

**Advanced Communications Breakout Board
Chrysler Group, LLC
Design Team 5
ECE 480**



CHRYSLER

Team Members:
Richard Hendrick – Leader
Meng Cao – Lab Coordinator
Andrew Haumersen – Presentation
Sana Siddique – Web Designer
Xuran An – Documentation

Facilitator: Dr. Jian Ren

Executive Summary

Chrysler currently does not have an advanced systemic approach to check for connectivity and bus communication with prototype car modules. The current method consists of having all of the modules plugged into one heap on the OBD II box and requires digging through this heap physically when an engineer needs to toggle a module on/off or take a voltage reading. As a result of this inconvenience, the process of module testing is tedious and takes much longer than it should which directly affects time to production. The goal of this project hence was to develop an advanced breakout box that will simplify the current procedure to get CAN voltage and module current readings by giving each module its own port on the new breakout board where it can be plugged in and its own connection to a communication bus that can be toggled and controlled by an easy-to-use graphical user interface. The graphical user interface is also able to change what CAN bus the module is connected to and take user input for the voltage supply. Using this device the engineer will not need to physically connect and disconnect the modules. The GUI also stores the CAN and module readings in an excel spreadsheet for every session. The new board has to interface with the existing OBD II box and hence requires multiplexing so that signals are not corrupted. We were able to successfully build the board and fulfill the requirements of the project and produce display readings with 1-5% tolerance.

Acknowledgement

We would like to thank the staff at Chrysler for giving us this opportunity, especially Brian Podczervinski for his advice, assistance and guidance throughout the project. We would also like to thank Dr. Jian Ren, the group's facilitator, for keeping the team on track and overlooking the teams progress from time to time. Finally, we would like to thank Roxanne Peacock for her assistance in ordering parts for the project.

Table of Contents

1 Introduction and Background	5
1.1 Introduction	5
1.2 Background	5
2 Solution Analysis and Approach	8
2.1 Design Specification	8
2.2 FAST Diagram	11
2.3 Conceptual Design Descriptions	11
2.4 Selection of Components	13
2.5 Proposed Design Solution	15
2.6 Proposed Schedule (Gantt)	17
3 Technical Description of the Project	19
3.1 Hardware design efforts	19
3.2 Hardware Implementation	21
3.3 Software and Interface Design Requirements	22
3.4 Software Implementation	23
4 Testing and Results	26
4.1 Testing	26
4.2 Result	28
5 Total Cost, Schedule, and Conclusion	29
5.1 Cost	29
5.2 Future Enhancements	31
5.3 Schedule	32
5.4 Conclusion	32
Appendix 1 Team Responsibilities	33
Appendix 2 References	38
Appendix 3 Detailed Technical Attachments	40

1 | Introduction and Background

1.1 | Introduction

Chrysler is an American automobile manufacturer which was founded by Walter Chrysler in 1925 and is headquartered in Auburn Hills, Michigan. It is one of the “Big Three” American automobile manufacturers. It is a well-known fact that Michigan is home to many other automobile companies. As the economy recovers and demands for new and advanced cars increase, it is essential for Chrysler to refurbish their testing process to keep up with these demands and produce models with new and better technology in a timely fashion, before other competitors. This particular project targets the testing of car modules at Chrysler.

Our goal for this project is to shorten the time to test modules by designing an advanced breakout board which will connect to the OBD II box and the Beaglebone Black microcontroller. The microcontroller will then communicate with the GUI program on the computer to display module and CAN bus readings. This device completely eliminates the need to remove and reconnect modules to the OBD II box. The board would allow engineers to use their laptops to control which modules are active and inactive on a bench-top communication bus. This device will decrease the amount of time during early development work, thus, allowing the engineers to identify issues with module connectivity prior to the installation on the prototype.

The advanced breakout box needs to handle the modules by successfully supplying sufficient power for any individual unit. The design will also be providing each module with stable communication buses that will not interfere with each other or pick up any outside noise from surrounding circuitry. All of these controls, along with voltage readings and current readings taken from the communications bus, are displayed on the graphical user interface. This data is also stored in an excel spreadsheet when the application is running which allows the engineer to analyze it at a later time.

1.2 | Background

It is important and necessary to test connectivity and communication of modules on a pre-prototype vehicle. On-board diagnostics, or OBD, is an automotive term referring to a vehicle's self-diagnostic and reporting capability. Various vehicle sub-system status are determined by the vehicle owner or repair technician using OBD systems.

The OBD-II standard specifies the type of diagnostic connector and its pin-out, the electrical signaling protocols available, and the messaging format. It also provides a candidate list of vehicle parameters to monitor along with how to encode the data for each. There is a pin in the connector that provides power for the scan tool from the vehicle battery, which eliminates the need to connect a scan tool to a power source separately.

This OBD-II comes in two models OBD-IIA and OBD-IIB. OBD-II standardization was prompted by emissions requirements. Although only emission-related codes and data are required to be transmitted through it, most manufacturers have made the OBD-II Data Link Connector the only one in the vehicle through which all systems are diagnosed and programmed.

OBD-II uses CAN bus to communicate with different modules on the vehicle. CAN bus is a serial communication bus. The frequency of the CAN bus can go as high 1 MHz. For Chrysler, two frequency standards are used, one is 500kHz, and the other is 250kHz. CAN bus operating voltage varies. Usually, the maximum voltage will be lower than 7V.

The picture below shows the current method of module testing at Chrysler.

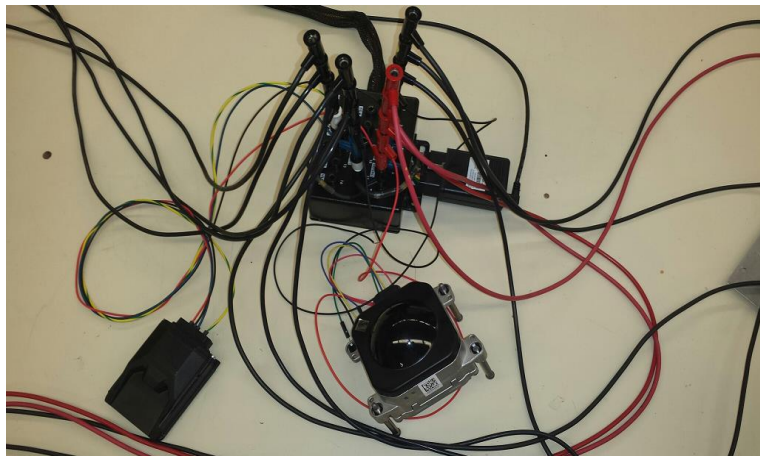


Figure 1.1

As one can tell, testing this way would involve connecting individual modules to the OBD II box using banana wires. To connect modules to the same bus the banana wires will need to be stacked on top of each other. Hence toggling modules on and off and changing bus will require the engineer to dig through a bunch of banana wires which makes this entire process of testing extremely slow. More importantly, there is no automated way of storing this data on to a database.

It is therefore essential to develop an advanced breakout board to significantly reduce the time needed to test modules. The module can be toggled on and off and can be connected to different CAN bus using this device. All the data from each session of the GUI application will be stored in an excel spreadsheet and is available to the engineer for later analysis.

2 | Solution Analysis and Approach

2.1 | Design Specification

After talking to Chrysler we were able to get a good idea about the Critical Customer Requirements (CCR) for our design. In order to meet all the criteria Chrysler has given us our breakout box must perform a number of basic functions. Our design must be able to support ten different modules, such as ABS, ECU, Headlights, HVAC, etc., being plugged in at the same time and connected to CAN busses via an OBDII port. These modules must be toggled on and off individually not only with battery power but also with the ignition signal in order to test for function, startup processes, and connectivity. The board should be able to handle up to 4 high current modules. These two different current connections will be split into having four high current connections which are able to draw up to 15 amps and five low current connections which can draw up to 250 milliamps. Another difficulty Chrysler faces is storing voltage and current readings of both power draw, and voltage on the CAN buses. To solve this problem our breakout box will be capable of displaying and saving real time current values of the power drawn by all ten individual modules as well as six different voltage readings from the CAN bus. These readings must all be accurate within plus or minus five percent. Another important test function that our breakout box is required to provide is the ability to change any module to communicate on any CAN bus simultaneously.

This will help engineers to see how certain modules behave on the three different can buses and also help to isolate any module that is not functioning properly or two modules that are interfering with each other's data signals. One important part of this function that we must meet is that we do not induce any signals or corrupt any signals on the CAN buses. This is achieved in our design through CAN multiplexing.

Since our breakout box will provide so many different functions the need to have an easy to use interface or way to display the information is a must. To accomplish this our breakout box can easily plug into a computer via usb and has an easy to use GUI that will provide all the necessary functions including: toggle power and ignition signals, display and record real time voltage and current readings, and change which can bus any individual module is communicating on. Lastly, the box has to be somewhat portable; it will be used in a test bench setting but must be able to be carried from bench to bench to perform different testing applications.

The table below is a breakdown of what we believe is the importance of each design parameter to the customer.

Design Parameter	Importance to Customer
Toggle power	5
Draw 15 amps(high current module)	5
Voltage/Current readings	5
Accuracy of readings(+/-5%)	5
Distribution of CAN signals	5
Integrity of CAN signals	5
Save data readings	5
Connection of 10 modules	4
Easy to use GUI	4
Mobility of breakout box	3
Low cost	1

We believe that each criteria listed above must be satisfied in order to make the design feasible and the customer happy. The cost of the device was of least importance to Chrysler although the budget provided for the project is 500 dollars. According to Chrysler development of this board will bring down the testing process time by at least 2 weeks.

The ultimate objective of our design is to make Chrysler's test procedure for new modules easier and less time consuming in order to speed up the process and help to reduce the time it takes a new vehicle to go from prototyping to production. Below is a

diagram of where our breakout box fits into the test procedures.

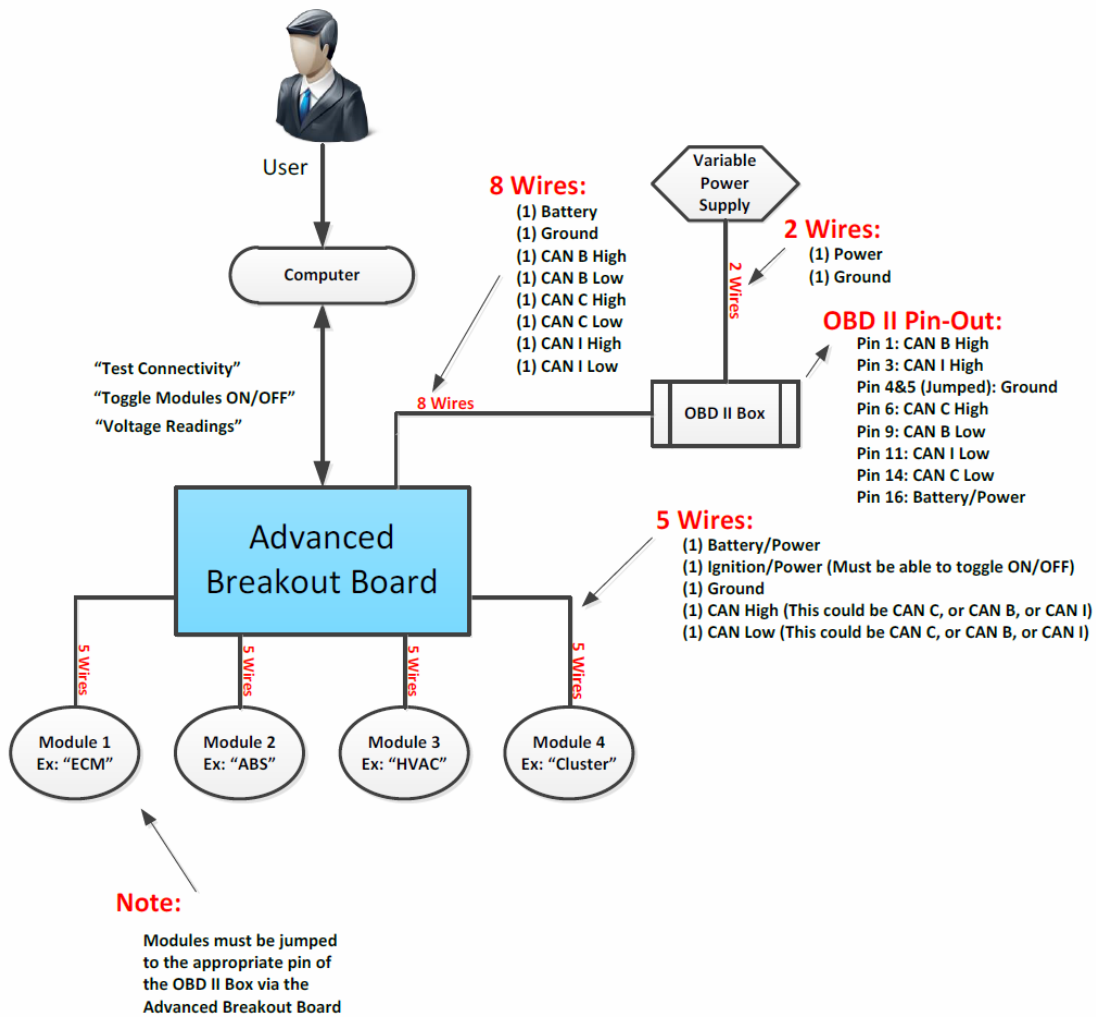


Figure 2.1

2.2 | FAST Diagram

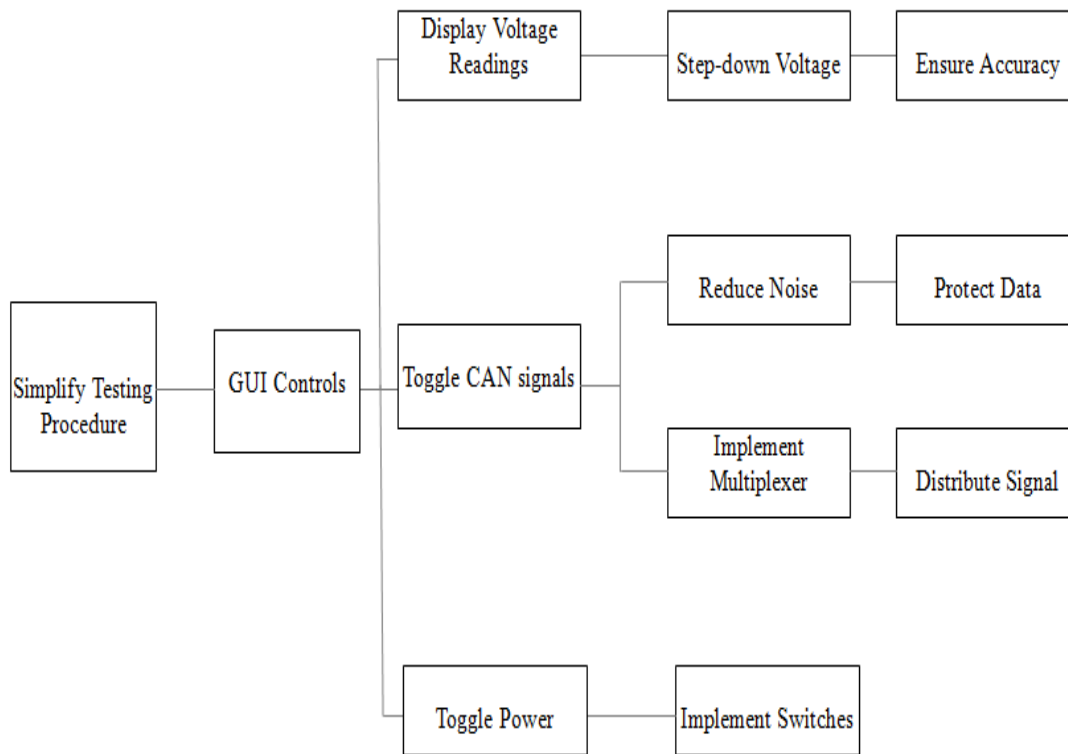


Figure 2.2

2.3 | Conceptual Design Descriptions

In order to achieve the requirement above, we have several options. The design can be divided into six different parts, microcontroller, A/D converter, multiplexer, switches, PCB design, and GUI.

