|  |  |
| --- | --- |
| CURRENT TRASFORMER METER  CTM-3501 | Abstract  This report contains the design, implementation, and testing of a non-invasive near to real-time current metering protocol for alternating systems.  Michael Giorgas, Alex Olsen & Clinton Elliott  Electrical Engineering BA (Electronic)  CC3501 |

You had me at “Hello World!”

-Anonymous

TABLE OF CONTENTS

[1. INTRODUCTION 1](#_Toc497138610)

[2. OVERVIEW OF DESIGN 3](#_Toc497138611)

[3. HARDWARE DESIGN 4](#_Toc497138612)

[3.1. Schematic 4](#_Toc497138613)

[3.1.1. Power Supply 4](#_Toc497138614)

[3.1.2. Operational Input Shifters 4](#_Toc497138615)

[3.1.3. Multiplexer 5](#_Toc497138616)

[3.1.4. ZigBee 6](#_Toc497138617)

[3.1.5. Mini-USB 6](#_Toc497138618)

[3.1.6. M5 & H5 Processors 7](#_Toc497138619)

[3.1.7. JTAG and SDA Headers 7](#_Toc497138620)

[3.1.8. 3-Stage Contingency 8](#_Toc497138621)

[3.1.9. Battery Selection 8](#_Toc497138622)

[3.2. Power Optimisation 10](#_Toc497138623)

[3.2.1. Operational Amplifiers 10](#_Toc497138624)

[3.2.2. Voltage Regulator 10](#_Toc497138625)

[3.2.3. XBee Pro 11](#_Toc497138626)

[3.2.4. MK20DX128VLH5 & MK20DX128VFM5 Processors 11](#_Toc497138627)

[3.2.5. SN74LVC125A Quadruple Bus Buffer Gate 12](#_Toc497138628)

[3.2.6. HEF4066B Quad Single-Pole Single-Throw Analog Switch (Multiplexer) 12](#_Toc497138629)

[3.2.7. Mini USB 12](#_Toc497138630)

[3.2.8. LEDs 12](#_Toc497138631)

[3.2.9. Battery Sizing 12](#_Toc497138632)

[3.2.10. Power-Saving Strategies 14](#_Toc497138633)

[3.3. PCB Design 15](#_Toc497138634)

[4. SOFTWARE DESIGN 17](#_Toc497138635)

[4.1. Kinetis Code 17](#_Toc497138636)

[4.1.1. Analog to Digital Converter 17](#_Toc497138637)

[4.1.2. Timing 17](#_Toc497138638)

[4.1.3. Data Percentage 17](#_Toc497138639)

[4.1.4. Data Calculation & Mapping 18](#_Toc497138640)

[4.1.5. ZigBee Transmission 19](#_Toc497138641)

[4.1.6. Calibration 21](#_Toc497138642)

[4.1.7. Boot Loading 21](#_Toc497138643)

[4.2. C++ Code 22](#_Toc497138644)

[4.2.1. ZigBee Data Acquisition 22](#_Toc497138645)

[4.2.2. Wi-Pi Upload 23](#_Toc497138646)

[4.2.3. ThingSpeak Display 24](#_Toc497138647)

[4.3. Webpage Code 25](#_Toc497138648)

[4.3.1. Hosting 25](#_Toc497138649)

[4.3.2. Hyper Text Mark-up Language (HTML) 25](#_Toc497138650)

[4.3.3. Cascading Style Sheet (CSS) 25](#_Toc497138651)

[4.3.4. File Transfer Protocol (FTP) 26](#_Toc497138652)

[5. TESTING & RESULTS 27](#_Toc497138653)

[5.1. Real-World Set Back 27](#_Toc497138654)

[5.2. Final Product 28](#_Toc497138655)

[5.3. Webpage Display 29](#_Toc497138656)

[5.4. Game-Day Performance 30](#_Toc497138657)

[5.5. Results 31](#_Toc497138658)

[6. LIMITATIONS & RECOMMENDATIONS 32](#_Toc497138659)

[6.1. Limitations 32](#_Toc497138660)

[6.2. Recommendations 33](#_Toc497138661)

[7. CONCLUSION 34](#_Toc497138662)

[8. REFERENCES 35](#_Toc497138663)

[APPENDIX A.1 – SCHEMATIC 37](#_Toc497138664)

[APPENDIX A.2 – SCHEMATIC 38](#_Toc497138665)

[APPENDIX B – PCB LAYOUT 39](#_Toc497138666)

[APPENDIX C – BILL OF MATERIALS 40](#_Toc497138667)

[APPENDIX D – KINETIS CODE 41](#_Toc497138668)

[APPENDIX E – RASPBERRY PI CODE 55](#_Toc497138669)

[APPENDIX F – WEBPAGE CODE 60](#_Toc497138670)

LIST OF FIGURES

[Figure 1 - Current Transformer [1] 1](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138671)

[Figure 2 - Overview of Operation 3](#_Toc497138672)

[Figure 3 - Power Supply and Corresponding Protection 4](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138673)

[Figure 4 - Operational Amplifiers and Input Shifting Circuit 5](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138674)

[Figure 5 - Multiplexer 5](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138675)

[Figure 6 - ZigBee Transmitting Device 6](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138676)

[Figure 7 - Mini USB Circuit 6](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138677)

[Figure 8 - Primary (left) and SDA (right) Processors and Oscillators (bottom right) 7](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138678)

[Figure 9 - SDA (left) and JTAG (right) Headers 7](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138679)

[Figure 10 - FRDM Board Headers 8](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138680)

[Figure 11 - Coin Cell Battery and Isolator 9](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138681)

[Figure 12 - Auto Routed Test of Placement 15](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138682)

[Figure 13 - SDA Processor with Poor Routing 15](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138683)

[Figure 14 - Finished PCB 16](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138684)

[Figure 15 - Definition of Timing in Circuit (seconds) 17](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138685)

[Figure 16 - Shifted Waveform from CT to ADC Input 18](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138686)

[Figure 17 - Mapping Layout of Data 19](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138687)

[Figure 18 - Frame Details 20](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138688)

[Figure 19 - Xbee API Packet 20](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138689)

[Figure 20 - Creation of Frame Packet 20](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138690)

[Figure 21 - Checksum and AS Transmission 21](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138691)

[Figure 22 - Receiving Code from Zigbee 22](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138692)

[Figure 23 - Libcurl Code 23](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138693)

[Figure 24 - Transmission with API Key (yellow) to ThingSpeak 24](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138694)

[Figure 25 - Hosting Website 25](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138695)

[Figure 26 - HTML Initial Setup 25](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138696)

[Figure 27 - Background Style Sheet 26](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138697)

[Figure 28 - Transferring Files to the Host 26](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138698)

[Figure 29 - Burnt PCB 27](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138699)

[Figure 30 - Finished PCB and One CT Connected 27](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138700)

[Figure 31 - Final Product with Case and One CT Connected 28](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138701)

[Figure 32 - The Checking Of Data Being Sent By CTM-3501 And Being Displayed On The Pi And Internet Webpage 29](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138702)

[Figure 33 - The CTM-3501 Setup 30](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138703)

[Figure 34 - Current Draw of Vacuum Cleaner 31](https://d.docs.live.net/c07c9239cf8ef362/4th%20Year%20Sem2/CC3501/Github/CT-Monitoring-Assignment/CTM%20-%20Report%20%5e0%20Manual/CT-Metering-Assignment-R_6.docx#_Toc497138704)

CURRENT TRASFORMER METER

# INTRODUCTION

The aim of this project was to design and implement the CTM-3501. This report outlines the design of the circuit with calculations and diagrams. The implementation in Code Warrior using Free Scale will be discussed and explained. The final product and testing will be evaluated, commented upon, and recommendations are given.

The CTM-3501 (CTM for short) stands for Current Transformer Meter and followed by the model number i.e. CTM-xxxx. The 35xx series are the top of the range currently released on the market. The CTM was initially designed to meet the minimum requirements of having an embedded system with a sensor which acquires data. The features to be implemented were given flexible conditions, while the team behind the CTM were driven to achieve a glorious end.

A current transformer (CT) is and electrical device used to step or scale current down to a safe working current which can be readily measured by meters and relays for protection and monitoring. The CT gives a proportional current in its secondary winding to the current in the primary winding based on the turns ratio, and should have approximately a negligible load. Owing to the construction of CTs, they provide electrical isolation from the higher voltage circuit through the windings, consequently, increasing safety. Generally, these transformers have a small number of primary turns and a larger number of secondary turns which defines the turns ratio [1]. Figure 1 shows a CT comprising of the primary and secondary windings, with the secondary winding earthed.

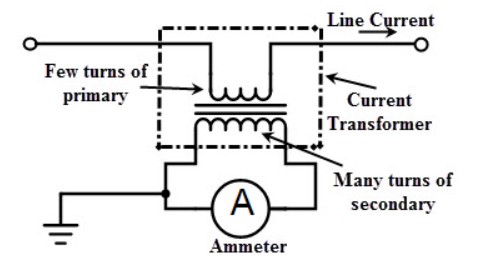


Figure 1 - Current Transformer [1]

There are several types of CTs including wound type, bar type and window type transformers. The CT behaves similar to a normal transformer in no load conditions (small burden) and therefore the secondary normally has low currents and is earthed for safety precautions. The current on the secondary winding is governed by the equation:

Where is the primary winding current, is the secondary winding current, and are the number of turns for the primary and secondary windings respectively. If the secondary winding is not shorted (by a burden resistor) than high dangerous voltages can occur governed by the equation:

To demonstrate this effect, we assume and are 1 and 100 with a primary voltage of 230. Substituting into the equation:

Therefore, a CT with a turns ratio of 100:1 can develop 23 kV with a standard (Australian) nominal phase voltage. Most insulation is rated to 1000 V and this high voltage can degrade the insulation creating risks. Current transformers are used in many aspects of a power system. They are utilised in power distribution, generating stations, substations and at domestic, commercial, and industrial levels [1].

All electronic systems require power supplies to operate, power management broadly refers to the generation and control of regulated voltages required to operate an electronic system. It encompasses more than simply power supply design. Today's systems require power supply design be integrated with system design to maintain high efficiency. Integrated circuit components such as switching regulators, linear regulators, switched capacitor voltage converters, and voltage references are typical elements of power management.

The battery life of the circuit is determined by two factors: firstly, the battery capacity, which is the total amount of current that the battery can supply during its lifetime; secondly, the load current of the circuit i.e. the amount of current being consumed by the circuit. The first factor can be found by the manufacturer details of the battery which gives the capacity in amp-hours or milliamp- hours, the second factor requires the calculation of the current drawn from the active devices in the circuit. The following formula is used to determine the battery life.

(1)

# OVERVIEW OF DESIGN

The fundamental operation of the CTM is displayed in Figure 2 and depicts the start of the data acquisition from the current transformers. The voltage is shifted using the wave shaping and converted to a digital signal. This signal is then computed to get a desired output based on the calibration phase and type of current transformer being used. The data is then sent over radio frequency to a base station Raspberry Pi and uploaded via WiPi to ThingSpeak and ctm-3501.com for webpage display.

Figure 2 - Overview of Operation

Essentially, the design aimed to monitor an alternating current using a current transformer (CT). The CT’s were to be part of a sensor node which sent information via wireless communication to a base station. The base station was a Raspberry Pi which could forward near to real-time data upload to the internet. The complete design process of the CTM is explained in this document and a corresponding user manual is attached.

# HARDWARE DESIGN

## Schematic

The CTM was based on the embedded system design of the Kinetis FRDM-K20DX128M5 board for the processing circuit. The schematic was compartmentalised during the design process and each part is discussed as per its design in the following sections.

### Power Supply

The two processors each required a 3.3 V power supply, so during the design this was the primary reason for selecting 3.3 V. The power supply design is shown in Figure 3.

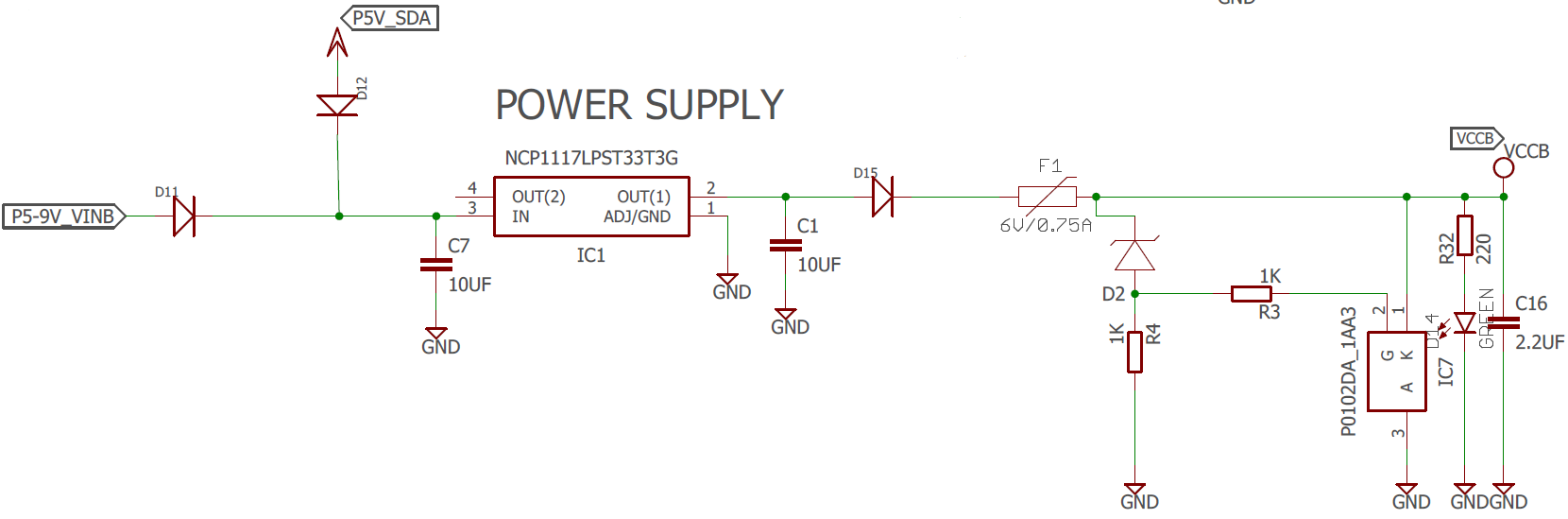


Figure 3 - Power Supply and Corresponding Protection

The supply for the circuit was from a 9 V coin-cell battery or from a 5 V USB with both passing through the diodes D11 and D12 to prevent incorrect connection. Each supply was fed into a power supply with the capacitor sizes selected from the data sheet for the NCP1117LPST33TCG. It was noted that at 3.3 V the regulator could output current in excess of 1.0 A. The poly fuse F1 was selected at 0.75 A as this would protect the regulator and was above the calculated maximum current draw of the circuit (See Section 3.2). The SCR, D2 and R4 provide Crowbar protection in the event of voltage spikes. D2 was selected at 3.5 V to prevent transients entering the processors and causing damage. The poly fuse provides over-current protection and the crowbar protection provides over-voltage protection.

### Operational Input Shifters

The CTs transduce the current into a voltage which is fed into the LM358 operational amplifier (op-amp). The output voltage is level shifted from the input bias voltage on the non-inverting pin from the voltage division over the potentiometer VPOT as shown in Figure 4.

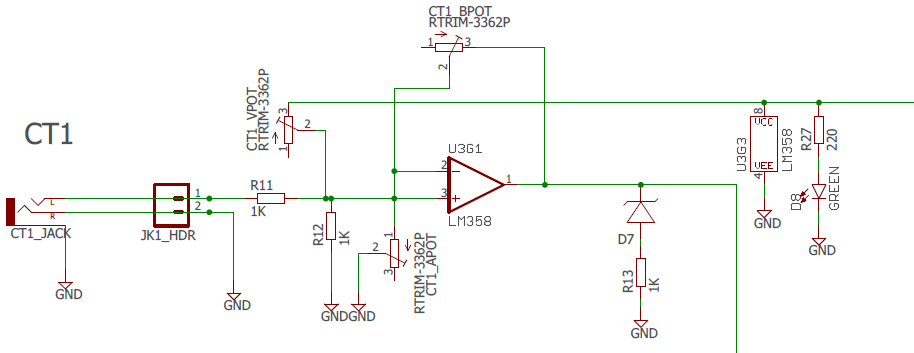


Figure 4 - Operational Amplifiers and Input Shifting Circuit

Figure 4 shows the diode D7 which clipped the output at 3.3 V. If the output voltage was higher than 3.3 V then it was clipped to earth protecting the input pins to the processor against transients. The original values for the POTs were calculated but it was found that each needed to be adjusted due to manufacturer tolerances. The waveform needed to be shifted positively half the maximum input voltage of 3.3 V. This meant 1.65 V input bias on the non-inverting pin, but this was affected during prototyping due to power supply voltages. Note the headers which were installed near the headphone jack for testing purposes.

