Western Michigan University
ScholarWorks at WMU

Honors Theses

Lee Honors College

4-16-2019

Communication Debugging Platform

Husam Beitello
Western Michigan University, hbeitello@gmail.com

Follow this and additional works at: https://scholarworks.wmich.edu/honors_theses

 Part of the Other Electrical and Computer Engineering Commons, and the Systems and Communications Commons

Recommended Citation

Beitello, Husam, "Communication Debugging Platform" (2019). *Honors Theses*. 3083.
https://scholarworks.wmich.edu/honors_theses/3083

This Honors Thesis-Open Access is brought to you for free and open access by the Lee Honors College at ScholarWorks at WMU. It has been accepted for inclusion in Honors Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact wmu-scholarworks@wmich.edu.

COMMUNICATIONS DEBUGGING PLATFORM

Written By: HUSAM BEITELLO, SCOTT BATZER, JOHN ROSS

Advised By: DR. JANOS GRANTNER, DR. DANIEL LITYNSKI

ELECTRICAL and COMPUTER ENGINEERING 4820

April 19th, 2019

WESTERN MICHIGAN UNIVERSITY

College of Engineering
and Applied Sciences

ABSTRACT

More and more products are becoming “connected” devices, integrating multiple communications subsystems into their design. These subsystems increase the complexity of a design, and require extensive testing and debugging before they reach an end user. The purpose of the project was to create a prototype for a peripheral device that enables bidirectional communication between a typical smartphone and another target device communicating using the infrared, Bluetooth, and/or CAN communication protocols. Such a device could then be used to debug the behavior of a product relying on these forms of communication, by running test procedures referred to as scripts from the smartphone, aiding in the development process of these products. The scope of the project also included developing a smartphone application to take advantage of the device, and proving its capability in some specific use cases.

This report details the process of developing, programming, constructing, and testing of the so dubbed “communications debugging platform” that fulfills the role described above. A full description of the design, including all hardware and software elements, are disclosed. In its current form, the communications debugging platform supports the necessary infrared, bluetooth, and CAN communication protocols, as well as the capacity for an unlimited number of other protocols. The mechanisms which allow for this are also described in this report.

DISCLAIMER

This report was generated by a group of engineering seniors at Western Michigan University. It is primarily a record of a project conducted by these students as part of curriculum requirements for being awarded an engineering degree. Western Michigan University makes no representation that the material contained in this report is error free or complete in all respects. Therefore, Western Michigan University, its faculty, its administration or the students make no recommendation for use of said material and take no responsibility for such usage. Thus persons or organizations who choose to use said material for such usage do so at their own risk.

WESTERN MICHIGAN UNIVERSITY
COLLEGE OF ENGINEERING AND APPLIED SCIENCES
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING
KALAMAZOO, MICHIGAN 49008

SENIOR DESIGN PROJECT REPORT RELEASE FORM

In accordance with the "Policy on Patents and Release of Reports Resulting from Senior Design Projects" as adopted by the Executive Committee Of the College of Engineering and Applied Sciences on Feb. 9, 1989, permission is hereby granted by the individuals listed below to release copies of the final report written for the Senior Design Project entitled:

PROJECT TITLE:_____

_____

PROJECT SPONSOR* Did this project have a sponsor? YES____ (see footnote) NO_____

Contact person and email address &/or telephone _____

Company Name_____

Design team has requested sponsor to verify in writing to course coordinator that all promised deliverables have been received. **YES____** **NO____** (please check)

TEAM MEMBERS NAMES:

NAME PRINTED	NAME SIGNED	DATE
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

* Those teams with a sponsor must have sponsor provide the **course coordinator** with written evidence that they have provided the sponsor with a copy of the final project report as well as with other items that the team has promised to the sponsor. The evidence could be a short note via email, fax or US mail from the sponsor indicating receipt of a copy of the report and all promised deliverables.

From Dr. John Gesink. Version 4 September 2013. © 2013 Damon A. Miller

ACKNOWLEDGEMENTS and PERMISSIONS

All members that worked on the communications debugging platform would like to thank Alex Bodurka, Kurosh Nahavandi, and Zane Shami for dedicating time to offer support and advice in the completion of this project. We would also like to thank Dr. Janos Grantner and Dr. Daniel Litynski for their support and direction during the same course of time.

Additionally, we would like to acknowledge Android developer Kai Morich for the development terminals he created for Bluetooth, UART, and WIFI communication that aided in the creation of our own application. It drastically hastened our development to have applications to compare our performance to, as well as to use as stand-ins for our own application to allow parallel development of our modules and application.

STATEMENT of NEEDS

The intent of the communications debugging platform was to streamline the development and debugging of communication subsystems for increasingly complex “smart” products. The project was sponsored by Stryker Medical, who requested a prototype for a smartphone-compatible device and partner application to facilitate the debugging of their own products. Such a system would allow their technicians and engineers greater flexibility when testing communication layers in their products, integrating the ever increasing computing power of smartphones with their portability, along with the flexibility of microcontrollers.

The communications debugging platform needed to communicate bidirectionally with any communication subsystem needing to be debugged, using the infrared, Bluetooth serial, and Bluetooth audio protocols and profiles. Stryker also required that an Android or iOS smartphone be used for user interaction, providing functionality for sending data out over any desired protocol, and receiving data over protocols in tandem. Any Bluetooth communication was also required to be handled by the smartphone, rather than a microcontroller. A block diagram of the layout of such a system can be seen below in Figure 1.

Stryker provided test “squawk box” hardware to verify every potential use case for the communications debugging platform, and the final version of the project was required to pass verification tests associated with the squawk box. Finally, the budget for a complete system was required to be under 750 USD.

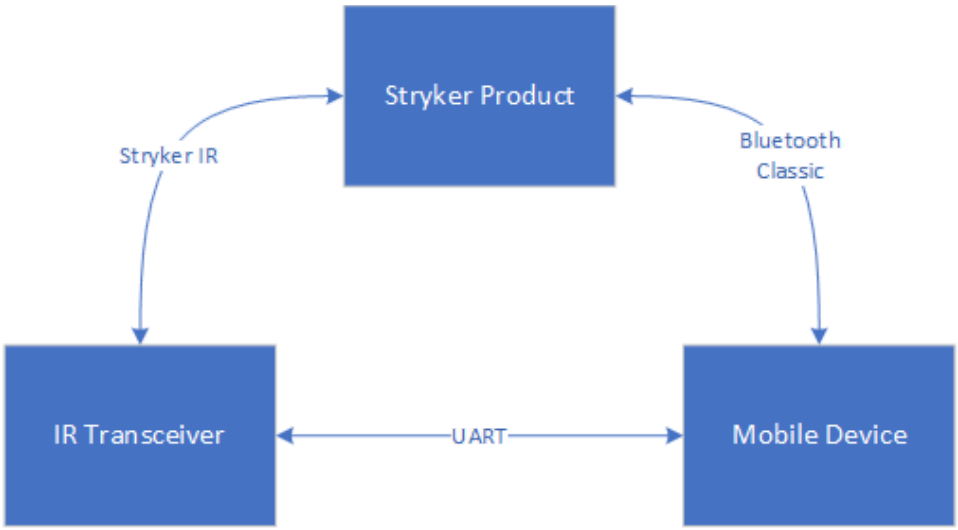


Figure 1: Block diagram showing how the project would be configured to meet the original needs of the sponsor, communicating to a product using infrared and Bluetooth.

ORIGINAL DESIGN

A complete list of the original specifications is shown in the next section, representing the initial plan for the communications debugging platform. This plan covered the complete life cycle for the product, and assumed an average user base of 25 Stryker engineers and technicians. Several of these specifications were chosen to emphasize the hand held nature of the design, such as not getting too warm during operations, having on board power and charging, and having an enclosure designed to mount on the smartphone for ease of use.

In addition to specifications designed to meet the needs of the sponsor, specifications were created to expand the project into a more generic debugging system. The infrared transceiver was specified to be wrapped into its own separate module circuit, and another module circuit was created for handling CAN communication. The connection between those modules and the phone was specified to be handled with an intermediary core tool circuit. More specifications were added to further enhance the modularity of the debugger, including a uniform physical connection between the tool and modules, as well as 3.3V and 5V power and communication using this connection. By expanding upon the original project this way, many more module circuits could be created for additional communication protocols, with minimal change to the existing architecture.

While the original software application needed only a static configuration for communication using Bluetooth and infrared and ran a fixed program, the new universal debugging platform would benefit greatly from a method to define these connections and programs dynamically during runtime. To implement such a feature, specifications were added to store such information in scripts which would be loaded once the application was started, and changed at will. These scripts were also required to be hosted from a remote database, so that multiple devices could be used at once.

Shown below in Figure 2 is an expanded block diagram, demonstrating how the system would be configured to meet our expanded specifications. Note that more modules are shown connected simultaneously to the core tool, demonstrating how the platform can be expanded to use more communication protocols.

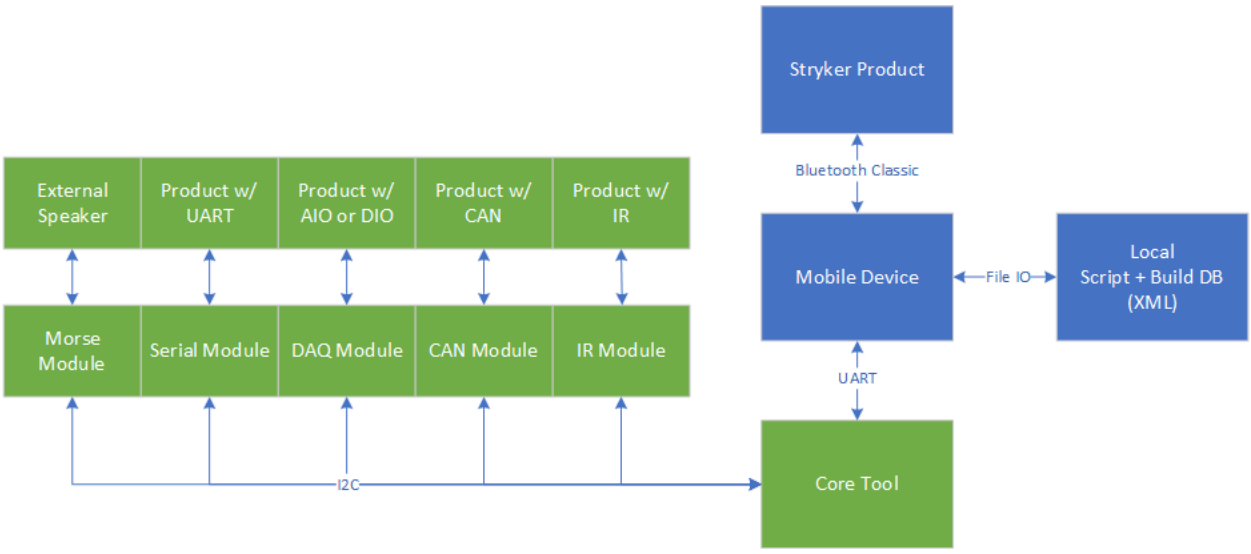


Figure 2: An expanded block diagram showing a configuration that meets all the specifications laid out in the following section

Given the specifications for the power and communication, the below Figure 3 shows the initial plan for the circuits and components needed to meet our specifications.

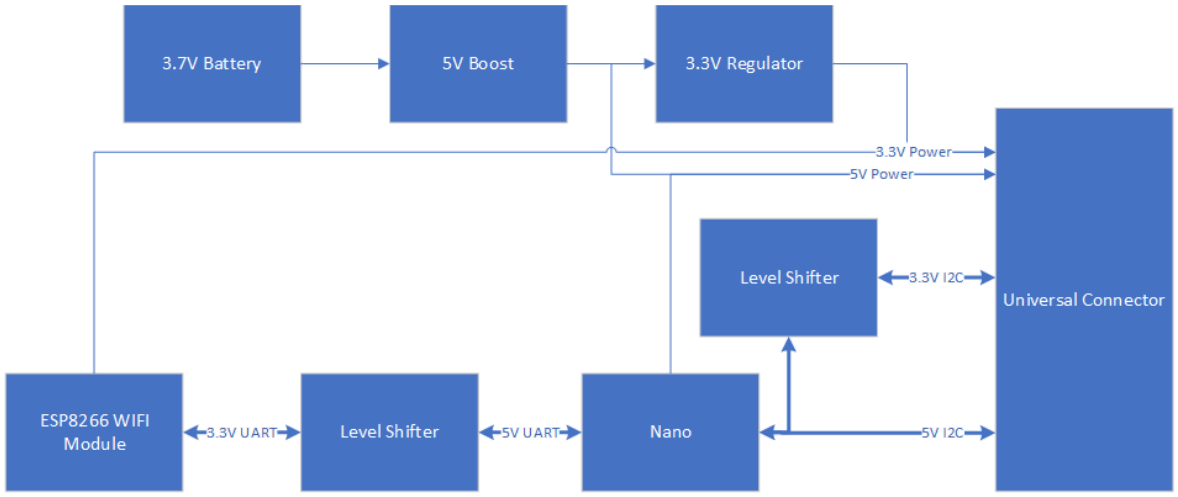


Figure 3: A block diagram laying out the major circuit components needed to fulfill the power requirements and communication requirements for the system.

Below in Figure 4 is the original line drawing for the communications debugging platform and smartphone, demonstrating our initial vision for how the tool would rest relative to the smartphone, as well as the modules.

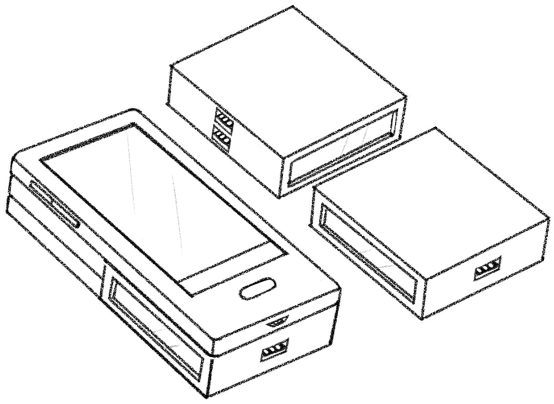


Figure 4: A line drawing for the original concept for the communication debugging platform. Showing how the tool and modules would rest against the phone (left) when a clamp was used, as well as isometric views of where the ports were expected to lie (top and right).

ORIGINAL SPECIFICATIONS

1. **Physical Characteristics**
   - 1.1. Enclosure
     - 1.1.1. Tool has 3D-printed enclosure
   - 1.2. Ergonomics
     - 1.2.1. Tool must have a mechanism for clamping onto the mobile device
     - 1.2.2. Tool should be ergonomic to hold
     - 1.2.3. Tool must not be too heavy to hold comfortably
     - 1.2.4. Tool must not get too hot to hold
   - 1.3. Materials
     - 1.3.1. No toxic materials used in development
2. **Functionality**
   - 2.1. Tool Functionality
     - 2.1.1. Tool must use a readily available development board (i.e Arduino, Raspberry Pi)
     - 2.1.2. Development board on tool must be capable of reading/writing serial data over a USB Type A port connecting from Lightning, USB Type-C, and Micro USB ports
     - 2.1.3. Tool has an OLED display fixed at the base of the tool
     - 2.1.4. The OLED display on tool should display bus load, name of loaded script, battery information, and network connection status
     - 2.1.5. Tool must have onboard battery and charging circuitry
     - 2.1.6. On board battery must be able to supply power for at least 12 hours
     - 2.1.7. Tool must be constructed such that all circuitry is fixed in place
   - 2.2. Tool to Phone
     - 2.2.1. Tool has a USB type A port for interfacing with mobile device
   - 2.3. Tool to Peripheral Device
     - 2.3.1. Tool must have a uniform connector for attaching additional modules
     - 2.3.2. Tool supports an additional module for charging onboard battery via 5V micro USB port
     - 2.3.3. Modules must be constructed such that all circuitry is fixed in place
     - 2.3.4. Module connector must have capability for 3.3/5V power and 5V communication
   - 2.4. Included Modules
     - 2.4.1. Tool must be packaged with IR communication module
     - 2.4.2. Tool should be packaged with CAN communication module
   - 2.5. Phone/App Functionality
     - 2.5.1. App should talk to a remotely hosted database
     - 2.5.2. App must display the infrared data read from the tool
     - 2.5.3. App must support scripting for commanding/monitoring via Wired/Bluetooth interface
     - 2.5.4. App must support Bluetooth communication to Stryker products via SFP, A2DP, HFP
     - 2.5.5. App must be iOS or Android compatible

2.5.6. App should be cross-platform compatible
2.5.7. App should be able to run on a Bluetooth enabled Windows PC with the tool connected via USB

3. Supplementary Materials

3.1. Tool must be packaged with a cord capable of data transmission from a USB Type A connector to a micro USB, USB Type C, or Lightning connector
3.2. Tool should be packaged with manual with instructions on accessing remotely hosted database and managing user scripts
3.3. Tool should be packaged with documentation of interchangeable communication between tool and modules to allow creation of new modules

4. Disposal

4.1. Tool and module enclosures should come apart for easy electrical component salvage or disposal

5. Applicable Standards

5.1. IEEE 802.15.1 Standard for Wireless Personal Area Network defines standards for Bluetooth frames

6. Constraints

6.1. Entire system (tool and necessary modules) must coss less than $750

DESIGN EVOLUTION

As needed changed and problems were encountered during the course of the project, the original design was altered several times. This section details all the significant improvements and changes made to the original design described previously.

Hardware Evolution

The hardware aspect of the design went through several iterations over the course of this project. There are three main design iterations, broken up into generations where major hardware changes were implemented. Generation one directly implements the original design, while generation two included several branching paths for the hardware design. This was deemed necessary to ensure that the minimum viable product would be completed in time, and original specifications were met. This generation included four designs, created and tested in parallel. A third and final generation, wherein we completed a fully featured prototype for the communications debugging platform with all of our primary and stretch goals implemented.

Generation One

The initial design needed to ensure all hardware specifications were met, but we also wanted to create a design for testing and verification of the individual components and software.

To ensure each component system of the initial design met its own specifications, testing points for each subsystem were included. These testing points would allow the operator to test the voltage and I2C bus behavior granularly, without having to observe a fully working system. Other features were included for testing and debugging purposes, including a bypass circuit to ensure that the device could run off of external power in the case that an internal power rail was not functioning.

The 5v booster and the 3.3v regulator circuits would generate the power rails needed for running the tool and connected modules. Each of the components and supporting circuitry draw power from a lithium polymer battery. For the initial design, the TPS61202 5V boost buck convertor was chosen for the 5v booster, and the ADP150-3.3 Linear Regulator was chosen for the 3.3v regulator.

At the sponsor's request, an enable circuit using two N-channel MOSFETs was designed. These two transistors would act as gates for the 5v booster and the 3.3v regulator, only allowing these parts to draw power from the battery if the mobile device was connected using a USB2.0 cable supplying an enable signal.

For the power supply of the system, a 2000 mAh capacity lithium polymer battery was chosen. A battery of this capacity would allow the core tool to operate for longer than the 12 hour minimum battery life originally laid out in the specs, given the power demands of the rest of the system. A charging circuit using the LTC4001 2A Synchronous LiPo charger was designed to allow for

simultaneously using the device and charging the battery. This charging circuit would also provide power at a safe charging speed of 1C, utilizing internal methods of battery and circuit protections in case of component or battery failure.

Communication between modules and the core tool was handled over a universal connector, implemented using the USB3.0 connection standard. The 9 pins in a USB3.0 connection allow for shared 5v and 3.3v power rails between the tool and modules, as well as parallel 5v and 3.3v I2C busses.

Several trace connection issues and component failures were encountered during the verification of the generation one circuit board. The 5V boost circuit failed verifications due to what is believed to be heat failure from the soldering procedure. To ensure a functional prototype for the hardware would be completed in time, four parallel paths were designed to test multiple options for the core tool circuit at once; the Minimum Viable Product (MVP) PCB design, a modular PCB design, a direct iteration of the initial PCB design, and the prototype board for implementing stretch goals.

Generation Two

Several changes from the initial design were implemented across the four parallel paths. One issue with the initial design was the 5v booster IC and battery charging IC. Both were quad flat no-lead components, which were difficult to affix to the PCB properly without applying heat damage. Another issue encountered with the first revision was a bridged trace between the enable circuit and battery input, which was fixed in Revision 2 and subsequent builds.

After failing to verify the 5v boost component integrity, an off the shelf 5v boost module was implemented into both the Modular and Revision 2 schematics. For the battery charger, another off the shelf module using a TP4065 standalone lithium ion battery charging IC was used. This module was included in all but the Revision 2 schematic.

As part of the specifications for the enable circuit and power saving, two N-channel power FETs were configured to gate the power supply to the 5V boost and 3.3V regulator. When the core tool is connected to the smartphone, 5V is intended to be supplied on the VCC pin. This is used to turn on both MOSFET devices, allowing our 3.7V battery to power the two modules. It was discovered during testing that Android devices do not automatically supply power to external devices, and there is no software option to enable this. Instead, an attachment called an On-The-Go (OTG) connector was used. This connector pulls the ID pin found in the micro USB plug to ground, which the Android device recognizes, resulting in power being supplied on the VCC pin, rather than accepted. To ensure we could continue to use the same data and power cable as in the previous revisions of the core tool, we were required to change the core tool USB port from USB 2.0 type A to a micro USB female connector. This is shown visually below, in Figure 5.

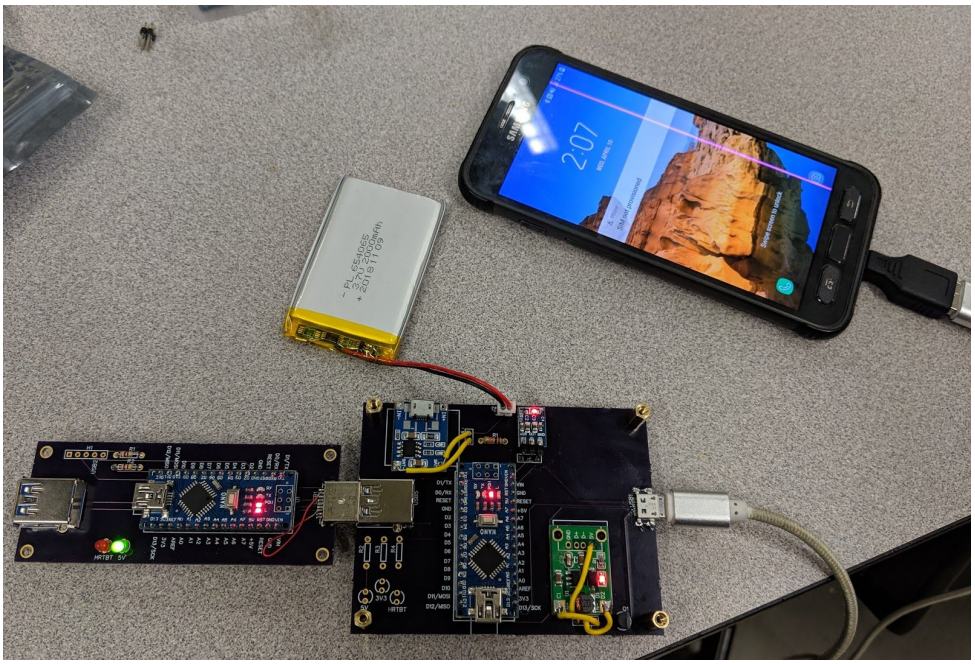


Figure 5: The core tool is shown interfacing with a typical Android smartphone using the OTG connector attached directly to the smartphone.

Although the Modular, MVP and final prototype boards did met specifications, Revision 2 still had issues with the battery charging IC that was used in the initial design. Moving forward, it was determined that several stretch goals could be implemented in the final prototype, including the ESP8266 WIFI module.

Final Design

The final design for the core tool was based on the design built on the prototype, as it met all of our core and expanded specifications nicely. The WIFI module was added to the prototype to allow for wireless communication between the application and tool. This was added to the design because the device running the application would not be able to communicate over UART and SPP simultaneously. Figure 6 shown below demonstrates how the addition of the WIFI module alters the communication pathways for the platform.

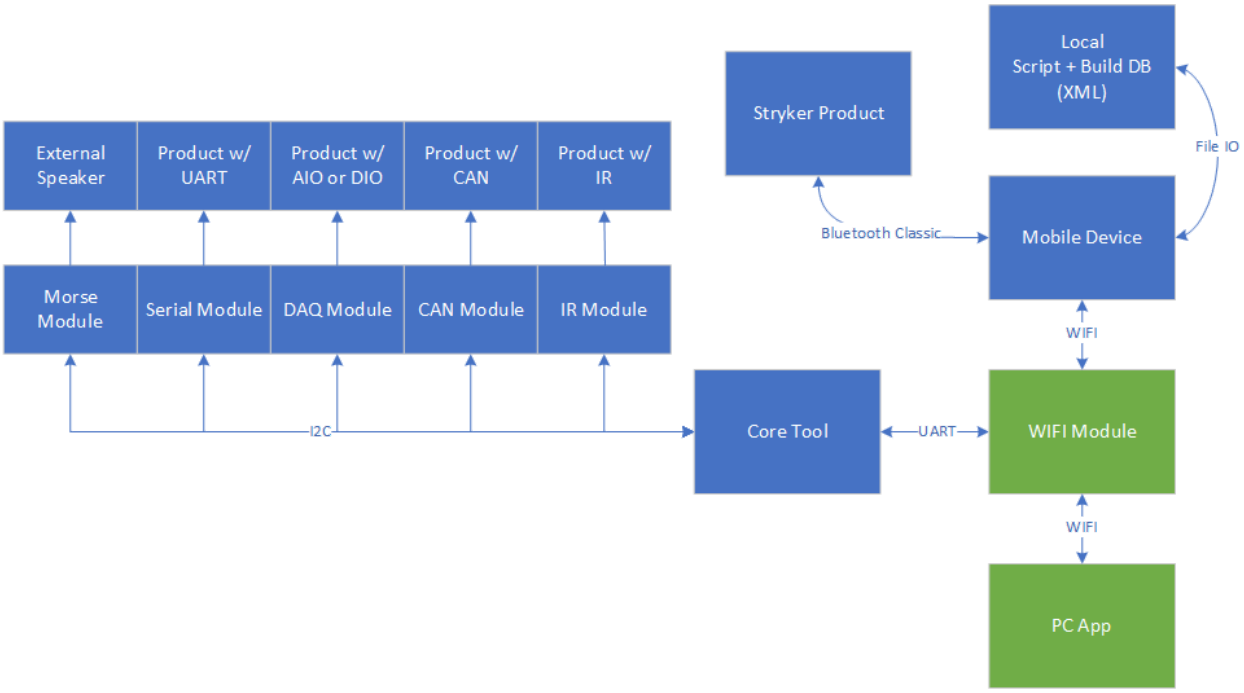


Figure 6: Finalized block diagram for the communication debugging platform, showing the addition of the WIFI module in the tool to phone communication.

Software Evolution

The primary front-end of the application consists of two views; one for selecting a script, and one for interacting with each script once selected. This feature remained relatively unchanged throughout the evolution of the software, while the majority of our software evolution manifested as reaching for a variety of stretch goals, such as expanding the capability of the tool to involve WIFI when that was considered previously to an extremely unlikely pursuit.

Initially, scripts were written in the JSON format, to remain uniform with other information stored on the database such as build versions. This was determined to be too cumbersome to maintain from a data structure perspective, as it was recognized early on that the format of our scripts would likely change as it was used more and more. As the use of a serializable data format requires a corresponding inheritance driven architecture, maintaining that structure would not be feasible and would add little benefit. As a result, one of the early changes to the design to address this issue was to shift off of JSON to a custom format wherein each command was separated with a new line.

The target platform shifted drastically, with the original design being solely on Android, which then shifted towards NodeJS using the Electron and Cordova frameworks to provide cross-platform capability. It was then realized that, when provided more information about our specifications, that the NodeJS iteration would not be feasible. This resulted in a shift back to Android, until later the WIFI capability of the tool would allow for a PC app, and given more time a truly cross-platform application.

The WIFI enabled tool was originally intended to be a variant of the core tool that - instead of using a Nano to interface with the module chain over I2C - provided great advantages in both user experience and data rates. It was later found that the ESP8266 WIFI-capable development board used, with the libraries provided to interface with the Arduino environment, was not actually capable of interfacing to an I2C bus as a slave, which was required for the desired behavior. By having the ESP8266 share a UART channel with the Arduino Nano and act as a TCP/IP to UART passthrough, however, wireless capability could be added to the core tool while maintaining the original I2C bus behavior.

Initially, the remotely hosted database was attempted in MySQL Firebase was ultimately used instead of MySQL for storing scripting information as well as software build information. This was especially natural because of the built in libraries for handling this connection because of the use of the Android platform. While not a specification, we originally desired to have data logging capability to the remotely hosted database. While we moved to Firebase for the remotely hosted database, we ultimately found that MySQL for just data logging would be cumbersome, and thus left the data logging file as a direct time-stamped log local to the mobile device. This is simpler to maintain, and because these files can be easily sent off using email, it results in a simpler experience. This implementation also operates even when the user does not have an internet connection, which is especially important when taking advantage of the WIFI enabled tool, as the phone must disconnect from any Internet connected WIFI access points to then connect to the tool.

Scripting Lexicons

The following section details the formatting and command structure of the script files. Using these commands, the programmer has full control over the input and output behavior of the system, allowing for declaration of data sources and data sinks, sequencing transmit and receive events, and defining conditions for each event.

Table 1: GUI Script Lexicon

`GUI TXT sppTxSrcTxt STR sppTxSrc`	Creates a textbox with name sppTxSrcTxt that when text changes updates the register sppTxSrc
`GUI LBL sppTermLbl LOG sppSink`	Creates a scrolling log label with name sppTermLbl that updates when register sppSink is changed
`GUI LBL failLbl UPD failFlag`	Creates a label with name failLbl that will update when register failFlag is changed
`GUI LBL val1snk2`	Creates a label with name val1snk2 - to be used by GUI BTN SYC script command
`GUI BTN wifiConnect Connect`To`WiFi WIFICNT`	Creates a button with name wifiConnect with text "Connect To WiFi" to connect to WIFI
`GUI BTN usbConnect Connect`To`USB USBCNT`	Creates a button with name usbConnect with text "Connect To USB" to connect to USB
`GUI BTN sppConnect Connect`To`SPP SPPCNT macAddr`	Creates a button with name sppConnect with text "Connect To SPP" to connect to SPP with a MAC address specified by register macAddr
`GUI BTN sppTransmit Send`SPP SPPTX sppTxSrc`	Creates a button with name sppTransmit with text "Send SPP" to send the value in register sppTxSrc over SPP
`GUI BTN usbTransmit Send`Serial`Message USBTX SERIAL serialTx`	Creates a button with name usbTransmit with text "Send Serial Message" to send the value in register serialTx over USB to a SERIAL module
`GUI BTN bbidReqBtn Request`BBID WIFITX IR bbidReqCmd`	Creates a button with name bbidReqBtn with text "Request BBID" to send the value in register bbidReqCmd over WIFI to an IR module
`GUI BTN val1snk2push Sink`2`Push SYC val1snk2 val1src`	Creates a button with name val1snk2push with text "Sink 2 Push" to set the value in register val1snk2 to val1src
`GUI BTN initCmdList Send`Cmd`List CMD sampleList`	Creates a button with name initCmdList with text "Send Cmd List" to execute the command list sampleList
`GUI BTN playfanfare Play`Fanfare AUDIO fanfare`	Creates a button with name playfanfare with text "Play Fanfare" to play mp3 file fanfare*
`GUI GPH speedGraph SRC motorSpeed 0 100 0 1750`	Creates an XY graph named speedGraph with register source motorSpeed with X dimensions 0 to 100 and Y dimensions 0 to 1750

* For the mp3 file to be played, it must be added as a raw and launched with the application

Table 2: REG Script Lexicon

REG canRx String WIFISRC CAN	Defines a String register canRx that is sourced over WIFI from a CAN module
REG sppSink String SPPSRC	Defines a String register sppSink that is sourced over SPP
REG macAddr String TOOLSRC SERIAL	Defines a String register macAddr that is sourced over USB from a SERIAL module
REG motorSpeed Float PAYLOAD CAN canRx 706 0 4	Defines a Float register motorSpeed that is sourced via a canRx register with payload defining it is from CAN on PDO 706 starting at byte 0 with length of 4
REG msg1 String PAYLOAD SPP sppSink 1	Defines a String register msg1 that is sourced via a sppSink register with payload defining it is from SPP with target key 1
REG resSET2 String INIT 5501020100AA	Defines a String register resSET2 with initial value “5501020100AA”
REG sppTxSrc String	Defines a String register sppTxSrc

Table 3: CMD Script Lexicon

CMD RES msg1 SPPTX resSET2 DEL 1000	Defines a response to register msg1 that will send the value in register resSET2 over SPP after a 1000ms delay
CMD SPP ACK STRING	Configures SPP to ACK in STRING mode
CMD SPP ACK BYTE 1 2 3 4 55 170	Configures SPP to ACK in BYTE mode and defines, respectively, type, key, length, data indices as well as the start and end values
CMD ELE sampleList SPPTX SET`1`ON RX 2500 sppSink REP	Appends a command element to sampleList to send "SET 1 ON" over SPP and expect an ACK after 2500ms on register sppSink or repeat, and if received move to next command element
CMD ELE sampleList SPPTX SET`2`ON DEL 1000	Appends a command element to sampleList to send "SET 2 ON" over SPP and move to the next command element after a 1000ms delay
CMD ELE sampleList SPPTX SET`3`ON RX 2500 sppSink FAIL failFlag	Appends a command element to sampleList to send "SET 3 ON" over SPP and expect an ACK after 2500ms on register sppSink or set a failFlag to fail
CMD CND motorSpeed ABV 1250 AUDIO fanfare	Attaches a conditional event to register motorSpeed to play audio file fanfare when value transitions above 1250*
CMD CND motorSpeed BEL 750 AUDIO jump	Attaches a conditional event to register motorSpeed to play audio file jump when value transitions below 750*

* For the mp3 file to be played, it must be added as a raw and launched with the application

Module Instruction Set

IR Module - When receives any message over module interconnect, will continuously search for first IR enabled product and ping it until it receives an ID

Morse Module - Plays any string received over module interconnect on speaker

UART Module - Simply connect via UART terminal to allow UART channel connection to module interconnect

Table 4: DAQ Module command examples

"0,1,0"	Set Digital Output pin 1 Low
"1,2,1"	Set Digital pin 2 to be an Output pin
"2,0"	Start listening to Analog pin 0 input
“3,500”	Set input sampling delay to 500ms

Table 5: CAN Module command examples

"0,0123,1122334455667788"	Write command to PID 0123 with payload 1122334455667788
"1,3210"	Tells the module to start listening to PID 3210
"2,3210,5,0,3,F"	Tells the module to start listening to 3210, but also sets that an average filter will be applied with a 5 sample count size, and the slot being used for this in the payload starts at byte 0 and ends at byte 3 and is of datatype float

Setup Android App

In the cdp/setup directory is a android-studio-ide-181.5056338-windows.exe file. Run and install this. You might need to update the version of your JRE and JDK in order to get proper ADB connection to the tablet. Once Android studio is installed and verified to operate, use the Open Window command to the cdp/cdp_app directory to open the app in Android Studio.

