Table of Contents

ROBOTIC MAZE-MAPPING AS A JUNIOR-LEVEL DESIGN PROJECT

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Olivia Rae Gustafson

MEng Field Advisor: Dr. Bruce Land

Degree Date: May 2015

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Robotic Maze-Mapping as a Junior-Level Design Project

Author: Olivia Rae Gustafson

Abstract: The Electrical and Computer Engineering Department wished to revamp the project of its junior-level design. The goal of this design project is to create and set up a course project (CP) that draws upon skills from the three sophomore-level core courses (analog circuits, signal processing, and digital logic) and organize the CP so that it has appropriate difficulty for junior students. Secondary goals are for the CP to have a heavy teamwork component as well as an impressive demo. The chosen CP topic is a robotic maze-mapping scenario, where a robot is let loose in a maze and wanders around to create a cohesive map of the entire area. In this design project, the CP will be embellished and fleshed out to have a complete prototype to serve as a demo and example for the class in the spring. The CP will also be organized in such a way so that the CP has appropriate difficulty and interest level to the students. Design choices and responses of the students taking the course in the spring will be used to create an overall picture of the results of this design project. At the end of this design project, the junior-level design course will have an interesting CP with an appropriate level of challenge and teamwork. Having a more interesting and breadth-oriented CP will create better managers, project organizers, and team members of the students who take the course.

Executive Summary

The Electrical and Computer Engineering Department wished to revamp the project of its junior-level design. The goal of this design project is to create and set up a course project (CP) that draws upon skills from the three sophomore-level core courses (analog circuits, signal processing, and digital logic) and organize the CP so that it has appropriate difficulty for junior students. Secondary goals are for the CP to have a heavy teamwork component as well as an impressive demo.

The chosen CP topic is a robotic maze-mapping scenario, where a robot is let loose in a maze and wanders around to create a cohesive map of the entire area. In this design project, the CP will be embellished and fleshed out to have a complete prototype to serve as a demo and example for the class in the spring. The CP will also be organized in such a way so that the CP has appropriate difficulty and interest level to the students. Design choices and responses of the students taking the course in the spring will be used to create an overall picture of the results of this design project.

Having a demo robot was desired by the department to have during Orientation Week as a way to get engineering freshman interested in and excited about ECE. The summer demo was created by all four students (including the author) that were involved over the summer; they are acknowledged at the end of this paper. The robot was fully functional aside from the lack of microphone circuit (implemented separately but never integrated); the base station was fully functional as well. However, the radio communication between them was not functional until several weeks after the Orientation Week demo.

During the lab weeks before Spring Break, the students worked on four structured labs, written by me. The first served to get them acquainted with the Arduino. The second was the creation of the microphone circuit that resulted in a detection of a 660 Hz tone (used for the competition). The third was coding the FPGA to display maze information on a monitor. The fourth and final lab revolved around the radio communication between the robot and base station. The lab handouts and other course handouts related to lab and the course project were written by me.

A second demo robot, Dagmar, was created by me during the Spring semester using the guidelines given to the students. It was very similar to the original demo robot, but it had three IR sensors, a new chassis, and completely different base station code.

The maze-mapping competition was at the end of the semester; all students took part. The competition has four rounds. Every team participates in the first two, and then the eight teams with the highest cumulative points proceed to the third round. The highest four performers then proceed to the final, fourth round. Each round had a different maze layout. Competition maze layouts, grading rubrics, and general competition coordination was done by me.

The project was a great success this year. Students were invested in creating the best robot they could, brainstorming and implementing new ideas to make their robot more robust, fast, and reliable. While it is clear that this project is a great improvement upon the last project, there is definitely still more work to be done to make it even better in the years to come.

Introduction

The original CP for Electrical and Computer Engineering Practice and Design (ECE 3400) was a plethysmograph, or a blood pulse reader. Using an IR LED and sensor, along with multiple active filters, a pulse could be detected from the pointer finger, much like the devices used at hospitals. The problem with this project was that it was circuit-intensive – almost all of the design had to do with analog circuitry, and it also wasn't reliable. More time was spent debugging than designing, and too often the solution was a loose connection somewhere in the too-cluttered breadboard.

Over the summer, meetings were held to decide on a new CP. An early idea was to use something like quadcopters that would avoid obstacles and deliver objects to a target. Quadcopters would make an impressive demo, but they inherently rely mostly on signal processing and coding; to add an analog component would be difficult. Between that concern and safety concerns, the idea was dropped. A second idea was to build a kind of "self-driving car" - a much simpler version of what Google has, and on a much smaller scale. A small town could be built and the car would have to obey traffic signs. This idea was generally well-received and was fleshed out a bit more, but the idea was eventually dropped because it was unclear if the difficulty level could be lowered to an appropriate level for juniors.

The Robotic Maze-Mapping idea was the project that was eventually settled on. Robotics projects generally are a little more signal processing and digital logic/coding heavy, with less analog circuitry, but since robots are so extendable, it's easy to come up with ways to incorporate analog circuits. The idea is to set a robot loose into a wooden maze for them to map by using various sensors and wireless communication. To help with location, a grid of electrical tape was put down on the ground so that robots could follow lines (if they choose) or use the crossings to track their location. The final piece of the project is for a computer monitor to show a visualization of the map on the screen, updated in real time.

The goal of this design project is not only to create and flesh out a CP for ECE 3400, but to also create a fully-functioning, one-robot prototype to be used as a demo. In fleshing out the CP, work must be done to create a starting point of material for students to use as a baseline to create their robot from. To analyze the success of this design project, students will be monitored and their methods collected to determine how appropriate the level of difficulty and creativity is for junior students.

Defining the Project

Over the summer, many meetings were held with the sole purpose of deciding on a project. This section will sample some that were proposed in addition to the robotic maze-mapping proposal. It is important to note that almost all proposed projects were viable solutions to the constraints considered, though it is argued that the robotic maze-mapping proposal was the best solution proposed.

The main constraints for the project were twofold. Firstly, it must use concepts learned in the three required ECE sophomore-level courses of analog circuits (2100), signals, (2200), and digital logic design (2300). Secondly, it must provide a demo that would be interesting to engineering students in general, in the hope of publicizing ECE and increasing the number of affiliated students.

Secondary desired (but not required) constraints were based on feedback given by the students on the original ECE Practice & Design (3400) course project. These constraints included an appropriate level of challenge, a polished end product, and more relatability between lectures and lab. The goal of lectures in 3400 is to teach design methods and engineering ethics that can be applied immediately to work in lab, and to industry and academia post-graduation.

One of the first proposed projects involved quadcopters. Drones have become more of a topic of controversy in everyday life, so having a project involving quadcopters would be sure to gain publicity and be a solid demo. The quadcopter could perhaps navigate to a destination or target, avoiding multiple obstacles, or perhaps deliver a package to a person or destination. Signal processing is a large component of this proposal because it involves heavy computer vision, so the question was then how to incorporate digital logic and analog circuitry. While various ideas were tossed around, the idea was eventually rejected due to safety concerns and lack of ability to test and fly objects in the given lab space.

A smaller version of Google's self-driving car was considered next, and was considered in some detail. A small robotic car would be required to follow traffic laws in a mock city. Cardboard pedestrians could cross streets, yield and stop signs would be created, and perhaps even traffic lights. The demise of this project was its complexity; creating a testing environment for lab sessions would be very difficult, and creating mock cities would require a lot of time and effort (especially if moving parts are incorporated). The complexity also extended to the expected workload for the students; it would need to be simpler in order to be completed in one semester, but the project idea didn't lend itself well to simplicity without becoming boring. This idea, however, is what fed into the idea of a robotic maze-mapping project.

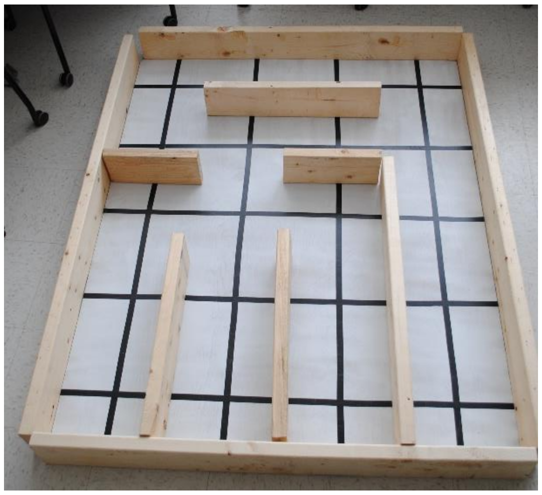
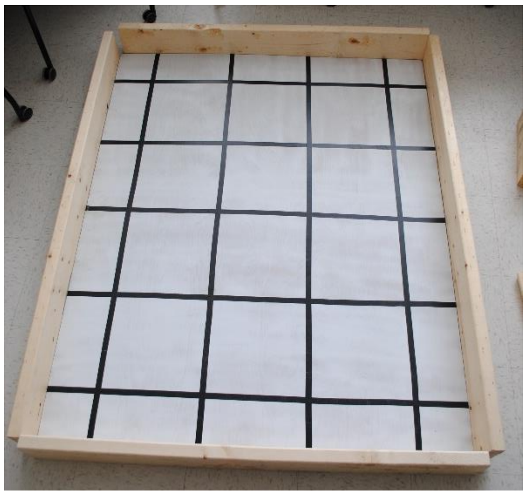
The maze-mapping project was proposed as a collaborative project – one that would require inter-team communication. Multiple robots would be set loose in a maze and communicate with each other to collaboratively map it in the smallest amount of time possible. Because the setup would be fairly simple (only walls need to be sensed to map the maze), signal processing would be a solid component like the other project proposals, but wouldn't overbalance the other aspects of the project. To add in a digital logic component, a field-programmable gate array (FPGA) would be used to display the maze on a computer monitor as it was being mapped in real time. Figuring out the analog circuitry component was more difficult, but it was eventually settled that a microphone circuit would be used to detect a start whistle, signaling that the robot could start mapping the maze.

