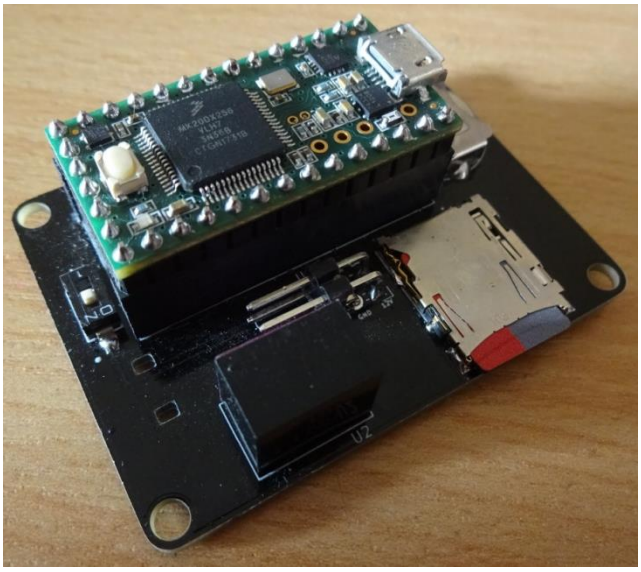
CBR015-0002

CAN DATALOGGER ASSY NOTES

Issue 01



Thomas Naughton

Issue No.	Issue Date	Description of Changes
01	09-12-2018	Original Document
02	04-03-2020	Document update to suit v2.0 software & v3.0 conversion script

Table of Contents

1. Summary 2
2. Hardware 3
2.1. Hardware Overview 3
2.1.1. Data 3
2.1.2. Power 3
2.2. Component Details 4
2.2.1. PJRC Teensy 3.2 Development Board 5
2.2.2. Microchip MCP2551 CAN-Bus Transceiver 5
2.2.3. Traco Power TSR 1-2450 Voltage Regulator 6
2.2.4. Keystone 3001 Coin Cell Battery Holder 6
2.2.5. Hirose DM3AT-SF-PEJM5 MicroSD Card Socket 7
2.3. PCB 7
2.4. Enclosure 8
2.5. Vehicle Connection 9
3. Code 11
3.1. Libraries 14
4. Data File Processing 16
Appendix A - Hardware Schematic 18
Appendix B - PCB Layout 19
Appendix C - Bill of Materials 21
Appendix D - Code 22
Appendix E - CAN Config 22

1. Summary

The purpose of this document is to provide a detailed description of the hardware and code used in the CBR015-0002 CAN DATALOGGER ASSY for ease of future reference.

The CAN DATALOGGER ASSY is designed around a Teensy 3.2 development board and its purpose is to receive data via automotive CAN-Bus and write the data to a MicroSD card.

The module was designed for the CBR250RRi project to record engine management data broadcast by the Microsquirt V3 ECU and also add the possibility for recording additional sensor data which is not provided by the ECU. i.e. chassis sensors and/or GPS/IMU data.

Prior to implementing the CAN DATALOGGER ASSY, any additional sensor data had to be sent to the Microsquirt ECU via CAN-Bus (limited to 8off channels) and then data logged on an external device (laptop or smartphone) using a specific software interface via Serial RS232 or Serial Bluetooth connection. The number of engine data channels available to be logged by the software interface is limited and the logging frequency is limited to c.15Hz by the data throughput of the serial connection.

In comparison to the serial connection, the CAN dash broadcast feature of the Microsquirt ECU allows more data channels to be accessed and at higher frequencies of up to 50Hz. Having a data logging module integrated in the vehicle also ensures that all vehicle activity is logged to the onboard MicroSD card which provides more effective troubleshooting capability in the event of engine management issues.

The CAN DATALOGGER ASSY is designed as the core of a modular data acquisition system which will include a GPS & IMU data module and analogue sensor expansion module. All additional modules in the system will broadcast acquired data via CAN-Bus to be logged by the core CAN DATALOGGER ASSY alongside engine management data.

Although the CAN DATALOGGER ASSY and associated expansion modules have been designed for the CBR250RRi motorcycle project, the nature of the modules will allow them to be utilised on many different vehicles with or without CAN-Bus capable engine management units. In the case of use with any other vehicle, the CAN DATALOGGER ASSY software will need to be adjusted to suit the incoming data CAN IDs and data transformation.

2. Hardware

2.1. Hardware Overview

2.1.1. Data

The CAN DATALOGGER ASSY receives data from external sources via an automotive CAN-Bus network. A 120Ω bus termination resistor is included on the board and a SPST single SIP switch is also included to allow the termination resistor to be disabled depending on the position of the CAN DATALOGGER ASSY in the CAN-Bus network.

An MCP2551 CAN-Bus Transceiver provides the interface between the vehicle CAN-Bus and the microcontroller.

The microcontroller, in this case a Teensy 3.2, reads the CAN messages received from the bus and converts the message data into readable engine management or sensor data to be logged based on a predefined message configuration. A detailed description of each message that is processed and logged by the CAN DATALOGGER ASSY is included in Appendix E.

The processed data is written to a MicroSD card at a frequency defined by the user. Data is written in binary format to a *.dat file.

The block diagram in Fig. 1 provides a simple overview of the data flow in the CAN DATALOGGER ASSY. A full electrical schematic can be found in Appendix A and further information on the individual components is contained in Section 2.2.

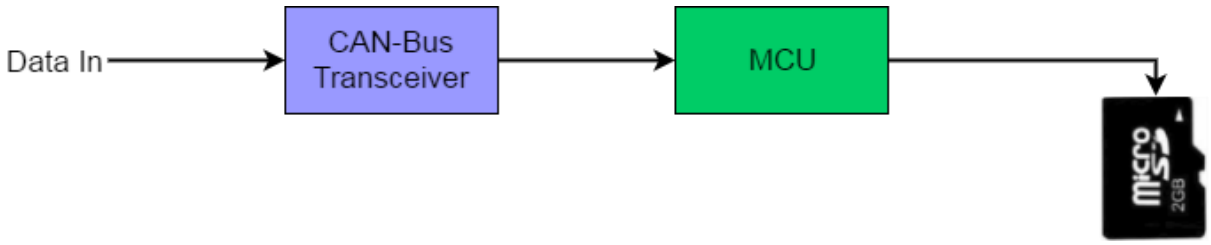


Figure 1 - CAN DATALOGGER ASSY Data Overview

2.1.2. Power

The CAN DATALOGGER ASSY is designed to be powered by general vehicle 12V DC power. Power to the unit should be connected such that the unit is powered on and off at the same time as the ECU as the unit has been configured to begin logging each time the board is powered on

The board can accept 6.5-36V DC power which is regulated down to 5V DC using a Traco Power TSR 1-2450 switching regulator.

