be seen. A sobel kernel is a 3x3 array containing specific values to determine changes in values (gradients) from one edge to another. This has to be done for both vertical and horizontal edges, so there will be a kernel for each. The values inside the kernel are what make it a Sobel kernel. Using different values would classify it as a different type such as, SobelâĂŞFeldman or Scharr.

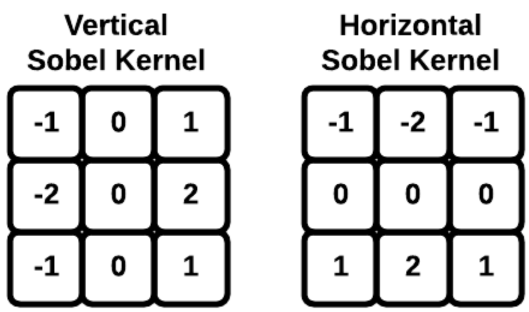


Fig. 45: Sobel Kernels

When moving through the image array, the center of the 3x3 kernel must never reach a border pixel otherwise a segmentation fault will prompt. This is because the pixel being evaluated (at the center of the kernel) relies on information from the surrounding pixels. If you were to attempt to pass the kernel through any part of the border, there will be some portion of the kernel that is out of bounds of the defined memory region. This is shown in figure 46.

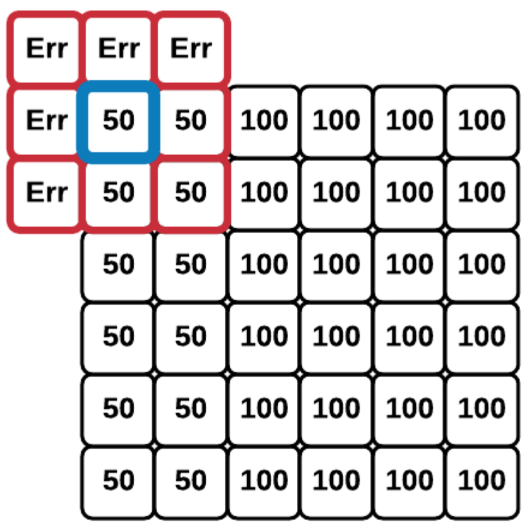


Fig. 46: Visual of what happens if the the sobel kernel is convolved along a boundary

Code Implementation (boundary detection):

```
for (k = Length + 1; k < bound - Length; k++){
   if ((k + 1) % Length == 0) {
ptr_GD = ptr_GD + 2;
ptr_PD = ptr_PD + 2;
k = k + 2;
}
```

Code is from sobel() in ImProc.C located in section XVI-J

Once boundaries are accounted for in the program, the kernel can be convolved through the image array. At each pixel location (inside of boundary pixels) the operation shown in figure (right below) is carried out, both for the X and Y gradient.

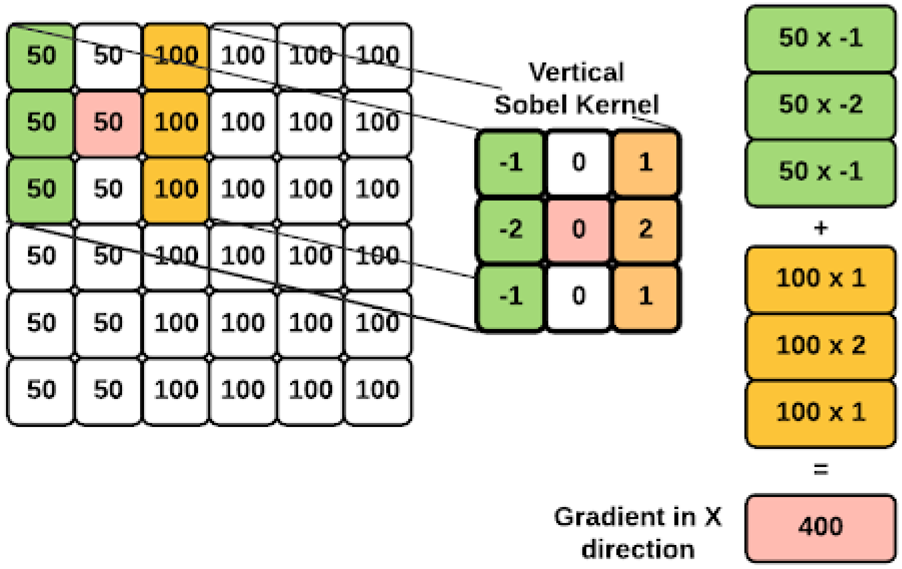


Fig. 47: Operation of the vertical kernel

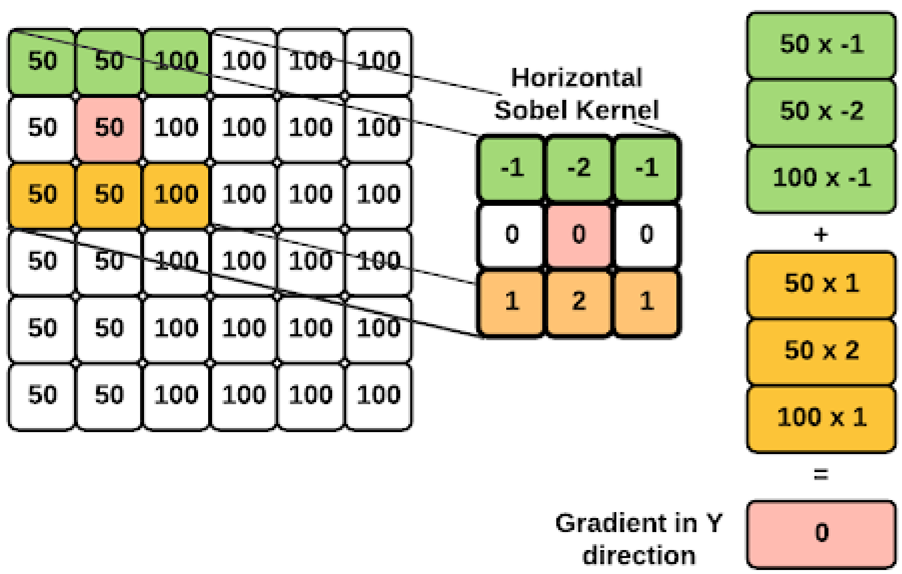


Fig. 48: Operation of the Horizontal kernel

Finding the gradient in both the X and Y direction at any given pixel location (aside from boundary pixels) allows the total magnitude to be found at that location. This is done by taking the Gx and Gy values and using the Pythagorean Theorem.

$$\boldsymbol{G} = \sqrt{(\boldsymbol{Gx}^2 + \boldsymbol{Gy}^2)}$$

Fig. 49: Using Pythagorean theorem to determine the magnitude of the gradient

Code Implementation (kernel convolution):

```
Gx =(sobel_kernel[0])*(*(ptr_GD - Length + 1)) + (sobel_kernel[1])*(*(ptr_GD + 1)) +
(sobel_kernel[2])*(*(ptr_GD + Length + 1)) - (sobel_kernel[0])*(*(ptr_GD - Length - 1)) -
(sobel_kernel[1])*(*(ptr_GD - 1)) - (sobel_kernel[2])*(*(ptr_GD + Length - 1));

Gy =(sobel_kernel[0])*(*(ptr_GD - Length + 1)) + (sobel_kernel[1])*(*(ptr_GD - Length)) +
(sobel_kernel[2])*(*(ptr_GD - Length - 1)) - (sobel_kernel[0])*(*(ptr_GD + Length + 1)) -
(sobel_kernel[1])*(*(ptr_GD + Length)) - (sobel_kernel[2])*(*(ptr_GD + Length - 1));

G = (sqrt(pow(Gx,2) + pow(Gy,2)));
```

Code is from sobel() in ImProc.C located in section XVI-J

The total magnitude G is now compared against a threshold value, which can be manipulated to adjust the strength of edge detection. If only sharp changes in intensity are desired, to be recognized as an edge then you would set the threshold value very high and if any change of intensity is desired then the threshold will be set very low. It is a trial and error to see if the image turns out correctly or not and is easily adjustable. Conditional statements are used to then compare against this threshold to set a pixel value to either white (detected edge, G is above threshold) or black (nothing detected, G is below threshold).

6) Results of Sobel Edge Detection:

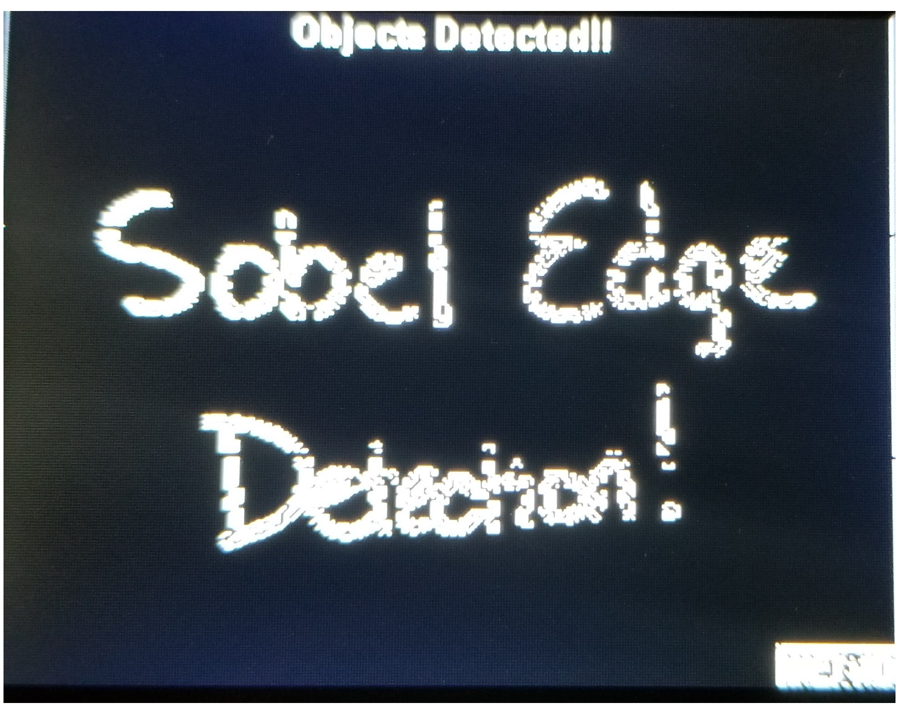


Fig. 50: Results of Sobel edge detection

7) Square_Detection.C Algorithm:

8) UART Communication - PSoC 6 to HUZZAH32 (server): Universal Asynchronous Receiver/Transmitter (UART) was chosen as the communication protocol between the main microcontroller (PSoC 6) and server microcontroller (HUZZAH32). This is the simplest of the three communication protocols used in the Smart Eraser. As the name suggests, communication is asynchronous, meaning there is no clock regulating the transfer of data between devices. Instead a baud rate is used, which is defined as the rate information can be transferred along a communication channel. Unlike SPI and I^2C, there is no master/slave relationship and thus communication relies on a few, simple parameters. Each device must be set to the same parameters in order to properly communicate. These parameters include; baud rate, parity, stop bit(s). The Smart Eraser system operates with a 115200 baud rate, parity enabled and set to even, with 1 stop bit and aheres to the RS232 standard of serial communication [25].