Once this is done, the tablet or mobile device needs to be configured for USB debugging. For our device, this was done by accessing the “About tablet” page in the Android Settings and tapping “Build number” seven times in order to access the developer options. Once the developer options was accessible, under Debugging check “USB Debugging”, and this might not be required for all devices but we needed to set “Select USB Configuration” to “Charging” in order for it to be detected on ADB.

Setup NodeJS Tools

In the cdp/setup directory is a node-v10.14.2-x64.msi file. Run and install, then open CMD prompt and run the following few commands.

```
node install npm
npm install firebase
```

This will allow the user to use the NodeJS tools described in the scripting overview section.

Revised Specifications

Several specifications that do appear in the original specifications were changed over the course of the project to meet changing demands and resolve issues with the design.

This includes things such as specification 2.2.1, which was changed to require a micro USB port rather than a USB type A port. In the original plan, it was assumed that whatever port existed on the phone (a micro USB in our test cases) could be configured to output voltage to power on the tool using software, allowing the tool to connect to the phone using any data-capable USB Type A to phone port cable. These are usually ready at hand for the smartphone user. Instead, the USB On-The-Go standard requires a specific hardware adapter to enable output voltage from the phone port, which converts the phone port to a USB Type A port.

Additionally, once specifications that satisfied core needs were met, our sponsor imposed additional specifications to ensure that any additional features we included in the design would be robust and useful to them.

This included the need for a bypass circuit which would allow the tool to run off the same power source used to charge the battery, for when no battery was connected or a fault occurred local to the charging circuitry or 5V boosting circuitry.

The project sponsor included more stringent requirements for the SPP and Bluetooth Audio specifications that made our original design path unfeasible, which ultimately forced us down the path of our design being Android exclusive. These changes included the requirement of a variable passkey for Bluetooth classic connection as well as configurable response to SPP communication. The data format for the SPP was also required to be changed from strictly text based to byte-array formatted.

We were also given a Stryker interface board near the end of the project for development, requiring additional development to verify the operation in that environment, which was out of scope for this project previously. Similarly, we also have a CAN module verification using the Stryker motor control board that was previously out of scope.

There was also certain misunderstandings with the Stryker provided data formats, such as the documentation provided implying all SPP communication would be string based, requiring a later alteration to how SPP communication happens. The system included this as an option to allow for more flexibility, with a scriptable flag controlling with mode SPP messages will operate in. This is explored in more detail in the corresponding verification section.

DESIGN AND PERFORMANCE MEASUREMENTS

Knowing that a design is ultimately created to meet a list of specifications, the sections that follow detail how each specification for the project was met as part of the current iteration of the design.

Specification Analysis - Hardware

2.1.1 Tool must use a readily available development board

The Arduino Nano was chosen as the primary development board for its versatility, size, and low power requirements. Having access to the Arduino IDE for programming was also an advantage. The Arduino Nano used on the tool was tasked to manage the detection and addressing of connected modules to the I2C bus, and to convert bidirectionally between addressed I2C messages on the bus and UART communication with the mobile phone. Each communication module connects to the tool-module I2C bus using an Arduino Nano as well, converting addressed I2C messages into whatever protocol that module was designed to handle, and sending out periodic heartbeats to stay connected to the tool.

2.1.3 Tool has an OLED display fixed at the base of the tool

This specification has evolved over the course of the design project. Initially it was determined that having tool feedback such as battery voltage and script information would be important to the device user. An OLED display was specified to fulfill this role, as it would allow the Arduino Nano to graphically display this information while remaining compact. The current design still includes the OLED display, but the same desired functionality was also implemented into the application design, knowing that the smartphone/tablet display will be a far more convenient point of interaction for the user. The component choice for the OLED display was I2C addressable, and the relevant four pin header for power and communication can be found on the PCB layouts.

2.1.4 The OLED display on tool should display bus load, name of loaded script, battery information, and network connection status

In order to have an OLED display these metrics, a separate I2C bus was emulated using pins D11 and D12 on the Arduino Nano, which are PWM enabled. A separate I2C bus was created to ensure that the OLED driver communication remained isolated from the tool to module communication. A test program which takes advantage of the true and emulated I2C busses simultaneously can be found in Appendix C under the OLED test section. This program bounces an integer between two Arduino Nanos over the true I2C bus, but with one endpoint of the bus also driving the OLED display and showing its stored value for the integer. A physical implementation of the test program is shown below, in Figure 7.

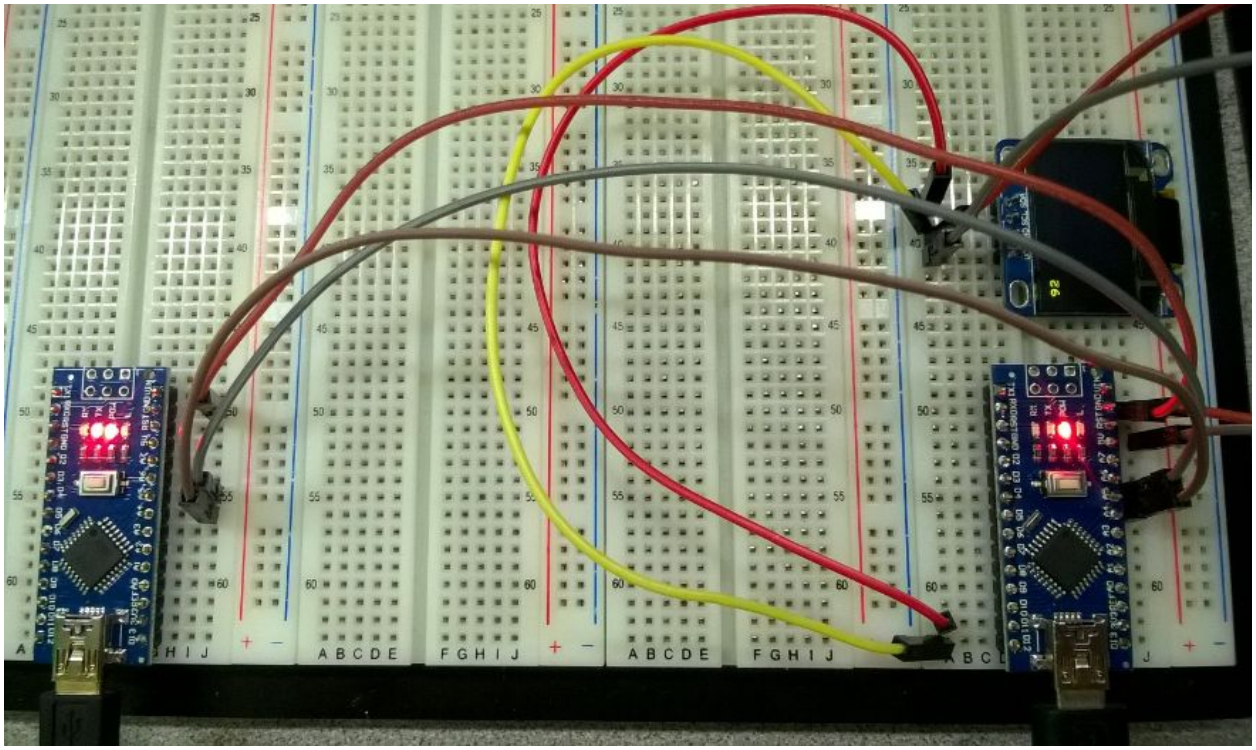


Figure 7: Physical implementation of two Arduinos acting as a counter over an I2C bus with a separate software-driven I2C bus handling an OLED display

2.1.5 Tool must have an onboard battery and charging circuit

The current design meets this specification using a 3.7V, 2000mAh lithium ion battery and the TP4056 standalone battery charger. While the TP4056 itself comes in an 8 pin SMD package, a breakout board was incorporated in the modular MVP and final prototype boards that provided a complete charging circuit, with status LEDs and a micro USB port for charging. The device can be seen below in Figure 8, mounted on our modular MVP PCB.

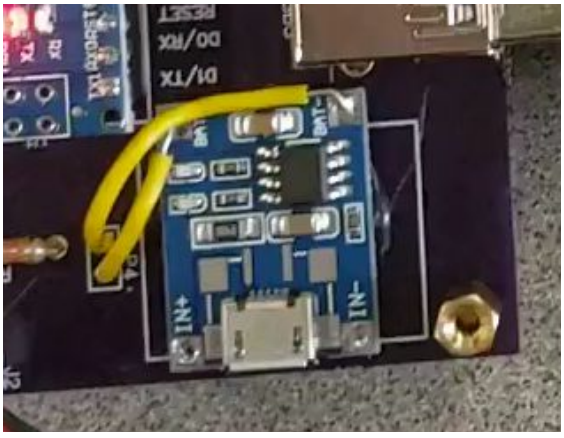


Figure 8: The TP4056 standalone battery charger found on the modular MVP core tool design.

2.1.6 On board battery must be able to supply power for at least 12 hours

Through testing all of our second generation tool designs, we found the 2000mAh capacity battery lasted over 12 hours in all cases. Expected battery life was determined by measuring the current drawn by the core tool when powered on and connected to a smartphone. The measured currents and battery life results can be found below in Table 6. With this verification, we know that our battery and power design meets all meet specifications.

Table 6: Power Draw and Battery Life for Revision 2 Tool Designs

Revision	Power Draw (mA)	Expected Bat. Life (hr)
Revision 2	73	27.4
Final Prototype w/o WIFI	34	58.8
Final Prototype w/ WIFI	120	16.7

2.1.7 Tool must be constructed such that all circuitry is fixed in place

All circuitry used was mapped out onto PCBs for each revision of the hardware, such that it could be soldered or (in the case of the breakout boards for the battery charger and 5V booster) glued into place. Similarly, the PCB itself is mounted to the inside of the enclosure using standoffs.

2.2.1 Tool has a USB type A port for interfacing with mobile device

This original specification has since been changed to reflect our better understanding of the USB interface on the typical smartphone. Rather than requiring a USB Type A port, the specification now requires a microUSB port. The USB Type A port was originally required so that we could take advantage of the typical smartphone data and power cables that terminate in a USB Type A plug, however this would not result in 5V power being supplied by the phone on the VCC pin. Instead, an OTG adapter would connect directly to the phone to enable the power supply, which itself terminates in a USB Type A port. Thus, to make sure the same cable would still be useable, the port on the tool changed, and was implemented in the modular MVP board.

2.3.1 Tool must have a uniform connector for attaching additional modules

The uniform connector was implemented using the USB3.0 standard. On the core tool, there is a USB 3.0 Type A port that accepts a USB3.0 Type A plug from a module. Each module also has a port for daisy chaining another module. The pins were selected to be at least partially backwards compatible with USB2.0 connectors - using 5V power, ground, and the 5V I2C bus. The 3.3V I2C bus and 3.3V power lines are unavailable when connecting with a USB 2.0 plug. The scheme for this connector is shown in Figure 9.

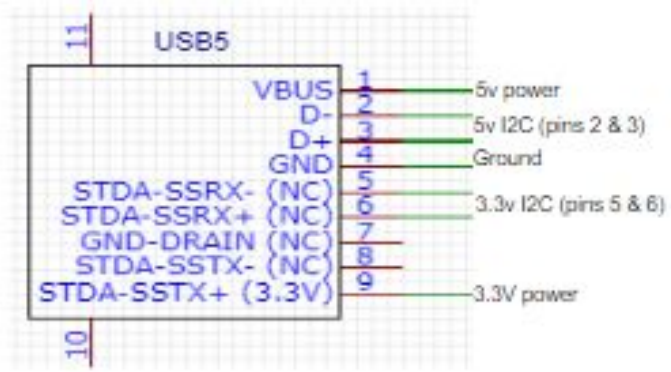


Figure 9: Pin Diagram for a USB 3.0 connector. The top 4 pins are shared with the USB2.0 standard are are used for 5V communication and power.

2.3.2 Tool supports an additional module for charging onboard battery via 5V micro USB port

The module used to charge the onboard battery via a 5V enabled microUSB port was the TP4056 breakout board. While the original specification might have envisioned a separate module PCB dedicated to charging the core tool, the TP4056 mounted directly on the core tool PCB fulfills all specifications with less concern for damaging our designed PCBs with overvoltage or overcurrent by connecting it along the daisy chain of modules.

2.3.3 Modules must be constructed such that all circuitry is fixed in place

Similar to specification 2.1.7, this specification was achieved by designing PCBs for each circuit, soldering components into place where possible, and also fixing the PCBs to enclosures using standoffs.

2.4.1 Tool must be packaged with IR communication module

As part of the final deliverables to the sponsor, a module capable of connecting to Stryker's proprietary IR transmitter/receiver will be included. This module translates infrared communication to and from I2C messages from the I2C bus connecting it to the core tool. A final prototype of the IR module can be seen below in Figure 10, being used with the WIFI core tool to read infrared data from a Stryker product.

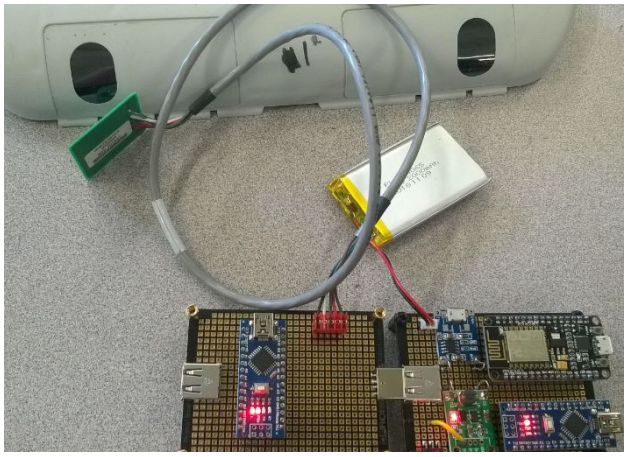


Figure 10: A demonstration showing the core tool (bottom right) interfaced with the infrared module prototype (bottom center) communicating with a Stryker product using an infrared transceiver designed by Stryker (in green top left).

2.4.2 Tool should be packaged with CAN communication module

Another module capable of CAN communication will also be delivered. Similarly to the IR module, this module converts I2C messages to CAN messages bidirectionally. Figure 11 shows the final prototype for the design.

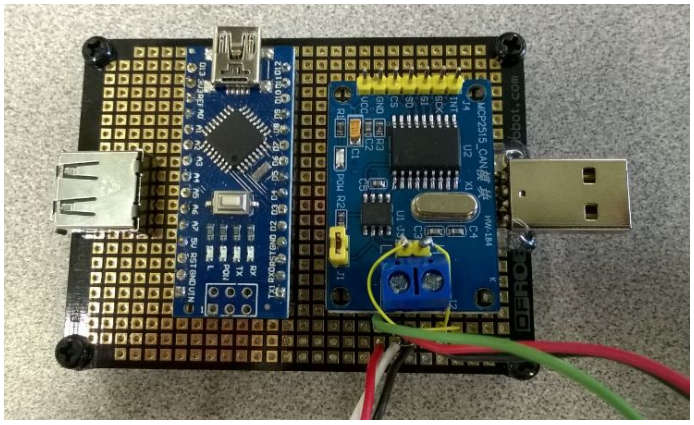


Figure 11: The final prototype for the CAN module.

Verification and Testing Procedures - Hardware

For the hardware testing, we went through several rounds of verification and performance measurements. The initial stage of verification was to verify individual systems would work, such as the 5V booster, battery charger and 3.3V regulator.

We chose the ADP150-3.3 linear voltage regulator Using LTSpice and a spice model provided by Analog Devices, we were able to verify that the linear regulation from battery power to 3.3V would work with the enable circuitry attached. In Figure 12, it was shown that when the enable pin is brought high by the enable MOSFET, the 3.7V input is brought down to 3.3V.

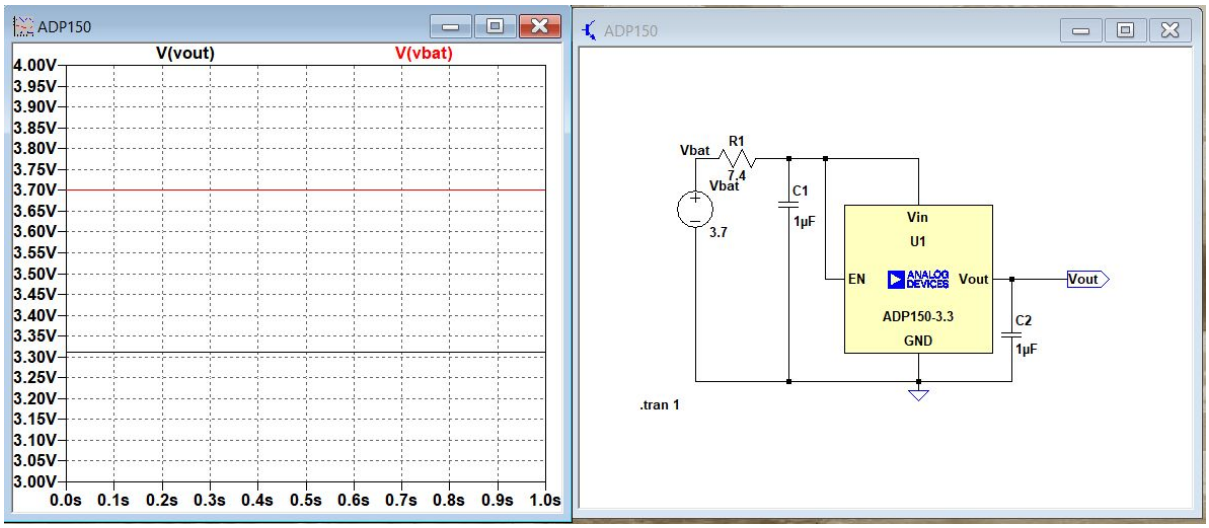


Figure 12: LTSpice verification that the 3.3V regulator performs as expected, for a 3.7V input (in red, left), we observe a 3.31 V output (in black, left)

For the battery charger, we purchased a developer kit for the LTC4001 2A Synchronous Buck Li-Ion Battery Charger from Analog Devices. Using the same configuration that the battery charger was designed for in our schematic, we were able to verify that the battery charger works as intended on the lithium polymer batteries that would be used for the project. Shown in Figure 13 is the results of the verification test where the battery voltage level is shown to be increasing from the nominal 3.7V to the 4.1V being read currently, as well as a charging current that is proportional to the charge in the battery.

The dev kit shown below was mainly used for testing and verification of our battery charging circuit. When comparing the values seen on our battery charger and the dev kit for the LTC4001, we noticed some discrepancies between the float voltages and power drawn from the input supply.

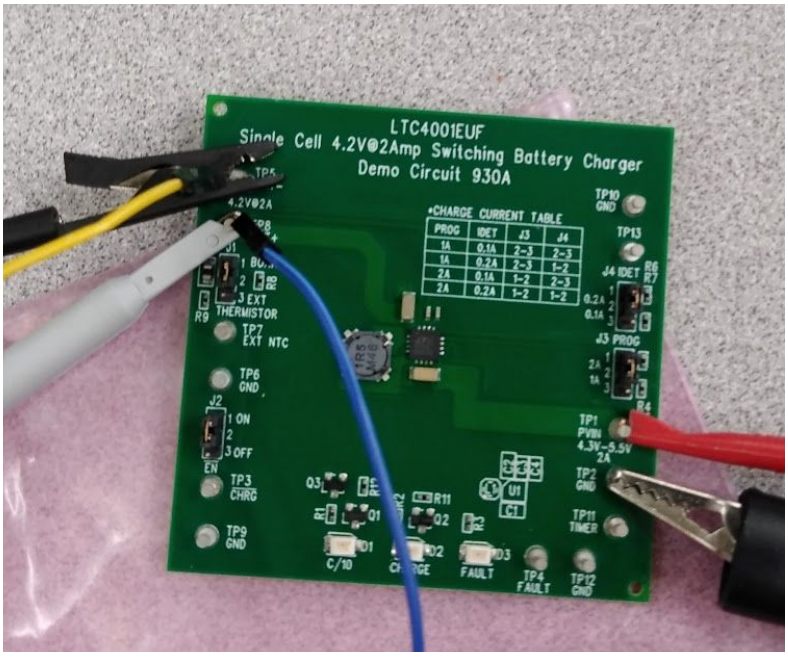


Figure 13: The LTC4001 development kit in the process of being tested.

For the 5V booster, we had to use Pspice to model the TPS61202. Due to complications with the PSpice model and code, we were unable to completely verify that the design would work. During the testing of the Revision one board, we were also unable to verify the component would work as designed due to complications with the integration of the IC with the PCB causing heat damage. This is what lead to the implementation of the 5v booster breakout board from Icstation.

As the design was iterated and changes were made to the design, new testing procedures needed to be written and carried out. These testing procedures were aimed at verifying the changes specific to the new design, such as the new modules that replaced the IC chips for the 5v boost, 3.3 regulator, and the battery charging circuit.

For the 5v boost module, the device needed to be tested over a wide range of input voltages to measure efficiency and consistency. Using a DC power supply, the input voltage was varied from 3.5v and 5v in 0.1v increments. The output was measured using a DMM capable of accurate results. The test results shown in Figure 14 verify that the 5v booster is outputting the expected values. It can be seen that as the 5v booster input increases, so does the output. For this projects scope, the output would not increase beyond the tolerances allowed by the Arduino Nano and the CAN module.

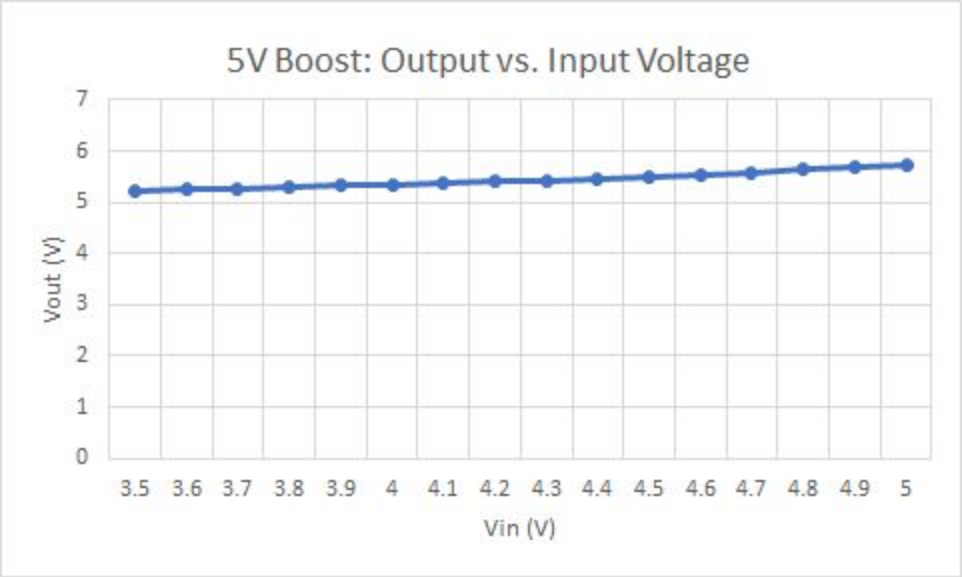


Figure 14: Input/Output verification for 5v booster module

The 3.3v regulator needed to consistently output 3.3v over a range of input voltages. Shown in Figure 15, the linear regulator is shown to consistently output 3.3v once the input voltage reaches ~4.4v. Because the 3.3v regulator draws its power from the 5v booster, this proves the 3.3v power rail is also generated correctly.

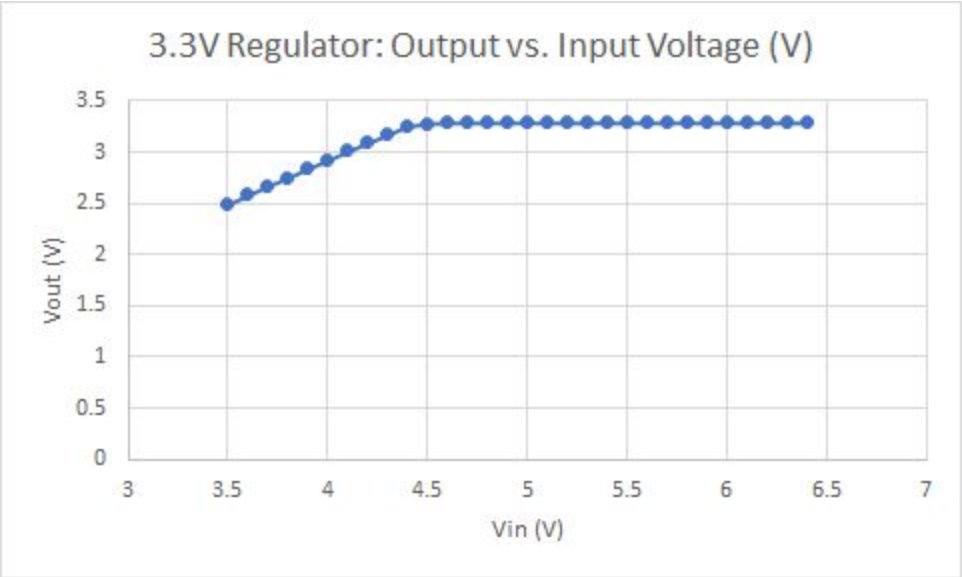


Figure 15: Input/Output verification for the 3.3V regulator.

With the power rail generation proven out, we also had to demonstrate bidirectional level shifting for the I2C busses. Shown below in Figure 16 is a test of the bidirectional level shifter used to convert between 5V logic and 3.3V logic.

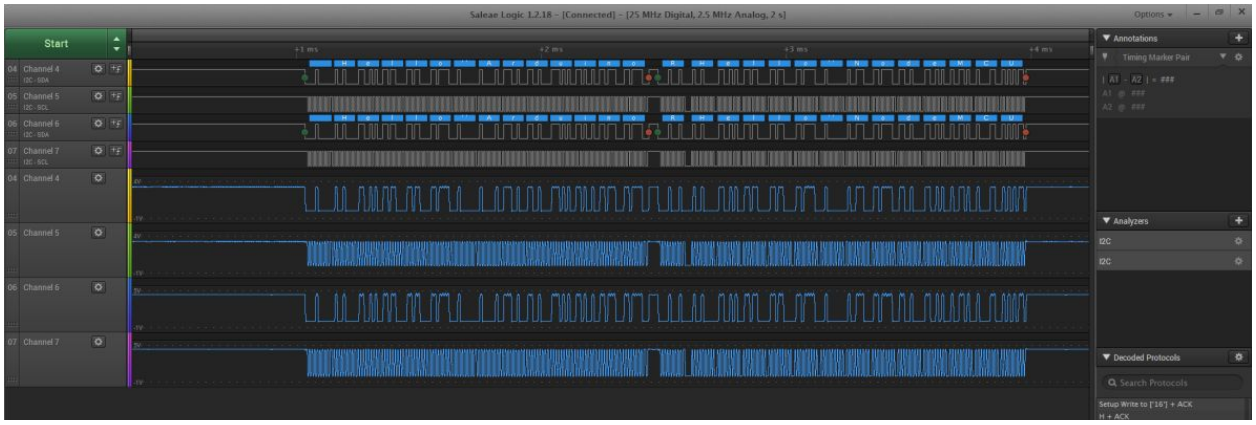


Figure 16: A screenshot showing the 3.3V and 5V I2C busses communicating in parallel, using a bidirectional level shifter.

The battery charging circuit needed to charge the battery at a reasonable speed while also terminating the charging sequence when the battery gains full charge. Using the battery charging module that implements the TP4056 battery charging IC, these requirements were verified. In Figure 17, the charging module is highlighted. When the charging cycle is complete, the LED will turn green, otherwise the LED will be illuminated red. In the figure, the LED is illuminated green, indicating that the charging cycle is complete and the charging current has dropped below the set limit, 120mA.

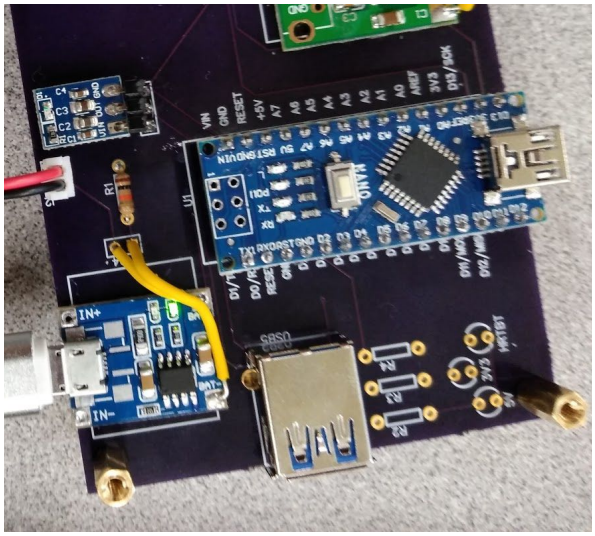


Figure 17: The TP4056 battery charger with a green LED illuminated, indicating a completed charge cycle.

To ensure that the the universal connector properly connected the core tool and and additional modules, each connection must be verified to connect the power rails to each microcontroller as well as connect the I2C lines for both 3.3v and 5v communication. The testing procedure involved ohming out each connection across the universal connector. This procedure caught several trace issues in the early revisions that were fixed in subsequent revisions.

The bypass circuit mentioned as part of the hardware design evolution could not be verified, due to issues with the circuit for the TPS61202 5V booster. Because this circuit was included as a debugging feature and was left off of the second generation of PCBs and beyond, attention was instead focused on features designed to meet specifications.

The following test image in Figure 18 serves as verification for several of our specifications. It depicts the revision 2 core tool and an IR module connected over the universal connector, sharing power and communicating. The green LEDS on both PCBs are brightly lit, indicating the 5V and 3.3V power rails are being generated correctly. Additionally, micro USB port on the far right of the core tool has been connected to a smartphone using an OTG connector, such that it supplied 5V to the enable circuit on the core tool. This verifies that the enable circuit works as intended as well. A test program was running on Nanos to verify that they were communicating using I2C. Because the red LED on the IR module's Nano is illuminated, the two modules successfully performed a handshake and can communicate.

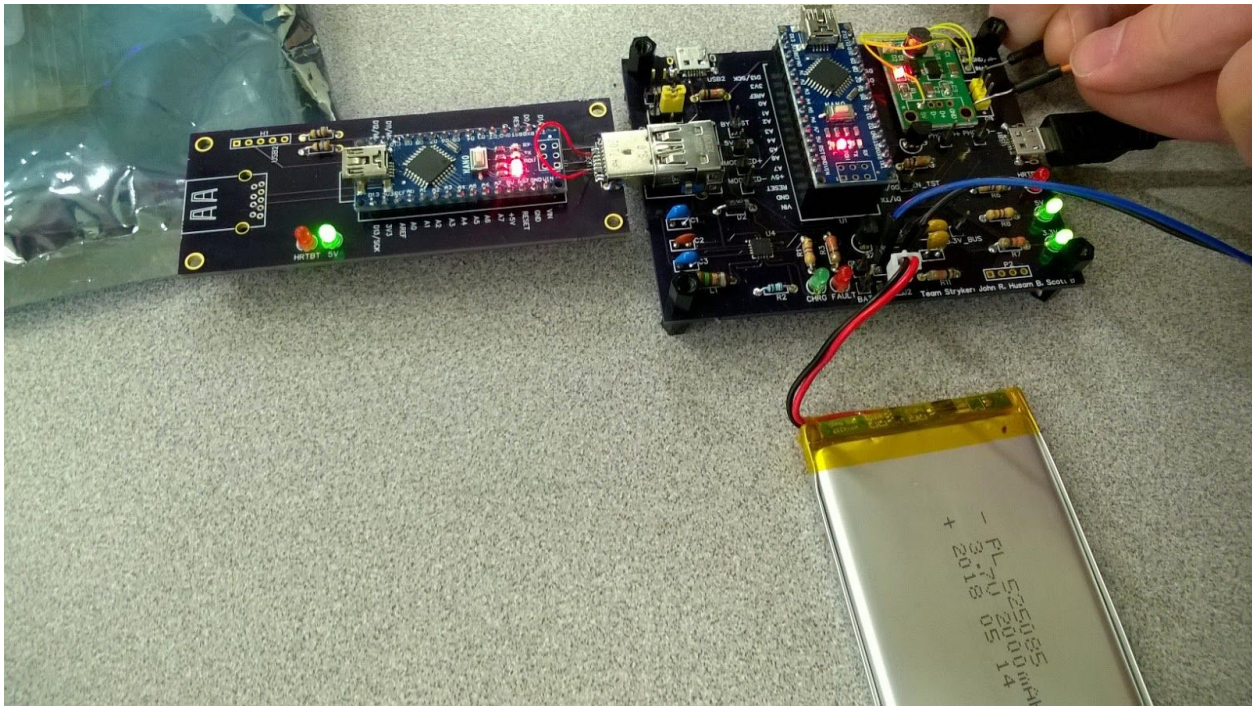


Figure 18: Verifications for several components of the hardware, including the power rail generation, universal connector, enable circuit, and I2C communications.