### Multiplexer

The multiplexer (MUX) was incorporated to reduce the power consumed by the op-amps and POTs. The MUX was supplied power by the microprocessor and switched power to the op-amp circuits. This allowed the op-amp circuits to be only powered while data was being recorded. The MUX is shown in Figure 5 and the timing is shown in Figure 15.

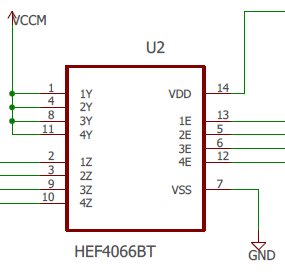


Figure 5 - Multiplexer

The MUX is powered during the cycle (see Figure 15) and switches between CT1, CT2, CT3 and CT4. This allows only one CT circuit to be powered at a time reducing the power consumed by four. The upshot of switching from the microprocessor was that the MUX was only powered during each cycle which reduced the amount of power consumed by a factor of 15.

### ZigBee

The Xbee module brand ZigBee made by Digi International was chosen for the wireless communication due to its low power and low data rate, which is ideal for battery applications. The ZigBee communication standard supports mesh networks and is used in applications with sensor networks that require machine-to-machine communications.

The ZigBee was selected due to familiarity and the ability to place the device into sleep mode by switching the sleep pin from the microprocessor and is shown in Figure 6. This meant that only once a phase the ZigBee had to be powered which decreased the power consumption.

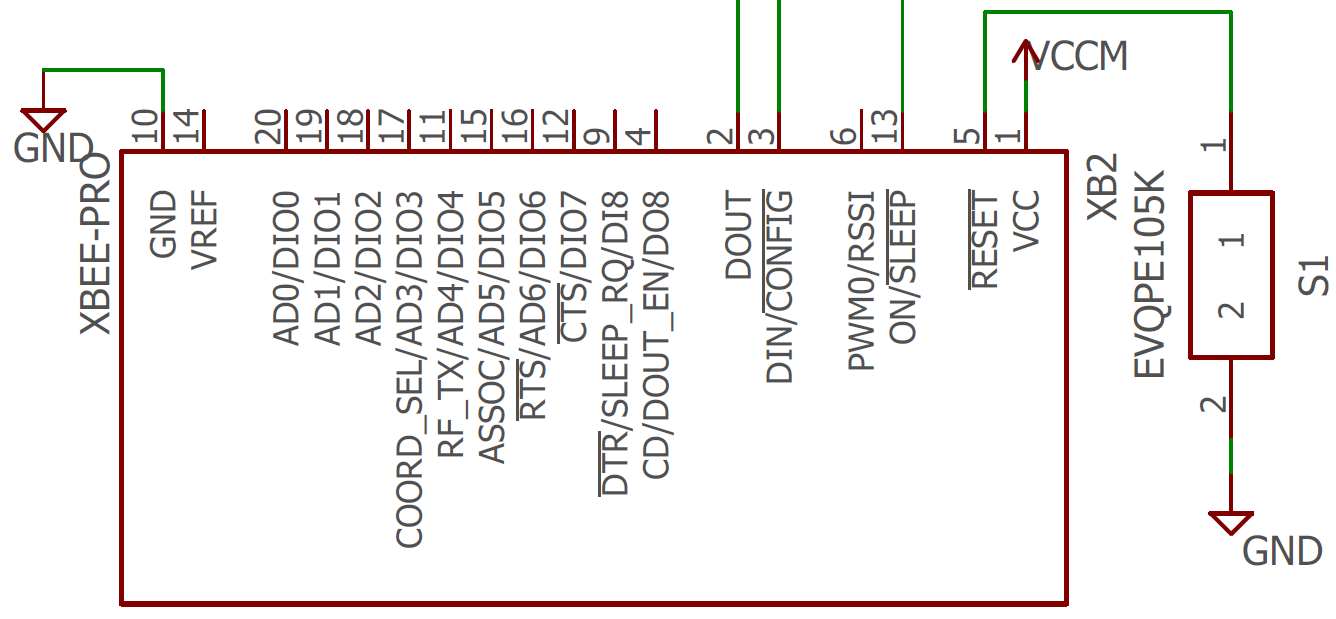


Figure 6 - ZigBee Transmitting Device

### Mini-USB

The mini USB (Figure 7) was selected to allow connection to the SDA processor to convert the signals to serial to program the primary processor. The BZT52C15S Zener (D1) was installed to protect the board from high input transients and static. The L1 and L2 inductors or ferrite beads were installed to further decrease these effects.

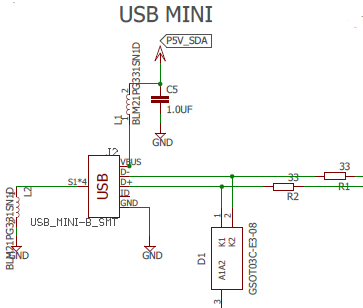


Figure 7 - Mini USB Circuit

### M5 & H5 Processors

There were two microcontrollers used in the design: the MK20DX128VLH5 and the MK20DX128VFM5 which will be referred to here after as the Primary and the SDA processors. As mentioned previously these two processors were selected as they were based off the FRDM board and are shown in Figure 8. These two processors required three oscillators shown in Figure 8 at frequencies of 32 kHz and 8 MHz.

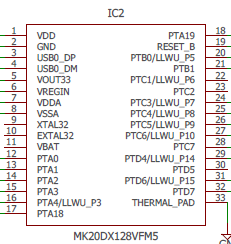
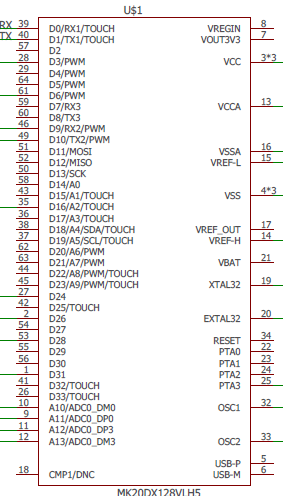
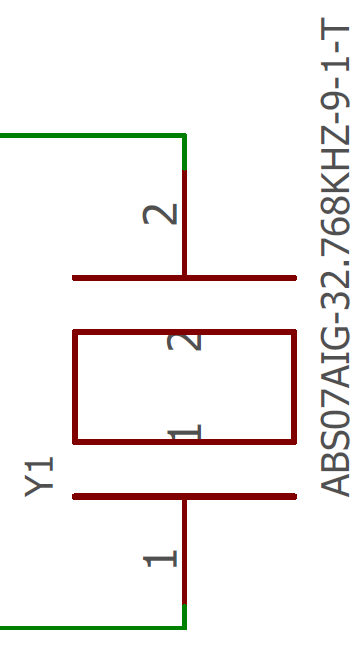
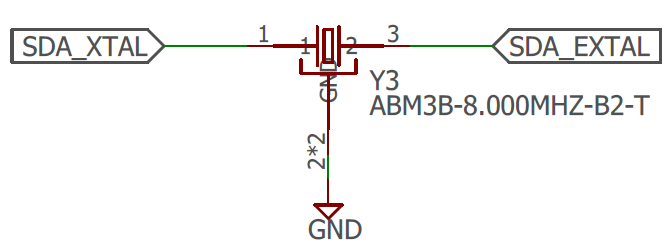


Figure 8 - Primary (left) and SDA (right) Processors and Oscillators (bottom right)



### JTAG and SDA Headers

The JTAG and SDA headers were installed to allow serial communication directly to both processors and are shown in Figure 9. The SDA header allowed communication to the primary processor and the JTAG header allowed communication to the SDA processor.

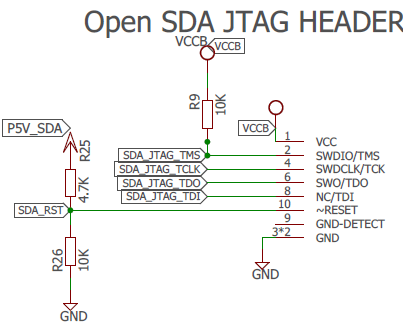
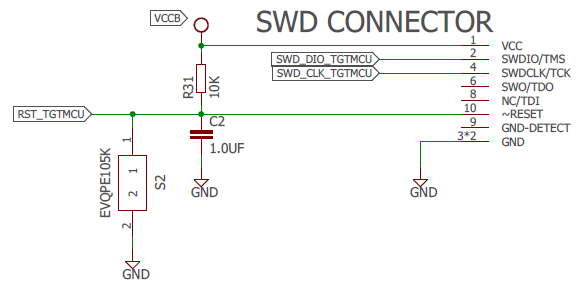


Figure 9 - SDA (left) and JTAG (right) Headers

### 3-Stage Contingency

During the development of the CTM system it was decided to include stages of contingency due to the frontiers which were being pushed.

* Stage 1 Contingency was to essentially build the board in reference to headers which could be connected to a FRDM board. This meant the CTM could be inserted into a FRDM board and operate. This was achieved by connecting to FRDM headers (see Figure 10) to all the input and outputs from the CTs. The CT circuits would receive power from the FRDM board and this supply rail was called the Main (VCCM).

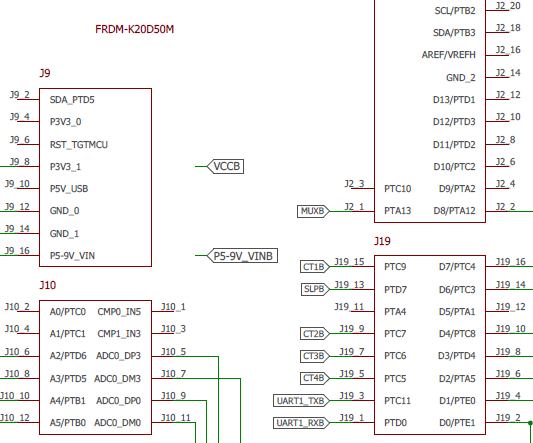


Figure 10 - FRDM Board Headers

* Stage 2 Contingency was having the primary processor drive the circuit connected across the FRDM headers by bridges. This circuit was powered by the onboard CTM power supply and labelled the Backup power supply (VCCB).
* Stage 3 Contingency was including the SDA chip and allowing the CTM to be programmed by mini USB interface.

### Battery Selection

The battery was isolated from the board by a 2-position dip switch which disconnected the positive and negative of the battery.

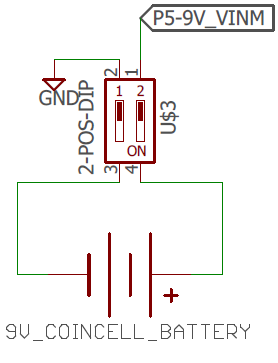


Figure 11 - Coin Cell Battery and Isolator

The Energizer 522 Alkaline 9-volt battery capacity is rated at 230 milliamp-hours at a 1000 milliamp discharge rate and was initially selected for its footprint but was pending the battery sizing calculations.

## Power Optimisation

### Operational Amplifiers

To determine the power draw of the op-amp the quiescent current was obtained from the data sheet and the formula:

The data sheet provided a typical value and a maximum value for the quiescent current, each value was used to obtain the power of the op-amp with the max value used as a worst-case scenario.

per Op-Amp

per Op-Amp

The component is a dual op-amp package; therefore, each package will draw and the board has 4, resulting in

### Voltage Regulator

The maximum package power dissipation of the NCP1117LP voltage regulator is given by the formula:

* - maximum junction temperature range
* - operating ambient temperature range
* - thermal Resistance, Junction−to−Ambient

The maximum power :

The maximum current output of the regulator is governed by:

* V is the voltage drop of the regulator

To calculate a typical power rating of the voltage regulator it is somewhat hidden within the data sheet. It can be calculated by looking at the following related specifications.

* - operating junction temperature range
* - thermal resistance junction-to-case
* - thermal resistance, junction−to−ambient

specifies the temperature of the “junction”, the active part of the regulator, can get before it goes into thermal shutdown. specifies how much temperature difference to be expected between the junction and the outside of the package. This is relevant if you cannot quickly remove heat from the package. With a perfectly coupled heat sink hooked to the package, for each watt the junction temperature would rise only above the temp of the heat sink. is how hot the junction gets when the regulator is dissipating a given amount of power and the regulator is sitting at a given ambient temperature. We designed our regulator to work under modest commercial conditions, such that, it will not exceed . The junction temperature needs to stay below , therefore, the maximum temperature rise allowable is The power dissipation is given by:

The current is :

### XBee Pro

The XBee Pro attached to the board will draw current during transmission and when in its sleep mode. The maximum current draw will be during transmission mode with the data sheet quoting a transmit current of The power down current of the XBee is

### MK20DX128VLH5 & MK20DX128VFM5 Processors

The absolute maximum ratings for the MK20DX128VFM5 and MK20DX128VLH5 are obtained from the device data sheet. The maximum power supply current , includes all current being sourced by the microcontroller pins in addition to the current used to operate the CPU and peripherals. For the MK20DX128VLH5 and MK20DX128VFM5 the current is:

### SN74LVC125A Quadruple Bus Buffer Gate

The absolute maximum current draw of the SN74LVC125A was calculated from the manufacturers data sheet by adding the continuous input current and the output current.

### HEF4066B Quad Single-Pole Single-Throw Analog Switch (Multiplexer)

The maximum supply current at worst case scenario for the Multiplexer was while the device is operating at an ambient temperature of .

### Mini USB

The specifications of USB 2.0 states that the maximum current draw was .

### LEDs

The current drawn from the LED’s can be found using the formula:

* Supply voltage
* LED forward voltage drop (found in data sheet)
* Resistor value

The board has a total of 6 LEDs taken the worst case that all LEDs are on at the same time the total current draw would be:

### Battery Sizing

Several factors affect the battery life of the design and the primary considerations include: devices active, sizes, and duty cycle. Table 1 lists the primary power consuming components and their maximum “worst case” current draw. These values are a worst-case scenario which are the absolute maximum current drawn by each component.

Table 1 - Current Draw of Design

|  |  |
| --- | --- |
| Component | Maximum Current |
| Op-Amps |  |
| Voltage Regulator |  |
| XBee Pro |  |
| MK20DX128VLH5 |  |
| MK20DX128VFM5 |  |
| NAND Gates |  |
| Multiplexer |  |
| LED’s |  |
| Mini USB |  |
|  |  |

The battery life calculation will be done using the worst-case scenario for each component, assuming that all components are at maximum draw at the same time and operating all the time. The total is , which is approximately 1000 mA (1 A).

Using the battery life equation (1):

The worst-case scenario indicates that the board could stand alone power itself for approximately 14 minutes. Therefore, power saving measures were undertaken.

### Power-Saving Strategies

To save power the following designs innovations were implemented:

* Reduction in clock speed
* Switched power to Multiplexor
* Switched power to CT channels
* If a low value was read, then it turned off a CT channel for 30 minutes
* Sleep mode utilised with ZigBee
* Waiting for interrupts and putting the CPU into low power mode

## PCB Design

The selected method of layout approach for the PCB was trial and error. This approach allowed many designs to be made, consuming copious amounts and time, and delivering maximum returns in the form of stress. The size of the board was limited to 80x100 mm as this is the maximum allowable board size in the free version of eagle. This proved challenging in the creation of the PCB.

All the components were placed on the board and it was auto-routed to give an indication of placement pattern. The result yielded 84.2 % routed (see Figure 12), which was not high and indicated poor placement. It was decided to place all the components and route manually.

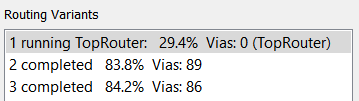


Figure 12 - Auto Routed Test of Placement

All of the components were successfully routed but the SDA processor, which had 32 pins and small package size, proved difficult routing which originally could not be routed as indicated by the three air wires in Figure 13.

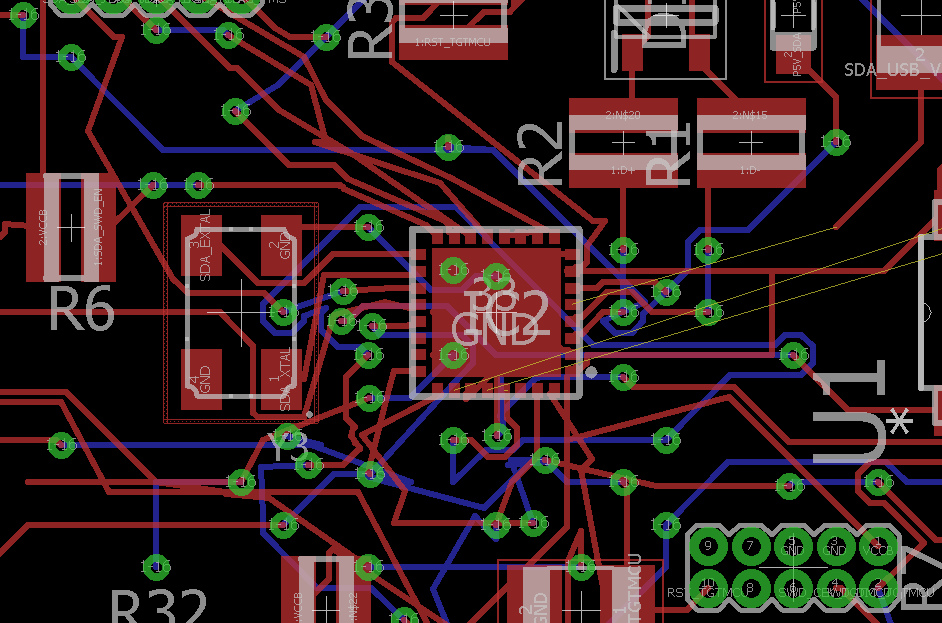


Figure 13 - SDA Processor with Poor Routing

Routing manually allowed for the optimal placement of the components, this included:

1. Placing all the CT circuits in their individual circuits off to the side
2. Increasing the free space around each processor for routing
3. Allow room outside the FRDM headers for routing
4. Placing the power supply components together in a corner
5. Relocating the USB outside of the FRDM Headers

The finished PCB is shown in Figure 14 and the areas of noted above are shown with their corresponding letters.