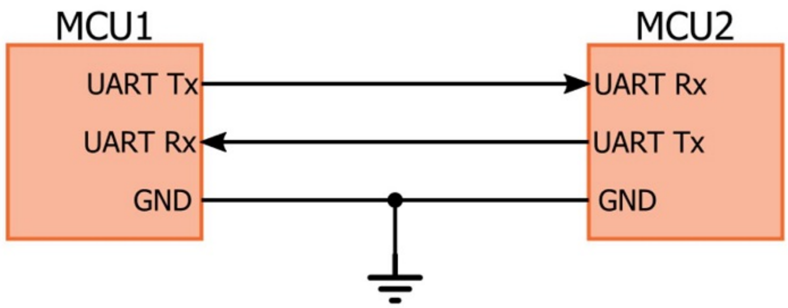


Fig. 51: UART Communication [16]

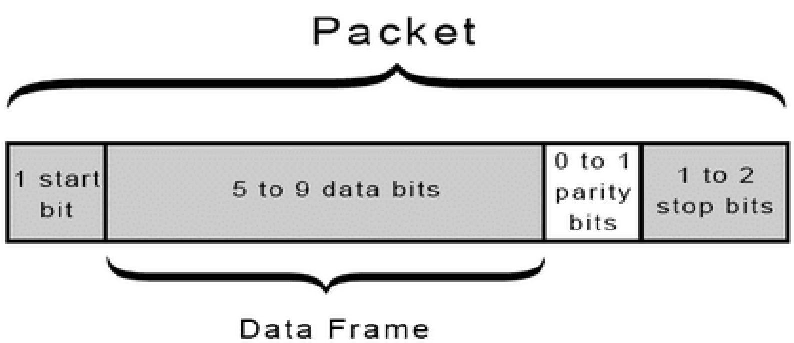


Fig. 52: UART packet contents [17]

The original plan was to communicate with I^2C because the PSoC 6 already had a configured SCB for master I^2C transferring capabilities. However, due to limitations with the HUZZAH32 (currently only configurable as master (both SPI and I^2C)) UART was chosen instead. While researching UART it was found that this is often the chosen method of communication between MCUs so it worked out being the best option anyways. The code implementation of sending packets to the server (written in C) and how the server receives the packets (written in Python) is shown below:

Code Implementation (UART TX):

```
Cy_SCB_UART_PutArray(KIT_UART_HW, ptr_PD, 25);
Cy_SysLib_Delay(1);
Cy_SCB_UART_ClearTxFifoStatus(KIT_UART_HW, CY_SCB_UART_TX_DONE);
```

Code is from Process_Image() in main.C located in section XVI-C

Code Implementation (UART RX):

```
def UARTread():
```

```
while(True):
while (uart.read() == None):
pass
buff = uart.read()
return buff
```

Code is from UARTread() in UART.py located in section adddddddd pytttttttttthhhhhhhhhoooooooooonnnnnnnnnn fillllllllllllleeeeeeeeeeeessssssss XVI-C

9) The Original Plan for Wireless Communication: When this project was originally started, the idea was to use wires to connect all of the components. This meant that a long wire would need to be used to connect the components on the Y-axis of the whiteboard in order to allow it to move. This idea was scrapped when everyone involved realized that not only would wireless communication allow for a more âĂIJpleasing to the eyeâĂİ design, but would also add a layer of complexity to the projectâĂŹs difficulty.

The original plan for the wireless communication was to use Raspberry Pi boards that would connect to the DE1_SoC wirelessly via a wireless dongle that would plug into the DE1_SoC. However, the transceivers that were found that would allow the Raspberry Pi boards to be wireless were only compatible with Arduino boards, so the Raspberry Pi was changed to Arduino. Soon after, problems arose from using the DE1_SoC, so the microcontroller changed to the PSoC 6, and the new plan became to hardwire the transmitter Arduino to the PSoC 6.

10) Arduino Wireless Communication: Arduino microcontrollers are notoriously straightforward to use because of their open-sourced nature, meaning that there are already many libraries that are made for the board for various other external devices they can be used with, as well as applications that they can be used for. Therefore, the wireless communication between the transmitter and receiver Arduinos was straightforward, as well as the connections between the board and the transceiver.

The transmitter Arduino that would send instructions to the stepper motors was the model MKR1000, and the transceiver that could process the wireless communication and send the information needed was a Kuman nRF24L01 with an attachable antenna to extend the range that the wireless radio signal could reach. The pin layouts for the MKR1000 and the transceiver are shown in the following figures.

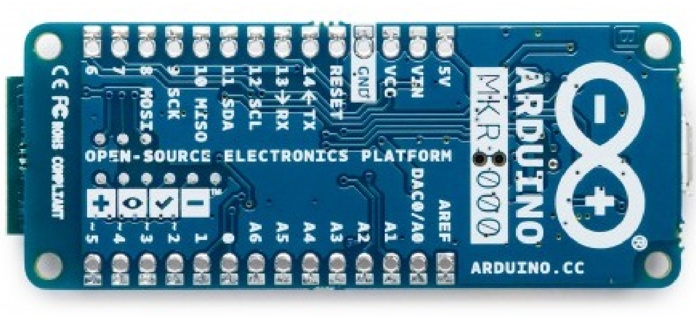


Fig. 53: Image and pin layout of the MKR1000 WiFi module by Arduino [18]

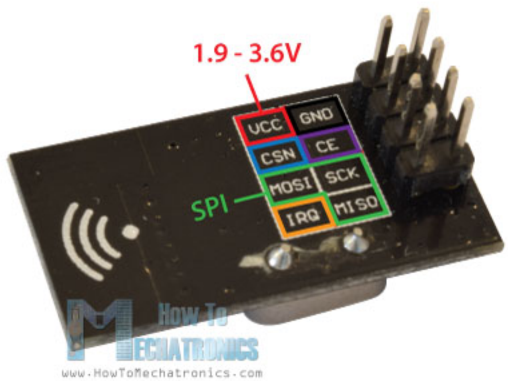


Fig. 54: Image and pin layout of the Kuman nRF24L01 transceivers[19]

A diagram showing the connections between these two chips is in the following figure.

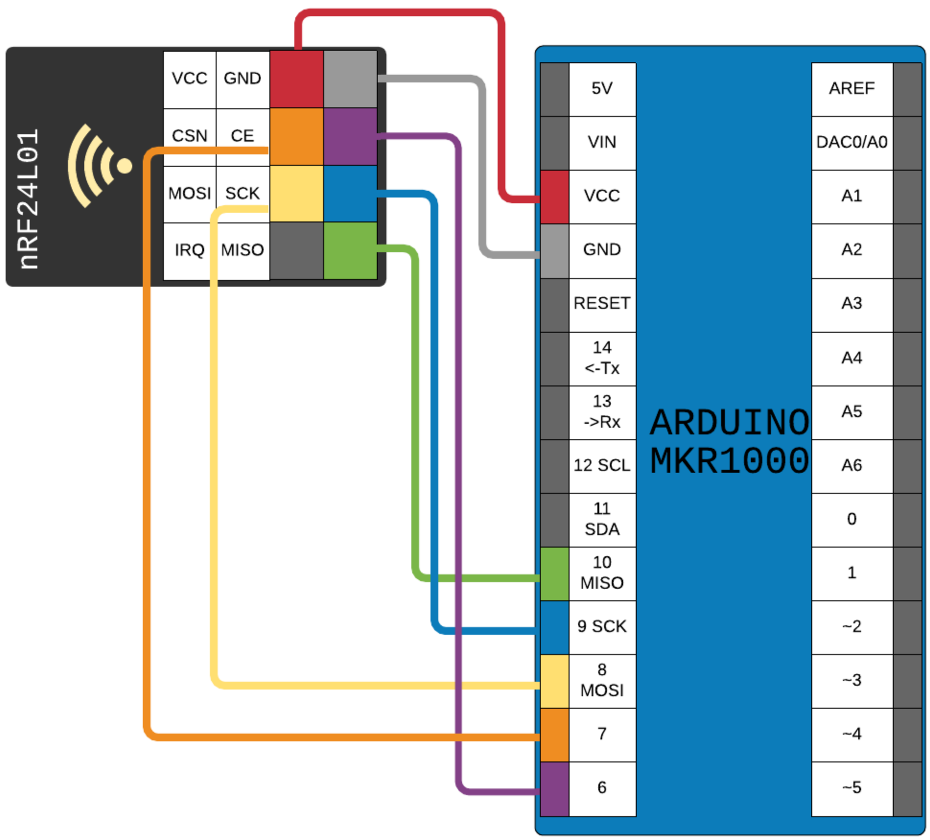


Fig. 55: Pin connections between the MKR1000 microcontroller and the wireless transceiver to create the transmitter module

For the code of the transmitter, the template code was taken from the example shown by Dejan Nedelkovski, and is shown in Appendix 1 of this report. In order to use the wireless transceiver, the specific library made for it needed to be included. The RF24 radio was then configured to use pins 6 and 7 on the Arduino as the CE and CSN pins shown in 54. In the setup of the program, the radio connection is initialized, the PA level is set to the minimum it can be, and the radio is set to be a transmitter with the stopListening() function. The addresses the transmitter will be sending data to are also outlined. The PA is the Power Amplifier factor, and the higher it is, the more unstable it can be, which would then require a bypass capacitor to be connected between the voltage power source and ground connected to the chip. Therefore, it was kept to a minimum, especially because the modules were so close together, so it was unnecessary to amplify it more.

Next, in the main loop of the program, in order to send a signal to a specific address, a writing pipe needs to be opened to that address. Only one writing pipe can be open at a time, so separate writing pipes to each receiver were opened and continuously alternated between in the loop. The number for the stepper motors to rotate a certain degree were then sent using the radio.write() function, whos parameters include a pointer to the name of the variable to send to the receiver, and the length of the data.

The receivers that were attached to each stepper motor were the Arduino UNO R3, and the transceivers were the same model used for the transmitter. The Arduino UNO R3 used is shown in the following figure.

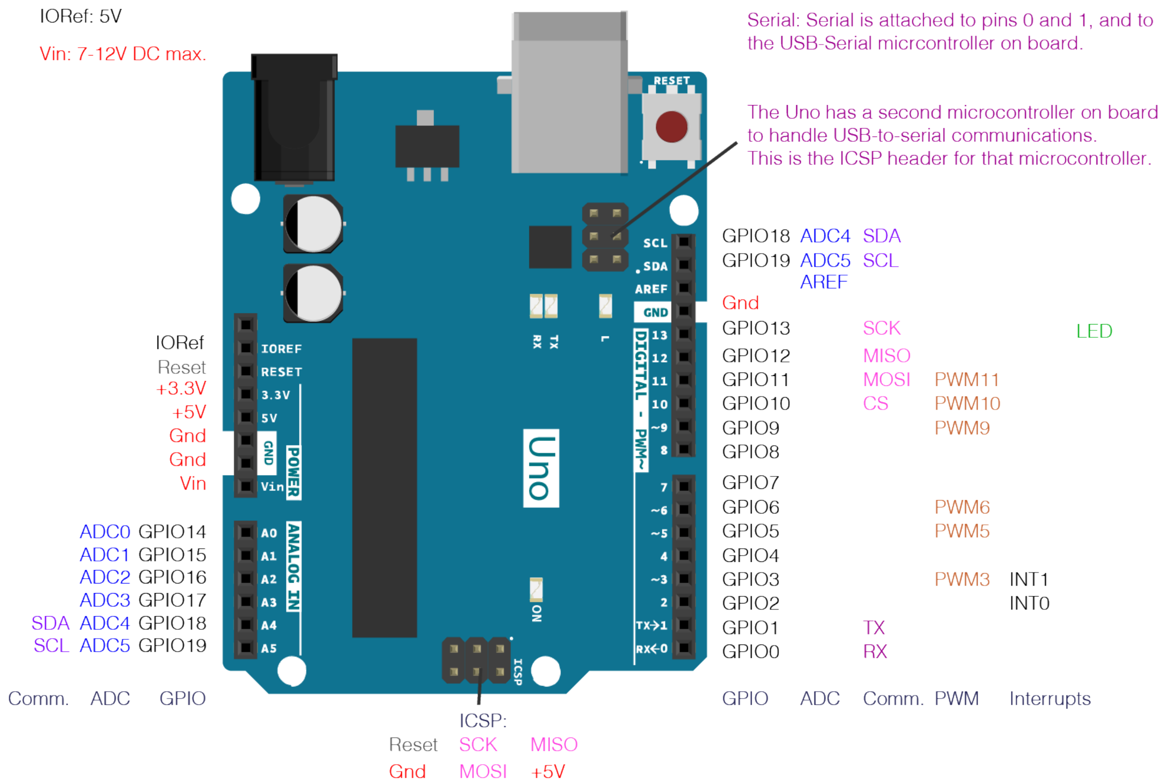


Fig. 56: Pin layout on the Arduino UNO R3 [20]

The pin connections between the UNO R3 and the wireless transceiver are shown in the following figure.

