Maseeh College of Engineering and Computer Science

PORTLAND STATE UNIVERSITY

PSU Capstone Final Report:

Porting Existing Smart Ballot Box 2 Software to an Arduino Uno-R3 ATmega328P Platform

Prepared By:

Ali Saad

Jonathan Christian

Nick Long

Jiaqi Liu

Prepared For:

Joe Kiniry

Joey Dodds

Daniel Zimmerman

Faculty advisor:

Tom Schubert

Revision Number: 3

Date of Revision: June 8, 2020

Date of Document Submission: June 12, 2020

FREE & FAIR

Executive Summary

Our sponsor, Free & Fair is a B Corp organization governed by a guiding mission statement, to make elections more verifiable. This leads them to create technological advances that help them achieve their mission. Over the last couple of years, they have been experimenting with a central component of the voting system - the smart ballot box. The result is a fully functioning Smart Ballot Box (SBB2019) prototype. It addresses serious concerns with electronic voting systems like fraud and manipulation, but it is expensive to reproduce as it features a high-performance Xilinx FPGA, limiting replication. Our task is to create a more affordable version of their Smart Ballot Box (SBB) while maintaining its functionality. However, we will forgo the full implementation of the SBB and use the CASCADIO board in combination with a microcontroller.

Voters have good reason to be concerned about how elections are conducted, particularly with the current trend towards using digital systems to automatically tabulate results. While elections could benefit greatly from this technological improvement, there remain serious issues to address. For example, the problems experienced by the Iowa Caucuses this year demonstrated technology selections are not up to the task of even collecting results, let alone providing a means to audit the outcome.

Free & Fair intends to solve this problem with their current Smart Ballot Box prototype. However, the current prototype costs over 9,000 dollars, which is something most voting centers cannot afford. Our contribution will help them achieve their goal. We have been tasked with seeing if an Arduino Uno Rev3 with an ATmega328P chip would be a suitable replacement for the original FPGA while providing the necessary processing power for the Smart Ballot Box. The hope is that it will be used in conjunction with the CASCADIO board during testing procedures. It incorporates all the necessary sensors and will replicate the functionality of the SBB2019 on a single device. PMOD connectors are available for external peripherals to be attached when using the CASCADIO board. This goal is to port over all the software, firmware, and FreeRTOS to a UNO microcontroller and yield a successful prototype built and assembled by May 29, 2020. A demonstration of our project will happen the second week in June to determine if this proposal and requirements meet Free & Fair's expectations.

Due to COVID-19 crisis that started in March, Portland State University has decided to close the campus and make all activities remote, including capstone projects. This will require all of us to work and communicate through online resources. Beyond those times, we will need to work in a predetermined development environment on our personal computers.

To ensure we have a successful outcome from this project, we have worked with Free & Fair and our PSU advisor. We started off getting an overview of the Smart Ballot Box and its accompanying software work first hand. From there we delved deeper into how the software on the backend worked and how we could use the CASCADIO board to test. We then worked on getting freeRTOS working for the Atmega328P chip. Once we reached success, we then worked

to take the SBB code, which was provided from Free & Fair, and implemented a copy around the FreeRTOS section. We were not able to upload to the Atmega 328p due to memory constraints. However, we were able to compile the majority of the code and realize that we will need a processor with significantly more memory to handle this program.

Problem background

In 2019, Free & Fair and Galois developed such a ballot box, which is the Smart Ballot Box (SBB2019). However, the current FPGA used as the CPU costs over $9,000 itself, making it an expensive system for voting centers. It is not acceptable as a focus of teaching and learning in a university setting. No course nor individual can reasonably afford a device priced as such. In order to decrease the cost of the box, we have been given the task of finding a more cost-effective CPU that will reduce the overall system cost below $500. The ideal price we are aiming for is $300.

Need statement

- Most voting centers cannot afford to spend $9000+ for a single smart ballot box
- The cost prevents private citizens from exploring the established security features and experimenting with how it works
- The current prototype uses some custom parts, not easily obtained outside of the specific manufacturers
- The current prototype can't be easily replicated and may not be safe to produce without proper training
- The current COTS components are not available everywhere and in limited supply (not long term)

Objective statement

The main objective of this project is to create a more affordable version of the Smart Ballot Box created by Free & Fair by substituting an Arduino UNO in place of the Xilinx FPGA while keeping the original functionality and security features.

- Make an affordable Smart Ballot Box (excluding external sensors/unit housing)
- Ensure the smart ballot box is easy to replicate
- Ensure the smart ballot box is safe to operate and manufacture
- Limit the number of custom parts and favor COTS components
- Promote experimentation of the smart ballot box by private citizens and explorers

Deliverables

1. Our Smart Ballot Box 2 will have the same functionality as the original SBB2019 (except network logging and DEFCON debugging) as described in the Github BVS2019 project with the deployment version of our choosing. However, physical housing is excluded.
2. The Arduino Uno will interface with the peripherals identically like the FPGA.
3. The system will be easy to replicate and program with a COTS microcontroller.
4. The cost will be below $500 to allow others to replicate our work.
5. The overall design will allow for easy exploration, debugging, modification, and experimentation for the future development of the Smart Ballot Box concept. This System is not a final product but apart from a more widespread effort to improve voting system technology.
6. Project Proposal, Biweekly/Weekly Progress Reports (with milestones), a Final Report, Detailed Documentation, and a Capstone Presentation.