B

E

B

A

D

C

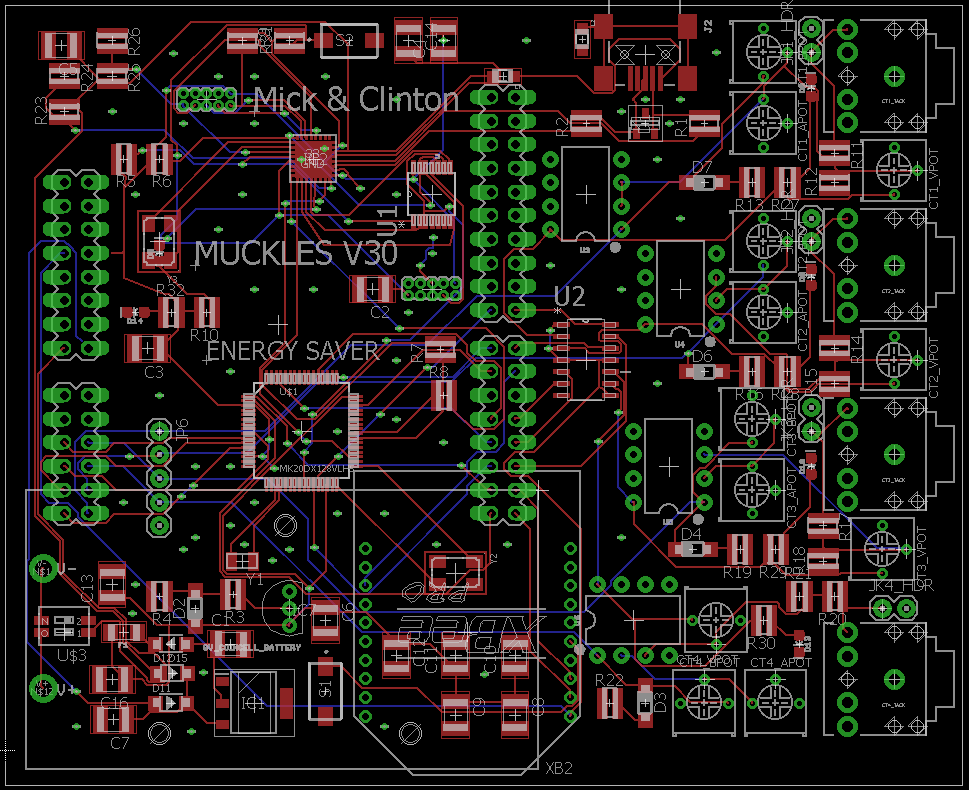


Figure 14 - Finished PCB

# SOFTWARE DESIGN

## Kinetis Code

### Analog to Digital Converter

The Kinetis code was developed by adding on each task successively. The first step was to setup the project and get it receiving data from the channels. The pins for the Analog to Digital Converter (ADC) channels were setup when designing the schematic. The advantage of using FRDM-K20D50M processor was the familiarity with the syntax of C and the onboard ADC.

### Timing

Figure 15 shows the operation of the circuit with respect to timing and defines a phase, cycle and period (note the units are seconds).

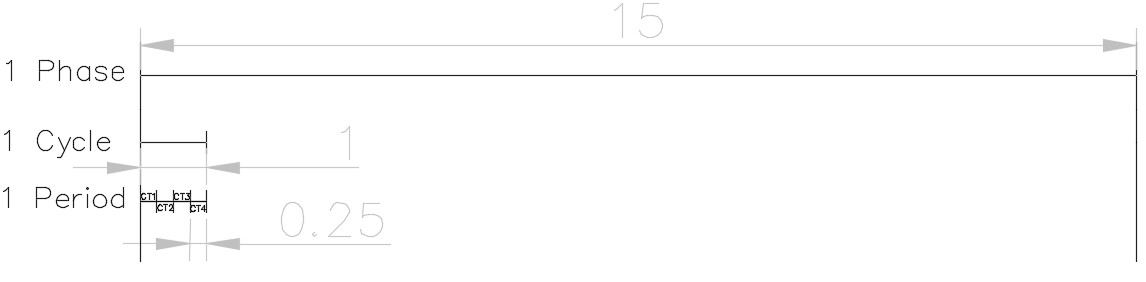


Figure 15 - Definition of Timing in Circuit (seconds)

Figure 15 shows there were three loops (timing sessions) occurring; a phase, a cycle, and a period. The program was sent into a wait for interrupt phase on start-up while enabling Timer 1 which was a 15 second timer that timed a phase. This would interrupt and disable itself starting Timer 2. Timer 2 would interrupt itself every 1 ms and on each interrupt a piece of data would be captured. After 250 ms of data acquisitions a switch statement would turn on another channel until all channels had data recorded. The switch statement would turn on a single channel at a time. A minimum value was set up that if a channel did not read above a (0.05 A) minimum value then it was disabled for 40 phases which was 10 minutes.

### Data Percentage

The values for each channel were sent to a Putty GUI (Figure 32) to verify when testing and display the state of each channel. This was important during the calibration phase and a clipping range was introduced. This clipping function occurred if the input ranges of the ADC were above 3.3-0 V which translated to 16-bit number of 65,535-0, while a buffer of 100 was placed on each side shifting it to 65,435-100. It was decided that it was easier to map this to a percentage which could be calibrated to the type of sensor (10A, 20A or 25A) during setup. This removed the necessary gain adjustment.

### Data Calculation & Mapping

The dual voltage from the input CT’s was shifted to a positive waveform using an op-amp circuit described and the effect on the waveform is shown in Figure 16. The input voltage from the CT (blue) is shown to be shifted from the output of the op-map circuit (green).

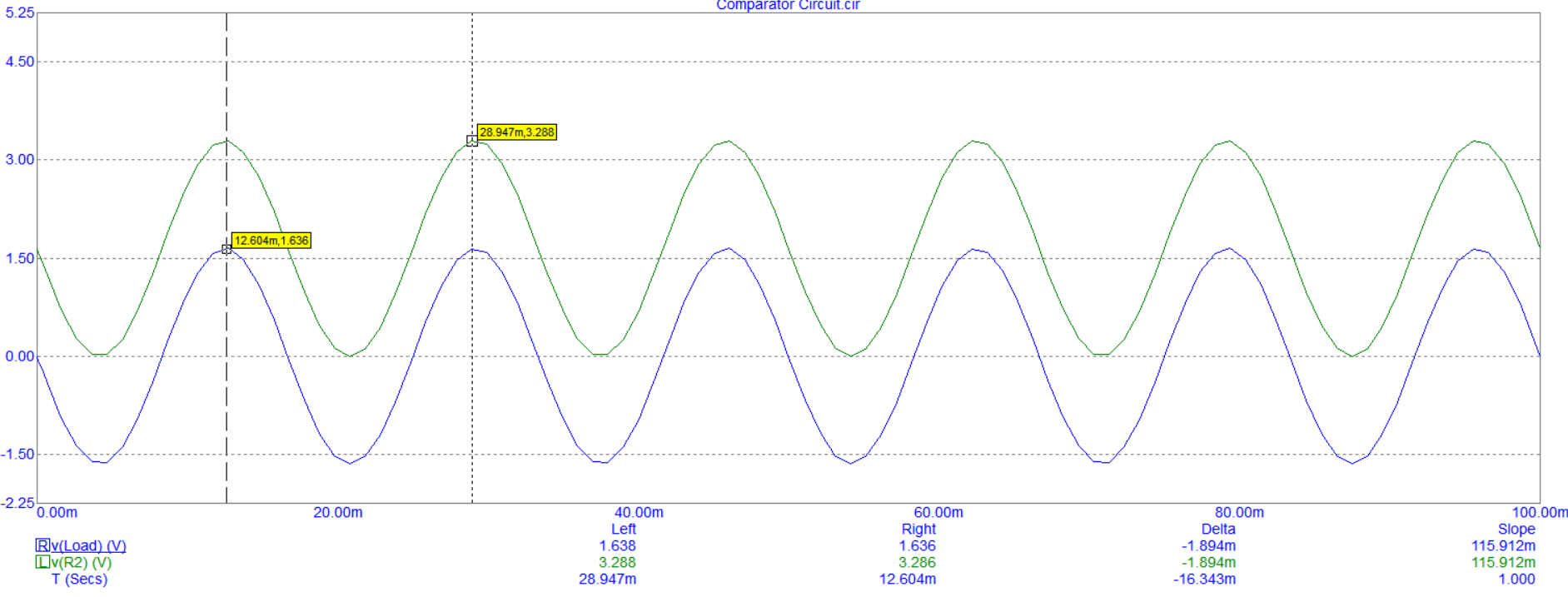


Figure 16 - Shifted Waveform from CT to ADC Input

Data was collected at each millisecond for 250 samples and stored into an array. The centre of each sample was calculated by basic averaging. The Root Mean Square (RMS) of voltage in an alternating current (AC) circuit is defined as a common mathematical method of finding the effective voltage compared to a direct current (DC) system. The voltage of the system can be used to determine the RMS as found in equation (2).

(2)

The voltage which was fed into the ADC was between 3.3V and 0V, which is equivalent to using the equation (3).

(3)

The mapping of the data went through four distinct stages and shown in Figure 17. It depicts the peak-to-peak voltage from the CT, being shifted to a positive voltage, being converted to a 16-bit number and the internal mapping to corresponding amperage.

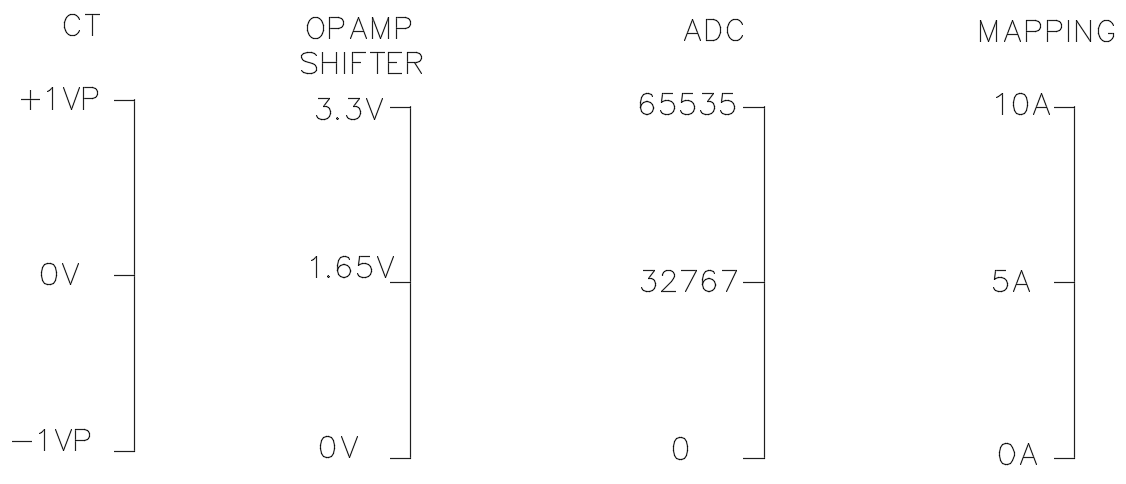


Figure 17 - Mapping Layout of Data

### ZigBee Transmission

The transmission of the data over the XBee module was done by placing the data into an array and using the Asynchroserial function to send one character at a time. The code shows the current sensor values placed into the buffer named message with the “*strlen"* function assigning the message size which was important before the API structures could be constructed.

**static** **char** message [100];

**snprintf**(message,100,"%f,%f,%f,%f\n",CT\_Current[0],CT\_Current[1],CT\_Current[2],CT\_Current[3]);

**int** message\_size = **strlen**(message);

Xbees provide a mode called Application Programming Interface (API) which provides users with a structured interface. The data is communicated through the serial interface and is organised into packets pre-set order. Data transmitted in the form of API packets or data frames have a very well-defined structure and understanding this structure is crucial to derive data from the frame.

Table 2 - API Fame Structure

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Start Delimiter | Length | | Frame Data | | | | | | | | Checksum |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | … | n | n+1 |
| 0x7E | MSB | LSB | API-specific structure | | | | | | | | Single byte |

An API frame has the structure as shown in Table 2. The specific structures can be found by connecting the Xbee to the XCTU program whilst the specific data is being sent. Figure 18 shows the API structures used in the CTM device.

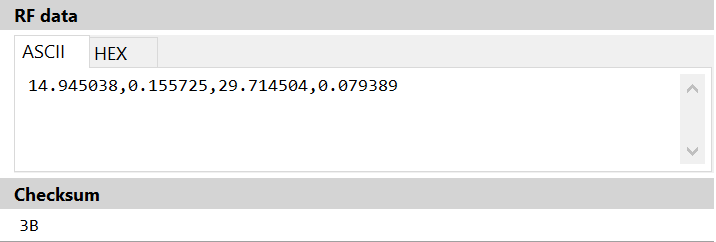


Figure 18 - Frame Details

This information was used to write the code in Kinetis to construct the XBee API packet as seen in Figure 19.

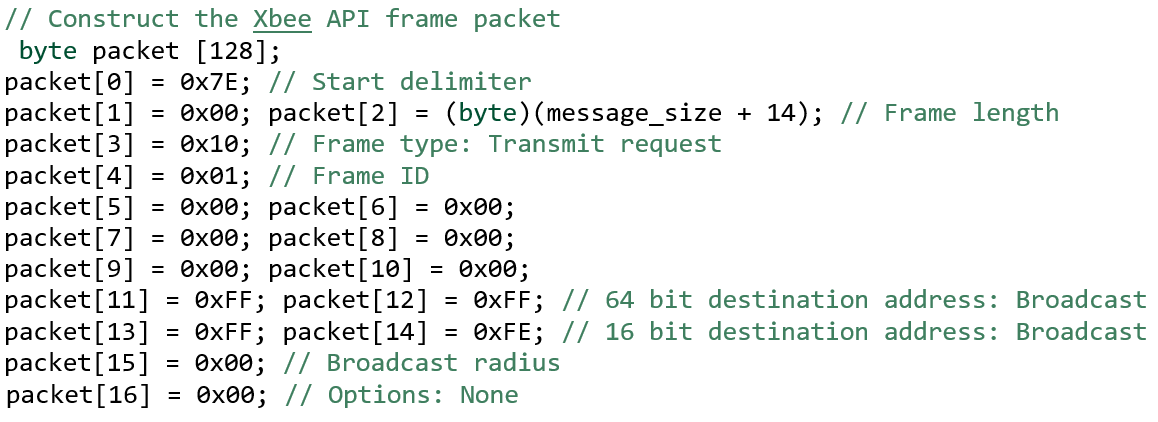


Figure 19 - Xbee API Packet

The message can then be placed into the frame packet once the API structures are established as depicted in Figure 20.

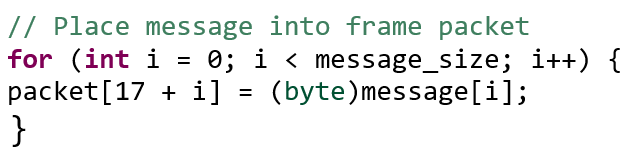


Figure 20 - Creation of Frame Packet

The Checksum is the last byte of the frame and helps test the data integrity. To calculate the checksum of an API frame all the bytes of the packet excluding the start delimiter and length are added. The lowest 8 bits only are kept from this result and this quantity is subtracted from 0xFF. If the checksum is incorrect frames sent through the serial interface will never be processed. The Checksum calculation and the data being sent using the asynchroserial function is shown in Figure 21.

