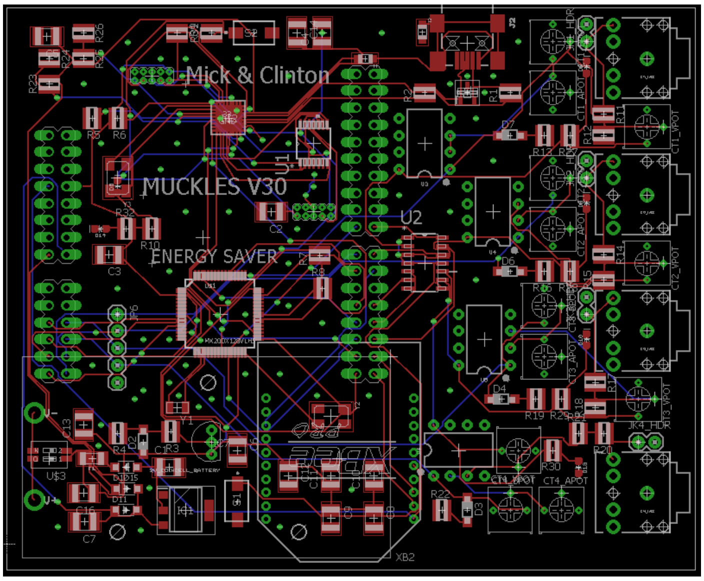


CURRENT TRASFORMER METER

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

This page intentionally left blank

You had me at "Hello World!"

-Anonymous

TABLE OF CONTENTS

LIST OF FIGURES

CURRENT TRASFORMER METER

1. 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.

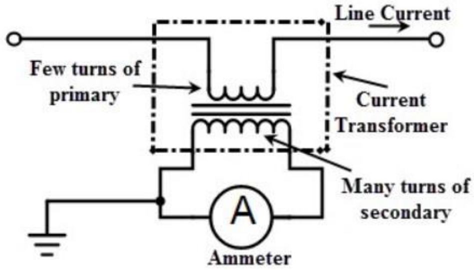


Figure 1 - Current Transformer [1]

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:

$$I_1 N_1 = I_2 N_2$$

Where I_1 is the primary winding current, I_2 is the secondary winding current, N_1 and N_2 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:

$$V_2 = V_1\left(\frac{N_2}{N_1}\right)$$

To demonstrate this effect, we assume N_1 and N_2, are 1 and 100 with a primary voltage of 230. Substituting into the equation:

$$V_2 = 230\left(\frac{100}{1}\right)$$

$$= 23{,}000\ kV$$

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.

$$BatteryLife(Hours) = \frac{Battery\ Capacity(Amp.hour)}{Current\ Draw\ (Amp)} \quad (1)$$

2. 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.