Microcontroller

This project requires toggling of modules using laptop, and sending data back to the laptop, microcontroller is essential to accomplish this. Rather than stand-alone chips, we decided to go with microcontroller development board that is already designed with protection circuit. It will be easier to program with, as well as convenient for communication with the computer. Three different type of low-cost microcontroller boards were considered for this design, Arduino, Raspberry Pi, and Beaglebone Black.

Arduino/Raspberry Pi/Beaglebone Black

A/D converter

None of the microcontrollers have the sufficient A/D converter. Therefore, additional A/D converters are required. For this design, also for prototype build, all the semiconductor chips need to be in DIP packet. Two chips were considered for this project, MCP3208 from MicroChip and ADS7822P.

MicroChip MCP3208 8-Channel 12-Bit Resolution A/D converter

Texas Instrument ADS7822P 4-Channel 12-Bit Resolution A/D converter

Multiplexer

Multiplexer also needs to be in DIP packet. The multiplexer is required to be at least 3 to 1. So 4 to 1 Mux is the multiplexer that best fits the requirement. Also, in order to make PCB look cleaner, dual 4x1 Multiplexer is the best choice.

Texas Instrument CD74HC4052E Dual 4x1 Multiplexer

VISHAY DG409DJ-E3 Dual 4x1 Multiplexer

Transistor

Transistors are used to toggle module on and off. Chrysler wants to send toggle signal to all the modules and at same time be able to turn voltage supply on and off for each module individually. In order to do that, transistors will be used as switches. As we are using transistor to cut off voltage source, we choose the PNP type transistor. For the high power modules, the current can go as high as 15A, the base current could be 150mA with a gain of 100 for the transistor. In order to reduce the current flow from the microcontroller, we decided to use another NPN transistor to control the PNP transistor for high power board.

PCB design

In this project, extra PCB design are required for the multiplexer chips and A/D converter chips. The extra PCB needs to be able to communicate with the microcontroller board.

GUI

For the GUI we considered two options- GTK+ toolkit and QT toolkit. GTK+ and Qt are open-source cross-platform user interface toolkits and development frameworks. These are very popular frameworks for Linux because they are open-source and give developers a powerful toolkit to design GUI. Qt has C++ based libraries and it supports Java, Perl, Python, PHP and Ruby based development. GTK+ has C based libraries. It supports several languages, including C++, Java, Perl, PHP, Ruby and JavaScript.

2.4 | Selection of Components

Microcontroller

	Arduino Uno	Raspberry Pi A/B	Beaglebone Black
CPU clock	16MHz	700MHz	1 GHz
On board memory	32 KB	256MB/512MB RAM	512MB RAM/ 2GB Storage
GPIO	14	8	46
A/D Converter	6	none	7
USB port	none	1/2	1
Supply Voltage	6-20V	5V	5V
Other		HDMI,Ethernet,SD Slot	HDMI,Ethernet,SD Slot
Price	\$27	\$25/\$35	\$45

	CPU	Memory	GPIO	A/D	USB	Supply Voltage	Price	Total
weight	1	2	5	4	2	2	4	25*5
Arduino	2	1	2	2	0	5	5	52
Raspberry Pi	4	3	1	0	5	2	5	44
Beaglebone Black	5	5	5	2	4	2	4	76

After comparison, Beaglebone Black fit the project best. For this project, the GPIO, A/D converter were the most important requirements, and BeagleBone Black clearly wins on both of these aspects. Beaglebone also has the fastest CPU speed and largest RAM, as well as it has large on board memory that can be used to record experiment's data. The price is a little higher than other microcontrollers, but it is most suitable for the project .

A/D converter

	MCP3208	ADS7822P
Manufacturer	MicroChip	Texas Instrument
Resolution	12-bit	12-bit
Channel	8	4
Packet and Pins	DIP-16	DIP-14
Price	3.48	4.16

	Resolution	Channel	Price	total
weight	5	5	5	15*5
MCP3208	4	5	5	70
ADS7822P	4	3	4	55

The resolution of the two chips are the same. The difference is the number of channels that can measure for each chip. MCP3208 can measure 8 different channel as compared to 4 different channel on ADS7822. This will save a lot of space on the PCB. Also, the price is lower for MCP3208, considering it can measure 8 channel on a single chip, this is much cheaper than ADS7822P.

Multiplexer

	CD74HC4052E	DG409DJ-E3
Manufacturer	Texas Instrument	VISHAY
Channel	Dual 4x1	Dual 4x1
Packet and Pins	DIP-16	DIP-16
Price	\$0.46	\$3.60

For multiplexer in the project, we need to toggle module to 3 different CAN buses. Each CAN bus has CAN high and CAN lower. For ten modules, that means 20 different multiplexers. So we decided to use Dual 4x1 multiplexer chip, which can toggle CAN high and CAN low at the same time. Also reduce the chip from 20 to 10 piece. These two multiplexer chips are basically the same. CD74HC4052E from TI which has the unbeatable price, so it becomes the first choice.

Transistor

For transistor used in this project, we picked three different transistors TIP36, TIP31 and TIP32. Both TIP32 and TIP36 are PNP type transistor, which will be connected directly in series with module and power supply. TIP31 is a NPN type transistor which is used to control TIP36.

PCB Design

The same problem arises with PCB. The device need to power both low power module and high power module. The high current on PCB will produce noise. If something wrong happens, high current may also blow up the chips. So we decided to use separate PCB for high current and low current.

GUI

As mentioned before, the top 2 choices for developing GUI were using GTK+ toolkit or QT toolkit. Qt can run directly on the hardware, without the need of X11 or a window manager whereas GTK+ applications require X11 server and window manager to run which means that a simple hello world application on GTK+ will have 3 processes running which will slow down the overall application. QT is also faster with some widget. Hence, QT seems to be the superior choice and we will be using python to program the GUI. QT has python bindings available as well. we will be using PyQT to build the GUI.

On selecting these components we were also very confident that the budget of our project will be well below \$500 dollars.

2.5 | Proposed Design Solution

To achieve all the features for a functional Advanced breakout board our microcontroller- the Beaglebone Black will need to run a GUI application that will allow the user to toggle modules on and off and display the current values for the modules and the CAN bus voltages in a timely fashion. It should also be able to transfer this data to a spreadsheet for each session of module testing. This will be accomplished by using the QT toolkit as mentioned earlier. Qt is a complete consistent framework. You can easily connect HTTP events to GUI elements, fill forms with results from a database query or build an interactive visualization of large datasets. Some of the notable QT 5

modules that will be used for this project are QtCore, QtGui and QtScript. The module voltages and CAN bus voltages will be displayed on the GUI using the A/D converters and the General Purpose Input Output ports on the BeagleBone. The BeagleBone will be using Ubuntu operating system to accomplish the above tasks.

To be able to connect any of the module to any CAN bus voltages while keeping intact the integrity of the CAN voltages we will have to do multiplexing and then send the data to the laptop. The multiplexing done makes it very simple for the user to select which module goes on what CAN bus.

4 module inputs out of 10 need to be able to handle high amounts of current ranging from 7A to 15A while the others will have a maximum current reading of 250mA. To accomplish this, we have split up our board into having four high current connections and five low current connections instead of making all ten able to handle the high current. We have chosen this method because it will save us money in production of our PCB board, our power switching components, and it will make our design compact as we won't have large traces on all of the PCB connections. The PCB board is going to be multi layered as well because this will make the device more portable and very easy to transport. Please refer to the appendix for actual PCB design.

2.6 | Proposed Schedule (Gantt)

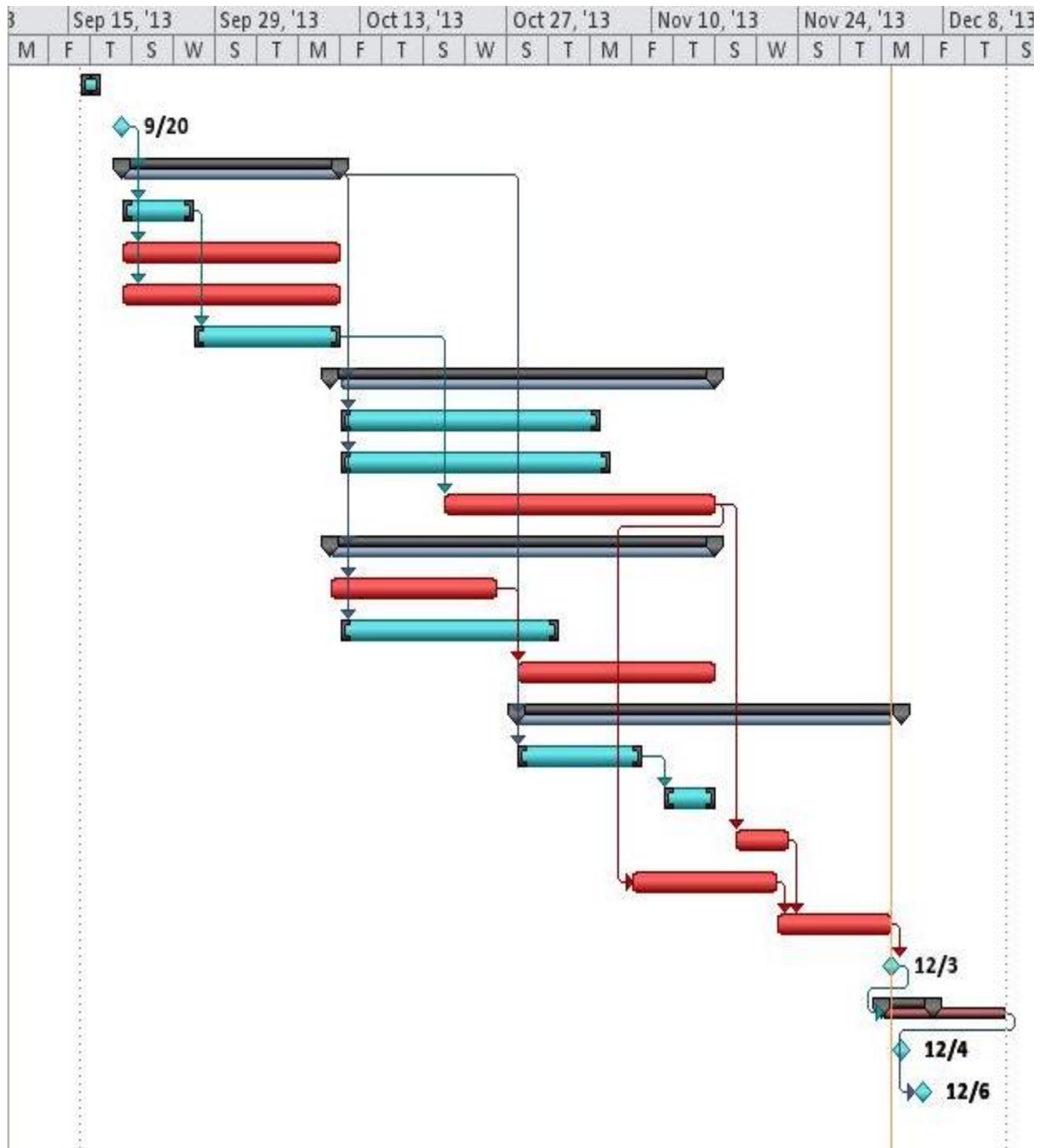


Figure 2.2

	Task Mode	Task Name	Duration	Start	Finish	Predecessors
		Pre-Proposal	2 days	Mon 9/16/13	Tue 9/17/13	
		First Meeting with Chrysler	0 days	Fri 9/20/13	Fri 9/20/13	
		 Determine Parts and Topology	15 days	Fri 9/20/13	Thu 10/10/13	2
		Choose Microprocessor	5 days	Fri 9/20/13	Thu 9/26/13	2
		Determine Sensor Topology	15 days	Fri 9/20/13	Thu 10/10/13	2
		Determine Module topology	15 days	Fri 9/20/13	Thu 10/10/13	2
		Determine Program language	10 days	Fri 9/27/13	Thu 10/10/13	4
		 Software Coding	27 days	Thu 10/10/13	Fri 11/15/13	3
		Hardware Level Coding	17 days	Fri 10/11/13	Mon 11/4/13	3
		Beaglebone Coding	18 days	Fri 10/11/13	Tue 11/5/13	3
		GUI coding	20 days	Mon 10/21/13	Fri 11/15/13	7
		 Hardware Design	27 days	Thu 10/10/13	Fri 11/15/13	3
		Schematic	12 days	Thu 10/10/13	Fri 10/25/13	3
		Build Prototype	15 days	Fri 10/11/13	Thu 10/31/13	3
		Layout PCB	15 days	Mon 10/28/13	Fri 11/15/13	13
		 Tesing	27 days	Mon 10/28/13	Tue 12/3/13	3
		Testing Individual Module	10 days	Mon 10/28/13	Fri 11/8/13	3
		Testing Beaglebone with Sensor	5 days	Mon 11/11/13	Fri 11/15/13	17
		Testing USB Communication	5 days	Mon 11/18/13	Fri 11/22/13	11
		Testing GUI	10 days	Fri 11/8/13	Thu 11/21/13	11
		Testing Prototype	7 days	Fri 11/22/13	Mon 12/2/13	19,20
		Prototype Demonstration	0 days	Tue 12/3/13	Tue 12/3/13	21
		 Finalize the Project	5 days	Mon 12/2/13	Fri 12/6/13	22
		Final Report (ECE480)	0 days	Wed 12/4/13	Wed 12/4/13	24
		Design Day	0 days	Fri 12/6/13	Fri 12/6/13	23

Figure 2.3

In order to organize our design effort efficiently, we created a Gantt chart (Fig. 2.2-2.3) showing the tentative schedule we decided upon as a group. By allocating a known time for each step of the hardware and software design process, we anticipated we would be able to coordinate our efforts and work in parallel on various components of the design. By the end of the design process, we had deviated from our proposed schedule by a few days because some tasks required more time to be completed. The shipment of PCB board other part held back our progress slightly. The design of the PCB also had to be changed along the way to make it better and more compact. Overall, we followed the Gantt chart as closely as we possibly could.

3 | Technical Description of the Project

3.1 | Hardware design efforts

3.1.1 | PCB Design

In order to handle the large number of modules that need to be connected to the three different CAN buses at any time, real time current and voltage readings, and toggling of power and ignition on and off a PCB was our only route to keep the design clean and condensed. Eagle was the software chosen to create our schematic which then ultimately was transferred to a board where traces could be run. The original design included a single large schematic of both the control circuit, low, and high power circuits which was then switched into three separate boards in order to keep our cost as low as possible, as one large board cost over \$600. In designing the schematic every component needed for our circuit had to first be chosen, and then a model found for our circuit. Some of these were not already in a library so one had to be built from scratch and uploaded into the library for use in our schematic. After all of our components were chosen and added to the schematic everything had to be wired together, by creating wire segments and naming all of our connections it made the schematic much easier to navigate, edit, and troubleshoot. See figure below. Detail schematic attached in Appendix 3.