These deliverables will be achieved by checking in with our advisor and sponsor regularly while updating them on the progress we are making.

Requirements

1. Functionality
    - The Product SHOULD have the same functionality as the original SBB2019 prototype excluding external sensors/unit housing
    - The Product MUST have cast and spoil buttons
    - The Product MUST have an LCD screen
2. Performance
    - The Product SHOULD recognize a ballot within 20 seconds of being scanned
    - The Product MUST weigh less than 10 pounds
    - The Product SHOULD be smaller than 24”x 24”x 48”
    - The Product MAY house the ballots securely with a lock
3. Economic
    - The Product MUST cost less than $500
    - The Product SHOULD cost less than $300
    - The Product MAY cost less than $200
4. Power/Energy
    - The Product MUST be energy efficient
    - The Product MUST be able to supply 12V
5. Health and Safety
    - The Product MUST not use harmful or toxic materials
    - The Product SHOULD be of durable construction

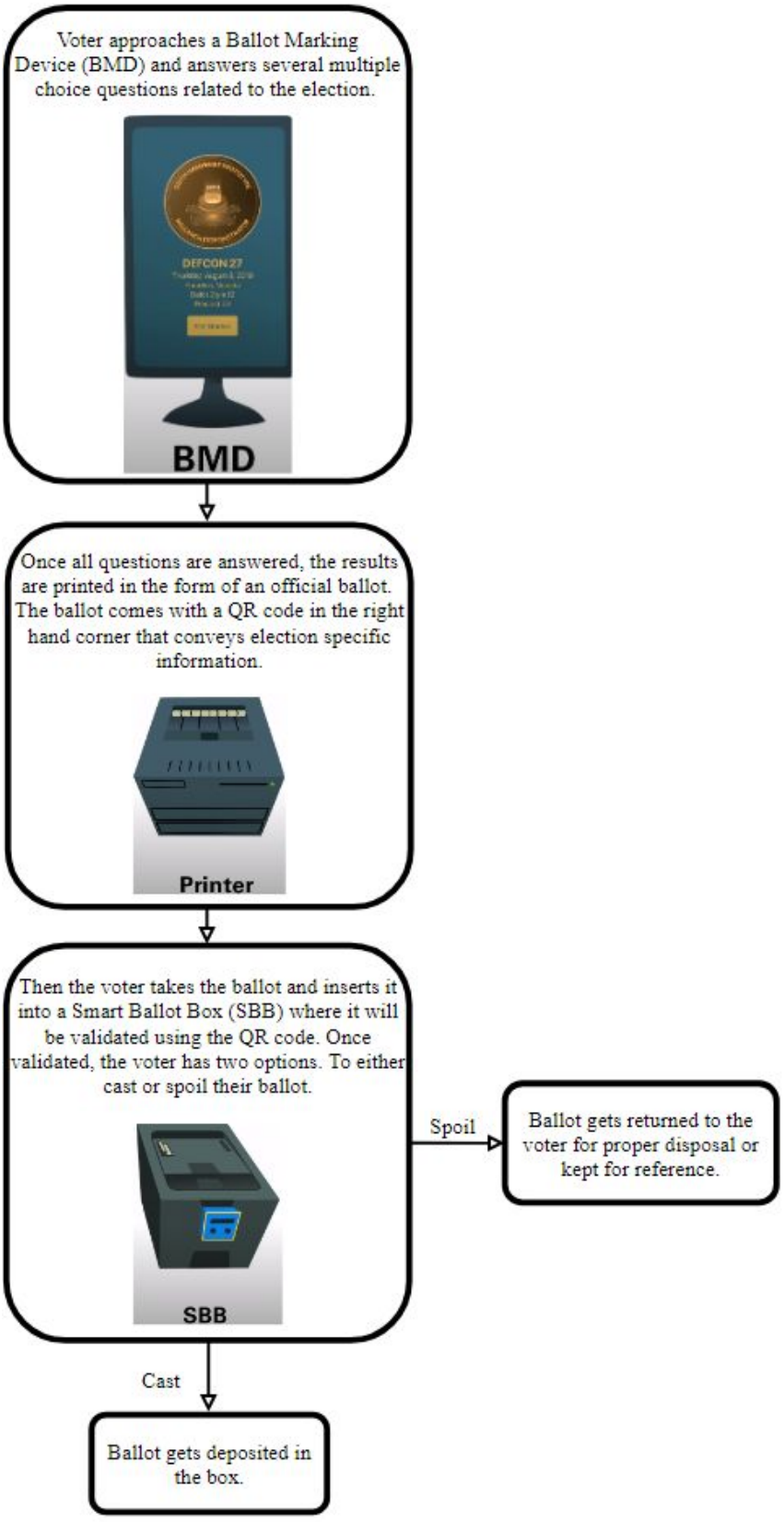
- The Product MUST not have sharp or jagged edges
- The Product SHOULD be enclosed, no point of contact with electrical components