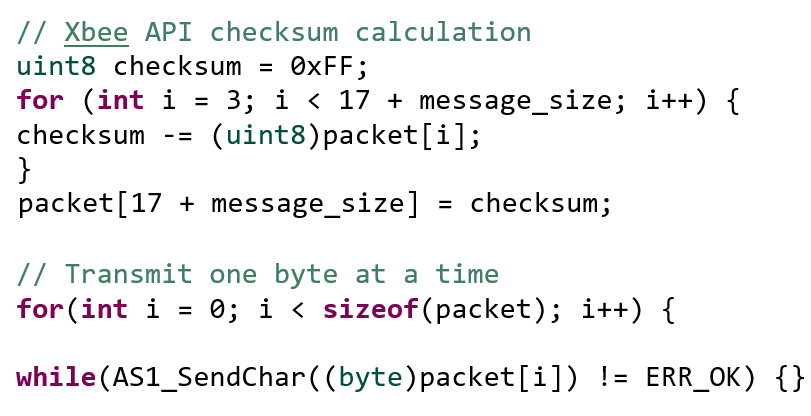


Figure 21 - Checksum and AS Transmission

### Calibration

The system was calibrated using known currents and developing a scaling factor. Several factors affected the CT% such as supply voltage, POT adjustment, and current transformer tolerances. The scaling factor for the CT was set at 0.1603.

|  |  |  |
| --- | --- | --- |
| Verifed Amps | CT % Avg | Amp/CT%Avg |
| 0.2 | 1.368 | 0.14619883 |
| 0.2 | 1.282 | 0.15600624 |
| 0.5 | 3.338 | 0.149790294 |
| 0.5 | 3.322 | 0.15051174 |
| 8.8 | 48.808 | 0.180298312 |
| 8.8 | 49.044 | 0.179430715 |
| Scaling Factor |  | 0.160372689 |

### Boot Loading

The SDA processor which we were going to use to convert the signals into serial for reading on the main processor had to be boot loaded. This is accomplished by including a button which is held down during connection to the computer. Windows then opens an explorer window on the thumb drive called “BOOTLOADER”. The new firmware has to be downloaded from pemicro.com/opensda. In our case it was the “MSD-DEBUG-FRDM-K20D50M\_Pemicro\_v118.SDA”.

## C++ Code

### ZigBee Data Acquisition

The receiving of the data packets on the raspberry pi required the use of unions and structures of the C++ programming language. A union in C++ programming is a user defined variable which may hold members of different sizes and types which all members share the same memory location. A structure is a convenient tool for handling a group of logically related data items. Structures help to organise complex data in an effective way. The following code was implemented to handle the XBee protocol (Figure 22).

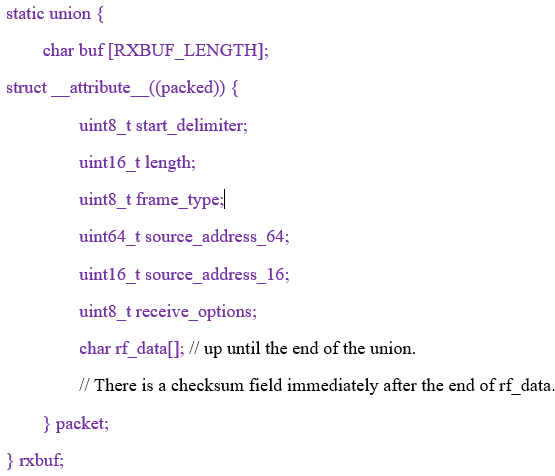


Figure 22 - Receiving Code from Zigbee

The incoming bytes are placed into “rxbuf.buf[]”. The struct “rxbuf.packet” allows for named access to particular fields within the binary protocol. The packet receive function was written with several “if” statements that read from the serial port into “rxbuf” until a complete packet has been received. It returns the length of the data payload or -1 if receive failed. Before the loop of the function was run, the buffer was zeroed and an index was initiated to 0.

memset(&rxbuf, 0, sizeof(rxbuf));

rxbuf\_idx = 0;

The API frame structure is known, therefore, within code there are checks, as we are expecting the start delimiter, the following was written:

*if ((rxbuf\_idx == 0) && (c != 0x7E)) {*

This *checks* the data to see if the first byte is the expected start delimiter. If this does not occur, we are not synchronised with the Xbee and we discard bytes by restarting this loop body until we see a start of frame delimiter. If the start delimiter is received, then the characters are saved into the buffer. Once “rxbuf\_idx” is 3, this acknowledges that we have received the length of the packet. The received packet length is in big endian format. Endianness refers to the sequential order in which bytes are arranged into larger numerical values, when stored in computer memory, or when transmitted over digital links.

*rxbuf.packet.length = be16toh(rxbuf.packet.length);*

When “rxbuf\_idx” is greater or equal to 4 this indicates that the number of bytes received is “rxbuf\_idx”, this is due to 4 bytes not being counted in length which are the start delimiter, the 2 bytes of length and checksum.

} else if (rxbuf\_idx >= 4) {

The complete packet is received when we have the length plus 4 bytes.

if (rxbuf\_idx >= rxbuf.packet.length+4) {

The function performs a number of checks including the check sum calculation, to confirm it is indeed the required data, the function returns the data by subtracting 12 off the length as there are 12 bytes of header included in packet.length.

return rxbuf.packet.length - 12;

### Wi-Pi Upload

To connect the Raspberry Pi to the internet, the Wi-Pi module needed to be configured. The network configurations were modified in the command line editor to set up a wi-fi connection on the JCU/Android/Home network. Transmitting the logged data is by done by sending specially crafted HTTP requests. A simple HTTP library is called “libcurl” and was installed on the Pi. Libcurl is a free and easy-to-use client-side URL transfer library, it is a computer software project providing a library and command-line tool for transferring data using various protocols (Figure 23).

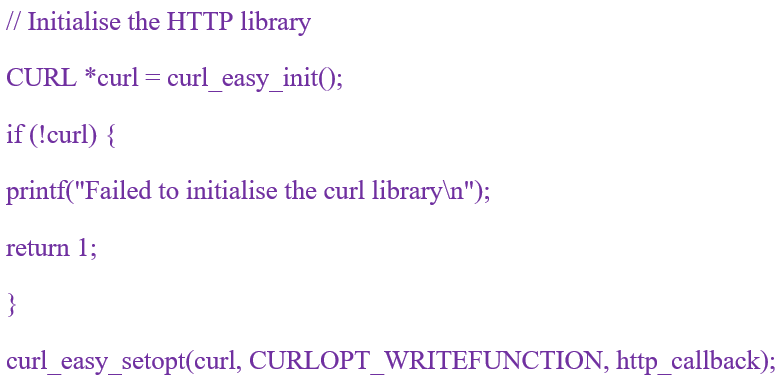


Figure 23 - Libcurl Code

### ThingSpeak Display

The data stream was created by accessing the Thingspeak website, after signing up for a free account, and creating a new channel at *thingspeak.com/channels.* Four independent data fields were specified as this corresponded to the four sensors on the CTM. Once the channels were created the writing API key (yellow in Figure 24) was found and copied into the code. The writing key is required to upload data to the channel, it provides a URL to use in the code.

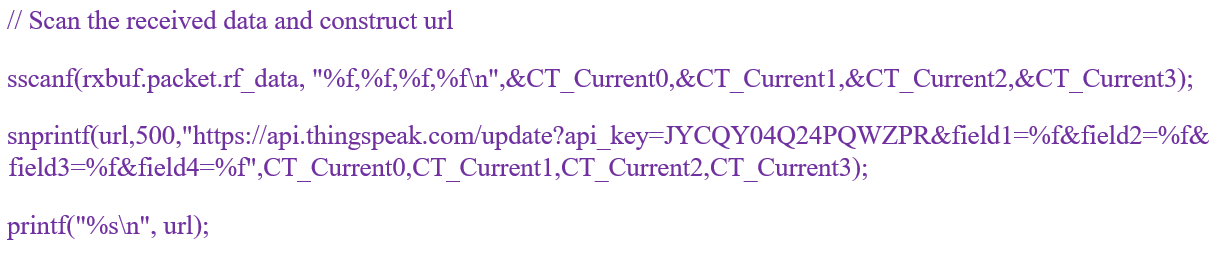


Figure 24 - Transmission with API Key (yellow) to ThingSpeak

## Webpage Code

### Hosting

The domain name was selected as wwww.ctm-3501.com and was purchased from an online source. The Zuver hosting source is shown in Figure 25 and is where the website is hosted.

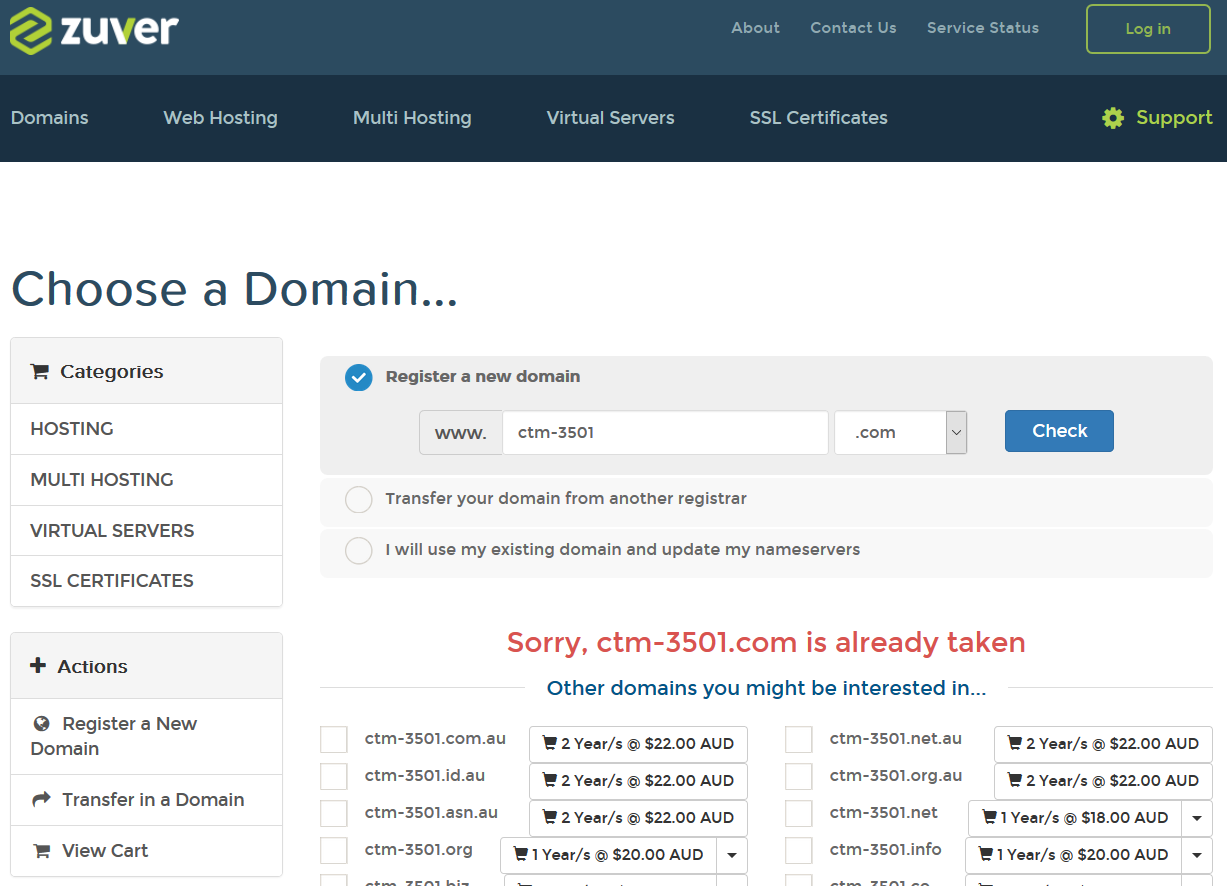


Figure 25 - Hosting Website

### Hyper Text Mark-up Language (HTML)

The webpage template was compiled from online research, downloadable bootstrap templates and the starting code is shown in Figure 26 (full code in the Appendix F).

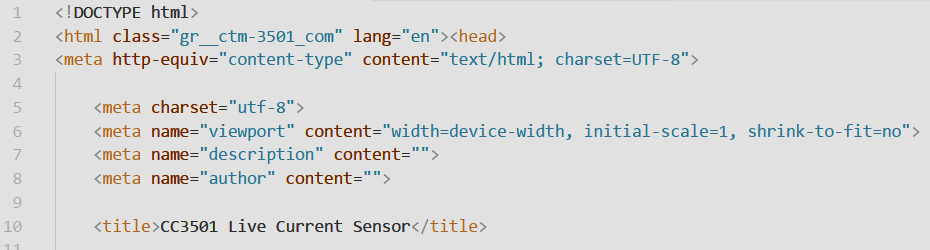


Figure 26 - HTML Initial Setup

### Cascading Style Sheet (CSS)

The fonts for the webpage were created to keep information in the proper display format. The background proved difficult to manage and both the z-index and height were changed as shown in Figure 27.

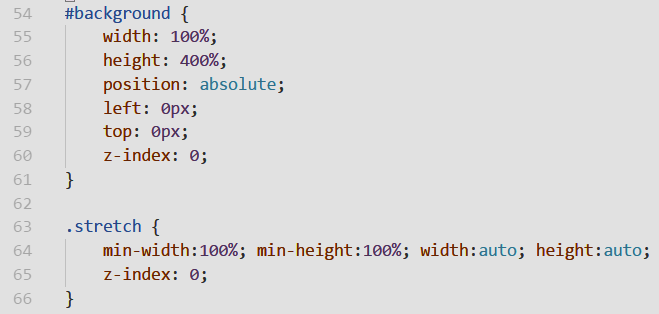


Figure 27 - Background Style Sheet

### File Transfer Protocol (FTP)

FTP is used to transfer files between computers on a network. FTP is used to exchange files between computer accounts, transfer files between an account and a desktop computer, or access online software archives. FileZilla is a powerful and free software for transferring files over the Internet. FileZilla is a very popular FTP client and is used by webmasters from all over the world and was selected to transfer the HTML and CSS to the Zuver host account. The file transfer process to ctm-3501.com in FileZilla is shown in Figure 28.

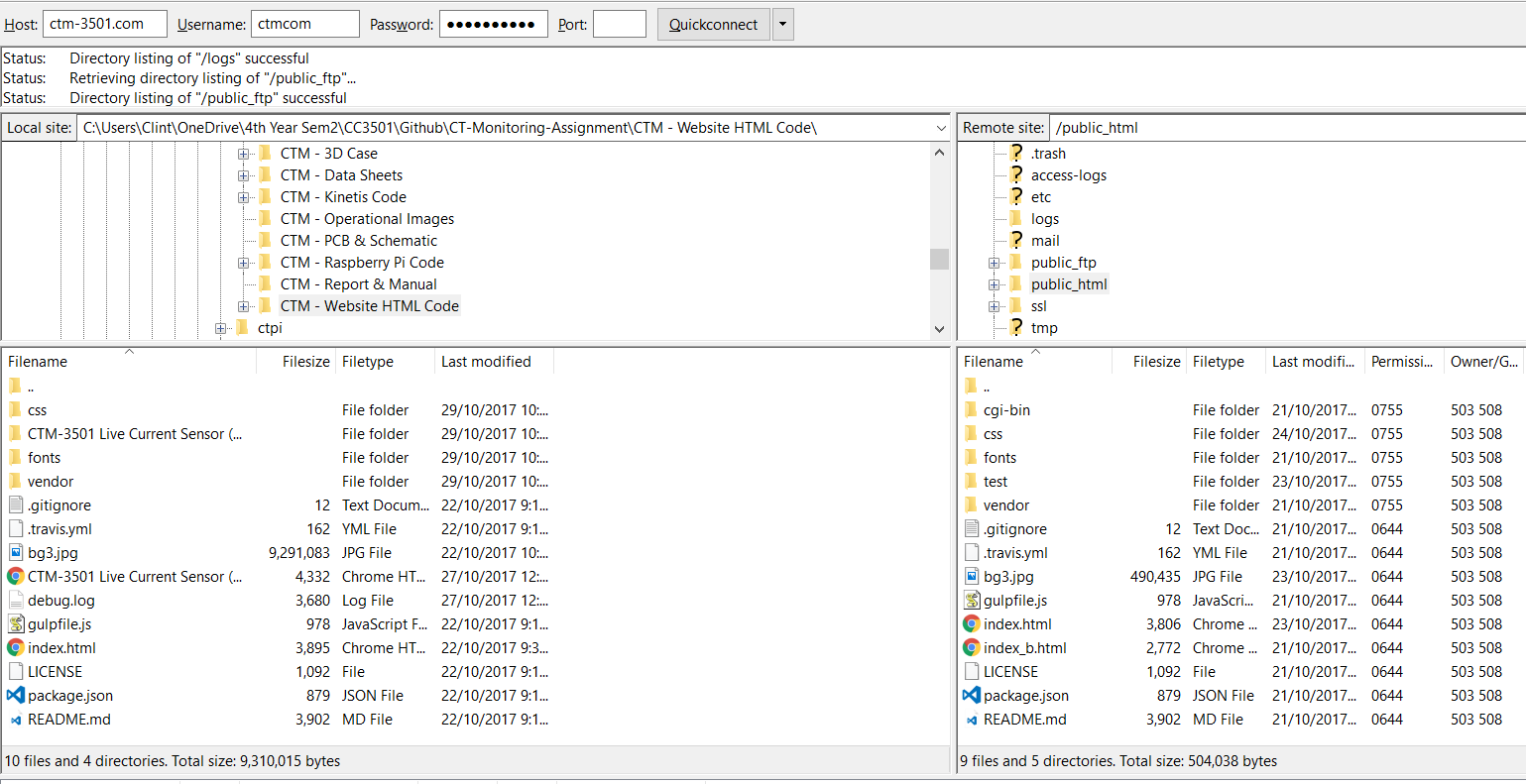


Figure 28 - Transferring Files to the Host

# TESTING & RESULTS

## Real-World Set Back

Due to unforeseen circumstances, the vicissitudes of fate shadowed the project and overheated our PCB while in the oven. The result is shown in Figure 29 and demonstrates that the high temperatures caused the silk screen layer to peel off and the solder paste to boil.