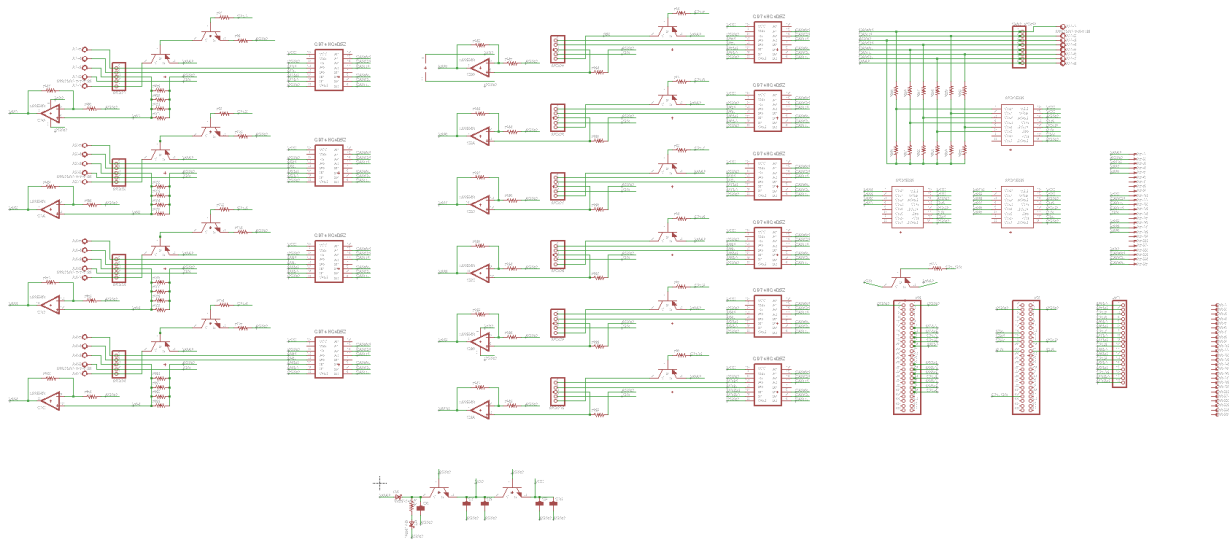


Figure 3.1.1

Once the schematic was finalized a board could be created and set up to run traces. Upon creating a board all of the components from the schematic needed to be placed strategically where they would allow for easy traces to be run and where they

would be located in the final product. This required a lot of trial and error with many slight modifications and three major redesigns of our circuit layout. In total our circuit includes ten separate five pin pin outs for each module, eight pins for OBDII connection, five dual channel multiplexers (one per two modules) for CAN bus distribution, thirty pins for control bit connection to the BeagleBone to control multiplexers and toggle power, three A/D converters for voltage readings, 14 transistors for toggling power to each module (two per high power and one per low power module), one transistor for toggling ignition, two transistors for our power circuit, five capacitors for our power circuit, three four channel amplifier chips for current readings, and seventy resistors. All these components are split up over three separate smaller boards and all connected between the boards with d-sub. Once the final layout was selected and every trace was run the boards were ready to be submitted, the final design uses a two layer board for our control circuit (Figure A3.6 in Appendix 3) and four layer boards for both high and low power modules (Figure A3.2 and Figure A3.4 in Appendix 3).

3.1.2 | Multiplexer

One issue we ran into with needing distribution to three different CAN buses both high and low totaling to sixty different connections was the number of control bits needed to control our multiplexers. This was cut in half to twenty control bits by using dual four channel multiplexers (CD4052BE) which allowed both the high and low connection for each CAN bus to be controlled with the same control bits since you would never connect to the high of one CAN bus and the low of another. Each module needs its own multiplexer totaling to ten multiplexers for our entire design.

3.1.3 | Current and Voltage Sensor

This project is required to measure the voltage of different CAN buses, which is from 0V to 7V. The power of each module also needed to be measured. Power can not be measured directly, so current are measured to calculate the power. Analog-to-Digital converter chip can convert different analog voltage levels to 12-bit digital decimal number. Another resistor are added in to convert current into voltage for measurement using Analog-to-Digital Chip.

3.1.4 | Microcontroller

This projects needs a controller to communicate with different Analog-to-Digital Converter chips, Multiplexer chips. That requires the microcontroller to have multiple GPIO. In this project, 3 I/O needed for each module, and 9 GPIO needed total for communication between the microcontroller and Analog-to-Digital Converter chip. In addition, the microcontroller needs to be able to communicate with a laptop, taking command and sending datas.

3.1.5 | Other

All modules not only need to be toggled to different CAN buses, they also need to be turned on/off. CAN bus system running on a voltage level of 12V, while most microcontroller running with a voltage of 3.3V. Multiplexer chips also require 12V for the voltage supply and control bits to toggle voltage signal under 12V range.

The devices in the project also need different voltage levels. The ADC chips need 3.3 volts. The multiplexer chips need 12 volts. For the op-amps, 5V are used to have enough power to boost voltage to 3.3V and set a 5V voltage limit for safety.

3.2 | Hardware Implementation

3.2.1 | PCB

The PCB design is used to connect four high power and six low power modules all to three separate CAN buses on the OBDII at any time, toggle power for any individual module, toggle ignition for all modules at the same time, and take real time voltage and current readings for each module and CAN bus. The BeagleBone is connected with thirty control bits and nine data bits and ultimately displays all this information on an easy to use GUI.

3.2.2 | Analog-to-Digital Converter (MSP3208)

In order to measure up to 16 measurement, three analog-to-digital converter chips are used, each of them measures 8 different channels. The ADC chips are distributed by the measurements, one chip measuring all the CAN bus voltages, one chip measuring four high power module and the other one measuring the current of six low power module.

The ADC chip can measure from 0V to the supply voltage, which is 3.3V with 12-bit resolution. For the CAN bus voltage measurement, the range is set to 0V to 12V. The purpose of measuring voltage of the CAN bus is to detect failures in the module. If a failure happens, the bad module will drop down the voltage or pull up the voltage for the whole CAN bus. That is why the range of measurement is set to 0V to 12V, slightly larger than CAN bus operating voltage.

For the current measurement, resistors are also needed to convert current into voltage, since the ADC chips can only convert voltage into digital signal. Resistor are designed with small resistance values, to avoid taking too much power. However, that also reduces the voltage crosses that resistor. In order to fully use the resolution, an op-amp added to step up the voltage for each module.

The ADC chips communicate with the beaglebone black using SPI bus. SPI is a serial communication bus. It requires only 3 wires per bus with 1 extra wire for chip

select. For three chips, two SPI buses are used, result in 9 wires total for SPI communication. The chip measuring CAN bus voltage on one SPI bus, and two chips measure module current on the other SPI bus.

3.2.3 | Voltage Shifter (CD40109BE)

In order to boost the GPIO voltage of the Beagbone Black, voltage shifter are used to shift the voltage level up to a proper voltage. In this project, the voltage shifter boost voltage from 3.3V to 12V, with 3.3V and 12V voltage supply.

3.2.4 | Microcontroller (Beaglebone Black)

Beaglebone Black is chosen to be the microcontroller. Beaglebone has the most GPIO compare to Raspberry Pi and other microcontroller board, with more than 40 GPIO available. 30 GPIOs are used for toggling modules on/off and to different CAN bus. The additional GPIO can be used for future extension. 20 of the 30 are connected to the control bits of the multiplexer, going throw voltage shifter to shift voltage level up.

Beaglebone Black has different build-in communication buses, such as UART, I2C, and SPI. It also has USB, ethernet, that provides different communications with computer.

Beaglebone Black is running an embedded linux system onboard, so the programming language can vary.

3.3 | Software and Interface Design Requirements

The BeagleBone is required to run a program that is capable of reading module and CAN bus data. This program on the board will then interact with the GUI on the laptop.

Beaglebone Black does not have built-in drivers for Serial Peripheral Interface (SPI). So SPI driver needs to be added into the embedded linux system. Other than that, for using python as the programming language, libraries for GPIO and SPI need to be provided.

GUI is an essential part of this project because the main purpose of this product is to eliminate the need to physically connect and disconnect the modules to the OBD II board. The main requirement of the project was to build a GUI which displays the current readings of the modules and CAN bus voltages. In addition to this, the GUI also should allow the user to toggle the modules on and off and select what CAN bus they are connected to. Another important requirement is to take the readings from the modules and CAN bus routinely and store it in a database or a spreadsheet.

3.4 | Software Implementation

3.4.1| Communication with GPIO pins

All the code development for this project has been done in python because of the wide range of libraries and extensive documentation available for this language. The program that will run on the board and takes readings from the modules is called Initial.py. In order to properly communicate with the GPIO pins that are available on the BeagleBone Black, we had to install and utilize a GPIO library called Adafruit_BBIO. This library allowed us to setup and toggle pins high or low, depending on what results we were looking for from our circuit. The program that runs on the BeagleBone itself takes the input from the GUI running on the computer and executes the commands that the GUI requests. For example, the input string that the BeagleBone code looks for is in the format of “0/1/2’ ‘Module Number’ ‘Command” where ‘0/1/2’ represents a digit of 0, 1, or 2, ‘Module Number’ is which module the user has selected for change, and ‘Command’ could be any of the CAN buses (B, C, I) or “TRUE”/“FALSE”. The TRUE/FALSE values are sent whenever a module needs to be turned on or off, “TRUE” for on and ‘FALSE” for off.

3.4.2 | Serial Peripheral Interface

Since the linux system has no build-in Serial Peripheral Interface(SPI) drivers, SPI driver needs to be added. The actual driver file are in the appendix. Different drivers are needed for different SPI buses. For example, the driver for SPI1 need to specify there are two different chips, with SPI0 there is only one chip. The finished driver attached in the appendix.

After the SPI drivers are ready, a SPI library for python is also needed. For using SPI in python, the library needs to be written in C and then imported to python. We were able to find a library called SPImodule which prints the received data. Changes had to be made to this library to return the value to the python program (Initial.py), not print it out. There are three different functions in the final library, SPI.transfer, SPI.open, and SPI.close. SPI.open is the function to connect to some certain device. SPI.close is used to close the connection with that device. SPI.transfer is the actual transfer function, so that the program on the Beaglebone can take those values and send those to the GUI on the computer. The finished library file attached in the appendix.

3.4.3 | Graphical User Interface

To accomplish the graphical user interface requirements we had to install PyQt5 which comes with a QT designer application which makes designing the GUI very

simple. *Qt Designer* is Trolltech's tool for designing and building graphical user interfaces (GUIs) from Qt components. It allows designing and building of widgets and dialogs using on-screen forms using the same widgets that will be used in the application. Components created with *Qt Designer* can also take advantage of Qt's signals and slots, and they can be previewed so that you can ensure that they will look and feel exactly as you intended. This application generates a .ui file which has to be converted to a .py file. In the python file I have to implement functions for any activity that takes place on the GUI i.e., every possible kind of click on the GUI has to be translated by a function (slot) and take action accordingly. Below is a picture of the GUI application that was built on the QT designer.

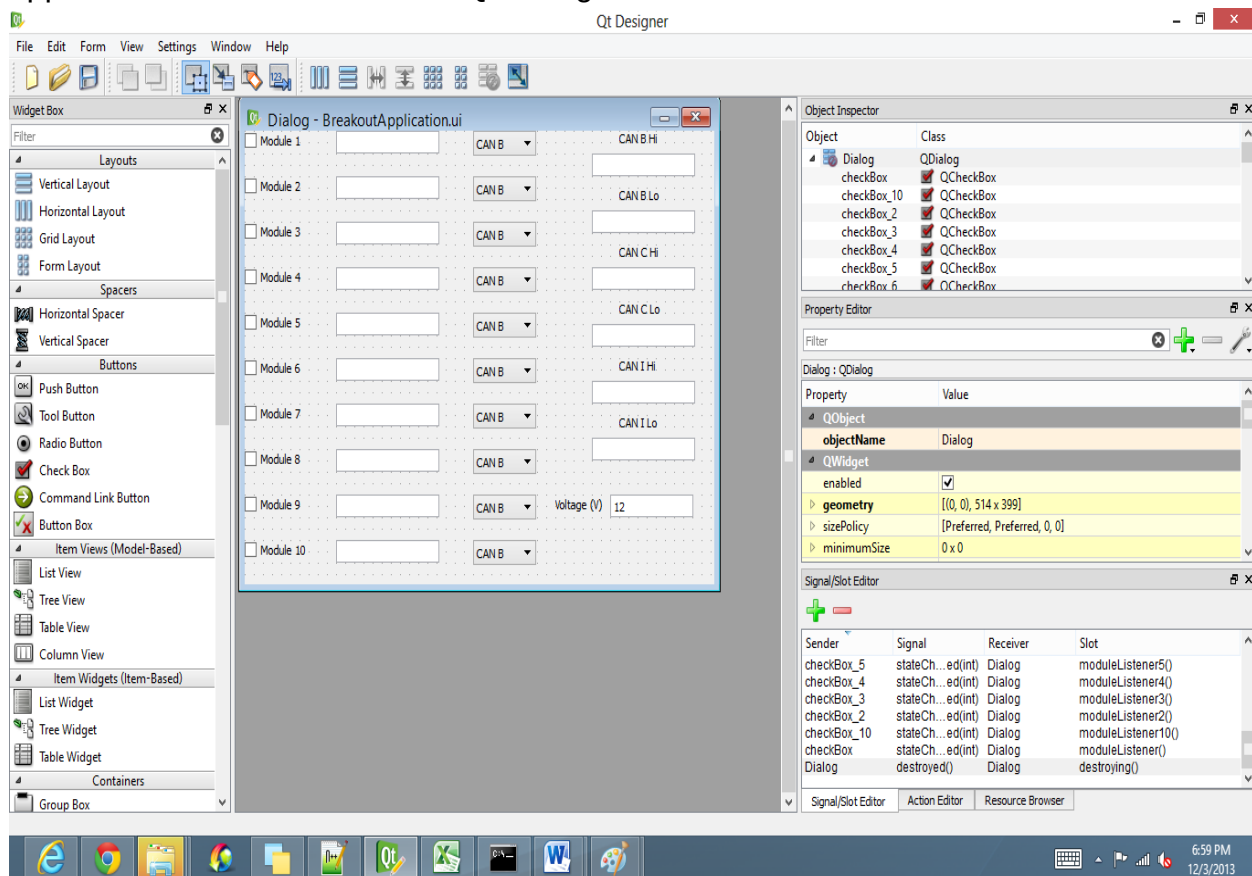


Figure 3.2

Since another main requirement of the project is to display the values in real time, I retrieve the current and voltage data from the program Initial.py which is running on the Beaglebone Black every 0.5 seconds using multithreading and timer. A new thread is generated which calls a function to read data values from the microcontroller every 0.5 seconds. Every time the GUI reads the data from the microcontroller it also stores it in the excel spreadsheet. A continuous graph of module current and CAN voltages can then be easily generated by the spreadsheet data. The module values returned to the GUI from the board are all voltages which need to be converted to current before

displaying. This is done using the ohms law since the resistance values are decided by us and remain fixed. Testing of the GUI application was mostly independent of the hardware part of the project. We just had to make sure that it was able to receive the values from Initial.py program running on the board and display it correctly on the user interface. We also had to determine the format of input and output for convenient parsing of data and make sure that it was able to send correct data to Initial.py program. The library used for establishing serial COM connection with the BeagleBone was pyserial. The program is able to auto-detect the COM port that is being used by the Beaglebone. The command to run Initial.py program on the board is given by the GUI program after establishing the serial connection. To export the data to a spreadsheet I used `xlsxwriter` library of python. We were able to successfully generate an executable for the GUI along with all of its dynamic-link library(dll) files. Hence the application is a stand-alone application and will not require the user to download other dependencies and have python installed to run it.

The most important issues faced while developing the GUI program was being able to kill all the threads that were being created by the program (2 per second). This was necessary to not have the memory run out and also to make sure that the serial connection was being closed after every thread has accessed the required data from the board. If any thread wasn't killed it would keep the serial port engaged even after exiting the application. The only way to free the COM port would be to restart the machine. This is something that was fixed successfully in the program by handling multithreading carefully and by creating a new class that inherits the QT dialog box class and re-implements the function that is accessed on exiting the application so that it closes the serial port.

4 | Testing and Results

4.1 | Testing

Once we received and built all three of our PCBs it was time for testing, the initial test was connecting to power and checking that every chip and module was receiving the correct voltage.

After testing voltage a different signal was connected to all three CAN buses to simulate each one and all of the control bit combinations were selected, looking at the output of each module to check that it was displaying the correct signal that corresponds with each simulated CAN bus. Our next area to test was voltage and current readings, each CAN bus was supplied with a known voltage and the output supplied to the GUI was checked for accuracy. To test our current readings just like our voltage a known current was supplied to each module and again tested for accuracy.