6. Environmental
    - The Product SHOULD be recyclable
7. Legal
    - The Product SHOULD abide by the law and not violate citizens rights
8. Political
    - The Product MUST remain neutral in all elections
9. Usability
    - The Product MUST not require training to operate
10. Documentation
    - The Product MUST have clear and detailed documentation
11. Marketing
    - The Product MUST be cost-effective
12. Reliability & Availability
    - The Product MUST execute cast or spoil correctly
13. Physical Operation
    - The Product MUST be usable by polling center employees that have no prior technical knowledge and little training.
14. Manufacturing
    - The Product MUST allow others to replicate the work
15. Maintainability
    - The Product MUST be easily serviced
16. Software Security
    - The time a ballot is submitted SHOULD be tracked
    - SHOULD recognize whether a ballot has already been scanned
    - Every QR code SHOULD different/randomly generated but yet has the contents it needs to like the time and what election it is
    - SHOULD count how many ballots are cast, that way if tampering occurs we know can verify how many ballots are in the SBB
    - A user MUST not be able to configure any of the peripheral devices, allowing access to the system

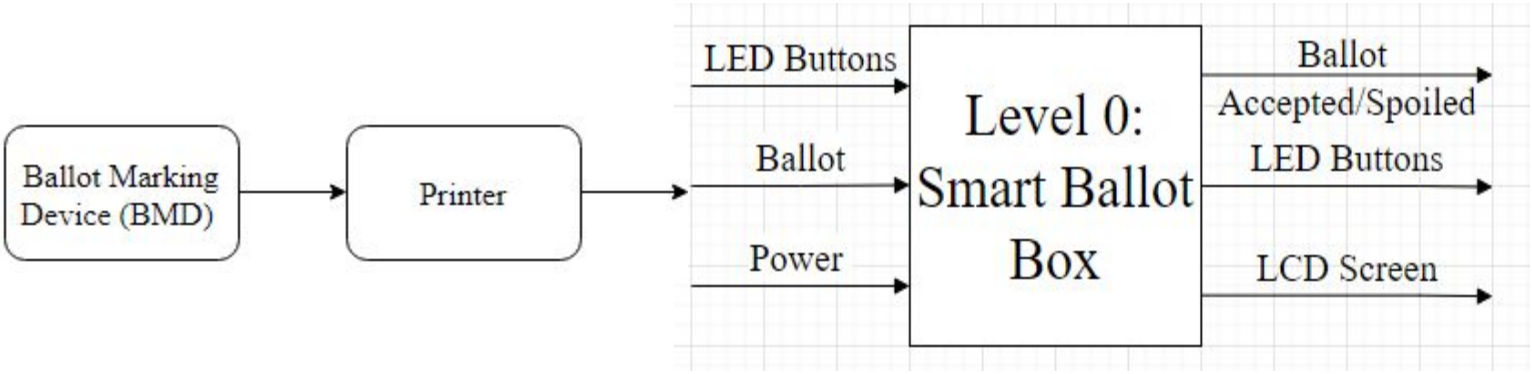
Design

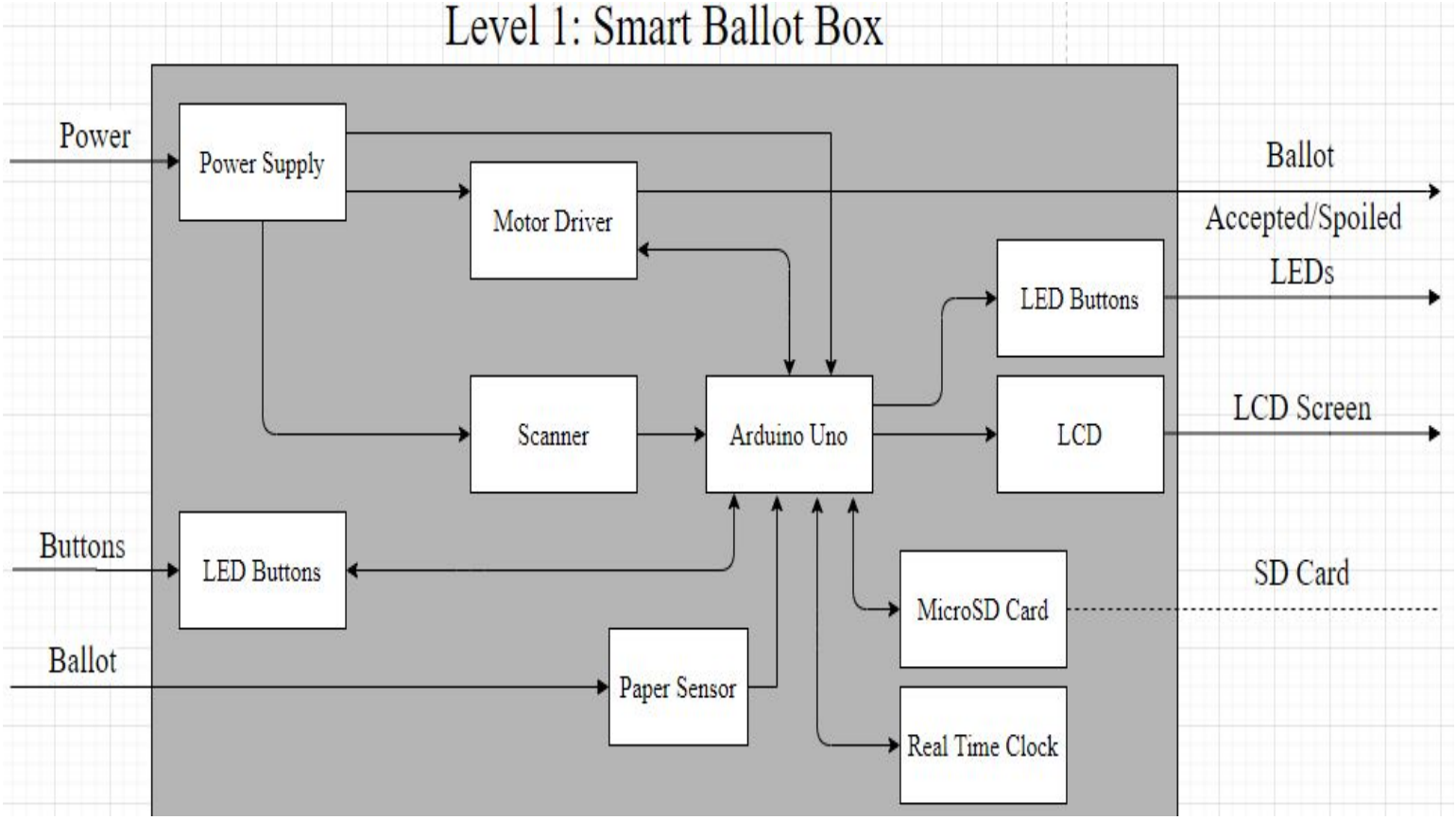
As mentioned above, we are proposing to make a more affordable version of the Smart Ballot Box. The way we are going to achieve this is to analyze and learn from the Github repositories kindly provided by Free & Fair. Resulting in a SBB driven by an Arduino Uno.

Voter flow chart:



Block Diagram:

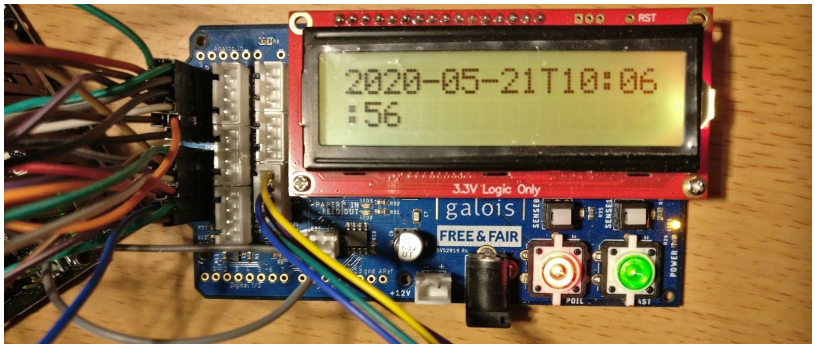




Implementation

Hardware:

As for the hardware part, we were successful in performing all the unit tests involving the CASCADIO board and the UNO. This consisted of running Arduino test programs to verify that all connections and sensors were operating correctly. The combination was constructed by referring to schematics and diagrams provided by our sponsor. Their documentation was created for another Arduino microcontroller so cross-referencing was key. This led to the pin layout diagram in Appendix A. We connected them based on the schematic provided and got the LED turn on and LCD working with showing the real-time. The SD card also works well and it can record and save the data in it.



Software:

- The way we implemented the software is described below in the post mortem.

Test Plan

Level 1: Unit Testing

- LCD Screen Performance Testing
- Barcode Scanner Testing
- Power Supply Performance Testing
- Real-Time Clock Performance Testing
- MicroSD Card Performance Testing
- UNO Has The Correct Firmware
- Cast or Spoil Performance Testing

Level 2: Software Testing

- Test the functionality of the Software
- Make Sure Each I/O Is Connected To The Proper Port

Correct RTOS Working Properly on UNO
SBB Software Working Properly on UNO

Refer to the Test Plan Document for more details!

Results

For the hardware portion of this project, the deliverables achieved are a combination of working hardware consisting of the Arduino UNO and the CASCADIO board (shield). We were given SD cards, LEDs, and LCD working properly. However, for the scanner part, we only do a unit test, which just tested the effectiveness of the barcode scanner, but didn't Implement any further code. Due to de COVID-19, we cannot get more components to go further in the hardware part.