The board is not reverse polarity protected.

5V power is distributed to both the CAN-Bus Transceiver and the microcontroller.

The microcontroller board incorporates a 3.3V regulator which provides power to the MicroSD card connector.

A 3V coin cell battery is also included on the board. This provides power for the microcontroller Real Time Clock (RTC) to allow timekeeping even when the CAN DATALOGGER ASSY is not powered by the vehicle.

The block diagram in Fig. 2 provides a simple overview of the power distribution in the CAN DATALOGGER ASSY. A full electrical schematic can be found in Appendix A and further information on the individual components is contained in Section 2.2.

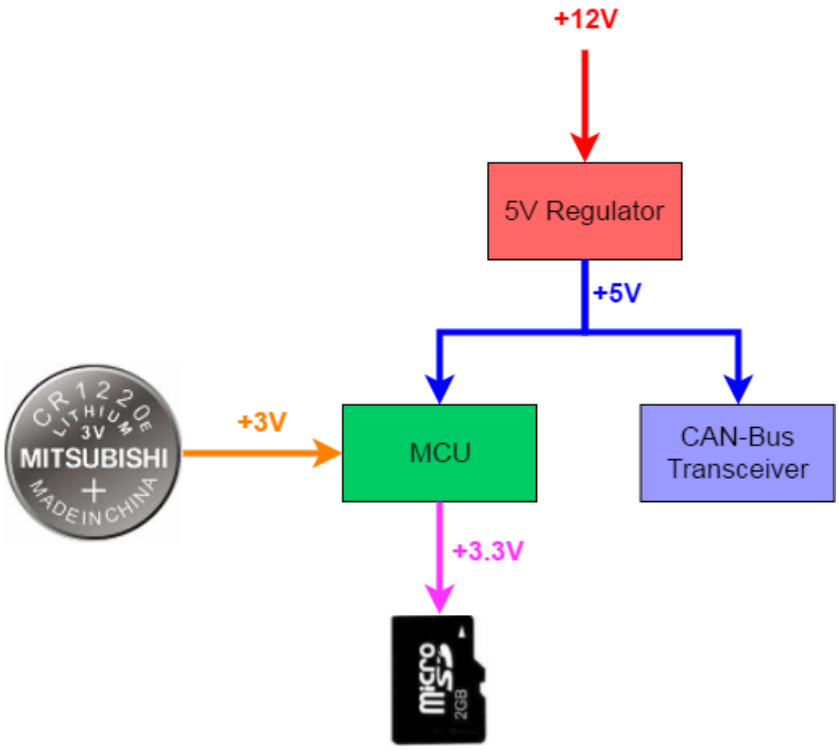


Figure 2 - CAN DATALOGGER ASSY Power Overview

2.2. Component Details

The major components used in the CAN DATALOGGER ASSY are as follows:

1. PJRC Teensy 3.2 Development Board
2. Microchip MCP2551 CAN-Bus Transceiver
3. Traco Power TSR 1-2450 Voltage Regulator
4. Keystone 3001 Coin Cell Battery Holder
5. Hirose DM3AT-SF-PEJM5 MicroSD Card Socket

2.2.1. PJRC Teensy 3.2 Development Board

The PJRC Teensy 3.2 development board was chosen as the MCU for the CAN DATALOGGER ASSY because of its local CAN support, small package size and ease of programming via the Arduino IDE.

The Teensy 3.2 uses a 32 bit ARM Cortex-M4 72 MHz CPU which is overclockable to 120 MHz. It is programmed via an on-board Micro USB port and the Arduino IDE. It is compatible with all existing Arduino software and code libraries.

The Teensy 3.2 has an inbuilt RTC which can be enabled by adding a 32.768 kHz, 12.5 pF crystal to the underside of the board. More details can be found at: https://www.pjrc.com/teensy/td_libs_Time.html

Figure 3 - Teensy 3.2 Board

Datasheet: https://media.digikey.com/pdf/Data%20Sheets/Sparkfun%20PDFs/DEV-13736_Web.pdf

2.2.2. Microchip MCP2551 CAN-Bus Transceiver

A Microchip MCP2551 CAN transceiver is used to interface with the vehicle bus. An SMD SOIC-8 version of the MCP2551 is used for packaging purposes.

Fig. 4 below shows the connections required for the MCP2551 where TX_CAN & RX_CAN are connected to pins D3 & D4 respectively on the Teensy 3.2 and CANH & CANL are the vehicle bus connections.

A 120Ω bus termination resistor is included as well as a SIP switch to toggle the termination resistor as required. With the switch in ON position, the termination resistor is enabled.