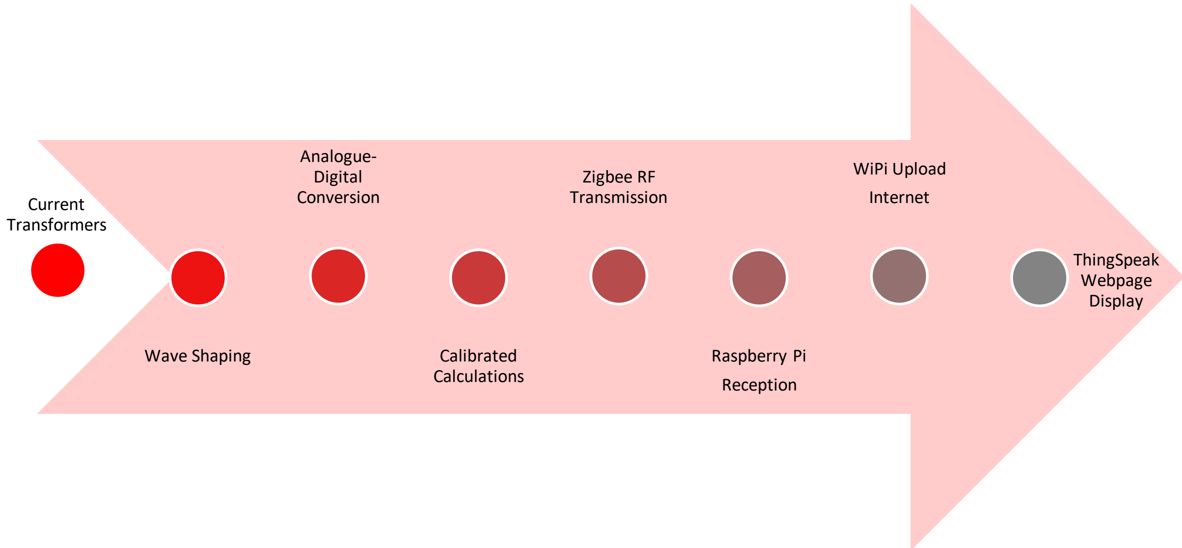


Figure 2 - Overview of Operation

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.

3. HARDWARE DESIGN

3.1. 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.

3.1.1. 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.

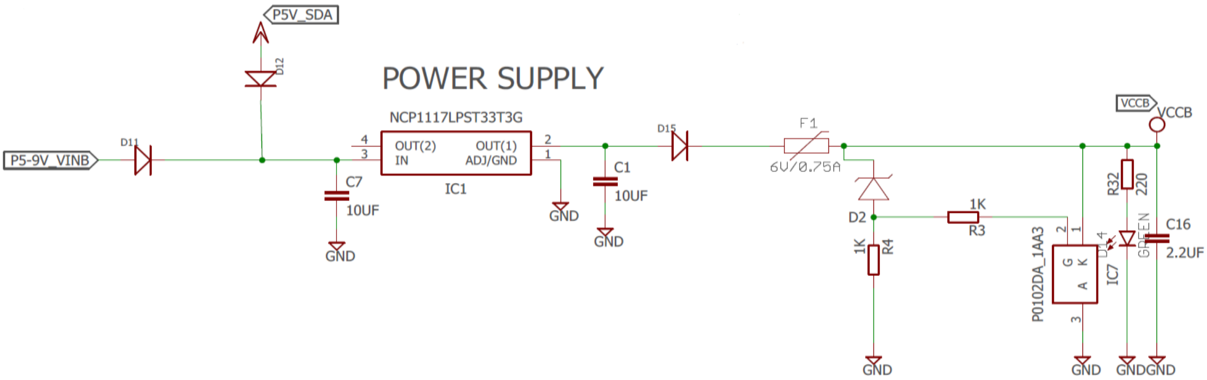


Figure 3 - Power Supply and Corresponding Protection

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.

3.1.2. 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.

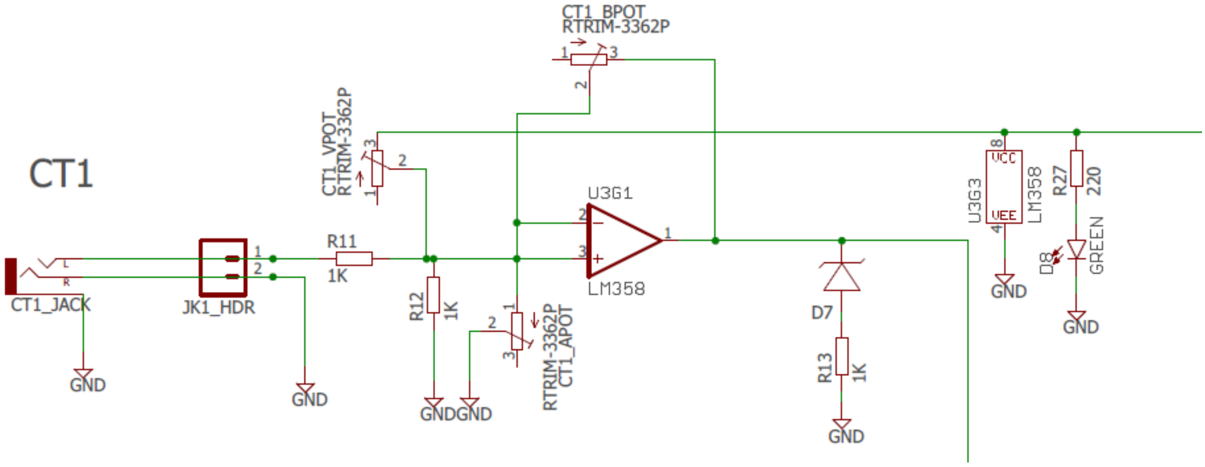


Figure 4 - Operational Amplifiers and Input Shifting Circuit

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.