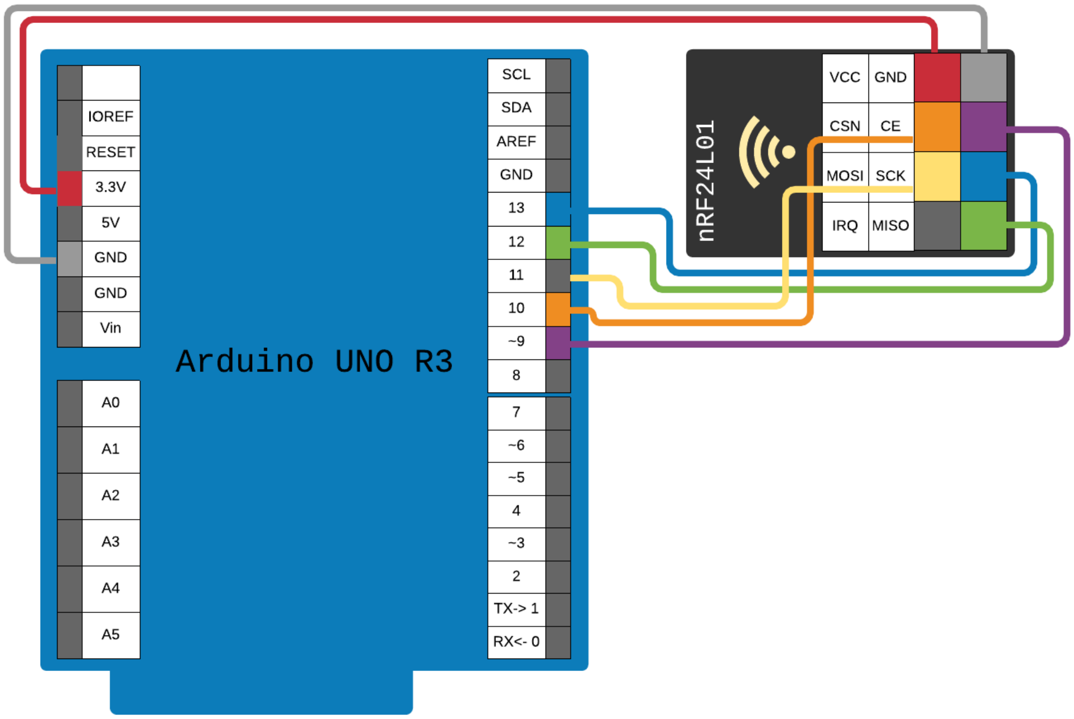


Fig. 57: Pin connections between the UNO R3 microcontroller and the wireless transceiver to create the receiver modules

For the code of the receiver, the template code was taken from the example shown by Dejan Nedelkovski, and is shown in Appendix 2 of this report. In order to use the wireless transceiver, the specific library made for it needed to be included. The RF24 radio was then configured to use pins 9 and 10 on the Arduino as the CE and CSN pins. In the setup of the program, the radio connection is initialized, the PA level is set to the minimum it can be, and the radio is set to be a receiver with the startListening() function. The address of the receiver is set, and the baud rate is also set for the Serial Monitor within the Arduino IDE so it will read in data at the specified rate. Next, in the main loop of the program, a loop is created to check if there is data available to receive. If there is, the data is read into a variable to be used for the stepper motor movement.

When both of these programs ran, they each received the instructions wirelessly and were able to move the stepper motors the right degree of rotation. However, unfortunately, due to an unknown reason the wireless communication stopped working with about 30 days left to complete the project. The most likely culprit for the malfunction may have been due to faulty transceivers; upon further inspection of the reviews left by those who have used these devices in the past, many did not work correctly upon arrival. Therefore, this method of wireless communication had to be replaced with three HUZZAH32 Feather boards by Adafruit, with one being used as a host server (a.k.a. the transmitter) and two being used as clients to the server (a.k.a. the receivers). This leads to the current wireless communication system, which is a connection between a ‘host’ or ‘server’ (which acts as the transmitter) and the ‘client’ (which acts as the receievr).

11) The Current Wireless Communication System (Server to Clients): The decided upon wireless communication system was to use a client/server implemented on HUZZAH32s. One would act as the server, directly interacting with the PSoC 6 and the other two would be clients, attached to the stepper motor systems. Using micropython, the server uses libraries and socket programming in order to achieve this. The server is set up as an access point in order for the clients to connect to its network and then socket programming is used for the transfer of data. There are few things that are necessary in order for this to work. There needs to be two separate IP addresses for the network and server. Most implementations of client/sever is done on an existing network, but since we are using the HUZZAH32 itself as the network, common methods of creating socket connections were insufficient. We believe this was due to the network and server being assigned the same IP address. Once it was made sure that these two addresses were separate, successful binding of sockets occurred. The other thing that is needed is a defined port on the server side so that the client knows what to look for. Lastly, as this is a TCP connection there must be acknowledgment for every packet sent.

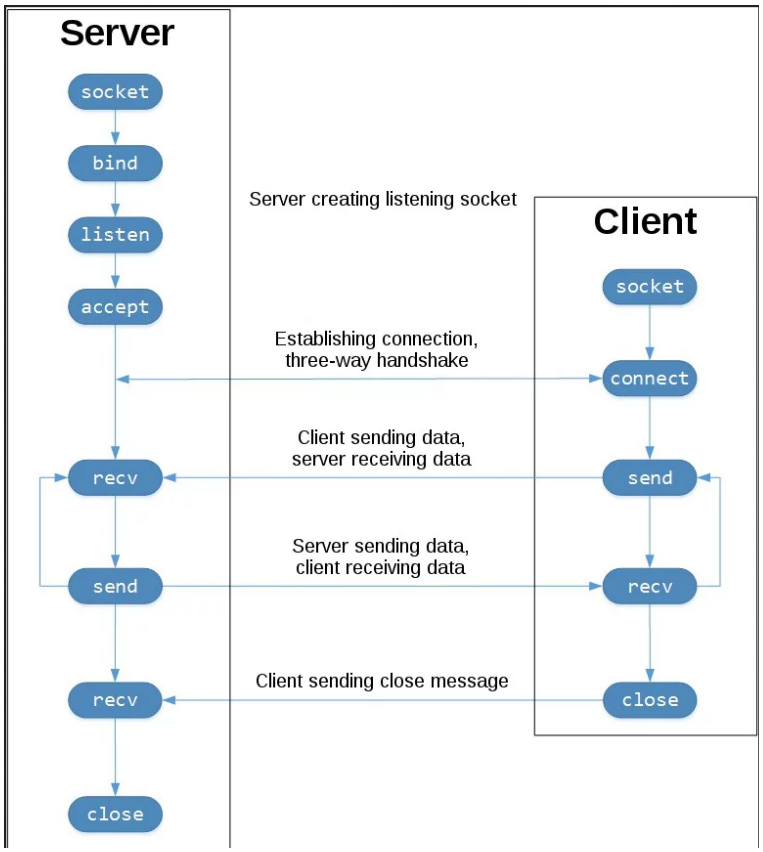


Fig. 58: Overview of client/server interaction [4]

12) *Serpentine.C Algorithm:*
13) *Stepper Motor Movement:*

D. Hardware

1) *Using the Logic Analyzer:*
2) *Testing with the Arduino and Arducam:*
3) *Stepper Motor and Driver Connections:*
4) *Power Parameters:*

E. Project Prototype - Planning

1) *Initial Prototype Design:*
2) *Final Prototype Design:*
3) *Dimensions:*

F. Project Prototype - Building

1) *Component Mounting Order:*

2) Final Product:

XIII. Stakeholder List

1) Aaron Stillmaker, Ph.D.: As the professor of the Senior Design course, wants to guide the students working on the Smart Eraser toward success through the completion of this project. Is also the manager of the project deliverable due dates.

2) Hovannes Kulhandjian, Ph.D.: As a technical advisor of this project, he wants to see the success of this idea. He also held the original concept of this project, therefore he wants to help give technical advice to the students working on the Smart Eraser, as well as see the final product of their efforts.

3) Roger Moore: As a technical advisor of this project, he wants to see the success of this idea. He has contributed his opinions on some of the technology and methods used for the image processing program, and wishes to see this project come to fruition. Although he has offered advice and his expertise, he does not wish to be apart of the Approvals signature section of this report. Therefore, in that section, his name is omitted.

4) Richard Martinez, Research and Development - DPS Telecom: Sponsor that will be contributing funds to this project. He will also be providing the expertise of those who work at his company, as well as any additional resources that may be needed for testing the product and creating any deliverables. Although his company has contributed these funds to this project, he does not wish to be a part of the Approvals signature section of this report, as he considers the fund a âĂIJdonationâĂİ rather than a âĂIJcontributionâĂİ. Therefore, in the Approvals section, his name is omitted.

5) Heather Libecki: Project Manager, Team Member

6) Chris Quesada: Team Member

7) Juan Colin: Team Member

XIV. Project Approval Requirements

The following list details the requirements that must be met in order for the Smart Eraser to be approved by the various stakeholders in the project.

- All requirements listed in the High-level requirements section of this project charter, which state that the project must:
  - Be completed within the outlined budget.
  - Be implementable within 2 semesters.
  - Be complex enough to warrant the title “senior design project”.
  - Produce a complete Project Charter outlining the various project information, figures, and tables of the Smart Eraser.
  - Have significant, roughly equal portions of the project be completed by each team member.
  - Utilize material learned in core and technical elective classes throughout college careers.
  - Produce a deliverable that can be presented in the Senior Project Presentation Day event held in the Satellite Student Union building.
- Additional requirements requested due to the acceptance of the sponsership from DPS Telecom:
  - Deliver a project report.
  - Create and manage a project archival on GitHub (or similar).
  - Implement a working prototype.
  - Create a 3-4 minute video, including a business plan with some marketing information.

XV. Approvals

Signature: ____________________________________
Heather Libecki - Project Manager

Signature: ____________________________________
Chris Quesada - Team Member

Signature: ____________________________________
Juan Colin - Team Member

Signature: ____________________________________
Hovannes Kulhandjian, Ph.D. - Technical Advisor

Signature: ____________________________________
Aaron Stillmaker, Ph.D. - Course Instructor

References

[1] ArduCAM.com, "ArduCAM-Mini-5MP-Plus OV5642 Camera Module," http://www.arducam.com/downloads/shields/ArduCAM_Mini_5MP_Plus_OV5642_Camera_Module_Hardware_Application_Note.pdf.

[2] gnovice, "How can I convert between RGB565 and RGB24 image formats in MATLAB?" https://stackoverflow.com/questions/3578265/how-can-i-convert-between-rgb565-and-rgb24-image-formats-in-matlab.

[3] S. Sodha, "An Implementation of Sobel Edge Detection," https://www.projectrhea.org/rhea/index.php/An_Implementation_of_Sobel_Edge_Detection.

[4] A. Anand, "Client and Server Chatting Application in Python," https://www.kodefork.com/articles/client-and-server-chatting-application-in-python/.

[5] STEPPERONLINE, "Stepper Motor," https://www.omc-stepperonline.com/download/23HS22-2804S.pdf.

[6] ——, "Nema 23 CNC Stepper Motor 2.8A 178.5oz.in/1.26Nm CNC Stepping Motor DIY CNC Mill," https://www.amazon.com/dp/B00PNEPF5I/?coliid=I3APKEM9Y17HEP&colid=22YAOKWOJAKA9&psc=0&ref_=lv_ov_lig_dp_it.

[7] ——, "User's Manual for DM542T," https://www.omc-stepperonline.com/download/DM542T.pdf.

[8] Adafruit.com, "Assembled Adafruit HUZZAH32 âĂŞ ESP32 Feather Board - with Stacking Headers," https://www.adafruit.com/product/3619.

[9] EBL, "EBL Li-ion 9V Battery Charger 5 Bay 9V 6F22 Lithium-ion Rechargeable Batteries," https://www.amazon.com/dp/B07GB3TSKH/?coliid=I3FJDEJ8MXP6RY&colid=22YAOKWOJAKA9&ref_=lv_ov_lig_dp_it&th=1.

[10] Accuride, "Instructions Manual," https://www.accuride.com/amfile/file/download/file_id/155/product_id/2909/.

[11] Amazon.com, "4pcs Nema 23 Stepper Motor Steel Mounting Bracket with Mounting Screws," https://www.amazon.com/dp/B073V77VLD/?coliid=I3S0XHJ6LUPARP&colid=22YAOKWOJAKA9&psc=0&ref_=lv_ov_lig_dp_it.

[12] Uxcell, "uxcell Aluminum GT2 40 Teeth 6.35mm Bore Timing Belt Pulley Flange Synchronous Wheel for 3D Printer," https://www.amazon.com/dp/B0728PDWY5/?coliid=I3K7SWFRMQ3A2F&colid=22YAOKWOJAKA9&psc=0&ref_=lv_ov_lig_dp_it.

[13] Mercury, "Mercurry 5 Meters GT2 timing belt width 6mm Fit for RepRap Mendel Rostock Prusa GT2-6mm Belt ," https://www.amazon.com/dp/B071K8HYB4/?coliid=I3KCLVNOTYFU5Y&colid=22YAOKWOJAKA9&psc=0&ref_=lv_ov_lig_dp_it.

[14] C. Basics, "Basics of the I2C Communication Protocol," http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/.

[15] S. VHDL, "How to Design SPI Controller in VHDL)," https://surf-vhdl.com/how-to-design-spi-controller-in-vhdl/.

