Cornell University Baja Racing Team

Electronics Subteam

Spring 2019 Technical Report

Sujith Naapa Ramesh

May 14, 2019

Table of Contents

1 Executive Summary **3**

2 Motivation **4**

3 Firmware **5**

3.1 Purpose 5

3.2 Raspberry Pi 6

3.3 SDIO 6

3.4 SPI 8

4 PCB Design **10**

4.1 Purpose 10

4.2 Schematic Symbol and Footprint 11

4.3 Schematic Capture 13

4.4 PCB Layout 17

4.4 Future Work 19

5 Reflection **19**

1 Executive Summary

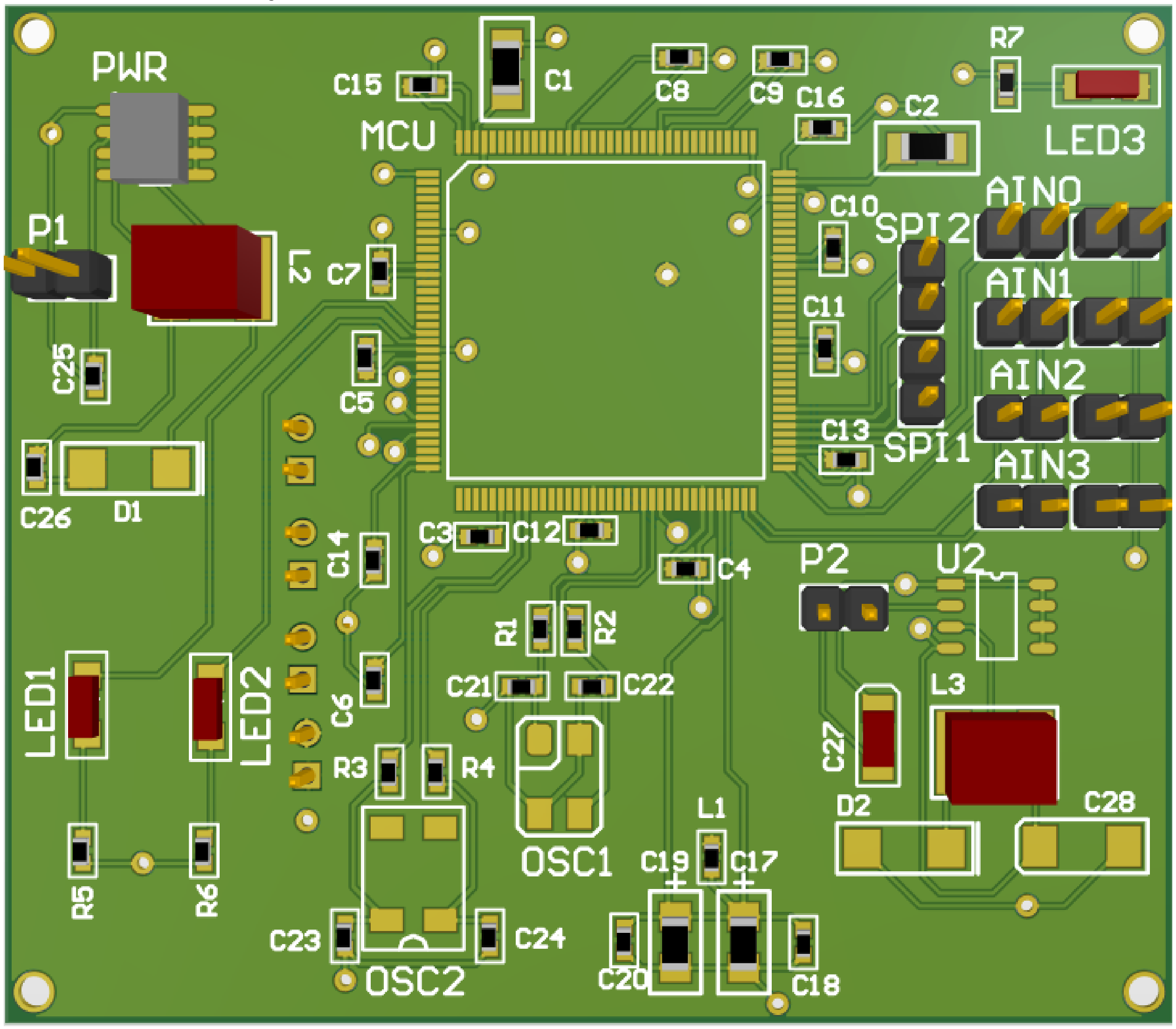


Image 1: Image 1 shows the 3D layout of the DAQLite.

The purpose of my project this semester was to continue upon the work I was doing last semester in redesigning a simplified DAQ system for the team. While the system I designed last semester certainly still works, it is unfortunately very weighty and hard to use. In addition, users had to rely on a TCP connection between the DAQ and host computer to gather log files and analyze them which is a somewhat unintuitive and difficult to use. The new system I have designed is a combination of the two previous full-fledged DAQs designed by the team. I will be using the same MCU as the Jay Fetter DAQ but instead of using a Raspberry Pi to wirelessly stream my data, I have settled upon writing the data to an SD card like the Brian Curless DAQ. By writing the data directly to an SD card, I have been able to achieve a much smaller form factor than the Jay Fetter DAQ. Ignoring the wireless streaming capabilities which has not proven to be too much of a deficit as many of the mechanical engineers on the team were not using those capabilities anyway. The solution I have now settled upon is called the DAQLite. As of now, I have developed the code for writing to the SD card on an STM32 Discovery Board which uses the same MCU that will be used on the project. I have also designed a first revision of the DAQLite PCB which will be printed and brought up next semester.