3.1.3. 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.

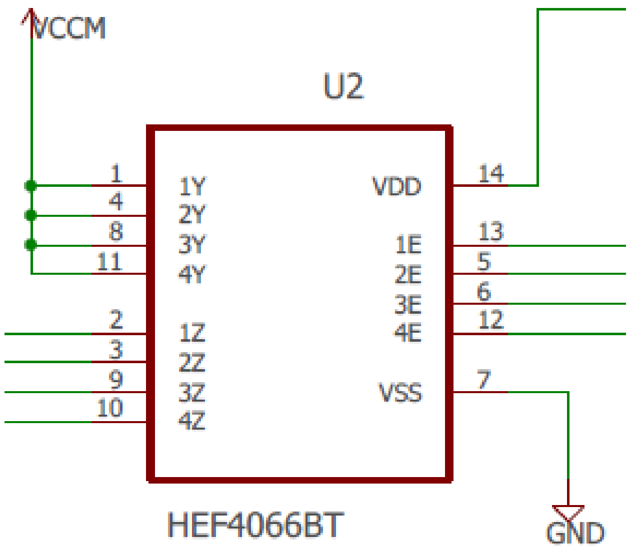


Figure 5 - Multiplexer

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.

3.1.4. 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

3.1.5. 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.

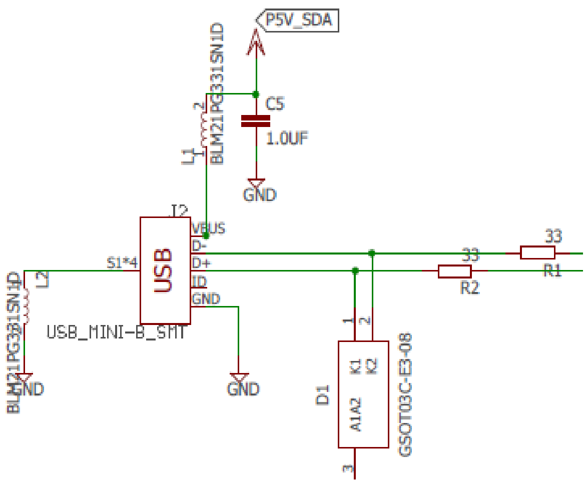


Figure 7 - Mini USB Circuit

3.1.6. 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.