The remaining hardware needed for testing was provided by Chrysler, which includes a number of different prototype modules that resemble the kinds that they currently test every day. They also provided an OBDII scanner tool which will be used to check the CAN buses at any time to validate the measurements.

The software testing was mostly independent of the hardware testing. To test initial .py we had to make sure it was getting the right information from the board. Once the program was written and it was time for testing, the initial debugging steps started with just using a multimeter to measure the output voltage of different GPIO pins as different commands were issued from the program. When this was all working, a single low power circuit was built and connected to the necessary pins to check operation of the different multiplexing chips along with toggling the connection on and off. This testing procedure was consistently used as more circuits were built and added to the final design. Finally, we had to make sure the GUI and the program running on the board were communicating correctly, i.e., the GUI is able to display the values that the program on the board is trying to send. We also had to check if the program on the board was receiving signals from the GUI correctly and handling it using the right functions. Figure 4.1 below is a screenshot of putty showing the data send from Beaglebone Black to the GUI. Figure 4.2 shows the data being received by the GUI and displayed correctly. This screenshot was taken prior to adding calculations in GUI to convert the module voltage values received to module current.

```
192.168.7.2 - PuTTY
login as: root
root@192.168.7.2's password:
root@beaglebone:~# python Initial.py
can't open deviceAborted
root@beaglebone:~# echo BB-SPI0-01 > /sys/devices/bone_capemgr.*/slots
root@beaglebone:~# echo BB-SPI1-01 > /sys/devices/bone_capemgr.*/slots
root@beaglebone:~# python Initial.py
0 0 0
['3012', '1025', '3013', '1025', '3012', '1026']
['3013', '1024', '3014', '1024', '3012', '1026']
['3014', '1024', '3011', '1026']
```

Figure 4.1

Dialog

☒ Module 1

3014

CAN B

CAN B Hi

3012

☒ Module 2

1024

CAN I

CAN B Lo

1025

☒ Module 3

3011

CAN B

CAN C Hi

3013

☒ Module 4

1026

CAN B

CAN C Lo

1025

☒ Module 5

3013

CAN I

CAN I Hi

3012

☒ Module 6

1024

CAN B

CAN I Lo

1026

☒ Module 7

3014

CAN C

Voltage (V)

12

☒ Module 8

1024

CAN B

Port

☒ Module 9

3012

CAN B

☒ Module 10

1026

CAN B

Figure 4.2

The first list displayed on putty consist of the raw voltage data of the six wires of the three different CAN bus. The second list contains the values for low current modules and the third list has the values for high current modules. Please note that modules 1-4 are high current modules in the GUI.

4.2 | Result

After all of our testing was completed we had a successful working breakout board which completed all of our initial requirements. Up to ten modules, four high power and six low power, can be connected and toggled between all three CAN buses with no corruption, both power and ignition can be individually toggled, nearly real-time current readings of each module and voltage readings of all three can buses are displayed in an easy to use GUI. The data was also successfully saved on a spreadsheet. The data that we saved includes the module current and the CAN bus connection for every module. In addition we also save all the CAN voltage values during the session. This fulfills all the requirements that were given to us by Chrysler.

5 | Total Cost, Schedule, and Conclusion

5.1 | Cost

As with any prototype, most of the initial designs were revamped later on in the design process. There were some design flaws with the first design, such as choosing a multiplexer that ran off too low of a voltage. This is why there were over 20 multiplexer chips purchased. After fixing this flaw, we were able to go on and implement the need for voltage shifters. Since with the increased supply voltage, VCC, to the multiplexers, we are required to have the voltages for the selection bits to be equivalent to VCC in order for the multiplexers to operate properly. The voltage shifters take the 3.3V signal voltages from the BeagleBone GPIO pins and produces a 12V toggleable signal. The tables below show both the initial cost for creating the first prototype, and the reproduction cost* that removes any unnecessary or redundant purchases that are not necessary to recreate the final working design.

<u>Item</u>	<u>Quantity</u>	<u>Total (\$)</u>
12-Bit A/D Converter	3	10.44
Connectivity DIP Socket	15	5.51
Quad Voltage Level Shifter	8	1.58
Analog 20V Multiplexer/Demultiplexer	12	5.72
Analog 9V Multiplexer/Demultiplexer	12	5.28
Male/Male, Female/Male, Female/Female Jumper Cable	1	7.79
25 Pin D-SUB Connector, RCPT	2	2.16
25 Pin D-SUB Connector, PLUG	2	5.52
5 Pin Terminal Block, 28- 12AWG	4	3.06
2 Pin Terminal Block, 26-	2	0.50

18AWG		
8 Pin Terminal Block, 24-12AWG	2	3.44
0.01 Ohm Resistor	25	2.10
1 Row 10-Pin Header	4	0.54
1 Row 2-Pin Header	10	0.23
TIP36A Transistor	4	8.56
Control PCB	1	69.69
High Power PCB	1	102.69
Low Power PCB	1	102.69
BeagleBone Black	2	90.00
Total		427.50

Table 5.1: Initial Production Cost Per Unit

<u>Item</u>	<u>Quantity</u>	<u>Total (\$)</u>
12-Bit A/D Converter	3	10.44
Connectivity DIP Socket	15	2.85
Quad Voltage Level Shifter	7	1.37
Analog Multiplexer/Demultiplexer	10	4.70
Male/Male, Female/Male, Female/Female Jumper Cable	1	7.79
25 Pin D-SUB Connector, RCPT	2	2.16
25 Pin D-SUB Connector, PLUG	2	5.52
5 Pin Terminal Block, 28-	4	3.06

12AWG		
2 Pin Terminal Block, 26-18AWG	2	0.50
8 Pin Terminal Block, 24-12AWG	2	3.44
0.01 Ohm Resistor	25	2.10
1 Row 10-Pin Header	4	0.54
1 Row 2-Pin Header	10	0.23
TIP36A Transistor	4	8.56
Control PCB	1	33.00
Voltage Shifter PCB	1	33.00
High Power PCB	1	66.00
Low Power PCB	1	66.00
BeagleBone Black	1	45.00
Total		296.26

Table 5.2: Reproduction Cost

*Note: Since shipping prices vary and can depend on outside factors such as the vendor, delivery time and any possible pre-existing contracts, any and all shipping charges have been removed from reproduction cost calculations.

5.2 | Future Enhancements

The main concern we wish we could have addressed is creating a proper enclosure for the entire breakout circuit. For now, everything is out and exposed to “work bench” elements that could cause damage over time. Also, for reproduction purposes, the voltage shifter and control boards could be designed and implemented together on the same board to save money and reduce the size of the overall footprint. This would also decrease the cost of a future case or enclosure, since the overall size would be smaller. Another small change a future design could have, if the voltage shifter board were to be integrated into the control board, the connections between the BeagleBone and this new board could be rearranged so that the pins lined up perfectly with the ports on the microcontroller, thus reducing the extra wires and connection

points. Less wires and connection points will result in a lower chance of something coming loose and becoming a point of failure.

5.3 | Schedule

To kick off our project, each member was assigned a different aspect of hardware that would need to be used to implement the prototype with the necessary requirements. The main focus points were multiplexing, analog to digital conversions, graphical user interface development, required number of GPIO pins, and different available microcontrollers on the market. These elements were researched and different ideas were formed and presented to Chrysler to get their input and to fully understand the requirements. After going over the requirements in depth, parts were chosen and ordered to begin implementation. Over the course of the semester, different designs were tested and improved both in the hardware and the software. We were able to stick to the Gantt chart up until a few weeks before the end of the semester, when a few last minute changes needed to be made and new parts needed to be ordered. After this minor setback, however, we were still able to accomplish Chrysler's requirements from the beginning of the semester.

5.4 | Conclusion

We were presented with a design project from Chrysler to help improve their "system" for testing pre-prototype electronic control units. To achieve the desired outcome, we first needed to know where to start and what they were currently working with. After understanding what they wanted out of our design, we were able to develop a device that not only meets their initial requirements, but goes above and beyond them. Our design has the complete capability to not only be reproduced easily but can also be expanded to handle more than ten modules at a time. Unfortunately the negatives by far outweigh the positives in this situation, as the addition of more modules would result in needing more PCBs and additional hardware that would end up making the footprint of a single unit bigger than two modules are at the current size.

Appendix 1 | Team Responsibilities

Richard Hendrick: Team Manager



Richard's main technical role was managing and allocating the BeagleBone's GPIO pins and writing the code for the program that runs and operates on the microcontroller itself. This program, Initial.py, needed to be able to take in input from the GUI that runs on the user's PC and return the appropriate values to the GUI and make the necessary changes to the different parts of the circuit. This program was also responsible for being able to take voltage readings as fast as the initial project requirements asked for. There was also extensive research that Richard looked into that would be needed for being able to support multiple external analog to digital converter chips all while not slowing down any of the communication that the microcontroller was handling between the circuit and the GUI. Richard also helped with getting the PCBs completed and submitted in a timely manner to help meet deadlines. He also played a helping hand in troubleshooting and debugging both the PCBs as they came in, but also the connections between the boards and the microcontroller, which made actual testing of the circuit and overall design much more user friendly when it came to checking different readings and possible failure points. Whenever any parts were needed off spare PCB circuits or chips needed to be removed and transferred to new PCBs, Richard was able to get everything removed and prepped for its intended use without holding up any other member of the group who may have needed any of these extra parts. Whenever there were waiting periods or down time for parts, Richard was always in the lab either working on the next document that the team had due or making sure that the team stayed on task with the Gantt chart created at the beginning of the semester.

Sana Siddique: Webmaster



Sana's main technical role in the project was to do research and implement the Graphical User Interface. She did extensive research on what resources are currently available to implement a user-friendly GUI. In particular, the two main packages that the research was narrowed down to are the GTK+ toolkit and QT toolkit. Both of these are open-source, cross-platform, user interface toolkits and development frameworks.

Based on her research she came to the conclusion that using QT toolkit for GUI development is the better choice due to the requirements of the project. Qt can run directly on the hardware, without the need of X11 or a window manager whereas GTK+ applications require X11 server and window manager to run and since the project requires the application to be as "plug and play" as possible, her decision to choose QT was indeed the right one.

Although QT has mainly C++ libraries and most of its documentation is in C++, Sana's expertise and prior experience in programming made it very simple to switch from C++ and develop the GUI in python. It was again her decision to use python for GUI development instead of C++ because during her research she had found useful python libraries like `xlswriter` (to write to excel spreadsheet), `cxFreeze` (to generate executable from python file) and `pySerial` (to establish serial COM connection with the board).

Sana was able to make the application run close to real-time by implementing multi-threading for reading of data from the `Initial.py` file running on the board. Sana also decided what format would be used to send data from GUI to `Initial.py`, and from `Initial.py` to GUI so as to minimize data parsing in the programs.

Besides this, Sana assisted the group in deciding which microcontroller would be best for the purposes of this project. She also did research on trying to implement the GUI on the Microcontroller itself but came to the conclusion that it's not a very feasible solution as it would overheat the BeagleBone. Lastly, Sana was entirely responsible for creating and maintaining the website for her team.

Meng Cao: Lab-Coordinator



Meng's technical role is working on the Analog-to-Digital Converter Chip and the communication between the chip and Beaglebone Black (our control board). He did research on different current and voltage sensing method, picked the Analog-to-Digital Converter chip, and designed the sensing circuit for current and voltage measurement. In order to save more GPIO on the control board for other purpose, he choose a chip that uses SPI bus. The chip is 12-bit resolution, for normal type, it requires 12 GPIO (parallel communication) plus 4 GPIO (identification bits for 16 channel). As using SPI

bus, it only takes 4 GPIO (SPI Bus 0) plus 5 GPIO (SPI Bus 1) saves a lot of GPIOs for other purposes, such as multiplexing.

Also, in order to using more than one chips on a single SPI bus, Meng had to write driver file, since there is no driver file ready to be used. The drivers he found can only support one chip for each SPI bus. By editing the driver, the Beaglebone Black can access 2 chips for each SPI bus. For this project, we picked python as the primary programming language to reduce the amount of work on programming. He found a library for using SPI bus in python, but the library crashed when using one of the function. Another library he found only prints out the receiving data, but does not return the data back to the program. So he had to rewrite the SPI library for python. That library is written in C, so he made the library be able to return data by editing some of the code, and recompiled it.

Other than that, He picked every component this project need, such as multiplexer, voltage shifter. He also draw the schematic for the design, worked on finalize the PCB design, helped on the programming on the control board and communication between Beaglebone board and laptop, as well as debugging some of the code.

Andrew Haumersen: Presentation



Andrew's main technical role was determining the multiplexing circuits, picking a multiplexer for CAN bus distribution, complete design of the schematic and PCB layout including wiring traces, and assembly of circuits onto PCBs. My original task was to figure out how to design a circuit to handle all of the multiplexing for making up sixty different connections, part of this task also included researching which multiplexer to use and all the specifications of that multiplexer. After this circuit was completed we had picked almost all of the components for the rest of the

PCB which I helped to determine some of the resistor values and voltage circuit components. With all these components chosen I had to create the first main schematic for the PCB layout. This first included one main schematic with all the modules and control circuits in one, after the cost was determined to be too high I had to redesign into three individual schematics for the high power, lower power, and control PCBs. After everything was named in the schematic I had to take it over to a board and decide our final layout of all the components and how they would interconnect, be powered, and receive their signal bits. Once I had determined the layout I created all the traces on where all the wiring for the PCBs would go. I initially had three four layer boards in order to connect everything but to minimize cost reduced the control board to only two layers and connected each with D-sub's. Once these boards were submitted I was in charge of soldering all of the components onto the boards once they arrived and ultimately connecting and testing for functionality. Another task I completed was designing, creating, and wiring up our voltage shifter PCB which was used to raise the voltage on all of our control bits.

Xuran An: Document



Xuran's main technical role was doing the research about ATD converter, CAN bus, and the data transferring information used in GUI. He also help to write code for transferring data to spreadsheet. We need ADC converter chip for measuring voltage. Based on the research, we can use two kinds ATD chip which are MCP3208 and ADS7822. The MCP3208 is programmable to provide eight single-ended inputs. Another one only provide four inputs.

MCP3208 can save space when designing PCB, so it is the first choice for us. He also found PIN function table used for implementing these chips. CAN bus allows microcontrollers and devices to communicate with each other. The communication relies on a voltage differential between CAN high and CAN low. In this case, we can use ATD converter to measure the voltage of them in order to check whether the module is better or not. For Chrysler requirements, we need to transfer the data after testing to an excel spreadsheet. Based on research and with group member help, there are several libraries we can use for writing excel spreadsheet, such as OpenPyXL and Xlsx Writer. We choose Xlsx Writer at last, because it is a little bit simpler when writing formatting data code than the other one. Our spreadsheet format is that all the names are in the A1 to Z1. The names are ten modules, ten module bus and CAN buses. We store data in B1 to Z1. He also helping group preparing poster, presentations and reports.