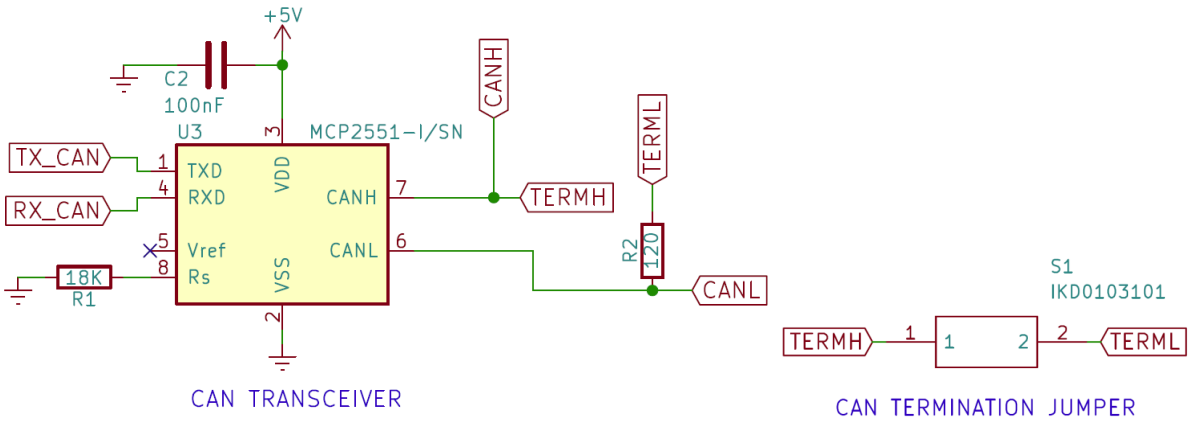


Figure 4 - MCP2551 Ancillary Components

Datasheet: http://ww1.microchip.com/downloads/en/DeviceDoc/21667E.pdf

2.2.3. Traco Power TSR 1-2450 Voltage Regulator

Board voltage regulation is carried out using a Traco Power TSR 1-2450 5V, 1A switching regulator. The regulator accepts a large input voltage range of 6.5-36V DC. The regulator has an efficiency of c.90% and is suited to small enclosures as only a single capacitor is required between Vin & GND. Although the physical size is not particularly compact, it can be tolerated due to the height of the Teensy 3.2 board when placed on 2.54mm header pin sockets.

Datasheet: https://www.tracopower.com/products/tsr1.pdf

2.2.4. Keystone 3001 Coin Cell Battery Holder

A 3V coin cell battery is required to keep the Teensy RTC active while not being powered by the vehicle battery. A through-hole 3001 holder from Keystone is used to retain the battery.

The holder accepts BR1216, CR1216, BR1220, CL1220, CR1220 or BR1225 battery cells.

The 3001 holder uses only spring clips to retain the battery which may not be sufficient when subjected to vibrations from use on a vehicle. To provide additional retention, the holder is placed at the edge of the board such that the wall of the enclosure acts as a secondary battery retention.

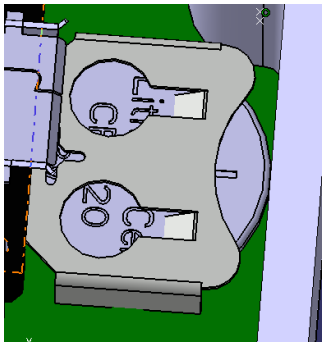


Figure 5 - Coin Cell Battery Retention

Series Catalogue: http://www.keyelco.com/userAssets/file/M65p7-9.pdf

Drawing & Model: http://www.keyelco.com/product.cfm/product_id/778

2.2.5. Hirose DM3AT-SF-PEJM5 MicroSD Card Socket

A Hirose DM3AT-SF-PEJM5 connector is used to house the MicroSD card. This connector is a push-push type to retain the MicroSD card under the vibrations associated with on-vehicle use.

Product Website: https://www.hirose.com/product/en/products/DM3/DM3AT-SF-PEJM5/

2.3. PCB

The CBR015-0003 CAN DATALOGGER PCB is designed using KiCAD software and manufactured by Seedstudio Fusion.

The size of the PCB is kept to a minimum while fitting all components on both layers of the board.

The PCB includes 4off through holes in each corner to enable mounting in the enclosure.

2off slots are also included to allow the vehicle connection wire bundle to be cable tied to the PCB for strain relief.

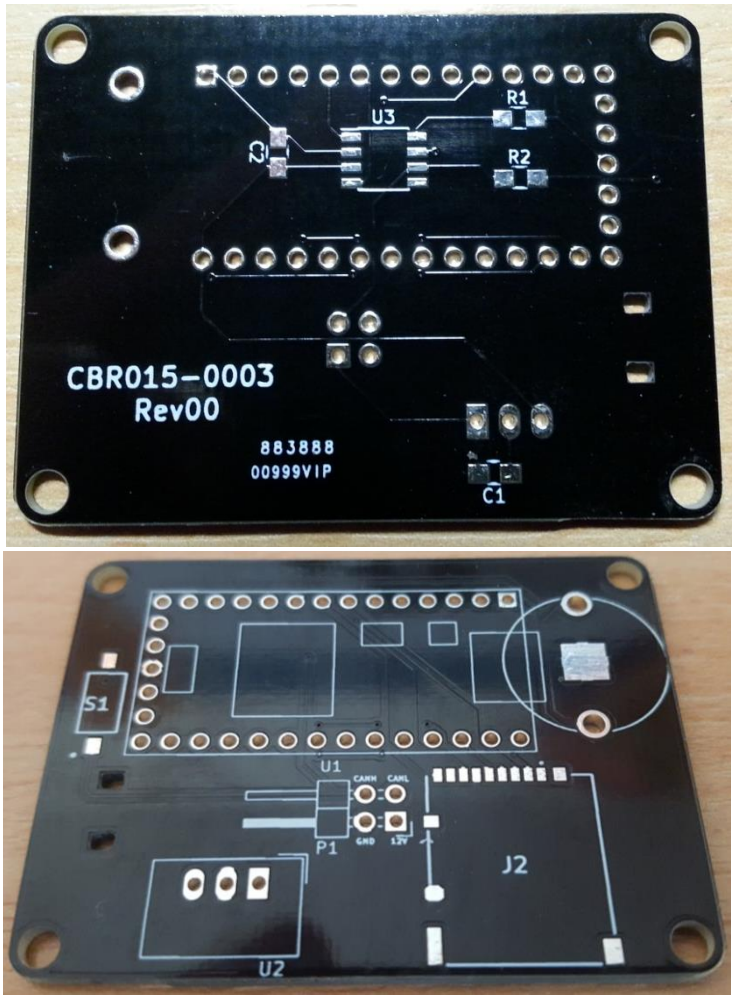


Figure 6 - CBR015-0003 CAN DATALOGGER PCB

2.4. Enclosure

The enclosure for the CAN DATALOGGER ASSY is a custom 3D printable design which encloses all components and supports the PCB. While a sealed enclosure is preferable, a slot is included in the enclosure to allow access to the MicroSD card without disassembly of the enclosure.

As the enclosure is not fully sealed, it is required to be placed in an area where the likelihood of water ingress is minimal.

A wire seal is also 3D printed from flexible TPU filament in order to provide a small level of sealing around the wire bundle cut-out and provide additional strain relief for the wire bundle.

The board is clamped between the two parts of the enclosure using 4off 2.9x13mm self-tapping screws.