The idea of robotic maze-mapping for the course project was a success because of many aspects. First and foremost, it contained balanced aspects of all three sophomore-level ECE courses. Secondly, it is an intriguing demo that would be relatively simple to set up and easy to impress. Thirdly, it easily broke down into submodules that would allow teams to divide and conquer and therefore be forced to employ good communication methods to get work done on time. Finally, it was a project that had real-world applications (reconnaissance and exploration, for one).

Summer Project Demo Robot

Once the project was decided on, the next big action item was to create a demo of the project in time for the incoming engineering freshmen to see during Orientation Week. In order to create a working prototype, the project needed to be more well-defined. The following definitions were made:

- The maze would be 5’ by 4’, and a grid of black electrical tape would be all across the floor of the maze to assist with robot navigation and location memory. The robot could choose to follow the lines and use the intersections of the lines to update location if it so desired. A photo of an empty maze is below on the right; a photo of an example maze is below on the left. Wooden blocks were used as the walls so they could be easily rearranged.

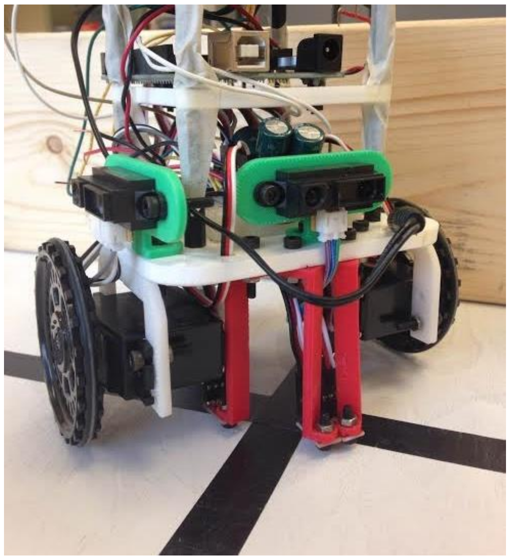



- Walls of the maze would be placed in one-foot increments, much like a maze drawn in a children’s activity book.

- A custom chassis would be designed. This way, no students would be more familiar with the platform than other students, and this project would stand out from other robotic competitions that used pre-packaged “kits.”
- The Arduino Uno would be used as the central microcontroller. The Uno was chosen because it is common, user-friendly, and simple; the focus of this project is not on coding but on combining multiple components with a microcontroller.
- The robot will start in the bottom-right corner of the maze, facing forward.

The summer demo was created by all four students (including the author) that were involved over the summer; they are acknowledged at the end of this paper. The robot was fully functional aside from the lack of microphone circuit (implemented separately but never integrated); the base station was fully functional as well. However, the radio communication between them was not functional until several weeks after the Orientation Week demo.

The Robot (DemoBot)



The robot had two platforms used on top of the custom chassis. The bottom layer housed a breadboard that served to connect all sensors and components to the appropriate power supply, ground the two separate power supplies together, and serve as wire management for connecting sensors and the radio to the Arduino. See the section “Common and Surprising Project Solutions” for descriptions on how students chose to design their robots and base stations.

Wheels

DemoBot used two Parallax Continuous Rotation Servos as wheels. Controlled by PWM, they were very intuitive to use as the Arduino has a simple Servo library and using servos eliminated the need for an H-Bridge to control motors.

Line-Following

Four line sensors (grayscale sensors) were used to detect lines and intersections. The two line sensors in front (seen with red mounts) were dedicated to line-following. Two sensors, side-by-side, are needed at minimum for line-following so that it can be determined which direction a robot is veering when it goes off the line. Two more sensors are used, spaced further apart, to detect line intersections. When all four sensors are on top of lines, DemoBot is at an intersection. At each intersection, DemoBot updates its location and polls its sensors to see where walls are.

Wall detection

Two long-range IR sensors, one on the front of DemoBot and one on the right, were used to detect walls. In the image above, they are shown with green sensor mounts on top of the chassis' base. The long-range sensors can measure distance up to several feet, but the measurement was thresholded to only sense if a wall was less than 6 inches from DemoBot.

Navigation

A simple right-wall-following algorithm was used to navigate the maze. Wall-following algorithms only work in a closed maze, where there are no open spaces (similar to children's mazes in an activity book). At each intersection, DemoBot looked to see where walls were. If there was no wall to the right, it would turn right and continue. If there was a wall to the right but none in front, it would go straight. If there were walls to the right and front, it would turn left and repeat the process (since it cannot see if there is a wall to its left). The students were expected to create a more general navigation algorithm that would successfully navigate all types of mazes.

Radio communication

DemoBot communicated with the base station by sending its entire array of maze information at once via a wireless radio. The base station would take that information, parse it out, and display it on a monitor. Unfortunately, the radio communication did not work by the time the demo was shown. It was functional a few weeks later.

The Base Station

The base station was not able to receive messages from the DemoBot at the time of the Orientation Week demo. However, as seen in the above figure, it was able to show a flashing checkerboard demo that demonstrated its functionality.

Radio communication

The base station's Arduino had another wireless radio that received messages from the DemoBot and parsed the information into messages that could be sent via wired communication to the FPGA. Since the radio received information regarding the entire maze, the information had to be translated into packages that gave one block's information at a time.

Wired communication

Eight wires were used for parallel communication between the Arduino and FPGA. Each message contained information about a particular location and its walls. Parallel communication was chosen because of its ease; less debugging needed to be done because parallel communication has no real "protocol."

VGA display

The FPGA converted the maze information into pixel coordinates and colors, then outputted them using VGA protocol. An 8-bit DAC was used to achieve 8-bit color. To prevent using all of the available memory, "sprites" which held specific predetermined patterns were used. Each block of pixels mapped to a sprite which held the pattern.

Elaboration of Project Guidelines and Rules

Since the project was brand-new to the students and instructors, firm rules and a solid description needed to be provided. Aside from the setup mentioned previously, the requirements below were given to the students, with an aim of providing a structured, ruled environment within which they may make design choices that allow them to at least partially customize their setup.

Robot Design

Your team's robot uses an Arduino Uno and a custom chassis. Over four structured labs and many free weeks, your team will construct a robot that can successfully incorporate multiple sensors to accurately and quickly map any given maze on the 5x4 grid. While it is desired for your robot to be as innovative as you like, the baseline requirements for the robot are as follows:

- It must use the Arduino Uno, custom chassis, and Parallax Continuous Rotation Servos.
- It must have two analog circuits, one of which is the microphone circuit from Lab 2.
- It must use the Nordic RF24 radio to communicate with the video controller system.
- The radio communication must comply with FCC and US Government laws.

Your team has a budget of $70 to use on sensors, servos, and other parts (note: this does NOT include the two continuous rotation servos you will use as wheels). The cost breakdown is as follows:

- Line (grayscale) sensors: $5
- IR sensors: $15
- Parallax servos (not continuous rotation): $15
- Basic components (resistors, capacitors, LEDs, etc.): $0

If there are other sensors or components you'd like to use, speak to the staff as early as you can. Ultrasonic sensors will not be available for use this year. The staff will assign a cost to the component and order it, if approved. It will take several days to get your parts in if they are in stock, and an unknown time if they are not in stock.

Video Controller Design

The second part of the final project is the video controller. Using another Arduino Uno, a wireless radio, and an FPGA, the video controller will show the maze on a computer monitor as it is being mapped by the robot (updating in real time). The baseline requirements for the video controller system are as follows:

- It must use the Arduino Uno and DE0-Nano FPGA.
- It must use the Nordic RF24 radio to communicate with the video controller system.
- The radio communication must comply with FCC and US Government laws.
- While the visuals are up to the discretion of the team, there must be a discernable visual difference between what is a wall, what is open space, what is unexplored space, and what is unexplorable space (closed-off areas).

Any additional parts that you wish to add to your video controller are subject to parts availability, safety concerns, and staff approval.

Elaboration of Lab Work

During the lab weeks before Spring Break, the students worked on four structured labs. The first served to get them acquainted with the Arduino. The second was the creation of the microphone circuit that resulted in a detection of a 660 Hz tone (used for the competition). The third was coding the FPGA to display maze information on a monitor. The fourth and final lab revolved around the radio communication between the robot and base station. The lab handouts and other course handouts related to lab and the course project were written by me.

A description of the labs, along with their required steps, is given below.

Lab 1: Using the Arduino Uno

Objective

In this introductory lab, you will learn how to use the various functionalities of the Arduino Uno and the Arduino IDE, as well as the GitHub repository system. Your team will brainstorm and construct a simple functional Arduino program using multiple external components and the Arduino Uno. Once you have this basic understanding, you'll be able to start building your robot!

Procedure

1) Connect the Arduino Uno to the computer with a USB cable. Open up the Arduino IDE and program the "Blink" example sketch onto the Arduino. The Arduino's LED should begin blinking. Review the code in the sketch and make sure you understand it.
2) Modify the "Blink" sketch to work for an external LED.
3) Connect a potentiometer to the Arduino and use the Serial connection to the computer top print out the analog readings of the potentiometer.
4) Use the pulse-width modulation (PWM) commands to manage the brightness of an external LED. To determine the PWM value, map a potentiometer's value to the PWM, so that when you turn the potentiometer, the brightness of the LED changes.
5) Using various buttons, LEDs, photoresistors, resistors, and/or pushbuttons, create a simple circuit that uses the Arduino to operate.
6) Connect one of the Parallax Continuous Rotation Servos to the Arduino and control it using the Arduino Servo library. Set the wheel to "stop," and then turn the physical knob on the servo until it stops moving so it is calibrated.

Lab 2: Analog Circuity and FFTs

Objective

In this lab, you'll be creating two analog circuits to attach to your Arduino. One is a microphone circuit that will detect a whistle blow signifying the beginning of your mapping race; the other is a sensing circuit of your choice.

Procedure

1) Download the Open Music Labs FFT library.
2) Use the example sketch that reads in a value from an analog pin as a basis for your FFT code. The whistle blow will be 660 Hz.
3) Create the microphone circuit so that it amplifies the signal received from the electret microphone and filters out noise.
4) Connect the microphone circuit to the Arduino and use a method of your choice to determine if a 660 Hz tone is present.
5) Create an additional analog circuit of your choice that you think may be useful for your robot.