Specification Analysis - Software

2.1.2. Development board on tool must be capable of reading/writing serial data over a USB

Type A port connecting from Lightning, USB Type-C, and micro-USB
The wired serial interface between the tool and the mobile device was implemented early on through the use of an Android device's micro-USB to communicate over UART to an Arduino Nano. Doing this required the interfacing of the Android device's native FTDI driver to facilitate the UART communication. While this is implemented, the later implementation of using the tool's UART to talk directly to an ESP8266 WIFI module is much more powerful and thus while this functionality will be still available it is recommended to use the TCP/IP connection facilitated by using the ESP8266 as a WIFI access point.

2.5.1. App should talk to a remotely hosted database

The remotely hosted database to store build information and script contents was ultimately constructed using Firebase, which has many easy to use Google APIs as well as a front-end interface that is built-in to the Firebase console website. This saved time on designing a SCRUD interface for the project. As Firebase is managed by Google, Android has many plugins for interfacing directly with it. In the final version of the app, if a WIFI connection is present the user is able to press a button to refresh the local copy of the database, which acts as a buffer for the Firebase information. This is done because of the realization that many of the places that Stryker employees will be using our product will not have available WIFI connections, as well as opening up the product to using the ESP8266 tool variant as a WIFI access point. Doing so will remove mobile device's internet connection, so a local copy of the database is necessary.

2.5.2. App must display the infrared data read from the tool

The IR module, as shown in the verifications, can be commanded to ping out repeatedly for an IR product and once data is received successfully it will transfer the data from the tool to the app, where the dynamic GUI generation capability of the scripting engine will display the infrared data as a hex string.

2.5.3. App must support scripting for commanding/monitoring via Wired/Bluetooth interface

The scripting engine constructed allows for dynamic data routing as well as response and transmit pairing in addition to the dynamic GUI generation capability. This allows for the app to generate very unique user interfaces as well as facilitating complete control over the sources and sinks of data along WIFI, Wired, and Bluetooth data channels.

2.5.4. App must support Bluetooth communication to Stryker products via SPP, A2DP, HFP

The app is capable of supporting Bluetooth communication to multiple Stryker products and development kits using both the module communication as well as the Bluetooth Classic profiles SPP, A2DP, and HFP.

2.5.5. App must be iOS or Android compatible

The mobile app is only Android compatible.

2.5.6. App should be cross-platform compatible

Due to time constraints, the mobile app is only Android compatible. This is mostly due to the requirements of the Lightning connector as well as the requirements for Bluetooth classic. It was found that there were no available development environments that could provide Bluetooth classic support that also supported cross-platform development, so the only way an iOS app was possible would have been to create an entirely new app in that environment, which was not feasible considering our other stretch goal priorities.

2.5.7. App should be able to run on a Bluetooth enabled Windows PC with the tool connected via USB

While another implementation was done that allowed for USB connection, this is not the direction that this platform will go in and thus was scrapped. While it does not support scripting or Bluetooth, there is a Windows PC application implemented in C# that is capable of connecting to the WIFI variant of the tool. This was done to highlight the ideal future of our project, as it is hoped that a Bluetooth classic module will be created and thus the platform requirements can be opened up to simply require an internet connection, which is very accessible in a cross-platform domain.

Verification and Testing Procedures - Software

2.1.2. Development board on tool must be capable of reading/writing serial data over a USB Type A port connecting from Lightning, USB Type-C, and micro-USB

This capability had been proven out and verified before the end of ECE 4810, so that sample will be used again. The implementation of the FTDI driver on the Android interface is unchanged, and the fact that each of the scripts is able to operate acts as an additional verification of this capability. In addition, however, it should be noted that every subsequent verification showing the communication to the tool is either a verification of the serial data reading/writing or using TCP/IP for data reading/writing.

To verify the operation of the device to tool communication, a simple sample was created on a Nano to act as a counter that communicated over the UART channel. The USB port of the Nano was then connected to the Lenovo TBX304F tablet, which ran a custom application capable of receiving UART communication from FTDI devices such as the CH340 compatible Arduino Nano. The code for this verification is included in the appendix under "Arduino to Android UART Test", which also includes the USB Serial java class used to act as an FTDI wrapper for the Android's hardware. The app displaying feedback from the Nano is shown in Figure 19.

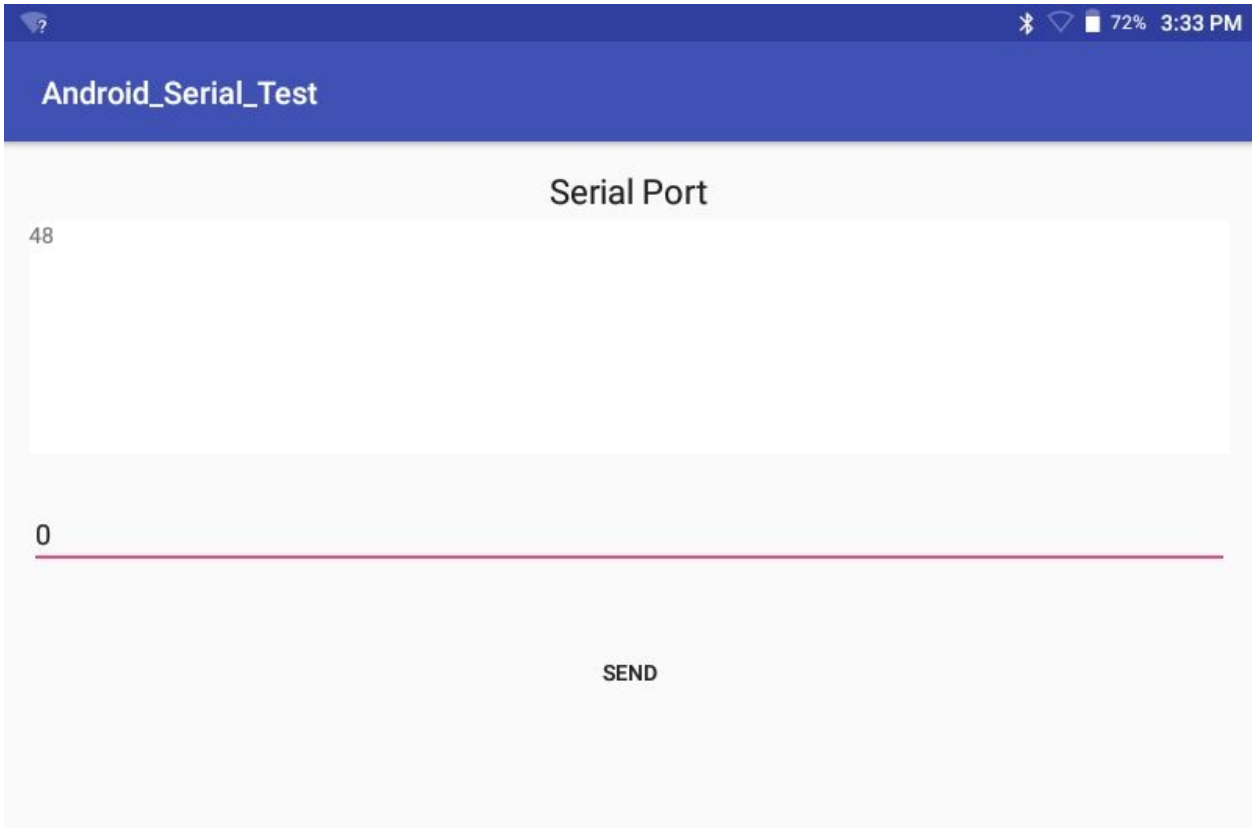


Figure 19: Screenshot of the application communicating over UART to the simulated tool

2.5.1. App should talk to a remotely hosted database

As mentioned previously, Firebase was ultimately used instead of MySQL as the remotely hosted database for storing scripting information as well as software build information. This was especially natural because of the built in libraries for handling this connection because of the use of the Android platform. The major items that were implemented with respect to the remotely hosted database were the ability to refresh a local copy of the database using the remotely hosted database, load the local copy, display its information in a dashboard, and allow for a particular script to be loaded into an execution screen. A block diagram of how the database in the app is handled is shown below in Figure 20. All of these conditions are also mirrored in the event that a user attempts to press the "Refresh Database" button at the top of the script dashboard.

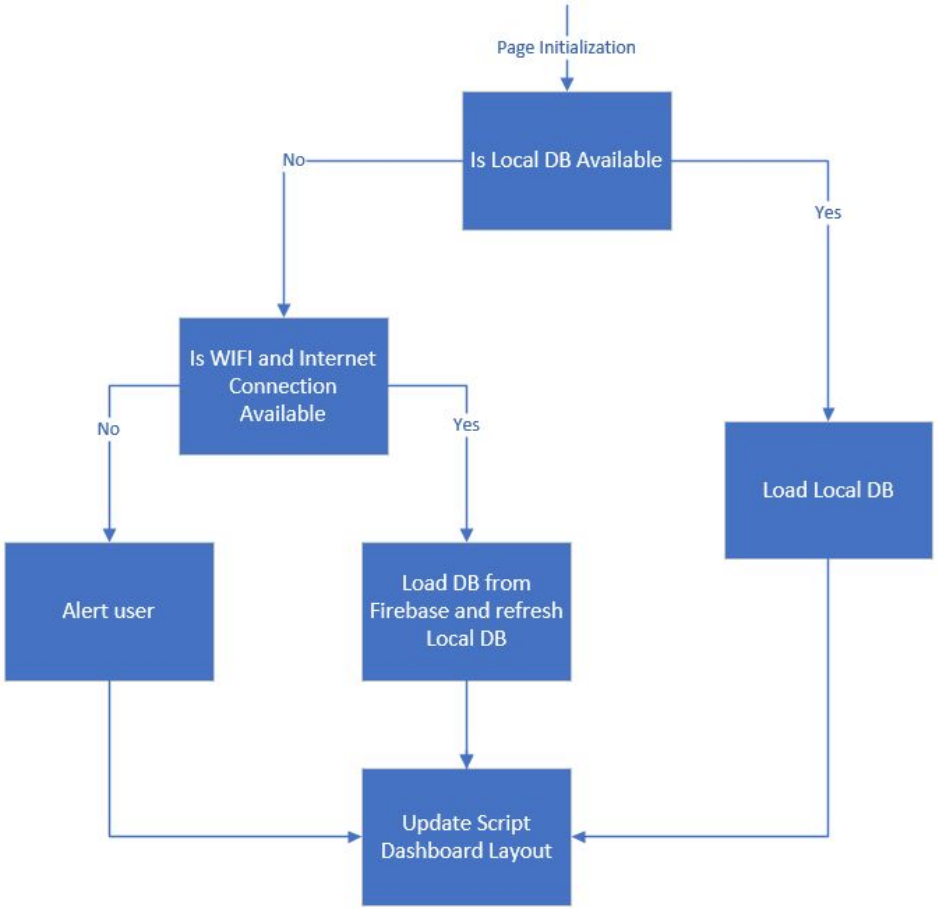


Figure 20: Flowchart of Database Local Copy management - the same flowchart is repeated when it is requested to refresh the database

To start off then, when the app starts up for the first time and the script dashboard is loaded it will actually be completely empty. This is because the first time the app is loaded on a new mobile device the local copy of the app has not been created, as shown below in Figure 21.

Figure 21: First time launch of application, so no local copy of database found

Shortly after, it will check for an active WIFI connection that has internet, and if one is found it will load the database from Firebase and create a new local copy of it, as shown below in Figure 22.

Figure 22: Successfully syncing local database after initial launch

If an internet connection via WIFI is not present, an error message will be displayed indicating that there is no internet connection. Note that there is an important distinction here where a WIFI connection might be present, but no internet connection. This is intended because when using the tool's WIFI access point there will be a stable WIFI connection, but it will not have access to the internet. In the case of both a WIFI connection and internet connection, the previously shown message will display after the database is loaded and copied - otherwise the following shown in Figure 23 will display to indicate that an internet connection is not present.

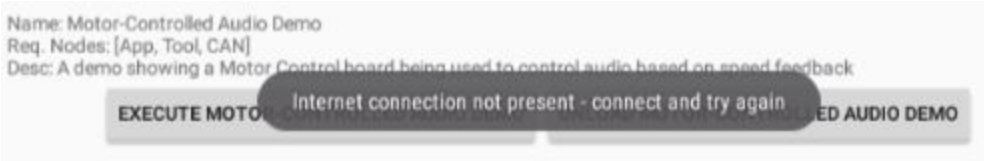


Figure 23: Failure to sync local database because of a lack of internet connection.

The actual data from Firebase is stored through the use of an internal file, which as per Android specification is actually hidden from the user. Two separate internal files are used as private files - scripts.ser and builds.ser, storing a serialized array list of script and build objects respectively. Since these files are hidden from the user, a verification for them must lie in the fact that persistence of this data does occur even without an internet connection. The Firebase web application console provided by Google is shown below in Figure 24, showing both how scripts and build information is stored and how it is visible in the native interface.

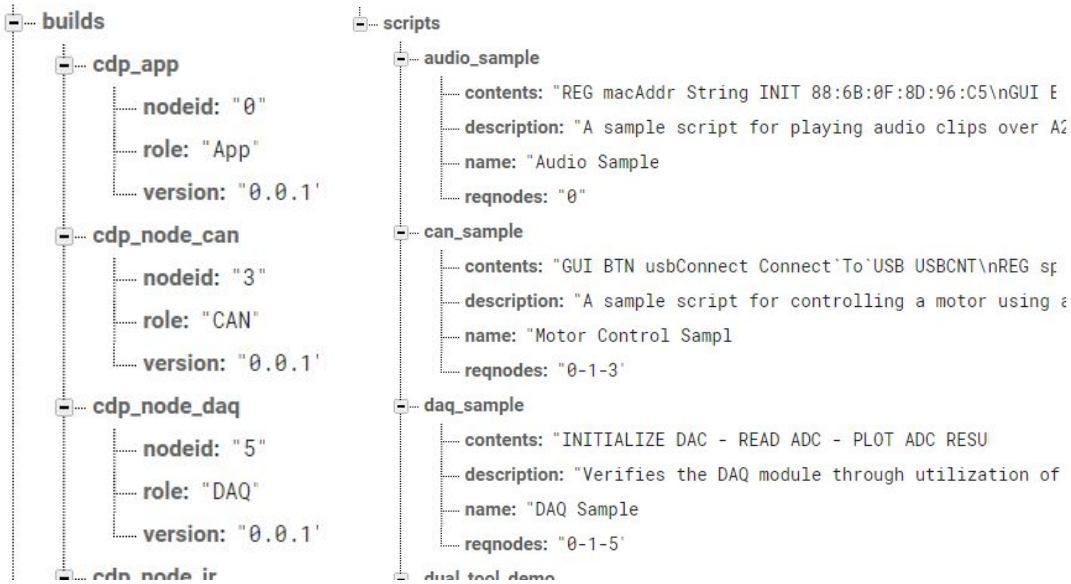


Figure 24: Build and Script information stored in the Firebase database

The app, however, isn't the only part of our platform that interfaces with the remotely hosted database. In order to maintain usability down the line, NodeJS utilities were created to interface with Firebase and both read and write scripts. Handling the build system is done manually through the Firebase web application console provided by Google. This inconvenience is done because realistically only administratives to the system will be creating new modules and thus are the only ones who will need to have this capability, although very minor modifications would need to be made to the read and write script capabilities to allow for build management. To show this off, the format of a script file is shown in Figure 25, and the nature of the script files will be discussed further in the verification of specification 2.5.3.

```
NAME: GUI Sample
DESC: A sample script for testing the GUI generation and preliminary scripting engine
REQN: 0

REG val1src String
GUI TXT val1src STR val1src
GUI LBL val1snk UPD val1src
GUI LBL val1snk2
GUI BTN val1snk2push Sink`2`Push SYC val1snk2 val1src
```

Figure 25: A script file, gui_sample.s

In this file, it is clear that there is a header section that lays out the name of the script, followed by a description and what nodes are required for that script to operate. These required nodes are expressed by the ID provided in the build information database, and in the script dashboard are displayed by their string name described in the corresponding build record. To upload the above script, simply run invoke the write_script file using the name of the target file, and after a small delay in time a successful message will display, as shown in Figure 26.

```
C:\Users\SBatzer\Desktop\cdp\cdp_scripts>node write_script.js gui_sample.s
gui_sample successfully written to db
```

Figure 26: Successfully writing the gui_sample.s script file to the Firebase Script record

Now that the gui_sample.s script file is joined with the Script record, we can invoke our other NodeJS capability to allow for verification that the proper value is in the database now, as shown in the following Figure 27.

```
C:\Users\SBatzer\Desktop\cdp\cdp_scripts>node read_script.js gui_sample
{ contents:
   'REG val1src String\nGUI TXT val1src STR val1src\nGUI LBL val1snk UPD val1src\nGUI LBL val1snk2\n
GUI BTN val1snk2push Sink`2`Push SYC val1snk2 val1src\n',
  description:
   'A sample script for testing the GUI generation and preliminary scripting engine',
  name: 'GUI Sample',
  reqnodes: '0' }
```

Figure 27: Successfully reading the gui_sample.s script file from the Firebase Script record

With this, the user is able to manage and maintain a large array of scripts and thus the definition of a variety of different systems, able to define the sinks and sources of data throughout and how the interaction between those modules occurs, such as shown in Figure 28. This really shows just how using our system, a design with only one use-case can but morphed into one with many use-cases.

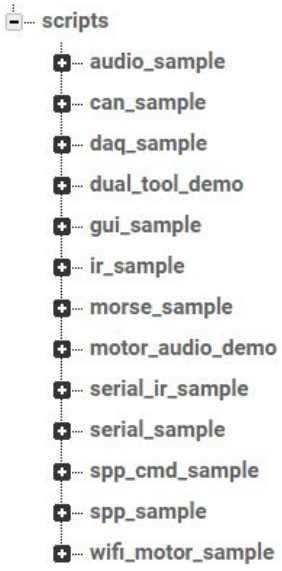


Figure 28: List of a variety of scripts currently stored in the Script record

2.5.2. App must display the infrared data read from the tool

This verification relies heavily on our IR module, which is used to, when commanded, ping out repeatedly for an IR product. It pings repeatedly because the target IR products actually sleep for an extended period of time, so there is actually a high likelihood that the IR module will attempt to connect to the product at a time that the product is asleep and thus no data will be transmitted. The IR module assumes that it did connect, and will start listening to the input from the transceiver for a small amount of time, and if no data has been received it will initiate another request for connection. Once data transmission starts, the IR module will continuously log that data until it has filled the full payload corresponding to Stryker's IR protocol, then send that up to the app for display, as shown in the block diagram in Figure 29. This block diagram is shown physically in Figure 30.

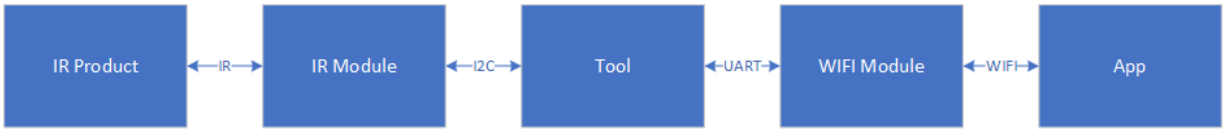


Figure 29: Block diagram of IR product and module verification

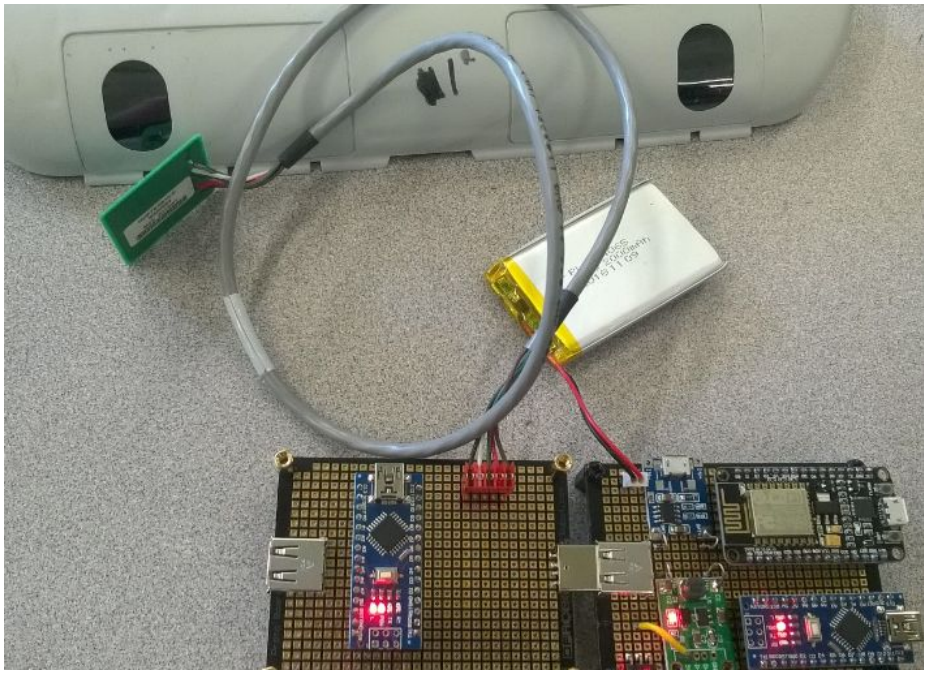


Figure 30: Physical realization of Figure 29's block diagram

The timing of this interchange cannot be disclosed because of the proprietary nature of Stryker's IR protocol, but it should be noted that the timing is on the microsecond level, and the implementation of the IR module is actually on an Arduino Nano. This means that it was necessary to use direct port manipulation as well as using looped NOP assembly injection to allow for 62.5ns resolution on our delay scheme, as the existing Arduino delay functions do not provide high enough timing resolution. This timing, complete with attempt cancellation, is

shown in figure 30. The first channel is the output to the IR transceiver, the second channel the input from the IR transceiver, and the third a sync bit used for calibrating the timing of when to sample the IR transceiver input. Again, this is purposely obfuscated to protect Stryker's IP.

Figure 30: Timing diagram of sampling and request exchange, with attempt cancellation

This connection is also a verification of the script handling further detailed later in the verification of specification 2.5.3, and the script corresponding to this verification is shown below in Figure 31. Simply put, it establishes a button for connecting to the tool via WIFI, initializes a register for storing the request for an IR product's ID as well as the response of the request, indicating that the response will be provided via a WIFI source and its connected IR module. Additionally, there is a logging label for showing the received value on the app. This GUI is shown on Figure 32.

```
NAME: WiFi BBIDWM IR Sample
DESC: A sample script for reading a BBID from the IR module over WiFi
REQN: 0-1-2

GUI BTN wifiConnect Connect`To`WiFi WIFICNT
REG bbidReqCmd String INIT AA
GUI BTN bbidReqBtn Request`BBID WIFITX IR bbidReqCmd
REG irRx String WIFISRC IR
GUI LBL irRxLog LOG irRx
```

Figure 31: Script that utilizes the IR module to retrieve ID information from a Stryker IR product

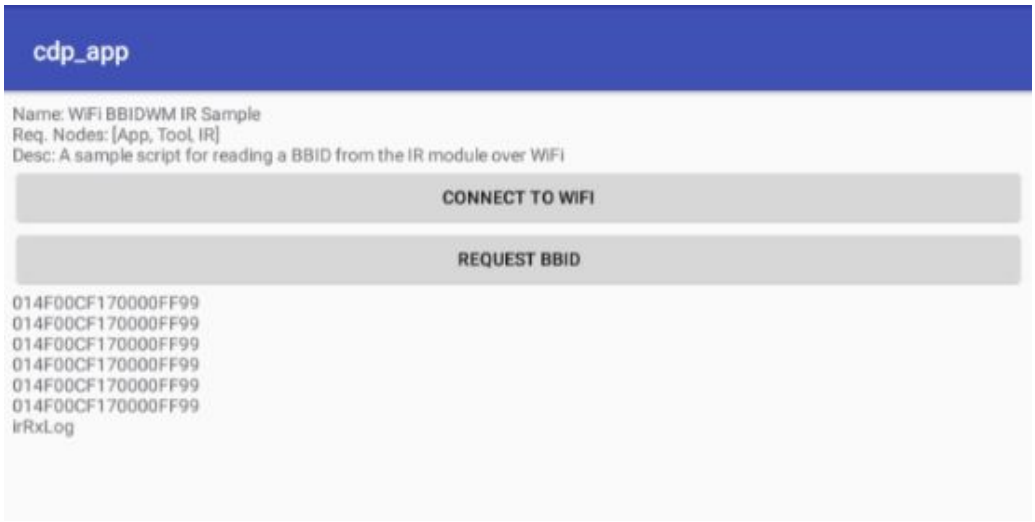


Figure 32: App running the script from Figure 31

2.5.3. App must support scripting for commanding/monitoring via Wired/Bluetooth interface

An implied specification that is not included in the original specifications is that for scripting to occur, it must be both read from the database as well as parsed. Originally, our method of script parsing was to use JSON formatting that would be deserialized, as shown in Figure 33, with relation to a generic script object that would be further inherited into specific types of script objects. The goal of this was to be able to map stimulation with the corresponding response, as shown in Figure 34.

```
{
	"script": {
		"stim_response_pair": {
			"stimulation" : {
				"protocol": "IR",
				"data": "0b1010"
			}
			"response" : {
				"protocol": "IR",
				"data": "0b0101",
				"delay": "10s"
			}
		}
		"delayed_stim" : {
			"stimulation": {
				"protocol": "IR",
				"data": "0b1111",
				"delay": "15s"
			}
		}
	}
}
```

Figure 33: Example script showing configuration using JSON format

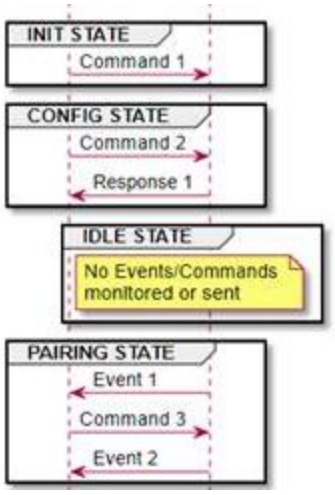


Figure 34: Example sequence diagram used for scripting purposes

Later in our project we found this Object Oriented Design approach would not scale well with the inevitable redesign that would follow based on expanding feedback and from experience with trying to implement certain systems. Thus, designing a full inheritance-driven design for this would require significant redesign and development time, whereas a flat text file with very

flexible string formatting could actually resolve that inflexibility of JSON formatting, so a pivot was made early on, to the format shown previously in the IR verification. This is shown again in the form of another demo, shown in Figure 35.

```
NAME: Dual-Tool Demo
DESC: A demo showing an application using two separate tools and module chains
REQN: 0-1-4-9

GUI BTN wifiConnect Connect`To`WiFi WIFICNT
GUI BTN usbConnect Connect`To`USB USBCNT
REG morseTx String TOOLSRC SERIAL
GUI LBL morseTxLbl UPD morseTx
GUI BTN wifiTransmit Send`Morse`Message WIFITX MORSE morseTx
```

Figure 35: Script detailing a demo using two separate module chains with a tool connected via WIFI and another connected UART

The major advantage of our scripting engine is the capability to dynamically generate GUI and handle dynamic data routing, which allows for creating really flexible systems. The above Figure 35, a simple system is defined - a register that stores data from the serial module, and then a button to send out the data stored in the serial register. This is mirrored in how the block diagram shows the system, also showing that we can connect to two separate tools over both USB and WIFI and handle that data routing with our scripting, below in Figure 36. This consideration of a system as merely data sources and data sinks is both natural and intuitive, and reflects very nicely as a block diagram.

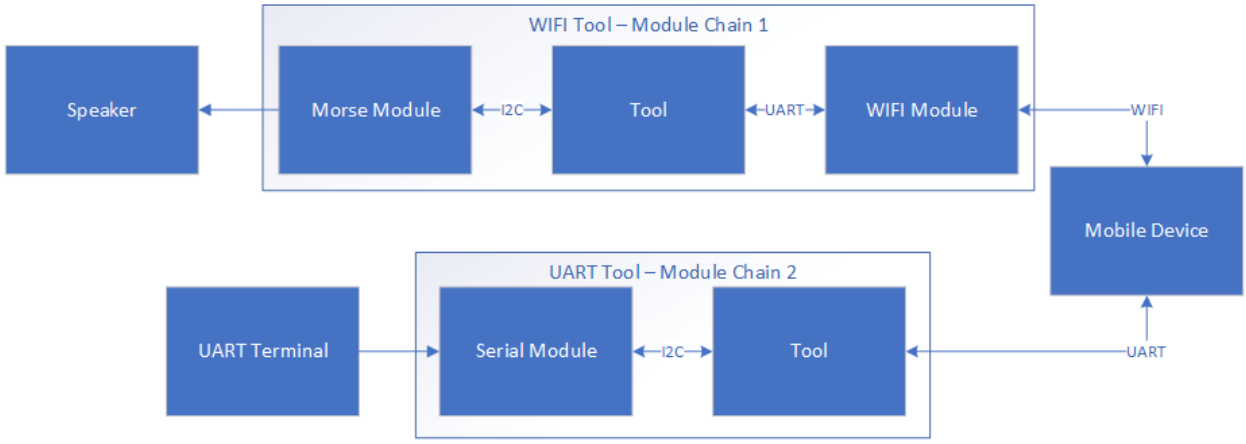


Figure 36: Block diagram showing two separate module chains to drive both a serial module and a morse module.

This sample also adds the mention of two modules that have not been explained yet - a Serial Module and a Morse Module. Simply put, the Serial Module interfaces a UART channel to the modular interconnect, allowing for UART input via terminal into the system. The Morse Module simply plays a received string according to Morse code, and due to time constraints does not include parsing audible Morse code into string format in form of receive commands. That block diagram declares that when UART input is put into the Serial Module from a terminal, it will be

logged by the app and then at a button press played on the Morse Module in the form of audio on a speaker. The WIFI enabled tool and Morse Module are shown in Figure 37, and the Serial Module is connected to our Rev2 Modular PCB as shown in Figure 38.

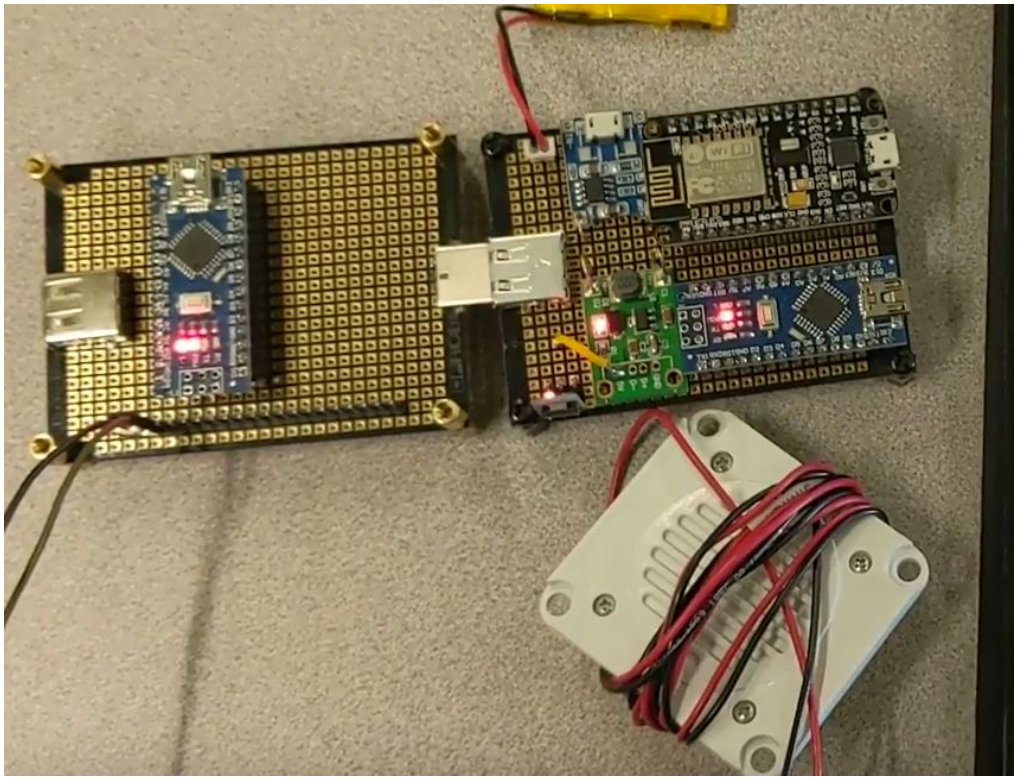


Figure 37: WIFI enabled tool, Morse Module, and a speaker for playing audio

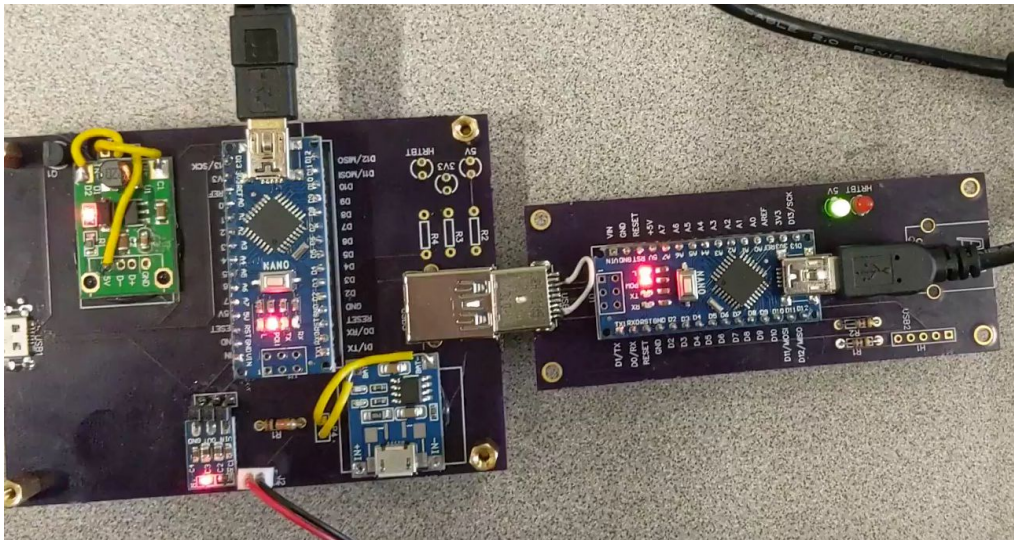


Figure 38: Rev2 Modular PCB connected to a Serial Module, connected to a UART terminal

2.5.4. App must support Bluetooth communication to Stryker products via SPP, A2DP, HFP

The following also applies to the verification of specification 2.5.3, as all of the following verification requires a great deal of scripting. Additionally, the method of which Bluetooth Audio was verified utilized a demo centralized on another of the modules, the CAN Module. This particular module uses the MCP2515 and TJA1050 CAN Controller and Transceiver via SPI to facilitate a CAN bus, but we also took this a bit further to also allow for the filtering of data and also added the ability to ignore unwanted PDOs. This is especially important for our design as all this data is being routed down to a UART channel to be passed off to either the ESP8266 or mobile device directly, so that UART can act as a bottleneck if the CAN overloads it. The CAN Module is shown below in Figure 39. Additionally, Figure 40shows that the resulting TJA1050 CAN Transceiver properly outputs CANH and CANL with bus compatible levels.