2 Motivation

My motivation for the project was to get experience designing a full-fledged system in which I would have to design the hardware and write the firmware for the system. Up until now, I have helped design electrical systems, bring other people's boards, and write firmware for projects, but I wanted an experience in which I would be able to put together the whole project myself. Over the last winter break, I researched possibilities for writing to an SD card from a microcontroller and I was able to find some viable methods for doing so. My research inspired to apply this knowledge so that we would be able to better collect data on our off-road vehicle. Doing this project has enabled me to learn many new skills like programming in C, using STM32CubeMX, and Altium Designer. Often times I had to do extensive amounts of research and self-learning on my own before tackling the various stages of the project, and as a result, I feel like I have grown significantly as an engineer.

3 Firmware

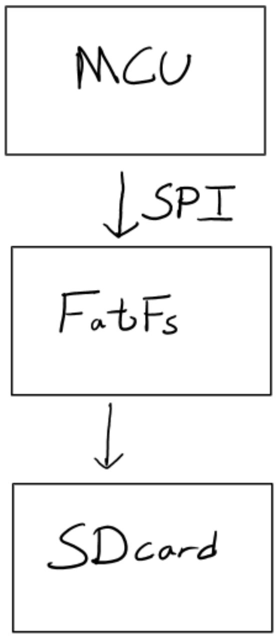


Figure 1: Figure 1 shows the outline of the firmware for writing to an SD card.

3.1 Purpose

The purpose of this stage of the project was to determine the feasibility of building a DAQ system that relies on writing to an SD card. Doing some research showed me that the most common way of writing to an SD card was through an SPI bus where the microcontroller is the master. I have never written an SPI bus before, so the whole process seemed a little daunting at first. I wanted to write the firmware for sending data to an SD card before designing the board so that I would know that designing the board will not be a waste of time and money. I created a timeline for myself in which I would write to an SD card using a Raspberry Pi first as that is the microcontroller, MCU, that I am most comfortable with. I would then replicate my success using the STM32 Discovery Boards because I would be using an STM32 MCU on my PCB. After some more research, I realized that while communicating to the SD card would require knowledge of SPI, I would have to use FatFs, which is a filesystem module that will enable me to access the filesystem present in the SD card. FatFs is not designed to interact with any specific MCU, so I would need to implement the FatFs specific functions specifically for whatever MCU I am using which added an extra layer of complexity. The various parts of the firmware are better pictured in Figure 1.

3.2 Raspberry Pi

The Raspberry Pi portion of writing the firmware was amongst the most stressful and hard to understand parts of the project. My main goal was to utilize the example project that comes with FatFs for a general MCU and rewrite small sections of it to make it more suitable for the Raspberry Pi. I followed this blog post: http://blogsmayan.blogspot.com/p/interfacing-sd-card.html which detailed how to modify the existing code from the FatFs library, but the tutorial was written for a Raspberry Pi 2, and used the bcm2835 library for setting up the GPIO. The Raspberry Pi 3 B+ used for this project relies on a different chip, BCM2837, so I did not feel that the bcm2835 library would be appropriate to use for this project. So, I used a similar library, called WiringPi, to interact with the GPIOs on the Raspberry Pi. However, I was left with limited success as I was not able to ever receive acknowledgement from the SD card. I also did not have a debugger to use on the Raspberry Pi so I was unable to step through the code and understand why my code was failing. Faced with writing the whole SPI library to interface with the Raspberry Pi on a preliminary part of the project seemed like an unwise use of time, so I decided to move on and try different things. If you would like to try using the Raspberry Pi for a similar implementation, I would only recommend using the tutorial if you have a board that uses the BCM2835.

3.3 SDIO

While I was experimenting with the Raspberry Pi, I had decided that I would use the STM32F103C8TC MCU on my board. I settled on this MCU because unlike the STM32F4 chip used on the previous iteration of the DAQ, the STM32F1 MCU would be much easier to design a PCB for and the forty-eight-pin package would be relatively lightweight in terms of bringing up. However, after some more research, I was able to determine that the STM32F4 line comes with a native SDIOMMC controller that would make writing to an SD card significantly easier. Also, I discovered the STM32CubeMX development tool makes developing for the STM chips significantly easier. I found a tutorial for setting up the SDIO bus online using the STM32F4 Discovery Board: https://www.youtube.com/watch?v=0NbBem8U80Y and followed the tutorial. I was able to get writing to the SD card to work seamlessly and I was able to cross a huge hurdle along the way. I decided to use the STM32F429ZIT6U chip used on the Discovery Board on my project as well as the SDIO bus because of how robust the implementation seemed to be.

The SDIOMMC controller on the STM32F4 MCU allows for four bit and one bit SDIO, The four bit SDIO uses four data lines as opposed to the one bit SDIO and SPI so data transfer is much faster. In fact data transfer using SDIO is universally much faster than data transfer using SPI which was another reason for choosing to use the SDIO bus. Also, STM32CubeMX natively integrates the FatFS and SDIO bus so there is no need to manually edit any of the FatFs functions created by CubeMX. The only thing that you would need to do is to create FatFs and file objects to interact with the filesystem. I have pushed the code to a repository that contains both four bit and one bit SDIO which contains code that writes to an SD card. I will spend time over the summer updating these repositories to include timers and ADC functionality so that I will have a full-fledged project to port to my PCB next semester. Another word of advice is that once you get the four bit SDIO to work, there is no need to write additional code using all four data lines. Any code that you write for the one bit SDIO will work with four bits as well, so I recommend using the one bit SDIO to make the wiring a lot easier during debugging.