Lab 3: FPGA Video Controller

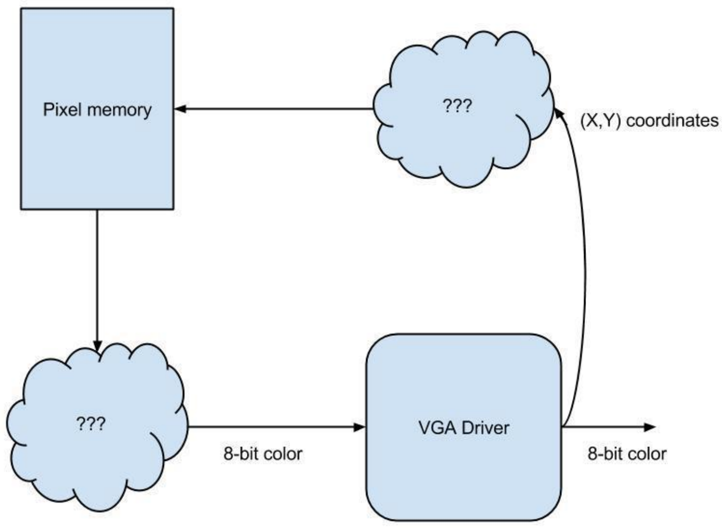
Objective

In this lab, you will create a video controller with an FPGA, controlled by an Arduino Uno. To do this, you will need to familiarize yourself with the DE0-Nano FPGA development board, develop a system for transferring image information from the Arduino to the FPGA, and learn how to interact with the video memory and VGA driver. From a hardware perspective, you will also need to construct a basic Digital-to-Analog Converter (DAC).

By the end of the lab, you should be able to draw a representation of the maze on the monitor such that this representation is on the Arduino, is then sent to the FGPA, and finally to the monitor.

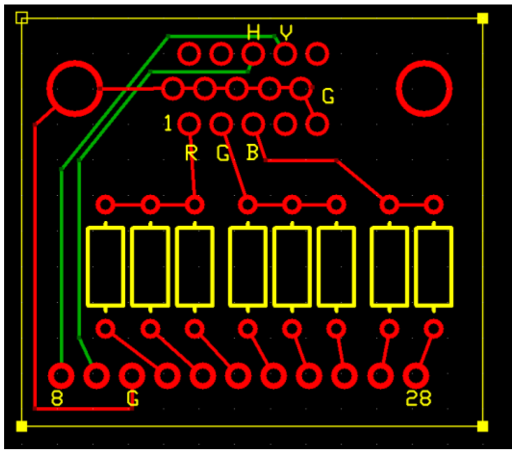
Procedure

1) Design and code a memory system for pixels. There is not enough memory on the FPGA to house 8-bit color information for every pixel, so you must determine how to store all necessary color information on the FPGA. The VGA driver, given, tells what pixel's color it needs to display (x and y coordinates) and uses the 8-bit color given to it to output on the next cycle. A diagram is shown below to visualize what is given to you and what you need to code yourself.



2) Create a communication method between the Arduino and FPGA. The Arduino will be receiving maze information from the robot; you must decide how to parse this information and send it to the FPGA. In addition, know that the Arduino outputs 5 Volts while the FPGA accepts 3.3, so a voltage divider needs to be constructed.

3) Solder a DAC (PCB shown below) to convert the FPGA's 8-bit output to 3 RGB channels. Use the fact that the monitors have a 50 Ohm resistance to scale your resistor values.



PCB design by Dr. Bruce Land

Lab 4: Radio Communication

Objective

In this lab, you'll be working on the final major component of your project – radio communication. Using the Nordic nRF24L01+ transceivers and the corresponding Arduino RF24 library, you'll get the robot and video controller to talk to each other. At the end of this lab, you should ideally be able to send messages from one Arduino to the receiving Arduino (simulating actual maze information) and have the FPGA show it on the monitor.

Procedure

1) Solder the headers to two of the RF24-to-Arduino PCB adapters.
2) Use the Arduino RF24 library's "Getting Started" sketch to send timestamps back and forth between the two radios. Use identifier numbers that are unique to your team.
3) Perform two different methods of sending maze information – the entire maze all at once as a char array, and only the relevant maze information (packed into a few bytes).
4) Decide how you want to store and send maze information, and implement that method.

Results and Observations of Lab Work

Lab 1: Using the Arduino Uno

Most students completed this lab early as it was fairly straightforward and simple. Some of the simple circuits created included an adder with LED outputs, a dimmable LED with an on/off switch, and others. Overall, the lab was a success as it acquainted the students with the Arduino programming language, basic functions, and layout.

Lab 2: Analog Circuitry and FFTs

As many analog circuitry labs go, the students had many small problems creating their circuits. A constraint that stooped many students was the necessary DC offset to translate the AC signal from the microphone to a 0-5V signal that could be interpreted by the Arduino.

The students are generally well-versed in analog filters, making the tuning and implementation of the filters relatively easy. They generally spent more time working on the FFT software end of the lab rather than the physical circuit construction.

Many groups began by thresholding the bin for 660 Hz. However, they quickly found that employing a threshold wasn't sufficient because of the variability in 660 Hz noise and the lack of robustness for if/when the physical circuit is modified. After this realization, teams moved to more complex statistical methods to analyze the noise and determine if the 660 Hz tone is indeed present.

The second half of the lab required creating another analog circuit of the teams' choice; many chose to create a Schmitt trigger to threshold the provided line sensors. This was so that students could use several line sensors without using up all the available analog pins on the Arduino. However, later on many teams used analog muxes to maximize the number of analog inputs they could use, making their extra circuit unnecessary.

Lab 3: FPGA Video Controller

Arguably the most difficult lab of the four, this lab was allotted three weeks for completion and many teams still did not complete it in time. While some teams decided to take on more work by creating more proportional or pretty graphics, almost all teams still struggled with understanding Verilog and implementing designs.

Proper brainstorming and design techniques were often not employed by the team, which most likely contributed to the challenges they faced. Block diagrams were largely not created, which would have made Verilog coding much easier.

Most of the teams chose to implement SPI as the communication protocol between the Arduino and FPGA. SPI is generally not taught in Digital Logic Design (2300), so the students had to teach themselves the protocol without ever having used it before. A few teams chose to use parallel communication, which entails less debugging but can be more troublesome due to extra wires and lack of a clock.

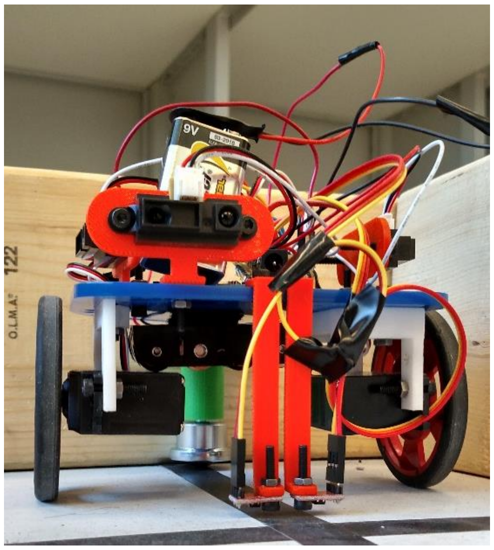
The TAs all met with teams before Lab 3 began, making sure they had an idea of what they were planning on doing and answering any questions. The general feedback was that the students were not well-prepared for this difficult lab.

Lab 4: Radio Communication

This lab was significantly easier than Lab 3; many teams completed the lab in less than the two weeks allotted. Once the teams had their radios working with the Getting Started sketch, they brainstormed how they wanted to format their maze information into packets.

Some teams were able to hook up the receiving Arduino to their FPGA and have it display information that was sent via radio, but most teams weren't able to get this integration working until the open lab sessions after Spring Break.

Spring Demo Robot



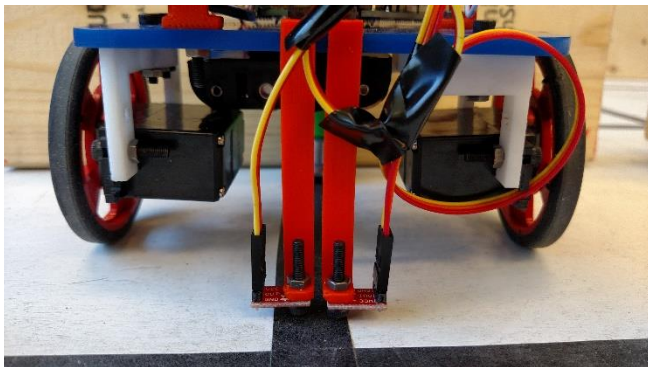
Another demo bot (Dagmar) was made by me during the Spring semester alongside the students, using the same constraints the students were given. A few changes were made compared to the previous DemoBot; these differences are noted below. In addition, it did not exhibit 100% functionality. Namely, it did not have the following attributes required for the students:

- It did not have a microphone circuit incorporated. A microphone circuit was built and tested in isolation; however, it was not integrated into Dagmar's final design.
- Dagmar did not have the knowledge to determine what areas were unexplored (not yet visited) versus unexplorable (blocked off). Thus, she would not be able to accurately represent the maze.
- A simple right-wall-following navigation algorithm was used. While Dagmar does stop exploring when the maze has been completely explored, she doesn't have a mechanism that allows her to deviate from the simple algorithm. This means that Dagmar cannot explore all types of mazes.