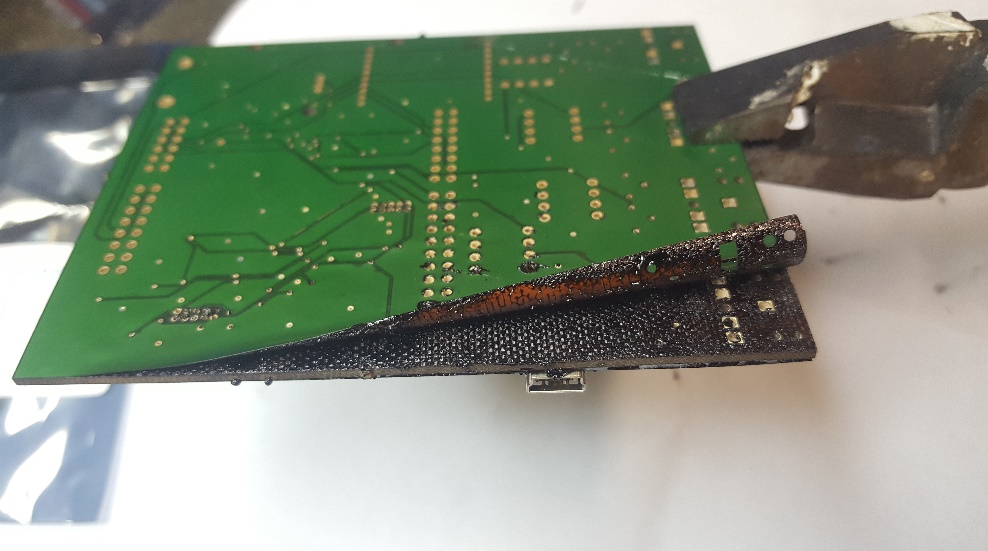


Figure 29 - Burnt PCB

There was another board which was printed but the bottom tracks were not. This was circumvented by manually soldering wires where these tracks should have been. This resulted in limiting the amount of CT channels to one, as shown in Figure 30.

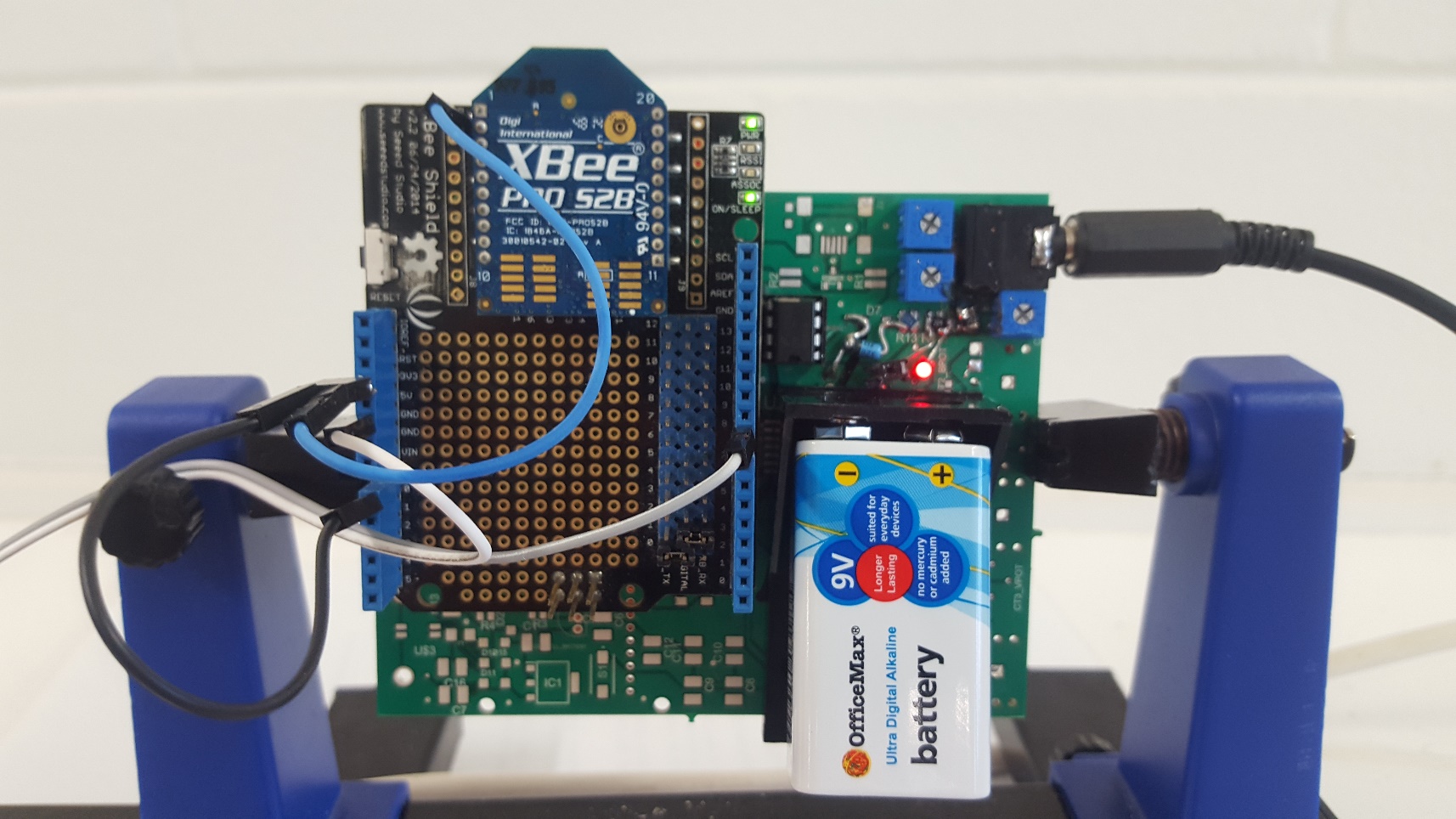


Figure 30 - Finished PCB and One CT Connected

## Final Product

A case was 3D printed for the PCB which allowed: connection of the CT circuits; allowed the ZigBee to transmit data unobscured; had provision for vents and cooling fan; and LED indication of when data was being transmitted.



Figure 31 - Final Product with Case and One CT Connected

## Webpage Display

The values were checked on Putty being sent and reception on the Pi and webpage.

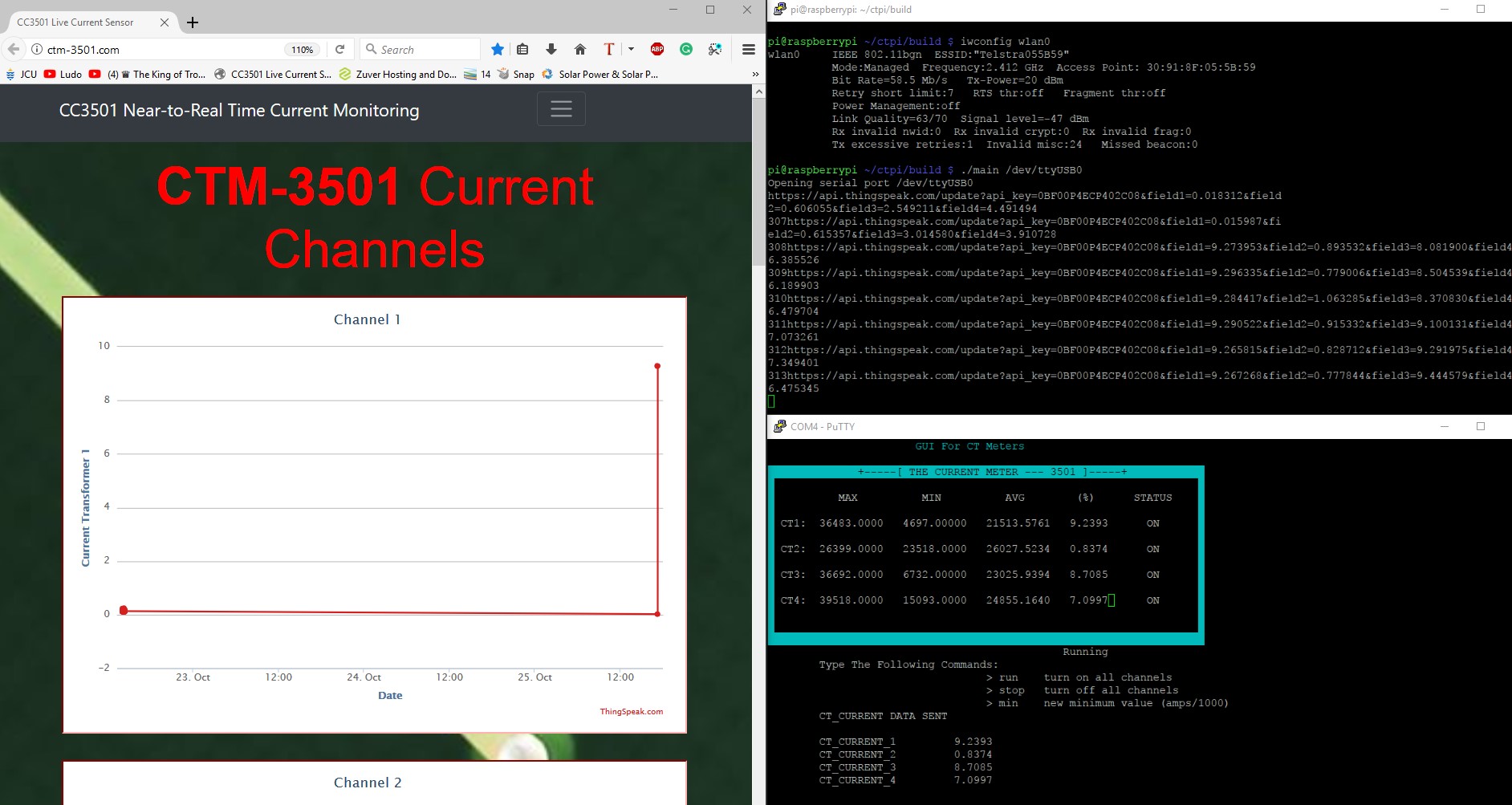


Figure 32 - The Checking Of Data Being Sent By CTM-3501 And Being Displayed On The Pi And Internet Webpage

## Game-Day Performance

The entire circuit including: 10 A CT, CTM-3501, ZigBees, Raspberry Pi, Wi-Pi, and webpage.

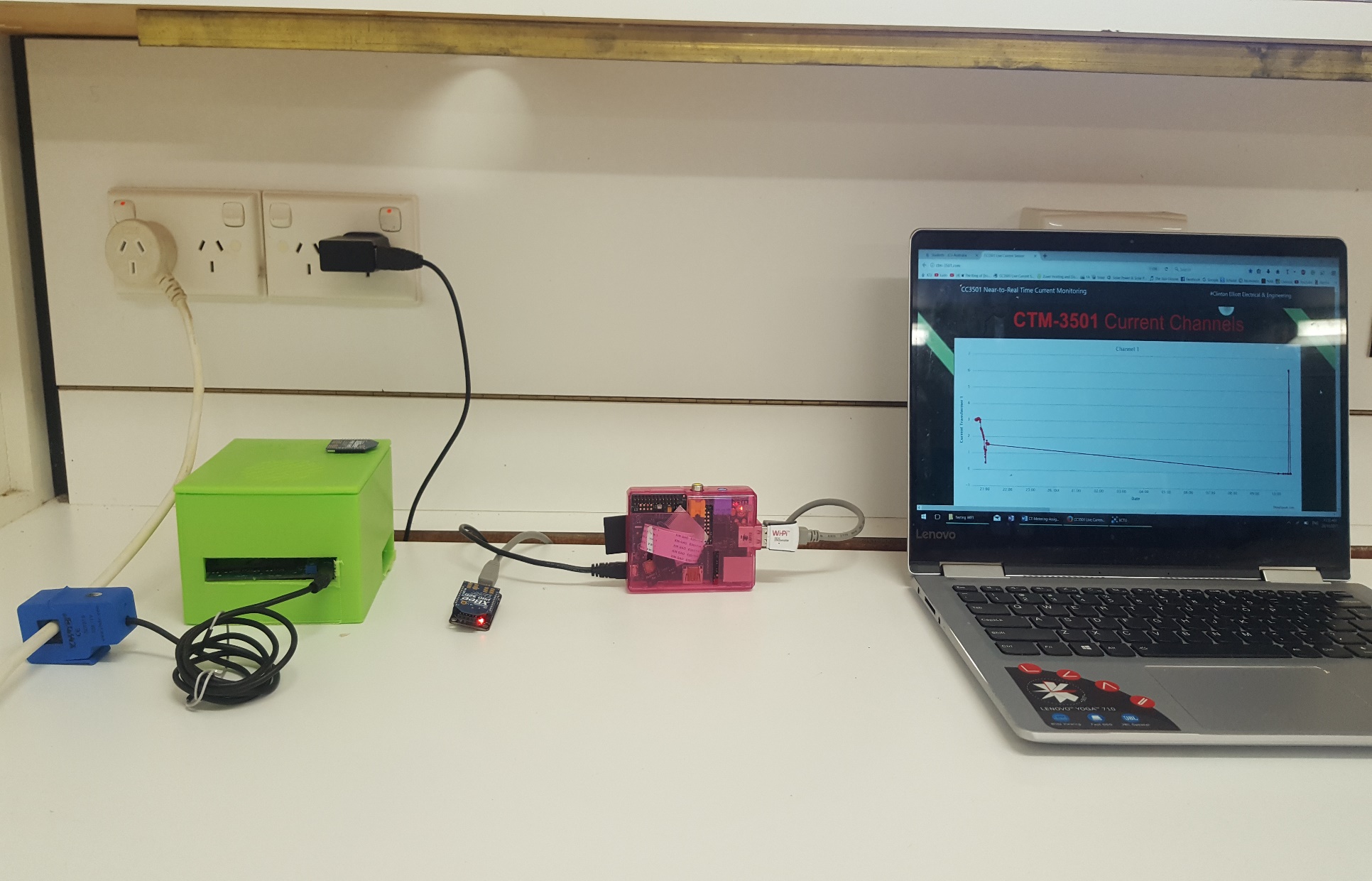


Figure 33 - The CTM-3501 Setup

## Results

The range of the CT which was installed was rated between 0-10 A. Therefore, a vacuum cleaner which drew approximately 6 A was perfect for testing. During the session two types of amp meters were used to verify the current drawn by the vacuum cleaner, which was between 6.2-6.4 A. The result from the CTM-3501 is shown in Figure 34.

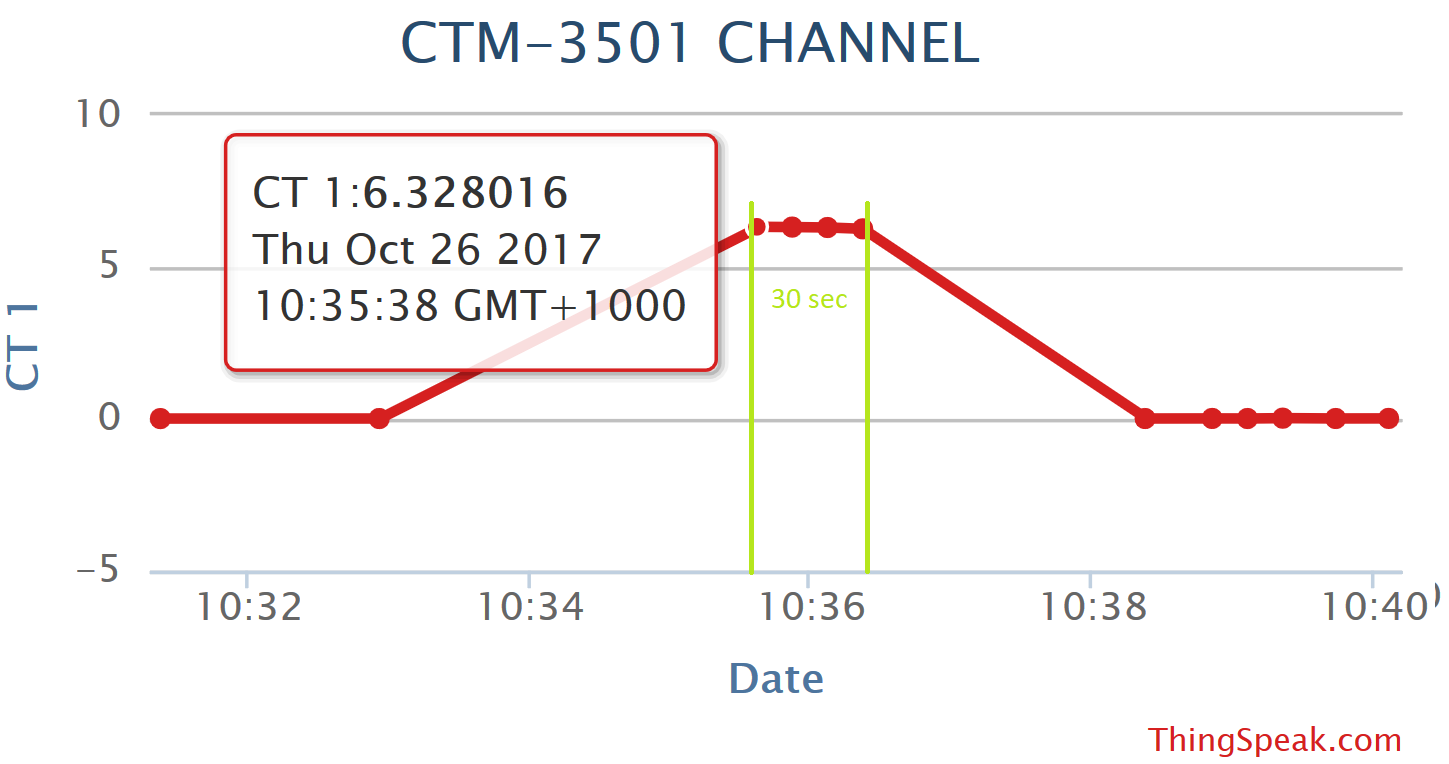


Figure 34 - Current Draw of Vacuum Cleaner

From Figure 34 it is clear that the test lasted approximately 30 seconds and the current draw was constant during this period. The value recorded was 6.32 A and was within the limits of the 6.2-6.4 A. This is approximately within 2 % error. It is worth noting the vacuum cleaner was operated for approximately 30 seconds, and this duration is shown between the bars (green) in Figure 34.