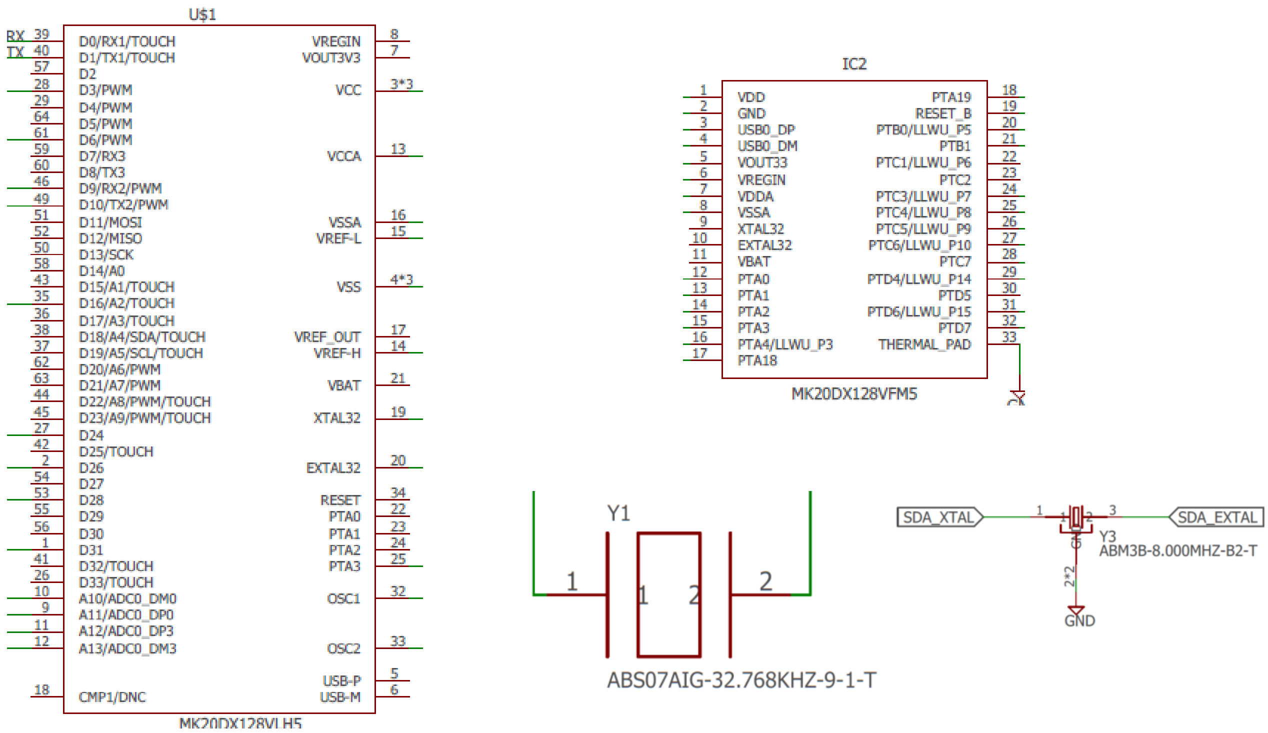


Figure 8 - Primary (left) and SDA (right) Processors and Oscillators (bottom right)

3.1.7. 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.

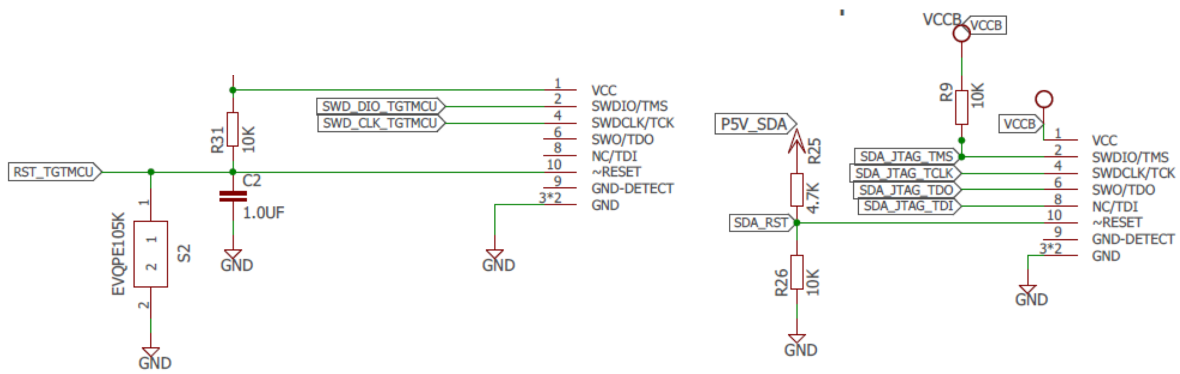


Figure 9 - SDA (left) and JTAG (right) Headers

3.1.8. 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).