The Robot (Dagmar)

Line-Following

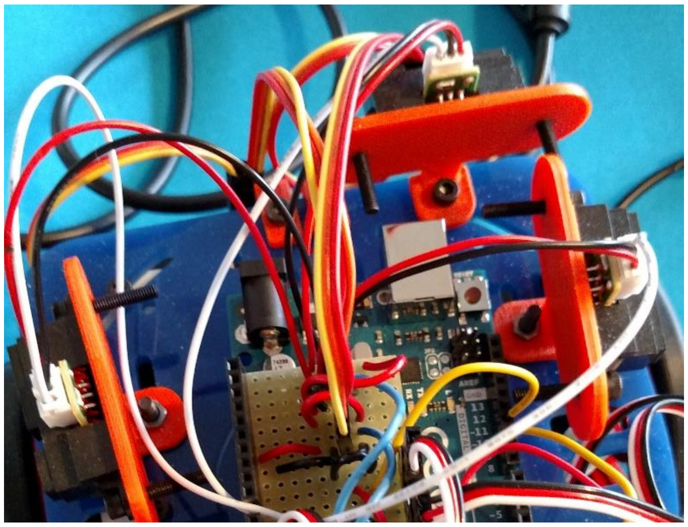
Dagmar uses only two line-following sensors instead of the four used on DemoBot. By having two line sensors that straddle the black line, it is able to both line-follow and catch intersections. When both line sensors say that they are on a line, an intersection has been detected. The Arduino PID library was used to link the line sensor readings to the servo speeds. Code can be found in the Appendix.



Wall detection

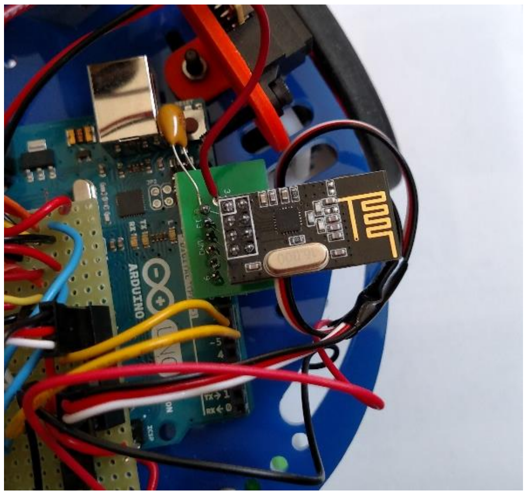
Three short-range IR sensors, one in front and one on the left and right sides, were used to speed up maze-mapping. Even though Dagmar has the same navigation algorithm as DemoBot, she is able to map some mazes with open spaces because she has three IR sensors.

The short-range IR sensors can only sense up to about 1 foot away, so Dagmar only records what is immediately around her.



Radio communication

Dagmar had fully functioning radio communication. For simplicity, the entire maze was sent every time Dagmar hit an intersection. The whole maze was characterized by a 9x11 char array, so the size was 99 bytes. The maximum payload for a single message is 32 bytes, so the maze was broken up into four messages for simplicity.



The Base Station

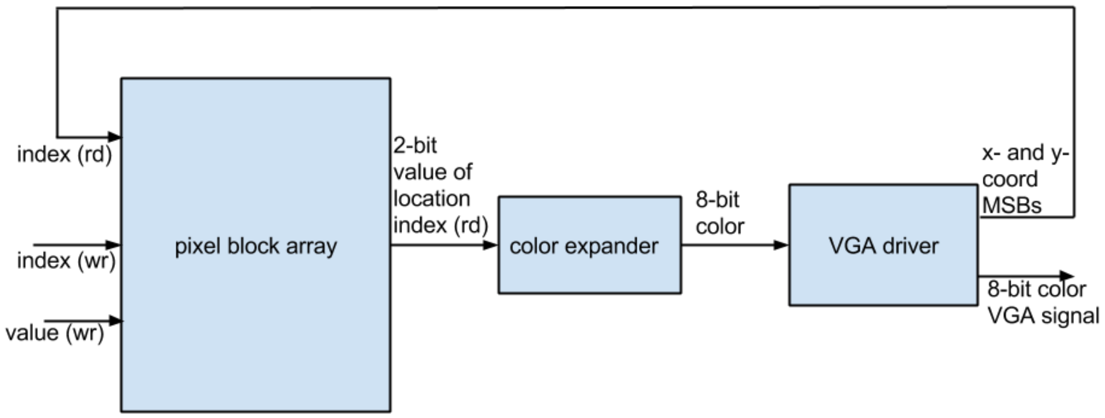
Wired communication

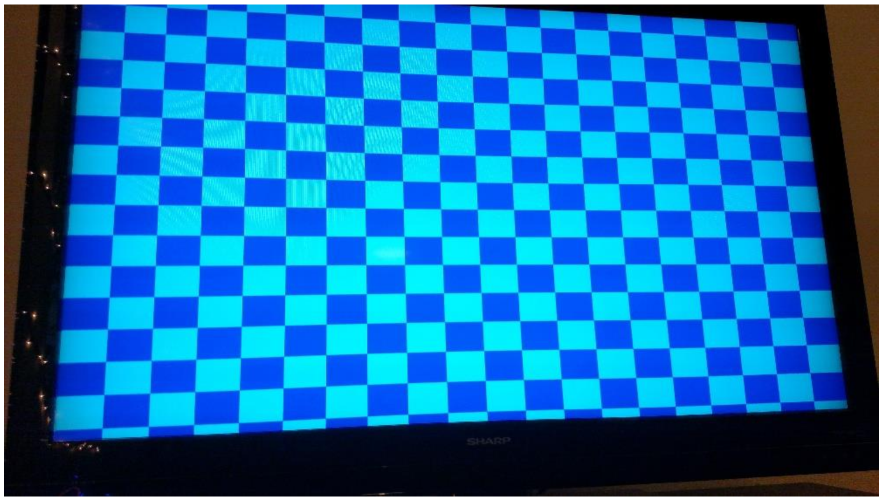
The Arduino and FPGA communicated via SPI; since the RF24 radio also uses SPI, a soft SPI was created in C++. Each packet has 10 bits – 4 bits for an x coordinate, 4 bits for a y coordinate, and 2 bits for type of block (which translates directly to a color). Code can be found in the Appendix.

Initially, a parallel communication method was implemented, but it was scratched because it was unnecessarily buggy; it is hypothesized that the output ports did not change in time, which would create intermediary results that would be picked up by the FPGA. To combat this, another wire could be added that goes high when a new message is sent, but adding this would be nearly as complex as SPI, so the change was made.

VGA display

The FPGA saves each block’s color in a 2D array. The VGA driver’s requested x and y coordinates are chopped so that only the MSBs of the coordinates index into the array. This creates the effect of having squares mapped to entries of the 2D array. Each 2-bit entry in the 2D array was then expanded to a full 8-bit, predetermined color. A block diagram of the setup is shown immediately below; a checkerboard demo is shown below the block diagram.





Common and Surprising Project Solutions

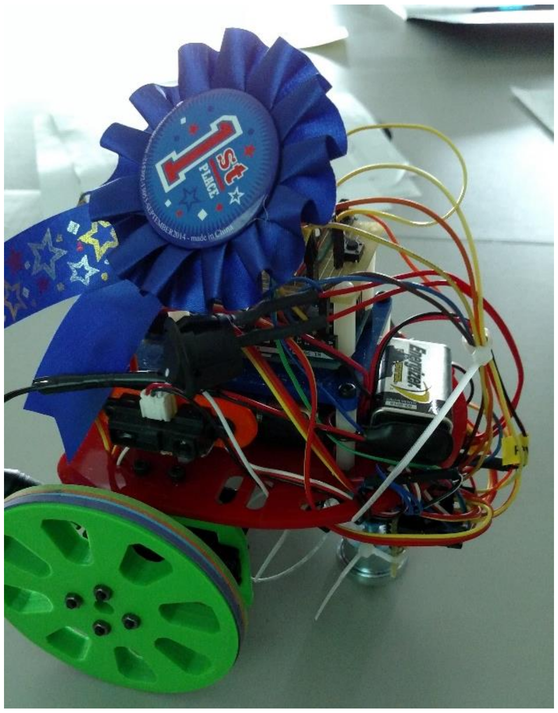
While the students were given strict rules to follow, no two robots were the same. In this section, common solutions will be given, as well as solutions that were unique and unexpected.

The Robot

<u>Wheels</u>

Most students were happy with using the given Parallax Continuous Rotation servos. One team asked if they could purchase omni-wheels, which could go in any direction. This would save time because the robot would not be required to turn. However, the wheels in question were outside of the given $70 budget.

Another team 3D-printed larger wheels so that their robot went faster. The end result worked (they came in 1st place at the competition), but their line-following was very jerky because any small correction in speed was magnified by the larger wheels.



Gears were implemented in one team's robot; however, the added speed wasn't enough to make it worth the extra hassle. Ultimately, they were dismantled and the original wheels were used.

Line-following

Quite a few teams used two line-sensors that straddled the line (similar to Dagmar). Several teams used three sensors in a line, and yet others used four in a similar fashion to DemoBot. There didn't seem to be a difference in performance for the number of line sensors used.

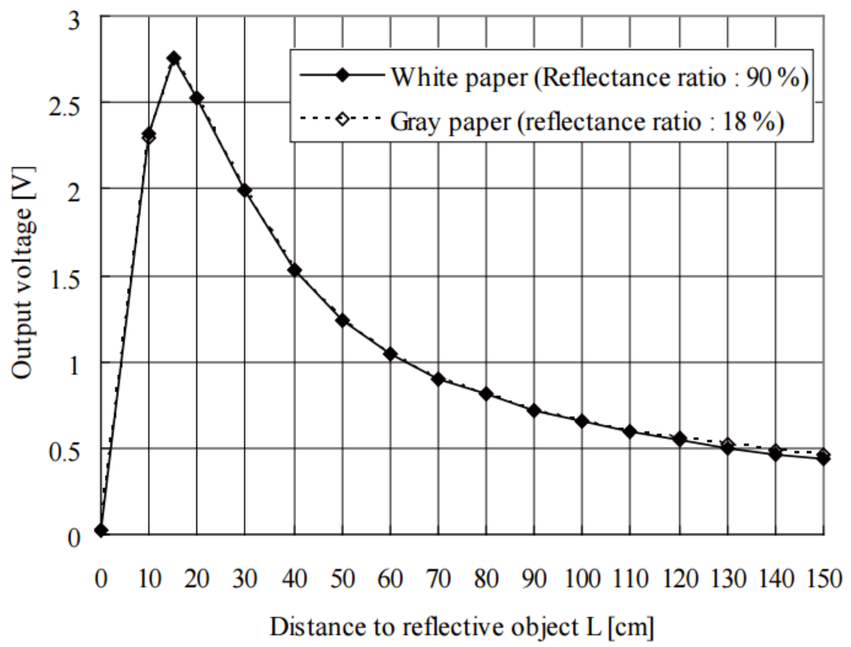
For a line-following algorithm, every team used PID or some kind of version of it. Arduino has a PID library (which worked fine with Dagmar), but a few teams decided to implement it themselves or derive an ad-hoc version of PID.