Appendix 2 | References

"Using the Adafruit_BBIO Library Technologies" A. L. (n.d.). Retrieved November 4, 2013, from Adafruit Learning System: <http://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/using-the-bbio-library>

"BeagleBone Black." *Community Supported Open Hardware Computers for Making*. N.p., n.d. Web. 04 Dec. 2013 retrieved online from <http://beagleboard.org/Products/BeagleBone%20Black>

"CD4052BE Datasheet" *Texas Instrument* N.p., n.d. Web. 04 Dec. 2013 retrieved from: <http://www.ti.com/lit/ds/symlink/cd4052b.pdf>

"Beaglebone Black Enable SPIDEV" *eLinux.org* N.p., n.d. Web. 04 Dec. 2013 retrieved from: http://elinux.org/BeagleBone_Black_Enable_SPIDEV

"SPI module.c" *GitHub* N.p., n.d. Web. 04 Dec. 2013 retrieved from: <https://github.com/thomasmacpherson/piface/blob/master/python/piface/spimodule.c>

"Python controlling SPI Bus on Beagleboard XM, and Raspberry Pi" *eLinux.org* N.p., n.d. Web. 04 Dec. 2013 retrieved from: <http://www.brianhensley.net/2012/02/python-controlling-spi-bus-on.html>

"MCP3208 Datasheet" *Microchip* N.p., n.d. Web. 04 Dec. 2013 retrieved from: <http://ww1.microchip.com/downloads/en/DeviceDoc/21298c.pdf>

"Qt Designer Manual." *Qt Project*. N.p., n.d. Web. 04 Dec. 2013 retrieved from: <http://qt-project.org/doc/qt-4.8/designer-manual.html>

"Using Qt Designer." *Using Qt Designer — PyQt 5.1.1 Reference Guide*. N.p., n.d. Web. 04 Dec. 2013 retrieved from: <http://pyqt.sourceforge.net/Docs/PyQt5/designer.html>

"GTK vs Qt." - *WikiVS*. N.p., n.d. Web. 04 Dec. 2013 retrieved from: https://www.wikivs.com/wiki/GTK_vs_Qt

"Welcome to PySerial's Documentation." *Welcome to PySerial's Documentation — PySerial 2.7 Documentation*. N.p., n.d. Web. 04 Dec. 2013 retrieved from: <http://pyserial.sourceforge.net/>

"XlsxWriter 0.5.1 : Python Package Index." *XlsxWriter 0.5.1 : Python Package Index*. N.p., n.d. Web. 04 Dec. 2013 retrieved from: <https://pypi.python.org/pypi/XlsxWriter>

"Cx_Freeze." *Cx_Freeze*. N.p., n.d. Web. 04 Dec. 2013 retrieved from: <http://cx-freeze.sourceforge.net/>

"Top Questions." *Stack Overflow*. N.p., n.d. Web. 04 Dec. 2013 retrieved from: www.stackoverflow.com