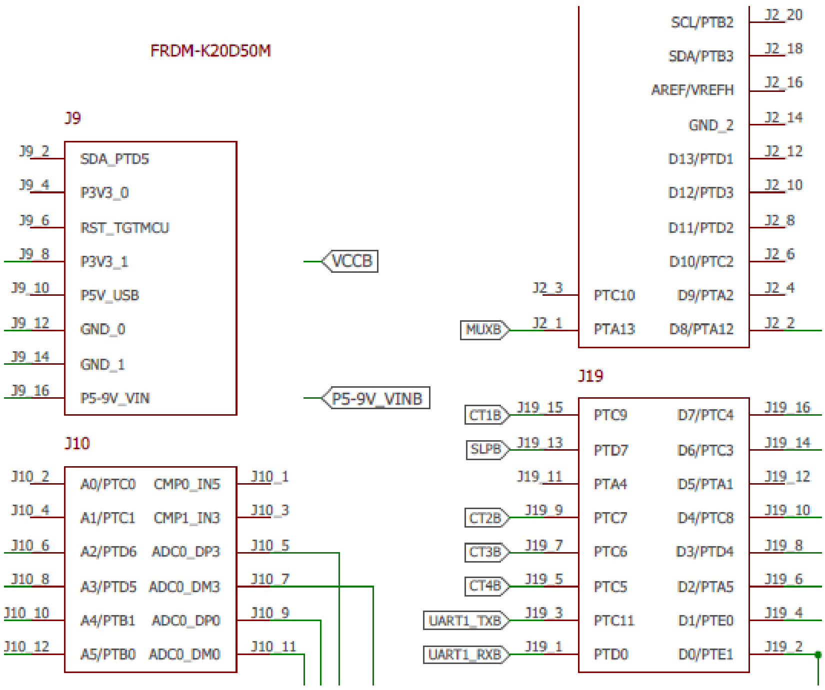


Figure 10 - FRDM Board Headers

- 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.

3.1.9. Battery Selection

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

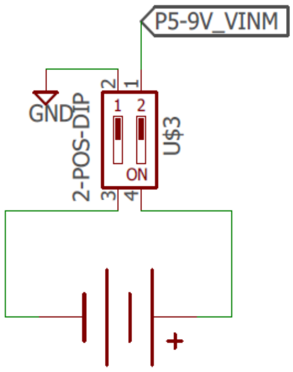


Figure 11 - Coin Cell Battery

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.

3.2. Power Optimisation

3.2.1. Operational Amplifiers

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

$$Power = (V_{cc}^{+} - V_{cc}^{-}) * I_q$$

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.

$$I_q\ typical = 0.5\ mA \qquad V_{cc}^{+} = 3.3\ V$$

$$I_q \max = 1.2\ mA \qquad V_{cc}^{-} = 0\ V$$

$$Power_{typical} = (3.3 - 0) * 0.5 \times 10^{-3}$$

$$Power_{typical} = 1.65\ mW \text{ per Op-Amp}$$

$$Power_{maximum} = (3.3 - 0) * 1.2 \times 10^{-3}$$

$$Power_{maximum} = 3.96\ mW \text{ per Op-Amp}$$

The component is a dual op-amp package; therefore, each package will draw 2.4 mA and the board has 4, resulting in 9.6 mA.

3.2.2. Voltage Regulator

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

$$P_D = \frac{T_{J(max)} - T_A}{R_{\theta JA}}$$

- $T_{J(max)}$ - maximum junction temperature range $= 150\ °C$
- T_A - operating ambient temperature range $= 25$ °C
- $R_{\theta JA}$ - thermal Resistance, Junction–to–Ambient$= 108$ °C$/W$

The maximum power $P_D = \frac{150-25}{108}$:

$$P_D = 1.15\ W$$

The maximum current output of the regulator is governed by:

$$I = \frac{P}{V}$$

- V is the voltage drop of the regulator

$$V = 9 - 3.3 = 5.7\ V$$

$$I = \frac{1.15}{5.7}$$

$$I = 201.75\ mA$$

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.

- T_{OP} - operating junction temperature range = $0 - 125\ °C$
- $R_{\theta JC}$ - thermal resistance junction-to-case = $15\ ℃/W$
- $R_{\theta JA}$ - thermal resistance, junction–to–ambient= $108\ ℃/W$

T_{OP} specifies the temperature of the "junction", the active part of the regulator, can get before it goes into thermal shutdown. $R_{\theta JC}$ 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 15 ℃ above the temp of the heat sink. $R_{\theta JA}$ 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 60 ℃. The junction temperature needs to stay below $125\ °C$, therefore, the maximum temperature rise allowable is $125 - 60 = 65$℃. The power dissipation is given by:

$$P = \frac{65℃}{R_{\theta JA}}$$

$$P = \frac{65}{108}$$

$$P = 0.602\ W$$

The current is $I = \frac{P}{V}$:

$$I = \frac{0.602}{5.7}$$

$$I = 106\ mA$$

3.2.3. 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 $205\ mA$. The power down current of the XBee is $3.5\ \mu A$.

3.2.4. MK20DX128VLH5 & MK20DX128VFM5 Processors

The absolute maximum ratings for the MK20DX128VFM5 and MK20DX128VLH5 are obtained from the device data sheet. The maximum power supply current I_{DD}, 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:

$$I_{DD} = 155\ mA$$

3.2.5. 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.

$$I_{max} = 50mA + 100mA$$

$$I_{max} = 150mA$$

3.2.6. HEF4066B Quad Single-Pole Single-Throw Analog Switch (Multiplexer)

The maximum supply current at worst case scenario for the Multiplexer was 7.5 μA while the device is operating at an ambient temperature of 125℃.