[16] R. Keim, "Back to Basics: The Universal Asynchronous Receiver/Transmitter (UART)," https://www.allaboutcircuits.com/technical-articles/back-to-basics-the-universal-asynchronous-receiver-transmitter-uart/.

[17] C. Basics, "Basics of UART Communication," http://www.circuitbasics.com/basics-uart-communication/.

[18] Arduino, "ARDUINO MKR1000 WIFI," https://store.arduino.cc/usa/arduino-mkr1000.

[19] H. to Mechatronics, "Arduino Wireless Communication âĂŞ NRF24L01 Tutorial," https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/.

[20] Bob5421, "which pins should i take for i2c on arduino uno," https://stackoverflow.com/questions/42022000/which-pins-should-i-take-for-i2c-on-arduino-uno/42022566.

[21] I. S. Association, "IEEE 802.15.3e-2017 - IEEE Standard for High Data Rate Wireless Multi-Media Networks–Amendment 1: High-Rate Close Proximity Point-to-Point Communications," https://standards.ieee.org/standard/802_15_3e-2017.html.

[22] IEEE, "IEEE Editorial Style Manual ," https://www.ieee.org/content/dam/ieee-org/ieee/web/org/conferences/style_references_manual.pdf.

[23] N. E. M. Association, "NEMA Standards Publication ICS 16 Motion/Position Control Motors, Controls, and Feedback Devices," https://www.nema.org/Standards/SecureDocuments/ICS16.pdf.

[24] W. contributors, "Rec. 709," https://en.wikipedia.org/wiki/Rec._709.

[25] electronicsnotes, "Serial data transmission standards," https://www.electronics-notes.com/articles/connectivity/serial-data-communications/transmission-standards.php.

[26] T. Helland, "Seven grayscale conversion algorithms (with pseudocode and VB6 source code)," http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/.

XVI. Appendices

A. Appendix 1 - transmitter_xy.ino

```
/*
* SMART ERASER SENIOR DESIGN PROJECT
* Stepper Motor Transmitter
* by Heather Libecki
*
* with help from Dejan Nedelkovski
*/

/*transmitter should be MRK1000*/
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Stepper.h>

RF24 radio(6, 7); // CE, CSN
const byte address[][6] = {"00001", "00002"};
void setup() {
  radio.begin();
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}
void loop() {
  int steps1 = 3200; //revolutions specific to stepper motor
  radio.openWritingPipe(address[0]);
  radio.write(&steps1, sizeof(steps1));
  Serial.println(steps1);
  delay(500);
  int steps2 = 3600;
  radio.openWritingPipe(address[1]);
  radio.write(&steps2, sizeof(steps2));
  Serial.println(steps2);
  delay(500);
}
```

B. Appendix 2 - receiver_x.ino

```
/*
* SMART ERASER SENIOR DESIGN PROJECT
* Stepper Motor Receiver for X-axis
* by Heather Libecki
*
* with help from Dejan Nedelkovski
*/

/*receiver should be UNO*/
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Stepper.h>

// initialize the stepper library on pins 7 through 4
int stepsPerRev = 1600;
Stepper myStepper(stepsPerRev, 7, 6, 5, 4);

RF24 radio(9, 10); // CE, CSN
const byte address[6] = "00001";
void setup() {
  Serial.begin(9600);
  radio.begin();
  radio.openReadingPipe(1, address);
  radio.setPALevel(RF24_PA_MIN);
  myStepper.setSpeed(120);
  radio.startListening();
}
void loop() {
  if (radio.available()) {
    int steps1;
    radio.read(&steps1, sizeof(steps1));
    myStepper.step(steps1);
//   Serial.println("clockwise");
    Serial.println(steps1);
    delay(500);
    myStepper.step(-steps1);
//   Serial.println("counterclockwise");
    Serial.println(steps1);
    delay(500);
  }
}
```

C. Appendix 3 - main.c

```
/****************************************************************************
* File Name:  main.c
*
* Version:   1.0
*
* Description: Uses SPI and I2C communication to receive image form Arducam,
*        process the image to find edges, translates detected edges to
*        coordinates for stepper motors, and sends coordinate information
*        to HUZZAH32 server via UART communication.
*
*        This code was modified from a template provided by Cypress
*        Semiconductor Corporation -> TFTemWin example
*
* Author:  Chris Quesada
*****************************************************************************/
#include "cy_pdl.h"
#include "cycfg.h"
#include "GUI.h"
#include "cycfg_qspi_memslot.h"
#include "cy_smif.h"
#include <Interface.h>
#include <SPIMaster.h>
#include <I2CMaster.h>
#include <stdbool.h>
#include <time.h>
#include <ImProc.h>
#include <math.h>
#include <Arducam.h>

/****************************************************************************
* Global constants
****************************************************************************/

#define NUMBER_OF_PAGES    (1u)
#define BTN_PRESSED       (0u)
#define BTN_RELEASED      (1u)

/************************ Function Prototypes ********************************/
int Trigger_Capture(void);
void Process_Image(void);
void (*PageArray[NUMBER_OF_PAGES])(void) = {Process_Image, /*func1, func2,
   func3.....*/};

/*****************************************************************************/

/* Reference to bitmap test image */
extern GUI_CONST_STORAGE GUI_BITMAP bmExampleImage;


/*****************************************************************************
* Function Name: void ShowStartupScreen(void)
*****************************************************************************
*
* Summary:   This function displays the startup screen. This was provided by
*           Cypress.
```

```
*
* Parameters: None
*
* Return:   None
*
********************************************************************************/
void ShowStartupScreen(void)
{
   /* Set font size, foreground and background colors */
   GUI_SetFont(GUI_FONT_24B_1);
   GUI_SetColor(GUI_WHITE);
   GUI_SetBkColor(GUI_BLACK);

   /* Clear the screen */
   GUI_Clear();

   /* Print the text CYPRESS EMWIN GRAPHICS DEMO TFT DISPLAY */
   GUI_SetTextAlign(GUI_TA_HCENTER);
   GUI_DispStringAt("Smart Eraser", 160, 80);
   GUI_SetTextAlign(GUI_TA_HCENTER);
   GUI_DispStringAt("Version 1.0", 160, 100);
   GUI_SetTextAlign(GUI_TA_HCENTER);
   GUI_DispStringAt("Initializing....", 160, 120);
}

/*******************************************************************************
* Function Name: void ShowInstructionsScreen(void)
********************************************************************************
*
* Summary:   This function shows screen with instructions to press SW2 to
*           scroll through various display pages. This was provided by
*           Cypress.
*
* Parameters: None
*
* Return:   None
*
********************************************************************************/
void ShowInstructionsScreen(void)
{
   /* Set font size, background color and text mode */
   GUI_SetFont(GUI_FONT_24_1);
   GUI_SetBkColor(GUI_BLACK);
   GUI_SetColor(GUI_WHITE);
   GUI_SetTextMode(GUI_TM_NORMAL);

   /* Clear the display */
   GUI_Clear();

   /* Display instructions text */
   GUI_SetTextAlign(GUI_TA_HCENTER);
   GUI_DispStringAt("PRESS SW2 ON THE KIT", 160, 90);
   GUI_SetTextAlign(GUI_TA_HCENTER);
   GUI_DispStringAt("TO CAPTURE IMAGE", 160, 110);
}

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

```
* Function Name: Trigger_Capture(void)
******************************************************************************
*
* Summary:   This functions sends a trigger command to the Arducam module
*        via SPI
*
* Parameters: None
*
* Return:  int status
*
*****************************************************************************/
int Trigger_Capture(void)
{
  /* Buffer to hold command packet to be sent to the slave by the master */
  uint16_t txBuffer  = {0};

  /* Buffer to save the received data by the slave */
  uint16_t rxBuffer  = {0};

  MyCam_Test(rxBuffer, txBuffer);
  MyCam_Trigger(rxBuffer, txBuffer);

  return 0;
}

/*****************************************************************************
* Function Name: Process_Image(void)
******************************************************************************
*
* Summary:   This function takes an image, converts it to grayscale, detects
*        edges, implements square detection on detected edges, sends
*        results to HUZZAH32 via UART
*
* Parameters: None
*
* Return:  None
*
*****************************************************************************/
void Process_Image(void)
{
  int        flag              = 0;
  int       k                 = 0;
  uint16_t    txBuffer             = {0};
  uint16_t    rxBuffer             = {0};
  uint16_t   Dummy[1]          = {0};
  uint16_t   *ptr_Dmmy           = &Dummy;
  uint16_t   Pixel_Data[Length*Height]   = {0};      // Will hold processed pixels
  uint16_t   Gray_Data[Length*Height]   = {0};       // Will hold grayscale pixels to
     be run through Sobel edge detection
  uint16_t     *ptr_PD             = &Pixel_Data;     // Pointer to processed pixels
     array
  uint16_t   *ptr_GD             = &Gray_Data;     // Pointer to grayscale pixels array
  uint16_t     *ptr_Image_PD         = bmExampleImage.pData;  // Pointer to Pre-loaded
     image for image processing testing
  //uint16_t   *ptr_Image_PD         = &Pixel_Data;     // Pointer to Pre-loaded image
     for image processing testing
  int        start               = 0;
```

```
  int      middle              = 0;
  int        middleGStart        = 1;
  int        middleBStart        = 2;
  uint8 BayerFilter[2][Length]     = {0};


  GUI_SetBkColor(GUI_BLACK);
  GUI_Clear();
  GUI_DrawBitmap(&bmExampleImage, 0, 0);

//-------------------------- STILL IN DEVELOPEMENT----------------------------//

  /* ---------------------------Initializations---------------------------- */
  /*MyCam_Init();
  Trigger_Capture();

  while (flag != CAPTURE_COMPLETE) {
  flag = MyCam_Check_Capture_Status(rxBuffer, txBuffer);
  }

  Cy_SysLib_Delay(1000);

  GUI_BITMAP Image = {
    (unsigned int)Length,     // xSize
    (unsigned int)Height,     // ySize
    640,                   // BytesPerLine
    16,                    // BitsPerPixel
    (unsigned char *)Pixel_Data,   // Pointer to picture data
    NULL,                  // Pointer to palette
    GUI_DRAW_BMPM565,        // Specifies RGB format
  };

  Cy_SysLib_Delay(1000);

  MyCam_Pixel_Read(txBuffer, ptr_PD, start, &BayerFilter);
  middle = middleGStart;

  for (k = 0; k < Height - 1; k++) {
    ptr_PD = MyCam_Pixel_Read(txBuffer, ptr_PD, middle, &BayerFilter);
    if (middle == middleGStart)
      middle = middleBStart;
    else if (middle == middleBStart)
      middle = middleGStart;
  }

  ptr_PD = &Pixel_Data;
  GUI_DrawBitmap(&Image, 0, 0);
  Cy_SysLib_Delay(750);

  for (k = 0; k < Height - 1; k++) {
    ptr_PD = MyCam_Pixel_Read(txBuffer, ptr_PD, middle, &BayerFilter);
    if (middle == middleGStart)
      middle = middleBStart;
    else if (middle == middleBStart)
      middle = middleGStart;
  }
```

```
ptr_PD = &Pixel_Data;
GUI_DrawBitmap(&Image, 0, 80);
Cy_SysLib_Delay(750);

for (k = 0; k < Height - 1; k++) {
  ptr_PD = MyCam_Pixel_Read(txBuffer, ptr_PD, middle, &BayerFilter);
  if (middle == middleGStart)
    middle = middleBStart;
  else if (middle == middleBStart)
    middle = middleGStart;
}
ptr_PD = &Pixel_Data;
GUI_DrawBitmap(&Image, 0, 160);
Cy_SysLib_Delay(750);

/* GUI_BITMAP is a structure the EMWIN graphics driver
 * uses to display an image */

GUI_BITMAP Image = {
    (unsigned int)Length,      // xSize
    (unsigned int)Height,      // ySize
    640,                    // BytesPerLine
    16,                     // BitsPerPixel
    (unsigned char *)Pixel_Data,   // Pointer to picture data
    NULL,                   // Pointer to palette
    GUI_DRAW_BMPM565,          // Specifies RGB format
  };

/*******************************************************************************
* Function Call: conv2gray()
********************************************************************************
*
* Summary:   This function takes an image and converts it to grayscale. Each
*        time the function is called, 80 rows of pixels are evaluated and
*        the pointer to the image data is returned to be the starting
*        point of the next function call. The array that holds the processed
*        pixels is what will be displayed on the screen. This is just for
*        demonstration purposes and won't be used in the final system.
*
*        first call -> 0 - 79;
*        second call -> 80 - 159;
*        third call -> 160 - 239
*******************************************************************************/