Wall detection

Most teams had three IR sensors. Three IR sensors make it easier to map a general maze, but only one is needed minimum. One team considered putting an IR sensor on top of a servo so it could look around without moving the actual robot, but since having three IR sensors does about the same thing and is still within the budget, teams opted for this arrangement instead.

Originally, only short-range IR sensors were provided, but later in the semester some long-range IR sensors were made available as well. It is estimated that a little over half of the teams chose to stick with only short-range IR sensors, while a little less than half used a mix of the two.

Naively, once the students found out that long-range IR sensors were in stock, they flocked to use them, thinking that there could be no possible downside since they simply could "see" further and therefore give them more information to use in mapping. However, the reason that many teams ultimately decided not to use long-range sensors was because of its output curve, seen below:



from Sharp GP2Y0A02YK0F datasheet

As you can see, the range of 10-25cm (4-10 inches) is aliased – the output voltage for values in that range could imply two different distances. In addition, the IR sensor is marketed as a distance sensor for 20-150cm. When a robot is at an intersection, if there is a wall in front of it, it is about 3 inches away. The long-range IR sensor cannot be counted on to say that it is 3 inches away instead of 15, and it cannot even be counted on to give a reliable reading for 3 inches (data point is not even on the graph). Therefore, the long-range IR sensor works best in conjunction with a short-range. Since the IR sensors "cost" the students $15, the added distance sensing range wasn't worth the effort or cost for many of the students.

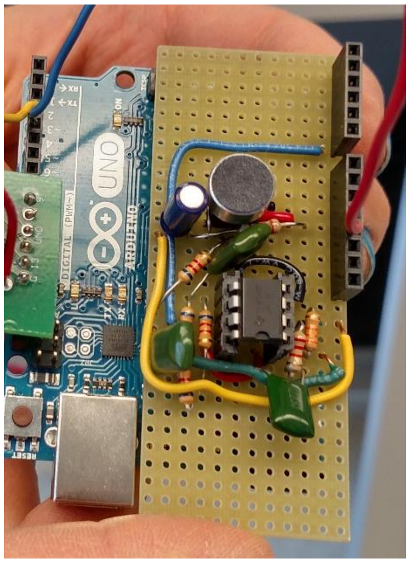
Microphone circuit

Because it was used in lab, all teams used the Open Music Labs Arduino FFT library. The library itself is flexible and reliable; however, it did use up much of the available memory, which put some teams in trouble closer to the competition. Those that were affected mainly chose to remove the microphone circuit entirely and incur the 2-point penalty for each round of the competition.

In detecting the tone, teams employed a variety of different methods; some simply thresholded the bin that contained 660 Hz. Others measured the noise and made sure the bin for 660 was a certain amount higher than the average noise level. Still others used the standard deviation of the bins to determine if 660 Hz was present.

For the actual circuit, most teams used a Chebyshev filter to both filter and amplify the microphone's signal. The Chebyshev filter is a bandpass filter. A few teams chose to use a high-pass filter, expecting there to be little noise above 660 Hz, which is close to a high E, not commonly used in speaking. All teams used op-amps at minimum to amplify the signal and perform some kind of filtering.

Some students soldered their microphone circuit together instead of leaving it on the solderless breadboard; a soldered solution is shown below.



Navigation

When the teams began putting the different components of their robot together, they often employed a "turn right if you can" or right-/left-wall-following algorithm. This is because they are easy to monitor on the fly and notice when something goes wrong.

For the actual competition, most teams used a depth-first search or breadth-first search algorithm. Several teams added heuristics to their search algorithms to avoid repeating previously-searched areas or to avoid returning to the starting point before saying that the maze had been completely discovered.

One team used the A* search algorithm, which is known for finding the shortest path to a destination. A* could be employed by having an initial destination and changing it, or perhaps having an out-of-reach destination that would force the robot to search all possible paths.

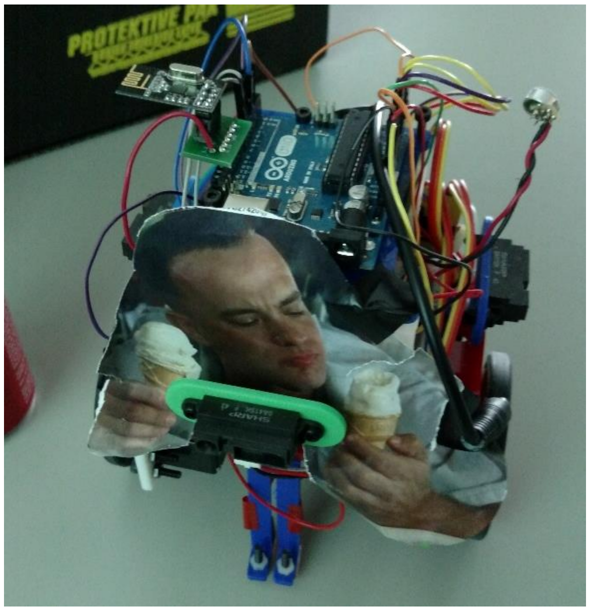
Radio Communication

The decision between sending an entire maze representation versus sending packets only with new maze information was largely one-sided because of the 32-byte maximum payload of the radio. For many robots, sending four maximum-sized packets representing the entire maze did not slow the robot down or cause complications. For some, sending multiple rapid-fire messages made the robot's operation less smooth because their navigation algorithm was also computation-heavy, asking too much of the Arduino in too little time.

While some groups were then forced to create smarter, smaller packages, other groups chose to do so from the beginning. Putting the parsing logic on the robot made it easier to translate to the VGA – if a packet was only 1 byte, the receiving Arduino could just feed the message through to the FPGA without much thought.

Aesthetics

5% of the final project grade is overall aesthetics. This encouraged the teams to clean up their robot and have good wire management. Several teams went above and beyond by pasting photos of celebrities on their robot or decorating it with stickers.



The Base Station

Radio communication

As mentioned above, teams had to decide how to send the maze information. If the teams sent the entire maze, the base station Arduino had to parse the information so it could be sent to the FPGA.

Wired communication

Most teams chose to use SPI as the protocol for communication between the Arduino and FPGA. No SPI Verilog code was given, so the students coded their own implementation. Because the RF24 radios also use SPI, the teams had to decide how to manage using two devices; they either used the soft-coded SPI like in Dagmar’s implementation or managed the slave-select.

VGA display

Dagmar’s setup was given as an example for the students; since it was mostly complete, many of the students chose to use that as their method of display. The result was a blocky display of the maze that was crude but correct.

Several groups went above and beyond by either spicing up the graphics or making the visuals more proportional to the actual maze. With the provided setup, every block is the same size, so walls are just as large as open spaces. Several teams made the spaces between physical blocks smaller so that walls could look more like borders and therefore more proportional.

The most impressive graphics display still had a block-like proportionality, but the blocks themselves were interesting patterns like brick and grass, and the background of the maze flashed different colors when the maze was mapped completely.

Competition Setup

The competition has four rounds. Every team participates in the first two, and then the eight teams with the highest cumulative points proceed to the third round. The highest four performers then proceed to the final, fourth round. Each round had a different maze layout.

The scoring is determined by the following criteria:

Speed

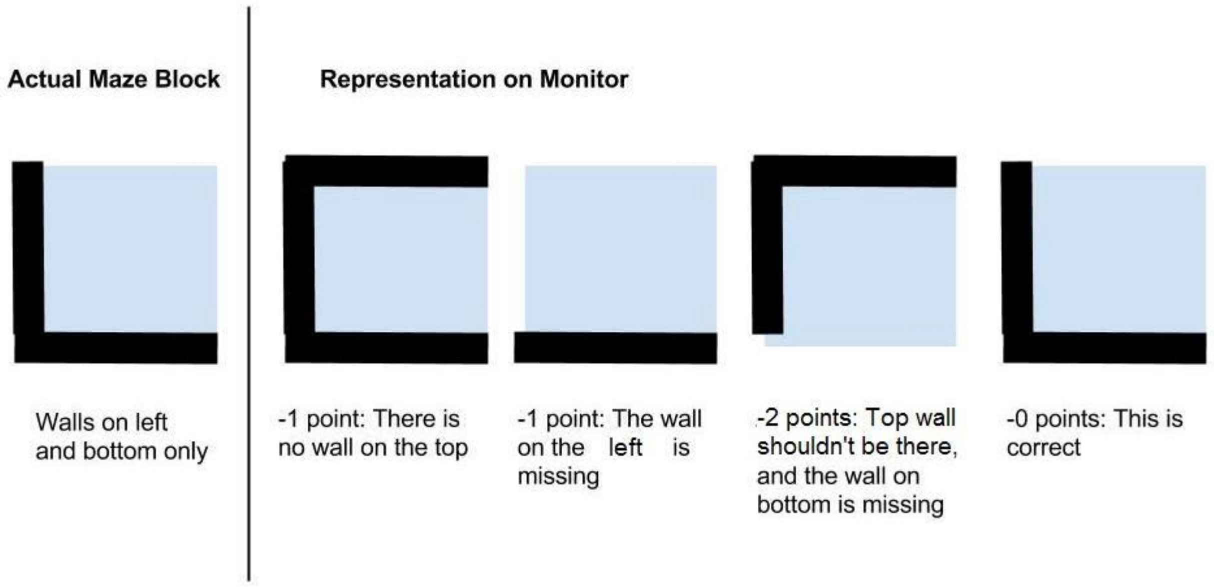
The team that completes exploring the maze in the least amount of time, in a given round, gets 10 points. The team that completes exploring the maze the slowest, in a given round, gets 0 points.