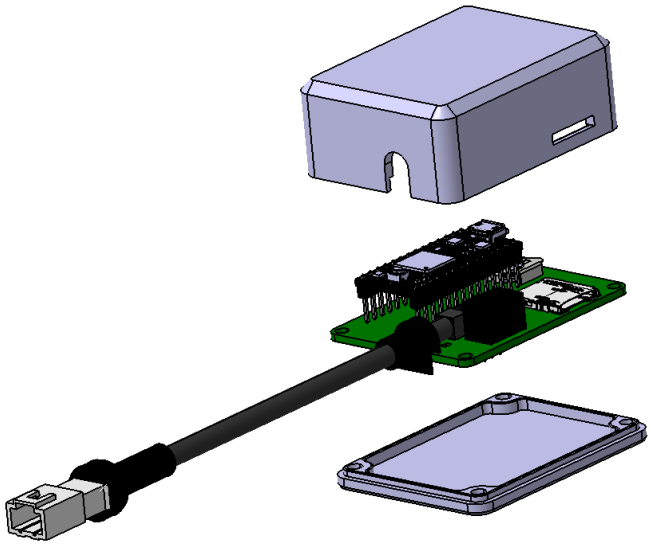


Figure 7 - Enclosure & Vehicle Connection Harness CAD

Figure 8 - 3D Printed Enclosure

2.5. Vehicle Connection

A 4-way, 2.54mm pitch, angled, 2 row pin header is included on the PCB to allow connection to vehicle. Fig. 9 & Table 1 below detail the connections required from the vehicle.

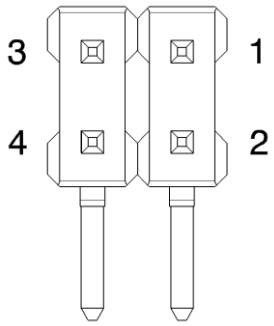


Figure 9 - Board Vehicle Harness Connector

Table 1 - Board Vehicle Connector Pinout

Pin No.	Description
1	+12V
2	GND
3	CANL
4	CANH

The board and enclosure have been designed on the assumption that the wiring bundle will consist of 4off 0.35mm^2 FLRY automotive spec cables sleeved with Raychem DR-25 heat shrink sleeving.

The CAN-Bus wires should be a twisted pair with a lay length of c.20mm.

An example wire bundle is P/N CBR015-0019 which is used on the CBR250RRi project.

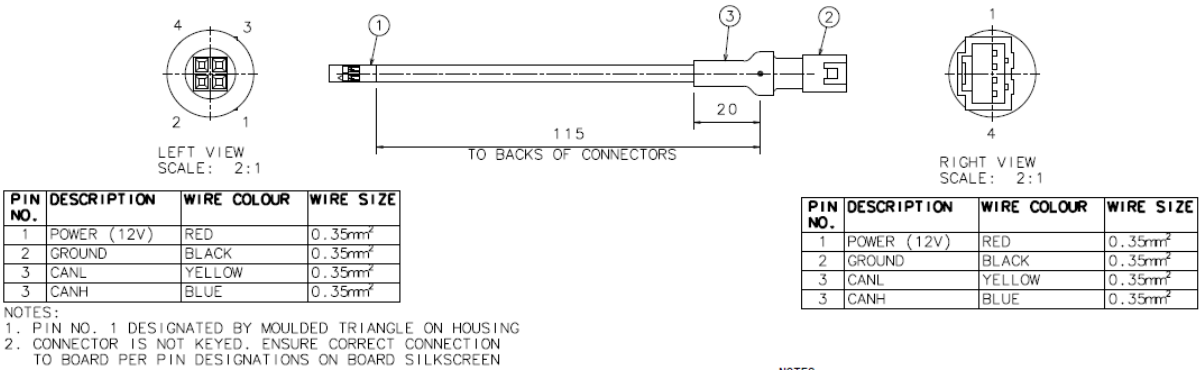


Figure 10 - CBR015-0019 Vehicle Harness Adapter Example

3. Code

The CAN DATALOGGER ASSY code is loosely based around the SDFat library example code.

The code creates a new data file (.dat) on the MicroSD card on boot-up. The filename contains a timestamp obtained from the Teensy RTC which ensures a unique filename and enables the user to identify files when they are downloaded from the MicroSD card.

To allow data to be logged at higher frequencies, the data is written to the MicroSD card as raw binary data.

The dataset to be logged is handled by the Teensy as a struct.

Data is read from the CAN-Bus receive buffer on each pass of the main loop. This ensures that the data struct is updated with new data as quickly as possible to try and minimise data losses.

As well as calling the CAN receive function, each main loop also checks a timer corresponding to the desired logging frequency and calls the log data function when required.

The log data function updates the Time parameter (used by MegaLogViewer as the default base). Time refers to the time since log start and it only starts counting from the first time the log data function is called. After the Time parameter is updated in the struct, the struct data is written to the MicroSD card.

Software versions 1.X decoded the data channels upon receipt and so passed already decoded data to the struct and SD card. Software version 2.0 & later passes raw CAN message data directly to the struct. The struct is ordered by CAN message ID and so is a constant 8 bytes wide. This method of data capture reduces the amount of processing required by the logger, helping to increase reliability and logging rate. It also compresses the data into a smaller footprint as the majority of data is packed into 16-bit chunks while decoded data would get stored as a 32-bit floating point number.

In order to protect for future data logging expansion, the number of message IDs being added to the struct was padded out to 32 messages. This makes the struct size a fixed 256 bytes (8 x 32), allowing the provision for buffering and writing 2 data records to SD at once (512 bytes) and allowing higher logging rates. Additional message IDs are left empty and can be used in the future simply by updating the decoding .dbc file.

The changes made at software version 2.0 make the logger much more flexible in terms of what data is being recorded. It simply requires the struct and message IDs to be based on the required ECU data IDs with additional IDs continuing on from base. If the ECU is changed, the software version will need to be updated to match the different base ECU data IDs.

The code simply loops through reading CAN-Bus data and passing it to the SD card data file until the module is powered off. When the unit is then powered on again, the process begins again but logging the data to a new file. The logger code has been built in this way so that

there is no intervention required by the user to start/stop logging. As long as the logger is powered by the same 12V switched power supply as the vehicle ECU, any time that the ECU is powered on is recorded in a data file.

CAN-Bus speed, logging frequency and logged message IDs are currently defined in the code meaning the base code will need to the changed any time these variables are to be changed. CAN-Bus speed and logging frequency may be adjusted without a code version change. Due to the importance of keeping the logged message IDs aligned with the *.dat file conversion .dbc configuration (see Sect. 4), any adjustments to the logged message IDs must incur a code version update.