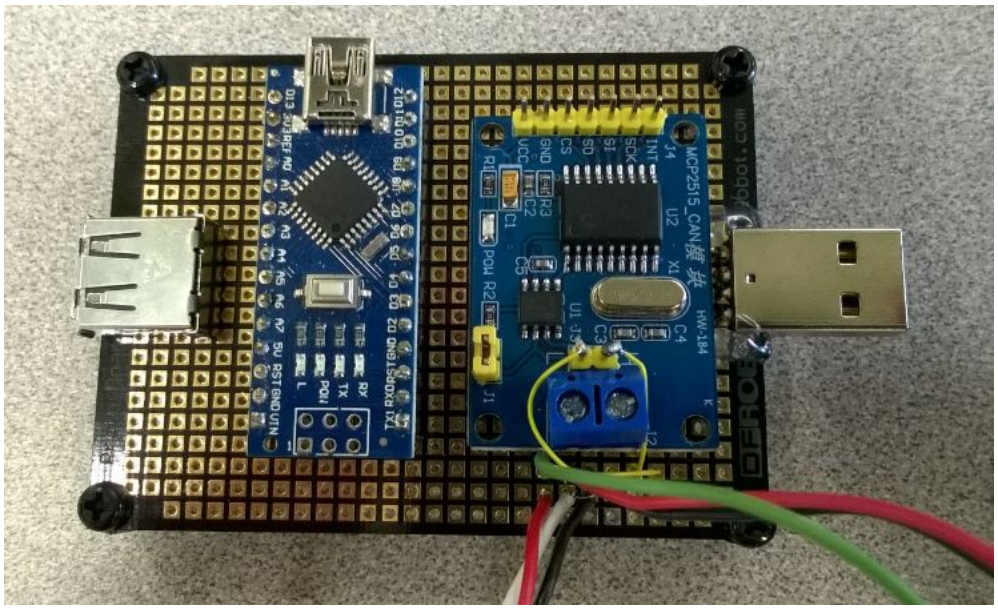


Figure 39: The CAN Module

Figure 40: The CAN bus (CANH and CANL) operating with proper differential behavior

To verify that we can actually do proper RX and TX using the CAN module, we also used an off-the-shelf IXXAT USB-to-CAN module and MiniMon, shown in Figure 41, to verify that we could operate in a commercial CAN environment, and also verified that our module is capable of communicating to a car using its OBDII port using the same environment. This, we were able to confirm that no matter the environment, our system will be able to interface with the CAN bus.

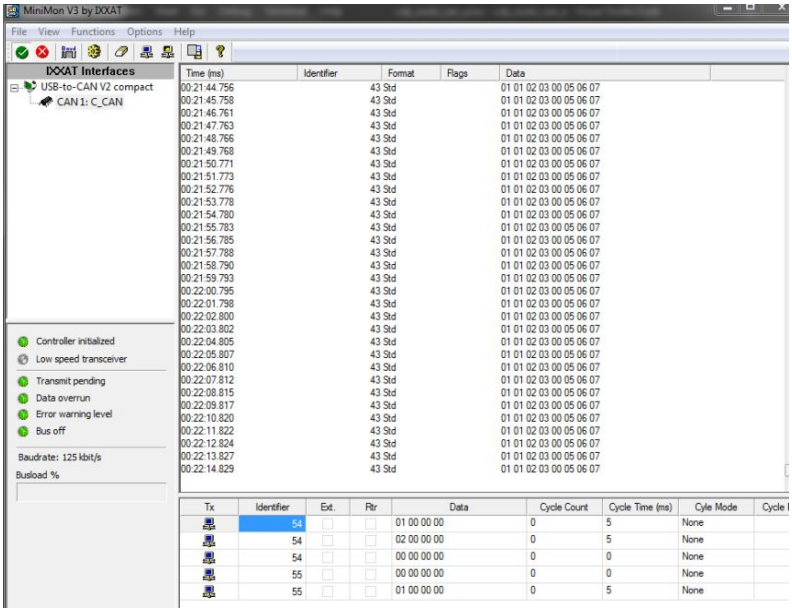


Figure 41: MiniMon CAN Terminal connected to via USB-to-CAN IXXAT device

With that in mind, the CAN module has a lot of functionality that needs to be explained before proceeding. It is capable of reading and writing CAN messages, as well as elect to listen to specific CAN PDOs. Additionally, it also can filter certain PDOs by indicating a section of the CAN payload, a target PDO, and the resulting data type. Reading is done passively, so the following Table 7 details the syntax and operation of each of the three commands to the module. Note that in this, we have to explicitly tell the CAN Module to listen to a particular PDO - this is because most buses have a multitude of messages and this could easily bog down the UART channel.

Table 7: CAN Module command examples

"0,0123,1122334455667788"	Write command to PID 0123 with payload 1122334455667788
"1,3210"	Tells the module to start listening to PID 3210
"2,3210,5,0,3,F"	Tells the module to start listening to 3210, but also sets that an average filter will be applied with a 5 sample count size, and the slot being used for this in the payload starts at byte 0 and ends at byte 3 and is of datatype float

Previously, we've talked about data routing for Strings, but our system is also capable of handling numeric Data Types like Floats. For example, as shown in the block diagram in Figure 42, if we get data from WIFI, the node id will tell us whether it is from the CAN or IR module, and the PDO will tell us which register to put that data in, based on which bytes and what format we want. We can then attach on-set condition checking to this register, such as playing audio clips when it either transitions above 1250 or below 750. This payload mapping and conditional mapping is evident in the script for a motor controlled audio demo shown in Figure 43.

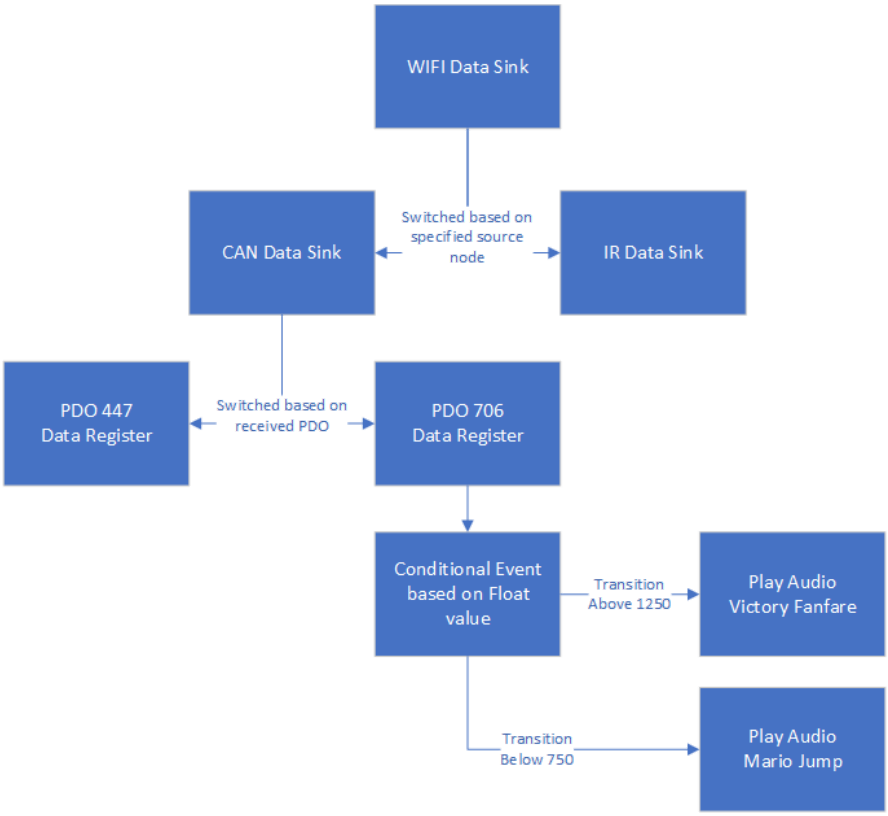


Figure 42: Block diagram showing how data sinks are propagated to handle data parsing

```
NAME: Motor-Controlled Audio Demo
DESC: A demo showing a Motor Control board being used t
REQN: 0-1-3

REG macAddr String INIT 88:6B:0F:8D:96:C5
GUI BTN sppConnect Connect`To`BT`SPP SPPCNT macAddr
REG sppSink String SPPSRC
GUI BTN wifiConnect Connect`To`WiFi WIFICNT
REG speedStop String INIT 0,447,0000000000000000
REG speedLow String INIT 0,447,0000C84100000000
REG speedMed String INIT 0,447,0000484200000000
REG speedHigh String INIT 0,447,0000964200000000
GUI BTN cmdStop Command`Motor`Stop WIFITX CAN speedStop
GUI BTN cmdLow Command`Motor`25% WIFITX CAN speedLow
GUI BTN cmdMed Command`Motor`50% WIFITX CAN speedMed
GUI BTN cmdHigh Command`Motor`75% WIFITX CAN speedHigh
REG canRx String WIFISRC CAN
GUI LBL canRxLog UPD canRx
REG motorSpeed Float PAYLOAD CAN canRx 706 0 4
GUI LBL speedLbl UPD motorSpeed
CMD CND motorSpeed ABV 1250 AUDIO fanfare
CMD CND motorSpeed BEL 750 AUDIO jump
GUI GPH speedGraph SRC motorSpeed 0 100 0 1750
```

Figure 43: Script detailing a Bluetooth Audio demo controlled by Motor RPM

This script shown in Figure 43 is realized to create the following block diagram below in Figure 44, which details a Stryker Motor Controller being connected to the CAN Module as the device controlling the motor, and is physically realized in Figure 45.

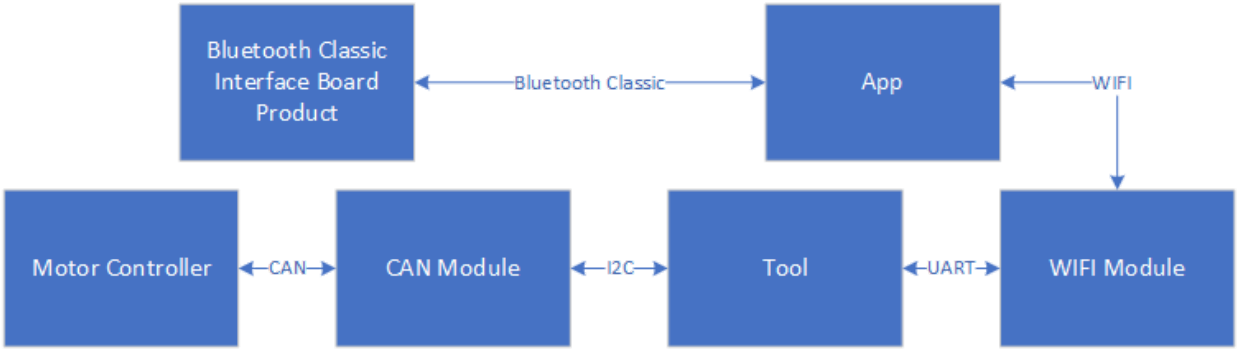


Figure 44: Block diagram for a Bluetooth Audio demo controlled by Motor RPM

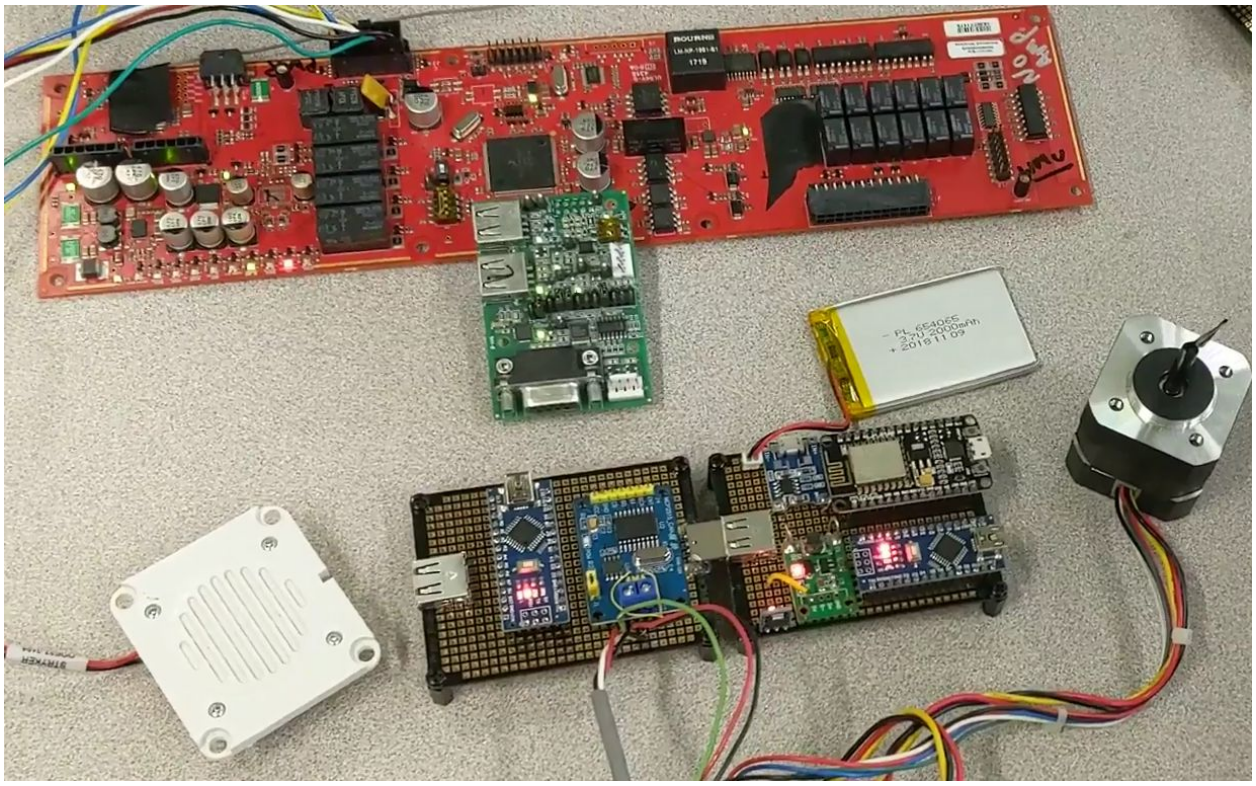


Figure 45: A Stryker Bluetooth Enabled Interface Board connected via SPP, A2DP, HFP to the mobile device, which is also connected via WIFI to our tool to facilitate a CAN bus to a Stryker Motor Control Board (not pictured to protect IP)

Another awesome component of this system is that we can select numeric registers like the motorSpeed register for this example and mark them for graphing, which will then render and log an XY graph to log all changes in that value with respect to time, which is visible in Figure 46.

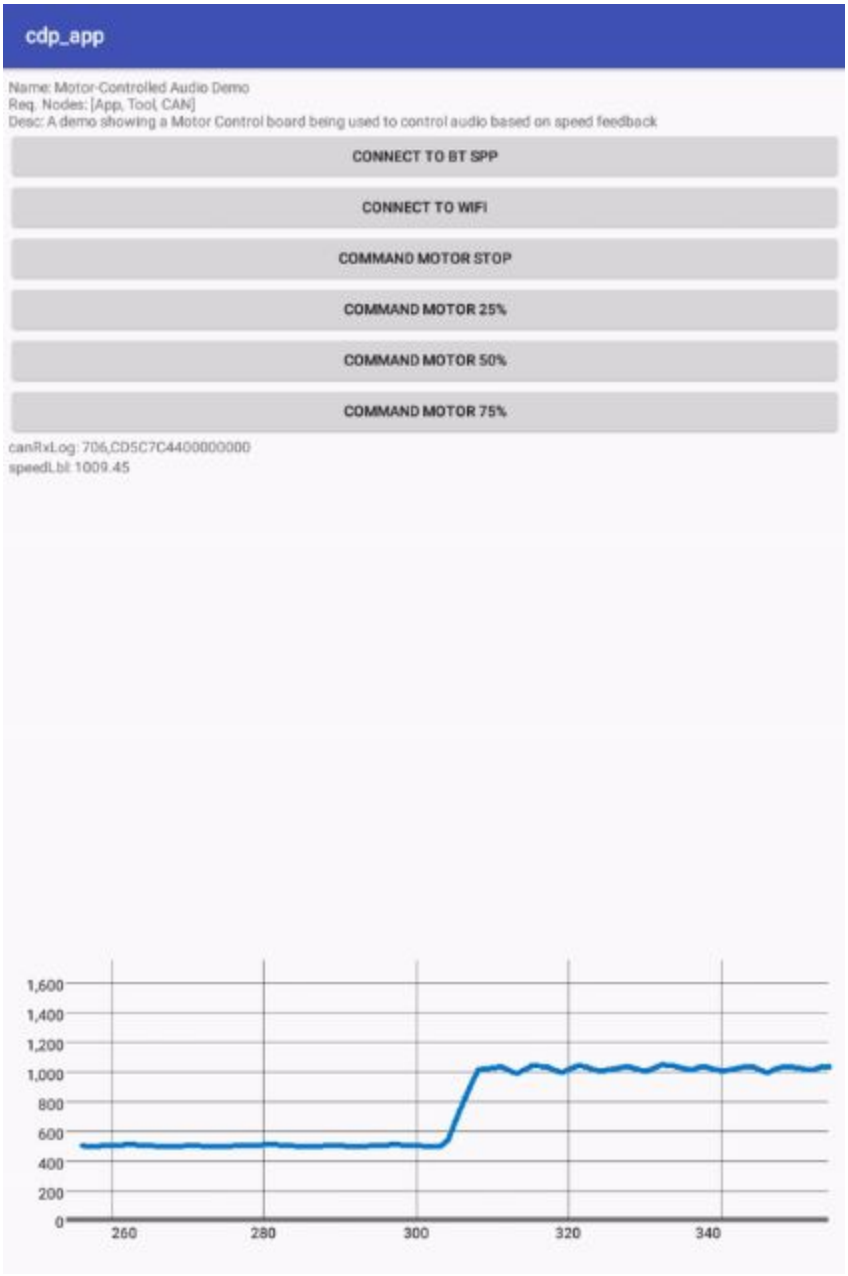


Figure 46: Mobile device running app using the motor controlled audio demo, showing the available GUI elements and XY Graphing capability of motor speed

While the SPP interface is used for the Bluetooth Audio demo above, it is done solely to facilitate a continuous connection to the Stryker Interface board that is being connected to the app. To show a verification of our system truly talking over SPP, we can also add more detail to the scripting verification to include more robust sequence modeling capabilities that allows for our systems to not just act as data sinks and sources but also act intelligently based on feedback and truly model how a system operates. As shown in Figure 47, we can actually create command lists using our scripting engine to allow for our transmits to expect responses of a certain nature, and if they are not received then certain actions will be made. These actions may include either

repeated a transmit or setting a fail flag, and the setting of a fail flag might actually be the trigger of another command list or event. This allows for extremely powerful modeling of a system, done easily with very few lines of scripted code.

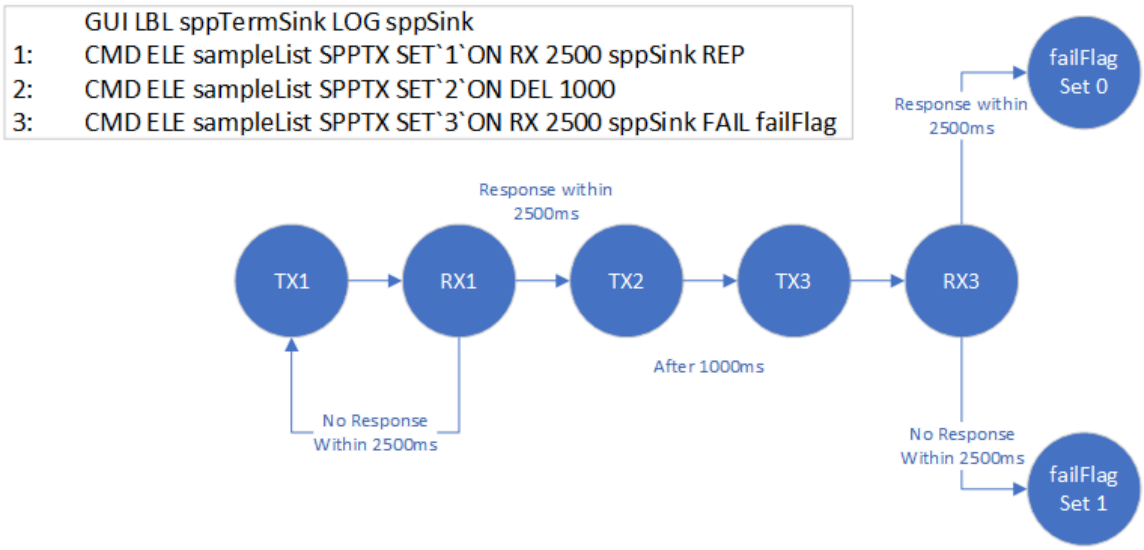


Figure 47: Diagram detailing an example command list and the resulting sequence

To show this capability running on hardware, we increased the complexity of the previous diagram to include a bit more functionality, such as another step in the command list chain and some registers to facilitate the interface, as shown in Figure 48.

```
NAME: SPP Command Sample
DESC: A sample script for testing SPP communication and command list handling
REQN: 0

REG macAddr String INIT 00:07:80:CC:7C:63
GUI BTN sppConnect Connect`To`SPP SPPCNT macAddr
REG sppTxSrc String
GUI TXT sppTxSrc STR sppTxSrc
GUI BTN sppTransmit Send`SPP SPPTX sppTxSrc
REG sppSink String SPPSRC
REG failFlag String INIT Pass
GUI LBL failLbl UPD failFlag
GUI LBL sppTermSink LOG sppSink
CMD ELE sampleList SPPTX SET`1`ON RX 2500 sppSink REP
CMD ELE sampleList SPPTX SET`2`ON DEL 1000
CMD ELE sampleList SPPTX SET`3`ON RX 2500 sppSink REP
CMD ELE sampleList SPPTX SET`4`ON RX 2500 sppSink FAIL failFlag
GUI BTN initCmdList Send`Cmd`List CMD sampleList
```

Figure 48: Script for the SPP command list sample

This sample, because the Stryker interface board has many additional messages that would interfere with this test, is actually being ran on a separate development board that uses the same Bluetooth classic module as the Stryker board (hidden to protect IP), shown in Figure 49. The corresponding block diagram for the sample is shown in Figure 50.

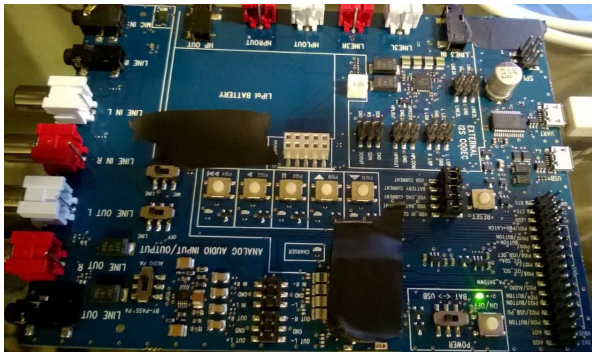


Figure 49: Bluetooth classic development board

Figure 50: Block diagram detailing the system-level operation of the SPP command list sample

One important consideration was necessary for this testing was that this particular development board has a minor glitch that results in a single SPP packet occasionally being sent as multiple packets. This required additional logic to account for appending received data to a register, which would then continuously append until that data contained a new-line character. Additionally, it also required the facilitation of dual-mode SPP communication, where in the previous motor-control sample communication was performed via byte-arrays rather than ASCII formatted, human readable strings, as shown in the following Figure 51. This is a configuration in the script file.

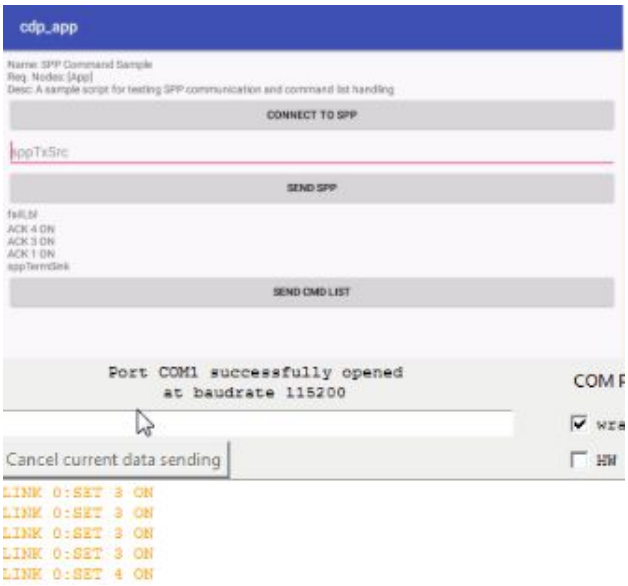


Figure 51: On the bottom, the Bluetooth classic devkit interfacing via UART to report the data logged through the interaction with the app on top

2.5.6. App should be cross-platform compatible

As mentioned in the specification analysis, the mobile application ultimately was not cross-platform due to concerns regarding the Lightning connector and Bluetooth Audio difficulties when implemented on a cross-platform framework.

2.5.7. App should be able to run on a Bluetooth enabled Windows PC with the tool connected via USB

While the app does not run cross-platform, a PC-compatible version was created that connected to the tool via TCP/IP. This does meet the end goal of what environment the project will be used in and likely serves as a more suitable environment than the existing Android app in the long run. USB connection has been done as well using this version, but the TCP/IP communication is preferred for its future extensibility. The WIFI access point capability is shown to be operational below in Figure 52.

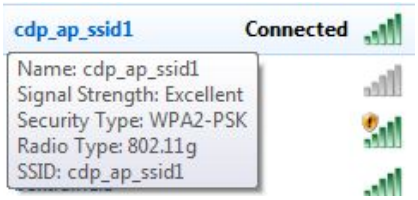


Figure 52: PC successfully connecting to the WIFI access point of the tool

To show this functionality a sample was created to use our DAQ module, the simplest of our modules, to connect a potentiometer input to the PC app. By using the DAQ module, our tool is capable of controlling the function of each pin on the Arduino Nano and can set digital output and read both digital and analog input, with configurable sampling rate for any inputs. It has 11 channels for digital input and output, and 6 channels for analog input, and is shown in Figure 53.

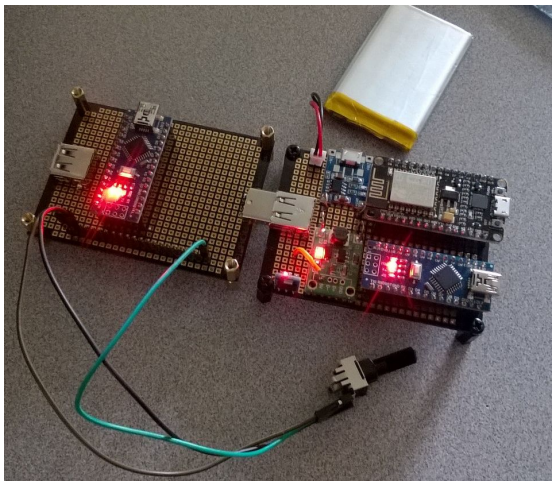


Figure 53: DAQ module connected to a WIFI connectable iteration of the tool with a potentiometer for controlling input

Figure 54 and 55 shows the app connected to the WIFI enabled tool with a DAQ module connected, and the feedback corresponding to the tuning of the potentiometer. Note that the user has sent out 5-2,2, and unshown the user had previously put 5-2,1 and 5-2,0, and this is visible in the app by the expanding analog input being read on the app. The initial token, 5, is the ID of the DAQ module, and the following tokens indicate the command type and payload of the message, in all cases here the configuring of an analog input on pin 0, 1, and 2. Additionally, if the command type 0 is sent, the next token will allow for a digital output pin to be set and the final token will state what the output will be set to. This is relayed in Table 8.

Table 8: DAQ Module command examples

"0,1,0"	Set Digital Output pin 1 Low
"1,2,1"	Set Digital pin 2 to be an Output pin
"2,0"	Start listening to Analog pin 0 input
“3,500”	Set input sampling delay to 500ms

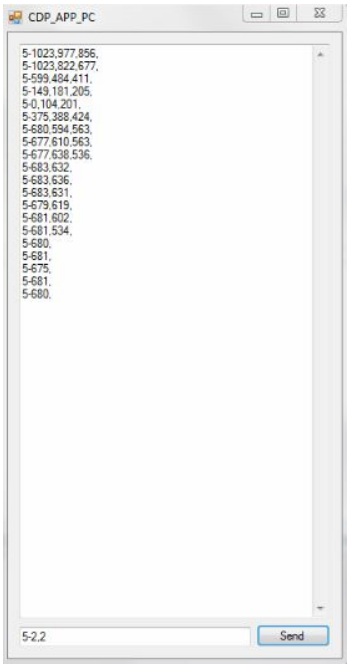


Figure 54: DAQ module connected via WIFI

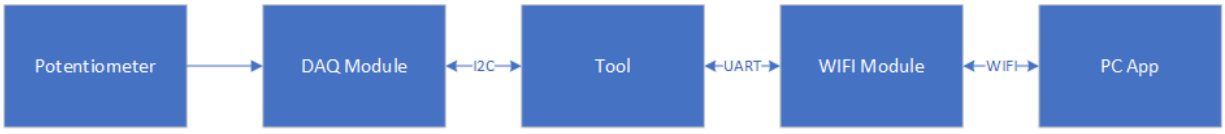


Figure 55: Block diagram of the verification of PC application connection via TCP/IP along the Tool’s Access Point using the DAQ module and a potentiometer

CONCLUSION: FINAL EVALUATION of SPECIFICATIONS MET

All of the physical characteristic specifications have been met - there is a 3D-printed enclosure (1.1.1), and if the user chooses there is a velcro approach for fastening the tool to a mobile device (1.2.1), however it is not recommended. The device is both light (1.2.3) and does not heat excessively (1.2.4), and while it isn't form fitting it is not exactly a burden to hold (1.2.2). Additionally, there are no toxic materials used in the construction of the tool or any modules (1.3.1).

Our choice of the Arduino Nano for the tool's microcontroller satisfies specifications 2.1.1 and 2.1.2, being readily available from several suppliers in bulk, and supporting serial communication.

Specifications 2.1.3 and 2.1.4 called for the inclusion of an OLED display for general status indication on the tool. In the final state of the project, the OLED capability has been implemented, but left off the hardware. As our project went on, we decided that this is likely not a necessary feature considering that the user will be using it to connect to a mobile device, which is a much more powerful display device. As such, this capability is not included with the suggested product, yet its inclusion has been verified and the header on the tool PCB remains - if the user wishes to use the OLED they will just need to reflash the tool and populate the OLED header.

In order to meet specifications 2.1.5 and 2.1.6, the core tool houses a TP4056 standalone lithium ion battery charger board, which is used to charge the onboard 2000mAh 3.7v lithium ion battery pack. With a measured current draw of the total system, this allows us to achieve a minimum 12 hour battery life as specified.

In order to achieve specification 2.1.7 as well as specification 2.3.3, all standalone modules used in the construction of the hardware circuits were soldered into our PCBs, or hot-glued in place their layouts did not allow for the prior option. Beyond that, the entire boards is mounted on standoffs which secure to the enclosure, further fixing all the circuitry in place.

As mentioned previously, specification 2.2.1 was modified from the original. Rather than requiring a USB Type A port for interfacing with the phone, a micro USB port was required, and one can be found on the core tool PCB intended for this interface. For specification 2.3.1, a standard USB 3.0 Type A port was placed on the core tool, interfacing with a standard USB 3.0 Type A plug placed on each module.

Specification 2.3.2 was met, but also in a manner other than what was detailed in the original proposal. The original specification implied, but did not necessitate, a module that would be connected to the universal connector for charging the battery. Instead, an off-the-shelf module was put on the tool itself to handle the battery charging capability as required to hit specification 2.1.5. This off-the-shelf component has itself a 5V micro USB port that can be used to either charge the battery or power the system without a battery.

The addition of a 5V DC to DC step up boost converter and a 3.3V linear regulator allowed us to meet specification 2.3.4, by providing 5V and 3.3V power rails off of the lithium ion battery rated for 3.7V to 4.2V.

Specifications 2.4.1 and 2.4.2 were met, we will provide fully functional infrared and CAN communication modules to Stryker along with the core tool.