3.4 SPI

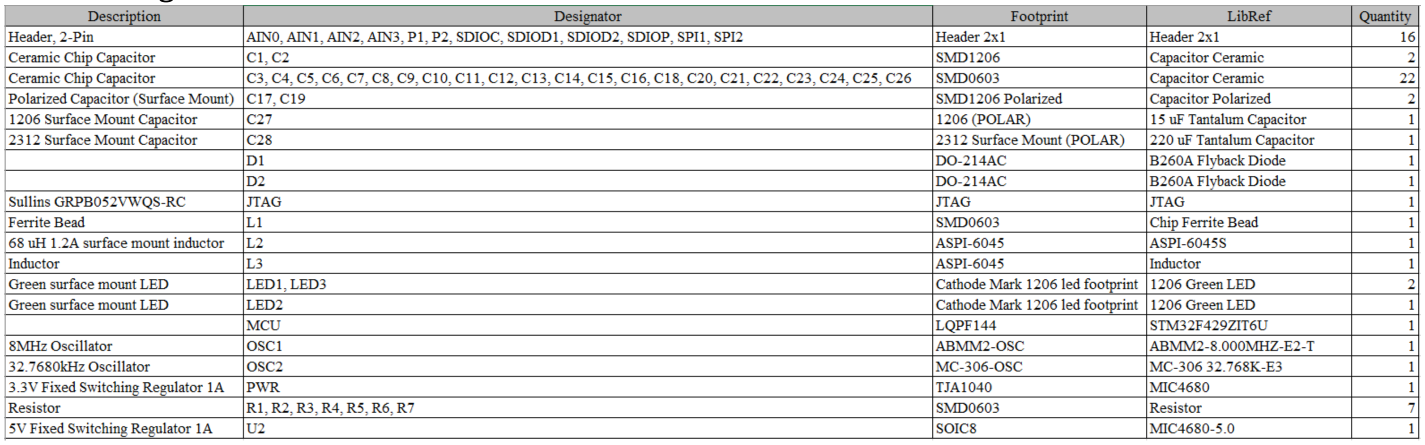
```
int main(void)

{

  HAL_Init();

  /* Configure the system clock */

  SystemClock_Config();

  /* Initialize all configured peripherals */

  MX_GPIO_Init();

  MX_SPI1_Init();


  uint8_t cmd_arg [ 6 ] ;

  uint32_t Count =  0xFFFF ;


  /** Send SPI message in Deselect state and put it in standby state. **/

  DESELECT ( ) ;


  for ( int i =  0 ; i <  10 ; i ++ )

  {

    SPI_TxByte ( 0xFF ) ;

  }


  /** SPI Chips Select **/

  SELECT ( ) ;


            uint8_t CMD0 = 0x40 + 0;

  /** Initial GO_IDLE_STATE state transition **/

  cmd_arg [ 0 ]  =  (CMD0|0x40) ;

  cmd_arg [ 1 ]  =  0 ;

  cmd_arg [ 2 ]  =  0 ;

  cmd_arg [ 3 ]  =  0 ;
cmd_arg [ 4 ]  =  0 ;
cmd_arg [ 5 ]  =  0x95 ;

 /** Send the command **/
 for  ( int i =  0 ; i <  6 ; i ++)
 {
   SPI_TxByte ( cmd_arg [ i ] ) ;
 }

 /** Wait for a response **/
 uint8_t data = SPI_RxByte();
 while  (data  !=  0x01 ) {
   HAL_GPIO_WritePin ( GPIOG , GPIO_PIN_14 ,  GPIO_PIN_SET ) ;
        HAL_GPIO_WritePin ( GPIOG , GPIO_PIN_13 ,  GPIO_PIN_SET ) ;
        data =  SPI_RxByte();
        }
 if (data == 0x01){
               HAL_GPIO_WritePin ( GPIOG , GPIO_PIN_14 ,  GPIO_PIN_SET ) ;
               HAL_GPIO_WritePin ( GPIOG , GPIO_PIN_13 ,  GPIO_PIN_RESET ) ;
        }
}
```

Image 2: Image 2 shows some of the logic used in the SPI acknowledgement code.

While playing around with the SDIO code generated by STM32CubeMX, I ran into an error and I decided to step through the code using the debugger on Keil uVision. I realized that because much of the code was generated by CubeMX, I only had a shaky understanding of what was going on and I figured that I needed a contingency plan for writing firmware for my PCB as my PCB is not going to be identical to the Discovery Board. Since there is not a lot of information on the internet about the SDIOMMC bus, I decided to make my contingency plan based off the SPI bus. I found a guide online that would help me write the SPI functionality with FatFs: https://blog.naver.com/eziya76/221188701172. The link is in Korean, but Google Translate works fairly well in translating the text.

The tutorial is written for an STM32F1 chip but the code for the STM32F4 MCU does not work any differently. To ensure that I can get SPI working, I patched together just the acknowledgement section of the SPI interaction which is shown above in Image 2. The code essentially shows 0x40, 0x0, 0x0, 0x0, 0x0, and 0x95 being repeatedly sent to the SD card with hopes of it returning a 0x01. If 0x01 is returned then two LEDs are lit up to indicate that the SD card has responded. I was able to get this code working and I have pushed it to a Git repository. I plan on fleshing out the rest of the SPI implementation working over the summer.