3.2.7. Mini USB

The specifications of USB 2.0 states that the maximum current draw was $100mA$.

3.2.8. LEDs

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

$$R = \frac{V_s - V_f}{i}$$

- $V_s = 3.3\ V$ Supply voltage
- $V_f = 2.6\ V$ LED forward voltage drop (found in data sheet)
- $R = 220\ \Omega$ Resistor value

$$i = \frac{3.3 - 2.6}{220}$$

$$i = 3.18\ mA$$

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:

$$I = 3.18 \times 6$$

$$I = 19.08\ mA$$

3.2.9. 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 (mA)
Op-Amps	9.6
Voltage Regulator	201.7
XBee Pro	205
MK20DX128VLH5	155
MK20DX128VFM5	155
NAND Gates	150
Multiplexer	0.0075
LED's	19.08
Mini USB	100
	$Total = 995.3875$

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 $995.3875\ mA$, which is approximately 1000 mA (1 A).

Using the battery life equation (1):

$$BatteryLife(Hours) = \frac{Battery\ Capacity(A.hour)}{Current\ Draw\ (A)}$$

$$BatteryLife(hr) = \frac{230(mA\,.hr)}{1000\ (mA\ )}$$

$$BatteryLife(hr) = 0.23\ hr$$

$$BatteryLife(min) = 0.23 * 60\ \mathrm{min} = 13.8\ min$$

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

3.2.10. 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

3.3. 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.

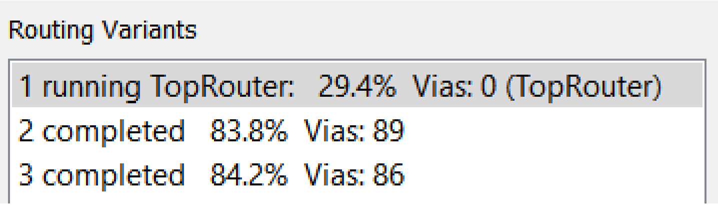


Figure 12 - Auto Routed Test of Placement

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.

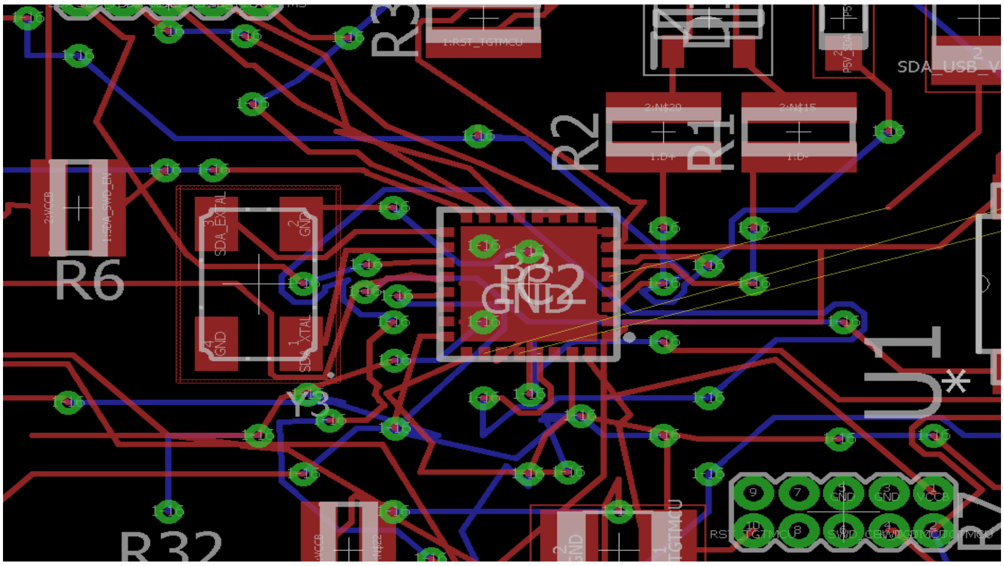


Figure 13 - SDA Processor with Poor Routing

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

A. Placing all the CT circuits in their individual circuits off to the side
B. Increasing the free space around each processor for routing
C. Allow room outside the FRDM headers for routing
D. Placing the power supply components together in a corner
E. 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.