Appendix 3 | Detailed Technical Attachments

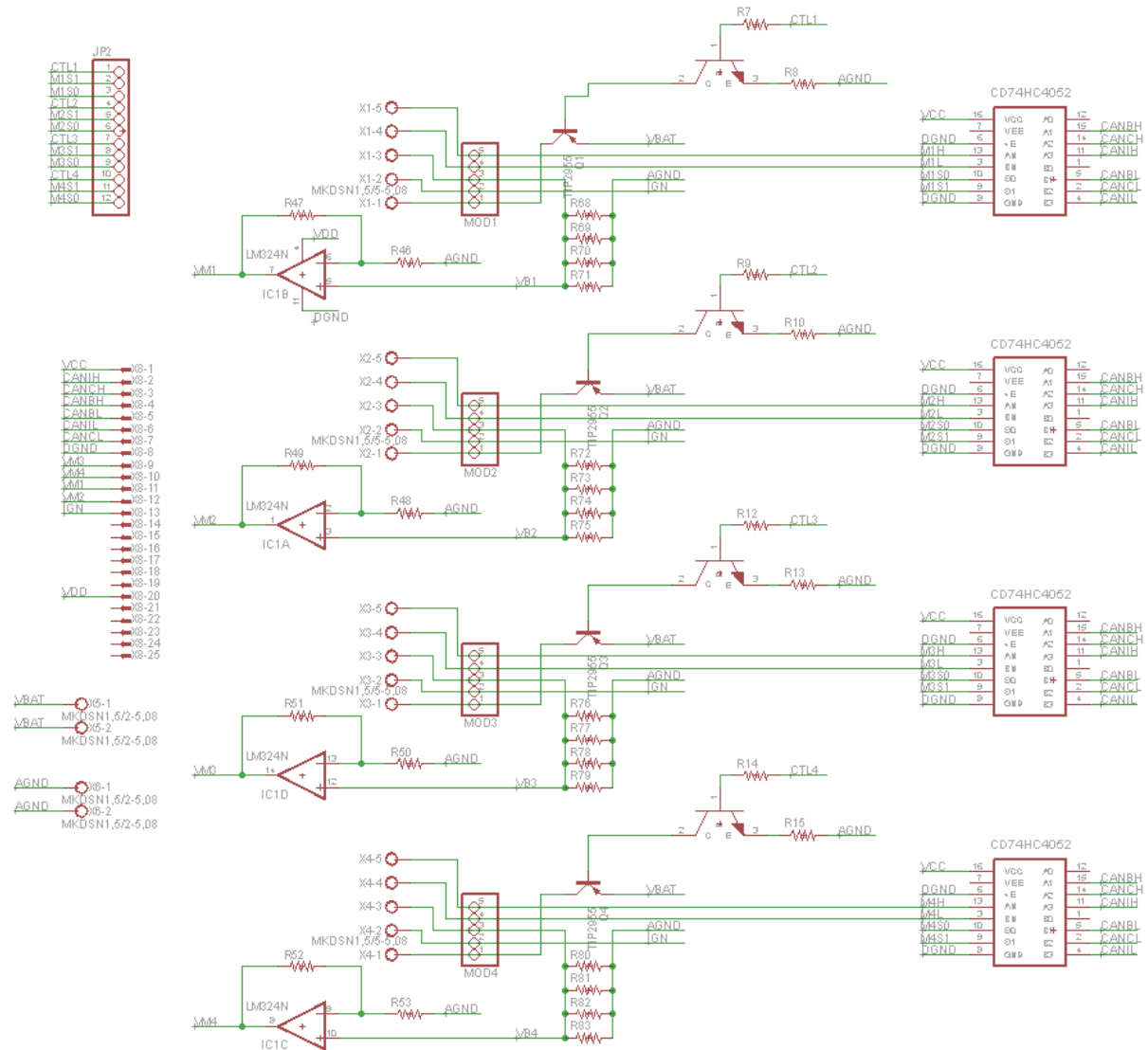


Figure A3.1: High power schematic

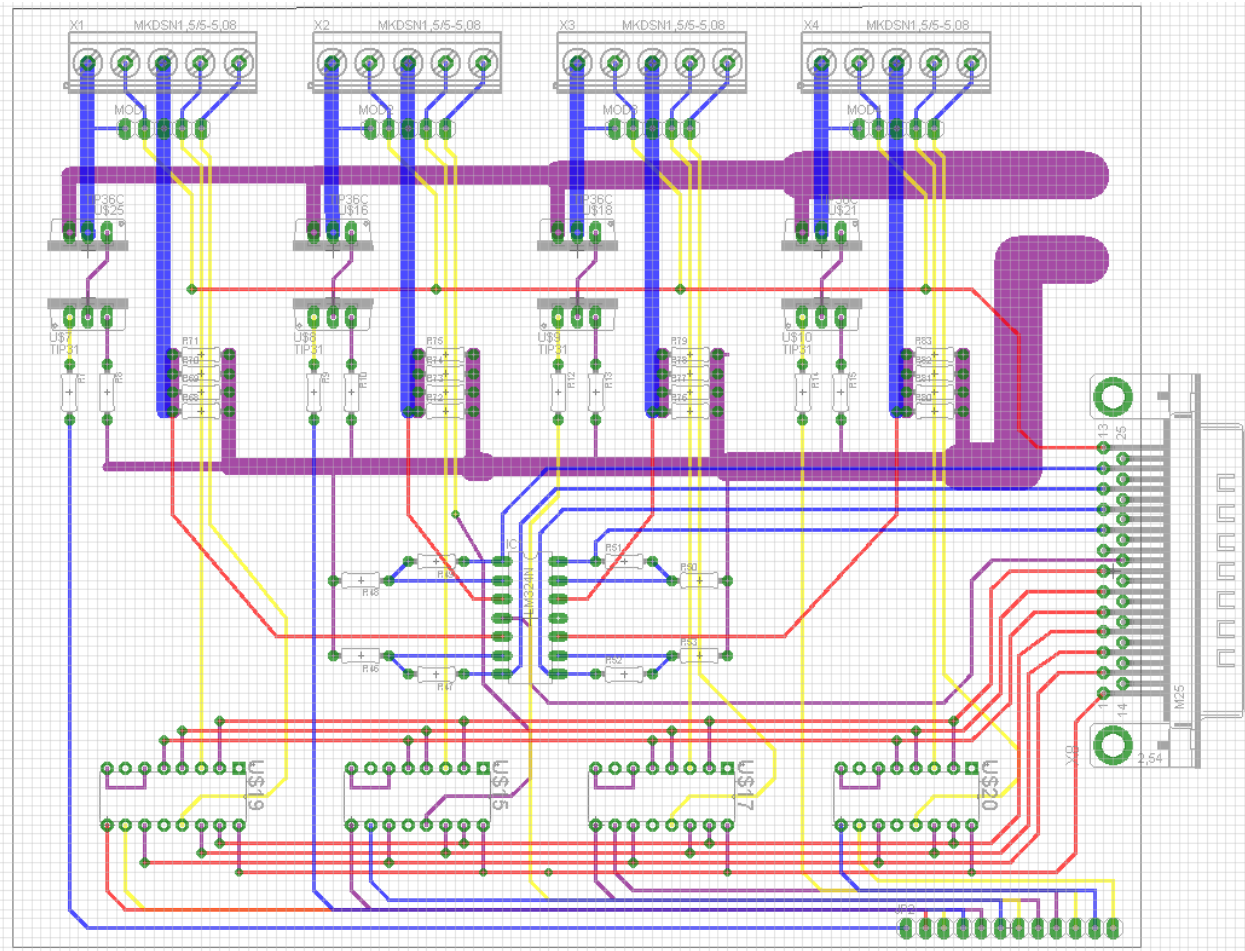


Figure A3.2: High Power PCB

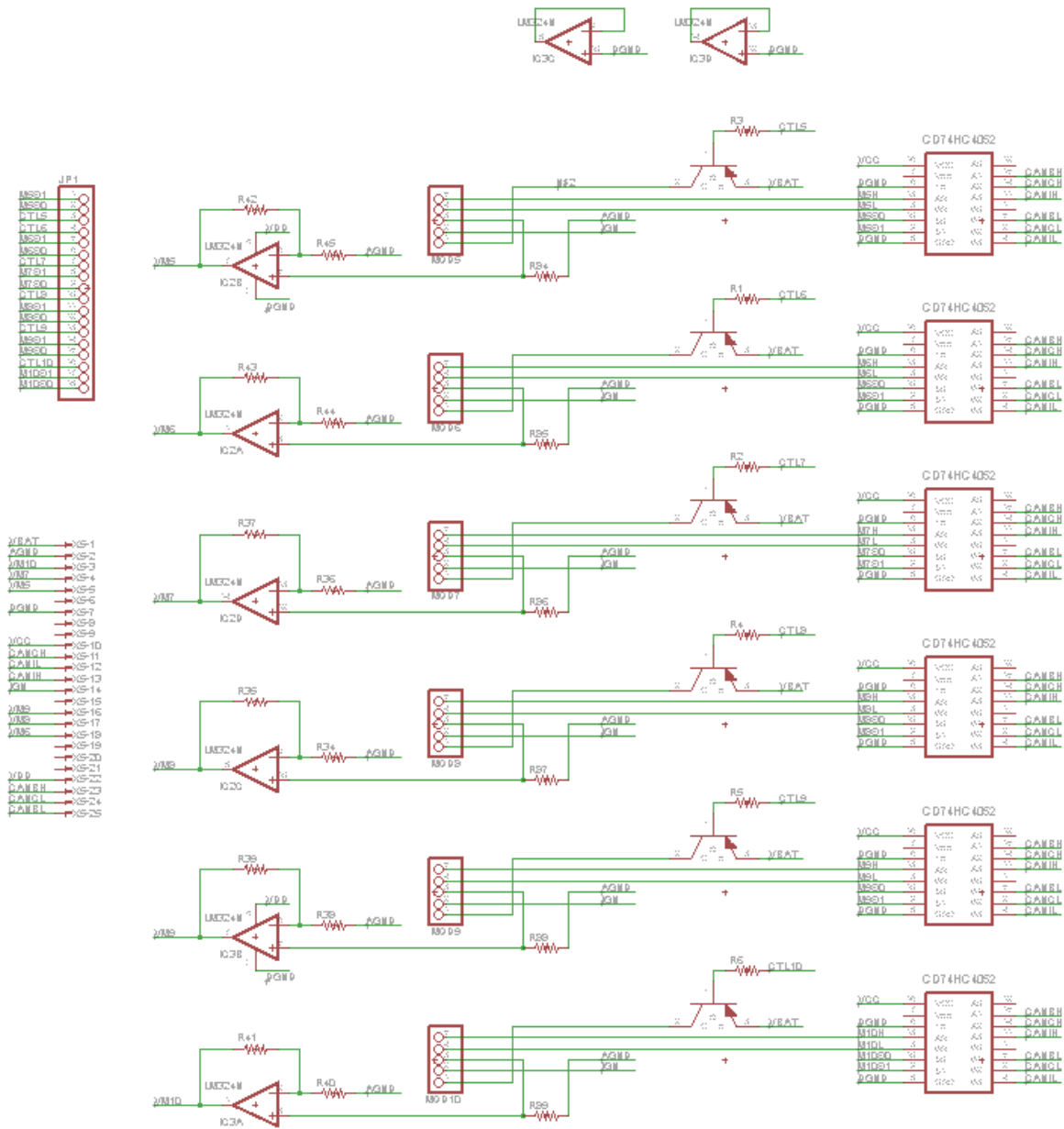


Figure A3.3: Low power schematic

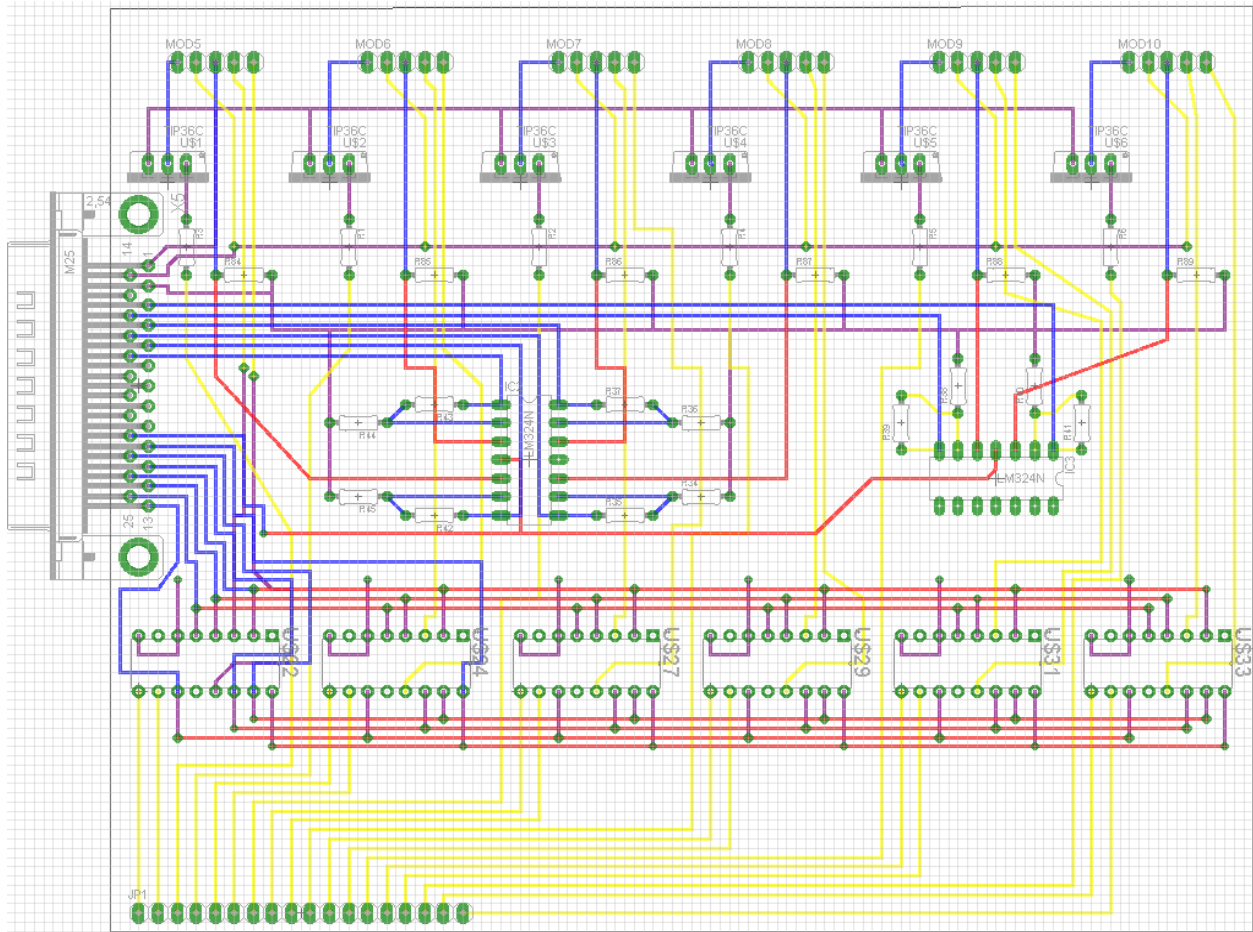


Figure A3.4: Low Power PCB



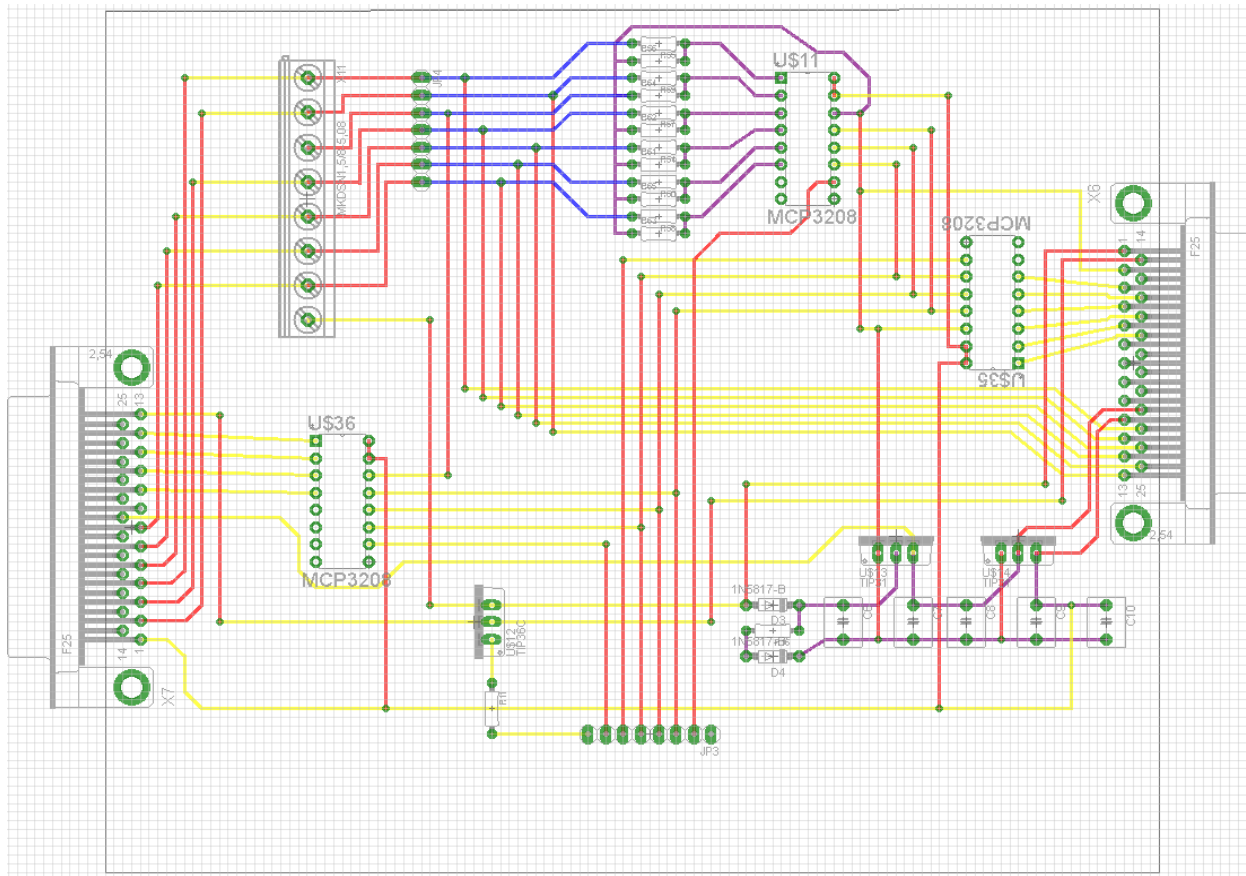


Figure A3.6: Control PCB

Graphical User Interface code implementation:

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'BreakoutApplication.ui'
#
# Created: Fri Nov 29 16:10:17 2013
# by: PyQt5 UI code generator 5.1.1
#
import serial
import time
import threading
import sys
import xlswriter
import os.path
import sched, time
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QApplication, QWidget, QDialog
```

```

from time import sleep
import signal
from threading import Timer
volts=""
port=""
count=0
leave=-1
t=1
serclose=-1
started=0
prin=0
scheduler = sched.scheduler(time.time, time.sleep)

def moduleLis(self):
    global started
    global t
    if started==0: # t isn't assigned yet so cannot kill the thread
        started=1
    else:
        t.cancel() #thread from prior run of the function is killed here
        if serclose==1:
            ser.close()
    out=""
    ser.write(bytearray('0 0 True'+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    if(']' in out):
        display(self,out)
    if leave==1:
        ser.close()
    t=threading.Timer(0.5, moduleLis,[self]) #has to be in a global function
    t.start() # new thread is created here

def display(self,out):
    print(out)
    out=out.split('\n')
    print (type(out))
    n=0
    global count
    #below is all the necessary parsing required to examine the data received from Initial.py

    while(len(out[n].split(",")<4 and n<9):
        n=n+1
    if n>=9:
        return
    first=out[n].replace('[','').replace(']', '').replace('\r','').replace(' ','').strip().split(", ")
    second=out[n+1].replace('[','').replace(']', '').replace('\r','').replace(' ','').strip().split(", ")
    third=out[n+2].replace('[','').replace(']', '').replace('\r','').replace(' ','').strip().split(", ")
    # change all the values on the GUI and store them on the spreadsheet
    x=0

```

```

count+=1
self.lineEdit.setText(str(third[0]))
worksheet.write(count, x, third[0])
worksheet.write(count, x+1, str(self.comboBox.currentText() )
x=x+2
self.lineEdit_2.setText(str(third[1]))
worksheet.write(count, x, third[1])
worksheet.write(count, x+1, str(self.comboBox_2.currentText() )
x=x+2
self.lineEdit_3.setText(str(third[2]))
worksheet.write(count, x, third[2])
worksheet.write(count, x+1, str(self.comboBox_3.currentText() )
x=x+2
self.lineEdit_4.setText(str(third[3]))
worksheet.write(count, x, third[3])
worksheet.write(count, x+1, str(self.comboBox_4.currentText() )
x=x+2
self.lineEdit_5.setText(str(second[0]))
worksheet.write(count, x, second[0])
worksheet.write(count, x+1, str(self.comboBox_5.currentText() )
x=x+2
self.lineEdit_6.setText(str(second[1]))
worksheet.write(count, x, second[1])
worksheet.write(count, x+1, str(self.comboBox_6.currentText() )
x=x+2
self.lineEdit_7.setText(str(second[2]))
worksheet.write(count, x, second[2])
worksheet.write(count, x+1, str(self.comboBox_7.currentText() )
x=x+2
self.lineEdit_8.setText(str(second[3]))
worksheet.write(count, x, second[3])
worksheet.write(count, x+1, str(self.comboBox_8.currentText() )
x=x+2
self.lineEdit_9.setText(str(second[4]))
worksheet.write(count, x, second[4])
worksheet.write(count, x+1, str(self.comboBox_9.currentText() )
x=x+2
self.lineEdit_10.setText(str(second[5]))
worksheet.write(count, x, second[5])
worksheet.write(count, x+1, str(self.comboBox_10.currentText() )
x=x+2
self.lineEdit_11.setText(str(first[0]))
worksheet.write(count, x, first[0])
x=x+1
self.lineEdit_12.setText(str(first[1]))
worksheet.write(count, x, first[0])
x=x+1
self.lineEdit_13.setText(str(first[2]))
worksheet.write(count, x, first[0])
x=x+1
self.lineEdit_14.setText(str(first[3]))
worksheet.write(count, x, first[0])
x=x+1
self.lineEdit_15.setText(str(first[4]))
worksheet.write(count, x, first[0])

```

```

x=x+1
self.lineEdit_16.setText(str(first[5]))
worksheet.write(count, x, first[0])
x=x+1

class Ui_Dialog(object):
    def __exit__():
        serclose=1
    #implementation for a listener function for every module when the CAN bus is changed
    def canListener(self):
        out=""
        val=str(self.comboBox.currentText()).replace('CAN ','')
        ser.write(bytearray('1 1 '+str(val)+'\r','ascii'))
        sleep(0.1)
        while ser.inWaiting() > 0:
            out += ser.read(1).decode("utf-8")
        if(out!=""):
            display(self,out)

    def canListener2(self):
        out=""
        val=str(self.comboBox_2.currentText()).replace('CAN ','')
        ser.write(bytearray('1 2 '+str(val)+'\r','ascii'))
        sleep(0.1)
        while ser.inWaiting() > 0:
            out += ser.read(1).decode("utf-8")
        if(out!=""):
            display(self,out)

    def canListener3(self):
        out=""
        val=str(self.comboBox_3.currentText()).replace('CAN ','')
        ser.write(bytearray('1 3 '+str(val)+'\r','ascii'))
        sleep(0.1)
        while ser.inWaiting() > 0:
            out += ser.read(1).decode("utf-8")
        if(out!=""):
            display(self,out)

    def canListener4(self):
        out=""
        val=str(self.comboBox_4.currentText()).replace('CAN ','')
        ser.write(bytearray('1 4 '+str(val)+'\r','ascii'))
        sleep(0.1)
        while ser.inWaiting() > 0:
            out += ser.read(1).decode("utf-8")
        if(out!=""):
            display(self,out)

    def canListener5(self):
        out=""
        val=str(self.comboBox_5.currentText()).replace('CAN ','')
        ser.write(bytearray('1 5 '+str(val)+'\r','ascii'))

```

```

sleep(0.1)
while ser.inWaiting() > 0:
    out += ser.read(1).decode("utf-8")
if(out!=""):
    display(self,out)

def canListener6(self):
    out=""
    val=str(self.comboBox_6.currentText()).replace('CAN ','')
    ser.write(bytearray('1 6 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    if(out!=""):
        display(self,out)

def canListener7(self):
    out=""
    val=str(self.comboBox_7.currentText()).replace('CAN ','')
    ser.write(bytearray('1 7 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    if(out!=""):
        display(self,out)

def canListener8(self):
    out=""
    val=str(self.comboBox_8.currentText()).replace('CAN ','')
    ser.write(bytearray('1 8 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    if(out!=""):
        display(self,out)

def canListener9(self):
    out=""
    val=str(self.comboBox_9.currentText()).replace('CAN ','')
    ser.write(bytearray('1 9 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    if(out!=""):
        display(self,out)

def canListener10(self):
    out=""
    val=str(self.comboBox_10.currentText()).replace('CAN ','')
    ser.write(bytearray('1 10 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    if(out!=""):

```

```

        display(self,out)

# implementation for the listener functions for every module when they are toggled on/off
def moduleListener(self):
    out=""
    val=self.checkBox.isChecked()
    ser.write(bytearray('2 1 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    display(self,out)

def moduleListener2(self):
    out=""
    val=self.checkBox_2.isChecked()
    ser.write(bytearray('2 2 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    display(self,out)

def moduleListener3(self):
    out=""
    val=self.checkBox_3.isChecked()
    ser.write(bytearray('2 3 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    display(self,out)

def moduleListener4(self):
    out=""
    val=self.checkBox_4.isChecked()
    ser.write(bytearray('2 4 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    display(self,out)

def moduleListener5(self):
    out=""
    val=self.checkBox_5.isChecked()
    ser.write(bytearray('2 5 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
    display(self,out)

def moduleListener6(self):
    out=""
    val=self.checkBox_6.isChecked()
    ser.write(bytearray('2 6 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:

```

```

        out += ser.read(1).decode("utf-8")
        display(self,out)

def moduleListener7(self):
    out=""
    val=self.checkBox_7.isChecked()
    ser.write(bytearray('2 7 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
        display(self,out)

def moduleListener8(self):
    out=""
    val=self.checkBox_8.isChecked()
    ser.write(bytearray('2 8 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
        display(self,out)

def moduleListener9(self):
    out=""
    val=self.checkBox_9.isChecked()
    ser.write(bytearray('2 9 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
        display(self,out)

def moduleListener10(self):
    out=""
    val=self.checkBox_10.isChecked()
    ser.write(bytearray('2 10 '+str(val)+'\r','ascii'))
    sleep(0.1)
    while ser.inWaiting() > 0:
        out += ser.read(1).decode("utf-8")
        display(self,out)

def setVolts(self):
    global volts
    volts=str(self.lineEdit_18.getText())

def setPort(self):
    global port
    port=str(self.lineEdit_17.getText())

def setupUi(self, Dialog):
    Dialog.setObjectName("Dialog")
    Dialog.resize(514, 399)
    self.checkBox = QtWidgets.QCheckBox(Dialog)
    self.checkBox.setGeometry(QtCore.QRect(0, 0, 70, 17))
    self.checkBox.setObjectName("checkBox")

```

```

self.checkBox_2 = QtWidgets.QCheckBox(Dialog)
self.checkBox_2.setGeometry(QtCore.QRect(0, 40, 70, 17))
self.checkBox_2.setObjectName("checkBox_2")
self.checkBox_3 = QtWidgets.QCheckBox(Dialog)
self.checkBox_3.setGeometry(QtCore.QRect(0, 80, 70, 17))
self.checkBox_3.setObjectName("checkBox_3")
self.checkBox_4 = QtWidgets.QCheckBox(Dialog)
self.checkBox_4.setGeometry(QtCore.QRect(0, 120, 70, 17))
self.checkBox_4.setObjectName("checkBox_4")
self.checkBox_5 = QtWidgets.QCheckBox(Dialog)
self.checkBox_5.setGeometry(QtCore.QRect(0, 160, 70, 17))
self.checkBox_5.setObjectName("checkBox_5")
self.checkBox_6 = QtWidgets.QCheckBox(Dialog)
self.checkBox_6.setGeometry(QtCore.QRect(0, 200, 70, 17))
self.checkBox_6.setObjectName("checkBox_6")
self.checkBox_7 = QtWidgets.QCheckBox(Dialog)
self.checkBox_7.setGeometry(QtCore.QRect(0, 240, 70, 17))
self.checkBox_7.setObjectName("checkBox_7")
self.checkBox_8 = QtWidgets.QCheckBox(Dialog)
self.checkBox_8.setGeometry(QtCore.QRect(0, 280, 70, 17))
self.checkBox_8.setObjectName("checkBox_8")
self.checkBox_9 = QtWidgets.QCheckBox(Dialog)
self.checkBox_9.setGeometry(QtCore.QRect(0, 320, 70, 17))
self.checkBox_9.setObjectName("checkBox_9")
self.checkBox_10 = QtWidgets.QCheckBox(Dialog)
self.checkBox_10.setGeometry(QtCore.QRect(0, 360, 70, 17))
self.checkBox_10.setObjectName("checkBox_10")
self.lineEdit = QtWidgets.QLineEdit(Dialog)
self.lineEdit.setGeometry(QtCore.QRect(100, 0, 113, 20))
self.lineEdit.setObjectName("lineEdit")
self.lineEdit_2 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_2.setGeometry(QtCore.QRect(100, 40, 113, 20))
self.lineEdit_2.setObjectName("lineEdit_2")
self.lineEdit_3 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_3.setGeometry(QtCore.QRect(100, 80, 113, 20))
self.lineEdit_3.setObjectName("lineEdit_3")
self.lineEdit_4 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_4.setGeometry(QtCore.QRect(100, 120, 113, 20))
self.lineEdit_4.setObjectName("lineEdit_4")
self.lineEdit_5 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_5.setGeometry(QtCore.QRect(100, 160, 113, 20))
self.lineEdit_5.setObjectName("lineEdit_5")
self.lineEdit_6 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_6.setGeometry(QtCore.QRect(100, 200, 113, 20))
self.lineEdit_6.setObjectName("lineEdit_6")
self.lineEdit_7 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_7.setGeometry(QtCore.QRect(100, 240, 113, 20))
self.lineEdit_7.setObjectName("lineEdit_7")
self.lineEdit_8 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_8.setGeometry(QtCore.QRect(100, 280, 113, 20))
self.lineEdit_8.setObjectName("lineEdit_8")
self.lineEdit_9 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_9.setGeometry(QtCore.QRect(100, 320, 113, 20))
self.lineEdit_9.setObjectName("lineEdit_9")
self.lineEdit_10 = QtWidgets.QLineEdit(Dialog)

```