For the software part, this was a great learning experience as we faced obstacles we have never come across before in all our years in college. We began by reading up on an operating system that is known as FreeRTOS, which we have no experience in. We took on the challenge of trying to optimize a code that was originally run on a 32-bit RISC-V process to be compiled/run on an 8-bit ATmega328P processor. We began our approach by trying to understand how everything in the SBB works on top of FreeRTOS. We decided the best way is to work from the ground up and use everything in Free & Fair's FreeRTOS folder as a resource when we are blocked. Our goal is to wrap everything back together and combine our FreeRTOS file with Free & Fair's SBB functionality to make it work together.

We originally tried getting FreeROTS to work on the Arduino IDE to find out that the code takes 140% of the UNO capacity and it was mostly in processing so this wouldn't work when most of Free & Fair work is coded in C, which the IDE is better used for processing. From here we went on to download a VM and get Ubuntu running and changing the environment to working in Linux. The first action we took was downloading FreeRTOS from the freertos.org and seeing how we are going to implement it to work for the Atmega328P. This was very difficult because we had to critically think which files we didn't need and which we can get rid of as well as which files we would need to optimize in order to fit on the 32k flash memory we had to work with.

The main obstacle we faced was there wasn't a FreeRTOS created for an ATmega328P before our doings, so it was difficult looking for any previous examples. The closest processor that we found FreeRTOS for was for an ATmega323. We compared both files and removed and added the needed dependencies and variables in order to have everything pointed to the correct path. This led to FreeRTOS compiling but we wanted to test to make sure we have FreeRTOS truly working on the UNO by creating a flashing LED task and running at different speeds. We were required to get the correct bits to hit the correct PORT. With what seemed like the correct bits we were still blocked on getting the LED to blink. The issue was the program never got to the point of calling the correct function to make the LED flash. With the help of Joe, this was

accomplished by getting a working FreeRTOS and getting a blinking LED on the Arduino UNO R3 with an ATmega328P processor.

After we claimed victory on compiling FreeRTOS on the UNO successfully, we tested multiple PORTs on the UNO to make sure we had FreeRTOS running properly. We made sure we had the correct bits for each port, which led to it working properly no matter what PORT we used. From here the next obstacle we faced was now porting in our existing FreeRTOS over to Free & Fair's SBB code. This was challenging since all the code is built to work with a RISC-V processor as well as all the files are connected to their FreeRTOS file which we needed to change for it to point to ours. This was very tricky to find out where all the paths are being directed too as well as what commands are specifically for the RISC-V processor.

Due to working with such a small memory size, Free & Fair allowed us to remove anything to do with networking, logging, and crypto. We worked with Joe to work through the dependencies and figure out the correct file paths and includes, in order to compile all the .o files that were required to build and get the final measurements of the project. All the .o files together would figure out a rough estimate on how big of memory size is required to build FreeRTOS + SBB functionality code. The results below show the memory size of each subsection required to have a fully functioning Smart Ballot Box.

File Size				
	Free & Fair Results (Before optimization)	Free & Fair Results (Optimized)	Team 4 Results (Our FreeRTOS with Free & Fair SBB code)	FreeRTOS running with Blinking LED (before importing with Free & Fair Code)
FreeRTOS Results:	188k	122k	72K	11k
SBB Results:	178k	115k	112k	N/A
Crypto Results (Part of SBB Code):	66k	N/A	11.7k	N/A
Networking	N/A	N/A	N/A	N/A
Logging	N/A	N/A	N/A	N/A
Total Memory Result:	366k	237k	184K	N/A

Unfortunately, this didn't meet the stated requirements as we were supposed to be able to have full functionality of the SBB on the CASCADIO board but with the data in the table above, we concluded that the 32K flash memory and 2K RAM in the Arduino UNO is not enough to compile the full SBB functionality. With Joe's help, we were able to successfully compile the .o files and get a rough estimate of the total memory size required to build a Smart Ballot Box.

This way we set up next year's capstone to continue this project by choosing a microprocessor with approximately 300k memory and be able to build a Smart Ballot Box successfully. We appreciate all the help from Free & Fair and learned a lot throughout this whole experience specifically in understanding the complexity of how Makefiles work.

Post Mortem

As mentioned above, we got the logging function involving the SD cards, LEDs, and LCD working, and the scanner did not work fully. It is also acceptable for the hardware part requirement and overall is good. We have learned a lot in the hardware part. First of all, we know more about dealing with Arduino. Not only on the wire connection part but also deal with running coding in the Arduino IDE. Though the shield is provided by Free & Fair, in order to know how it works, we still need to dig into the schematic to find it out. By researching the schematic we can get the idea about the pin layout, the connection between each component. We made some mistakes when we connected the wires, finally, we found out the pin layout in the design for the shield is regularly and easy to be connected, so that there should not come out the strange connection. We learn that when we design a kind of schematic in Eagle Cad, we should make the connection easier to connect, it also helps to check the connections or even use a group of wire to connect them directly. As for working with Arduino IDE, Free & Fair also provides many codes for testing and setting the Arduino. There were many troubles when we tried to test them when we struggled in some of the parts, we dug into the code and knew the reason for it. With this help, we also learn that when we write down some code, and there is some error, we need to give enough guidance for the user to know how to fix that part. In another word, write down the full description and help the user and ourselves.