# LIMITATIONS & RECOMMENDATIONS

## Limitations

There were several limitations both of the design and process which are discussed below:

* The number of CT channels was limited to four. This could be increased depending on application. Although, this would not be possible to build in Eagle as the board dimensions are limited.
* The operation time for the board was calculated at 14 minutes, which is why power-saving strategies were implemented.
* The distance of effective transmission by the ZigBees was not tested, although a confirmed 20 metres used.
* The power dissipation of the CTM-3501 was not verified.
* The entire design (processors and four CTs) could not be tested, due to oven failure.
* The time for the PCBs to arrive caused the entire process to be delayed.
* Some parts were not ordered from Element14, and of which we were not notified, causing undue delay.
* Slow internet speeds affected uploading and caused lagging.
* The Raspberry Pi and Wi-Pi had to be started by exciting a script at the command line which is higher level knowledge and would prove difficult to a common user.
* The current measurement was limited to 10 A for testing purposes.
* Upload to ThinkSpeak was limited to 15 second intervals.
* It was found during calibration that if the 10 A rated limit of the CTs was passed then magnetic saturation of the coil occurred, resulting in non-linear current readings.

## Recommendations

There were several recommendations both of the design and process which are discussed below:

* It was noted that when switching both the multiplexer and op-amps at the same time, the data was not consistent, so this was circumvented by having the mux already powered.
* Wait-for-interrupts were used with timers in the Kinetis coding, while it would be considerably quicker if RTOS was utilised with MUTEXs and Semaphores.
* It is recommended to use a bigger board >(100mmx60mm) with larger components, e.g. 1210 packages, during the prototyping phase. This makes the entire process easier and in the event of de/soldering, it is especially helpful. We had to change many components to save space due to board size restrictions.
* The slow internet connection in Annandale (Townsville) affected uploading to the internet. It is recommended to put something into the code to prevent this affecting the design (clearing a buffer or time-out option).
* We only utilised one op-amp each on the dual op-amp packages. This was done for fault finding purposes, but the design could be minimised by removing two of the op-amps.
* Reducing the clock speed further would save power.
* Using a linear voltage regulator instead of the dissipation method is more practical.
* The first 3D box was printed with the red filament and did not work. It is recommended to make the words large on the box for printing purposes and at a slower speed.
* Build a GUI for the Pi or have an executable script which could connect to the internet and run the program. If at points the internet speed is slow, then it could empty the buffer.
* It is recommended that the calibration process for the CT channels is simplified.
* It is recommended to buy a larger processor, or have them pinned out on small sample PCBs - for prototyping.

# CONCLUSION

The aim of this project was to design and implement the CTM-3501. This report details the design, implementation, and verification. The CTM-3501 was implemented successfully using C language, C++, and HTML. During testing, it was successfully verified to have an error of <2 %. The design allowed a sensor node to operate analogue circuitry, biased to the midpoint of an ADC range, by a 9 V battery without connection to mains electricity. The worst-case current draw reduced standalone operation to 14 minutes, whilst this limitation was successfully circumvented using scavenging power-saving strategies. The CTM-3501 allowed wireless communication between the sensor node and the Raspberry Pi base station which allowed near-to-real time uploading to the CTM-3501 hosted webpage.

In conclusion, the CTM-3501 project was a very interesting multi-faceted design project. It allowed our group to explore aspects of battery life, micro processing, Linux language, web design, and communication protocols which we had not encountered previously. It was unfortunate that the ambition to replicate the FRDM-K20DX128M5 was not realised. Overall, this project was a success by both being as interesting as it was educational.

Thanks a lot for all your help during the semester Mostafa and Alex, it’s been a pleasure 😊

# REFERENCES

[1] E. Hub. (2017, 17/10*). Current Transformer*. Available: <http://www.electronicshub.org/current-transformer/>

[2] J. C. University, "CC3501 - Lecture Notes," 2017.

[3] Data sheets, “GitHub” – Various data sheets, 2017, https://github.com/clintonelliott23/CT-Monitoring-Assignment.

LIST OF APPENDICES

[APPENDIX A.1 – SCHEMATIC 37](#_Toc497138705)

[APPENDIX A.2 – SCHEMATIC 38](#_Toc497138706)

[APPENDIX B – PCB LAYOUT 39](#_Toc497138707)

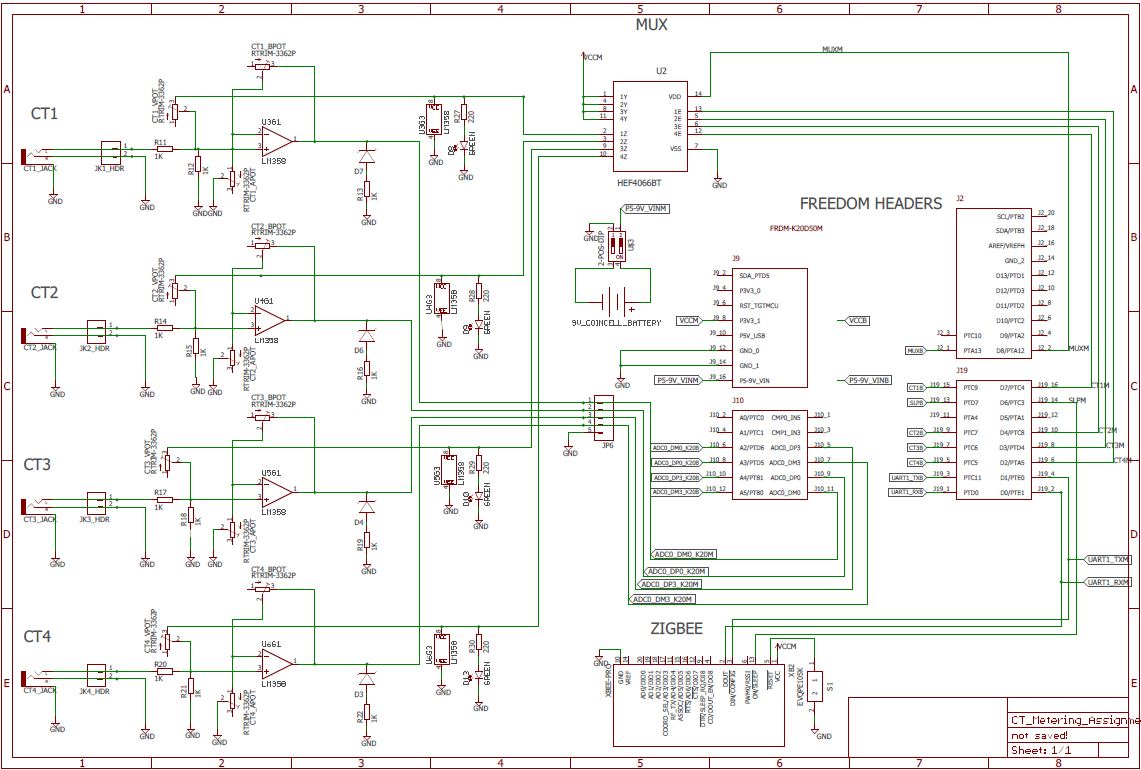
[APPENDIX C – BILL OF MATERIALS 40](#_Toc497138708)

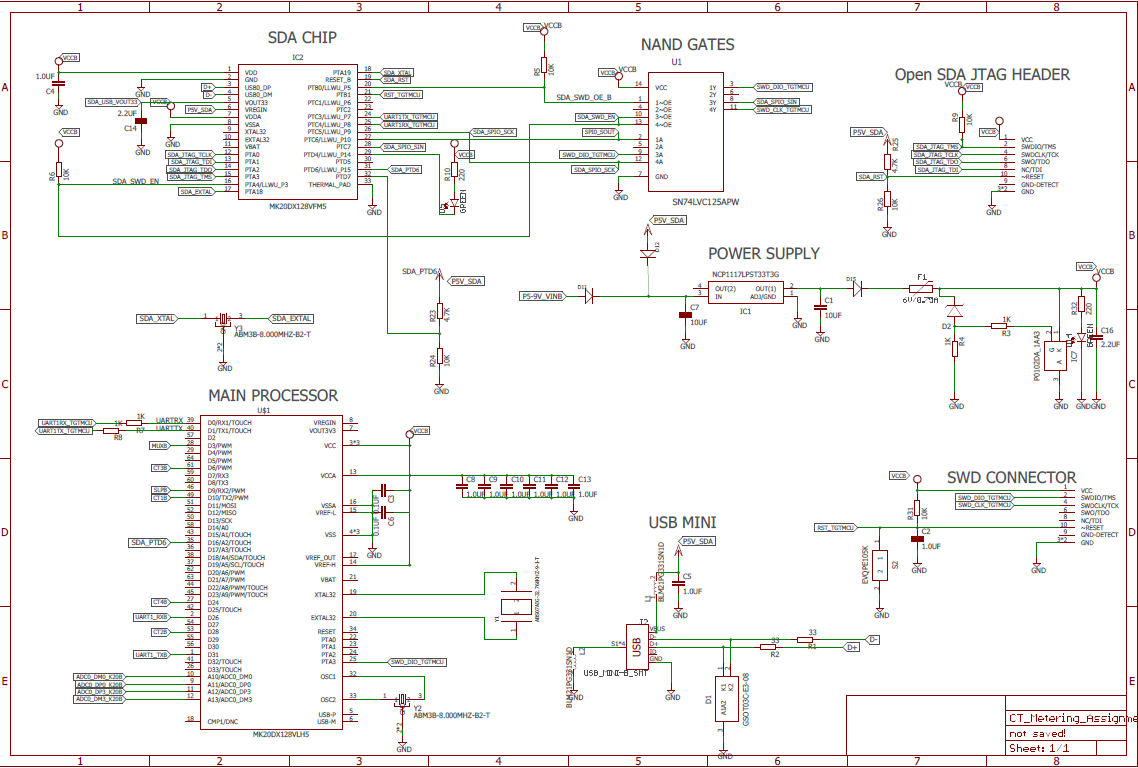
[APPENDIX D – KINETIS CODE 41](#_Toc497138709)

[APPENDIX E – RASPBERRY PI CODE 55](#_Toc497138710)

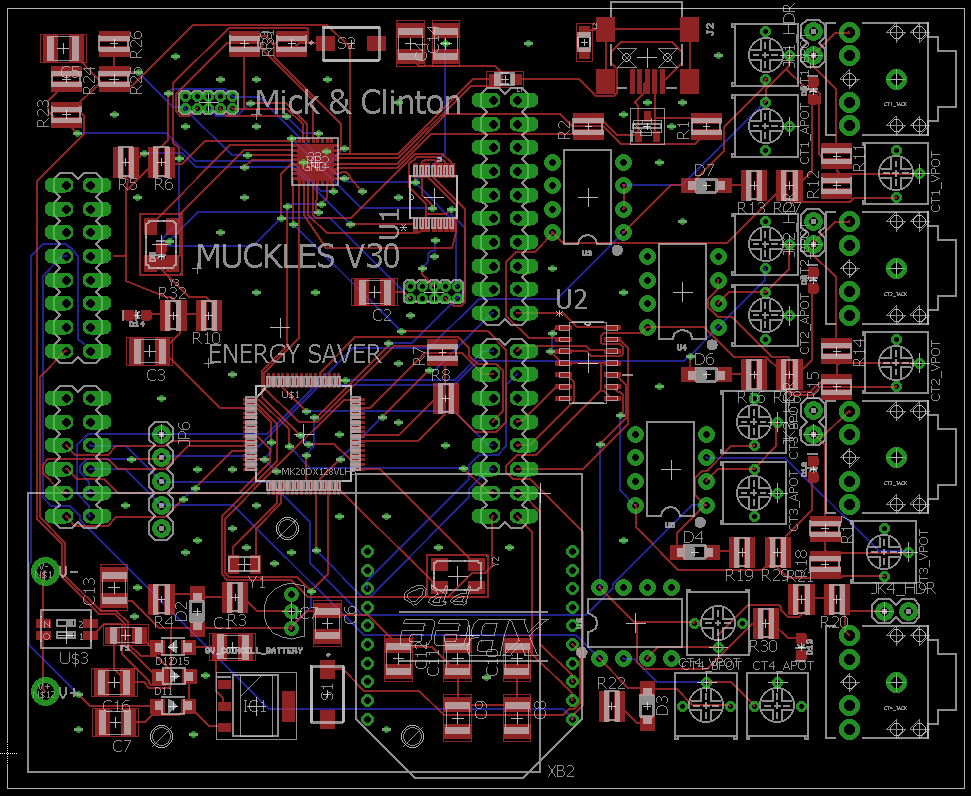
[APPENDIX F – WEBPAGE CODE 60](#_Toc497138711)

APPENDIX A.1 – SCHEMATIC



APPENDIX A.2 – SCHEMATIC

APPENDIX B – PCB LAYOUT



APPENDIX C – BILL OF MATERIALS

APPENDIX D – KINETIS CODE

/\* ###################################################################

;\*\* Filename : main.c

\*\* Project : Lab2

\*\* Processor : MK20DN128VLH5

\*\* Version : Driver 01.01

\*\* Compiler : GNU C Compiler

\*\* Date/Time : 2017-08-04, 12:23, # CodeGen: 0

\*\* Abstract :

\*\* Main module.

\*\* This module contains user's application code.

\*\* Settings :

\*\* Contents :

\*\* No public methods

\*\*

\*\* ###################################################################\*/

/\*!

\*\* @file main.c

\*\* @version 01.01

\*\* @brief

\*\* Main module.

\*\* This module contains user's application code.

\*/

/\*!