//ptr_Image_PD = conv2gray(ptr_Image_PD, ptr_PD, ptr_GD);
//GUI_DrawBitmap(&Image, 0, 0);

//ptr_Image_PD = conv2gray(ptr_Image_PD, ptr_PD, ptr_GD);
//GUI_DrawBitmap(&Image, 0, 80);

//ptr_Image_PD = conv2gray(ptr_Image_PD, ptr_PD, ptr_GD);
//GUI_DrawBitmap(&Image, 0, 160);

//ptr_Image_PD = bmExampleImage.pData;  // Reset test image pointer to beginning

/*******************************************************************************
* Function Call: sobel()
```

```
*****************************************************************************
*
* Summary:   This function takes an image, uses conv2gray() to convert an image
*        to grayscale, and the uses sobel edge detection to detect edges
*        in the image. Each time the function is called, 80 rows of pixels
*        are evaluated and the pointer to the image data is returned to be
*        the starting point of the next function call.
*
*        first call -> 0 - 79;
*        second call -> 80 - 159;
*        third call -> 160 - 239
*****************************************************************************/

  ptr_Image_PD = sobel(ptr_Image_PD, ptr_PD, ptr_GD);
  GUI_DrawBitmap(&Image, 0, 0);

  for (k = 0; k < 1024; k++){
    Cy_SCB_UART_PutArray(KIT_UART_HW, ptr_PD, 25);
    Cy_SysLib_Delay(1);
    Cy_SCB_UART_ClearTxFifoStatus(KIT_UART_HW, CY_SCB_UART_TX_DONE);
    ptr_PD = ptr_PD + 128;
}

  ptr_PD = &Pixel_Data;

  Cy_SysLib_Delay(500);

  ptr_Image_PD = sobel(ptr_Image_PD, ptr_PD, ptr_GD);
  GUI_DrawBitmap(&Image, 0, 80);

  for (k = 0; k < 1024; k++){
    Cy_SCB_UART_PutArray(KIT_UART_HW, ptr_PD, 25);
    Cy_SysLib_Delay(1);
    Cy_SCB_UART_ClearTxFifoStatus(KIT_UART_HW, CY_SCB_UART_TX_DONE);
    ptr_PD = ptr_PD + 128;
  }

  ptr_PD = &Pixel_Data;

  Cy_SysLib_Delay(500);

  ptr_Image_PD = sobel(ptr_Image_PD, ptr_PD, ptr_GD);
  GUI_DrawBitmap(&Image, 0, 160);

  for (k = 0; k < 1024; k++){
    Cy_SCB_UART_PutArray(KIT_UART_HW, ptr_PD, 25);
    Cy_SysLib_Delay(1);
    Cy_SCB_UART_ClearTxFifoStatus(KIT_UART_HW, CY_SCB_UART_TX_DONE);
    ptr_PD = ptr_PD + 128;
  }

  ptr_PD = &Pixel_Data;

  /* -------------------Set font size, font color to black-------------------- */
  GUI_SetFont(GUI_FONT_16B_1);
  GUI_SetColor(GUI_WHITE);
```

```
    /* -------Set text mode to transparent, underlined and center aligned------- */
    GUI_SetTextMode(GUI_TM_TRANS);
    GUI_SetTextAlign(GUI_TA_HCENTER);

    /* ----------------------Print the page title text-------------------------- */
    GUI_DispStringAt("Objects Detected!!", 160, 0);

}

/*******************************************************************************
* Function Name: bool IsBtnClicked
********************************************************************************
*
* Summary:   This non-blocking function implements SW2 button click check.
*
* Parameters: None
*
* Return:  Status of the SW2 button:
*        true when button was pressed and then released and
*        false in other cases
*
*******************************************************************************/
bool IsBtnClicked(void)
{
   uint32 currBtnState;
   static uint32 prevBtnState = BTN_RELEASED;

   bool result = false;

   currBtnState = Cy_GPIO_Read(SW2_PORT, SW2_PIN);

   if((prevBtnState == BTN_RELEASED) && (currBtnState == BTN_PRESSED))
   {
      result = true;
   }
   prevBtnState = currBtnState;

   Cy_SysLib_Delay(5);

   return result;
}


/*******************************************************************************
* Function Name: int main(void)
********************************************************************************
*
* Summary:   This is the main for this code example. This function does the following
*          1. Initializes the EmWin display engine
*          2. Displays startup screen for 3 seconds
*          3. In an infinite loop, displays the following screens on
*            key press and release
*            a. grayscale and image detection
*
* Parameters: None
*
* Return:  None
```

```
*
****************************************************************************/
int main(void)
{
  uint8 pageNumber = 0;
   /* ------------------------Initializations------------------------------- */
   init_cycfg_all();
   __enable_irq();
   GUI_Init();
   initSPIMaster();
   initI2CMaster();

   cy_stc_scb_uart_context_t KIT_UART_context;

   Cy_SCB_UART_Init(KIT_UART_HW, &KIT_UART_config, &KIT_UART_context);
   Cy_SCB_UART_Enable(KIT_UART_HW);

  /* --------------Display the startup screen for 2 seconds----------------- */
  ShowStartupScreen();
  Cy_SysLib_Delay(2000);

  /* ----------------------Show Instructions Screen------------------------- */
  ShowInstructionsScreen();

  /* ------------------Display various pages in a loop---------------------- */
   for(;;)
   {
      if(IsBtnClicked())
      {
         /* Using pageNumber as index, update the display with a demo screen
           Following are the functions that are called in sequence
               ShowProcess_Image()
         */
         (*PageArray[pageNumber])();

         /* Increment page number */
         pageNumber = pageNumber + 1;

         /* If pagenumber exceeds maximum pages, reset */
         if(pageNumber >= NUMBER_OF_PAGES)
         {
            pageNumber = 0;
         }
      }
   }
}


/* [] END OF FILE */
```

D. Appendix 4 - SPImaster.c

```
/*******************************************************************************
* File Name:  SPIMaster.c
*
* Version:   1.0
*
* Description:  This file contains function definitions for SPI Master.
*           - These functions were provided from Cypress and
*             were altered to fit the Smart Eraser application.
*
* Author:  Chris Quesada
********************************************************************************/
#include "SPIMaster.h"
#include "Interface.h"
#include "stdio.h"

/*******************************************************************************
* Function Name: initMaster
********************************************************************************
*
* Summary:   This function initializes the SPI Master based on the
*        configuration done in design.modus file.
*
* Parameters:  None
*
* Return:  (uint32) INIT_SUCCESS or INIT_FAILURE
*
*******************************************************************************/
uint32 initSPIMaster(void)
{
   cy_en_scb_spi_status_t initStatus;

  /* Configure SPI block */
  initStatus = Cy_SCB_SPI_Init(mSPI_HW, &mSPI_config, NULL);

  /* If the initialization fails, return failure status */
  if(initStatus != CY_SCB_SPI_SUCCESS)
  {
    return(INIT_FAILURE);
  }

  /* Set active slave select to line 0 */
  Cy_SCB_SPI_SetActiveSlaveSelect(mSPI_HW, CY_SCB_SPI_SLAVE_SELECT0);

  /* Enable SPI master block. */
  Cy_SCB_SPI_Enable(mSPI_HW);

  /* Initialization completed */

  return(INIT_SUCCESS);
}


/*******************************************************************************
* Function Name: Write_SPI
*******************************************************************************
```

```
*
* Summary:  This function sends the data to the slave. Note that the below
*         function is blocking until all the bytes are transferred.
*
* Parameters:  (uint32 *) txBuffer - Pointer to the transmit buffer
*         (uint32) transferSize - Number of bytes to be transmitted
*
* Return:  None
*
*******************************************************************************/

uint16_t Write_SPI(uint16_t data)
{
  uint16_t readData;

  /* ----------------Send read command to the camera--------------------- */
  Cy_SCB_SPI_Write(mSPI_HW, data);

  /* -------Wait for RX buffer to get filled with 2 bytes of data----- */
  while ( CY_SCB_SPI_RX_NOT_EMPTY != (Cy_SCB_SPI_GetRxFifoStatus(mSPI_HW) &
     CY_SCB_SPI_RX_NOT_EMPTY)) {}

  readData = Cy_SCB_SPI_Read(mSPI_HW);

   /* -----------------Clear the RX and TX buffers---------------- */
  Cy_SCB_SPI_ClearTxFifo(mSPI_HW);
  Cy_SCB_SPI_ClearRxFifo(mSPI_HW);
  Cy_SCB_SPI_ClearRxFifoStatus(mSPI_HW, CY_SCB_SPI_RX_NOT_EMPTY);

  return(readData);
}


/* [] END OF FILE */
```

E. Appendix 5 - SPImaster.h

```
/*****************************************************************************
* File Name:  SPIMaster.h
*
* Version:   1.0
*
* Description:  This file contains function prototypes for SPI Master.
*          - These functions were provided from Cypress and
*             were altered to fit the Smart Eraser application.
*
* Author:  Chris Quesada
*****************************************************************************/

#ifndef SOURCE_SPIMASTER_H_
#define SOURCE_SPIMASTER_H_

#include "cy_pdl.h"
#include "cycfg.h"

/***************************************
*            Macros
****************************************/
#define MASTER_ERROR_MASK (CY_SCB_SPI_SLAVE_TRANSFER_ERR |
   CY_SCB_SPI_TRANSFER_OVERFLOW | \
                     CY_SCB_SPI_TRANSFER_UNDERFLOW)

/* -------------Transmit and receive packet information------------------ */
#define NUMBER_OF_SPI_ELEMENTS  (1UL)
#define SIZE_OF_SPI_PACKET   (NUMBER_OF_SPI_ELEMENTS * SIZE_OF_SPI_ELEMENT)
#define SIZE_OF_SPI_ELEMENT   (16UL)

/* ------Delay between successive SPI Master command transmissions------- */
#define CMD_DELAY     (1000)
#define TIMEOUT_1_MS     (1000ul)

/***************************************
*      Function Prototypes
****************************************/
uint32 initSPIMaster(void);
uint16_t Write_SPI(uint16_t txBuffer);


#endif /* SOURCE_SPIMASTER_H_ */
```

F. Appendix 6 - I2Cmaster.c

```c
/*******************************************************************************
* File Name:  I2CMaster.c
*
* Version:   1.0
*
* Description:  This file contains function definitions for I2C Master.
*           - These functions were provided from Cypress and
*             were altered to fit the Smart Eraser application.
*
* Author:  Chris Quesada
********************************************************************************/

#include "I2CMaster.h"
#include "Interface.h"

/*******************************************************************************
*         Constants
*******************************************************************************/
/* I2C slave address to communicate with */
#define I2C_SLAVE_ADDR (0x3C)

/* Buffer and packet size */
#define RX_PACKET_SIZE  (3UL)

/* Buffer and packet size */
#define RX_PACKET_SIZE (3UL)
#define BUFFER_SIZE    (PACKET_SIZE)

/* Command valid status */
#define STS_CMD_DONE   (0x00UL)
#define STS_CMD_FAIL   (0xFFUL)

/* Command valid status */
#define TRANSFER_ERROR (0xFFUL)
#define READ_ERROR     (TRANSFER_ERROR)

/* Timeout */
#define LOOP_FOREVER   (0UL)
#define I2C_TIMEOUT    (100UL)

/********************************************************************************
* Global variables
********************************************************************************/
cy_stc_scb_i2c_context_t mI2C_context;

/********************************************************************************
* Function Name: Write_I2C
*****************************************************************************//**
*
* Buffer is assigned with data to be sent to slave.
* Low level PDL APIs are used to control I2C SCB to send data.
* Errors are handled depending on the return value from the appropriate function.
*
* \param buffer
*
```

```
* \param bufferSize
*
* \return
* returns the status after command is written to slave.
* TRANSFER_ERROR is returned if any error occurs.
* TRANSFER_CMPLT is returned if write is successful.
* \ref uint32
*
*******************************************************************************/
uint8 Write_I2C(uint8* buffer, uint32 bufferSize)
{
   uint8 status = TRANSFER_ERROR;
   cy_en_scb_i2c_status_t errorStatus;

   /* Sends packets to slave using low level PDL library functions. */
   errorStatus = Cy_SCB_I2C_MasterSendStart(mI2C_HW, I2C_SLAVE_ADDR,
      CY_SCB_I2C_WRITE_XFER, I2C_TIMEOUT, &mI2C_context);
   if(errorStatus == CY_SCB_I2C_SUCCESS)
   {

      uint32 cnt = 0UL;