```

self.lineEdit_10.setGeometry(QtCore.QRect(100, 360, 113, 20))
self.lineEdit_10.setObjectName("lineEdit_10")
self.comboBox = QtWidgets.QComboBox(Dialog)
self.comboBox.setGeometry(QtCore.QRect(250, 0, 69, 22))
self.comboBox.setObjectName("comboBox")
self.comboBox.addItem("")
self.comboBox.addItem("")
self.comboBox.addItem("")
self.comboBox_2 = QtWidgets.QComboBox(Dialog)
self.comboBox_2.setGeometry(QtCore.QRect(250, 40, 69, 22))
self.comboBox_2.setObjectName("comboBox_2")
self.comboBox_2.addItem("")
self.comboBox_2.addItem("")
self.comboBox_2.addItem("")
self.comboBox_3 = QtWidgets.QComboBox(Dialog)
self.comboBox_3.setGeometry(QtCore.QRect(250, 80, 69, 22))
self.comboBox_3.setObjectName("comboBox_3")
self.comboBox_3.addItem("")
self.comboBox_3.addItem("")
self.comboBox_3.addItem("")
self.comboBox_4 = QtWidgets.QComboBox(Dialog)
self.comboBox_4.setGeometry(QtCore.QRect(250, 120, 69, 22))
self.comboBox_4.setObjectName("comboBox_4")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_4.addItem("")
self.comboBox_5 = QtWidgets.QComboBox(Dialog)
self.comboBox_5.setGeometry(QtCore.QRect(250, 160, 69, 22))
self.comboBox_5.setObjectName("comboBox_5")
self.comboBox_5.addItem("")
self.comboBox_5.addItem("")
self.comboBox_5.addItem("")
self.comboBox_6 = QtWidgets.QComboBox(Dialog)
self.comboBox_6.setGeometry(QtCore.QRect(250, 200, 69, 22))
self.comboBox_6.setObjectName("comboBox_6")
self.comboBox_6.addItem("")
self.comboBox_6.addItem("")
self.comboBox_6.addItem("")
self.comboBox_6.addItem("")
self.comboBox_6.setItemText(3, "")
self.comboBox_7 = QtWidgets.QComboBox(Dialog)
self.comboBox_7.setGeometry(QtCore.QRect(250, 240, 69, 22))
self.comboBox_7.setObjectName("comboBox_7")
self.comboBox_7.addItem("")
self.comboBox_7.addItem("")
self.comboBox_7.addItem("")
self.comboBox_8 = QtWidgets.QComboBox(Dialog)
self.comboBox_8.setGeometry(QtCore.QRect(250, 280, 69, 22))
self.comboBox_8.setObjectName("comboBox_8")
self.comboBox_8.addItem("")
self.comboBox_8.addItem("")
self.comboBox_8.addItem("")
self.comboBox_9 = QtWidgets.QComboBox(Dialog)
self.comboBox_9.setGeometry(QtCore.QRect(250, 320, 69, 22))
self.comboBox_9.setObjectName("comboBox_9")

```

```

self.comboBox_9.addItem("")
self.comboBox_9.addItem("")
self.comboBox_9.addItem("")
self.comboBox_10 = QtWidgets.QComboBox(Dialog)
self.comboBox_10.setGeometry(QtCore.QRect(250, 360, 69, 22))
self.comboBox_10.setObjectName("comboBox_10")
self.comboBox_10.addItem("")
self.comboBox_10.addItem("")
self.comboBox_10.addItem("")
self.label = QtWidgets.QLabel(Dialog)
self.label.setGeometry(QtCore.QRect(410, 0, 46, 13))
self.label.setObjectName("label")
self.label_2 = QtWidgets.QLabel(Dialog)
self.label_2.setGeometry(QtCore.QRect(410, 250, 46, 13))
self.label_2.setObjectName("label_2")
self.label_3 = QtWidgets.QLabel(Dialog)
self.label_3.setGeometry(QtCore.QRect(410, 50, 46, 13))
self.label_3.setObjectName("label_3")
self.label_4 = QtWidgets.QLabel(Dialog)
self.label_4.setGeometry(QtCore.QRect(410, 150, 46, 13))
self.label_4.setObjectName("label_4")
self.label_5 = QtWidgets.QLabel(Dialog)
self.label_5.setGeometry(QtCore.QRect(410, 100, 46, 13))
self.label_5.setObjectName("label_5")
self.label_6 = QtWidgets.QLabel(Dialog)
self.label_6.setGeometry(QtCore.QRect(410, 200, 46, 13))
self.label_6.setObjectName("label_6")
self.lineEdit_11 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_11.setGeometry(QtCore.QRect(380, 20, 113, 20))
self.lineEdit_11.setObjectName("lineEdit_11")
self.lineEdit_12 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_12.setGeometry(QtCore.QRect(380, 70, 113, 20))
self.lineEdit_12.setObjectName("lineEdit_12")
self.lineEdit_13 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_13.setGeometry(QtCore.QRect(380, 120, 113, 20))
self.lineEdit_13.setObjectName("lineEdit_13")
self.lineEdit_14 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_14.setGeometry(QtCore.QRect(380, 170, 113, 20))
self.lineEdit_14.setObjectName("lineEdit_14")
self.lineEdit_15 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_15.setGeometry(QtCore.QRect(380, 220, 113, 20))
self.lineEdit_15.setObjectName("lineEdit_15")
self.lineEdit_16 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_16.setGeometry(QtCore.QRect(380, 270, 113, 20))
self.lineEdit_16.setObjectName("lineEdit_16")
self.label_7 = QtWidgets.QLabel(Dialog)
self.label_7.setGeometry(QtCore.QRect(340, 320, 61, 16))
self.label_7.setObjectName("label_7")
self.label_8 = QtWidgets.QLabel(Dialog)
self.label_8.setGeometry(QtCore.QRect(350, 360, 46, 13))
self.label_8.setObjectName("label_8")
self.lineEdit_17 = QtWidgets.QLineEdit(Dialog)
self.lineEdit_17.setGeometry(QtCore.QRect(400, 360, 91, 20))
self.lineEdit_17.setObjectName("lineEdit_17")
self.lineEdit_18 = QtWidgets.QLineEdit(Dialog)

```

```

self.lineEdit_18.setGeometry(QRect(400, 320, 91, 20))
self.lineEdit_18.setObjectName("lineEdit_18")

self.retranslateUi(Dialog)
self.checkBox.stateChanged['int'].connect(self.moduleListener)
self.comboBox.currentTextChanged['QString'].connect(self.canListener)
self.checkBox_2.stateChanged['int'].connect(self.moduleListener2)
self.checkBox_3.stateChanged['int'].connect(self.moduleListener3)
self.checkBox_4.stateChanged['int'].connect(self.moduleListener4)
self.checkBox_5.stateChanged['int'].connect(self.moduleListener5)
self.checkBox_6.stateChanged['int'].connect(self.moduleListener6)
self.checkBox_7.stateChanged['int'].connect(self.moduleListener7)
self.checkBox_8.stateChanged['int'].connect(self.moduleListener8)
self.checkBox_9.stateChanged['int'].connect(self.moduleListener9)
self.checkBox_10.stateChanged['int'].connect(self.moduleListener10)
self.comboBox_2.currentTextChanged['QString'].connect(self.canListener2)
self.comboBox_3.currentTextChanged['QString'].connect(self.canListener3)
self.comboBox_4.currentTextChanged['QString'].connect(self.canListener4)
self.comboBox_5.currentTextChanged['QString'].connect(self.canListener5)
self.comboBox_6.currentTextChanged['QString'].connect(self.canListener6)
self.comboBox_7.currentTextChanged['QString'].connect(self.canListener7)
self.comboBox_8.currentTextChanged['QString'].connect(self.canListener8)
self.comboBox_9.currentTextChanged['QString'].connect(self.canListener9)
self.comboBox_10.currentTextChanged['QString'].connect(self.canListener10)
self.lineEdit_18.textChanged['QString'].connect(self.setVolts)
self.lineEdit_17.textChanged['QString'].connect(self.setPort)
QtCore.QMetaObject.connectSlotsByName(Dialog)

```

```

def retranslateUi(self, Dialog):
    _translate = QtCore.QCoreApplication.translate
    Dialog.setWindowTitle(_translate("Dialog", "Dialog"))
    self.checkBox.setText(_translate("Dialog", "Module 1"))
    self.checkBox_2.setText(_translate("Dialog", "Module 2"))
    self.checkBox_3.setText(_translate("Dialog", "Module 3"))
    self.checkBox_4.setText(_translate("Dialog", "Module 4"))
    self.checkBox_5.setText(_translate("Dialog", "Module 5"))
    self.checkBox_6.setText(_translate("Dialog", "Module 6"))
    self.checkBox_7.setText(_translate("Dialog", "Module 7"))
    self.checkBox_8.setText(_translate("Dialog", "Module 8"))
    self.checkBox_9.setText(_translate("Dialog", "Module 9"))
    self.checkBox_10.setText(_translate("Dialog", "Module 10"))
    self.comboBox.setItemText(0, _translate("Dialog", "CAN B"))
    self.comboBox.setItemText(1, _translate("Dialog", "CAN C"))
    self.comboBox.setItemText(2, _translate("Dialog", "CAN I"))
    self.comboBox_2.setItemText(0, _translate("Dialog", "CAN B"))
    self.comboBox_2.setItemText(1, _translate("Dialog", "CAN C"))
    self.comboBox_2.setItemText(2, _translate("Dialog", "CAN I"))
    self.comboBox_3.setItemText(0, _translate("Dialog", "CAN B"))
    self.comboBox_3.setItemText(1, _translate("Dialog", "CAN C"))
    self.comboBox_3.setItemText(2, _translate("Dialog", "CAN I"))
    self.comboBox_4.setItemText(0, _translate("Dialog", "CAN B"))
    self.comboBox_4.setItemText(1, _translate("Dialog", "CAN C"))
    self.comboBox_4.setItemText(2, _translate("Dialog", "CAN I"))
    self.comboBox_5.setItemText(0, _translate("Dialog", "CAN B"))

```

```

self.comboBox_5.setItemText(1, _translate("Dialog", "CAN C"))
self.comboBox_5.setItemText(2, _translate("Dialog", "CAN I"))
self.comboBox_6.setItemText(0, _translate("Dialog", "CAN B"))
self.comboBox_6.setItemText(1, _translate("Dialog", "CAN C"))
self.comboBox_6.setItemText(2, _translate("Dialog", "CAN I"))
self.comboBox_7.setItemText(0, _translate("Dialog", "CAN B"))
self.comboBox_7.setItemText(1, _translate("Dialog", "CAN C"))
self.comboBox_7.setItemText(2, _translate("Dialog", "CAN I"))
self.comboBox_8.setItemText(0, _translate("Dialog", "CAN B"))
self.comboBox_8.setItemText(1, _translate("Dialog", "CAN C"))
self.comboBox_8.setItemText(2, _translate("Dialog", "CAN I"))
self.comboBox_9.setItemText(0, _translate("Dialog", "CAN B"))
self.comboBox_9.setItemText(1, _translate("Dialog", "CAN C"))
self.comboBox_9.setItemText(2, _translate("Dialog", "CAN I"))
self.comboBox_10.setItemText(0, _translate("Dialog", "CAN B"))
self.comboBox_10.setItemText(1, _translate("Dialog", "CAN C"))
self.comboBox_10.setItemText(2, _translate("Dialog", "CAN I"))
self.label.setText(_translate("Dialog", "CAN B Hi"))
self.label_2.setText(_translate("Dialog", "CAN I Lo"))
self.label_3.setText(_translate("Dialog", "CAN B Lo"))
self.label_4.setText(_translate("Dialog", "CAN C Lo"))
self.label_5.setText(_translate("Dialog", "CAN C Hi"))
self.label_6.setText(_translate("Dialog", "CAN I Hi"))
self.label_7.setText(_translate("Dialog", "Voltage (V)"))
self.label_8.setText(_translate("Dialog", "Port"))
self.lineEdit_18.setText(_translate("Dialog", "12"))

```

```

class redefine(QtWidgets.QDialog):

```

```

    def closeEvent(self, event):
        ser.close()
        serclose=1
        workbook.close()

```

```

if __name__ == "__main__":

```

```

    import sys
    port = 'COM3'
    import win32com.client
    #auto-detection of port
    wmi = win32com.client.GetObject("winmgmts:")
    for s in wmi.InstancesOf("Win32_SerialPort"):
        if s.Description=='Gadget Serial':
            port= s.Name
    port=str(port[-5:-1])
    #establish serial connection
    ser = serial.Serial(
        port,
        baudrate=9600,
        parity=serial.PARITY_ODD,
        stopbits=serial.STOPBITS_TWO,
        bytesize=serial.SEVENBITS
    )
    ser.isOpen()
    out=""
    while ser.inWaiting() > 0:

```

```

    out += ser.read(1).decode("utf-8")
time.sleep(1)
    #login to the board
ser.write(bytearray("\r",'ascii'))
sleep(2)
ser.write(bytearray('root' + '\r','ascii'))
#sleep(3)
ser.write(bytearray("\r",'ascii'))
ser.write(bytearray('echo BB-SPI0-01 > /sys/devices/bone_capemgr.* /slots'+'\r','ascii'))
ser.write(bytearray('echo BB-SPI1-01 > /sys/devices/bone_capemgr.* /slots'+'\r','ascii'))
    #run the Initial.py program on the board
ser.write(bytearray('python Initial.py'+'\r'+'\r','ascii'))

app = QtWidgets.QApplication(sys.argv)
Dialog = redefin()
ui = Ui_Dialog()
ui.setupUi(Dialog)
Dialog.show()
#create new workbook and worksheet to store data
workbook = xlswriter.Workbook('ModuleData.xlsx')
worksheet = workbook.add_worksheet()

worksheet.set_column('A:A', 15)

# Add a bold format to use to highlight cells.
bold = workbook.add_format({'bold': 1})

# Text with formatting.
worksheet.write('A1', 'Module1', bold)
worksheet.write('B1', 'Module1 bus', bold)
worksheet.write('C1', 'Module2', bold)
worksheet.write('D1', 'Module2 bus', bold)
worksheet.write('E1', 'Module3', bold)
worksheet.write('F1', 'Module3 bus', bold)
worksheet.write('G1', 'Module4', bold)
worksheet.write('H1', 'Module4 bus', bold)
worksheet.write('I1', 'Module5', bold)
worksheet.write('J1', 'Module5 bus', bold)
worksheet.write('K1', 'Module6', bold)
worksheet.write('L1', 'Module6 bus', bold)
worksheet.write('M1', 'Module7', bold)
worksheet.write('N1', 'Module7 bus', bold)
worksheet.write('O1', 'Module8', bold)
worksheet.write('P1', 'Module8 bus', bold)
worksheet.write('Q1', 'Module9', bold)
worksheet.write('R1', 'Module9 bus', bold)
worksheet.write('S1', 'Module10', bold)
worksheet.write('T1', 'Module10 bus', bold)
worksheet.write('U1', 'CAN B Hi', bold)
worksheet.write('V1', 'CAN B Lo', bold)
worksheet.write('W1', 'CAN C Hi', bold)
worksheet.write('X1', 'CAN C Lo', bold)
worksheet.write('Y1', 'CAN I Hi', bold)
worksheet.write('Z1', 'CAN I Lo', bold)

```