\*\* @addtogroup main\_module main module documentation

\*\* @{

\*/

/\* MODULE main \*/

/\* Including needed modules to compile this module/procedure \*/

**#include** "Cpu.h"

**#include** "Events.h"

**#include** "TU1.h"

**#include** "Term1.h"

**#include** "Inhr1.h"

**#include** "ASerialLdd1.h"

**#include** "TU2.h"

**#include** "AS1.h"

**#include** "ASerialLdd2.h"

**#include** "CsIO1.h"

**#include** "IO1.h"

**#include** "I2C.h"

**#include** "IntI2cLdd1.h"

**#include** "ADC.h"

**#include** "AdcLdd1.h"

**#include** "CT1\_BIT.h"

**#include** "MUXM.h"

**#include** "BitIoLdd2.h"

**#include** "TI1.h"

**#include** "TimerIntLdd1.h"

**#include** "TI2.h"

**#include** "TimerIntLdd2.h"

**#include** "CT1\_BIT.h"

**#include** "BitIoLdd3.h"

**#include** "SLP.h"

**#include** "BitIoLdd4.h"

**#include** "CT2\_BIT.h"

**#include** "BitIoLdd6.h"

**#include** "CT3\_BIT.h"

**#include** "BitIoLdd5.h"

**#include** "CT4\_BIT.h"

**#include** "BitIoLdd7.h"

/\* Including shared modules, which are used for whole project \*/

**#include** "PE\_Types.h"

**#include** "PE\_Error.h"

**#include** "PE\_Const.h"

**#include** "IO\_Map.h"

/\* User includes (#include below this line is not maintained by Processor Expert) \*/

////////////////////////////////// User Includes \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\/

**#include** <math.h>

**#include** <stdlib.h>

**#include** <stdio.h>

**#include** "string.h"

////////////////////////////////// Variables for Code \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\/

// Variables to be used over putty to recieve a message

**volatile** **char** buffer[100];

**volatile** **char** buffer2[100];

**volatile** **int** index;

**volatile** bool command\_recieved = 0;

**volatile** bool command\_sent = 0;

**volatile** bool hold = 0;

// Variables which may be altered

**#define** number\_samples 250 // Number of samples from the ADC / ms

**float** min\_val = 0.01; // Min value for the channels to DEACTIVATE

**short** **int** drop\_out = 3; // Amount of cycles the timer is DEACTIVATED

// Variables to receive data from the ADC and number of samples

**volatile** **int** sample\_index;

**unsigned** **short** CT\_raw\_values[4];

**unsigned** **short** CT1\_Raw[number\_samples];

**unsigned** **short** CT2\_Raw[number\_samples];

**unsigned** **short** CT3\_Raw[number\_samples];

**unsigned** **short** CT4\_Raw[number\_samples];

uint16 ADC\_measure;

// Variables for RMS Current

**float** CT\_Current [4];

/\*float CT2\_Current = 0;

float CT3\_Current = 0;

float CT4\_Current = 0;\*/

// Variables for Centre

**float** CT1\_Centre = 32767;

**float** CT2\_Centre = 32767;

**float** CT3\_Centre = 32767;

**float** CT4\_Centre = 32767;

//Variables for Min and Max

**float** max; **float** min;

**float** CT1\_max; **float** CT1\_min;

**float** CT2\_max; **float** CT2\_min;

**float** CT3\_max; **float** CT3\_min;

**float** CT4\_max; **float** CT4\_min;

**volatile** bool gain\_adjust = 0;

**float** range = 0;

//Variables to flag interrupts

**volatile** bool timer1\_interrupted = 0;

**volatile** bool timer2\_interrupted = 0;

// Variable to make channels active with first activated

**volatile** bool ch1\_measure = 1;

**volatile** bool ch2\_measure = 0;

**volatile** bool ch3\_measure = 0;

**volatile** bool ch4\_measure = 0;

// Variables to disable the CT channels

**short** **int** CT\_Counter = 1;

uint16 min\_input;

**short** **int** disable1 = 0;

**short** **int** disable2 = 0;

**short** **int** disable3 = 0;

**short** **int** disable4 = 0;

**short** **int** reable\_counter1 = 0;

**short** **int** reable\_counter2 = 0;

**short** **int** reable\_counter3 = 0;

**short** **int** reable\_counter4 = 0;

// RMS function variables

**float** CT\_RMS;

**float** centre = 0;

**signed** **short** CT\_shifted = 0;

// Create values for the columns and rows in GUI

**int** c1 = 9; **int** r1 = 7;

**int** c11 = 22;

**int** c2 = 35; **int** r2 = 9;

**int** c3 = 48; **int** r3 = 11;

**int** c4 = 60; **int** r4 = 13;

//////////////////////////////////// Functions \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\/

//Delay function

**void** **Delay**(){

**for** (**int** i = 0; i <2000000; i++){}

}

**void** **drawGUI**(**void**) {// Function: drawing the initial GUI

//start up check and message

Term1\_Cls(); Term1\_MoveTo(10,10);

Term1\_SendStr("I Current Wait.........!!!");Term1\_CursorDown(1);

Delay();

// Create nice terminal window

Term1\_Cls(); // Clear Terminal

// Draw title top left

Term1\_MoveTo(c11+2, 1); Term1\_SetColor(clCyan, clBlack);

Term1\_SendStr("GUI For CT Meters");

// Set boarder colour

Term1\_SetColor(clBlack, clCyan);

// Draw crane settings title

Term1\_MoveTo(1, 3);

Term1\_SendStr(" +-----[ THE CURRENT METER --- 3501 ]-----+ ");

// Draw two border columns

**for** (**int** i = 4; i <= 16; i++) {

// Draw left column of first box

Term1\_MoveTo(1, i); Term1\_SendStr(" ");

// Draw right column of first box

Term1\_MoveTo(68, i); Term1\_SendStr(" ");

}

// Draw bottom Row

**for** (**int** i = 1; i <= 68; i++) {

Term1\_MoveTo(i, 16); Term1\_SendStr(" ");

}

// Write all information and categories

Term1\_SetColor(clWhite, clBlack);

Term1\_MoveTo(c1, 18); Term1\_SendStr("Type The Following Commands:");

Term1\_MoveTo(c2, 19); Term1\_SendStr("> run turn on all channels");

Term1\_MoveTo(c2, 20); Term1\_SendStr("> stop turn off all channels");

Term1\_MoveTo(c2, 21); Term1\_SendStr("> min new minimum value (amps/1000)");

Term1\_MoveTo(47,17); Term1\_SendStr("Running");

Term1\_MoveTo(c1+3,5); Term1\_SendStr("MAX");

Term1\_MoveTo(c11+3,5); Term1\_SendStr("MIN");

Term1\_MoveTo(c2+3,5); Term1\_SendStr("AVG");

Term1\_MoveTo(c3+1,5); Term1\_SendStr("(%)");

Term1\_MoveTo(c4-2,5); Term1\_SendStr("STATUS");

Term1\_MoveTo(3,r1); Term1\_SendStr("CT1:");

Term1\_MoveTo(3,r2); Term1\_SendStr("CT2:");

Term1\_MoveTo(3,r3); Term1\_SendStr("CT3:");

Term1\_MoveTo(3,r4); Term1\_SendStr("CT4:");

Term1\_MoveTo(c4,r1); Term1\_SendStr("ON");

Term1\_MoveTo(c4,r2); Term1\_SendStr("ON");

Term1\_MoveTo(c4,r3); Term1\_SendStr("ON");

Term1\_MoveTo(c4,r4); Term1\_SendStr("ON");

}

// Command reaction for coding for input to putty

**void** **CommandReact**(){

// Calculates if the input is valid and discerns outputs

**if** (0 == **strcmp**(buffer, "stop")) {

Term1\_MoveTo(1,17); Term1\_EraseLine();

Term1\_MoveTo(47,17); Term1\_SendStr("STOPPED!");Term1\_MoveTo(1,17);

CT1\_BIT\_PutVal(0);

CT2\_BIT\_PutVal(0);

CT3\_BIT\_PutVal(0);

CT4\_BIT\_PutVal(0);

MUXM\_PutVal(0);

hold = 1;

} **else** **if** (0 == **strcmp**(buffer,"run")){

Term1\_MoveTo(1,17); Term1\_EraseLine();

Term1\_MoveTo(47,17); Term1\_SendStr("RUNNING...");Term1\_MoveTo(1,17);

hold = 0;

} **else** **if** (**sscanf**((**char** \*)buffer, "min %hu", &min\_input)){

min\_val = (**float**)(min\_input);

min\_val = min\_val/1000;

Term1\_MoveTo(1,17); Term1\_EraseLine();

Term1\_MoveTo(47,17); Term1\_SendStr("Min Val = ");Term1\_SendFloatNum(min\_val);Term1\_MoveTo(1,17);

} **else** {

Term1\_MoveTo(1,17); Term1\_EraseLine();

Term1\_MoveTo(47,17); Term1\_SendStr("Doesn't make sense bra!");Term1\_MoveTo(1,17);

Delay();

Term1\_MoveTo(1,17); Term1\_EraseLine();

hold = 0; } //endif

command\_sent = 0; //Reset flag

TI1\_EnableEvent();

}//end command react

// RMS Function

**float** **RMS\_calculator**(**unsigned** **short** \*CT\_data){

// Reset variables

centre = 0;

range = 0;

**int** avg\_total = 0;

**int** squared = 0;

// find the centre using averages

**for** (**int** i = 0; i < number\_samples; i++) {

centre = centre + CT\_data[i];

}//end-for

// set centre point for min and max and calculate centre

centre = centre/number\_samples;

max = centre;

min = centre;

//Calculates the RMS using squared, sum and square-root

**for** (**int** i = 0; i < number\_samples; i++) {

CT\_shifted = CT\_data[i]-centre;

squared = (CT\_shifted\*CT\_shifted)/number\_samples;

avg\_total = avg\_total + squared;

// Find the min and max of the adc channel

**if** (CT\_data[i]< min){

min = (**float**)CT\_data[i];

} **else** **if** (CT\_data[i]> max){

max = (**float**)CT\_data[i];

}

}//end-for

//Determine if Gain should be adjusted

range = max - min;

**if** ( range > (65500)){

gain\_adjust = 1;

} **else** **if** ((max > 65534)||(min < 1)){

gain\_adjust = 1;

} **else** {

gain\_adjust = 0;

}

// CT\_RMS = (((sqrt((float)range)\*100)/65535)\*30.656);

CT\_RMS = (((((**float**)range\*100)/65530))\*0.164908\*(6.55/7.86));

**return** CT\_RMS;

}//end-function

// updates all the values for putty so it can be removed to testing

**void** **Update\_Putty\_CT\_Values**(){

//CT1 Values

Term1\_MoveTo(c1,r1); Term1\_SendFloatNum(CT1\_max);

Term1\_MoveTo(c11,r1); Term1\_SendFloatNum(CT1\_min);

Term1\_MoveTo(c2,r1); Term1\_SendFloatNum(CT1\_Centre);

Term1\_MoveTo(c3,r1); Term1\_SendFloatNum(CT\_Current[0]);

//CT2 Values

Term1\_MoveTo(c1,r2); Term1\_SendFloatNum(CT2\_max);

Term1\_MoveTo(c11,r2); Term1\_SendFloatNum(CT2\_min);

Term1\_MoveTo(c2,r2); Term1\_SendFloatNum(CT2\_Centre);

Term1\_MoveTo(c3,r2); Term1\_SendFloatNum(CT\_Current[1]);

//CT3 Values

Term1\_MoveTo(c1,r3); Term1\_SendFloatNum(CT3\_max);

Term1\_MoveTo(c11,r3); Term1\_SendFloatNum(CT3\_min);

Term1\_MoveTo(c2,r3); Term1\_SendFloatNum(CT3\_Centre);

Term1\_MoveTo(c3,r3); Term1\_SendFloatNum(CT\_Current[2]);

//CT4 Values

Term1\_MoveTo(c1,r4); Term1\_SendFloatNum(CT4\_max);

Term1\_MoveTo(c11,r4); Term1\_SendFloatNum(CT4\_min);

Term1\_MoveTo(c2,r4); Term1\_SendFloatNum(CT4\_Centre);

Term1\_MoveTo(c3,r4); Term1\_SendFloatNum(CT\_Current[3]);

}

/////////////////////////////////////// Main User Code \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\/

/\*lint -save -e970 Disable MISRA rule (6.3) checking. \*/

**int** **main**(**void**)

/\*lint -restore Enable MISRA rule (6.3) checking. \*/

{ /\* Write your local variable definition here \*/

/\*\*\* Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! \*\*\*/

PE\_low\_level\_init();

/\*\*\* End of Processor Expert internal initialization. \*\*\*/

///////////////////////////////////// One Shot Operations \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\/

//Disable the timers so they do not interrupt startup operations

TI1\_Disable();

TI2\_Disable();

// Draws the terminal window

drawGUI(); //uses functions

Update\_Putty\_CT\_Values(); // set initial values

// Calibrate the ADC

ADC\_Calibrate(TRUE);

// Turn off all bits

CT1\_BIT\_PutVal(0); //ct1

CT2\_BIT\_PutVal(0); //ct2

CT3\_BIT\_PutVal(0); //ct3

CT4\_BIT\_PutVal(0); //ct4

SLP\_PutVal(0); //zigbee

MUXM\_PutVal(0); //mux to switch opamps

// Start the Main Timer

TI1\_Enable(); //creates the first interrupt at 10 seconds

////////////////////////////////////// Primary FOR Loop \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\/

**for** (;;) { //for-loop-1

// Check for the timer 1 interrupt and nothing has been typed

**if**((timer1\_interrupted == 0 )&& (hold == 0));{

TI1\_EnableEvent();//enables events from time 1 to count 5 second intervals

**\_\_asm**("wfi"); //// wfi = "wait for interrupt" instruction puts the CPU in a low power state

}

// Check for timer1 interrupt and nothing has been typed into putty

**if** ((timer1\_interrupted == 1) && (hold == 0)) {//if-main

// This code creates a loop to count 250 samples of the sinewave

TI2\_Enable();//enable timer2

**while**(sample\_index < number\_samples){//collect the data samples at 1msec intervals

**if**(timer2\_interrupted == 1 ){// activates on the interrput

MUXM\_PutVal(1);//turn on mux in sampling loop

ADC\_Measure(TRUE);//gets a new measurement

// Each channel is activated depending on which needs to be measured

**if** ((ch1\_measure) && (disable1 == 0)) {

CT1\_BIT\_PutVal(1);//turn on ct op-amp for sampling

ADC\_GetChanValue16(3, &ADC\_measure);

CT1\_Raw[sample\_index] = ADC\_measure; // pin 1

} **else** **if** ((ch2\_measure) && (disable2 == 0)) {

CT2\_BIT\_PutVal(1);//turn on ct op-amp for sampling

ADC\_GetChanValue16(2, &ADC\_measure);

CT2\_Raw[sample\_index] = ADC\_measure; // pin 2

} **else** **if** ((ch3\_measure) && (disable3 == 0)) {

CT3\_BIT\_PutVal(1);//turn on ct op-amp for sampling

ADC\_GetChanValue16(0, &ADC\_measure);

CT3\_Raw[sample\_index] = ADC\_measure; // pin 3

} **else** **if** ((ch4\_measure) && (disable4 == 0)) {

CT4\_BIT\_PutVal(1);//turn on ct op-amp for sampling

ADC\_GetChanValue16(1, &ADC\_measure);

CT4\_Raw[sample\_index] = ADC\_measure; // pin 4

}//if2

sample\_index++;// increment the sample index until all samples have been taken

}//if1

timer2\_interrupted = 0;//sets the interrupt flag back to zero

//\_\_asm("wfi");// wfi = "wait for interrupt" instruction puts the CPU in a low power state

}//while loop

TI2\_Disable();//stop timer2 from interrupting (turn off)

sample\_index = 0;//set the sample index back to zero to count for the next channel measurements

// Out of data sampling loop, turn off and power to each opamp

CT1\_BIT\_PutVal(0);

CT2\_BIT\_PutVal(0);

CT3\_BIT\_PutVal(0);

CT4\_BIT\_PutVal(0);

// Switch statement toggles through each CT to turn on and measure, disable and send data to the screen

**switch** (CT\_Counter) {

**case** 1:

//Disables the CT if not needed

**if** (disable1 == 0){

CT\_Current[0] = RMS\_calculator(&CT1\_Raw);//calculates the RMS

CT1\_Centre = centre;

CT1\_min = min;

CT1\_max = max;

// initiate a reenable counter

reable\_counter1 = 0;

}

// Test to see if the channel is measuring or can be shut down

**if** ((CT\_RMS < min\_val) && (reable\_counter1 == 0)) {

disable1 = 1;

reable\_counter1 = 1;

Term1\_MoveTo(c4,r1);Term1\_SendStr("OFF ");

} **else** **if** (reable\_counter1 == drop\_out) {

disable1 = 0;

} **else** **if** (gain\_adjust == 1){

Term1\_MoveTo(c4,r1);Term1\_SendStr("CLIPPED");//for when the gain is clipping

} **else** {

reable\_counter1++;

Term1\_MoveTo(c4,r1);Term1\_SendStr("ON ");

}

//increment the counted to switch to next channel and turn on next channel

CT\_Counter++;

ch1\_measure = 0;

ch2\_measure = 1;

ch3\_measure = 0;

ch4\_measure = 0;

**break**;

**case** 2:

//Disables the CT if not needed

**if** (disable2 == 0){

CT\_Current[1] = RMS\_calculator(&CT2\_Raw);//calculates the RMS

CT2\_Centre = centre;

CT2\_min = min;

CT2\_max = max;

// initiate a reenable counter

reable\_counter2 = 0;

}

// Test to see if the channel is measuring or can be shut down

**if** ((CT\_RMS < min\_val) && (reable\_counter2 == 0)) {

disable2 = 1;

reable\_counter2 = 1;

Term1\_MoveTo(c4,r2);Term1\_SendStr("OFF ");

} **else** **if** (reable\_counter2 == drop\_out) {

disable2 = 0;

} **else** **if** (gain\_adjust == 1){

Term1\_MoveTo(c4,r2);Term1\_SendStr("CLIPPED");//for when the gain is clipping

} **else** {

reable\_counter2++;

Term1\_MoveTo(c4,r2);Term1\_SendStr("ON ");

}

//increment the counted to switch to next channel and turn on next channel

CT\_Counter++;

ch1\_measure = 0;

ch2\_measure = 0;

ch3\_measure = 1;

ch4\_measure = 0;

**break**;

**case** 3:

//Disables the CT if not needed

**if** (disable3 == 0){

CT\_Current[2] = RMS\_calculator(&CT3\_Raw);//calculates the RMS

CT3\_Centre = centre;