      /* Read data from the slave into the buffer */
      do
      {
         /* Write byte and receive ACK/NACK response */
         errorStatus = Cy_SCB_I2C_MasterWriteByte(mI2C_HW, buffer[cnt], I2C_TIMEOUT,
            &mI2C_context);
         ++cnt;
      }
      while((errorStatus == CY_SCB_I2C_SUCCESS) && (cnt < bufferSize));
   }

   /* Check status of transaction */
   if ((errorStatus == CY_SCB_I2C_SUCCESS) ||
      (errorStatus == CY_SCB_I2C_MASTER_MANUAL_NAK) ||
      (errorStatus == CY_SCB_I2C_MASTER_MANUAL_ADDR_NAK))
   {
      /* Send Stop condition on the bus */
      if (Cy_SCB_I2C_MasterSendStop(mI2C_HW, I2C_TIMEOUT, &mI2C_context) ==
         CY_SCB_I2C_SUCCESS)
      {
         status = TRANSFER_CMPLT;
      }
   }

   return (status);
}

/*******************************************************************************
* Function Name: initI2CMaster
*******************************************************************************
*
* This function initiates and enables master SCB
*
* \param None
*
```

```
* \return
* Status of initialization
*
*******************************************************************************/
uint32 initI2CMaster(void)
{
  /* Initialize the master I2C. */

   cy_en_scb_i2c_status_t initStatus;

  /* Configure component. */
  initStatus = Cy_SCB_I2C_Init(mI2C_HW, &mI2C_config, &mI2C_context);
  if(initStatus!=CY_SCB_I2C_SUCCESS)
  {
    return INIT_FAILURE;
  }

  /* Enable I2C master hardware. */
  Cy_SCB_I2C_Enable(mI2C_HW);
  return INIT_SUCCESS;
}
```

G. Appendix 7 - I2Cmaster.h

```
/*****************************************************************************
* File Name:  I2Cmaster.h
*
* Version:   1.0
*
* Description:  This file contains function prototypes for I2C Master.
*           - These functions were provided from Cypress and
*             were altered to fit the Smart Eraser application.
*
* Author:  Chris Quesada
******************************************************************************/
#ifndef SOURCE_I2CMASTER_H_
#define SOURCE_I2CMASTER_H_

#include "cy_pdl.h"
#include "cycfg.h"

/******************************************************************************
* Constants
******************************************************************************/
#define TRANSFER_CMPLT      (0x00UL)
#define READ_CMPLT          (TRANSFER_CMPLT)
#define NUMBER_OF_I2C_ELEMENTS  (3UL)
#define SIZE_OF_I2C_ELEMENT   (8UL)
#define SIZE_OF_I2C_PACKET   (NUMBER_OF_I2C_ELEMENTS * SIZE_OF_I2C_ELEMENT)
/* Packet positions */

#define I2C_ADDRESS         (0UL)
#define I2C_CONFIG       (1UL)

/******************************************************************************
* Function prototype
******************************************************************************/
uint8 Write_I2C(uint8* buffer, uint32 bufferSize);
uint32 initI2CMaster(void);

#endif /* SOURCE_I2CMASTER_H_ */
```

H. Appendix 8 - Arducam.c

```c
/*****************************************************************************
* File Name:  Arducam.c
*
* Version:   1.0
*
* Description: This file holds all the function definitions for the Arducam
*
* Author:  Chris Quesada
*****************************************************************************/

#include <Arducam.h>
#include <SPImaster.h>
#include <I2CMaster.h>
#include <interface.h>
#include <ImProc.h>
#include <math.h>

struct sensor_reg
{
  uint16_t reg;
  uint8 val;
};

struct sensor_reg OV5642_1280x960_RAW[RAWinit] =
{
{0x00ff,0x01},
{0x3008,0x80},
{0x3103,0x93},
{0x3008,0x02},
{0x3017,0x7f},
{0x3018,0xf0},
{0x3615,0xf0},
{0x3000,0xF8},
{0x3001,0x48},
{0x3002,0x5c},
{0x3003,0x02},
{0x3005,0xB7},
{0x3006,0x43},
{0x3007,0x37},
{0x300f,0x06},
{0x3011,0x08},
{0x3010,0x20},
{0x3012,0x00},
{0x460c,0x22},
{0x3815,0x04},
{0x370c,0xA0},
{0x3602,0xFC},
{0x3612,0xFF},
{0x3634,0xC0},
{0x3613,0x00},
{0x3622,0x00},
{0x3603,0x27},
{0x4000,0x21},
{0x401D,0x02},
{0x3600,0x54},
```

```
{0x3605,0x04},
{0x3606,0x3F},
{0x5020,0x04},
{0x5197,0x01},
{0x5001,0xFF},
{0x5500,0x10},
{0x5502,0x00},
{0x5503,0x04},
{0x5504,0x00},
{0x5505,0x7F},
{0x5080,0x08},
{0x300E,0x18},
{0x4610,0x00},
{0x471D,0x05},
{0x4708,0x06},
{0x3710,0x10},
{0x3632,0x41},
{0x3631,0x01},
{0x501F,0x03},
{0x3604,0x40},
{0x4300,0x00},
{0x3824,0x11},
{0x5000,0x4F},
{0x3818,0xC1},
{0x3705,0xDB},
{0x370A,0x81},
{0x3621,0xC7},
{0x3800,0x03},
{0x3801,0xE8},
{0x3802,0x03},
{0x3803,0xE8},
{0x3804,0x38},
{0x3805,0x00},
{0x3806,0x03},
{0x3807,0xC0},
{0x3808,0x05},
{0x3809,0x00},
{0x380A,0x03},
{0x380B,0xC0},
{0x380C,0x0A},
{0x380D,0xF0},
{0x380E,0x03},
{0x380F,0xE8},
{0x3827,0x08},
{0x3810,0xC0},
{0x5683,0x00},
{0x5686,0x03},
{0x5687,0xC0},
{0x3A1A,0x04},
{0x3A13,0x30},
{0x3004,0xDF},
{0x350C,0x07},
{0x350D,0xD0},
{0x3500,0x35},
{0x3501,0x00},
{0x3502,0x00},
{0x350A,0x00},
```

```
{0x350B,0x00},
{0x3503,0x00},
{0x5682,0x05},
{0x3A0F,0x78},
{0x3A11,0xD0},
{0x3A1B,0x7A},
{0x3A1E,0x66},
{0x3A1F,0x40},
{0x3A10,0x68},
{0x3030,0x0B},
{0x3A01,0x04},
{0x3A02,0x00},
{0x3A03,0x78},
{0x3A04,0x00},
{0x3A05,0x30},
{0x3A14,0x00},
{0x3A15,0x64},
{0x3A16,0x00},
{0x3A17,0x89},
{0x3A18,0x00},
{0x3A19,0x70},
{0x3A00,0x78},
{0x3A08,0x12},
{0x3A09,0xC0},
{0x3A0A,0x0F},
{0x3A0B,0xA0},
{0x3A0D,0x04},
{0x3A0E,0x03},
{0x3C00,0x04},
{0x3C01,0xB4},
{0x5688,0xFD},
{0x5689,0xDF},
{0x568A,0xFE},
{0x568B,0xEF},
{0x568C,0xFE},
{0x568D,0xEF},
{0x568E,0xAA},
{0x568F,0xAA},
{0x589B,0x04},
{0x589A,0xC5},
{0x528A,0x00},
{0x528B,0x02},
{0x528C,0x08},
{0x528D,0x10},
{0x528E,0x20},
{0x528F,0x28},
{0x5290,0x30},
{0x5292,0x00},
{0x5293,0x00},
{0x5294,0x00},
{0x5295,0x02},
{0x5296,0x00},
{0x5297,0x08},
{0x5298,0x00},
{0x5299,0x10},
{0x529A,0x00},
{0x529B,0x20},
```

```
{0x529C,0x00},
{0x529D,0x28},
{0x529E,0x00},
{0x5282,0x00},
{0x529F,0x30},
{0x5300,0x00},
{0x5302,0x00},
{0x5303,0x7C},
{0x530C,0x00},
{0x530D,0x0C},
{0x530E,0x20},
{0x530F,0x80},
{0x5310,0x20},
{0x5311,0x80},
{0x5308,0x20},
{0x5309,0x40},
{0x5304,0x00},
{0x5305,0x30},
{0x5306,0x00},
{0x5307,0x80},
{0x5314,0x08},
{0x5315,0x20},
{0x5319,0x30},
{0x5316,0x10},
{0x5317,0x08},
{0x5318,0x02},
{0x5380,0x01},
{0x5381,0x20},
{0x5382,0x00},
{0x5383,0x4E},
{0x5384,0x00},
{0x5385,0x0F},
{0x5386,0x00},
{0x5387,0x00},
{0x5388,0x01},
{0x5389,0x15},
{0x538A,0x00},
{0x538B,0x31},
{0x538C,0x00},
{0x538D,0x00},
{0x538E,0x00},
{0x538F,0x0F},
{0x5390,0x00},
{0x5391,0xAB},
{0x5392,0x00},
{0x5393,0xA2},
{0x5394,0x08},
{0x5301,0x20},
{0x5480,0x14},
{0x5482,0x03},
{0x5483,0x57},
{0x5484,0x65},
{0x5485,0x71},
{0x5481,0x21},
{0x5486,0x7D},
{0x5487,0x87},
{0x5488,0x91},
```

```
{0x5489,0x9A},
{0x548A,0xAA},
{0x548B,0xB8},
{0x548C,0xCD},
{0x548D,0xDD},
{0x548E,0xEA},
{0x548F,0x10},
{0x5490,0x05},
{0x5491,0x00},
{0x5492,0x04},
{0x5493,0x20},
{0x5494,0x03},
{0x5495,0x60},
{0x5496,0x02},
{0x5497,0xB8},
{0x5498,0x02},
{0x5499,0x86},
{0x549A,0x02},
{0x549B,0x5B},
{0x549C,0x02},
{0x549D,0x3B},
{0x549E,0x02},
{0x549F,0x1C},
{0x54A0,0x02},
{0x54A1,0x04},
{0x54A2,0x01},
{0x54A3,0xED},
{0x54A4,0x01},
{0x54A5,0xC5},
{0x54A6,0x01},
{0x54A7,0xA5},
{0x54A8,0x01},
{0x54A9,0x6C},
{0x54AA,0x01},
{0x54AB,0x41},
{0x54AC,0x01},
{0x54AD,0x20},
{0x54AE,0x00},
{0x54AF,0x16},
{0x3406,0x00},
{0x5192,0x04},
{0x5191,0xF8},
{0x5193,0x70},
{0x5194,0xF0},
{0x5195,0xF0},
{0x518D,0x3D},
{0x518F,0x54},
{0x518E,0x3D},
{0x5190,0x54},
{0x518B,0xC0},
{0x518C,0xBD},
{0x5187,0x18},
{0x5188,0x18},
{0x5189,0x6E},
{0x518A,0x68},
{0x5186,0x1C},
{0x5181,0x50},
```

```
{0x5182,0x11},
{0x5183,0x14},
{0x5184,0x25},
{0x5185,0x24},
{0x5025,0x82},
{0x5583,0x40},
{0x5584,0x40},
{0x5580,0x02},
{0x3633,0x07},
{0x3702,0x10},
{0x3703,0xB2},
{0x3704,0x18},
{0x370B,0x40},
{0x370D,0x02},
{0x3620,0x52},

{0xffff,0xff},
};

struct sensor_reg OV5642_320x240_RAW[RAWres] =
{
  /*
{0x3800,0x03},
{0x3801,0xE8},
{0x3802,0x03},
{0x3803,0xE8},
{0x3804,0x38},
{0x3805,0x00},
{0x3806,0x03},
{0x3807,0xC0},
*/
{0x3808,0x01},
{0x3809,0x40},
{0x380A,0x00},
{0x380B,0xf0},
{0xffff,0xff},
};



/*******************************************************************************
* Function Name: MyCam_Test
********************************************************************************
*
* Summary:   Tests SPI communication between PSoC 6 and Arducam
*
* Parameters:  uint16_t *rxBuffer,
*        uint16_t *txBuffer
*
* Return:  Value in test register of Arducam (rxBuffer[0])
*
*******************************************************************************/
void MyCam_Test(uint16_t rxBuffer, uint16_t txBuffer) {

  uint16_t response = 0;

  do {
```

```
    txBuffer = (((uint16_t)WRITE + (uint16_t)TEST_SPI) << 8) + (uint16_t)TEST_VALUE;
    Write_SPI(txBuffer);
    txBuffer = (((uint16_t)READ + (uint16_t)TEST_SPI) << 8) + (uint16_t)DUMMY;
    response = Write_SPI(txBuffer);
    } while ( response != TEST_VALUE);
}