For the software part, we worked on getting the Smart Ballot Box to run on an Atmega328P aboard the Arduino Uno. We had to start by getting FreeRTOS on the Uno to make sure our operating system and microcontroller were compatible. We set up a task to blink an LED. That was successful, we blinked the LED connected to pin 13 at varying speeds. Then we moved on to getting the Smart Ballot Box to work on the Uno. That task broke down into two parts. Part one was compiling all of the SBB code and part two was uploading that code on to the Uno. We were somewhat successful with part one. The SBB has a core functionality and accompanying auxiliary functionality. The core is the bare minimum needed to run the SBB and the auxiliary being extra functionality like logging, network connectivity, and cryptography.

We focused on compiling the core and cryptography functionality of the SBB since this would result in a safe yet small-sized program. We were able to compile the core components for

the SBB. We were not able to compile the cryptography part since we could not find all the necessary components to make it work with the processor on the Uno. This also means we were not able to compile the networking or logging components since we weren't going to be using those.

After a successful compilation, we then looked at uploading the program on to the Uno. We were unsuccessful in this since the available memory was drastically less than what was needed to hold the program. However, this was not a failure in our opinion. We set out to see if we could get the Smart Ballot Box to run on an Atmega328P aboard the Arduino Uno, which it didn't but we now have an idea of what we need in the next platform to try on. We learned what the size was for the SBB on an Atmega328P. If we were to continue this project, we now know the specs required for a microprocessor to run the SBB.

As a team, we learned a lot about what it is like to work in the industry. We learned how to pick up someone else's work and continue developing it while still keeping their original functionality. We also learned a lot about working with microprocessors, particularly an Arduino compatible microprocessor. We had no idea that there were so many ways to program them besides using the Arduino IDE. Other IDEs work similar to the Arduino IDE or you can work via the command line like us. It was very interesting to find out that Arduino IDE on a simple level is just a text editor that will compile your code and upload it to a microprocessor with AVRDUDE. And lastly, a large part of what we learned was what FreeRTOS is and how to work with it. We are used to code running in loops over and over hitting switch cases and for loops to execute some functionality. But never had we worked in an environment where you can have multiple tasks controlled from one supervisor, or in our case an operating system.

Collaboration Site and Repository

https://github.com/jonathancpdx/Capstone-BallotBox
On top of Github, our team collaborated on **Google Drive and Keybase**
Used Google Hangouts as our primary communication (Due to COVID-19)

Reference Links

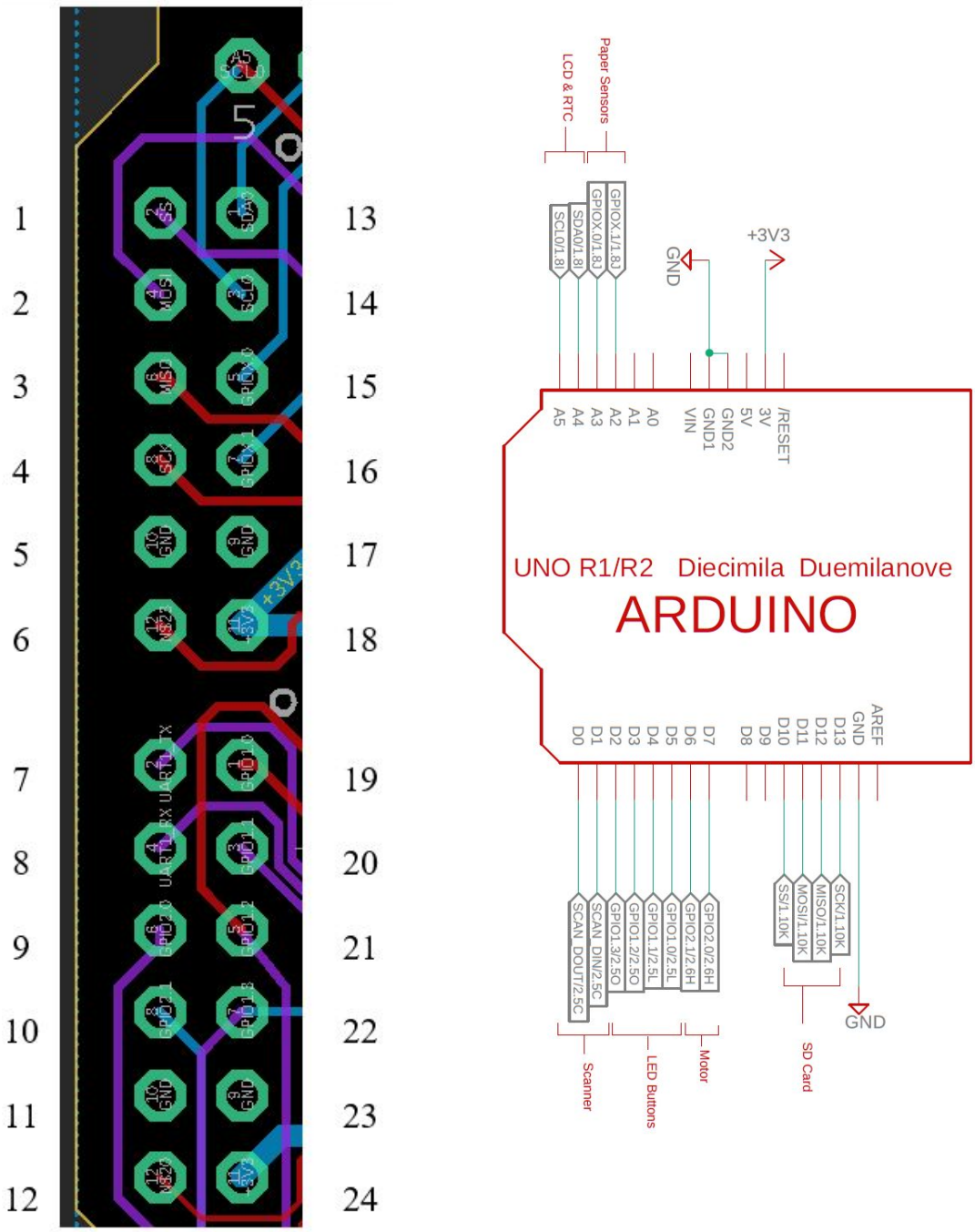
- https://github.com/jonathancpdx/Capstone-BallotBox
- http://web.cecs.pdx.edu/~faustm/capstone/projectdescriptions/2020/ed04f8d4-0c86-473c-9faf-5abff111ec65/SBB%202019%20Port%20Capstone%20(PSU).pdf
- https://securehardware.org/
- https://www.crowdsupply.com/free-and-fair/cascades
- https://github.com/GaloisInc/BESSPIN-GFE-2019