All other teams between the fastest and slowest times get allocated points on a linear scale. For example, if the slowest team completes the maze in 1:10 (one minute and ten seconds) and the fastest team completes the maze in 0:10 (ten seconds), a team that completes the maze in 0:30 would receive 8 points and a team that completes the maze in 1:00 would receive 1 point.

To signify that a robot is done exploring the maze, it must come to a complete stop. If a robot stops before having explored the entire maze, 0 points will be given for speed – this will be obvious by the incomplete map of the maze on the computer monitor. If a robot is stuck in a loop and will never stop even after fully exploring a maze, 0 points will be given for speed.

Accuracy

Each team will start with 10 points allotted for accuracy. For each piece of information that is displayed incorrectly, 1 point will be deducted for a minimum of 0 points. Below is a visual example for how one 1’x1’ block would be graded:


Actual Maze Block
Representation on Monitor
Walls on left and bottom only
-1 point: There is no wall on the top
-1 point: The wall on the left is missing
-2 points: Top wall shouldn't be there, and the wall on bottom is missing
-0 points: This is correct

Errors will not propagate, so if the left wall was missing in this example, another point would not be deducted (for a total of 2 points off) because the right wall would then be missing in the block to the left of the example block.

If the difference between unexplored and unexplorable space is not distinguished, 3 points will be deducted from the score.

Miscellaneous Point Alterations

For each time a wall is knocked over by the robot, 1 point is deducted, and then the wall is promptly replaced by the staff. If this situation requires a human to move the robot, another point is deducted (addressed in the paragraph below). If the microphone override button is pressed to start the robot, 2 points are deducted. No timing penalty is incurred.

If a situation arises that requires human intervention, 1 point is deducted per instance. This can be added to a wall-falling situation for a total of 2 deducted points. This would occur, for example, when a robot fails to notice a wall in front of it, so it runs into the wall, knocking it over, and then gets stuck because it cannot drive over the fallen wall. This would result in 2 points deducted because the robot would need to be manually moved to resume navigation.

If the video base station needs to be restarted or requires human intervention during the race, 1 point is deducted and any information lost cannot be regained. If a team decides that their robot is behaving erratically, they may restart their robot for a penalty of 3 points. This includes an optional reset of the video base station. Only one restart your robot once per round is permitted.

Results and Observations of Competition

Overall, students were ready for the competition. On the morning of, many were in lab at 7 a.m. to put the finishing touches on their robot and make sure that everything was programmed with the desired code.

At the competition, the students had their robots and FPGAs ready to go so that when they were up for competing, they simply plugged in their FPGA to the provided monitor and power supply and then turned on their robot.

A few hiccups were present, as to be expected; one team’s robot broke after Round 2 due to a mechanical failure. Another team’s robot went slower than expected because the batteries were low. About four teams had robots that were not fully functional. Even so, the competition was a success and the last two rounds were heated, with several robots competing neck-and-neck.

A large crowd was generated with both passersby and fellow students. Duffield Atrium certainly is the venue of choice for this competition. In the future, a larger display will be used for the VGA maze display, and possibly for a live stream of the robots in action as well.

Future Work

Overall, the new project was a success and the students enjoyed working on a project that was different, diverse, and with real-world applications. However, there are many improvements that can be made.

Labs should be completely rethought. While at the end of the labs, the students had distinct parts of the project complete, the integration took a lot of effort and could have been made part of the labs fairly easily. A proposed revised lab structure is below.

- Lab 1 (1 week) – Intro to the Arduino and basic robot assembly
- Lab 2 (2 weeks) – Microphone circuit (remove the second analog circuit), with the goal of having the robot turn in a circle when it hears 660 Hz. Circuit should be soldered for reliability. Instead of using the Open Music Labs FFT library (which is pretty large and takes up a lot of the memory on the Arduino), a simpler C version could be used for the FFT calculation.
- Lab 3 (1 week, can be combined with another lab) – Line-following, turning, and wall sensing. Teams will use PID or other algorithms to line-follow and implement an algorithm for efficient turns. They will also incorporate IR sensors to make sure robots never run into a wall. At the end of this lab, the robot should only be missing the radio communication and navigation algorithms.
- Lab 4 (3 weeks) – VGA display. This lab will be largely unchanged from the original Lab 3, but the pre-lab will be required and the students will have more preparation. This will be described in more detail below.

- Lab 5 (1 week) – Radio communication. This lab will be unchanged from the original Lab 4, except the end result will be full integration with the VGA display setup from Lab 4.

By the end of the revised Lab 5, the students will simply need to create and implement a navigation algorithm for their robot and fine-tune the other components.

Another change that should be implemented are the Deliverables. Historically, the students have been largely unimpressed with lectures and generally wait until the last minute to complete a Deliverable. While it is not necessarily a goal to impress the students during lecture, it is a goal to make the Deliverables useful. The simplest way to do this is to weave them in with the lab.

This can be done, for example, by requiring that a specific design choice algorithm be used in determining what kind of filter to use for the microphone circuit, due before Lab 2 begins. Or a block diagram with several iterations could be required before Lab 4 begins. When determining how many sensors and what kind of sensors to use, the students could be required to analyze different combinations and use the design that they determine is the best.

Finally, more finite milestones must be required. Currently, lab reports are graded on quality of writing and not on functionality or design of the project. While this can still be true, having dedicated functionality milestones that contribute to a portion of the final project grade would discourage procrastination.

While having this amount of work required during lab sessions may seem a bit much, the students tend to have a larger workload toward the end of the semester. By forcing them to frontload, they will likely have a much less stressful semester in total.

Conclusion

All in all, implementing this new robotic maze-mapping project into ECE 3400 was a great success. The students were invested in creating the best robot they could, the department gained publicity (and hopefully more prospective affiliates), and the students learned a lot about project management. While there are many changes that can be implemented in the coming years, the amount of hard work put into getting this project rolling has clearly laid a solid foundation upon which things can only improve. The project has outperformed its plethysmograph predecessor by encouraging more teamwork, delegation, and smart design decisions. While at the least it is something good to put on a resume, it will inevitably create better engineers to enter the workforce and academia.

Acknowledgements

I would like to thank Professor David Albonesi first and foremost for asking me to become involved in this project last summer. I would also like to thank Jay Fetter, Alex Jaus, and Scott Zhao for brainstorming with me over the summer and creating a great demo for the students to look at during O-Week, as well as meeting with me over the course of the school year to wrack your brain for suggestions and technical help.

I am eternally grateful for Dr. Bruce Land, my advisor and the lab teacher for ECE 3400, who has been a great resource, constantly asking questions about my methods and proposing alternatives and great ideas while simultaneously making me feel like he had complete confidence in my abilities as a TA and computer engineer. Thank you to Professor Carl Poitras, the official teacher for ECE 3400, for giving me a ton of free reign when it came to the lab project, allowing me to design lab handouts, create grading rubrics, and make huge decisions that affected the students in lab.

Thank you to you, the reader, for allowing my work to extend to today instead of mid-May, 2015.

And a big thank you to all the ECE 3400 students from Spring 2015 for willingly being my guinea pigs. You all took the challenge head-on and came up with some truly remarkable robots. I thank you for your constant stream of wild ideas and questioning of my methods. Complacency is no fun!

Finally, thanks to all my friends and family, especially my parents who pushed me through my final years of college, my sister for dealing with me for 3 years of college instead of 2, and my boyfriend for letting me vent without ever complaining. I love you all!