The block diagrams in Figs. 11, 12, 13 & 14 describe the basic flow of the code used in the CAN DATALOGGER ASSY. A link to the full base code is provided in Appendix D.

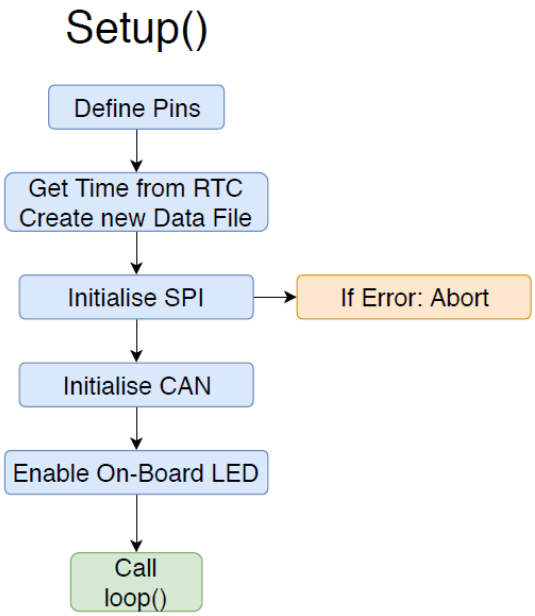


Figure 11 - Setup Code Block Diagram

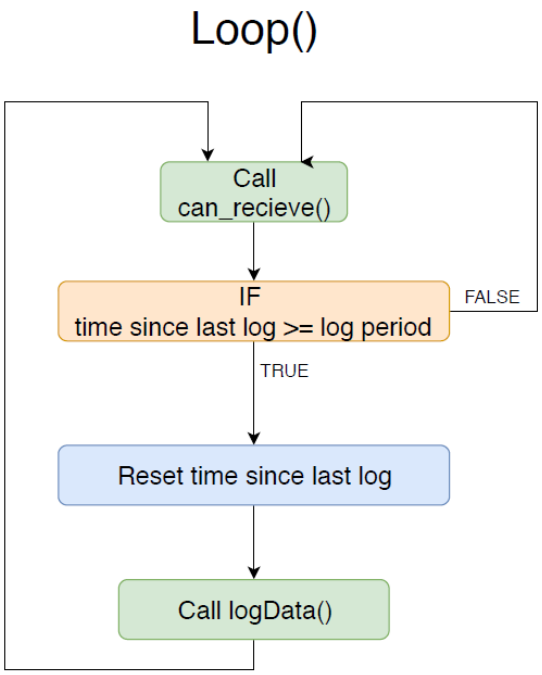


Figure 12 - Loop Code Block Diagram

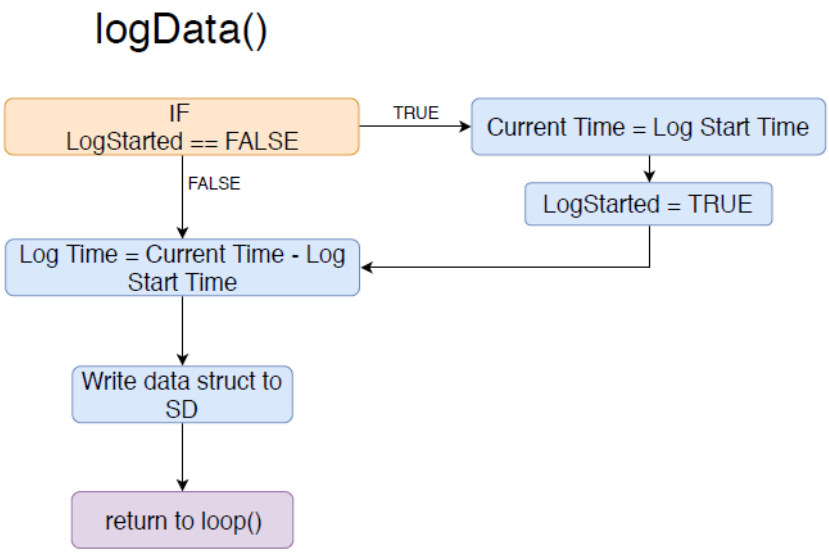


Figure 13 - Log Data Code Block Diagram

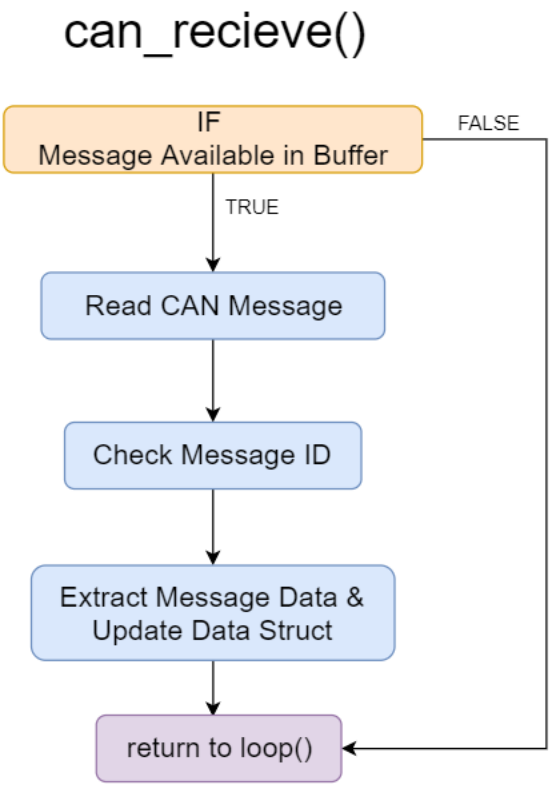


Figure 14 - CAN Recieve Code Block Diagram

3.1. Libraries

The following code libraries are utilised by the software.

- SPI
- SdFat
- TimeLib
- FlexCAN

The SPI library is a standard library included with the Arduino IDE and is used to communicate with the MicroSD card.

In order to read/write to the SD card as efficiently as possible, the SDFAT library is used.

https://github.com/greiman/SdFat

The MicroSD card used should be formatted by a special formatting program for optimum performance and not formatted by the default Windows application.

https://www.sdcard.org/downloads/formatter_4/eula_windows/index.html

The TimeLib and FlexCAN libraries are used for the RTC & CAN-Bus operations respectively. These are both included with the Teensyduino add-on.