4 PCB Design



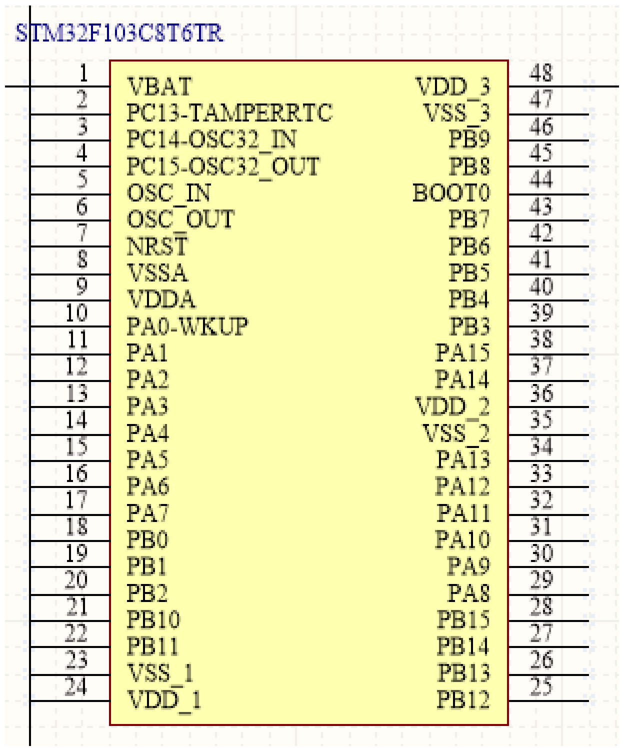
Description	Designator	Footprint	LibRef	Quantity
Header, 2-Pin	AIN0, AIN1, AIN2, AIN3, P1, P2, SDIOC, SDIOD1, SDIOD2, SDIOP, SPI1, SPI2	Header 2x1	Header 2x1	16
Ceramic Chip Capacitor	C1, C2	SMD1206	Capacitor Ceramic	2
Ceramic Chip Capacitor	C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16, C18, C20, C21, C22, C23, C24, C25, C26	SMD0603	Capacitor Ceramic	22
Polarized Capacitor (Surface Mount)	C17, C19	SMD1206 Polarized	Capacitor Polarized	2
1206 Surface Mount Capacitor	C27	1206 (POLAR)	15 uF Tantalum Capacitor	1
2312 Surface Mount Capacitor	C28	2312 Surface Mount (POLAR)	220 uF Tantalum Capacitor	1
	D1	DO-214AC	B260A Flyback Diode	1
	D2	DO-214AC	B260A Flyback Diode	1
Sullins GRPB052VWQS-RC	JTAG	JTAG	JTAG	1
Ferrite Bead	L1	SMD0603	Chip Ferrite Bead	1
68 uH 1.2A surface mount inductor	L2	ASPI-6045	ASPI-6045S	1
Inductor	L3	ASPI-6045	Inductor	1
Green surface mount LED	LED1, LED3	Cathode Mark 1206 led footprint	1206 Green LED	2
Green surface mount LED	LED2	Cathode Mark 1206 led footprint	1206 Green LED	1
	MCU	LQPF144	STM32F429ZIT6U	1
8MHz Oscillator	OSC1	ABMM2-OSC	ABMM2-8.000MHZ-E2-T	1
32.7680kHz Oscillator	OSC2	MC-306-OSC	MC-306 32.768K-E3	1
3.3V Fixed Switching Regulator 1A	PWR	TJA1040	MIC4680	1
Resistor	R1, R2, R3, R4, R5, R6, R7	SMD0603	Resistor	7
5V Fixed Switching Regulator 1A	U2	SOIC8	MIC4680-5.0	1

Table 1: Table 1 shows the bill of materials for this project.

4.1 Purpose

The purpose of designing my own custom PCB for this project was two-fold. Firstly, I wanted to design my own PCB because I would be able to create a relatively light weight PCB using only some of the ports of the MCU as opposed to the weighty Discovery Board. Furthermore, designing my own PCB will allow me to create a more natural incorporation with the sensors used on the team as well as a better incorporation into the Baja car. Secondly, I wanted to design a PCB because I wanted to gain exposure to designing PCBs by making my own on Alitum Designer. I wanted to be able to learn everything associated with the process of designing a PCB like schematic symbol creation, footprint creation, schematic capture, and footprint layout and I was able to do so during this semester. I was able to learn these skills from a host of online resources mainly including Robert Feranec's YouTube channel: https://www.youtube.com/channel/UCJQkHVpk3A8bgDmPlJlOJOA. I highly recommend following his tutorials if you want a through introduction to PCB design and Altium Designer. I have also relied on two board reviews and constructive criticism from my teammates to design my board.

4.2 Schematic Symbol and Footprint



STM32
F103C8T6
Z

Image 3 and 4: Images 3 and 4 show the schematic symbol and footprint for the STM32F103C8T6 MCU.

Prior to switching over to the STM32F4 MCU, I was pretty committed to using the forty-eight-pin package for the STM32F103C8T6, so I began simultaneously creating the schematic symbol and footprint for the STM32F103C8T6. I followed some of Robert Feranec's tutorial for generating the schematic symbol and linking it with the footprint. I was able to find a CAD model for the STM32F103C8T6 online, so I did not have to create my own. I highly recommend using Symbol Wizard for editing the schematic symbol and Footprint Wizard for creating the footprint as they saved me a lot of time that I had lost in the beginning. Although I did not end up using this MCU in my project, the effort I put in to create this component was worthwhile as it is presently in the Baja Integrated PCB library for future use. When I ended up using the same MCU as Jay Fetter did on the previous iteration of the DAQ, I no longer had to worry about making my own schematic symbol or footprint as they already existed in the Baja Integrated PCB library.