Appendix

Robot Code (robot.ino)

```
#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>

#include <Servo.h>
#include <PID_v1.h>
#include <SPI.h>

boolean start = false;

boolean justStarted = true;
boolean hardCoreMode = false;

/* Sensor variables */
//PID
double inputL, outputL;
double setpoint = 350;
double leftWheel = 110;
double rightWheel = 70;
double inputR, outputR; //setpoint is same
PID lineFollowerL(&inputL, &outputL, &setpoint, 1, 0, 0,DIRECT);
PID lineFollowerR(&inputR, &outputR, &setpoint, 1, 0, 0,DIRECT);

//distance sensors
int frontIR = 0;
int leftIR = 1;
int rightIR = 2;

//line sensors
int leftLine = 3;
int rightLine = 4;

//Servos
int leftServoPin = 6;
Servo leftServo;
int rightServoPin = 5;
Servo rightServo;

//Radio
RF24 radio(9,10);
const uint64_t pipe = 0x000000001ELL;

//bookkeeping
int maze[100] = {2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 0, 0, 0, 0, 0, 0, 0, 2,
                2, 0, 0, 0, 0, 0, 0, 0, 2,
                2, 0, 0, 0, 0, 0, 0, 0, 2,
                2, 0, 0, 0, 0, 0, 0, 0, 2,
                2, 0, 0, 0, 0, 0, 0, 0, 2,
                2, 0, 0, 0, 0, 0, 0, 0, 2,
                2, 0, 0, 0, 0, 0, 0, 0, 2,
                2, 0, 0, 0, 0, 0, 0, 0, 2,
                2, 0, 0, 0, 0, 0, 0, 0, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2};   //extra is for even 100.
int location[2] = {7, 11};
int direction = 0;


void setup() {
  //radio
  radio.begin();
  radio.setRetries(15,15);
  radio.setAutoAck(true);
  radio.setChannel(0x50);
  radio.setPALevel(RF24_PA_HIGH);
  radio.setDataRate(RF24_250KBPS);
  radio.openWritingPipe(pipe);

  //servos and sensors
  pinMode(OUTPUT, leftServoPin);
  pinMode(OUTPUT, rightServoPin);
  leftServo.attach(leftServoPin);
  rightServo.attach(rightServoPin);
  lineFollowerL.SetMode(MANUAL);
  lineFollowerR.SetMode(MANUAL);
}

void loop() {
  while (!start) {
    //wait for 660 tone (currently always returns true)
    start = check660();
  }
  //turn on PID
  lineFollowerL.SetMode(AUTOMATIC);
  lineFollowerR.SetMode(AUTOMATIC);
  lineFollow();

  //hit an intersection
  if (analogRead(leftLine) > 850 && analogRead(rightLine) > 800) {
    //turn off PID
    lineFollowerL.SetMode(MANUAL);
    lineFollowerR.SetMode(MANUAL);
    updateLocation();
    updateMaze();
    sendMaze();
    decideWhatToDo();
    //turn on PID
    lineFollowerL.SetMode(AUTOMATIC);
    lineFollowerR.SetMode(AUTOMATIC);

  }

}

//checks if the robot is done exploring
//if not, decides where to go next
void decideWhatToDo() {

  int i = 0;
  boolean nothingToExplore = true;
  while (i < 99 && nothingToExplore) {
    if (maze[i] == 0)
      nothingToExplore = false;
    i++;
  }

  while (nothingToExplore) {
    leftServo.write(90);
    rightServo.write(90);
  }

  if (!getSensorRight()) //no wall to the right
    turnRight();
  else if (!getSensorFront()) //no wall in front
    forward();
  else if (!getSensorLeft()) //no wall to left
    turnLeft();
  else {
    turnAround();
    if (getSensorFront()) {//trapped
      leftServo.write(90);
      rightServo.write(90);
      while(1);
    }
  }
  return;
}

void forward() {
 leftServo.write(100);
 rightServo.write(80);
 delay(90);
}

void turnLeft() {
  forward();
  delay(150);
  leftServo.write(60);
  rightServo.write(80);
  delay(500);
  while (analogRead(leftLine) < 500);
  delay(50);
  direction = (direction == 0 ? 3 : (direction == 1 ? 2 : (direction == 2 ? 0
: 1)));
  return;
}

void turnRight() {
  forward();
  delay(150);
  leftServo.write(100);
  rightServo.write(100);
  delay(500);
  while (analogRead(rightLine) < 500);
  delay(50);
  direction = (direction == 0 ? 2 : (direction == 1 ? 3 : (direction == 2 ? 1
: 0)));
  return;
}

void turnAround() {
  forward();
  delay(150);
  leftServo.write(100);
  rightServo.write(100);
  delay(500);
  while (analogRead(rightLine) < 500);
  delay(500);
  while (analogRead(rightLine) < 500);
  delay(50);
  direction = (direction == 0 ? 1 : (direction == 1 ? 0 : (direction == 2 ? 3
: 2)));
  return;
}

int getSensorFront() {
 if (analogRead(frontIR) > 130) {
  return 1;
 }
 return 0;
}

int getSensorLeft() {
 if (analogRead(leftIR) > 200) {
  return 1;
 }
 return 0;
}

int getSensorRight() {
 if (analogRead(rightIR) > 200) {
  return 1;
 }
 return 0;
}

//replace with microphone code
boolean check660() {
  return true;
}

void lineFollow() {
  inputL = analogRead(leftLine);
  inputR = analogRead(rightLine);
  lineFollowerL.Compute();
  lineFollowerR.Compute();
  //use PID outputs to set wheel speeds.
  leftWheel = 130 + outputL - outputR;
  rightWheel = 60 - outputR + outputL;
  //set lower and upper limits
  if (leftWheel < 95) {
    leftWheel = 95;
  }
  else if (leftWheel > 180) {
    leftWheel = 179;
  }
  if (rightWheel > 85) {
    rightWheel = 85;
  }
  else if (rightWheel <= 0) {
    rightWheel = 1;
  }
  leftServo.write(leftWheel);
  rightServo.write(rightWheel);
  return;
}

//updates the current space to be open
//also updates the robot's location
void updateLocation() {
  int currentSpace = location[1]*9 + location[0];
  if (currentSpace < 99 && currentSpace > 0)
    maze[currentSpace] = 1;
  if (direction == 0)
    location[1] -= 2;
  else if (direction == 1)
    location[1] += 2;
  else if (direction == 2)
    location[0] += 2;
  else
    location[0] -= 2;
  currentSpace = location[1]*9 + location[0];
  maze[currentSpace] = 1; //open space

  if (currentSpace == 88 && direction == 0) {
    if (justStarted)
      justStarted = false;
    else
      //hardCoreMode is for later functionality
      //in hardCoreMode, robot would deviate from
      //right-wall-following algorithm.
      hardCoreMode = true;
  }

  return;
}

//updates maze based on what the robot sees and sends it to the base station
void updateMaze() { //also sends maze
  int currentSpace = location[1]*9 + location[0];
  int spaceAhead = (direction == 0 ? currentSpace-9 : (direction == 1 ? 
currentSpace+9 :
                    (direction == 2 ? currentSpace+1 : currentSpace-1)));
  int spaceLeft = (direction == 0 ? currentSpace-1 : (direction == 1 ? 
currentSpace+1 :
                    (direction == 2 ? currentSpace-9 : currentSpace+9)));
  int spaceRight = (direction == 0 ? currentSpace+1 : (direction == 1 ? 
currentSpace-1 :
                    (direction == 2 ? currentSpace+9 : currentSpace-9)));
  int cornerUL = currentSpace-10;
  int cornerUR = currentSpace-8;
  int cornerDL = currentSpace+8;
  int cornerDR = currentSpace+10;

 // maze[currentSpace] = 1;
  maze[spaceAhead] = getSensorFront() + 1; //1 for open space, 2 for wall, 0
for unexplored.
  maze[spaceLeft] = getSensorLeft() + 1;
  maze[spaceRight] = getSensorRight() + 1;

//extend walls
  if (maze[currentSpace-9] == 2 || maze[currentSpace-1] == 2)
    maze[cornerUL] = 2;
  else
    maze[cornerUL] == 1;
  if (maze[currentSpace-9] == 2 || maze[currentSpace+1] == 2)
    maze[cornerUR] = 2;
  else
    maze[cornerUR] = 1;
  if (maze[currentSpace+9] == 2 || maze[currentSpace-1] == 2)
    maze[cornerDL] = 2;
  else
    maze[cornerDL] = 1;
  if (maze[currentSpace+9] == 2 || maze[currentSpace+1] == 2)
    maze[cornerDR] = 2;
  else
    maze[cornerDR] = 1;

  return;
}

//sends maze in 4 packets
boolean sendMaze() {

  char maze0[26], maze1[26], maze2[26], maze3[26];
  maze0[0] = 0;
  maze1[0] = 1;
  maze2[0] = 2;
  maze3[0] = 3;
  for (int i = 0; i < 25; i++) {
    maze0[i+1] = maze[i];
    maze1[i+1] = maze[i+25];
    maze2[i+1] = maze[i+50];
    maze3[i+1] = maze[i+75];
  }

  bool end = radio.write(&maze0, 26);
  end = end && radio.write(&maze1, 26);
  end = end && radio.write(&maze2, 26);
  end = end && radio.write(&maze3, 26);

}
```

Base Station Arduino Code (robotReceiver.ino)

```
#include <nRF24L01.h>
#include <RF24.h>
#include <RF24_config.h>
#include <SPI.h>

RF24 radio(9,10);
const uint64_t pipe = 0x000000001ELL;

void setup() {
  //set up SPI
 setup_softSPI();
  // set up radio stuff
 radio.begin();
 radio.setRetries(15,15);
 radio.setAutoAck(true);
 radio.setChannel(0x50);
 radio.setPALevel(RF24_PA_HIGH);
 radio.setDataRate(RF24_250KBPS);
 radio.openReadingPipe(1,pipe);
 radio.startListening();
}

//data is the message holder, maze is the maze holder
char data[26];
char maze[100];

boolean dataNew = false;
byte location;
byte color;

void loop() {
  if (radio.available()) {
    boolean done = false;
    while (!done) {
      done = radio.read( &data, 26);
    }
    if (data[0] == 0)
      for (int i = 0; i < 25; i++)
        maze[i] = data[i+1];
    else if (data[0] == 1)
      for (int i = 0; i < 25; i++)
        maze[i+25] = data[i+1];
    else if (data[0] == 2)
      for (int i = 0; i < 25; i++)
        maze[i+50] = data[i+1];
    else {
      //once we receive the last packet, need to send updated maze via SPI
      dataNew = true;
      for (int i = 0; i < 25; i++)
        maze[i+75] = data[i+1];
    }
  }
  if (data[0] == 3 && dataNew) {
    for (int i = 0; i < 99; i++) {
      //4 bits x, 4 bits y, 2 bits color
      location = ((byte) (int) (i/9)) & B00001111; //4 MSB x, 4 LSB y
      location += ((i%9 << 4) & B11110000);
      color = (byte) maze[i] & B00000011;
      //send 10 bits as a 2-byte message via SPI
      sendMsg(location, color);
    //no new data to send as of yet
    dataNew = false;
  }
}
```

Base Station SPI Arduino Code (soft_spi.ino)

```
/* This code made for Cornell University's ECE 3400 class, Spring '15.
   Please do not share or copy this code with anyone who is not enrolled
   in the course. */

void setup_softSPI() {
  int CS = 2; //chip select pin (also called slave select)
  int clk = 3; //clock pin
  int data = 4; //data pin

  pinMode(CS, OUTPUT);
  pinMode(clk, OUTPUT);
  pinMode(data, OUTPUT);
  digitalWrite(CS, HIGH);
  return;
}

void sendMsg(byte message1, byte message2) {
  int CS = 2; //chip select pin (also called slave select)
  int clk = 3; //clock pin
  int data = 4; //data pin

  //toggles the clock once with chip select high,
  //then sets chip select low and sends data.
  digitalWrite(clk, LOW);
  delayMicroseconds(5);
  digitalWrite(clk, HIGH);
  delayMicroseconds(5);
  digitalWrite(clk, LOW);
  digitalWrite(CS, LOW);
  digitalWrite(data, HIGH && (message1 & B00000001));
  digitalWrite(clk, HIGH);
  delayMicroseconds(5);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message1 & B00000010));
  digitalWrite(clk, HIGH);
  delayMicroseconds(5);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message1 & B00000100));
  digitalWrite(clk, HIGH);
  delayMicroseconds(5);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message1 & B00001000));
  digitalWrite(clk, HIGH);
  delayMicroseconds(5);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message1 & B00010000));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message1 & B00100000));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message1 & B01000000));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message1 & B10000000));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message2 & B00000001));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message2 & B00000010));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message2 & B00000100));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message2 & B00001000));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message2 & B00010000));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message2 & B00100000));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message2 & B01000000));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(data, HIGH && (message2 & B10000000));
  digitalWrite(clk, HIGH);
  delayMicroseconds(10);
  digitalWrite(clk, LOW);
  digitalWrite(CS, HIGH);
  return;
}
```