- https://github.com/GaloisInc/BESSPIN-FreeRTOS-2019
- https://github.com/GaloisInc/BESSPIN-Voting-System-Demonstrator-2019/
- https://github.com/kokke/tiny-AES-c
- https://github.com/fspmarshall/tiny-sha256
- https://github.com/tomcumming/base64

Appendix A: Hardware Pin Layout

- **Hardware**

  Pin Layout: SHIELD Pin Diagram ARDUINO Pin Diagram



Arduino Pin	Connection	Shield Pin
A0	None	Not Used
A1	None	Not Used
A2	GPIOX.0	**16** - Paper Sensor
A3	GPIOX.1	**15** - Paper Sensor
A4	SDA0	**13** - LCD & RTC
A5	SCL0	**14** - LCD & RTC
D0	SCAN_DOUT	**8** - Scanner
D1	SCAN_DIN	**7** - Scanner
D2	GPIO1.3	**22** - LED Buttons
D3	GPIO1.2	**21** - LED Buttons
D4	GPIO1.1	**5** - LED Buttons
D5	GPIO1.0	**19** - LED Buttons
D6	GPIO2.1	**10** - Motor
D7	GPIO2.0	**9** - Motor
D8	None	Not Used
D9	None	Not Used
D10	SS	**1** - SD Card
D11	MOSI	**2** - SD Card
D12	MISO	**3** - SD Card
D13	SCK	**4** - SD Card
GND		**17**
3.3V		**24, 18**
Power GND		**23**
Power GND		**11, 5**

PORTB: Correspond to Arduino UNO pins 8-13, DDRB
PORTC: Correspond to Arduino UNO, A0-A5, Analog input pins, DDRC
PORTD: Correspond to Arduino UNO pins 0-7, 0-RX, 1 -TX, DDRD
https://www.arduino.cc/en/Reference/PortManipulation

Appendix B：Solutions to Software Issues

One of the biggest issues we had with the project was getting the code to compile. There were several makefiles from the original project that are called in a nested fashion. Since this was something we had never seen there was a learning curve to get to the point we could start making progress. Once we learned how to work with multiple makefiles, we started the task of creating a way to make our project. We took what we learned from when we compiled FreeRTOS on its own during our efforts to port it to the 328p. Originally, we started out reworking the makefiles to only compile the program for our Atmega328P, but after meeting with our team sponsor we changed that approach. Our team sponsor, Joe Kirny, advised us on how to be better programmers. He showed us how to properly make new makefiles to incorporate the ones that already existed. This would allow for future teams to work on this project and have the makefiles that would compile for the different supported processors all from the same location. There would be no need to look for missing pieces of code like we had to do for the 328P when we tried to compile the bare FreeRTOS.

Once we learned how to better utilize the existing makefile and plan for the future groups, we had to fill in our newly created makefiles. This would include filling out all the right file paths to get to the code dependencies. It also included merging over the correct compiler and compiler flags. The original tools used for the project were different than the ones we were going to use so we needed to bring over that information and create switches to trigger the compiler to read the correct information. This was easier than it sounds. We spent a lot of time lost in the file hierarchy tracing down path issues with files that needed to be included.