/*******************************************************************************
* Function Name: MyCam_Trigger
********************************************************************************
*
* Summary:   Clears FIFO Flag and Sends trigger signal to have camera
*        capture image.
*
* Parameters:  uint16_t *rxBuffer,
*        uint16_t *txBuffer
*
* Return:  none
*
*******************************************************************************/
void MyCam_Trigger(uint16_t rxBuffer, uint16_t txBuffer) {

    txBuffer = (((uint16_t)WRITE + (uint16_t)FIFO_CONTROL) << 8) +
        (uint16_t)CLEAR_FIFO_FLAG;
    Write_SPI(txBuffer);
    Cy_SysLib_Delay(0.5);

    txBuffer = (((uint16_t)WRITE + (uint16_t)FIFO_CONTROL) << 8) +
        (uint16_t)CLEAR_FIFO_FLAG;
    Write_SPI(txBuffer);
    Cy_SysLib_Delay(0.5);

    txBuffer = (((uint16_t)WRITE + (uint16_t)FIFO_CONTROL) << 8) + (uint16_t)CAPTURE;
    Write_SPI(txBuffer);
    Cy_SysLib_Delay(0.5);
}

/*******************************************************************************
* Function Name: MyCam_Check_Capture_Status
********************************************************************************
*
* Summary:   Check capture flag to see if capture is complete
*
* Parameters:  uint16_t *rxBuffer,
*        uint16_t *txBuffer
*
* Return:  none
*
*******************************************************************************/
uint16_t MyCam_Check_Capture_Status(uint16_t rxBuffer, uint16_t txBuffer) {

    uint16_t response = 0;

    txBuffer = (((uint16_t)READ + (uint16_t)CAPTURE_FLAG) << 8) + (uint16_t)DUMMY;
    response = Write_SPI(txBuffer);

    return response;
```

```
}
/****************************************************************************
* Function Name: MyCam_Single_FIFO_Read
****************************************************************************
*
* Summary:   Reads 2 bytes of data from cam and them attempts to demosaic
*        the raw data, still in development
*
* Parameters:  uint16_t *rxBuffer,
*        uint16_t *txBuffer
*
* Return:  none
*
****************************************************************************/
unsigned short* MyCam_Pixel_Read(uint16_t txBuffer, uint16_t *ptr_PD, int position,
   uint8 *BayerFilter) {

    uint16_t Red = 0;
    uint16_t Green1 = 0;
    uint16_t Green2 = 0;
    uint16_t Blue = 0;
    uint8 *start = BayerFilter;
    int j = 0;
    int k = 0;


    if (position == 0) {
      for (int j = 0; j < 1; j++) {
        for (int k = 0; k < Length; k++) {
        txBuffer = (((uint16_t)READ + (uint16_t)FIFO_READ_SINGLE) << 8) +
           (uint16_t)DUMMY;
        *BayerFilter = Write_SPI(txBuffer);
        BayerFilter = BayerFilter + 1;
        }
      }
      for (int k = 0; k < Length - 1; k++) {
        if (k % 2 == 0) {
          Blue  = *BayerFilter >> 3;
          Green1  = (*BayerFilter + 1) >> 2 << 5;
          Green2  = (*BayerFilter + Length) >> 2 << 5;
          Red  = (*BayerFilter + Length + 1) >> 3 << 11;
          *ptr_PD = Blue + ((Green1 + Green2) / 2) + Red;
          ptr_PD = ptr_PD + 1;
          BayerFilter = BayerFilter + 1;
          }
        else if (k % 2 != 0) {
          Green1  = *BayerFilter >> 2 << 5;
          Blue  = (*BayerFilter + 1) >> 3;
          Red  = (*BayerFilter + Length) >> 3 << 11;
          Green2  = (*BayerFilter + Length + 1) >> 2 << 5;
          *ptr_PD = Blue + ((Green1 + Green2) / 2) + Red;
          ptr_PD = ptr_PD + 1;
          BayerFilter = BayerFilter + 1;
          }
        }
    ptr_PD = ptr_PD + 1;
    BayerFilter = BayerFilter + 1;
```

```
Green1  = *BayerFilter >> 2 << 5;
Blue  = (*BayerFilter - 1) >> 3;
Red  = (*BayerFilter - Length) >> 3 << 11;
Green2  = (*BayerFilter - Length - 1) >> 2 << 5;
*ptr_PD = Blue + ((Green1 + Green2) / 2) + Red;
}

else if (position == 1) {

  for (k = 0; k < Length; k++) {
    *BayerFilter = *BayerFilter + Length;
    BayerFilter = BayerFilter + 1;
  }

  for (int k = 0; k < Length; k++) {
    txBuffer = (((uint16_t)READ + (uint16_t)FIFO_READ_SINGLE) << 8) +
       (uint16_t)DUMMY;
    *BayerFilter = Write_SPI(txBuffer);
    BayerFilter = BayerFilter + 1;
  }

  BayerFilter = start;

  for (int k = 0; k < Length - 1; k++) {
    if (k % 2 == 0) {
      Blue  = *BayerFilter >> 3;
      Green1  = (*BayerFilter + 1) >> 2 << 5;
      Green2  = (*BayerFilter + Length) >> 2 << 5;
      Red  = (*BayerFilter + Length + 1) >> 3 << 11;
      *ptr_PD = Blue + ((Green1 + Green2) / 2) + Red;
      ptr_PD = ptr_PD + 1;
      BayerFilter = BayerFilter + 1;
      }
    else if (k % 2 != 0) {
      Green1  = *BayerFilter >> 2 << 5;
      Blue  = (*BayerFilter + 1) >> 3;
      Red  = (*BayerFilter + Length) >> 3 << 11;
      Green2  = (*BayerFilter + Length + 1) >> 2 << 5;
      *ptr_PD = Blue + ((Green1 + Green2) / 2) + Red;
      ptr_PD = ptr_PD + 1;
      BayerFilter = BayerFilter + 1;
      }
    }
ptr_PD = ptr_PD + 1;
BayerFilter = BayerFilter + 1;
Green1  = *BayerFilter >> 2 << 5;
Blue  = (*BayerFilter - 1) >> 3 << 11;
Red  = (*BayerFilter - Length) >> 3 << 11;
Green2  = (*BayerFilter - Length - 1) >> 2 << 5;
*ptr_PD = Blue + ((Green1 + Green2) / 2) + Red;
}

else if (position == 2) {

  for (k = 0; k < Length; k++) {
    *BayerFilter = *BayerFilter + Length;
    BayerFilter = BayerFilter + 1;
```

```
        }

        for (int k = 0; k < Length; k++) {
          txBuffer = (((uint16_t)READ + (uint16_t)FIFO_READ_SINGLE) << 8) +
              (uint16_t)DUMMY;
          *BayerFilter = Write_SPI(txBuffer);
          BayerFilter = BayerFilter + 1;
        }

        BayerFilter = start;

        for (int k = 0; k < Length - 1; k++) {
          if (k % 2 == 0) {
            Green1  = *BayerFilter >> 2 << 5;
            Red  = (*BayerFilter + 1) >> 3 << 11;
            Blue  = (*BayerFilter + Length)>> 3;
            Green2  = (*BayerFilter + Length + 1)>> 2 << 5;
            *ptr_PD = Blue + ((Green1 + Green2) / 2) + Red;
            ptr_PD = ptr_PD + 1;
            BayerFilter = BayerFilter + 1;
            }
          else if (k % 2 != 0) {
            Red  = *BayerFilter >> 3 << 11;
            Green1  = (*BayerFilter + 1) >> 2 << 5;
            Green2  = (*BayerFilter + Length) >> 2 << 5;
            Blue  = (*BayerFilter + Length + 1) >> 3;
            *ptr_PD = Blue + ((Green1 + Green2) / 2) + Red;
            ptr_PD = ptr_PD + 1;
            BayerFilter = BayerFilter + 1;
            }
          }
      ptr_PD = ptr_PD + 1;
      BayerFilter = BayerFilter + 1;
      Red  = *BayerFilter >> 3 << 11;
      Green1  = (*BayerFilter - 1) >> 2 << 5;
      Green2  = (*BayerFilter - Length) >> 2 << 5;
      Blue  = (*BayerFilter - Length - 1) >> 3;
      *ptr_PD = Blue + ((Green1 + Green2) / 2) + Red;
      }
      return ptr_PD;
}

/*****************************************************************************
* Function Name: MyCam_Single_Dummy_Read
*****************************************************************************
*
* Summary:   Handles the first dummy byte of RGB565 data
*
* Parameters:  uint16_t *rxBuffer,
*        uint16_t *txBuffer
*
* Return:  none
*
*****************************************************************************/
void MyCam_Single_DUMMY_Read(uint16_t txBuffer, uint16_t *ptr_PD) {
   Write_SPI(txBuffer);
}
```

```
/******************************************************************************
* Function Name: MyCamInit
*******************************************************************************
*
* Summary:   Initializes camera sensor to RGB565 320x240
*
* Parameters:  None
*
* Return:  None
*
******************************************************************************/

void MyCam_Init(void) {

  int k                     = 0;
  uint8 buffer[NUMBER_OF_I2C_ELEMENTS]  = {0};

  for (k = 0; k < RAWinit; k++) {
    buffer[0] = OV5642_1280x960_RAW[k].reg >> 8;
    buffer[1] = OV5642_1280x960_RAW[k].reg;
    buffer[2] = OV5642_1280x960_RAW[k].val;
    Write_I2C(buffer, NUMBER_OF_I2C_ELEMENTS);
  }
  Cy_SysLib_Delay(1000);

  for (k = 0; k < RAWres; k++) {
    buffer[0] = OV5642_320x240_RAW[k].reg >> 8;
    buffer[1] = OV5642_320x240_RAW[k].reg;
    buffer[2] = OV5642_320x240_RAW[k].val;
    Write_I2C(buffer, NUMBER_OF_I2C_ELEMENTS);
  }
  Cy_SysLib_Delay(250);
};

/******************************************************************************
* Function Name: handle_error
*******************************************************************************
*
* Summary:   This is a blocking function. It disables the interrupt and waits
*        in an infinite loop. This function is called when an error is
*        encountered during initialization of the blocks or during
*        SPI communication. This was provided by Cypress
*
* Parameters:  None
*
* Return:  None
*
******************************************************************************/
void handle_error(void)
{
   /* Disable all interrupts. */
  __disable_irq();

  /* Infinite loop. */
  while(1u) {}
}
```

I. Appendix 9 - Arducam.h

```
/*****************************************************************************
* File Name:  Arducam.h
*
* Version:   1.0
*
* Description: This file holds all the function prototypes for the Arducam
*
* Author:  Chris Quesada
******************************************************************************/

#ifndef ARDUCAM_H
#define ARDUCAM_H

#include "cy_pdl.h"
#include "cycfg.h"
#include "ImProc.h"

#define RAWinit             (274UL)
#define RAWres              (5UL)

/* --------------------SPI Command Addresses for Arducam---------------- */

#define TEST_SPI       (0x0)
#define TEST_VALUE       (0x59)
#define SET_FRAME_COUNT     (0x01)
#define FIFO_CONTROL    (0x04)
#define CAPTURE_FLAG    (0x41)
#define FIFO_READ_SINGLE    (0x3d)

/* ---------------SPI Command operations for Arducam-------------------- */
#define WRITE          (0x80)
#define READ           (0x00)
#define DUMMY          (0x00)

/* ------------------SPI Command Data for Arducam----------------------- */
#define CLEAR_FIFO_FLAG     (0x01)
#define CAPTURE           (0x02)
#define CAPTURE_COMPLETE    (0x08)

void MyCam_Test(uint16_t rxBuffer, uint16_t txBuffer);
void MyCam_Trigger(uint16_t rxBuffer, uint16_t txBuffer);
void MyCam_Init(void);
unsigned short* MyCam_Pixel_Read(uint16_t txBuffer, uint16_t *ptr_PD, int position,
   uint8 *BayerFilter);
void MyCam_Single_DUMMY_Read(uint16_t txBuffer, uint16_t *ptr_PD);
uint16_t MyCam_Check_Capture_Status(uint16_t rxBuffer, uint16_t txBuffer);

#endif
```

J. Appendix 10 - Improc.c

```
/*******************************************************************************
* File Name:  ImProc.c
*
* Version:   1.0
*
* Description: This file contains the function definitions for image processing.
*        Only 80 rows of an image can be evaluated at a time. The expected
*        resolution is 320 x 240.
*
* Author:  Chris Quesada
********************************************************************************/