CT3\_min = min;

CT3\_max = max;

// initiate a reenable counter

reable\_counter3 = 0;

}

// Test to see if the channel is measuring or can be shut down

**if** ((CT\_RMS < min\_val) && (reable\_counter3 == 0)) {

disable3 = 1;

reable\_counter3 = 1;

Term1\_MoveTo(c4,r3);Term1\_SendStr("OFF ");

} **else** **if** (reable\_counter3 == drop\_out) {

disable3 = 0;

} **else** **if** (gain\_adjust == 1){

Term1\_MoveTo(c4,r3);Term1\_SendStr("CLIPPED");//for when the gain is clipping

} **else** {

reable\_counter3++;

Term1\_MoveTo(c4,r3);Term1\_SendStr("ON ");

}

//increment the counted to switch to next channel and turn on next channel

CT\_Counter++;

ch1\_measure = 0;

ch2\_measure = 0;

ch3\_measure = 0;

ch4\_measure = 1;

**break**;

**case** 4:

//Disables the CT if not needed

**if** (disable4 == 0){

CT\_Current[3] = RMS\_calculator(&CT4\_Raw);//calculates the RMS

CT4\_Centre = centre;

CT4\_min = min;

CT4\_max = max;

// initiate a reenable counter

reable\_counter4 = 0;

}

// Test to see if the channel is measuring or can be shut down

**if** ((CT\_RMS < min\_val) && (reable\_counter4 == 0)) {

disable4 = 1;

reable\_counter4 = 1;

Term1\_MoveTo(c4,r4);Term1\_SendStr("OFF ");

} **else** **if** (reable\_counter4 == drop\_out) {

disable4 = 0;

} **else** **if** (gain\_adjust == 1){

Term1\_MoveTo(c4,r4);Term1\_SendStr("CLIPPED");//for when the gain is clipping

} **else** {

reable\_counter4++;

Term1\_MoveTo(c4,r4);Term1\_SendStr("ON ");

}

//increment the counted to switch to next channel and turn on next channel

CT\_Counter = 1;

ch1\_measure = 1;

ch2\_measure = 0;

ch3\_measure = 0;

ch4\_measure = 0;

// Send Over the Serial

CT\_Current[2] = 6.666; //set to value for testing

CT\_Current[3] = 0;//set to value for testing

Term1\_MoveTo(c1, 22);Term1\_SendStr("CT\_CURRENT DATA SENT ");

Term1\_MoveTo(c1, 24);Term1\_SendStr("CT\_CURRENT\_1 ");Term1\_SendFloatNum(CT\_Current[0]);

Term1\_MoveTo(c1, 25);Term1\_SendStr("CT\_CURRENT\_2 ");Term1\_SendFloatNum(CT\_Current[1]);

Term1\_MoveTo(c1, 26);Term1\_SendStr("CT\_CURRENT\_3 ");Term1\_SendFloatNum(CT\_Current[2]);

Term1\_MoveTo(c1, 27);Term1\_SendStr("CT\_CURRENT\_4 ");Term1\_SendFloatNum(CT\_Current[3]);

**static** **char** message [100];

**snprintf**(message, 100, "%f,%f,%f,%f\n",CT\_Current [0],CT\_Current [1],CT\_Current [2],CT\_Current [3]);

**int** message\_size = **strlen**(message);

// Construct the Xbee API frame packet

byte packet [128];

packet[0] = 0x7E; // Start delimiter

packet[1] = 0x00; packet[2] = (byte)(message\_size + 14); // Frame length

packet[3] = 0x10; // Frame type: Transmit request

packet[4] = 0x01; // Frame ID

packet[5] = 0x00; packet[6] = 0x00;

packet[7] = 0x00; packet[8] = 0x00;

packet[9] = 0x00; packet[10] = 0x00;

packet[11] = 0xFF; packet[12] = 0xFF; // 64 bit destination address: Broadcast

packet[13] = 0xFF; packet[14] = 0xFE; // 16 bit destination address: Broadcast

packet[15] = 0x00; // Broadcast radius

packet[16] = 0x00; // Options: None

// Place message into frame packet

**for** (**int** i = 0; i < message\_size; i++) {

packet[17 + i] = (byte)message[i];

}

// Xbee API checksum calculation

uint8 checksum = 0xFF;

**for** (**int** i = 3; i < 17 + message\_size; i++) {

checksum -= (uint8)packet[i];

}

packet[17 + message\_size] = checksum;

// Transmit one byte at a time

**for**(**int** i = 0; i < **sizeof**(packet); i++) {

**while**(AS1\_SendChar((byte)packet[i]) != ERR\_OK) {}

}

// Update values on GUI

Update\_Putty\_CT\_Values();

//Reset flags, turn off and save power

timer1\_interrupted = 0;

MUXM\_PutVal(0); //turn off mux

SLP\_PutVal(0);//turn of zigbee

**break**;

}

//If command is reciered it sends it to the functions above to check what to do

**if** ((command\_recieved == 1) & (command\_sent == 0)) {//Recieve Flag

CommandReact(); //function uses logic above

command\_sent = 0;

command\_recieved = 0;

}//if-flag

}// if-main

}//end-for-loop-1

/////////////////////////////////// End Main User Code \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\/

/\*\*\* Don't write any code pass this line, or it will be deleted during code generation. \*\*\*/

/\*\*\* RTOS startup code. Macro PEX\_RTOS\_START is defined by the RTOS component. DON'T MODIFY THIS CODE!!! \*\*\*/

**#ifdef** PEX\_RTOS\_START

PEX\_RTOS\_START(); /\* Startup of the selected RTOS. Macro is defined by the RTOS component. \*/

**#endif**

/\*\*\* End of RTOS startup code. \*\*\*/

/\*\*\* Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! \*\*\*/

**for**(;;){}

/\*\*\* Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! \*\*\*/

} /\*\*\* End of main routine. DO NOT MODIFY THIS TEXT!!! \*\*\*/

/\* END main \*/

/\*!

\*\* @}

\*/

/\*

\*\* ###################################################################

\*\*

\*\* This file was created by Processor Expert 10.5 [05.21]

\*\* for the Freescale Kinetis series of microcontrollers.

\*\*

\*\* ###################################################################

\*/

APPENDIX E – RASPBERRY PI CODE

|  |
| --- |
| /home/pi/ctpi/src/main.cpp in pi@192.168.10 |

#include <stdio.h>

#include <termios.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#include <string.h>

#include <stdint.h>

#include <stdlib.h>

#include <arpa/inet.h>

#include <curl/curl.h>

//------------------------------------------------------------------------------

// Function prototypes

// The "static" keyword means that these functions are local to this .cpp file

// and not visible to other .cpp files in the project. Effectively, it signals

// that they are an implementation detail rather than an externally useful

// interface.

static bool init\_serial(char \*port);

static int receive\_packet();

// Incoming bytes are placed into rxbuf.buf[]. The struct rxbuf.packet allows

// for named access to particular fields within the binary protocol.

// The static keyword means that this variable is local to this .cpp file.

#define RXBUF\_LENGTH 500

static union {

char buf [RXBUF\_LENGTH];

struct \_\_attribute\_\_((packed)) {

uint8\_t start\_delimiter;

uint16\_t length;

uint8\_t frame\_type;

uint64\_t source\_address\_64;

uint16\_t source\_address\_16;

uint8\_t receive\_options;

char rf\_data[]; // up until the end of the union.

// There is a checksum field immediately after the end of rf\_data.

} packet;

} rxbuf;

static size\_t rxbuf\_idx = 0; // index of first unused item in rxbuf.buf

// File descriptor for the serial port

static int serial\_port = 0;

static bool init\_serial(char \*port)

{

printf("Opening serial port %s\n", port);

// Open the serial port for reading and writing.

// Returns a file descriptor that can be used with standard Linux functions

// read and write. See:

// $ man 2 read

// $ man 2 write

serial\_port = open(port, O\_RDWR);

if (serial\_port == -1) {

fprintf(stderr, "Failed to open serial port:\n%s\n", strerror(errno));

return false;

}

// Configure the serial port

termios tio; // termios is a struct defined in termios.h

memset(&tio, 0, sizeof(termios)); // Zero out the tio structure

tio.c\_cflag = CS8; // Select 8 data bits

tio.c\_cc[VMIN] = 1; // Demand at least 1 char from every call to read(), i.e. block execution until a char is received

cfsetospeed(&tio, B9600); // baud rate for output

cfsetispeed(&tio, B9600); // baud rate for input

tcsetattr(serial\_port, TCSANOW, &tio); // Apply these settings

// Done

return true;

}

size\_t http\_callback(void \*buffer, size\_t sz, size\_t nmemb, void \*userp)

{

size\_t size = sz \* nmemb;

// Was data received?

if (size > 0) {

// Is the first byte a 1 (to indicate success)?

char \*buf = (char \*)buffer;

if (buf[0] == '1') {

printf("Uploaded successfully\n");

} else {

fwrite(buffer, sz, nmemb, stdout);

}

} else {

printf("Empty response.\n");

}

return size;

}

int main(int argc, char\* argv[])

{

// argc is the number of command-line arguments provided to the program.

// The first argument (argv[0]) is always the name of the program.

if (argc < 2) {

printf("Usage:\n");

printf("%s /dev/ttyXXX\n", argv[0]);

return 1;

}

// Initialise the serial port

if (!init\_serial(argv[1])) {

return 1;

}

// Initialise the HTTP library

CURL \*curl = curl\_easy\_init();

if (!curl) {

printf("Failed to initialise the curl library\n");

return 1;

}

curl\_easy\_setopt(curl, CURLOPT\_WRITEFUNCTION, http\_callback);

// Repeatedly receive packets from the serial port

for (;;) {

int payload\_length = receive\_packet();

if (payload\_length >= 0) {

// The received packet is in rxbuf.packet

char url [RXBUF\_LENGTH+100];

float CT\_Current0, CT\_Current1, CT\_Current2, CT\_Current3;

// Add null terminator to rxbuf.packet

rxbuf.packet.rf\_data[payload\_length] = 0;

// Scan the received data and construct url

sscanf(rxbuf.packet.rf\_data, "%f,%f,%f,%f\n",&CT\_Current0,&CT\_Current1,&CT\_Current2,&CT\_Current3);

snprintf(url,500,"https://api.thingspeak.com/update?api\_key=0BF00P4ECP402C08&field1=%f&field2=%f&field3=%f&field4=%f",CT\_Current0,CT\_Current1,CT\_Current2,CT\_Current3);

printf("%s\n", url);

// Transmit to Thingspeak channel

curl\_easy\_setopt(curl, CURLOPT\_URL, url);

CURLcode res = curl\_easy\_perform(curl);

if (res != CURLE\_OK) {

fprintf(stderr, "curl\_easy\_perform() failed: %s\n", curl\_easy\_strerror(res));

}

}

}

// Exit

close(serial\_port);

return 0;

}

// This function reads from the serial port into rxbuf until a complete packet

// has been received. It returns the length of the data payload or -1 if

// receive failed.

int receive\_packet()

{

int bytes\_read;

char c;

// Zero the buffer

memset(&rxbuf, 0, sizeof(rxbuf));

rxbuf\_idx = 0; // Index of first unused char in rxbuf.buf

// Repeatedly receive characters

for (;;) {

// Receive a char

// This call will block (wait for a char) if tio.c\_cc[VMIN] (in init\_serial()) is > 0.

bytes\_read = read(serial\_port, &c, 1);

if (bytes\_read < 0) {

fprintf(stderr, "Failed to read from the serial port.\n");

fprintf(stderr, "%s\n", strerror(errno)); // Get text representing the reason for the failure

exit(1); // Quit the program

}

// Are we expecting a start of frame delimiter?

if ((rxbuf\_idx == 0) && (c != 0x7E)) {

// Expected a start of frame but didn't receive one.

//printf("Expected start of frame delimiter 0x7E, but received 0x%02x\n", c);

// We aren't synchronised with the xbee. Discard bytes (by restarting

// this loop body) until we see a start of frame delimiter.

continue;

}

// Save the character into the buffer

rxbuf.buf[rxbuf\_idx] = c;

rxbuf\_idx++;

// Abort if we overflow the buffer

if (rxbuf\_idx == (RXBUF\_LENGTH-1)) {

printf("Discarded packet that exceeded maximum length of %i bytes.\n", RXBUF\_LENGTH);

return -1;

}

// Once rxbuf\_idx is 3, we have received the length of the packet.

if (rxbuf\_idx == 3) {

// The length is in big endian format. Convert to the host format.

// This function means "big endian 16 to host format".

rxbuf.packet.length = be16toh(rxbuf.packet.length);

} else if (rxbuf\_idx >= 4) {

// There are 4 bytes that are not counted in length: delimiter, 2 bytes of length, checksum.

// The number of bytes received is rxbuf\_idx.

if (rxbuf\_idx >= rxbuf.packet.length+4) {

// Received a complete packet.

// Test the checksum

uint8\_t checksum = 0;

for (int i = 0; i < rxbuf.packet.length; i++) {

checksum += rxbuf.buf[i+3];

}

checksum = 0xFF - checksum;

uint8\_t received\_checksum = rxbuf.buf[rxbuf.packet.length + 3];

if (checksum != received\_checksum) {

printf("Discarded packet that failed checksum. Expected 0x%02x, received 0x%02x\n", checksum, received\_checksum);

return -1;

}

// Check that it's a receive packet

if (rxbuf.packet.frame\_type != 0x90) {

printf("Discarded unknown frame type 0x%02x\n", rxbuf.packet.frame\_type);

return -1;

}

// Perform a byte order swap for the multibyte fields in the packet.

// Length has already been swapped

rxbuf.packet.source\_address\_64 = be64toh(rxbuf.packet.source\_address\_64);

// Success

return rxbuf.packet.length - 12; // there are 12 bytes of header included in packet.length

}

}

}

}

APPENDIX F – WEBPAGE CODE

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<meta name="description" content="">

<meta name="author" content="">

<title>CC3501 Live Current Sensor</title>

<!-- Bootstrap core CSS -->

<link href="vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">

<link href="https://fonts.googleapis.com/css?family=Open+Sans" rel="stylesheet">

<!-- Custom styles for this template -->

<link href="css/full.css" rel="stylesheet">

</head>

<body>

<div id="background">

<img src="C:\Users\Clint\OneDrive\4th Year Sem2\CC3501\Github\CT-Monitoring-Assignment\CT\_Webpage\bg3.jpg" class="stretch" alt="" />

</div>

<!-- Navigation -->

<nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">

<div class="container">

<a class="navbar-brand" href="#">CC3501 Near-to-Real Time Current Monitoring</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarResponsive">

<ul class="navbar-nav ml-auto">

<li class="nav-item active">

<a class="nav-link" href="#"><font color="white">#Clinton Elliott Electrical & Engineering

<span class="sr-only">(current)</span>

</a>

</ul>

</div>

</div>

</nav>

<!-- Page Content -->

<div class="container">

<div class="mid">

<h1 class="big"><b><font color="red">CTM-3501</b> Current Channels</h1>

</div>

</div>

<div class="row">

<div class="col-md-1"></div>

<div class="col-md-10">

<iframe height="400" class="graphs" src="https://thingspeak.com/channels/349528/charts/1?api\_key=IIAZ75FAB8EKHRRX&height=max&width=max&title=Channel 1&yaxis=Current Transformer 1"></iframe>

</div>

<div class="col-md-1"></div>

</div>

<br>

<div class="row">

<div class="col-md-1"></div>

<div class="col-md-10">

<iframe class="graphs" src="https://thingspeak.com/channels/349528/charts/2?api\_key=IIAZ75FAB8EKHRRX&height=max&width=max&title=Channel 2&yaxis=Current Transformer 2"></iframe>

</div>

<div class="col-md-1"></div>

</div>

<br>

<div class="row">

<div class="col-md-1"></div>

<div class="col-md-10">

<iframe class="graphs" src="https://thingspeak.com/channels/349528/charts/3?api\_key=IIAZ75FAB8EKHRRX&height=max&width=max&title=Channel 3&yaxis=Current Transformer 3"></iframe>

</div>

<div class="col-md-1"></div>

</div>

<br>

<div class="row">

<div class="col-md-1"></div>

<div class="col-md-10">

<iframe class="graphs" src="https://thingspeak.com/channels/349528/charts/4?api\_key=IIAZ75FAB8EKHRRX&height=max&width=max&title=Channel 4&yaxis=Current Transformer 4"></iframe>

</div>

<div class="col-md-1"></div>

</div>

<div class="row">

<div class="col-md-12">

<div class="container">

<div class="mid">

<h1 class="medium"><b><font color="yellow"><br>Thank You <p>

</b> This project was powered by Mick Beans 8.2 & Dante, in collaboration with James Cook University.</h1>

</div>

</div>

</div>

<!-- Bootstrap core JavaScript -->

<script src="vendor/jquery/jquery.min.js"></script>

<script src="vendor/bootstrap/js/bootstrap.bundle.min.js"></script>

</body>

</html>