```
moduleLis(ui)
```

```
sys.exit(app.exec_())
```

Initial.py Code for the BeagleBone Black

```
import Adafruit_BBIO.GPIO as GPIO
import spi
import time
from time import sleep #Not neccessary, but required if any delay were desired

#Change to correct pin layouts once PCB is done
module_dict = { '1': ["P8_11", "P8_12", "P8_37"], '2': ["P8_13", "P8_14", "P8_38"], '3': ["P8_15",
"P8_16", "P8_39"], \
'4': ["P8_17", "P8_18", "P8_40"], '5': ["P8_19", "P8_26", "P8_41"], '6': ["P8_27", "P8_28", "P8_42"], '7':
["P8_29", "P8_30", "P8_43"], \
'8': ["P8_31", "P8_32", "P8_44"], '9': ["P8_33", "P8_34", "P8_45"], '10': ["P8_35", "P8_36", "P8_46"]}

can_dict = { 'B' : [GPIO.HIGH, GPIO.HIGH], 'C' : [GPIO.HIGH, GPIO.LOW], 'I': [GPIO.LOW,
GPIO.HIGH], 'OFF': [GPIO.LOW, GPIO.LOW]}

SPI_dict = { '1': "060000", '2': "064000", '3': "068000", '4': "06C000", '5': "070000", '6': "074000" }

can = spi.SPI(1,0) #CAN Voltage
high = spi.SPI(2,0) #High Power Voltages
low = spi.SPI(2,1) #Low Power Voltages
length_data = 3
def Module(modnum, bus): #Function for changing CAN bus
    GPIO.output(module_dict[modnum][0], can_dict[bus.upper()][0])
    GPIO.output(module_dict[modnum][1], can_dict[bus.upper()][1])

def ModuleToggle(modnum, OnOff): #Function for toggling modules on/off
    modnum = str(modnum)
    if OnOff.upper() == "TRUE":
        GPIO.output(module_dict[modnum][2], GPIO.HIGH)
    elif OnOff.upper() == "FALSE":
        GPIO.output(module_dict[modnum][2], GPIO.LOW)

def ReadVoltages(): #Function for reading voltages
    voltage = [0,0,0,0,0,0]
    for ele in SPI_dict:
        raw_volt = can.transfer(SPI_dict[ele], length_data)
        a = int(ele)
```

```

        voltage[a-1] = (raw_volt[1]-14*16)*256+raw_volt[2]
    # count = 0
    # for i in voltage:
        # voltage[count]="%.3f" %((i/4096.0)*3.27) #Be sure to update 3.3V value to reflect VCC
        # count +=1
    return voltage
    #This function will read the voltages from the 3 CAN buses and return values as list,
    #[CANBHigh-GND, CANBLow-GND, CANCHigh-GND, CANCLow-GND,CANIHigh-GND, CANILow-GND]

def ReadLowCurrent(modnum): #Read voltages on low current modules
    modnum = int(modnum)
    if modnum == 0:
        voltage = [0,0,0,0,0,0]
        for ele in SPI_dict:
            raw_volt = low.transfer(SPI_dict[ele], length_data)
            a = int(ele)
            voltage[a-1] = (raw_volt[1]-14*16)*256+raw_volt[2]
    else:
        voltage = [0]
        raw_volt = low.transfer(SPI_dict[modnum], length_data)
        voltage[0] = (raw_volt[1]-14*16)*256+raw_volt[2]
    # count = 0
    # for i in voltage:
        # voltage[count]="%.3f" %((((i/4096.0)*3.282)/18.18)*1.65+.002) #Be sure to update 3.3V
value to reflect VCC
        # count +=1
    return voltage

    #This function will take the readings from the specific SPI chip and return a value that will get passed
back to the GUI.
    #Modnum is which module the current needs to be read at, and 0 parameter returns all 10 current
readings in a list
    #[Mod1, Mod2, Mod3, Mod4, Mod5, Mod6, Mod7, Mod8, Mod9, Mod10]
def ReadHighCurrent(modnum): #Read voltages on high current modules
    modnum = int(modnum)
    if modnum == 0:
        voltage = [0,0,0,0]
        for ele in SPI_dict:
            if int(ele) < 5:
                raw_volt = high.transfer(SPI_dict[ele], length_data)
                a = int(ele)
                voltage[a-1] =(raw_volt[1]-14*16)*256+raw_volt[2]
            else:
                break
    else:
        voltage = [0]
        raw_volt = high.transfer(SPI_dict[modnum], length_data)
        voltage[0] = (raw_volt[1]-14*16)*256+raw_volt[2]
    return voltage

for pin in module_dict:
    GPIO.setup(module_dict[pin][0], GPIO.OUT)
    GPIO.setup(module_dict[pin][1], GPIO.OUT)

```

```

GPIO.setup(module_dict[pin][2], GPIO.OUT)
Module(pin, 'OFF')
#ModuleToggle('1', "on") TESTING
while(1):
    #Module('1', 'B')
    #print "CAN"
    #print ReadVoltages() #Reading CAN Bus Voltages
    #print "Current"
    #print ReadLowCurrent(0)
    #print ReadHighCurrent(0)
    command = raw_input() #Looking for user input from GUI
    # 0-1-2 ModuleNumber Command
    command = command.split() #Convert input string to list
    if command[0] == '1': #1 for if the CAN bus has been changed, therefore command[2] is the new bus
        Module(command[1], command[2])
    elif command[0] == '2': #2 for if the module's power state has changed, therefore command[2] is the
new power state
        ModuleToggle(command[1], command[2])
    #Input 0 for first element if no change has been made
    print ReadVoltages() #CAN Bus
    print ReadLowCurrent(0) #Low Voltage
    print ReadHighCurrent(0) #High Voltage

```

SPImodule.c (SPI bus Library for python)

```

/*
 * spimodule.c - Python bindings for Linux SPI access through spidev
 * Copyright (C) 2009 Volker Thoms <unconnected@gmx.de>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; version 2 of the License.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 *
 * Edited by Meng Cao 2013
 */

#include <Python.h>
#include "structmember.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>

```

```

#include <linux/spi/spidev.h>
#include <linux/types.h>
#include <sys/ioctl.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))
#define MAXPATH 16

PyDoc_STRVAR(SPI_module_doc,
    "This module defines an object type that allows SPI transactions\n"
    "on hosts running the Linux kernel. The host kernel must have SPI\n"
    "support and SPI device interface support.\n"
    "All of these can be either built-in to the kernel, or loaded from\n"
    "modules.\n"
    "\n"
    "Because the SPI device interface is opened R/W, users of this\n"
    "module usually must have root permissions.\n");

typedef struct {
    PyObject_HEAD

    int fd; /* open file descriptor: /dev/spi-X.Y */
    uint8_t mode; /* current SPI mode */
    uint8_t bpw; /* current SPI bits per word setting */
    uint32_t msh; /* current SPI max speed setting in Hz */
} SPI;

static PyObject *
SPI_new(PyTypeObject *type, PyObject *args, PyObject *kwargs)
{
    SPI *self;
    if ((self = (SPI *)type->tp_alloc(type, 0)) == NULL)
        return NULL;

    self->fd = -1;
    self->mode = 0;
    self->bpw = 0;
    self->msh = 0;

    Py_INCREF(self);
    return (PyObject *)self;
}

PyDoc_STRVAR(SPI_close_doc,
    "close()\n\n"
    "Disconnects the object from the interface.\n");

static PyObject *SPI_close(SPI *self)
{
    if ((self->fd != -1) && (close(self->fd) == -1)) {
        PyErr_SetFromErrno(PyExc_IOError);
        return NULL;
    }
}

```

```

self->fd = -1;
self->mode = 0;
self->bpw = 0;
self->msh = 0;

Py_INCREF(Py_None);
return Py_None;
}

PyDoc_STRVAR(SPI_transfer_doc,
"transfer([values]) -> [values]\n\n"
"Perform SPI transaction.\n"
"CS will be released and reactivated between blocks.\n"
"delay specifies delay in usec between blocks.\n");

static PyObject* SPI_transfer(SPI *self, PyObject *args)
{
    uint8_t bits = 8;
    int ret = 0;
    char* list;
    int length_list = 1;
    uint16_t delay = 5;
    uint32_t speed = 1000000;
    int i=0;

    PyArg_ParseTuple(args, "s|i:transfer", &list, &length_list);

    char hexbyte[3] = {0};
    uint8_t tx[length_list];
    for (i=0; i < (length_list); i++){

        hexbyte[0] = list[2*i];
        hexbyte[1] = list[(2*i)+1];

        sscanf(hexbyte, "%X", &tx[i]);
    }

    uint8_t rx[ARRAY_SIZE(tx)];

    /*This is the transfer part, and sets up
    the details needed to transfer the data*/
    struct spi_ioc_transfer tr = {
        .tx_buf = (unsigned long)tx,
        .rx_buf = (unsigned long)rx,
        .len = ARRAY_SIZE(tx),
        .delay_usecs = delay,
        .speed_hz = speed,
        .bits_per_word = bits,
    };

```

```

//The Actual Transfer command and data, does send and receive!! Very important!
ret = ioctl(self->fd, SPI_IOC_MESSAGE(1), &tr);
if (ret < 1)
    printf("ERROR: Can't send spi message");

//Return received data
return Py_BuildValue("i,i,i", rx[0], rx[1], rx[2]);
}

PyDoc_STRVAR(SPI_open_doc,
"open(bus, device)\n\n"
"Connects the object to the specified SPI device.\n"
"open(X,Y) will open /dev/spidev-X.Y\n");

static PyObject *SPI_open(SPI *self, PyObject *args, PyObject *kwargs)
{
    int bus, device;
    char path[MAXPATH];
    uint8_t tmp8;
    uint32_t tmp32;
    static char *kwlist[] = {"bus", "device", NULL};
    //Checking Errors
    if (!PyArg_ParseTupleAndKeywords(args, kwargs, "ii:open", kwlist, &bus, &device))
        return NULL;
    if (snprintf(path, MAXPATH, "/dev/spidev%d.%d", bus, device) >= MAXPATH) {
        PyErr_SetString(PyExc_OverflowError,
            "Bus and/or device number is invalid.");
        return NULL;
    }
    if ((self->fd = open(path, O_RDWR, 0)) < 0) {
        printf("can't open device");
        abort();
    }
    if (ioctl(self->fd, SPI_IOC_RD_MODE, &tmp8) == -1) {
        printf("can't get spi mode");
        abort();
    }
    self->mode = tmp8;
    if (ioctl(self->fd, SPI_IOC_RD_BITS_PER_WORD, &tmp8) == -1) {
        printf("can't get bits per word");
        abort();
    }
    self->bpw = tmp8;
    if (ioctl(self->fd, SPI_IOC_RD_MAX_SPEED_HZ, &tmp32) == -1) {
        printf("can't get max speed hz");
        abort();
    }
    self->msh = tmp32;

    Py_INCREF(Py_None);
    return Py_None;
}

```

```

static int SPI_init(SPI *self, PyObject *args, PyObject *kwds)
{
    int bus = -1;
    int client = -1;
    static char *kwlist[] = {"bus", "client", NULL};

    if (!PyArg_ParseTupleAndKeywords(args, kwds, "|ii: __init__ ",
                                     kwlist, &bus, &client))
        return -1;

    if (bus >= 0) {
        SPI_open(self, args, kwds);
        if (PyErr_Occurred())
            return -1;
    }

    return 0;
}

static PyMethodDef SPI_module_methods[] = {
    { NULL },
};

PyDoc_STRVAR(SPI_type_doc,
"SPI([bus],[client]) -> SPI\n\n"
"Return a new SPI object that is (optionally) connected to the\n"
"specified SPI device interface.\n");

static PyMethodDef SPI_methods[] = {
    {"open", (PyCFunction)SPI_open, METH_VARARGS | METH_KEYWORDS,
     SPI_open_doc},
    {"close", (PyCFunction)SPI_close, METH_NOARGS,
     SPI_close_doc},
    {"transfer", (PyCFunction)SPI_transfer, METH_VARARGS,
     SPI_transfer_doc},
    {NULL},
};

static PyTypeObject SPI_type = {
    PyObject_HEAD_INIT(NULL)
    0, /* ob_size */
    "SPI", /* tp_name */
    sizeof(SPI), /* tp_basicsize */
    0, /* tp_itemsize */
    0, /* tp_dealloc */
    0, /* tp_print */
    0, /* tp_getattr */
    0, /* tp_setattr */
    0, /* tp_compare */
    0, /* tp_repr */
    0, /* tp_as_number */
    0, /* tp_as_sequence */
    0, /* tp_as_mapping */

```

```

0,          /* tp_hash */
0,          /* tp_call */
0,          /* tp_str */
0,          /* tp_getattro */
0,          /* tp_setattro */
0,          /* tp_as_buffer */
Py_TPFLAGS_DEFAULT, /* tp_flags */
SPI_type_doc, /* tp_doc */
0,          /* tp_traverse */
0,          /* tp_clear */
0,          /* tp_richcompare */
0,          /* tp_weaklistoffset */
0,          /* tp_iter */
0,          /* tp_iternext */
SPI_methods, /* tp_methods */
0,          /* tp_members */
0,          /* tp_getset */
0,          /* tp_base */
0,          /* tp_dict */
0,          /* tp_descr_get */
0,          /* tp_descr_set */
0,          /* tp_dictoffset */
(initproc)SPI_init, /* tp_init */
0,          /* tp_alloc */
SPI_new,    /* tp_new */
};

#ifdef PyMODINIT_FUNC /* declarations for DLL import/export */
#define PyMODINIT_FUNC void
#endif

PyMODINIT_FUNC initspi(void)
{
    PyObject* m;

    if (PyType_Ready(&SPI_type) < 0)
        return;

    m = Py_InitModule3("spi", SPI_module_methods, SPI_module_doc);
    Py_INCREF(&SPI_type);
    PyModule_AddObject(m, "SPI", (PyObject *)&SPI_type);
}

```

BB-SPI0-01-00A0.dts (SPI0 Driver)

```

/dts-v1/;
/plugin/;

/ {
    compatible = "ti,beaglebone", "ti,beaglebone-black";

```

```

/* identification */
part-number = "spi0pinmux";

fragment@0 {
target = <&am33xx_pinmux>;
__overlay__ {
spi0_pins_s0: spi0_pins_s0 {
pinctrl-single,pins = <
0x150 0x30 /* spi0_sclk, P9_22, MODE0 */
0x154 0x30 /* spi0_d0, P9_21, MODE0 */
0x158 0x10 /* spi0_d1, P9_18, MODE0 */
0x15c 0x10 /* spi0_cs0, P9_17, MODE0 */
>;
};
};
};

fragment@1 {
target = <&spi0>;
__overlay__ {
#address-cells = <1>;
#size-cells = <0>;
cs-gpios = <0>;

status = "okay";
pinctrl-names = "default";
pinctrl-0 = <&spi0_pins_s0>;

spidev@0 {
spi-max-frequency = <24000000>;
reg = <0>;
compatible = "linux,spidev";
};
};
};
};

```

BB-SPI1-01-00A0.dts (SPI1 driver)

```

/dts-v1/;
/plugin/;

/* SPI1 */
/* D1 Output and D0 Input */

/ {
compatible = "ti,beaglebone", "ti,beaglebone-black";

/* identification */
part-number = "spi1mux";

```