4.3 Schematic Capture

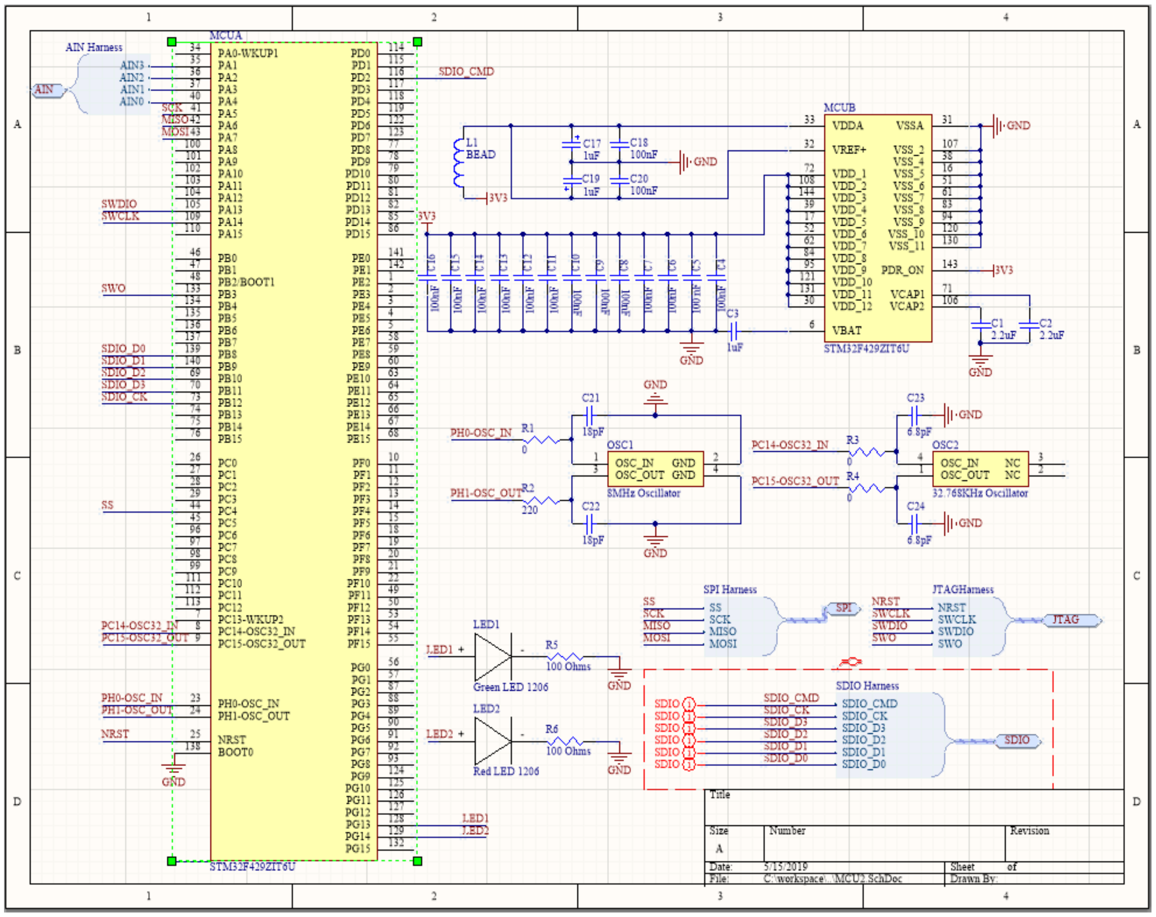


Figure 2: Figure 2 shows the schematic for the MCU.