Google's Firebase platform for cloud hosted databases convenient to use with Android, well documented, fully cross platform, and allowed us to meet specification 2.5.1 easily.

Specification 2.5.2 was met through the inclusion of GUI LBL script elements. By including a GUI LBL line in a script and associating it with a variable set to store received infrared data from the IR module, data could be displayed on the smartphone screen to the user as soon as it was available.

As mentioned before, the scripting capability was fully implemented into the application, moving above and beyond specification 2.5.3.

The application also fully supports Bluetooth classic communication, using the required serial and audio profiles, meeting specification 2.5.4.

Specification 2.5.5 was met; the application was developed in the Android environment and will be compatible with most new and old Android devices. The application is proven to be backwards compatible as far back as Android version 7.1.1.

Specification 2.5.6 was not fully met due to time constraints and issues with sending serial data through a Lightning connector; the project is currently fully compatible with Android smartphones and partially compatible with Windows. By adding a HiLetgo ESP8266 NodeMCU, which includes an ESP8266 WIFI module, however, we allow for a wireless connection to the phone, bypassing the issue of sending serial data over the Lightning connector and making an iOS application feasible to develop in the future. As a bonus, using a wireless connection offers improvements to device flexibility (as the tool is no longer tethered to the phone) and data rates.

Specification 2.5.7 was at last partially met as well, a Bluetooth enabled PC can be used to run the application, however some features have not yet been implemented including the scripting support.

Specifications 3.1 through 3.3, covering supplementary materials such as hardware cables and instruction manuals, will be met when this report has been delivered at the close of the project.

In order to meet specification 4.1, the enclosure is held shut using screws and nuts, allowing for disassembly using tools typically found in the workplace.

Specification 5.1 was met through our use of Android libraries designed to facilitate connection using the IEEE 802.15.1 standard, or Bluetooth.

The final cost for a core tool is approximately 20 USD, and the cost of a module ranges from 8 to 15 USD. Assuming a complete system consists of a core tool and five modules, we have met specification 6.1 and remained with the 750 USD budget.

We can confidently say that the project was summarily a success. In its current state, the device passes all the performance tests specifically required by Stryker, with additional verifications for any features added beyond the minimum viable product.

RECOMMENDATIONS

When attempting to create a dynamic and future-proof communication system like this, one of the most important things to stress is the master tool's communication. The formatting on the data must be both robust and elegant; too much of the former and it might be too difficult to add new modules, and no matter what extensibility is possible it must actually work. We elected for a uniform format of node id with a '-' character separating a data string. This is highly robust, but it wastes a fair few bytes - definitely lacking in elegance. If we spent more time to build out a format and a corresponding command lexicon for each module that the tool needed to manage directly to save data, the concerns of data rate might have been abated. Any team attempting to construct a similar system will need to balance how much time they want to spend designing their format and the ease of adding new modules.

It is really important to have good debugging capabilities when working on any system, which our system lacked for a good portion of time with respect to the wired connection. When debugging an Android app, it is required to have a wired connection to the development PC, but when testing the wired interface to the tool this is impossible. This made it extremely difficult to resolve issues that occurred when testing the tool, but when our tool began to use WIFI primarily these issues were resolved instantly and making further development much easier. It is extremely important to consider the debugging environment the team will have to endure, so it is highly recommended that any future teams do not include a wired interface to a mobile device that only has a single USB port available and instead rely on either Bluetooth or WIFI communication, except for when it is absolutely necessary. The combined requirements of more in depth control of the Bluetooth Audio profiles and wired interface limited our project to being solely on Android, so committing to WIFI earlier in the development would have allowed for more investigation into platforms allowing audio control cross-platform.

Our project also opens up for a fascinating future project through the use of its CAN module and mobile device interface when one considers the use of an OBDII port to attach to the CAN module. We were able to verify that we can get CAN data packets from a 2005 Buick Rendezvous, which given a bit of work could be used to create a tool that can be used to control the AC, windows, doors, and other aspects of the car. If the car has a suitable adapted cruise control unit (ACC), it can even be used to drive the car, so with the suitable hardware our project could be furthered by pursuing a preliminary self-driving car or app-driven car.

As a final recommendation, it would be great for our project to be taken further with fewer off-the-shelf modules and also using an FPGA for higher internal data rate. If we could have digitized all communication modules into a single block, theoretically our final tool could have been a single FPGA with many output ports that could be resynthesized to meet various communication protocols. It would still be a modular design, but that would be an internal architecture rather than the physical connections that are in the final design.

APPENDIX A: BILL of MATERIALS

Core Tool

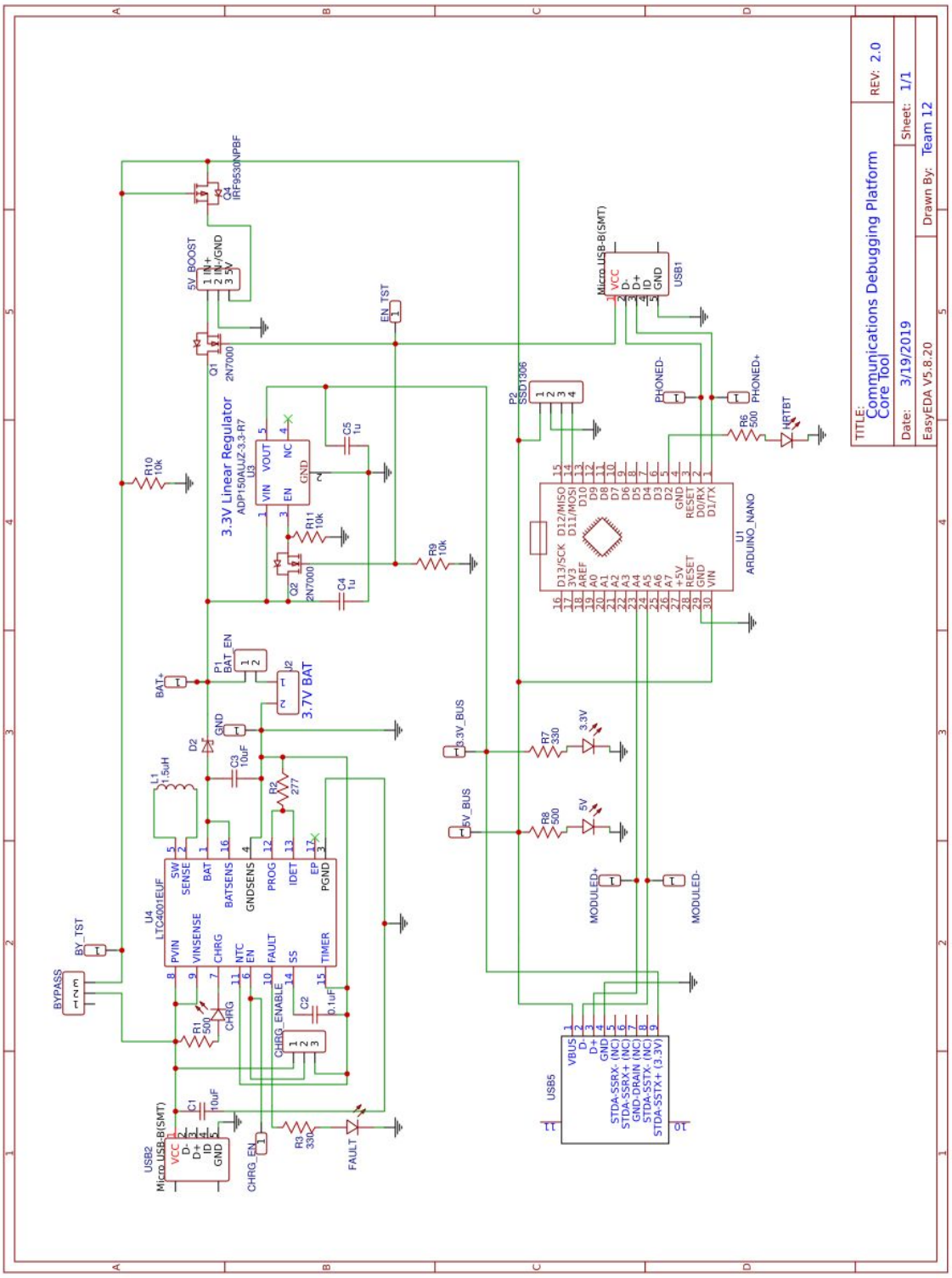
Designator	Name	Quan	Manufacturer Part	Manf	Supplier	Supplier Part
P1	3.3V Regulator	1	DC 4.75V-12V to 3.3V Voltage Regulator	Amnbest	Amazon	B07CP4P5XJ
J2	3.7V BAT Pins	1	2 Pin header	TE Connectivity	LCSC	C86471
J2	3.7V BAT	1	Lithium Polymer Battery	YDL	Amazon	B07BTVT2HH
P2	5V Booster	1	Step Up Boost Converter Power Supply Module 1V-5V	Icstation	Amazon	B01N6EKHZR
USB5	USB 3.0 Type A Female	1	CONN RCPT USB3.0 TYPEA 9POS R/A	Molex	Digikey	WM10413CT-ND
P4	McIgIcM TP4056 Battery Charger	1	P4056 Lithium Battery Charger Module	MCIGICM	Amazon	B06XQRQR3Q
P3	Logic Shifter Module	1	3.3V-5V 4 Channels Logic Level Converter Bi-Directional Shifter Module	FTCBlock	Amazon	B07H2C6SJJ
U1	ARDUINO_NANO	1	Arduino Nano V3.0	ELEGOO	Amazon	B0713XK923
U2	ESP8266 NodeMCU	1	ESP8266 NodeMCU	HiLetGo	Amazon	3-01-0268

Modules

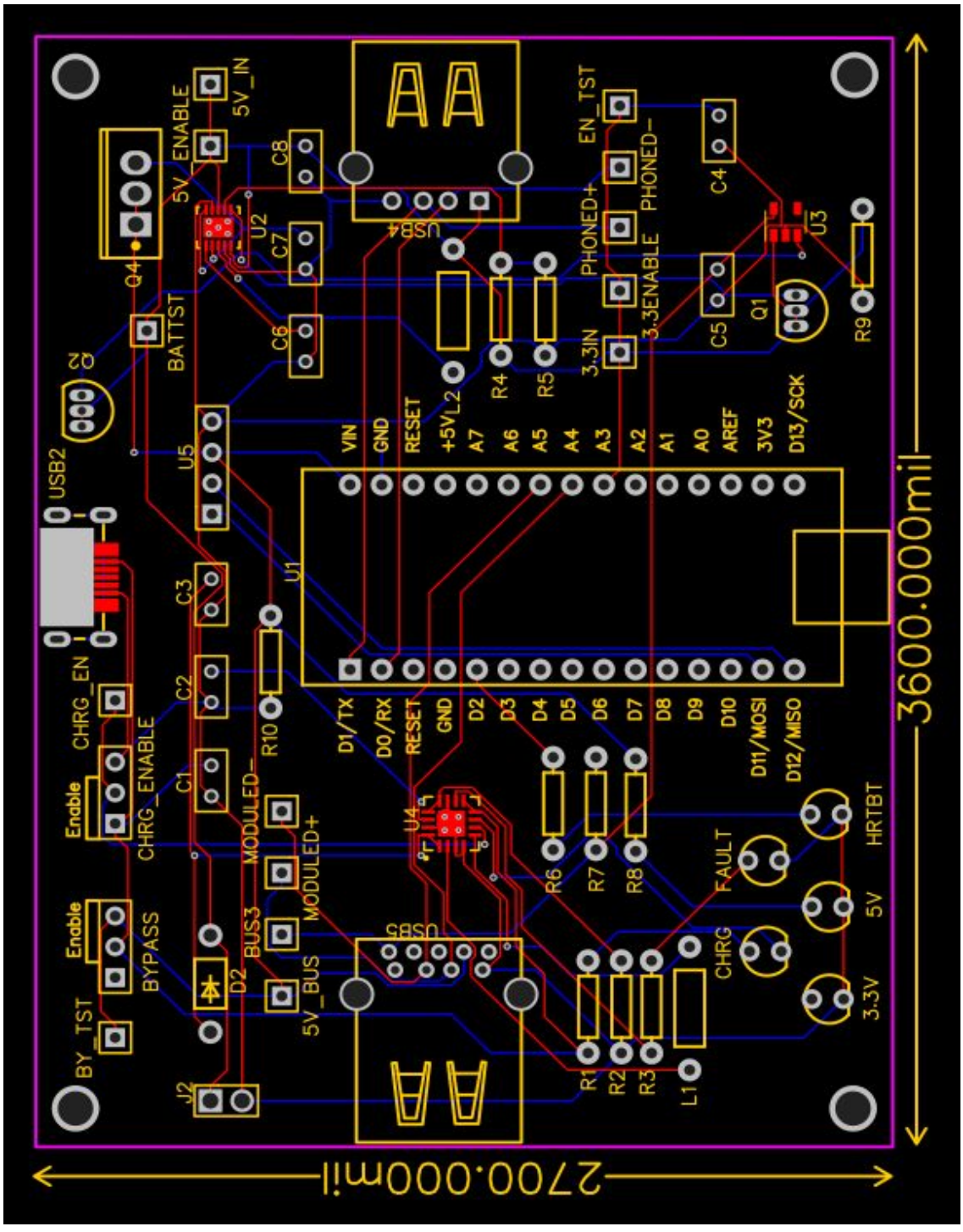
Designator	Name	Quan	Manufacturer Part	Manf	Supplier	Supplier Part
U1	ARDUINO_NANO	5	Arduino Nano V3.0	ELEGOO	Amazon	B0713XK923
USB5	USB 3.0 Type A Female	5	CONN RCPT USB3.0 TYPEA 9POS R/A	Molex	Digikey	WM10413CT-ND
USB4	USB 3.0 Type A Make	5	CONN PLUG USB3.0 TYPEA 9P SMD RA	Amphenol ICC	Digikey	GSB316441CEU