4. Data File Processing

As the logger writes data to the SD card in binary format, a separate operation is required to convert the binary data files to human readable files.

Conversion scripts prior to v3.0 carried out post processing with a script written in GNU Octave. These scripts used a text configuration file to define how to interpret the binary data within the recorded file. The configuration file contained the parameter names and units for the output file header and also the data precision.

From conversion scripts v3.0 and later, the script is written in Python and uses a standard CAN DBC file to decode the binary data and provide channel names & units for the output file header.

Using the DBC file format makes it less critical than previous versions to keep track of configuration files. Because the entire raw CAN message data is always written to the binary data file, if the user notices that the decoding is incorrect or a data channel is missing, the dbc file can be updated and the conversion script re-run.

Version 3.0 of the post processing script also has been designed to reduce user input as much as possible. The script relies on a defined folder structure (shown in Fig. 15) which contains the raw & processed data, configuration files and scripts. Using the defined folder structure, the script compares the names of raw data files with processed data files and automatically adds unprocessed files to a list.

The user is then asked to enter a comment for each data file. This comment is later written to an index file to help identify the content of the data file. Therefore, the more description used the better.

The .dbc configuration file is parsed and the number of message IDs determined. This determines the length of a data row. The .dbc file contains channel names & units which are also extracted.

The script exports the processed data in a .csv format which includes headers so that the data can be read directly into MegaLogViewer software for analysis.

For each file processed, information is collected and then written to a datalog index .csv file for quick reference. The information passed to the index file includes:

- Processed file name
- Date of datalog
- Comment
- Track name (provision for future lap timing update)
- Number of laps (provision for future lap timing update)
- Fastest lap time (provision for future lap timing update)
- Input file size
- Log duration in seconds
- Conversion time
- Conversion rate in kB/s

- Conversion rate in s/s

An executable file is created from the Python script which allows the script to be run from any computer without having to have Python and the other dependables installed.

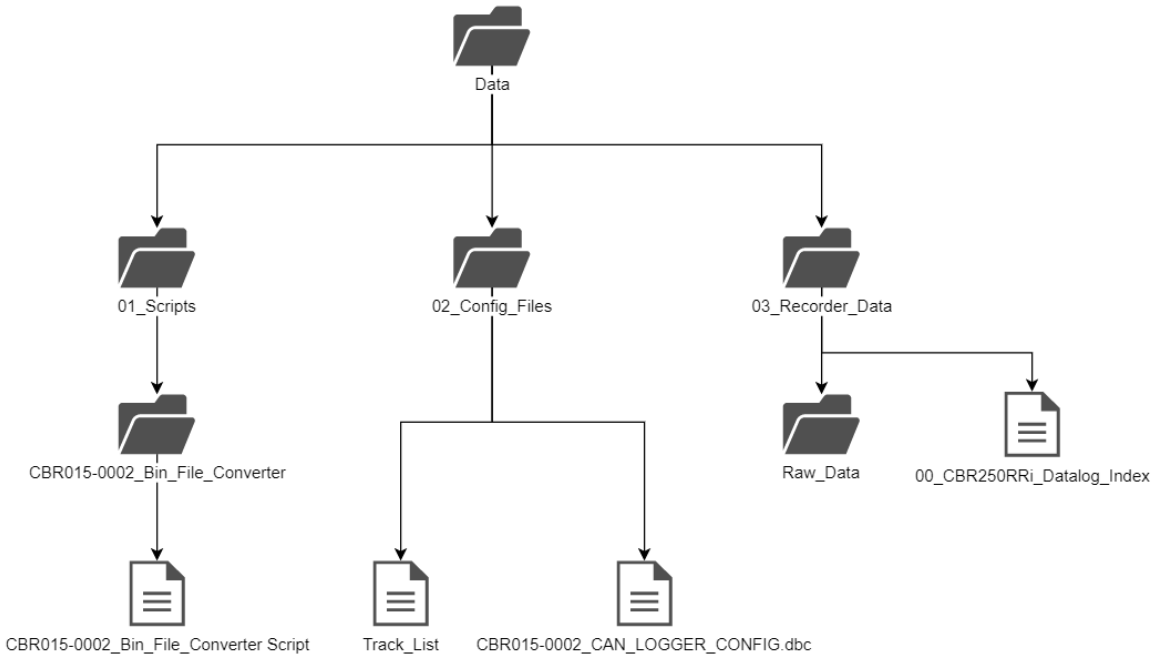
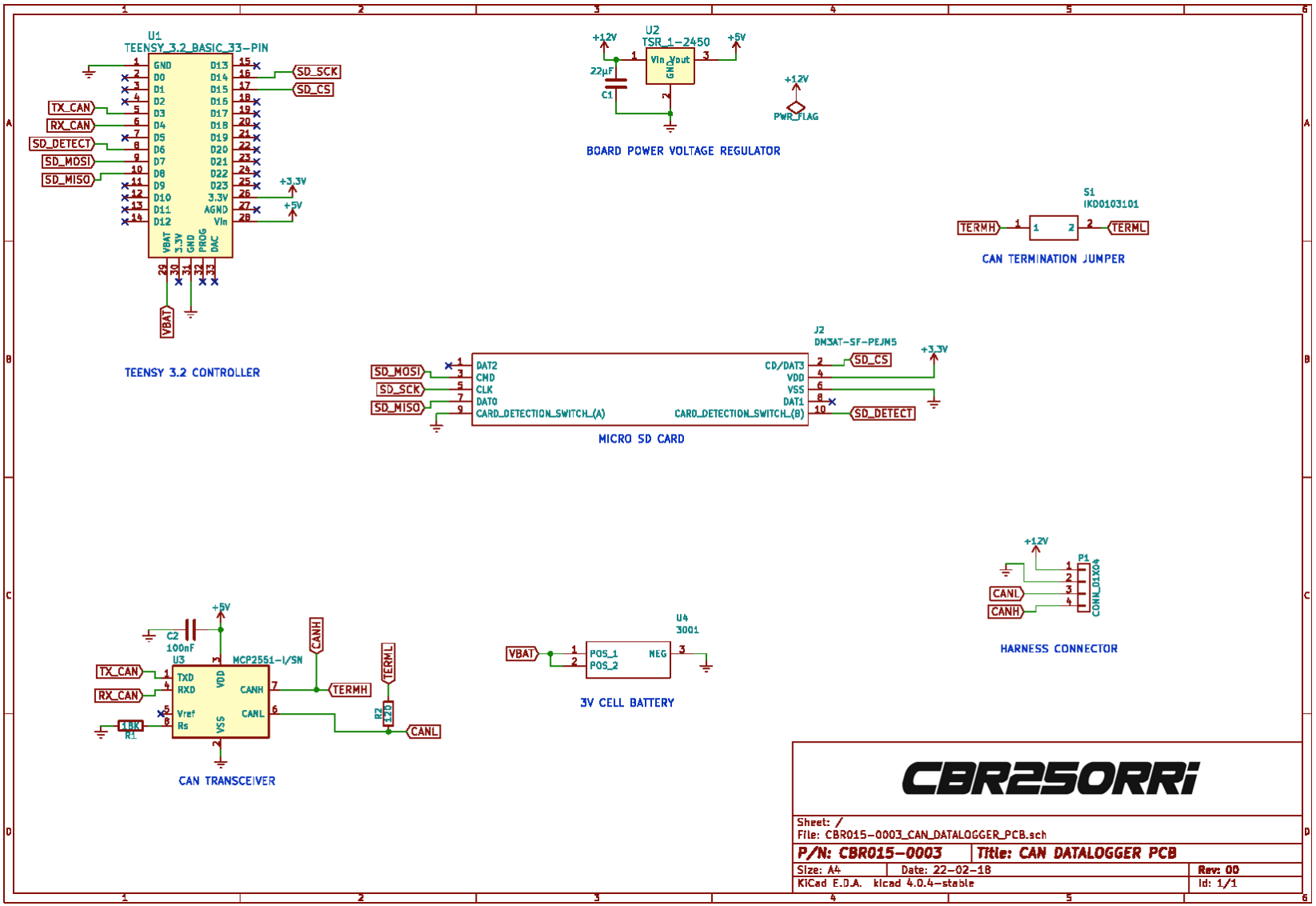