Base Station VGA logic code (VGAlogicSPI.v)

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

//THIS PROJECT DEVELOPED BY CORNELL ECE 3400 STAFF. PLEASE
//DO NOT REUSE OR DISTRIBUTE THIS CODE WITHOUT PERMISSION
//SPRING 2015

module VGAlogicSPI(

	//////////// CLOCK //////////
	CLOCK_50,

	//////////// KEY //////////
	KEY,

	//////////// GPIO_0, GPIO_0 connect to GPIO Default //////////
	GPIO,
	GPIO_IN,
	LED
);

//=======================================================
//  PARAMETER declarations
//=======================================================


//=======================================================
//  PORT declarations
//=======================================================

//////////// CLOCK //////////
input 		          		CLOCK_50;

//////////// KEY //////////
input 		     [1:0]		KEY;

//////////// GPIO_0, GPIO_0 connect to GPIO Default //////////
inout 		    [33:0]		GPIO;
input 		     [1:0]		GPIO_IN;
output			[7:0]			LED;


//=======================================================
//  REG/WIRE declarations
//=======================================================
reg CLOCK_25;
wire [9:0] PIXEL_COORD_X;
wire [9:0] PIXEL_COORD_Y;
wire [7:0] PIXEL_COLOR;

wire reset;
reg [1:0] pixel_in;
```

```
wire [9:0] pixel_r_index;
reg [9:0] pixel_w_index;
wire [1:0] pixel_out;
reg pixel_wen;

reg [9:0] counter; //for loop is too long to synthesize
reg counter_go; //counter enable

VGADriver driver(
      .CLOCK(CLOCK_25),
      .PIXEL_COLOR_IN(PIXEL_COLOR),
      .PIXEL_X(PIXEL_COORD_X),
      .PIXEL_Y(PIXEL_COORD_Y),
      .PIXEL_COLOR_OUT({GPIO[9],GPIO[11],GPIO[13],GPIO[15],GPIO[17],GPIO[19],
GPIO[21],GPIO[23]}),
      .H_SYNC_NEG(GPIO[7]),
      .V_SYNC_NEG(GPIO[5])
);

blockArray pixelArray(
      .clk(CLOCK_50),
      .r_index(pixel_r_index),
      .w_index(pixel_w_index),
      .value(pixel_in),
      .w_en(pixel_wen),
      .out(pixel_out)
);

//reg comm_rdy_in;
reg comm_val_in;
wire comm_rdy_out;
wire [3:0] comm_x_out;
wire [3:0] comm_y_out;
wire [1:0] comm_color_out;
wire [8:0] comm_command_in;

arduinoComm commModule( //TODO: declare these wires and hook them up
      .clk_in(GPIO[29]),
      .cs_in(GPIO[30]),
      .data_in(GPIO[28]),
      .val_out(comm_val_out),
      .x_out(comm_x_out),
      .y_out(comm_y_out),
      .color_out(comm_color_out),
      .reset(reset)
);

assign pixel_r_index[9:5] = PIXEL_COORD_X[9:5]; //shift by 5 for pixel->block
transform
assign pixel_r_index[4:0] = PIXEL_COORD_Y[9:5];

//=======================================================
//  Structural coding
//=======================================================
always @(posedge CLOCK_50) begin
      CLOCK_25 <= ~CLOCK_25; //VGA needs 25 MHz clock - FPGA has 50 MHz clock
      //reset logic
```

```
		if (reset) begin
			counter <= 0; //iterate through all memory locations
			counter_go <= 1;
			comm_val_in <= 0;
		end
		//reset logic cont'd
		else if (counter_go) begin
			pixel_wen <= 1; //write
			pixel_w_index <= counter; //location
			pixel_in <= 2'b00; //value
			if (counter == 10'b11_1111_1111) begin
				counter_go <= 0; //end of loop
				pixel_wen <= 0;
			end
			counter <= counter + 10'd1;
		end
		//regular operation logic
		else begin
			pixel_wen <= comm_val_out;
			pixel_w_index[9:5] <= comm_x_out + 5'b00000;
			pixel_w_index[4:0] <= comm_y_out + 5'b00000;
			pixel_in <= comm_color_out;
		end
	end

	assign reset = ~KEY[0];

	//translate 2 bit pixel array values to 8 bit RGB, 0 = unexplored (black), 1
= open (white), 2 = wall (blue-green), 3 = unexplorable (red)
	assign PIXEL_COLOR = (pixel_out == 2'b00 ? 8'b000_000_00 : (pixel_out ==
2'b01 ? 8'b111_111_11 : (pixel_out == 2'b10 ? 8'b000_111_11 :
8'b111_000_00)));
	assign LED[3:0] = comm_y_out;
	assign LED[7:4] = comm_x_out;

	endmodule
```

VGA Driver code (VGADriver.v) – by Alex Jaus

```
//THIS PROJECT DEVELOPED BY CORNELL ECE 3400 STAFF. PLEASE
//DO NOT REUSE OR DISTRIBUTE THIS CODE WITHOUT PERMISSION
//SPRING 2015

`define TOTAL_SCREEN_WIDTH 795
`define TOTAL_SCREEN_HEIGHT 525
`define VISIBLE_SCREEN_WIDTH 640
`define VISIBLE_SCREEN_HEIGHT 480

module VGADriver (
	CLOCK,
	PIXEL_COLOR_IN,
	PIXEL_X,
	PIXEL_Y,
	PIXEL_COLOR_OUT,
	H_SYNC_NEG,
      V_SYNC_NEG
);

/******
* I/O *
******/

input CLOCK; //PIXEL CLOCK - DRIVE AT 25MHZ for 60 Hz 640 x 480 VGA
input  [7:0] PIXEL_COLOR_IN; //COLOR GIVEN TO THE VGA DRIVER


output [9:0] PIXEL_X; //HORIZONTAL POSITION OF THE NEXT PIXEL;
output [9:0] PIXEL_Y; //VERTICLE POSITION OF THE NEXT PIXEL;
output [7:0] PIXEL_COLOR_OUT; //COLOR TO BE DISPLAYED
output       H_SYNC_NEG; //THE REVERSE POLARITY HORIZONTAL SYNC SIGNAL
output       V_SYNC_NEG; //THE REVERSE POLARITY VERTICAL SYNC SIGNAL

/****************************
* MEMORY AND INTERNAL WIRES *
****************************/

reg  [9:0] pixel_count;
reg  [9:0] line_count;

/************************
* LOGIC AND CONNECTIONS *
************************/
assign PIXEL_X    = pixel_count;
assign PIXEL_Y    = line_count;

assign PIXEL_COLOR_OUT = (pixel_count<(`VISIBLE_SCREEN_WIDTH))
                                                      ? (PIXEL_COLOR_IN) :
(8'b00000000) ; //ROUTE THE INPUT COLOR TO OUTPUT IF WITHIN VISIBLE BOUNDS

assign H_SYNC_NEG = (pixel_count>=656 && pixel_count<752) ? (1'b0) : (1'b1);
//ACTIVATE THE H SYNC PULSE AFTER FRONT PORCH

assign V_SYNC_NEG = (line_count>=490 && line_count<492) ? (1'b0) : (1'b1);
//ACTIVATE THE V SYNC PULSE AFTER FRONT PORCH

always @(posedge CLOCK) begin
      if (pixel_count == (`TOTAL_SCREEN_WIDTH-1)) begin
            pixel_count <= 0;
            if (line_count == (`TOTAL_SCREEN_HEIGHT-1)) begin
            line_count <= 0;
      end else begin
            line_count <= line_count + 1;
      end
      end else begin
            pixel_count <= pixel_count + 1;
            line_count <= line_count;
      end
end

endmodule
```

VGA pixel block code (blockArray.v)

```
//THIS PROJECT DEVELOPED BY CORNELL ECE 3400 STAFF. PLEASE
//DO NOT REUSE OR DISTRIBUTE THIS CODE WITHOUT PERMISSION
//SPRING 2015

module blockArray (
      clk,
      r_index,
      w_index,
      value,
      w_en,
      out
);

input clk;
input [9:0] r_index; //x&y coords
input [9:0] w_index; //x&y coords
input [1:0] value; //write value
input w_en; //write enable
output [1:0] out; //gives r_index's value

reg [1:0] array[1023:0]; //data array
reg [1:0] pre_out; //since out value is a wire

always @(posedge clk) begin
      if (w_en == 1) begin //write value
            array[w_index] <= value;
      end
      pre_out <= array[r_index];
end

assign out = pre_out;
endmodule
```

Base Station SPI code (arduinoComm.v)

```
module arduinoComm (
      clk_in,
      cs_in,
      data_in,
      val_out,
      x_out,
      y_out,
      color_out,
      reset
);

input reset;
input cs_in;
input clk_in;
input data_in;
output val_out;
output [3:0] x_out; //message 1 MSBs
output [3:0] y_out; //message 1 LSBs
output [1:0] color_out; //message 2 LSBs

reg val_holder;
reg [15:0] message_holder;
reg pre_val_out;
reg [15:0] message_out;
reg [3:0] counter;
reg done;

always @(posedge clk_in) begin
      if (cs_in == 0) begin
            val_holder <= 1;
            message_holder[counter] <= data_in;
            if (counter == 4'b1111) begin
                  message_out <= message_holder;
                  if ((message_holder[3:0] > 4'd11) || (message_holder[7:4] >
4'd9)) begin
                        pre_val_out <= 0;
                  end
                  else begin
                        pre_val_out <= val_holder;
                  end
            end
            counter <= counter + 4'd1;
      end
      else begin
            counter <= 4'd0;
      end
end

assign val_out = pre_val_out;
assign y_out = message_out[3:0];
assign x_out = message_out[7:4];
assign color_out = message_out[9:8];

endmodule
```