APPENDIX B: SCHEMATICS and PCBS

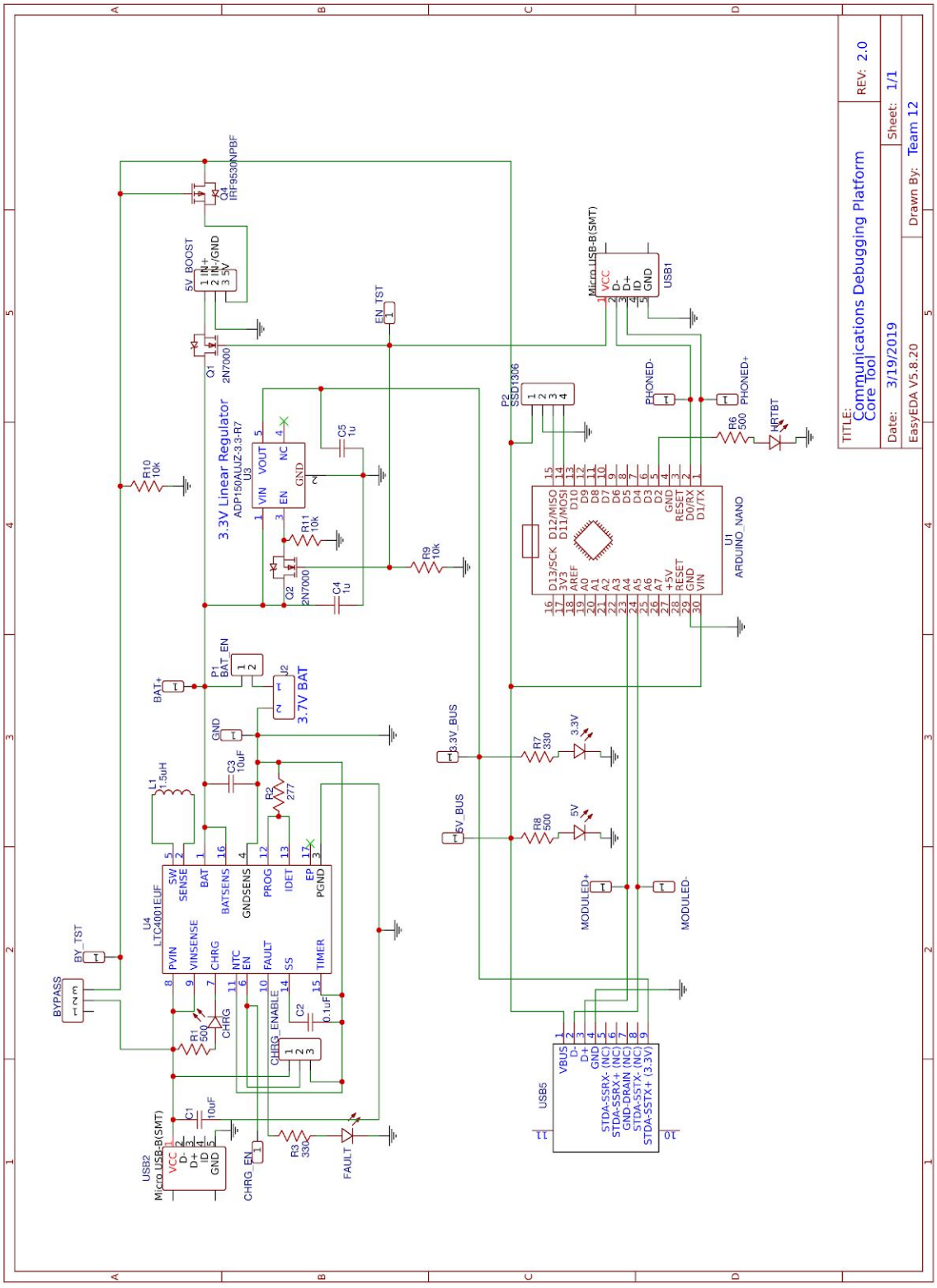
Initial Core Tool Schematic



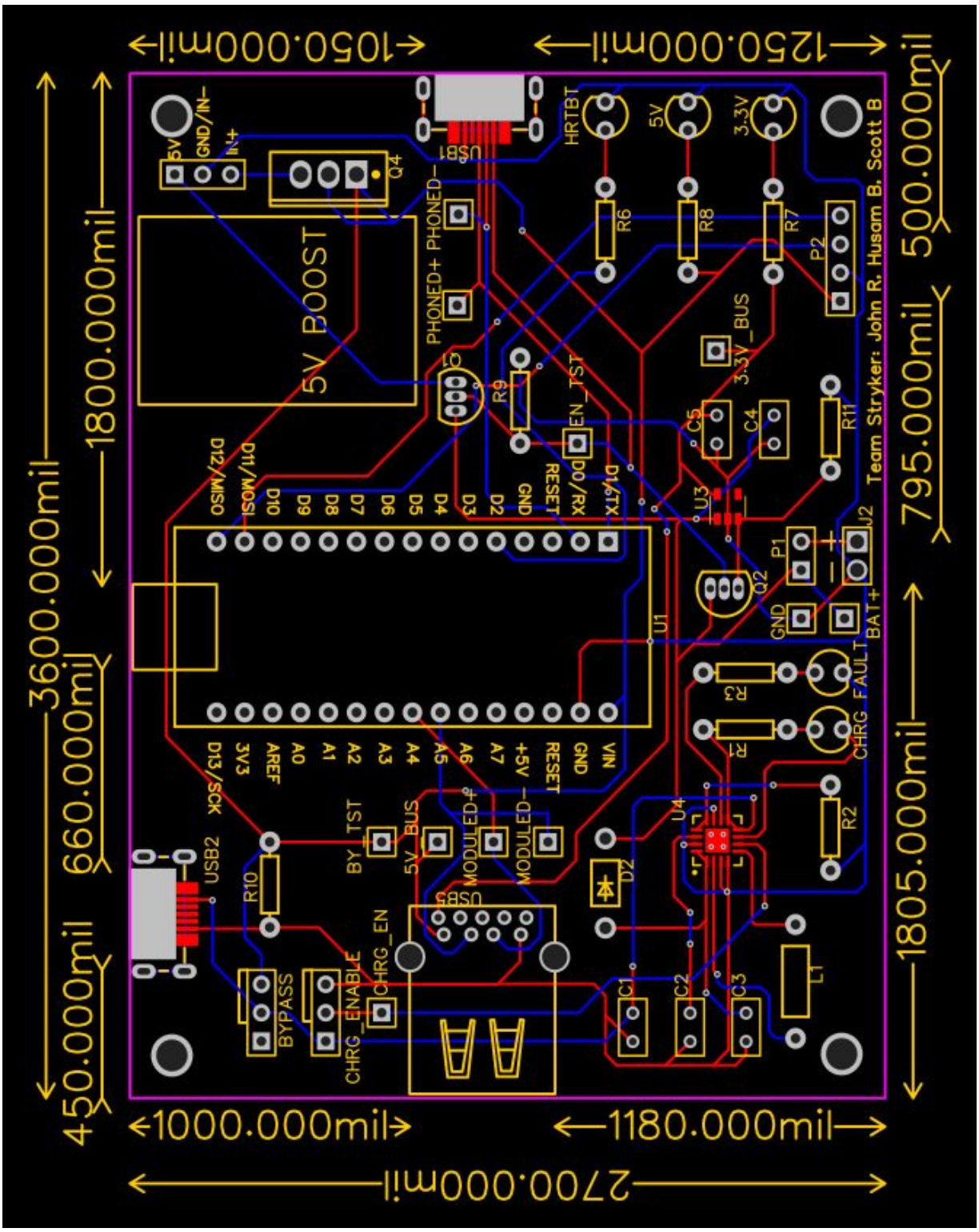
Revision 1 Core Tool PCB



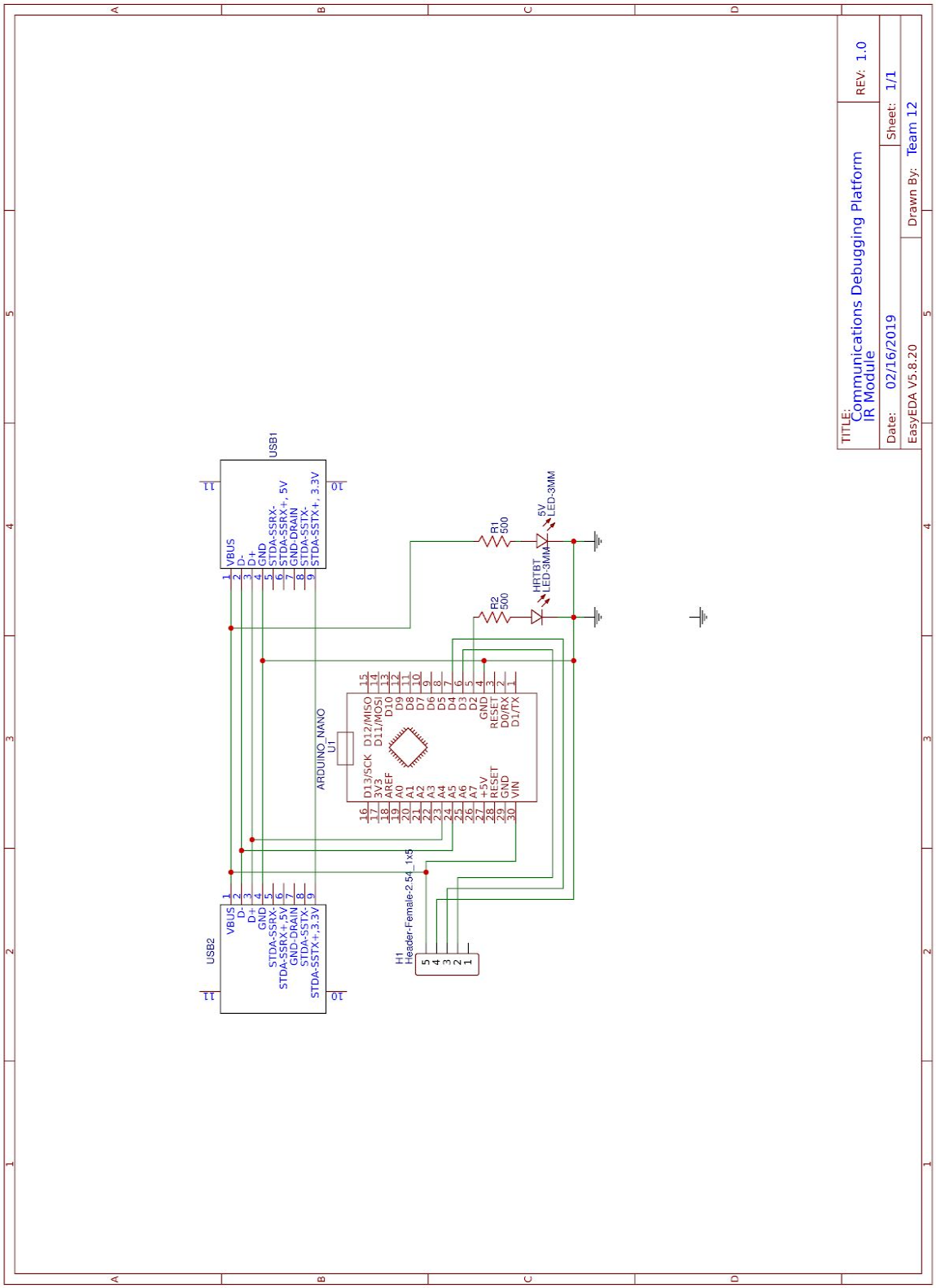
Revision 2 Core Tool Schematic



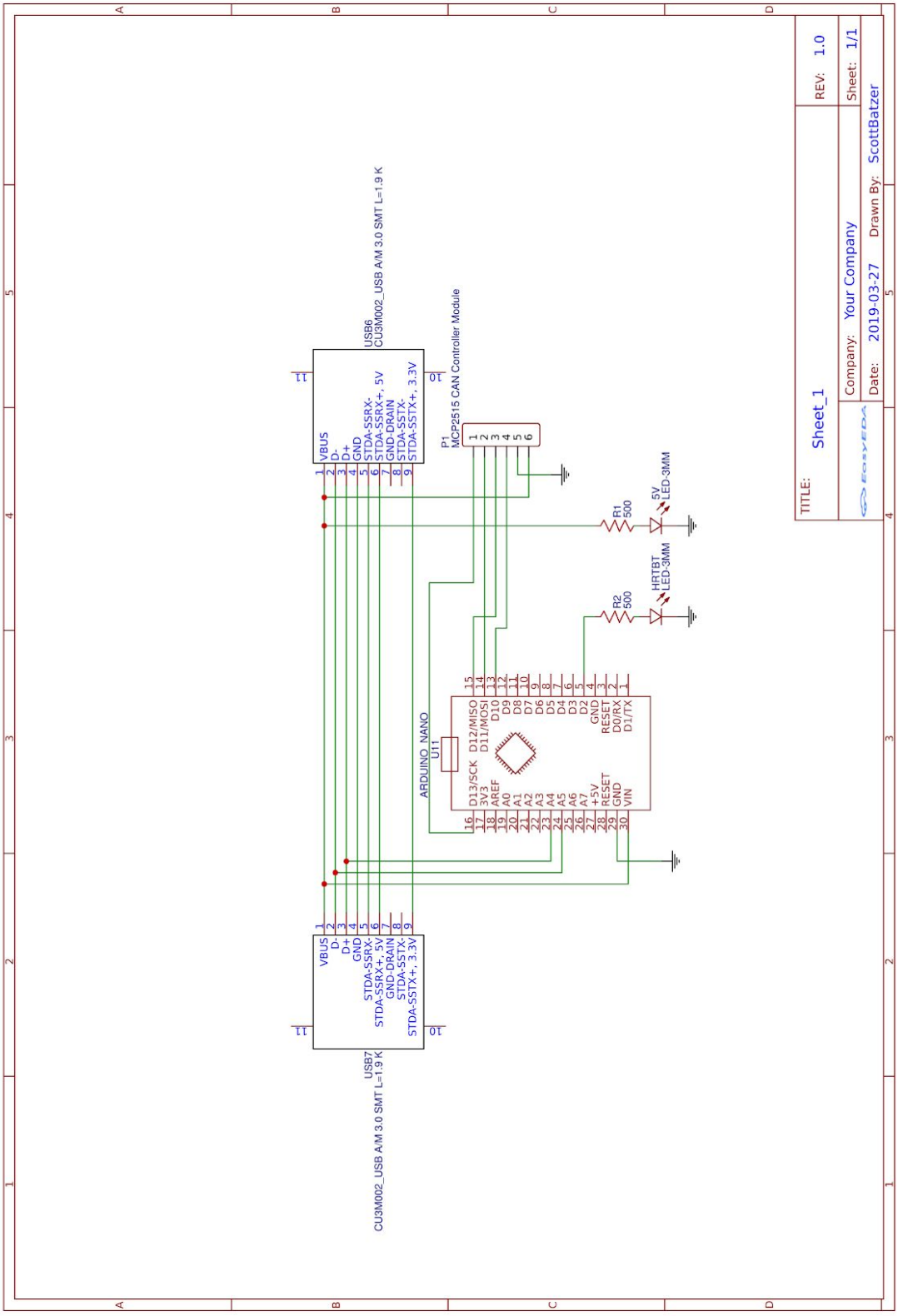
Revision 2 Core Tool PCB



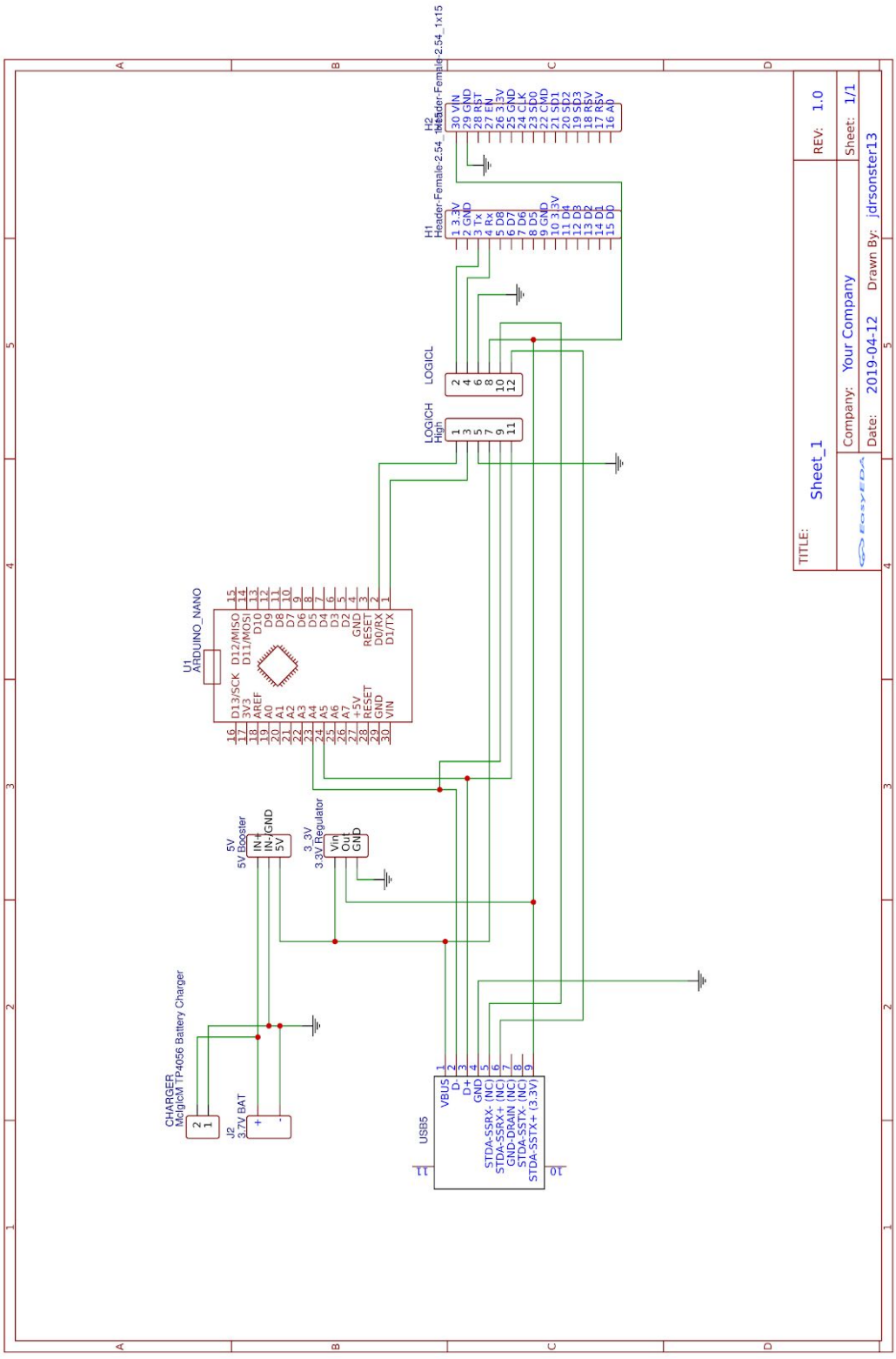
IR Module Schematic



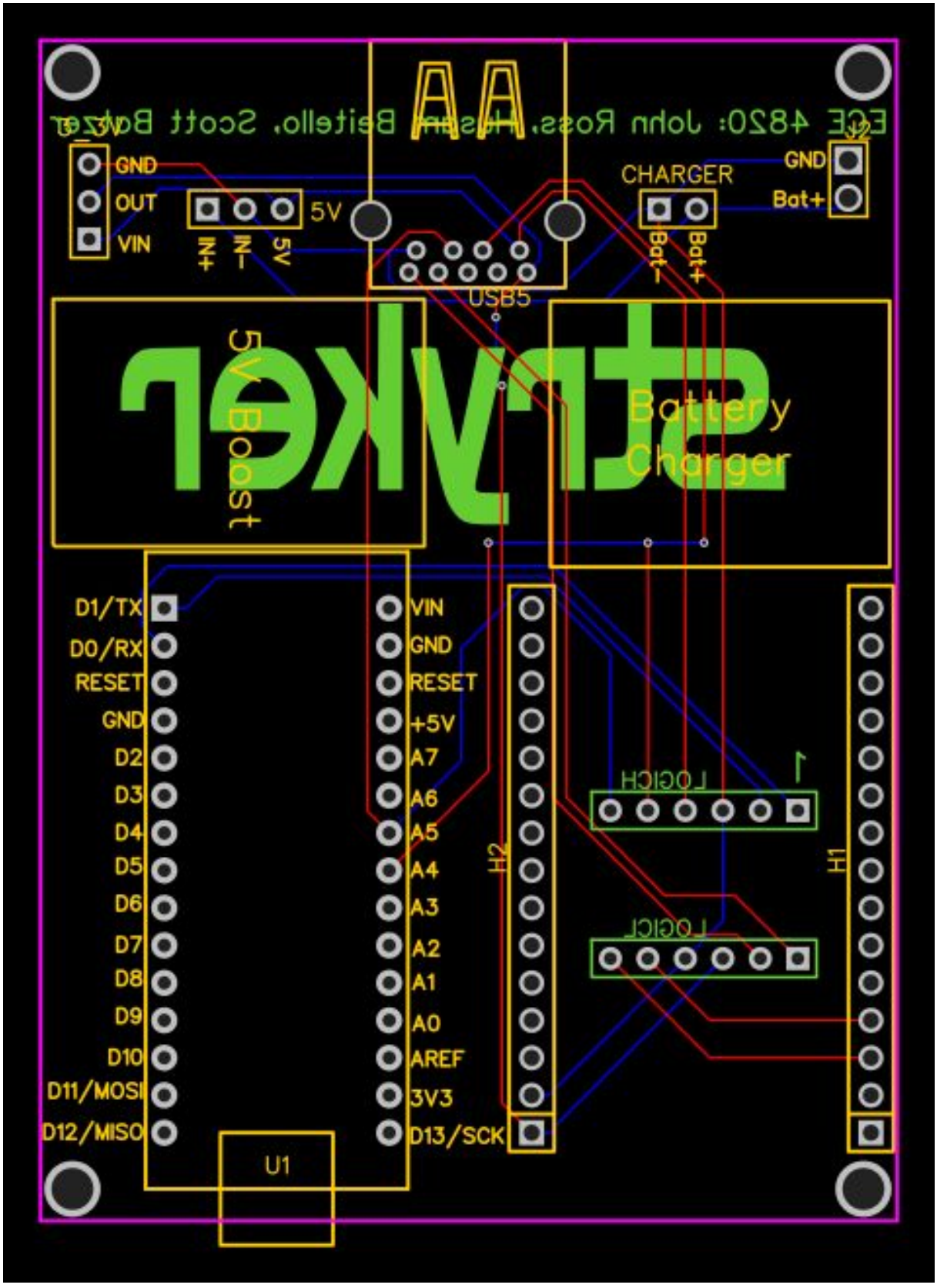
CAN Module Schematic



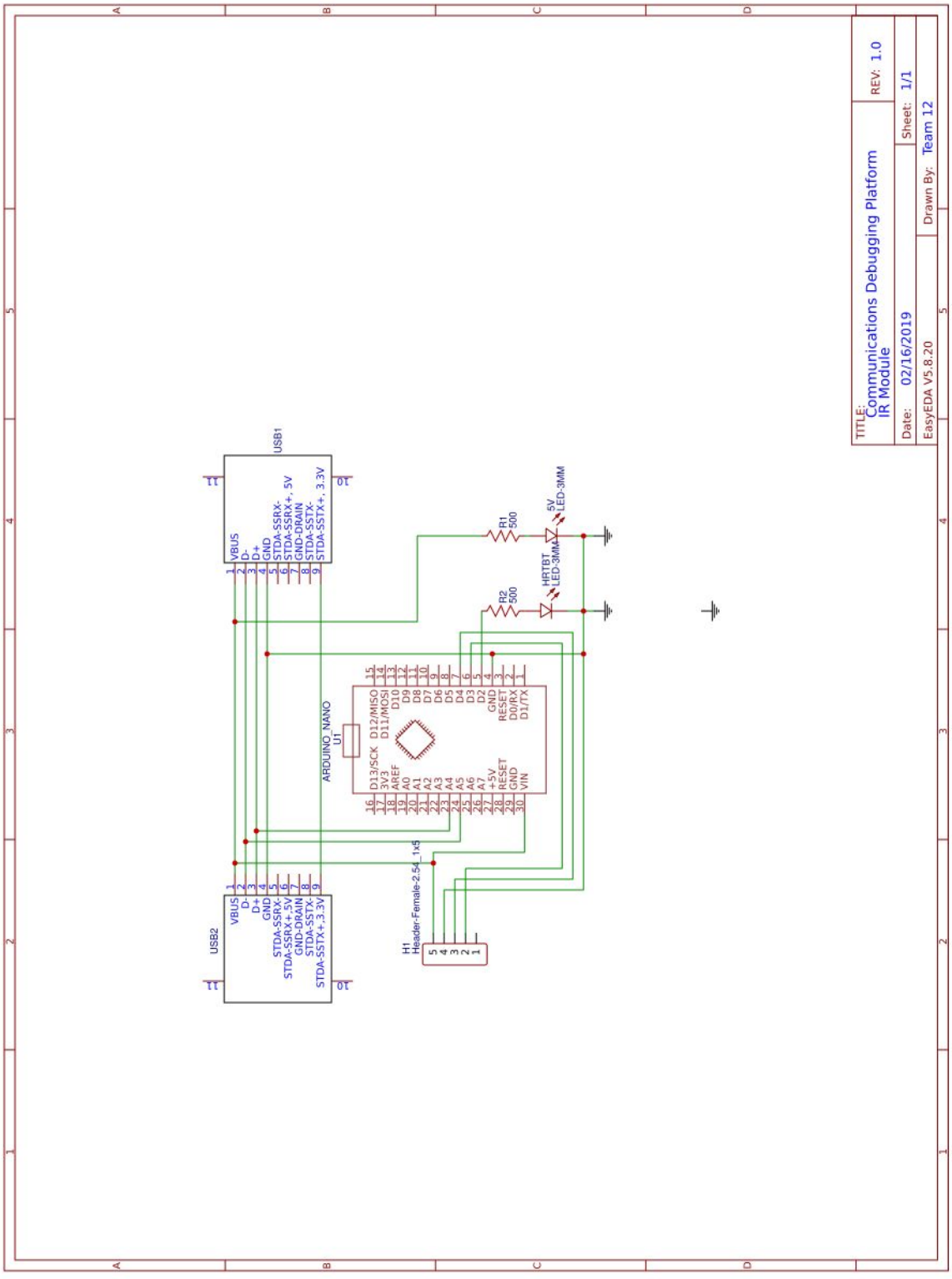
Final Revision Core Tool Schematic



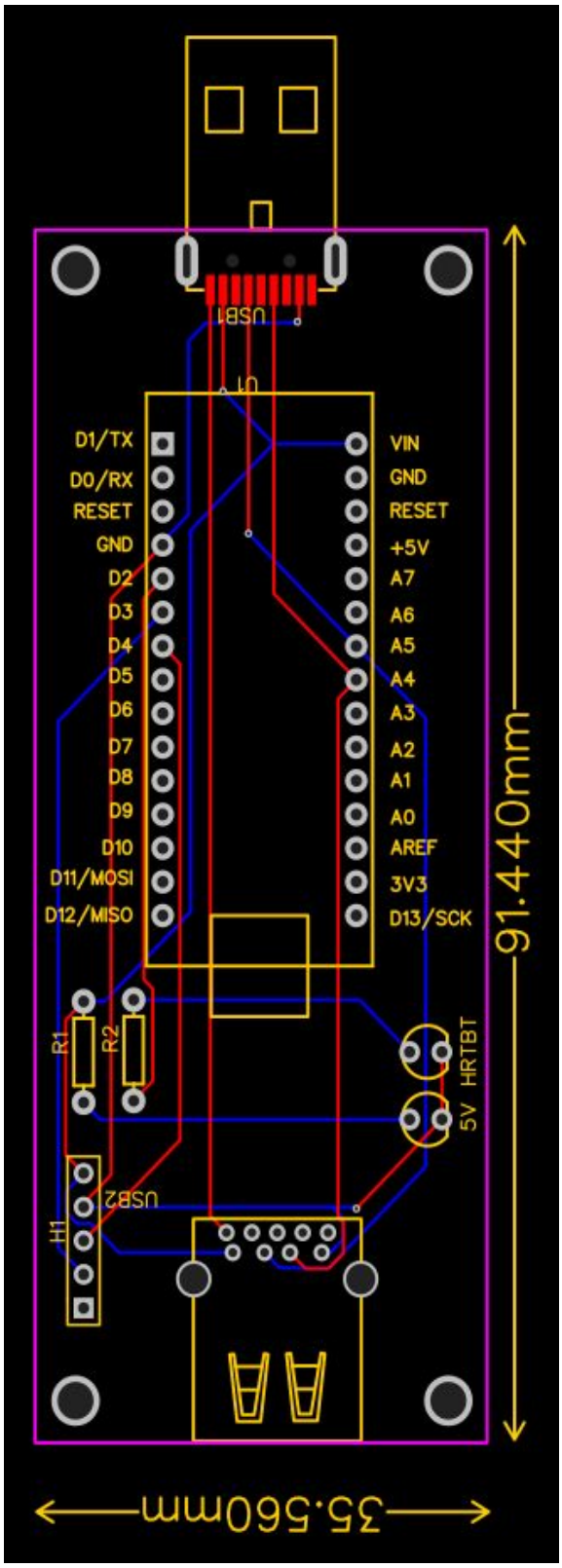
Final Revision Core Tool PCB



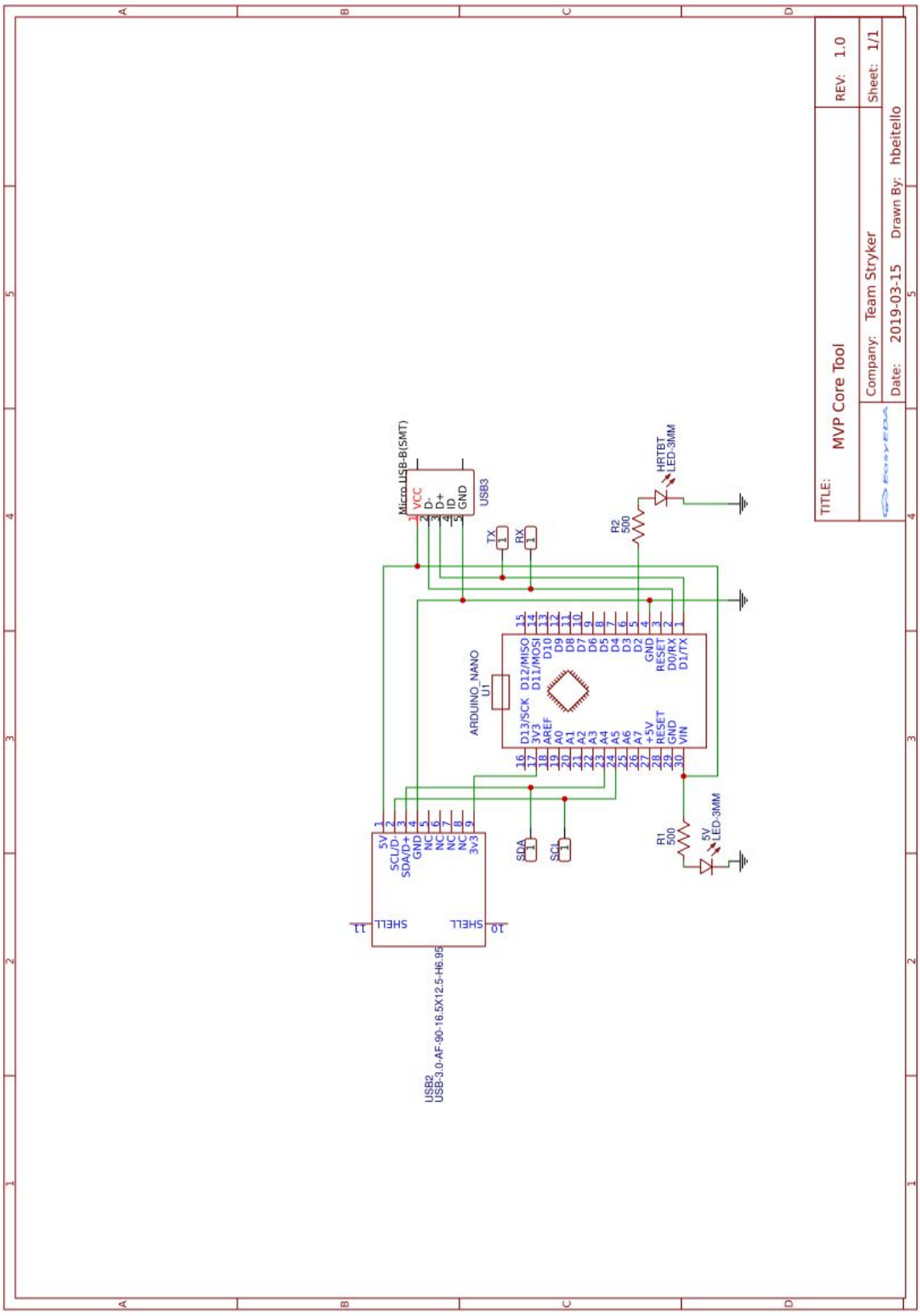
SIM Module Schematic



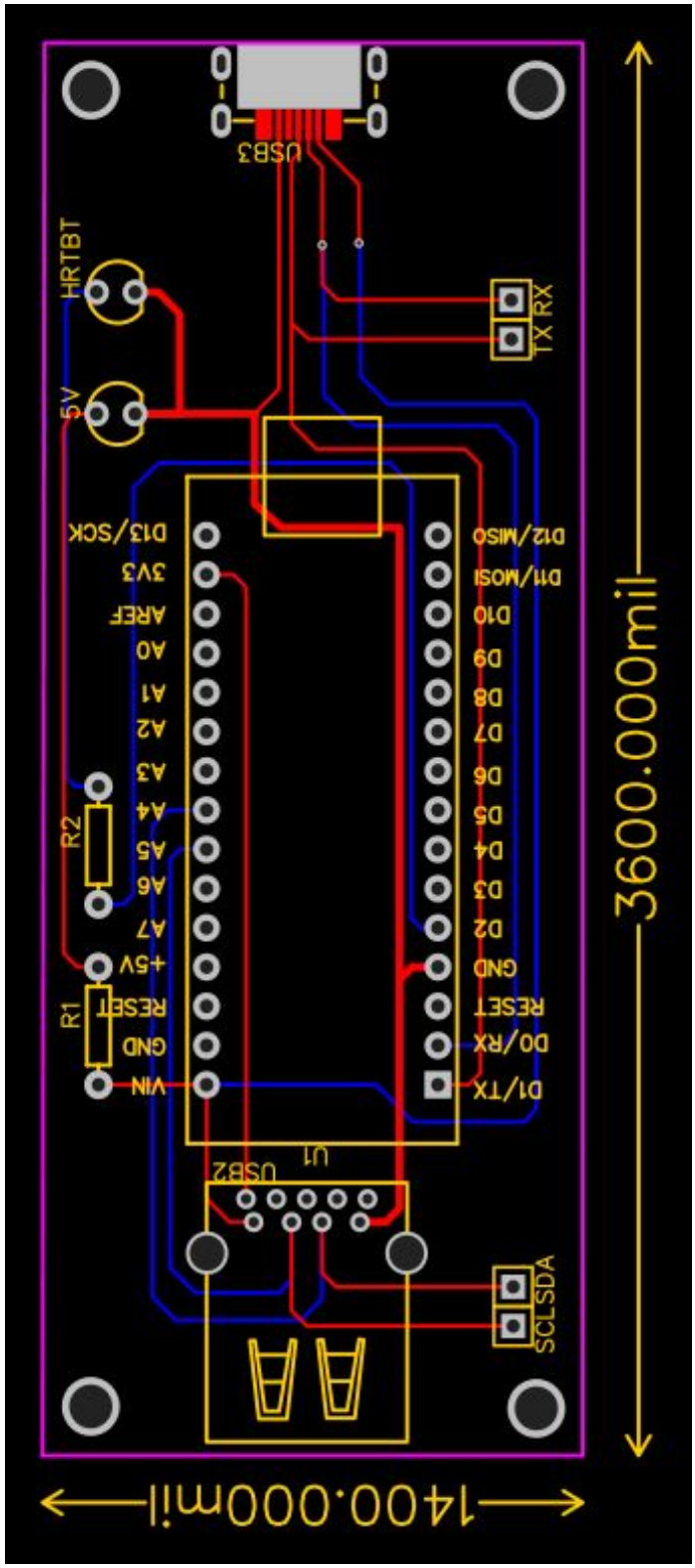
SIM Module PCB



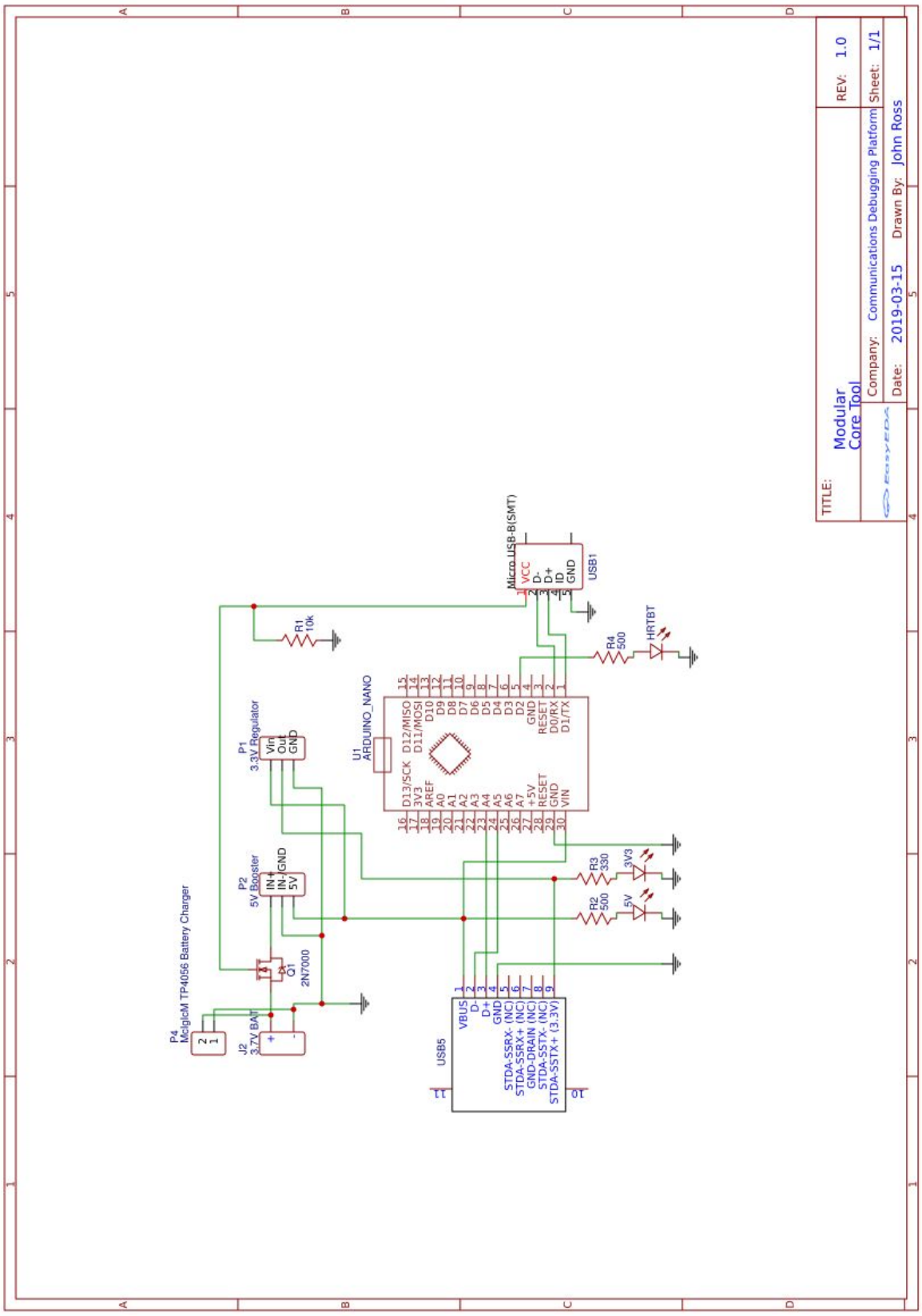
MVP Tool Variant



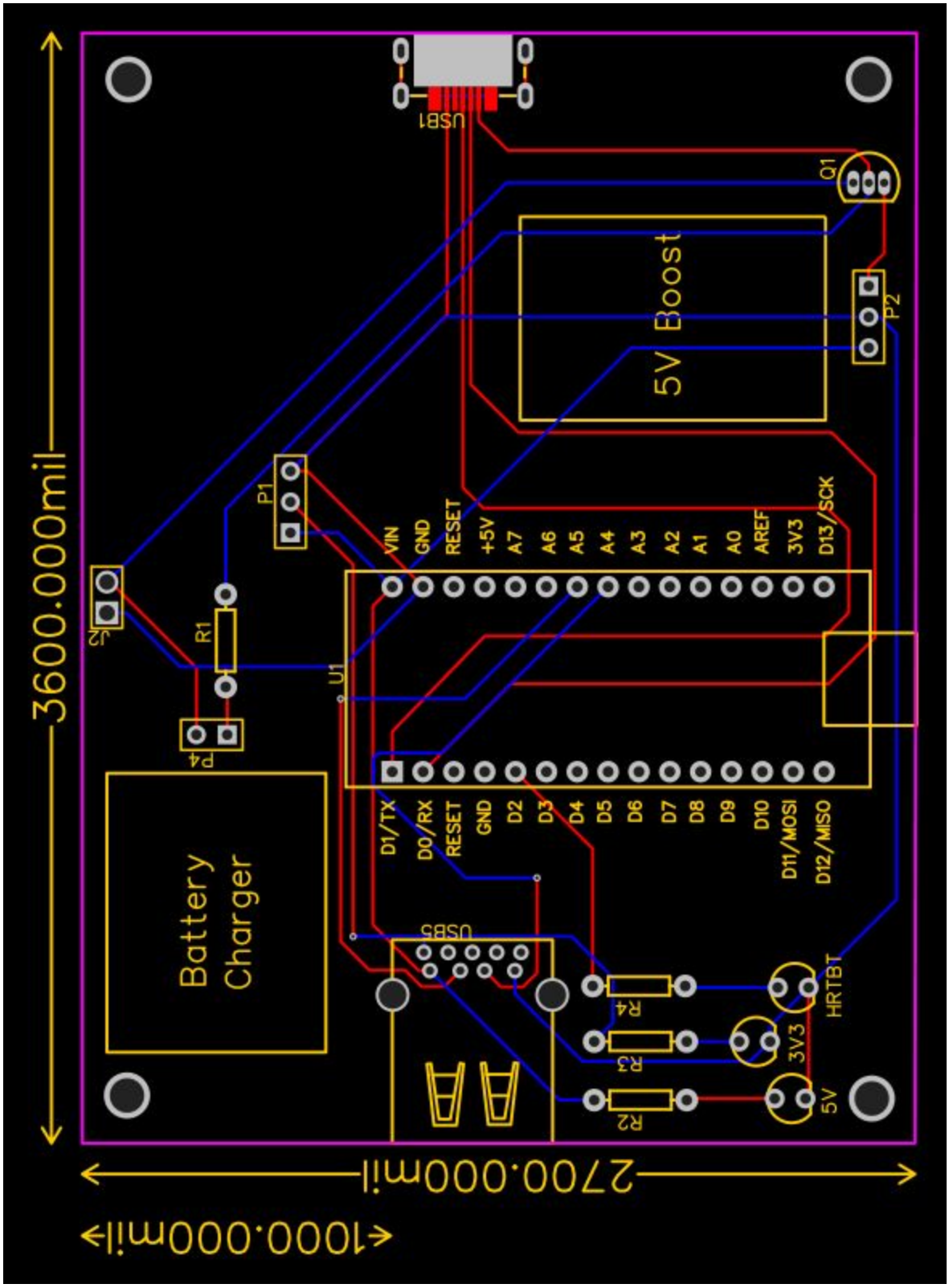
MVP Tool Variant PCB



Modular Tool Variant



Modular Tool Variant PCB



APPENDIX C: RELEVANT SELECTIONS of CODE

Script and Build Objects

Script.java

```
package com.example.sbatzer.cdp_app;

import android.os.Parcel;
import android.os.Parcelable;

import com.google.firebase.database.IgnoreExtraProperties;

import java.io.Serializable;
import java.util.HashMap;
import java.util.Map;

/**
* Script class. Each entry in the database will have one of these.
*/
@IgnoreExtraProperties
public class Script implements Parcelable, Serializable {
   public String name;
   public String desc;
   public String reqNodes;
   public String contents;

   /**
    * Constructor
    * @param name  Script name
    * @param desc  Description of script functionality
    * @param reqNodes  list of hardware nodes that must be connect for script to execute
    * @param contents  action-wait-expected_response commands to be run
    */
   public Script(String name, String desc, String reqNodes, String contents) {
       this.name = name;
       this.desc = desc;
       this.reqNodes = reqNodes;
       this.contents = contents;
   }

   //Parcelable functions
   @Override
   public int describeContents() {
       return 0;
   }

   @Override
   public void writeToParcel(Parcel dest, int flags) {
       dest.writeString(name);
       dest.writeString(desc);
       dest.writeString(reqNodes);
       dest.writeString(contents);
   }

   private Script(Parcel in) {
       this.name = in.readString();
       this.desc = in.readString();
       this.reqNodes = in.readString();
       this.contents = in.readString();
   }

   public static final Parcelable.Creator<Script> CREATOR = new Parcelable.Creator<Script>() {
       @Override
       public Script createFromParcel(Parcel source) {
```

```
            return new Script(source);
        }

        @Override
        public Script[] newArray(int size) {
            return new Script[size];
        }
    };

    public Map<String, Object> outMap() {
        HashMap<String, Object> out = new HashMap<String, Object>();
        out.put("name", name);
        out.put("description", desc);
        out.put("reqnodes", reqNodes);
        out.put("contents", contents);
        return out;
    }
}
```

Build.java

```
package com.example.sbatzer.cdp_app;

import android.os.Parcel;
import android.os.Parcelable;

import com.google.firebase.database.IgnoreExtraProperties;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

@IgnoreExtraProperties
public class Build implements Parcelable, Serializable {
    public String key;
    public String nodeid;
    public String role;
    public String version;

    public Build(String k, String n, String r, String v) {
        this.key = k;
        this.nodeid = n;
        this.role = r;
        this.version = v;
    }

    public Map<String, Object> outMap() {
        HashMap<String, Object> out = new HashMap<String, Object>();
        out.put("nodeid", nodeid);
        out.put("role", role);
        out.put("version", version);
        return out;
    }

    //Parcelable functions
    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(key);
```

```
        dest.writeString(nodeid);
        dest.writeString(role);
        dest.writeString(version);
    }

    private Build(Parcel in) {
        this.key = in.readString();
        this.nodeid = in.readString();
        this.role = in.readString();
        this.version = in.readString();
    }

    public static final Parcelable.Creator<Build> CREATOR = new Parcelable.Creator<Build>() {
        @Override
        public Build createFromParcel(Parcel source) {
            return new Build(source);
        }

        @Override
        public Build[] newArray(int size) {
            return new Build[size];
        }
    };

    public static String GetNodeID(String node, ArrayList<Build> builds) {
        String n = "";
        for (Build b : builds) {
            if (b.role.toUpperCase().equals(node)) {
                n = b.nodeid;
            }
        }
        return n;
    }

    public static String GetNodeName(String node, ArrayList<Build> builds) {
        String n = "";
        for (Build b : builds) {
            if (b.nodeid.equals(node)) {
                n = b.role;
            }
        }
        return n;
    }

    public static String GetNodeNames(String reqNodes, ArrayList<Build> builds) {
        ArrayList<String> names = new ArrayList<String>();

        for (String n : reqNodes.split("-")) {
            for (Build b : builds) {
                if (b.nodeid.equals(n)) {
                    names.add(b.role);
                }
            }
        }

        return Arrays.toString(names.toArray());
    }
}
```

PC App

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace cdp_app_pc
{
    public partial class Form1 : Form
    {
        delegate void SetTextCallback(string text);
        TcpClient client;
        NetworkStream ns;
        Thread t = null;

        public Form1()
        {
            InitializeComponent();
            client = new TcpClient("192.168.4.1", 80);
            ns = client.GetStream();
            String s = "Connected";
            byte[] byteTime = Encoding.ASCII.GetBytes(s);
            ns.Write(byteTime, 0, byteTime.Length);
            t = new Thread(DoWork);
            t.Start();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            String s = textBox1.Text;
            byte[] byteTime = Encoding.ASCII.GetBytes(s);
            ns.Write(byteTime, 0, byteTime.Length);
            ns.Flush();
        }

        public void DoWork()
        {
            byte[] bytes = new byte[1024];
            while (true)
            {
                int bytesRead = ns.Read(bytes, 0, bytes.Length);
                this.SetText(Encoding.ASCII.GetString(bytes, 0, bytesRead));
            }
        }

        private void SetText(string text)
        {
            if (this.textBox2.InvokeRequired)
            {
                SetTextCallback d = new SetTextCallback(SetText);
                this.Invoke(d, new object[] { text });
            }
            else
            {
                this.textBox2.Text = text + this.textBox2.Text;
            }
```

```
            }
        }
}
```

I2C Test with OLED

node1.ino

```
#include <Wire.h>
#include <Arduino.h>
#include <U8x8lib.h>
U8X8_SSD1306_128X64_NONAME_SW_I2C u8x8(12, 11, U8X8_PIN_NONE);

byte x = 0;
bool toSend = false;
char buf[10];

void setup() {
  Wire.begin(1);
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);

  Wire.beginTransmission(2);
  Wire.write(0x00);
  Wire.endTransmission();

  u8x8.begin();
  u8x8.setPowerSave(0);
}

void loop() {
  if (toSend) {
    x++;
    u8x8.setFont(u8x8_font_pxplusibmcgathin_f);
    if (x==0 || x==1) u8x8.clearLine(0);
    itoa(x, buf, 10);
    u8x8.drawString(0,0,buf);
    //Serial.println("Node 1 Transmitting: " + String(x));
    Wire.beginTransmission(2);
    Wire.write(x);
    Wire.endTransmission();
    toSend = false;
  }
}

void receiveEvent(int n) {
  x = Wire.read();
  //Serial.println("Node 1 Received: " + String(x));
  toSend = true;
}
```

node2.ino

```
#include <Wire.h>

byte x = 0;
bool toSend = false;

void setup() {
  Wire.begin(2);
  Wire.onReceive(receiveEvent);
  Serial.begin(9600);

  Wire.beginTransmission(1);
  Wire.write(0x00);
  Wire.endTransmission();
}

void loop() {
  if (toSend) {
```

```
    x++;
    Serial.println("Node 2 Transmitting: " + String(x));
    Wire.beginTransmission(1);
    Wire.write(x);
    Wire.endTransmission();
    toSend = false;
  }
}

void receiveEvent(int n) {
  x = Wire.read();
  Serial.println("Node 2 Received: " + String(x));
  toSend = true;
}
```

Arduino to Android UART Test

Serial_Ping.ino

```
void setup() {
  Serial.begin(9600);
  Serial.setTimeout(10);
  Serial.print(0);
}

void loop() {
  while (Serial.available() == 0);
  int val = Serial.parseInt() + 1;
  if (val > 255) val = 0;
  Serial.print(val);
  Serial.flush();
}
```

Main_Activity.java

```
package com.example.sbatzer.android_serial_test;

import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.io.Console;
import java.lang.ref.WeakReference;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.Set;

public class MainActivity extends AppCompatActivity {

    /*
     * Notifications from UsbService will be received here.
     */
    private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            switch (intent.getAction()) {
                case UsbService.ACTION_USB_PERMISSION_GRANTED: // USB PERMISSION GRANTED
                    Toast.makeText(context, "USB Ready", Toast.LENGTH_SHORT).show();
                    break;
                case UsbService.ACTION_USB_PERMISSION_NOT_GRANTED: // USB PERMISSION NOT GRANTED
                    Toast.makeText(context, "USB Permission not granted",
Toast.LENGTH_SHORT).show();
                    break;
                case UsbService.ACTION_NO_USB: // NO USB CONNECTED
                    Toast.makeText(context, "No USB connected", Toast.LENGTH_SHORT).show();
                    break;
                case UsbService.ACTION_USB_DISCONNECTED: // USB DISCONNECTED
                    Toast.makeText(context, "USB disconnected", Toast.LENGTH_SHORT).show();
```

```
                    break;
                case UsbService.ACTION_USB_NOT_SUPPORTED: // USB NOT SUPPORTED
                    Toast.makeText(context, "USB device not supported",
Toast.LENGTH_SHORT).show();
                    break;
            }
        }
    };
    public UsbService usbService;
    private TextView display;
    private EditText editText;
    private MyHandler mHandler;
    private final ServiceConnection usbConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName arg0, IBinder arg1) {
            usbService = ((UsbService.UsbBinder) arg1).getService();
            usbService.setHandler(mHandler);
        }

        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            usbService = null;
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mHandler = new MyHandler(this);

        display = (TextView) findViewById(R.id.textView1);
        editText = (EditText) findViewById(R.id.editText1);
        Button sendButton = (Button) findViewById(R.id.buttonSend);
        sendButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (!editText.getText().toString().equals("")) {
                    String data = editText.getText().toString();
                    if (usbService != null) { // if UsbService was correctly binded, Send data
                        display.append(data);
                         usbService.write(data.getBytes());
                    }
                }
            }
        });
    }

    @Override
    public void onResume() {
        super.onResume();
        setFilters();  // Start listening notifications from UsbService
        startService(UsbService.class, usbConnection, null); // Start UsbService(if it was not
started before) and Bind it
    }