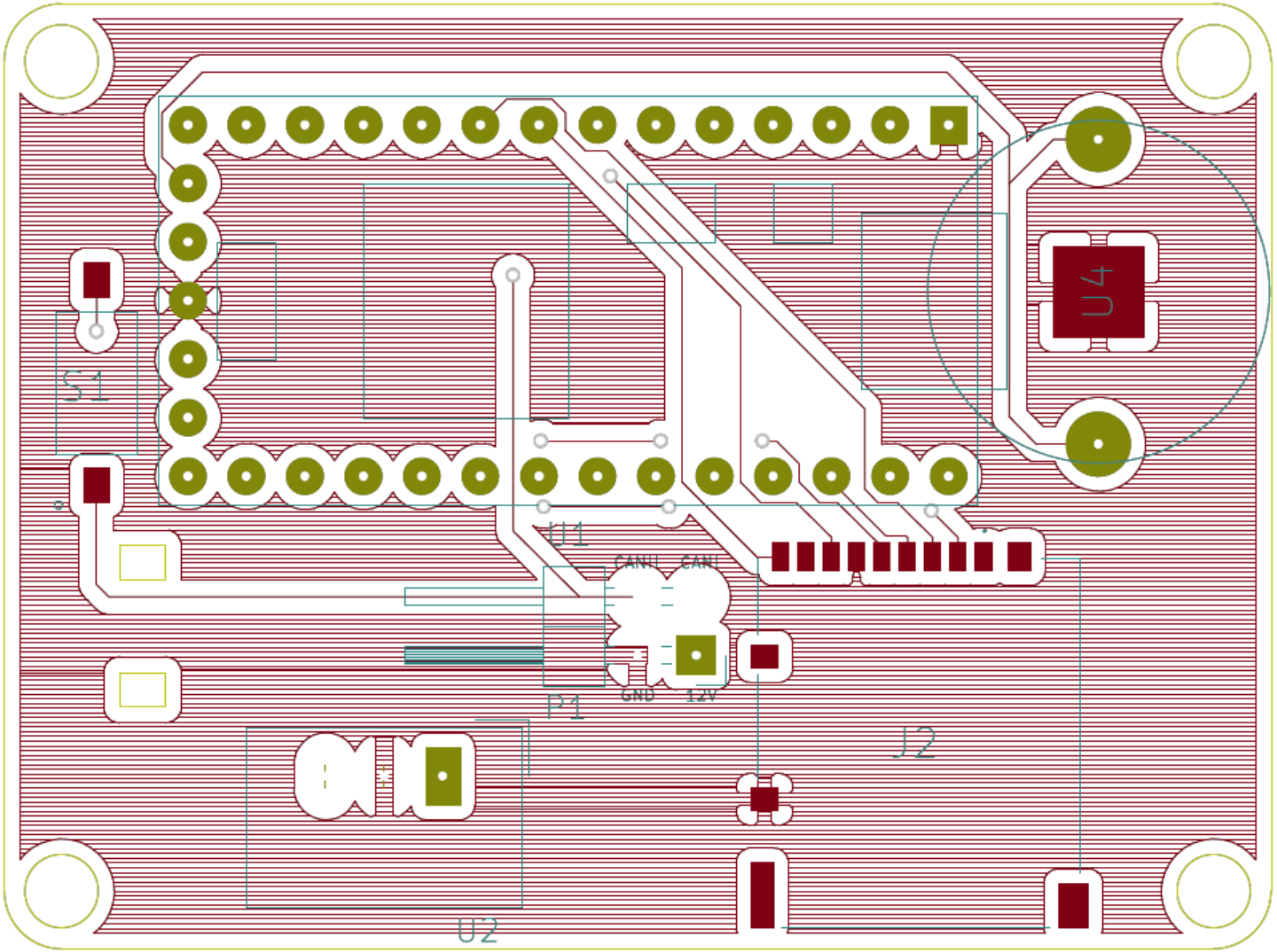
Figure 15 - Conversion Script Folder Structure