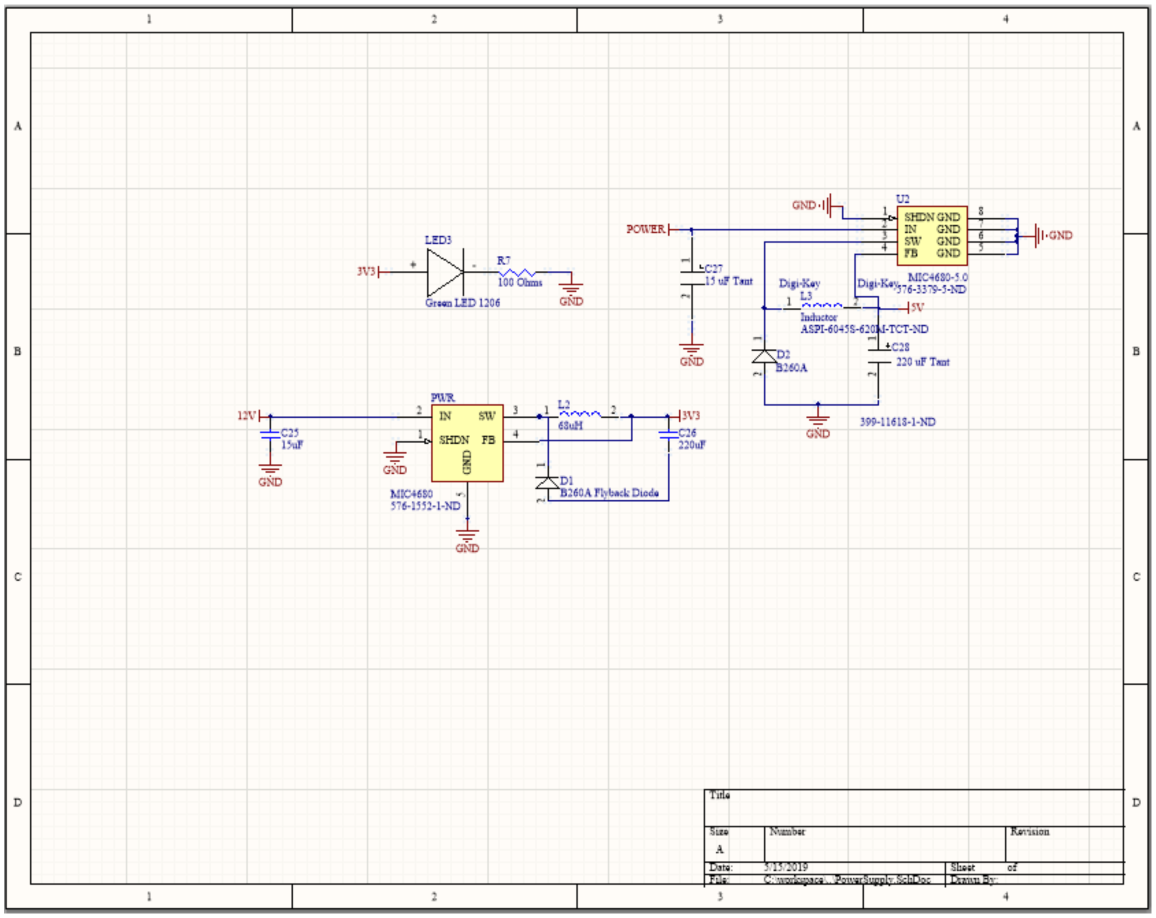


Figure 3: Figure 3 shows the schematic for the voltage regulators.

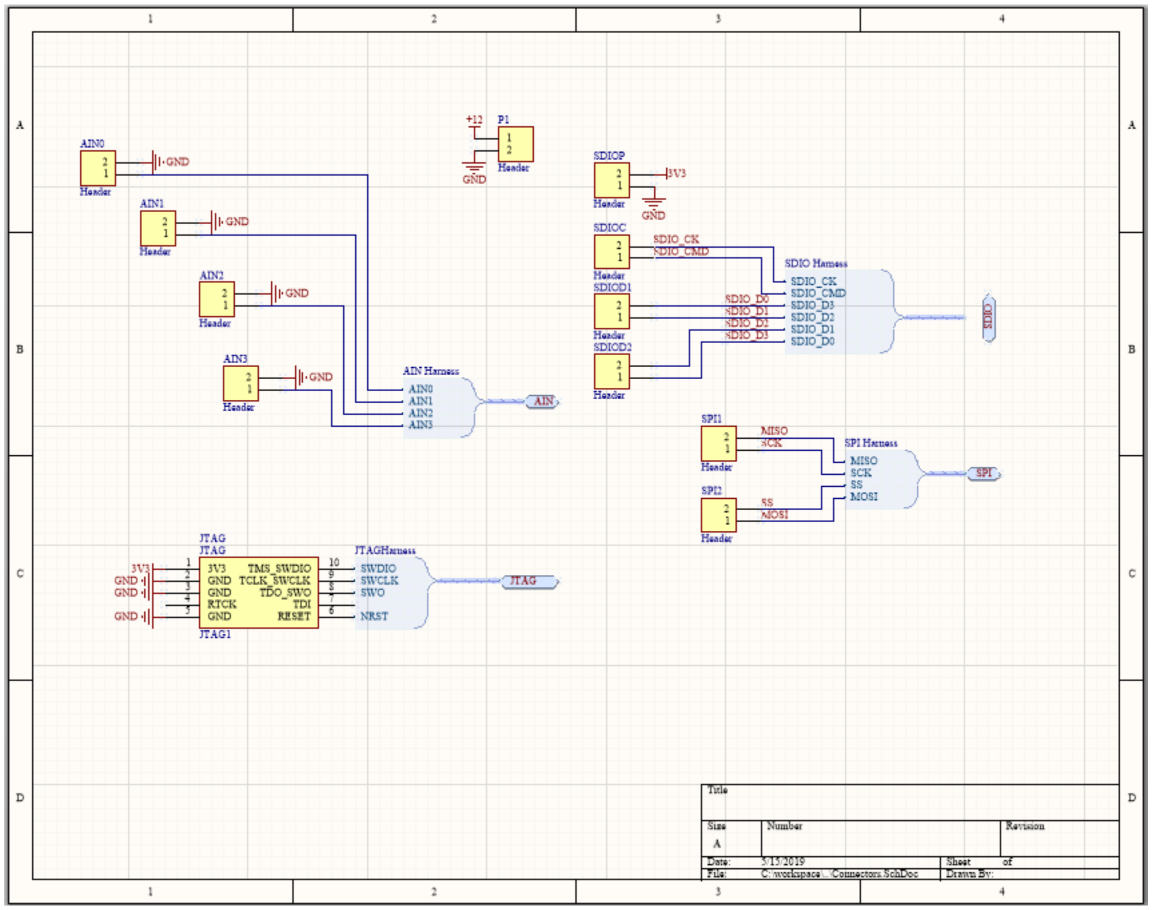


Figure 4: Figure 4 shows the schematic for the external connectors.