#include <ImProc.h>

/********************************************************************************
* Function Name: conv2gray(void)
*********************************************************************************
*
* Summary:   Converts an image to grayscale
*
* Parameters: uint16_t *ptr_Image_PD (points to Image data, unprocessed pixels),
*          uint16_t *ptr_PD (points to array holding processed pixels),
*          uint16_t *ptr_GD (points to array holding grayscale pixels)
*
* Return:  Current pointer position to orignal pixel data (*ptr_Image_PD)
********************************************************************************/
unsigned short* conv2gray( uint16_t *ptr_Image_PD, uint16_t *ptr_PD, uint16_t *ptr_GD
   ) {

  uint16_t  pixel     = 0;                              // original pixel data ->
     processed pixel data
  uint8  R        = 0;                              // Red intensity
  uint8  G        = 0;                              // Green intensity
  uint8  B        = 0;                              // Blue intensity
  uint16_t R_processed  = 0;                            // "Gray" red intensity
  uint16_t G_processed  = 0;                            // "Gray" green intensity
  uint16_t B_processed  = 0;                            // "Gray" blue intensity
  double  Gray_pixel   = 0;                             // Stores calculated gray
     intensity
  int    k        = 0;                              // counter

  for (k = 0 ; k < Length*Height ; k++) {
    pixel      = *ptr_Image_PD;                     // Pull image data
    R          = ((pixel & 0b1111100000000000) >> 11 << 3);    // Isolate red
       intensity, scale to 8-bits (0-255 scale)
    G          = ((pixel & 0b0000011111100000) >> 5 << 2);     // Isolate green
       intensity, scale to 8-bits (0-255 scale)
    B          = (pixel & 0b0000000000011111) << 3;      // Isolate blue intensity,
       scale to 8-bits (0-255 scale)
    Gray_pixel    = (R * 0.2126) + (G * 0.7152) + (B * 0.0722);    // Grayscale
       conversion algorithm, Luminance method (BT.709)
    R_processed    = ((uint16_t)Gray_pixel)>> 3 << 11;       // descale to 5-bits,
       deisolate and store "gray" red intensity
    G_processed    = ((uint16_t)Gray_pixel)>> 2 << 5;        // descale to 6-bits,
       deisolate and store "gray" green intensity
```

```
    B_processed   = (uint16_t)Gray_pixel >> 3;          // descale to 5-bits,
       deisolate and store "gray" blue intensity
    pixel         = (uint16_t)(R_processed + G_processed + B_processed);  // add gray
       intensities together to form grayscale pixel
    *ptr_GD       = pixel;                             // update grayscale data array
    *ptr_PD     = pixel;                             // update pixel data array
    ptr_GD        = ptr_GD + 1;                          // increment grayscale pointer
    ptr_PD        = ptr_PD + 1;                          // increment pixel pointer
    ptr_Image_PD   = ptr_Image_PD + 1;                     // increment image pointer
  }
  return ptr_Image_PD;                                 // returns current position in
     image array
}

/*********************************************************************************
* Function Name: sobel(void)
**********************************************************************************
*
* Summary:   Takes image data, calls conv2gray(), and detects edges using
*         Sobel operator.
*
* Parameters: uint16_t *ptr_Image_PD (points to Image data, unprocessed pixels),
*          uint16_t *ptr_PD (points to array holding processed pixels),
*          uint16_t *ptr_GD (points to array holding grayscale pixels)
*
* Return:  Current pointer position to orignal pixel data (*ptr_Image_PD)
*********************************************************************************/
unsigned short* sobel( uint16_t *ptr_Image_PD, uint16_t *ptr_PD, uint16_t *ptr_GD ) {

  int k                  = 0;
  int bound              = Length*Height;             // Specifies how many rows are
     being evaluated
  uint16_t *end          = conv2gray(ptr_Image_PD, ptr_PD, ptr_GD);  // last row
     evaluated

  /* **************************************************************************
   * Sobel kernel variant, negatives will be accounted for
   * in actual calculation. By using an array of {1,2,1} instead of all 9
   * values, as shown below, you can use the same kernel for both vertical and
      horizontal
   * gradients.
   *
   *          | -1| 0 | 1 |
   *          | -2| 0 | 2 |
   *          | -1| 0 | 1 |
   *
   * *************************************************************************/
  int sobel_kernel[3]     = { 1, 2, 1};

  float Gx               = 0;                          // Horizontal gradient
  float Gy               = 0;                          // Vertical gradient
  float G                = 0;                          // Magnitude of gradient

  /* **************************************************************************
   * Start kernel in correct location (not on a bound/edge of image array)
   *
   *          |bnd|bnd|bnd|
```

```
 *          |bnd| X | |
 *          |bnd| |   |
 *
 * ***********************************************************************/
ptr_GD                  = ptr_GD + (Length + 1);

/* ***********************************************************************
 * Evaluate area next to upper bound to keep image consistent
 * ***********************************************************************/
for (k = 1; k < Length; k++){
  if (*(ptr_PD + Length) == 0xFFF){
       *ptr_PD = 0xFFF;
       }
       else {
         *ptr_PD = 0x0000;
         ptr_PD = ptr_PD + 1;
       }
  }

ptr_PD = ptr_PD + 2;                              // Align pointer after upper
   bound evaluation

/* ***********************************************************************
 * Account for top, bottom and sides of gray pixel array data so that the
 * sobel kernel convolution does not go out of bounds.
 * ***********************************************************************/
for (k = Length + 1; k < bound - Length; k++){

  if ((k + 1) % Length == 0) {
    ptr_GD = ptr_GD + 2;
    ptr_PD = ptr_PD + 2;
    k = k + 2;
  }

  /* ----------------Horizontal gradient calculation----------------- */
  Gx = (sobel_kernel[0])*(*(ptr_GD - Length + 1))  +  (sobel_kernel[1])*(*(ptr_GD
     + 1))     +
     (sobel_kernel[2])*(*(ptr_GD + Length + 1))  -  (sobel_kernel[0])*(*(ptr_GD -
        Length - 1))  -
     (sobel_kernel[1])*(*(ptr_GD - 1))    -  (sobel_kernel[2])*(*(ptr_GD + Length
        - 1));

  /* ----------------Vertical gradient calculation------------------- */
  Gy = (sobel_kernel[0])*(*(ptr_GD - Length + 1))  +  (sobel_kernel[1])*(*(ptr_GD
     - Length))    +
     (sobel_kernel[2])*(*(ptr_GD - Length - 1))  -  (sobel_kernel[0])*(*(ptr_GD +
        Length + 1))  -
     (sobel_kernel[1])*(*(ptr_GD + Length))   -  (sobel_kernel[2])*(*(ptr_GD +
        Length - 1));

  /* -------------------Magnitude of gradient----------------------- */
   G = (sqrt(pow(Gx,2) + pow(Gy,2)));

   /* -----------store gradient to pixel data array------------------ */
  *ptr_PD = (uint16_t)G;

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

```
     * Check gradient against threshold value (threshold can be adjusted
     * in ImProc.h). Depending on whether or not the gradient is less than
     * or greater than, pixels will be set to either black or white.
     *******************************************************************/
    if (*ptr_PD > Threshold) {
           *ptr_PD = 0xFFFF;
        }
        else {
           *ptr_PD = 0x0000;
        }

    /* --------------------Increment necessary pointers---------------- */

    ptr_GD = ptr_GD + 1;
    ptr_PD = ptr_PD + 1;

  }

  /* *********************************************************************
   * Evaluate area next to lower bound to keep image consistent
   * *******************************************************************/

  for (k = bound - Length ; k < bound; k++){
    if (*(ptr_PD - Length) == 0xFFF){
      *ptr_PD = 0xFFF;
      }
      else {
        *ptr_PD = 0x0000;
        ptr_PD = ptr_PD + 1;
      }
  }
  return end;

}

/* [] END OF FILE */
```

K. Appendix 11 - Improc.h

```
/*****************************************************************************
* File Name:  ImProc.h
*
* Version:   1.0
*
* Description:  This file contains function prototypes for image processing.
*
* Author:  Chris Quesada
*****************************************************************************/
#ifndef IMPROC_H
#define IMPROC_H


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <stdint.h>
#include "cy_pdl.h"
#include "cycfg.h"

/* Image Processing Parameters */
#define Max_Length       (320)
#define Max_Height       (240)
#define Length           (320)
#define Height         (80)
#define Pixel_Count_1    (25600)
#define Pixel_Count_2    (16960)
#define Threshold      (40000)

//Prototype of functions

//allocation of memory
unsigned short* conv2gray( uint16_t *ptr_Image_PD, uint16_t *ptr_PD,uint16_t *ptr_GD
   ) ;
unsigned short* sobel( uint16_t *ptr_Image_PD, uint16_t *ptr_PD, uint16_t *ptr_GD ) ;

#endif
```

L. Appendix 12 - interface.h

```
/*****************************************************************************
* File Name:  SPIMaster.c
*
* Version:   1.0
*
* Description:  This file contains macros for various files.
*           - This file was provided by Cypress and modified to fiy
*             the Smart Eraser System.
*
* Author:  Chris Quesada
*****************************************************************************/
#ifndef SOURCE_INTERFACE_H_
#define SOURCE_INTERFACE_H_

#include "cy_pdl.h"
#include "cycfg.h"

/* I2C & SPI statuses */
#define INIT_SUCCESS    (0UL)
#define INIT_FAILURE    (1UL)

/* Communication status */
#define TRANSFER_COMPLETE   (0)
#define TRANSFER_FAILURE  (1)
#define TRANSFER_IN_PROGRESS (2)
#define IDLE        (3)

/* Element index in the packet */
#define PACKET_CMD_POS   (0UL)
#define PACKET_DAT_POS   (1UL)

void handle_error(void);

#endif
```

M. Appendix 13 - square_detection.c

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

#define MAX_NAME_SZ 256

int main() {
   /* Initalize variables */
   int picture[10][10]= {
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, //row 1
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, //row 2
     {0, 0, 0, 0, 1, 1, 1, 1, 0, 0}, //row 3
     {0, 0, 0, 0, 0, 1, 1, 1, 0, 0}, //row 4
     {0, 0, 0, 0, 0, 1, 1, 1, 0, 0}, //row 5
     {0, 0, 0, 0, 0, 1, 1, 0, 0, 0}, //row 6
     {0, 0, 0, 0, 1, 1, 1, 0, 0, 0}, //row 7
     {0, 0, 0, 0, 1, 1, 1, 1, 0, 0}, //row 8
     {0, 0, 0, 0, 0, 1, 1, 1, 0, 0}, //row 9
     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} //row 10
     };
  int  coord[4]; //will contain the final coordinates (coord[top, left, bottom,
     right])
  int top = 11;   //top-most row (11 is because it will never be 11; used as a
     comparison value to see if initial 1 has been found yet)
  int left = 11;  //left-most column
  int bottom = 11;  //bottom-most row
  int right = 11;  //right-most column
  int i, j;

  for(i = 0; i < 10; i++){ //row loop (matters for top and bottom)
    for(j = 0; j < 10; j++){ //column loop (matters for left and right)
      if(picture[i][j] == 1){
        if(top == 11 && left == 11 && bottom == 11 && right == 11){ //if the coord
           don't have values yet (start as NULL), give them the first coord to base
           the rest of the comparisons off of
          top = i;
          left = j;
          bottom = i;
          right = j;
        }
        else{ //if the coord do have values in them, begin comparing to find the
           outermost values
          if(i < top) top = i;
          else if (i > bottom) bottom = i;
          else if (j < left) left = j;
          else if (j > right) right = j;
        }
      }

    }

  }
/* top++;  //indexes values to where they would be on scale of 1 - 10
  left++;
  bottom++;
```

```
    right++; */
    coord[0]= top;
    coord[1]= left;
    coord[2]= bottom;
    coord[3]= right;
     /* Prints matrix. */
     printf("Here is your matrix:\n");
    for (i=0; i<10; i++)
    {
       for(j=0; j<10; j++)
          {
           printf("%d ", picture[i][j]);
          }
       printf("<< row %d", i);
      printf("\n");
     }

     printf("\nThese are the values that were found: \ntop-most = %d \nleft-most = %d
         \nbottom-most = %d \nright-most = %d\n\n", top, left, bottom, right);

     printf("Therefore, the coordinates to be sent to the Arduino are:\nCoord = [");
    for (i=0; i<4; i++)
    {
       printf("%d ", coord[i]);
     }
     printf("].");

     /* Print Message. */
     printf("\n\nI hope your values are what they were supposed to be!\n\n");

     /* Pause the Console */
     system("pause");

     /* Free memory and exit. */
     return 0;
}
```