Appendix A - Hardware Schematic

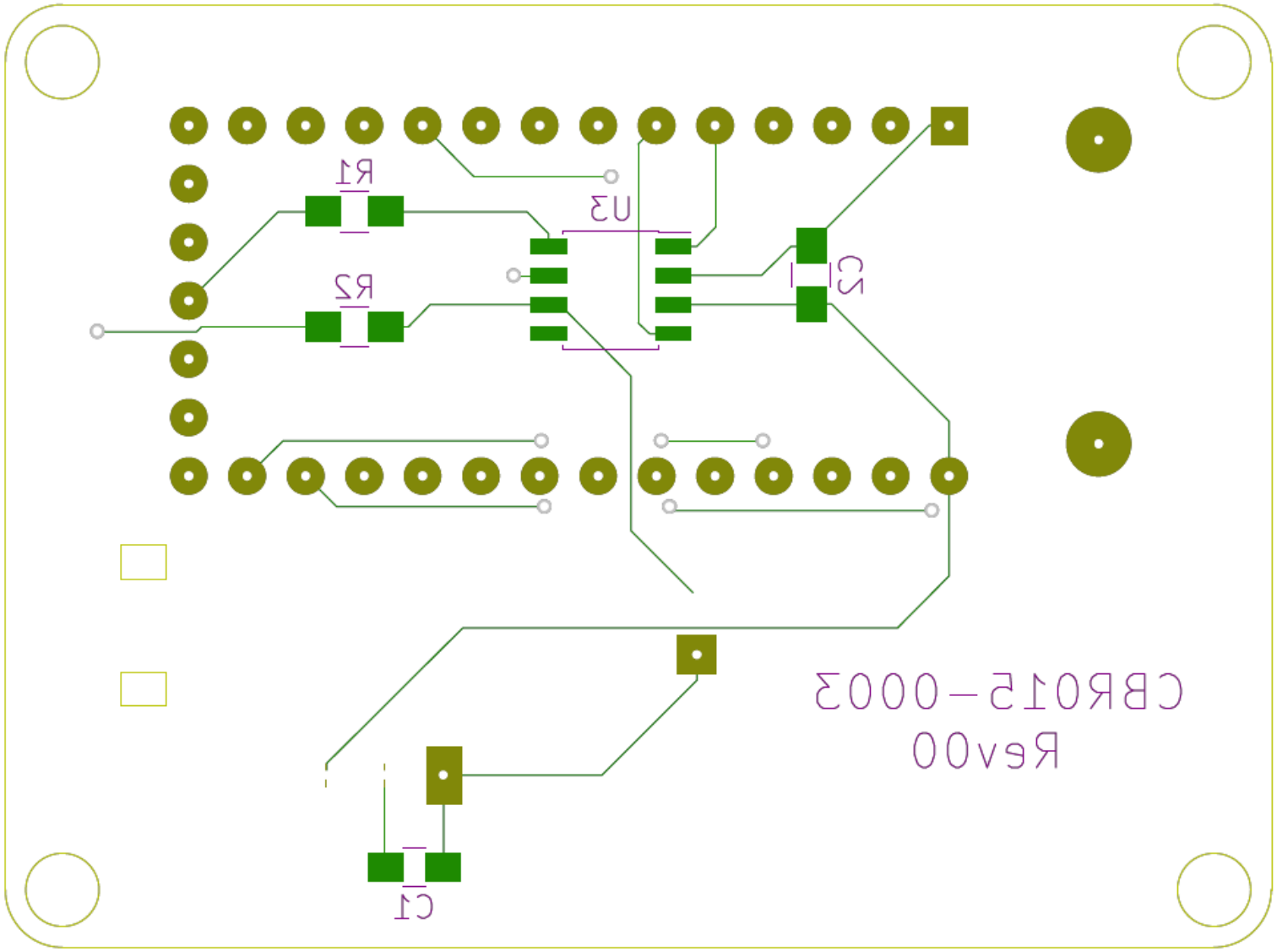


Appendix B - PCB Layout

Front Copper



Back Copper



Appendix C - Bill of Materials

Designation	Description	Value	Manufacturer	Part Number	Package	Supplier	Stock No. / Order Code
U1	Teensy 3.2		PJRC	Teensy 3.2	PDIP	HobbyTronics	
	Teensy Connector Socket	14-way, 1 Row, 2.54mm Socket	Harwin	M20-7823646		RS	488-1724
	Teensy Connector Socket	5-way, 1 Row, 2.54mm Socket	Harwin	M20-7823646		RS	488-1724
	Teensy RTC Crystal	32.768 kHz, 12.5 pF	IQD	LFXTAL025159		RS	148-4321
U2	Board Power Regulator	5V, 1A	Traco Power	TSR 1-2450	SIP 3	RS	666-4379
U3	CAN Transceiver		Microchip	MCP2551-I/SN	SOIC 8	RS	738-6036
U4	Coin Battery Holder		Keystone	3001		RS	219-7948
	Coin Battery	3V	Various	CR1216 or CR1220			
C1	Power Reg Smoothing Cap	22µF	Murata	GRM219B30G226ME66D	0805	RS	798-4624
C2	CAN Transciever Input Smoothing Cap	100nF	AVX	08055C104KAT2A	0805	RS	464-6688
R1	CAN Transciever Reset Resistor	18K	TE Connectivity	CRG0805F18K	0805	RS	223-0590
R2	CAN Termination Resistor	120	Panasonic	ERJP06F1200V	0805	RS	721-7674
J1	MicroSD Holder		Hirose	DM3AT-SF-PEJM5		RS	828-1884
S1	CAN Termination Jumper	Single SPST	APEM	IKD0103101		RS	877-2233
P1	Board Connector	4-Way, 2 Row, 2.54mm Right Angle Pin Header	RS PRO	251-8660		RS	251-8660

Appendix D - Code

The latest version of Arduino code can be found at the following location.

D:\Google Drive\Vehicles\Honda CBR250RR\Technical\015 - ELECTRICAL SYSTEM\CBR015-0002 CAN DATALOGGER ASSY\CBR015-0002 Arduino Code

The corresponding Python post-processing script source code can be found at the following location.

D:\Google Drive\Vehicles\Honda CBR250RR\Technical\015 - ELECTRICAL SYSTEM\CBR015-0002 CAN DATALOGGER ASSY\CBR015-0002 Bin File Converter

Appendix E - CAN Config

The CAN Configuration document corresponding to the parameters logged by the CAN DATALOGGER ASSY can be found at the following location.

D:\Google Drive\Vehicles\Honda CBR250RR\Electrical\CBR015-0002 CAN DATALOGGER ASSY

The .dbc file corresponding to the Excel document is located in the same folder.