While doing the schematic capture for the PCB, I relied heavily on the Discovery Board's schematics and Jay Fetter's schematics for inspiration. I also relied on the hardware development manual for this MCU: https://www.st.com/content/ccc/resource/technical/document/application_note/76/f9/c8/10/8a/33/4b/f0/DM00115714.pdf/files/DM00115714.pdf/jcr:content/translations/en.DM00115714.pdf. I first started with the MCU schematic which is shown in Figure 2. The two biggest symbols are part of the same package and represent the various pins on the MCU. However, the smaller symbol and its various connections were taken from both the Discovery Board schematics and Jay Fetter's schematics, but the configuration is not mentioned anywhere in the hardware development manual. I will try to schedule a meeting with Jay Fetter over the summer to get better clarification on why he did this. I settled on an 8MHz crystal oscillator for the high frequency oscillator and a 32.768 kHz crystal oscillator for the low frequency oscillator, but I do not imagine ever needing to go up to 8MHz in my design. In addition, I broke out connection for the programmer tool, labeled SWxxx, SDIO, labele SDIOxxx, SPI, and two LEDs. I also broke out connections from four twelve-bit ADC channels labeled AIN0, AIN1, AIN2, and AIN3. I plan on using the ADC to continuous read data from these pins and write them to an array/buffer which I will ultimately end up writing to an SD card. The SDIO lines are placed in the same net class, SDIO Net Class, because of some additional requirements placed on the SDIO bus by the hardware development manual.

My next schematic was the voltage regulators used in the project. I decided to use two buck boosts from the same design line that will step the voltage down to either 3.3V/1A and 5V/1A. I am not too worried about the current draw out of the 3.3V buck boost as 3.3V is used almost everywhere on the PCB and I am fairly certain that all the components will run within their current specifications. The 5V buck boost is only being used to power the four external sensors, so the current draw will need to be more closely monitored to ensure that everything will run within specifications. My last schematic was the one showing all the connectors used on the board. This was the easiest schematic to make as I mainly plan on using header pins to communicate with the SD card and the sensors. The PCB also contains an addition two pin header for each of the analog inputs to connect from the 5V buck boost and power the sensor. I have also included the connections for the programmer in this schematic.

4.4 PCB Layout

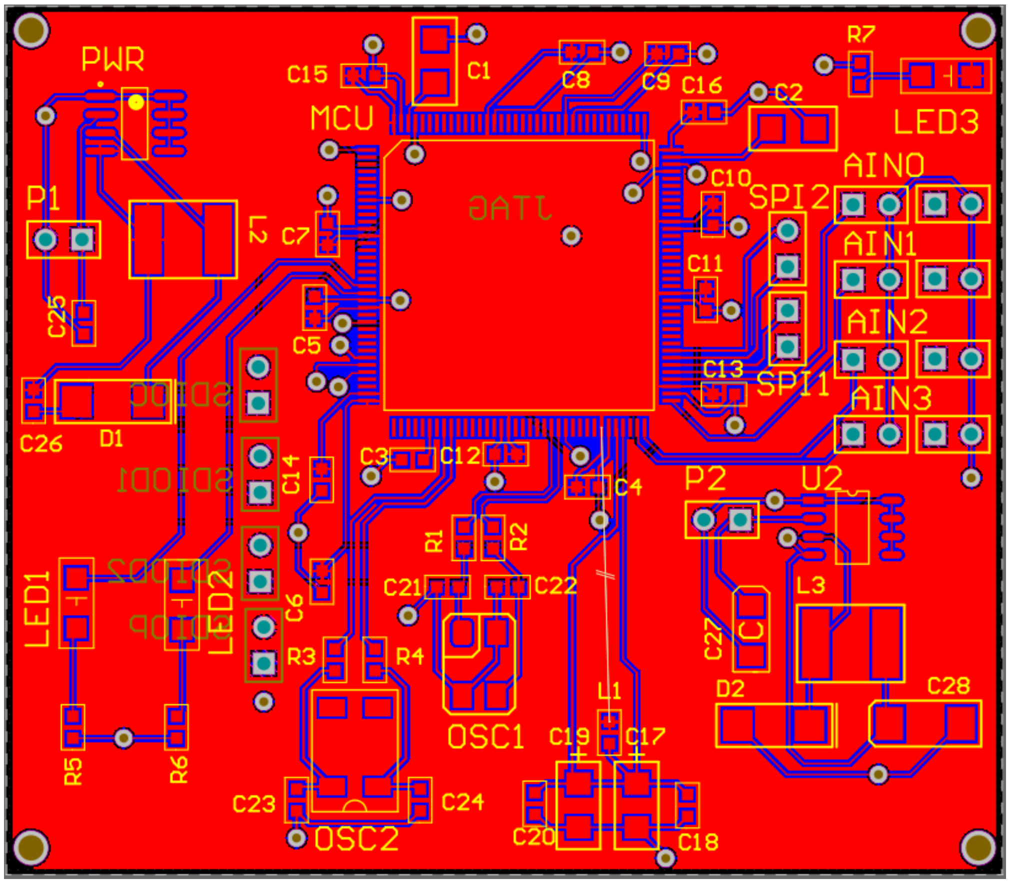


Image 5: Image 5 shows the top layer of the PCB. The red signifies a 3.3V plane.