    @Override
    public void onPause() {
        super.onPause();
        unregisterReceiver(mUsbReceiver);
        unbindService(usbConnection);
    }

    private void startService(Class<?> service, ServiceConnection serviceConnection, Bundle
extras) {
```

```
        if (!UsbService.SERVICE_CONNECTED) {
            Intent startService = new Intent(this, service);
            if (extras != null && !extras.isEmpty()) {
                Set<String> keys = extras.keySet();
                for (String key : keys) {
                    String extra = extras.getString(key);
                    startService.putExtra(key, extra);
                }
            }
            startService(startService);
        }
        Intent bindingIntent = new Intent(this, service);
        bindService(bindingIntent, serviceConnection, Context.BIND_AUTO_CREATE);
    }

    private void setFilters() {
        IntentFilter filter = new IntentFilter();
        filter.addAction(UsbService.ACTION_USB_PERMISSION_GRANTED);
        filter.addAction(UsbService.ACTION_NO_USB);
        filter.addAction(UsbService.ACTION_USB_DISCONNECTED);
        filter.addAction(UsbService.ACTION_USB_NOT_SUPPORTED);
        filter.addAction(UsbService.ACTION_USB_PERMISSION_NOT_GRANTED);
        registerReceiver(mUsbReceiver, filter);
    }

    /*
     * This handler will be passed to UsbService. Data received from serial port is displayed
through this handler
     */
    private static class MyHandler extends Handler {
        private final WeakReference<MainActivity> mActivity;

        public MyHandler(MainActivity activity) {
            mActivity = new WeakReference<>(activity);
        }

        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case UsbService.MESSAGE_FROM_SERIAL_PORT:
                    String data = (String) msg.obj;
                    mActivity.get().display.setText(data);

                    int dataInt = Integer.parseInt(data) + 1;
                    if (dataInt > 255) dataInt = 0;
                    String toSend = Integer.toString(dataInt);

                    if (mActivity.get().usbService != null) {
                        mActivity.get().usbService.write(toSend.getBytes());
                    }
                    break;
            }
        }
    }

    private final static char[] hexArray = "0123456789ABCDEF".toCharArray();
    public static String bytesToHex(byte[] bytes) {
        char[] hexChars = new char[bytes.length * 2];
        for ( int j = 0; j < bytes.length; j++ ) {
            int v = bytes[j] & 0xFF;
            hexChars[j * 2] = hexArray[v >>> 4];
            hexChars[j * 2 + 1] = hexArray[v & 0x0F];
        }
        return new String(hexChars);
    }
```

```
    public static byte[] intToBytes(int i, int s) {
        return ByteBuffer.allocate(s).putInt(i).array();
    }

    public static void reverse(byte[] array) {
        if (array == null) {
            return;
        }
        int i = 0;
        int j = array.length - 1;
        byte tmp;
        while (j > i) {
            tmp = array[j];
            array[j] = array[i];
            array[i] = tmp;
            j--;
            i++;
        }
    }
}
```

UsbService.java

```
package com.example.sbatzer.android_serial_test;

import android.app.PendingIntent;
import android.app.Service;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.hardware.usb.UsbDevice;
import android.hardware.usb.UsbDeviceConnection;
import android.hardware.usb.UsbManager;
import android.os.Binder;
import android.os.Handler;
import android.os.IBinder;

import com.felhr.usbserial.CDCSerialDevice;
import com.felhr.usbserial.UsbSerialDevice;
import com.felhr.usbserial.UsbSerialInterface;

import java.io.UnsupportedEncodingException;
import java.util.HashMap;
import java.util.Map;

public class UsbService extends Service {

    public static final String ACTION_USB_READY = "com.felhr.connectivityservices.USB_READY";
    public static final String ACTION_USB_ATTACHED =
"android.hardware.usb.action.USB_DEVICE_ATTACHED";
    public static final String ACTION_USB_DETACHED =
"android.hardware.usb.action.USB_DEVICE_DETACHED";
    public static final String ACTION_USB_NOT_SUPPORTED =
"com.felhr.usbservice.USB_NOT_SUPPORTED";
    public static final String ACTION_NO_USB = "com.felhr.usbservice.NO_USB";
    public static final String ACTION_USB_PERMISSION_GRANTED =
"com.felhr.usbservice.USB_PERMISSION_GRANTED";
    public static final String ACTION_USB_PERMISSION_NOT_GRANTED =
"com.felhr.usbservice.USB_PERMISSION_NOT_GRANTED";
    public static final String ACTION_USB_DISCONNECTED = "com.felhr.usbservice.USB_DISCONNECTED";
    public static final String ACTION_CDC_DRIVER_NOT_WORKING =
"com.felhr.connectivityservices.ACTION_CDC_DRIVER_NOT_WORKING";
    public static final String ACTION_USB_DEVICE_NOT_WORKING =
"com.felhr.connectivityservices.ACTION_USB_DEVICE_NOT_WORKING";
    public static final int MESSAGE_FROM_SERIAL_PORT = 0;
    private static final String ACTION_USB_PERMISSION = "com.android.example.USB_PERMISSION";
```

```
    private static final int BAUD_RATE = 9600; // BaudRate. Change this value if you need
    public static boolean SERVICE_CONNECTED = false;

    private IBinder binder = new UsbBinder();

    private Context context;
    private Handler mHandler;
    private UsbManager usbManager;
    private UsbDevice device;
    private UsbDeviceConnection connection;
    private UsbSerialDevice serialPort;

    private boolean serialPortConnected;
    /*
     *  Data received from serial port will be received here. Just populate onReceivedData with
your code
     *  In this particular example. byte stream is converted to String and send to UI thread to
     *  be treated there.
     */
    private UsbSerialInterface.UsbReadCallback mCallback = new
UsbSerialInterface.UsbReadCallback() {
        @Override
        public void onReceivedData(byte[] arg0) {
            try {
                String data = new String(arg0, "UTF-8");
                if (mHandler != null)
                    mHandler.obtainMessage(MESSAGE_FROM_SERIAL_PORT, data).sendToTarget();
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            }
        }
    };
    /*
     * Different notifications from OS will be received here (USB attached, detached, permission
responses...)
     * About BroadcastReceiver:
http://developer.android.com/reference/android/content/BroadcastReceiver.html
     */
    private final BroadcastReceiver usbReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context arg0, Intent arg1) {
            if (arg1.getAction().equals(ACTION_USB_PERMISSION)) {
                boolean granted =
arg1.getExtras().getBoolean(UsbManager.EXTRA_PERMISSION_GRANTED);
                if (granted) // User accepted our USB connection. Try to open the device as a
serial port
                {
                    Intent intent = new Intent(ACTION_USB_PERMISSION_GRANTED);
                    arg0.sendBroadcast(intent);
                    connection = usbManager.openDevice(device);
                    serialPortConnected = true;
                    new ConnectionThread().run();
                } else // User not accepted our USB connection. Send an Intent to the Main
Activity
                {
                    Intent intent = new Intent(ACTION_USB_PERMISSION_NOT_GRANTED);
                    arg0.sendBroadcast(intent);
                }
            } else if (arg1.getAction().equals(ACTION_USB_ATTACHED)) {
                if (!serialPortConnected)
                    findSerialPortDevice(); // A USB device has been attached. Try to open it as
a Serial port
            } else if (arg1.getAction().equals(ACTION_USB_DETACHED)) {
                // Usb device was disconnected. send an intent to the Main Activity
                Intent intent = new Intent(ACTION_USB_DISCONNECTED);
                arg0.sendBroadcast(intent);
```

```
                serialPortConnected = false;
                serialPort.close();
            }
        }
    };

    /*
     * onCreate will be executed when service is started. It configures an IntentFilter to listen
for
     * incoming Intents (USB ATTACHED, USB DETACHED...) and it tries to open a serial port.
     */
    @Override
    public void onCreate() {
        this.context = this;
        serialPortConnected = false;
        UsbService.SERVICE_CONNECTED = true;
        setFilter();
        usbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
        findSerialPortDevice();
    }

    /* MUST READ about services
     * http://developer.android.com/guide/components/services.html
     * http://developer.android.com/guide/components/bound-services.html
     */
    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        return Service.START_NOT_STICKY;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        UsbService.SERVICE_CONNECTED = false;
    }

    /*
     * This function will be called from MainActivity to write data through Serial Port
     */
    public void write(byte[] data) {
        if (serialPort != null)
            serialPort.write(data);
    }

    public void setHandler(Handler mHandler) {
        this.mHandler = mHandler;
    }

    private void findSerialPortDevice() {
        // This snippet will try to open the first encountered usb device connected, excluding usb
root hubs
        HashMap<String, UsbDevice> usbDevices = usbManager.getDeviceList();
        if (!usbDevices.isEmpty()) {
            boolean keep = true;
            for (Map.Entry<String, UsbDevice> entry : usbDevices.entrySet()) {
                device = entry.getValue();
                int deviceVID = device.getVendorId();
                int devicePID = device.getProductId();

                if (deviceVID != 0x1d6b && (devicePID != 0x0001 || devicePID != 0x0002 ||
devicePID != 0x0003)) {
```

```
                    // There is a device connected to our Android device. Try to open it as a
Serial Port.
                    requestUserPermission();
                    keep = false;
                } else {
                    connection = null;
                    device = null;
                }

                if (!keep)
                    break;
            }
            if (!keep) {
                // There is no USB devices connected (but usb host were listed). Send an intent to
MainActivity.
                Intent intent = new Intent(ACTION_NO_USB);
                sendBroadcast(intent);
            }
        } else {
            // There is no USB devices connected. Send an intent to MainActivity
            Intent intent = new Intent(ACTION_NO_USB);
            sendBroadcast(intent);
        }
    }

    private void setFilter() {
        IntentFilter filter = new IntentFilter();
        filter.addAction(ACTION_USB_PERMISSION);
        filter.addAction(ACTION_USB_DETACHED);
        filter.addAction(ACTION_USB_ATTACHED);
        registerReceiver(usbReceiver, filter);
    }

    /*
     * Request user permission. The response will be received in the BroadcastReceiver
     */
    private void requestUserPermission() {
        PendingIntent mPendingIntent = PendingIntent.getBroadcast(this, 0, new
Intent(ACTION_USB_PERMISSION), 0);
        usbManager.requestPermission(device, mPendingIntent);
    }

    public class UsbBinder extends Binder {
        public UsbService getService() {
            return UsbService.this;
        }
    }

    /*
     * A simple thread to open a serial port.
     * Although it should be a fast operation. moving usb operations away from UI thread is a good
thing.
     */
    private class ConnectionThread extends Thread {
        @Override
        public void run() {
            serialPort = UsbSerialDevice.createUsbSerialDevice(device, connection);
            if (serialPort != null) {
                if (serialPort.open()) {
                    serialPort.setBaudRate(BAUD_RATE);
                    serialPort.setDataBits(UsbSerialInterface.DATA_BITS_8);
                    serialPort.setStopBits(UsbSerialInterface.STOP_BITS_1);
                    serialPort.setParity(UsbSerialInterface.PARITY_NONE);
                    serialPort.setFlowControl(UsbSerialInterface.FLOW_CONTROL_OFF);
                    serialPort.read(mCallback);
```

```
                    // Everything went as expected. Send an intent to MainActivity
                    Intent intent = new Intent(ACTION_USB_READY);
                    context.sendBroadcast(intent);
                } else {
                    // Serial port could not be opened, maybe an I/O error or if CDC driver was
chosen, it does not really fit
                    // Send an Intent to Main Activity
                    if (serialPort instanceof CDCSerialDevice) {
                        Intent intent = new Intent(ACTION_CDC_DRIVER_NOT_WORKING);
                        context.sendBroadcast(intent);
                    } else {
                        Intent intent = new Intent(ACTION_USB_DEVICE_NOT_WORKING);
                        context.sendBroadcast(intent);
                    }
                }
            } else {
                // No driver for given device, even generic CDC driver could not be loaded
                Intent intent = new Intent(ACTION_USB_NOT_SUPPORTED);
                context.sendBroadcast(intent);
            }
        }
    }
}
```

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.sbatzer.android_serial_test.MainActivity">

    <TextView
        android:id="@+id/textViewTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:text="@string/serial_port"
        android:textAppearance="?android:attr/textAppearanceLarge"/>

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="150dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="34dp"
        android:background="#FFFFFF"/>

    <EditText
        android:id="@+id/editText1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView1"
        android:layout_alignRight="@+id/textView1"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="28dp"
        android:ems="10">

        <requestFocus/>
    </EditText>

    <Button
```

```
        android:id="@+id/buttonSend"
        style="?borderlessButtonStyle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editText1"
        android:layout_alignRight="@+id/editText1"
        android:layout_below="@+id/editText1"
        android:layout_marginTop="42dp"
        android:text="Send"/>
</RelativeLayout>
```

CDP App Activities

ScriptExecutionActivity.java

```
package com.example.sbatzer.cdp_app;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.ServiceConnection;
import android.media.MediaPlayer;
import android.net.wifi.WifiConfiguration;
import android.net.wifi.WifiManager;
import android.os.AsyncTask;
import android.os.CountDownTimer;
import android.os.Environment;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.util.AttributeSet;
import android.util.Log;
import android.util.Xml;
import android.view.Gravity;
import android.view.KeyEvent;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.example.sbatzer.cdp_app.graphview.GraphView;
import com.example.sbatzer.cdp_app.graphview.series.DataPoint;
import com.example.sbatzer.cdp_app.graphview.series.LineGraphSeries;

import org.xmlpull.v1.XmlPullParser;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;
import java.lang.ref.WeakReference;
import java.lang.reflect.Method;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Locale;
import java.util.Queue;
import java.util.Set;
import java.util.Timer;
import java.util.TimerTask;
```

```
import java.util.UUID;

interface OnEditTextEvent {
    void OnEditText(String s);
}

interface OnButtonPressEvent {
    void OnButtonPress();
}

interface OnMessageReceiveEvent {
    void OnMessageReceive(String s);
}

public class ScriptExecutionActivity extends AppCompatActivity {

    LinearLayout execList;
    Script script;
    ArrayList<Build> builds = new ArrayList<>();
    HashMap<String, DataItem> CDS = new HashMap<>();

    String logName;

    HashMap<String, ArrayList<OnEditTextEvent>> EditTextListeners = new HashMap<>();
    HashMap<String, ArrayList<OnButtonPressEvent>> ButtonPressListeners = new HashMap<>();
    HashMap<String, ArrayList<OnMessageReceiveEvent>> MessageRxListeners = new HashMap<>();
    HashMap<String, MediaPlayer> AudioManager = new HashMap<>();
    HashMap<String, ArrayList<CommandEvent>> CommandLists = new HashMap<>();
    HashMap<String, LineGraphSeries<DataPoint>> GraphSeries = new HashMap<>();

    TcpClient tcpClient;

    private SerialHandler sHandler;
    BluetoothManager mBluetooth;
    public UsbService usbService;
    Boolean sppConnect = false;
    Boolean usbConnect = false;
    // Tablet 1:  2C:FD:AB:83:11:00
    // Tablet 2:  2C:FD:AB:83:0F:0A
    // Headset:   BC:F2:92:5E:78:94
    // Devkit 1:  00:07:80:CC:7C:63

    BluetoothAudioManager audioManager;
    boolean BTReceiverRegistered = false;

    boolean sppByteMode = true;
    String sppLastMsg = "";
    int sppByteTypeIndex;
    int sppByteStartVal;
    int sppByteEndVal;
    int sppByteKeyIndex;
    int sppByteLenIndex;
    int sppByteDatIndex;
    boolean sppAck = true;

    boolean graphPresent = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_script_execution);

        script = getIntent().getParcelableExtra("script");
        builds = getIntent().getParcelableArrayListExtra("builds");

        String name = script.name.replace(' ', '_').toLowerCase();
```

```
        String date = new SimpleDateFormat("yyyy-MM-dd_HH:mm:ss",
Locale.getDefault()).format(new Date());
        logName = name + "_" + date + ".txt";

        loadAudio();

        execList = findViewById(R.id.execList);
        createScriptHeader();
        parseScript();

        mBluetooth = BluetoothManager.GetManager();
        mBluetooth.SetHandler(btHandler);
        sHandler = new SerialHandler(this);

        if (!graphPresent) {
            findViewById(R.id.execList2).setVisibility(View.GONE);
        }
//        registerBTAudioIntentReceiver();
//        audioManager = new BluetoothAudioManager(this);
    }

    protected void onDestroy() {
        super.onDestroy();
//        unregisterReceiver(intentReceiver);
    }

//    private void ConnectBTAudio(String mac) {
//        audioManager.startConnection(mac);
//    }

    private void ConnectBTAudio(String mac) {
        BluetoothSocket mSocket = null;
        try {
            mSocket =
BluetoothAdapter.getDefaultAdapter().getRemoteDevice(mac).createInsecureRfcommSocketToServiceR
ecord(UUID.fromString("0000110B-0000-1000-8000-00805F9B34FB"));
        } catch (IOException e1) {
            // TODO Auto-generated catch block
            Log.d("SCRIPT_EXEC","socket not created");
            e1.printStackTrace();
        }
        try{

            mSocket.connect();

        }
        catch(IOException e){
            try {
                mSocket.close();
                Log.d("SCRIPT_EXEC","Cannot connect");
            } catch (IOException e1) {
                Log.d("SCRIPT_EXEC","Socket not closed");
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }
    }

    private void registerBTAudioIntentReceiver() {
        IntentFilter intentFilter = new IntentFilter();
        intentFilter.addAction(BluetoothDevice.ACTION_FOUND);
        intentFilter.addAction(BluetoothDevice.ACTION_ACL_CONNECTED);
        intentFilter.addAction(BluetoothDevice.ACTION_ACL_DISCONNECTED);
        intentFilter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
        intentFilter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        intentFilter.addAction(BluetoothDevice.ACTION_UUID);
```

```
        if(!BTReceiverRegistered) {
            registerReceiver(intentReceiver, intentFilter);
            BTReceiverRegistered = true;
        }
    }

    private void loadAudio() {
        AudioManager.put("fanfare", MediaPlayer.create(this, R.raw.victory));
        AudioManager.put("jump", MediaPlayer.create(this, R.raw.jump));
        AudioManager.put("wilhelm", MediaPlayer.create(this, R.raw.wilhelm));
    }

    private void createScriptHeader() {
        TextView info = new TextView(this);
        info.setText(String.format("Name: %s", script.name));
        info.append(String.format("\nReq. Nodes: %s", Build.GetNodeNames(script.reqNodes,
builds)));
        info.append(String.format("\nDesc: %s", script.desc));
        info.setTag("Info");
        execList.addView(info);
    }

    private void parseScript() {
        for (String line: script.contents.split("\n")) {
            final String[] t = line.split(" ");
            if (t[0].equals("REG")) {
                if (t[2].equals("String")) {
                    CDS.put(t[1], new DataItem<String>(""));
                }
                else if (t[2].equals("Float")) {
                    CDS.put(t[1], new DataItem<Float>(0.0f));
                }
                if (t.length > 3) {
                    if (t[3].equals("TOOLSRC")) {
                        if (MessageRxListeners.get("tool_"+t[4]) == null) {
                            MessageRxListeners.put("tool_"+t[4], new
ArrayList<OnMessageReceiveEvent>());
                        }
                        MessageRxListeners.get("tool_"+t[4]).add(new OnMessageReceiveEvent() {
                            @Override
                            public void OnMessageReceive(String s) {
                                CDS.get(t[1]).SetItem(s);
                            }
                        });
                    }
                    else if (t[3].equals("WIFISRC")) {
                        if (MessageRxListeners.get("wifi_"+t[4]) == null) {
                            MessageRxListeners.put("wifi_"+t[4], new
ArrayList<OnMessageReceiveEvent>());
                        }
                        MessageRxListeners.get("wifi_"+t[4]).add(new OnMessageReceiveEvent() {
                            @Override
                            public void OnMessageReceive(String s) {
                                CDS.get(t[1]).SetItem(s);
                            }
                        });
                    }
                    else if (t[3].equals("SPPSRC")) {
                        if (MessageRxListeners.get("spp") == null) {
                            MessageRxListeners.put("spp", new
ArrayList<OnMessageReceiveEvent>());
                        }
                        MessageRxListeners.get("spp").add(new OnMessageReceiveEvent() {
                            @Override
                            public void OnMessageReceive(String s) {
                                CDS.get(t[1]).SetItem(s);
```

```
                if (sppAck) {
                    if (sppByteMode) {
                        byte[] data = DataUtilities.hexToBytes(s);
                        if (data[sppByteTypeIndex] == 0x01) {
                            data[sppByteTypeIndex] = (byte) 0x03;
                            SppTx(data);
                        } else if (data[sppByteTypeIndex] == 0x03) {
                            SppTx(data);
                        }
                    } else {
                        String[] x = s.split(" ");
                        if (x.length > 1) {
                            if (x[0].equals("SET")) {
                                x[0] = "ACK";
                                String tx = "";
                                for (int i = 0; i < x.length; i++) {
                                    tx += x[i] + " ";
                                }
                                SppTx(tx.trim());
                            }
                        }
                    }
                }
            }
        });
    }
    else if (t[3].equals("INIT")) {
        CDS.get(t[1]).SetItem(t[4]);
    }
    else if (t[3].equals("PAYLOAD")) {
        if (t[4].equals("CAN")) {
            CDS.get(t[5]).RegisterOnSetEvent(new OnSetEvent() {
                @Override
                public void OnSet(Object item, Object prev) {
                    String sink = item.toString();
                    String[] sT = sink.split(",");
                    if (sT.length == 2) {
                        String pdo = sT[0];
                        String dat = sT[1];
                        if (pdo.equals(t[6])) {
                            int start = Integer.parseInt(t[7]);
                            int length = Integer.parseInt(t[8]);
                            byte[] bytes = DataUtilities.hexToBytes(dat);
                            byte[] trgt = new byte[length];
                            System.arraycopy(bytes, start, trgt, 0, length);
                            if (t[2].equals("Float")) {
                                DataUtilities.reverse(trgt);
                                Float val = DataUtilities.bytesToFloat(trgt);
                                CDS.get(t[1]).SetItem(val);
                                LogData(t[1] + "-" + val.toString());
                            }
                        }
                    }
                }
            });
        }
        else if (t[4].equals("SPP")) {
            CDS.get(t[5]).RegisterOnSetEvent(new OnSetEvent() {
                @Override
                public void OnSet(Object item, Object prev) {
                    String sink = item.toString();
                    byte[] data = DataUtilities.hexToBytes(sink);
                    int trgt = Integer.parseInt(t[6]);
                    if ((int)data[sppByteKeyIndex] == trgt) {
                        CDS.get(t[1]).SetItem(sink);
                    }
```

```
                                }
                            });
                        }
                    }
                }
            }
            else if (t[0].equals("GUI")) {
                if (t[1].equals("LBL")) {
                    drawTextView(t[2], t[2]);
                    if (t.length > 3) {
                        if (t[3].equals("UPD")) {
                            CDS.get(t[4]).RegisterOnSetEvent(new OnSetEvent() {
                                @Override
                                public void OnSet(Object item, Object prev) {
                                    TextView view = (TextView)execList.findViewWithTag(t[2]);
                                    view.setText(String.format("%s: %s", t[2],
item.toString()));
                                }
                            });
                        }
                        else if (t[3].equals("LOG")) {
                            CDS.get(t[4]).RegisterOnSetEvent(new OnSetEvent() {
                                @Override
                                public void OnSet(Object item, Object prev) {
                                    TextView view = (TextView)execList.findViewWithTag(t[2]);
                                    view.setText(String.format("%s\n%s", item.toString(),
view.getText())); // prepend
//                                      view.append(String.format("\n%s", item.toString()));
// append
                                }
                            });
                        }
                    }
                }
                else if (t[1].equals("BTN")) {
                    drawButton(t[3].replace('`', ' '), t[2]);
                    if (t.length > 4) {
                        if (t[4].equals("SYC")) {
                            ButtonPressListeners.get(t[2]).add(new OnButtonPressEvent() {
                                @Override
                                public void OnButtonPress() {
                                    TextView view = (TextView)execList.findViewWithTag(t[5]);
                                    view.setText(String.format("%s: %s", t[5],
CDS.get(t[6]).GetItem().toString()));
                                }
                            });
                        }
                        else if (t[4].equals("USBCNT")) {
                            ButtonPressListeners.get(t[2]).add(new OnButtonPressEvent() {
                                @Override
                                public void OnButtonPress() {
                                    OpenUSB();
                                }
                            });
                        }
                        else if (t[4].equals("SPPCNT")) {
                            ButtonPressListeners.get(t[2]).add(new OnButtonPressEvent() {
                                @Override
                                public void OnButtonPress() {
                                    String mac = CDS.get(t[5]).GetItem().toString();
                                    OpenSPP(mac);
                                }
                            });
                        }
                        else if (t[4].equals("WIFICNT")) {
                            ButtonPressListeners.get(t[2]).add(new OnButtonPressEvent() {
```

```
                            @Override
                            public void OnButtonPress() {
                                openTcp();
                            }
                        });
                    }
                    else if (t[4].equals("SPPTX")) {
                        ButtonPressListeners.get(t[2]).add(new OnButtonPressEvent() {
                            @Override
                            public void OnButtonPress() {
                                if (sppByteMode) {
                                    byte[] msg =
DataUtilities.hexToBytes(CDS.get(t[5]).GetItem().toString());
                                    SppTx(msg);
                                }
                                else {
                                    String msg = CDS.get(t[5]).GetItem().toString();
                                    SppTx(msg);
                                }
                            }
                        });
                    }
                    else if (t[4].equals("USBTX")) {
                        ButtonPressListeners.get(t[2]).add(new OnButtonPressEvent() {
                            @Override
                            public void OnButtonPress() {
                                String msg = CDS.get(t[6]).GetItem().toString();
                                String id = Build.GetNodeID(t[5], builds);
                                String tx = id + "-" + msg;
                                UsbTx(tx.trim());
                            }
                        });
                    }
                    else if (t[4].equals("WIFITX")) {
                        ButtonPressListeners.get(t[2]).add(new OnButtonPressEvent() {
                            @Override
                            public void OnButtonPress() {
                                String msg = CDS.get(t[6]).GetItem().toString();
                                String id = Build.GetNodeID(t[5], builds);
                                String tx = id + "-" + msg;
                                writeTcp(tx.trim());
                            }
                        });
                    }
                    else if (t[4].equals("AUDIO")) {
                        ButtonPressListeners.get(t[2]).add(new OnButtonPressEvent() {
                            @Override
                            public void OnButtonPress() {
                                AudioManager.get(t[5]).start();
                            }
                        });
                    }
                    else if (t[4].equals("AUDCNT")) {
                        ButtonPressListeners.get(t[2]).add(new OnButtonPressEvent() {
                            @Override
                            public void OnButtonPress() {
                                ConnectBTAudio(CDS.get(t[5]).GetItem().toString());
                            }
                        });
                    }
                    else if (t[4].equals("CMD")) { // GUI BTN initCmdList Send`Cmd`List
CMD sampleList
                        ButtonPressListeners.get(t[2]).add(new OnButtonPressEvent() {
                            @Override
                            public void OnButtonPress() {
                                ArrayList<CommandEvent> cmds = CommandLists.get(t[5]);
```

```
                                if (cmds.size() > 0) {
                                    cmds.get(0).OnTx();
                                }
                            }
                        });
                    }
                }
            }
            else if (t[1].equals("TXT")) {
                drawEditText(t[2], t[2]);
                if (t.length > 3) {
                    if (t[3].equals("STR")) {
                        EditTextListeners.get(t[2]).add(new OnEditTextEvent() {
                            @Override
                            public void OnEditText(String s) {
                                CDS.get(t[4]).SetItem(s);
                            }
                        });
                    }
                }
            }
            else if (t[1].equals("GPH")) {
                drawGraph(t[2], Integer.parseInt(t[5]),  Integer.parseInt(t[6]),
Integer.parseInt(t[7]), Integer.parseInt(t[8]));
                CDS.get(t[4]).RegisterOnSetEvent(new OnSetEvent() {
                    @Override
                    public void OnSet(Object item, Object prev) {
                        GraphSeries.get(t[2]).appendData(new
DataPoint(GraphSeries.get(t[2]).getHighestValueX()+1, (float)item), true,
Integer.parseInt(t[6]));
                    }
                });
            }
        }
        else if (t[0].equals("CMD")) {
            if (t[1].equals("CND")) {
                if (t[3].equals("ABV")) {
                    CDS.get(t[2]).RegisterOnSetEvent(new OnSetEvent() {
                        @Override
                        public void OnSet(Object item, Object prev) {
                            float val = Float.parseFloat(t[4]);
                            if (((float)item > val) && ((float)prev < val)) {
                                if (t[5].equals("AUDIO")) {
                                    AudioManager.get(t[6]).start();
                                    LogData("Play Audio - " + t[6]);
                                }
                            }
                        }
                    });
                }
                else if (t[3].equals("BEL")) {
                    CDS.get(t[2]).RegisterOnSetEvent(new OnSetEvent() {
                        @Override
                        public void OnSet(Object item, Object prev) {
                            float val = Float.parseFloat(t[4]);
                            if (((float)item < val) && ((float)prev > val)) {
                                if (t[5].equals("AUDIO")) {
                                    AudioManager.get(t[6]).start();
                                    LogData("Play Audio - " + t[6]);
                                }
                            }
                        }
                    });
                }
            }
            else if (t[1].equals("RES")) {
```

```
                    CDS.get(t[2]).RegisterOnSetEvent(new OnSetEvent() {
                        @Override
                        public void OnSet(Object item, Object prev) {
                            if (t[3].equals("SPPTX")) {
                                final byte[] data =
DataUtilities.hexToBytes(CDS.get(t[4]).GetItem().toString());
                                if (t[5].equals("DEL")) {
                                    new Handler().postDelayed(new Runnable() {
                                        @Override
                                        public void run() {
                                            SppTx(data);
                                        }
                                    }, Integer.parseInt(t[6]));
                                }
                                else {
                                    SppTx(data);
                                }
                            }
                        }
                    });
                }
                else if (t[1].equals("SPP")) {
                    if (t[2].equals("ACK")) {
                        sppAck = true;
                    }
                    else if (t[2].equals("NACK")) {
                        sppAck = false;
                    }
                    if (t[3].equals("BYTE")) {
                        sppByteMode = true;
                        sppByteTypeIndex = Integer.parseInt(t[4]);
                        sppByteKeyIndex = Integer.parseInt(t[5]);
                        sppByteLenIndex = Integer.parseInt(t[6]);
                        sppByteDatIndex = Integer.parseInt(t[7]);
                        sppByteStartVal = Integer.parseInt(t[8]);
                        sppByteEndVal = Integer.parseInt(t[9]);
                    }
                    else if (t[3].equals("STRING")) {
                        sppByteMode = false;
                    }
                }
                else if (t[1].equals("ELE")) { // ie: CMD ELE sampleList SPPTX SET`1`ON RX
2500 sppSink REP
                    if (CommandLists.get(t[2]) == null) {
                        CommandLists.put(t[2], new ArrayList<CommandEvent>());
                    }
                    final String txMsg = t[4].replace("`"," ");       // Value to be sent
                    final String rxMsg = txMsg.replace("SET", "ACK"); // Value expected to
receive
                    final ArrayList<CommandEvent> cmds = CommandLists.get(t[2]);
                    final CommandEvent cmd = new CommandEvent() {
                        @Override
                        void OnTx() {
                            rx = false;
                            if (t[3].equals("SPPTX")) {
                                SppTx(txMsg);
                            }
                            new Handler().postDelayed(new Runnable() {
                                @Override
                                public void run() {
                                    OnDelay();
                                }
                            }, Integer.parseInt(t[6]));
                        }

                        @Override
```

```
                        void OnDelay() {
                            if (t[5].equals("DEL")) {
                                StartNext(cmds);
                            }
                            else if (!rx && t[8].equals("REP")) {
                                OnTx();
                            }
                            else if (!rx && t[8].equals("FAIL")) {
                                CDS.get(t[9]).SetItem("RxFail - " + txMsg);
                            }
                            else {
                                StartNext(cmds);
                            }
                        }
                    };
                    if (t[5].equals("RX")) {
                        CDS.get(t[7]).RegisterOnSetEvent(new OnSetEvent() {
                            @Override
                            public void OnSet(Object item, Object prev) {
                                if (item.toString().equals(rxMsg)) {
                                    cmd.OnRx();
                                }
                            }
                        });
                    }
                    cmds.add(cmd);
                }
            }
        }
    }

    /**
     * Builds an EditText view in the execution dashboard
     * @param hint The name of the EditText view. Also serves as the hint text for when no
text has been entered.
     * @param tag Pass the key for the relevant entry in the CDS
     */
    private void drawEditText(String hint, String tag) {
        EditText view = new EditText(this);
        view.setHint(hint);
        view.setTag(tag);
        ArrayList<OnEditTextEvent> events = new ArrayList<>();
        view.addTextChangedListener(createEditTextEvent(events));
        EditTextListeners.put(tag, events);
        execList.addView(view);
    }

    private void drawButton(String text, String tag) {
        Button button = new Button(this);
        button.setText(text);
        button.setTag(tag);
        button.setFocusable(true);
        ArrayList<OnButtonPressEvent> events = new ArrayList<>();
        button.setOnClickListener(createButtonEvent(events));
        ButtonPressListeners.put(tag, events);
        execList.addView(button);
    }

    private void drawTextView(String text, String tag) {
        TextView view = new TextView(this);
        view.setText(text);
        view.setTag(tag);
        execList.addView(view);
    }

    private void drawGraph(String tag, int minX, int maxX, int minY, int maxY) {
```

```
        GraphView graph = findViewById(R.id.graph);
        graph.getViewport().setXAxisBoundsManual(true);
        graph.getViewport().setMinX(minX);
        graph.getViewport().setMaxX(maxX);
        graph.getViewport().setYAxisBoundsManual(true);
        graph.getViewport().setMinY(minY);
        graph.getViewport().setMaxY(maxY);
        GraphSeries.put(tag, new LineGraphSeries<>(new DataPoint[] {}));
        graph.addSeries(GraphSeries.get(tag));
        graph.setTag(tag);
        graphPresent = true;
    }

    private TextWatcher createEditTextEvent(final ArrayList<OnEditTextEvent> events) {
        return new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {
            }

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {
            }

            @Override
            public void afterTextChanged(Editable s) {
                for (OnEditTextEvent e : events) {
                    e.OnEditText(s.toString());
                }
            }
        };
    }

    private View.OnClickListener createButtonEvent(final ArrayList<OnButtonPressEvent> events)
{
        return new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                for (OnButtonPressEvent e : events) {
                    e.OnButtonPress();
                }
            }
        };
    }

    private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            switch (intent.getAction()) {
                case UsbService.ACTION_USB_PERMISSION_GRANTED: // USB PERMISSION GRANTED
                    Toast.makeText(context, "USB Ready", Toast.LENGTH_SHORT).show();
                    break;
                case UsbService.ACTION_USB_PERMISSION_NOT_GRANTED: // USB PERMISSION NOT
GRANTED
                    Toast.makeText(context, "USB Permission not granted",
Toast.LENGTH_SHORT).show();
                    break;
                case UsbService.ACTION_NO_USB: // NO USB CONNECTED
                    Toast.makeText(context, "No USB connected", Toast.LENGTH_SHORT).show();
                    break;
                case UsbService.ACTION_USB_DISCONNECTED: // USB DISCONNECTED
                    Toast.makeText(context, "USB disconnected", Toast.LENGTH_SHORT).show();
                    break;
                case UsbService.ACTION_USB_NOT_SUPPORTED: // USB NOT SUPPORTED
                    Toast.makeText(context, "USB device not supported",
Toast.LENGTH_SHORT).show();
                    break;
```

```
            }
        }
    };

    private final ServiceConnection usbConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName arg0, IBinder arg1) {
            usbService = ((UsbService.UsbBinder) arg1).getService();
            usbService.setHandler(sHandler);
        }

        @Override
        public void onServiceDisconnected(ComponentName arg0) {
            usbService = null;
        }
    };

    Handler btHandler = new Handler(){
        public void handleMessage(android.os.Message msg){
            if(msg.what == BluetoothManager.MESSAGE_READ){
                if (sppByteMode) {
                    for (OnMessageReceiveEvent e : MessageRxListeners.get("spp")) {
                        e.OnMessageReceive(DataUtilities.bytesToHex((byte[])msg.obj));
                    }
                }
                else {
                    String rx = new String((byte[]) msg.obj);
                    if (!rx.contains("\n")) {
                        sppLastMsg += rx;
                    } else {
                        rx = sppLastMsg + rx;
                        rx = rx.replace("\n", "").replace("\r", "");
                        for (OnMessageReceiveEvent e : MessageRxListeners.get("spp")) {
                            e.OnMessageReceive(rx);
                        }
                        sppLastMsg = "";
                    }
                }
            }

            if(msg.what == BluetoothManager.CONNECTING_STATUS){
                if(msg.arg1 == 1) {
                    Toast.makeText(getApplicationContext(), "Connected to Device: " + (String)
(msg.obj), Toast.LENGTH_SHORT).show();
                }
                else
                    Toast.makeText(getApplicationContext(), "Connection Failed",
Toast.LENGTH_SHORT).show();
            }
        }
    };

    private void OpenUSB() {
        try {
            if (sppConnect)
                CloseSPP();
            setFilters();
            startService(UsbService.class, usbConnection, null);
            usbConnect = true;
        }
        catch (Exception e) {
            Toast.makeText(getApplicationContext(), e.getMessage(),
Toast.LENGTH_SHORT).show();
        }
    }
```

```
    private void OpenSPP(final String macAddress) {
        if (sppConnect) {
            Toast.makeText(getApplicationContext(), "SPP already connected",
Toast.LENGTH_SHORT).show();
        }
        else if (!macAddress.equals("")) {
            if (usbConnect)
                CloseUSB();
            mBluetooth.Connect(macAddress);
            sppConnect = true;

            if (sppByteMode) {
                new Handler().postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        if (!mBluetooth.btConnected) {
                            mBluetooth.Connect(macAddress);
                        }
                    }
                }, 2500);
            }
        }
    }

    private void CloseUSB() {
        unregisterReceiver(mUsbReceiver);
        unbindService(usbConnection);
        usbConnect = false;
    }

    private void CloseSPP() {
        mBluetooth.Close();
        sppConnect = false;
    }

    private void SppTx(String msg) {
        if (sppConnect) {
            mBluetooth.Write(msg);
        }
    }

    private void SppTx(byte[] msg) {
        if (sppConnect) {
            mBluetooth.Write(msg);
        }
    }

    private void UsbTx(String msg) {
        if (usbConnect) {
            Toast.makeText(getApplicationContext(), msg, Toast.LENGTH_SHORT).show();
            usbService.write(msg.getBytes());
        }
    }

    private void LogData(String msg) {
        String date = new SimpleDateFormat("HH:mm:ss", Locale.getDefault()).format(new
Date());

        try {
            File log = new
File(getExternalFilesDir(Environment.DIRECTORY_DOWNLOADS).getAbsolutePath() + "/" + logName);
            if (!log.exists()) {
                log.createNewFile();
            }
            String output = date + ": " + msg + "\n";
            FileOutputStream os = new FileOutputStream(log, true);
```

```
            os.write(output.getBytes());
            os.flush();
            os.close();
        } catch (IOException e) {
            Toast.makeText(getApplicationContext(), "Error writing to Log File",
Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onResume() {
        if (usbConnect)
            OpenUSB();
        super.onResume();
    }

    @Override
    public void onPause() {
        if (usbConnect)
            CloseUSB();
        super.onPause();
    }

    private void startService(Class<?> service, ServiceConnection serviceConnection, Bundle
extras) {
        if (!UsbService.SERVICE_CONNECTED) {
            Intent startService = new Intent(this, service);
            if (extras != null && !extras.isEmpty()) {
                Set<String> keys = extras.keySet();
                for (String key : keys) {
                    String extra = extras.getString(key);
                    startService.putExtra(key, extra);
                }
            }
            startService(startService);
        }
        Intent bindingIntent = new Intent(this, service);
        bindService(bindingIntent, serviceConnection, Context.BIND_AUTO_CREATE);
    }

    private void setFilters() {
        IntentFilter filter = new IntentFilter();
        filter.addAction(UsbService.ACTION_USB_PERMISSION_GRANTED);
        filter.addAction(UsbService.ACTION_NO_USB);
        filter.addAction(UsbService.ACTION_USB_DISCONNECTED);
        filter.addAction(UsbService.ACTION_USB_NOT_SUPPORTED);
        filter.addAction(UsbService.ACTION_USB_PERMISSION_NOT_GRANTED);
        registerReceiver(mUsbReceiver, filter);
    }

    private static class SerialHandler extends Handler {
        private final WeakReference<ScriptExecutionActivity> mActivity;

        public SerialHandler(ScriptExecutionActivity activity) {
            mActivity = new WeakReference<>(activity);
        }

        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case UsbService.MESSAGE_FROM_SERIAL_PORT:
                    String rx = msg.obj.toString();
                    String[] t = rx.split("-");
                    String src = "tool_" + Build.GetNodeName(t[0],
mActivity.get().builds).toUpperCase();
```

```
                    for (OnMessageReceiveEvent e :
mActivity.get().MessageRxListeners.get(src)) {
                        e.OnMessageReceive(t[1].trim());
                    }
                    break;
            }
        }
    }

    private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();
            String name;
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            if (device != null) name = device.getName();
            else name = "None";

            if (BluetoothDevice.ACTION_ACL_CONNECTED.equals(action)) {
                Log.i ("SCRIPT_EXEC", name + " (re)connected in ACL." +
                        "\n\tMAC: " + device.getAddress() + "\n");
            }
            else if (BluetoothDevice.ACTION_ACL_DISCONNECTED.equals(action)) {
                Log.i ("SCRIPT_EXEC", name + " ACL disconnected\n");
            }
            else if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                Log.i ("SCRIPT_EXEC",  name + " found\n");
            }
            else if (BluetoothAdapter.ACTION_DISCOVERY_STARTED.equals(action)) {
                Log.i ("SCRIPT_EXEC", "Discovery started.\n");
            }
            else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) {
                Log.i ("SCRIPT_EXEC","Discovery stopped.\n");
            }
            else if (BluetoothDevice.ACTION_UUID.equals(action)) {
                Log.i ("SCRIPT_EXEC", "Got UUIDs for headset. Now ready to try
connection.\n");
            }
        }
    };

    private void openTcp() {
        new TcpConnectTask().execute("");
    }

    private void writeTcp(String msg) {
        if (tcpClient != null) {
            tcpClient.sendMessage(msg);
        }
    }

    private void closeTcp() {
        if (tcpClient != null) {
            tcpClient.stopClient();
            tcpClient = null;
        }
    }

    public class TcpConnectTask extends AsyncTask<String, String, TcpClient> {

        @Override
        protected TcpClient doInBackground(String... message) {

            //we create a TCPClient object
            tcpClient = new TcpClient(new TcpClient.OnMessageReceived() {
                @Override
                //here the messageReceived method is implemented
```

```
                public void messageReceived(String message) {
                    //this method calls the onProgressUpdate
                    publishProgress(message);
                }
            });
            tcpClient.run();

            return null;
        }

        @Override
        protected void onProgressUpdate(String... values) {
            super.onProgressUpdate(values);
            String rx = values[0];
            String[] t = rx.split("-");
            String src = "wifi_" + Build.GetNodeName(t[0], builds).toUpperCase();
            for (OnMessageReceiveEvent e : MessageRxListeners.get(src)) {
                e.OnMessageReceive(t[1].trim());
            }
        }
    }
}
```