Once we solved that issue, a much larger issue would surface shortly thereafter. When we picked our board we knew that we wanted something cost-effective and that we had experience with. The sponsor provided a list of boards that could work, one of those was the Arduino Uno. We had used this board the prior term for prototyping purposes so we already had some laying around. What we didn’t realize we needed to consider so much, was the size of the available memory on the chip. As we got more and more files to successfully compile, we realized that just one of the files was going to take all the memory on the chip. Thus, there would be no way to get the whole program on the board. For the next team, we recommend that they thoroughly research the size capacity of the platforms they are considering.

Appendix C：Solutions to Hardware Issues

First of all, if there are some issues, the most important part is to check the connection between the component and the shield.

The connection and schematic for the shield are on this website.

https://github.com/GaloisInc/BESSPIN-Voting-System-Demonstrator-2019/tree/master/hardware/sbb-shield/R4-TA1

After connecting the component to the shield, run the code in

https://github.com/GaloisInc/BESSPIN-Voting-System-Demonstrator-2019/tree/master/hardware/arduino-sketches

In this site, there are 4 tests in the folder(CardInfo, RTCx_example,motor-test,system-check), just run that in the Arduino IDE. A symbol of the connection's success is the LCD turning on. When running the code, not only the monitor but also the LCD screen will show the solution or where it needs to be fixed. An example is shown below.

```
Serial.println("Could not find FAT16/FAT32 partition.\nMake sure you've formatted the card");
```

A probable issue can be the SD card for the shield needs to be formatted to FAT16/FAT32, after doing the formatting it can work. In this way, hardware issues can be fixed.

Appendix D：Future Progression

As for the hardware part, the future progression is to achieve a fully developed enclosure and have the sensors built-in. This will provide the best prototype necessary for proper testing and evaluation.

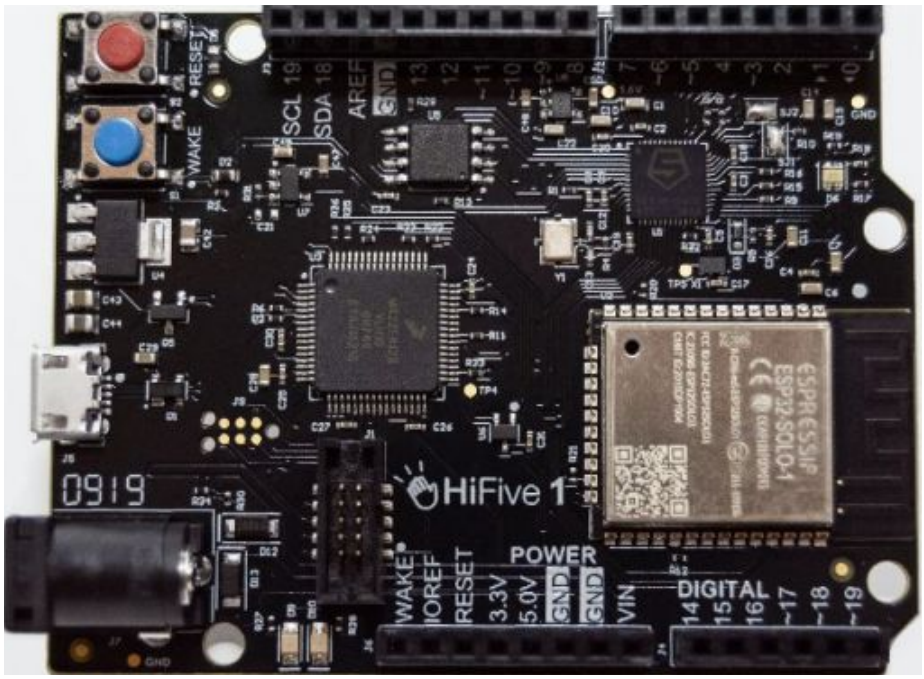
As for the software, the key is to choose an alternative microcontroller that has enough memory to run the Smart Ballot Box. We have provided two possible alternative boards to choose from that have well over the amount of capacity required for this project. We advise you to take the Arduino Due since our project was based around an Arduino UNO, there would be a lot of similarities for the future team otherwise, it's their pick on which board they would like to use. Once a team has chosen a microcontroller with the right amount of memory, the next step is to make sure they configure FreeRTOS to work with their microcontroller properly by testing out if they can get a blinking LED task going. Once they successfully get that done, they should have a good understanding of how FreeRTOS works and should start building SBB code on top of FreeRTOS. A key part that we learned throughout this project is to go slow and compile every two lines of change so you can locate errors quickly. I would recommend testing on a CASCADIO board before trying to do any testing on the SBB, this way you will know if your code is functioning correctly, as the outputs connected to the CASCADIO board should be the correct outputs required to run the SBB.

Appendix E：Alternative Microcontrollers

- **Arduino Due: Approximately $40**

Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz

- **HiFive1 Rev B: Approximately $60**



Input Voltage	5 V USB or 7-12 VDC Jack
IO Voltage	3.3 V
Digital I/O Pins	19
PWM Pins	9
External Wakeup Pins	1
Flash Memory	32 Mbit Off-Chip (ISSI SPI Flash)