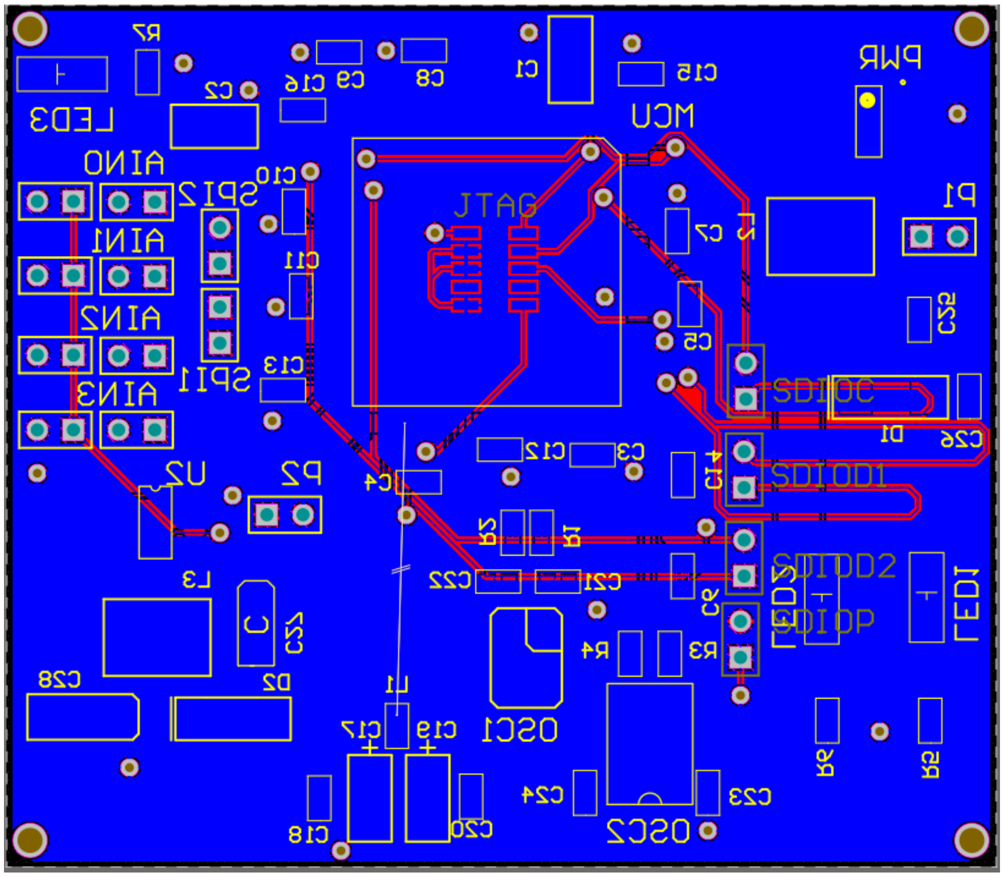


Image 6: Image 6 shows the bottom layer of the board. The blue signifies a ground plane.

PCB layout was by far the hardest part of the process and required careful retuning of the PCB. As can be seen in Image 5, the components are mostly evenly distributed on the front of the board except for the right side of the board. This congestion occurred simply because of the was the MCU package was laid out. The SPI and the AIN pins were unfortunately situated in the right side of the MCU as opposed to being more even spaced out. I did not want to route the headers too far from the respective pins on the MCU, and I especially did not want to do that for the SPI pins. On the bottom of the board I placed the oscillators as close together as possible and I tried to make the traces going in and out of the oscillator as close in length as possible. I placed the SDIO bus lines on the bottom of the board for two reasons. The first reason being that this is just the first revision of the board and in future versions I plan on adding the SD card holder directly onto the backside of the PCB. Secondly, the routing became much easier after I moved the SDIO bus to the bottom of the board. The hardware development guide also placed some strict guidelines on the trace lengths, impedance, capacitance, and inductance. Firstly, to meet the guideline that the trace lengths must be within 10mm off each other, I created a design rule for the SDIO Net Class. I then tried to use interactive length tuning to achieve the effect that I wanted but I was unconvinced that I would be able to maintain signal integrity, so I simply drew out traces until they matched the length I needed them to be. For the guideline stating that the trace impedance must be within ten percent of fifty ohms, I created another design rule that would change trace widths to achieve the right trace impedance. Finally, I checked the trace capacitance and inductance using online calculators to make sure that they were within specifications. In addition to placing the SDIO bus on the bottom of the board, I also placed the programmer of the MCU on the bottom layer. I also poured a 3.3V polygon over the top layer and ground polygon over the bottom layer. I also added four large vias on each corner of the board to use as mounting holes.

4.4 Future Work

The next step I would like to take is have one final board review with Jay Fetter. I would like to do this because Jay Fetter used the same MCU on his DAQ so I would like to know if there are any pitfalls I need to look out for or if I had done something wrong. I would then like to get this board printed out so that I can populate it and test it early next semester. In the meantime I plan on writing more of the firmware using the Discovery Board so that I can make sure I have a fully functioning firmware project ready to flash the PCB. After that, I would like to take the next steps more carefully. I want to meet with the mechanical engineers on the team, show them my project, and then ask them for any suggestions they might have for me to fix/make on future revisions.

5 Reflection

This last semester has been by far the busiest and most meaningful semester on Baja. I came back early for JanFab to help machine parts and build the car, and after that I have worked diligently on my project. I have learned more in this semester of Baja than I have in any other semester and that can be attributed to taking on a difficult project and meeting my goals for the semester. In the future, I hope to continue doing similar types of work for Baja, but my focus in the future is sorely placed on finishing this board up. I would really like to get this board working next semester and I hope to use the guidance and help from my teammates to get this project done. Next semester, I also hope to go out testing with the team more often so that I can see the needs of the mechanical engineers more closely. I want to do this to have a better understanding of what I will need to do in future semesters during my time on Baja.