ScriptDashboardActivity.java

```
package com.example.sbatzer.cdp_app;

import android.content.Context;
import android.content.Intent;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.Environment;
import android.os.Parcelable;
import android.support.annotation.NonNull;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.Gravity;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
```

```
/*
Resources:
https://developer.android.com/guide/topics/ui/declaring-layout
https://developer.android.com/guide/topics/ui/ui-events
https://developer.android.com/guide/topics/ui/dialogs
*/

/**
 * Small demonstration of a dashboard for the Communications Debugging Platform.
 * Author: Husam Beitello
 * Date: 2/19/19
 */
public class ScriptDashboardActivity extends AppCompatActivity {
    Button refreshDb;

    private DatabaseReference db;

    ArrayList<Script> scripts = new ArrayList<Script>(); //Array of scripts
    final ArrayList<Build> builds = new ArrayList<Build>();

    LinearLayout scriptList; //Build the list of TextViews and Buttons
    TextView selected;

    final String TAG = "CDP_Dashboard";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_script_dashboard);

        //Just for displaying things
        selected = findViewById(R.id.selected);

        //Get LinearLayout object for adding things
        scriptList = findViewById(R.id.scriptList);
        scriptList.setOrientation(LinearLayout.VERTICAL);

        refreshDb = findViewById(R.id.refreshDb);
        refreshDb.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (isWifiAvailable()) {
                    refreshLocalDb();
                }
                else {
                    Toast.makeText(getApplicationContext(), "Internet connection not present -
connect and try again", Toast.LENGTH_SHORT).show();
                }
            }
        });

        initDb();
    }

    private void initDb() {
        db = FirebaseDatabase.getInstance().getReference();
//        build b = new build("0", "App", "0.0.1");
//        db.child("builds").child("cdp_11app").updateChildren(b.outMap());
        if (!getLocalDb()) {
            if (isWifiAvailable()) {
                Toast.makeText(getApplicationContext(), "Local DB not found - refreshing from
remote DB", Toast.LENGTH_SHORT).show();
                refreshLocalDb();
            }
            else {
```

```
                Toast.makeText(getApplicationContext(), "Internet connection not present and
local DB not found", Toast.LENGTH_SHORT).show();
            }
        }
        buildScriptList();
    }

    private void refreshLocalDb() {
        DatabaseReference buildRef = db.child("builds").getRef();
        buildRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                builds.clear();
                for (DataSnapshot child : dataSnapshot.getChildren()) {
                    String k = child.getKey();
                    String n = child.child("nodeid").getValue().toString();
                    String r = child.child("role").getValue().toString();
                    String v = child.child("version").getValue().toString();
                    builds.add(new Build(k, n, r, v));
                }
                registerScriptListener();
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {

            }
        });
    }

    private void registerScriptListener() {
        DatabaseReference scriptRef = db.child("scripts").getRef();
        scriptRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                scriptList.removeAllViews();

                scripts.clear();
                for (DataSnapshot child : dataSnapshot.getChildren()) {
                    String name = child.child("name").getValue().toString();
                    String description = child.child("description").getValue().toString();
                    String reqnodes = child.child("reqnodes").getValue().toString();
                    String contents = child.child("contents").getValue().toString();
                    scripts.add(new Script(name, description, reqnodes, contents));
                }
                createLocalDb();
                buildScriptList();
            }

            @Override
            public void onCancelled(DatabaseError databaseError) {
            }
        });
    }

    public boolean isWifiAvailable() {
        if (networkConnectivity()) {
            try {
                Process p1 = Runtime.getRuntime().exec(
                        "ping -c 1 www.google.com");
                int returnVal = p1.waitFor();
                boolean reachable = (returnVal == 0);
                if (reachable) {
                    System.out.println("Internet access");
                    return reachable;
                } else {
```

```
                    return false;
                }
            } catch (Exception e) {
                return false;
            }
        } else
            return false;

    }

    private boolean networkConnectivity() {
        ConnectivityManager cm = (ConnectivityManager) getApplicationContext()
                .getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo networkInfo = cm.getActiveNetworkInfo();
        return networkInfo != null && networkInfo.isConnected();
    }

    private void buildScriptList() {
        for (Script s : scripts) {
            toLinear(s);
        }
    }

    private void createLocalDb() {
        try {
            FileOutputStream fos = getApplicationContext().openFileOutput("scripts.ser",
Context.MODE_PRIVATE);
            ObjectOutputStream os = new ObjectOutputStream(fos);
            os.writeObject(scripts);
            os.close();
            fos.close();

            fos = getApplicationContext().openFileOutput("builds.ser", Context.MODE_PRIVATE);
            os = new ObjectOutputStream(fos);
            os.writeObject(builds);
            os.close();
            fos.close();

            Toast.makeText(getApplicationContext(), "Local database synced successfully!",
Toast.LENGTH_SHORT).show();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private boolean getLocalDb() {
        boolean success = false;
        try {
            FileInputStream fis = getApplicationContext().openFileInput("scripts.ser");
            ObjectInputStream is = new ObjectInputStream(fis);
            scripts = (ArrayList<Script>)is.readObject();
            is.close();
            fis.close();

            fis = getApplicationContext().openFileInput("builds.ser");
            is = new ObjectInputStream(fis);
            builds.clear();
            for (Build b: (ArrayList<Build>)is.readObject()) {
                builds.add(b);
            }
            is.close();
            fis.close();
            success = true;
        } catch (FileNotFoundException e) {
```

```
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return success;
    }

    public void toLinear(final Script scr) {
        final TextView text = new TextView(this);
        text.setText(String.format("Name: %s", scr.name));
        text.append(String.format("\nReq. Nodes: %s", Build.GetNodeNames(scr.reqNodes,
builds)));
        text.append(String.format("\nDesc: %s", scr.desc));

        text.setTextIsSelectable(true); //Convenient, make it so you can highlight and copy
text.
        text.setPadding(0,16,0,0);

        final LinearLayout buttons = new LinearLayout(this); //Make a horizontal layout for
the two buttons
        buttons.setOrientation(LinearLayout.HORIZONTAL); //Make it horizontal
        buttons.setHorizontalGravity(Gravity.CENTER);
        buttons.setPadding(0,0,0,16);

        Button execButton = new Button(this);
        execButton.setText(String.format("Execute %s", scr.name));
        execButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getApplicationContext(),
ScriptExecutionActivity.class);
                intent.putExtra("script", (Parcelable)scr);
                intent.putParcelableArrayListExtra("builds", builds);
                startActivity(intent);
            }
        });
        buttons.addView(execButton);

        Button deleteButton = new Button(this);
        deleteButton.setText(String.format("Unload %s", scr.name));
        deleteButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                scriptList.removeView(text);
                scriptList.removeView(buttons);
                scripts.remove(scr);
            }
        });
        buttons.addView(deleteButton);

        scriptList.addView(text);
        scriptList.addView(buttons);
    }
}
```

CDP Serial Module Node

cdp_node_serial.ino

```
#include "src/node.h"
#include <String.h>
#include <Wire.h>

bool connected = true;
bool macReq = false;
//00:07:80:CC:7C:63

void setup() {
 pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(9600);
 Serial.setTimeout(10);
  Wire.begin(NodeID);
 Wire.onReceive(receiveEvent);
}

void loop() {
 if (!connected) {
   digitalWrite(LED_BUILTIN, LOW);
   Wire.beginTransmission(ToolID);
   Wire.write(NodeID);
   Wire.write(NodeVersion);
   Wire.endTransmission();
   delay(500);
 }
 else {
   digitalWrite(LED_BUILTIN, HIGH);
 }

 if (Serial.available()) {
   String inDat = Serial.readString();
   Wire.beginTransmission(ToolID);
   Wire.write(NodeID);
   Wire.write(inDat.c_str());
   Wire.endTransmission();
   Serial.println(inDat);
 }

 if (macReq) {
   String macAddr = "00:07:80:CC:7C:63";
   Wire.beginTransmission(ToolID);
   Wire.write(NodeID);
   Wire.write(macAddr.c_str());
   Wire.endTransmission();
   macReq = false;
 }
}

void receiveEvent(int n) {
 uint8_t id = Wire.read();
 if (connected) {
   String ver = "";
   while (Wire.available()) {
     ver += (char)Wire.read();
   }
   Serial.print(String(id) + '-' + ver);
   if (ver == "0xAA") {
     macReq = true;
   }
 }
 else {
   connected = (id == 0xFF) ? true : false;
```

```
 }
}
```

Node.h

```
#ifndef node_h
#define node_h

const uint8_t ToolID = 0x01;
const uint8_t NodeID = 0x09;
const char* NodeVersion = "v0.0.1";

String getValue(String data, char separator, int index)
{
   int found = 0;
   int strIndex[] = { 0, -1 };
   int maxIndex = data.length() - 1;

   for (int i = 0; i <= maxIndex && found <= index; i++) {
       if (data.charAt(i) == separator || i == maxIndex) {
           found++;
           strIndex[0] = strIndex[1] + 1;
           strIndex[1] = (i == maxIndex) ? i+1 : i;
       }
   }
   return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

#endif
```

CDP Morse Module Node

cdp_node_morse.ino

```
#include "src/node.h"
#include "src/morse.h"
#include <String.h>
#include <Wire.h>

#define MSG_COUNT 10

String msgQueue[MSG_COUNT];
uint8_t msgCount = 0;

bool connected = true;

void setup() {
 pinMode(LED_BUILTIN, OUTPUT);
 pinMode(2, OUTPUT);
  Wire.begin(NodeID);
 Wire.onReceive(receiveEvent);
}

void loop() {
 if (!connected) {
   digitalWrite(LED_BUILTIN, LOW);
   Wire.beginTransmission(ToolID);
   Wire.write(NodeID);
   Wire.write(NodeVersion);
   Wire.endTransmission();
   delay(500);
 }
 else {
   digitalWrite(LED_BUILTIN, HIGH);
   if (msgCount > 0) {
     msgCount--;
     String msg = msgQueue[msgCount];
     for (int i = 0; i < sizeof(msg) - 1; i++) {
       morse(msg[i]);
     }
   }
   pause(500);
 }
}

void receiveEvent(int n) {
 uint8_t id = Wire.read();
 if (connected) {
   String msg;
   while (Wire.available()) {
     msg += (char)Wire.read();
   }
   msgQueue[msgCount] = msg;
   msgCount++;
 }
 else {
   connected = (id == 0xFF) ? true : false;
 }
}
```

node.h

```
#ifndef node_h
#define node_h

const uint8_t ToolID = 0x01;
```

```
const uint8_t NodeID = 0x04;
const char* NodeVersion = "v0.0.1";

String getValue(String data, char separator, int index)
{
   int found = 0;
   int strIndex[] = { 0, -1 };
   int maxIndex = data.length() - 1;

   for (int i = 0; i <= maxIndex && found <= index; i++) {
       if (data.charAt(i) == separator || i == maxIndex) {
           found++;
           strIndex[0] = strIndex[1] + 1;
           strIndex[1] = (i == maxIndex) ? i+1 : i;
       }
   }
   return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

#endif
```

morse.h

```
#ifndef morse_h
#define morse_h

int spkPin = 2;             // output audio on pin 8
int note = 1200;            // music note/pitch
int dotLen = 50;            // length of the morse code 'dot'
int dashLen = dotLen * 3;   // length of the morse code 'dash'
int elemPause = dotLen;     // length of the pause between elements of a character
int Spaces = dotLen * 3;    // length of the spaces between characters
int wordPause = dotLen * 7; // length of the pause between words

char *lang = "**ETIANMSURWDKGOHVF*L*LPJBXCYZQ**";

void pause(int len) {
 noTone(spkPin);           // stop playing a tone
 delay(len);               // hold in this position
}

void dash() {
 tone(spkPin, note, dashLen); // start playing a tone
 delay(dashLen);              // hold in this position
 pause(elemPause);
}

void dot() {
 tone(spkPin, note, dotLen); // start playing a tone
 delay(dotLen);              // hold in this position
 pause(elemPause);
}

void conv(uint8_t dec) {
 if (dec) {
   conv(dec/2);
   if (dec != 1) dec % 2 ? dash() : dot();
 }
}

void morse(char c) {
 if (c >= 'a' && c <= 'z') c -= 32;
 if (c < 'A' || c > 'Z') return;
 uint8_t i = 0;
 while (lang[++i] != c);
 conv(i);
```

```
}

#endif
```

CDP Tool Node

cdp_node_tool.ino

```
#include "src/node.h"
#include <String.h>
#include <Wire.h>

#define NODE_COUNT 10

String moduleList[NODE_COUNT];
int moduleCount = 0;

int ackQueue[NODE_COUNT];
int ackCount = 0;

uint8_t trgtQueue[10];
String msgQueue[10];
uint8_t msgCount = 0;

bool tx_enable = true;
bool toggle = false;

void setup() {
 pinMode(LED_BUILTIN, OUTPUT);
 Serial.begin(115200);
 Serial.setTimeout(10);
  Wire.begin(NodeID);
 Wire.onReceive(receiveEvent);
}

void loop() {
 if (ackCount > 0) {
   ackCount--;
   Wire.beginTransmission(ackQueue[ackCount]);
   Wire.write(0xFF);
   Wire.endTransmission();
   // Serial.println("Ack-" + String(ackQueue[ackCount]));
 }

 if (msgCount > 0) {
   msgCount--;
   Wire.beginTransmission(trgtQueue[msgCount]);
   Wire.write(NodeID);
   Wire.write(msgQueue[msgCount].c_str());
   Wire.endTransmission();
 }

 if (Serial.available()) { // accepts input in the form "{trgt}-{data}"
   toggle = !toggle;
   digitalWrite(LED_BUILTIN, toggle);
   String input = Serial.readString();
   uint8_t trgtAddr = getValue(input, '-', 0).toInt();
   String trgtData = getValue(input, '-', 1);
   trgtQueue[msgCount] = trgtAddr;
   msgQueue[msgCount] = trgtData;
   msgCount++;
   tx_enable = true;
 }
 delay(10);
}

void receiveEvent(int n) {
 uint8_t id = Wire.read();
 if (moduleList[id] == "") {
   String ver = "";
```

```
    while (Wire.available()) {
      ver += (char)Wire.read();
    }
    ackQueue[ackCount] = id;
    ackCount++;
    moduleList[id] = ver;
    moduleCount++;
  }
  else {
    String dat = "";
    while (Wire.available()) {
      dat += (char)Wire.read();
    }
    if (tx_enable) {
      Serial.println(String(id) + '-' + dat);
    }
  }
}
```

CDP UART WIFI Tool Node

cdp_node_tool_wifi_uart.ino

```
#include "src/node.h"
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <String.h>

const char* ssid = "cdp_ap_ssid1";
const char* password = "cdp_ap_pass1";
WiFiServer server(80);
WiFiClient client;

void setup() {
 delay(1000);
 Serial.begin(115200);
 Serial.setTimeout(10);
 WiFi.softAP(ssid, password); //begin WiFi access point
 server.begin();
}

void loop() {
 if (client.connected()) {
   if (client.available()) {
     String wifiRX = client.readStringUntil('\n');
     wifiRX.trim();
     Serial.print(wifiRX);
   }
 }
 else {
   client = server.available();
 }
 if (Serial.available()) {
   String uartRX = Serial.readString();
   if (client.connected()) {
     client.print(uartRX);
   }
 }
}
```

node.h

```
#ifndef node_h
#define node_h

const uint8_t NodeID = 0x01;
const String NodeVersion = "v0.0.1";

String getValue(String data, char separator, int index)
{
   int found = 0;
   int strIndex[] = { 0, -1 };
   int maxIndex = data.length() - 1;

   for (int i = 0; i <= maxIndex && found <= index; i++) {
       if (data.charAt(i) == separator || i == maxIndex) {
           found++;
           strIndex[0] = strIndex[1] + 1;
           strIndex[1] = (i == maxIndex) ? i+1 : i;
       }
   }
   return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

#endif
```

CDP Serial Module Node

cdp_node_serial.ino

```
#include "src/node.h"
#include <String.h>
#include <Wire.h>

bool connected = true;
bool macReq = false;
//00:07:80:CC:7C:63

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);

  Serial.begin(9600);
  Serial.setTimeout(10);

  Wire.begin(NodeID);
  Wire.onReceive(receiveEvent);
}

void loop() {
  if (!connected) {
    digitalWrite(LED_BUILTIN, LOW);
    Wire.beginTransmission(ToolID);
    Wire.write(NodeID);
    Wire.write(NodeVersion);
    Wire.endTransmission();
    delay(500);
  }
  else {
    digitalWrite(LED_BUILTIN, HIGH);
  }

  if (Serial.available()) {
    String inDat = Serial.readString();
    Wire.beginTransmission(ToolID);
    Wire.write(NodeID);
    Wire.write(inDat.c_str());
    Wire.endTransmission();
    Serial.println(inDat);
  }

  if (macReq) {
    String macAddr = "00:07:80:CC:7C:63";
    Wire.beginTransmission(ToolID);
    Wire.write(NodeID);
    Wire.write(macAddr.c_str());
    Wire.endTransmission();
    macReq = false;
  }
}

void receiveEvent(int n) {
  uint8_t id = Wire.read();
  if (connected) {
    String ver = "";
    while (Wire.available()) {
      ver += (char)Wire.read();
    }
    Serial.print(String(id) + '-' + ver);
    if (ver == "0xAA") {
      macReq = true;
    }
  }
```

```
  else {
    connected = (id == 0xFF) ? true : false;
  }
}
```

CDP DAQ Module Node

cdp_node_daq.ino

```
#include "src/node.h"
#include <String.h>
#include <Wire.h>

#define MAX_ADC 10
#define MAX_DOUT 10

uint8_t adcList[MAX_ADC];
uint8_t adcCount = 0;

uint8_t doutList[MAX_DOUT];
uint8_t doutCount = 0;

int sampleDelay = 2000;

bool connected = true;

void setup() {
 pinMode(LED_BUILTIN, OUTPUT);
  Wire.begin(NodeID);
 Wire.onReceive(receiveEvent);
}

void loop() {
 if (!connected) {
   digitalWrite(LED_BUILTIN, LOW);
   Wire.beginTransmission(ToolID);
   Wire.write(NodeID);
   Wire.write(NodeVersion);
   Wire.endTransmission();
   delay(500);
 }
 else {
   digitalWrite(LED_BUILTIN, HIGH);
   if (doutCount > 0 || adcCount > 0) {
     getSamples();
     delay(sampleDelay);
   }
   else {
     delay(500);
   }
 }
}

void getSamples() {
 String outWrite = "";
 uint8_t i;
 uint16_t val;
 for (i = 0; i < adcCount; i++) {
   val = analogRead(adcList[i]);
   outWrite += String(val) + ",";
 }
 for (i = 0; i < doutCount; i++) {
   val = digitalRead(doutList[i]);
   outWrite += String(val) + ",";
 }

 Wire.beginTransmission(ToolID);
 Wire.write(NodeID);
 Wire.write(outWrite.c_str());
 Wire.endTransmission();
}
```

```
void receiveEvent(int n) {
 uint8_t id = Wire.read();
 if (connected) {
   String input = "";
   while (Wire.available()) {
     input += (char)Wire.read();
   }

   uint8_t cmd = getValue(input, ',', 0).toInt();
   uint16_t val = getValue(input, ',', 1).toInt();

   if (cmd == 0) { // Set digital output value
     uint8_t out = getValue(input, ',', 2).toInt();
     digitalWrite(val, out);
   }
   if (cmd == 1) { // Configure digital input/output
     uint8_t dir = getValue(input, ',', 2).toInt();
     pinMode(val, dir);
     if (dir) { // Digital input
       doutList[doutCount] = val;
       doutCount++;
     }
   }
   if (cmd == 2) {
     pinMode(val, INPUT);
     adcList[adcCount] = val;
     adcCount++;

   }
   if (cmd == 3) {
     sampleDelay = val;
   }
 }
 else {
   connected = (id == 0xFF) ? true : false;
 }
}
```

CDP CAN Module Node

cdp_node_can.ino

```
#include <SPI.h>
#include <mcp_can.h>
#include "src/node.h"
#include <String.h>
#include <Wire.h>

#define MSG_COUNT 10
#define MSG_SIZE 8

byte msgQueue[MSG_COUNT][MSG_SIZE];
int msgId[MSG_COUNT];
uint8_t msgCount = 0;
uint8_t val[8];

byte len = 0;
byte buf[MSG_SIZE];

const int spiCSPin = 10;

bool connected = true;

MCP_CAN CAN(spiCSPin);

void setup()
{
 Serial.begin(9600);
 while (CAN_OK != CAN.begin(CAN_125KBPS)) {
   delay(100);
 }
 Wire.begin(NodeID);
 Wire.onReceive(receiveEvent);
}

void loop()
{
 if (!connected) {
   Wire.beginTransmission(ToolID);
   Wire.write(NodeID);
   Wire.write(NodeVersion);
   Wire.endTransmission();
   delay(500);
 }
 else {
   if (msgCount > 0) {
     msgCount--;
     CAN.sendMsgBuf(msgId[msgCount], 0, 8, msgQueue[msgCount]);
   }
   if(CAN_MSGAVAIL == CAN.checkReceive())
   {
     char str[16];
     CAN.readMsgBuf(&len, buf);
     int canId = CAN.getCanId();
     byteArrayToString(buf, 8, str);
     String rx = String(canId) + "," + String(str);
     rx.trim();
     Wire.beginTransmission(ToolID);
     Wire.write(NodeID);
     Wire.write(rx.c_str());
     Wire.endTransmission();
   }
 }
}
```

```
void receiveEvent(int n) {
 uint8_t id = Wire.read();
 if (connected) {
   String msg;
   while (Wire.available()) {
     msg += (char)Wire.read();
   }
   int cmd = getValue(msg, ',', 0).toInt();
   int pid = getValue(msg, ',', 1).toInt();
   Serial.println(msg);

   if (cmd == 0) {
     String newMsg = getValue(msg, ',', 2);
     uint8_t* x = datahex(newMsg.c_str());
     for (int i = 0; i < 8; i++) {
       msgQueue[msgCount][i] = x[i];
     }
     msgId[msgCount] = pid;
     msgCount++;
   }
   else if (cmd == 1) {
     // NOT USED YET
   }
 }
 else {
   connected = (id == 0xFF) ? true : false;
 }
}
```

node.h

```
#ifndef node_h
#define node_h

const uint8_t ToolID = 0x01;
const uint8_t NodeID = 0x03;
const char* NodeVersion = "v0.0.1";

String getValue(String data, char separator, int index)
{
   int found = 0;
   int strIndex[] = { 0, -1 };
   int maxIndex = data.length() - 1;

   for (int i = 0; i <= maxIndex && found <= index; i++) {
       if (data.charAt(i) == separator || i == maxIndex) {
           found++;
           strIndex[0] = strIndex[1] + 1;
           strIndex[1] = (i == maxIndex) ? i+1 : i;
       }
   }
   return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}

void byteArrayToString(uint8_t arr[], uint8_t len, char buff[]) {
   for (uint8_t i = 0; i < len; i++)
   {
       uint8_t nib1 = (arr[i] >> 4) & 0x0F;
       uint8_t nib2 = (arr[i] >> 0) & 0x0F;
       buff[i*2+0] = nib1  < 0xA ? '0' + nib1  : 'A' + nib1  - 0xA;
       buff[i*2+1] = nib2  < 0xA ? '0' + nib2  : 'A' + nib2  - 0xA;
   }
   buff[len*2] = '\0';
}

uint8_t* datahex(char* string) {
```

```
   if(string == NULL)
      return NULL;

   size_t slength = strlen(string);
   if((slength % 2) != 0) // must be even
      return NULL;

   size_t dlength = slength / 2;

   uint8_t* data = malloc(dlength);
   memset(data, 0, dlength);

   size_t index = 0;
   while (index < slength) {
       char c = string[index];
       int value = 0;
       if(c >= '0' && c <= '9')
         value = (c - '0');
       else if (c >= 'A' && c <= 'F')
         value = (10 + (c - 'A'));
       else if (c >= 'a' && c <= 'f')
         value = (10 + (c - 'a'));
       else {
         free(data);
         return NULL;
       }

       data[(index/2)] += value << (((index + 1) % 2) * 4);

       index++;
   }

   return data;
}

#endif
```

APPENDIX D: ENCLOSURE PROTOTYPE