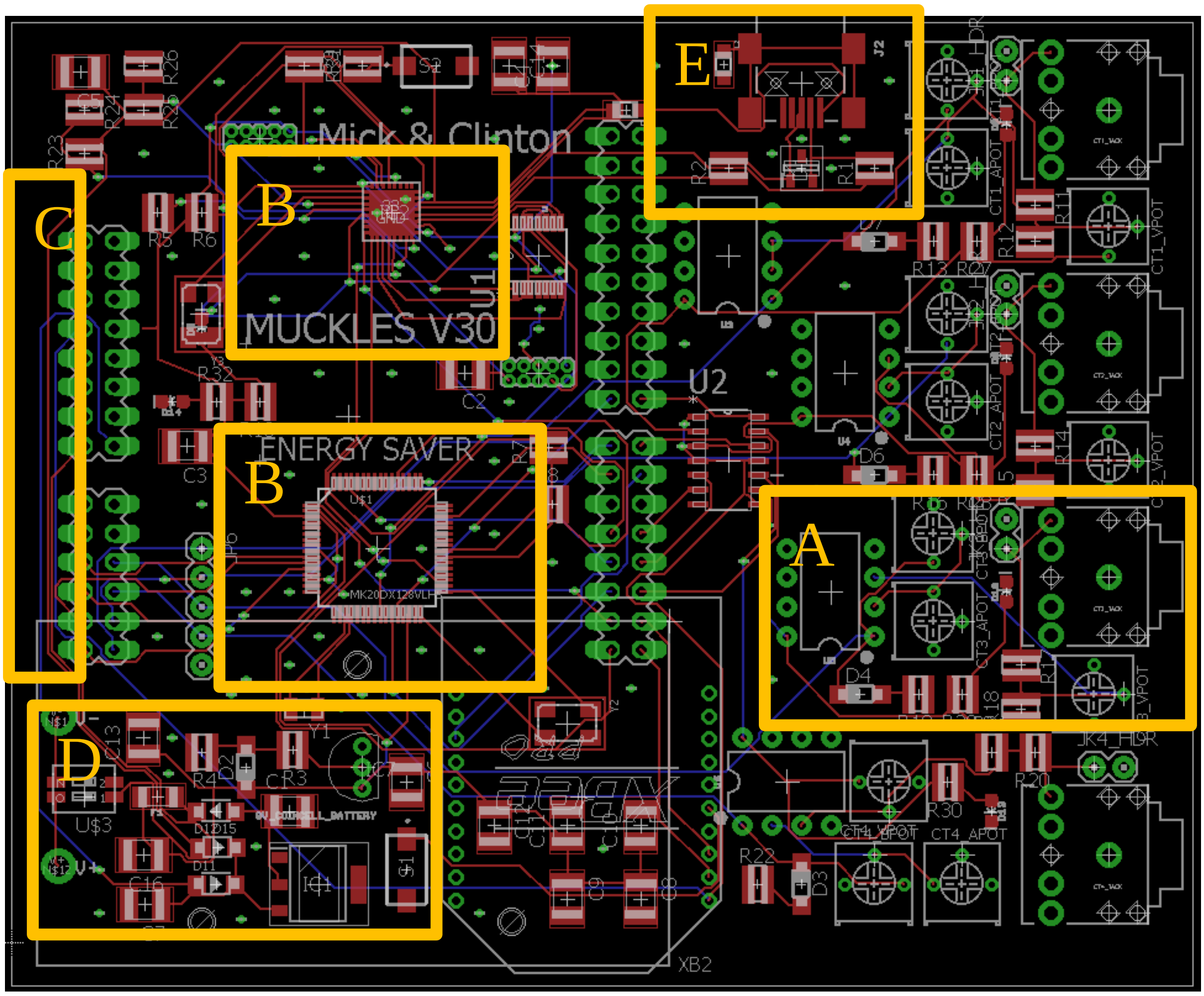


Figure 14 - Finished PCB

4. SOFTWARE DESIGN

4.1. Kinetis Code

4.1.1. 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.

4.1.2. 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).

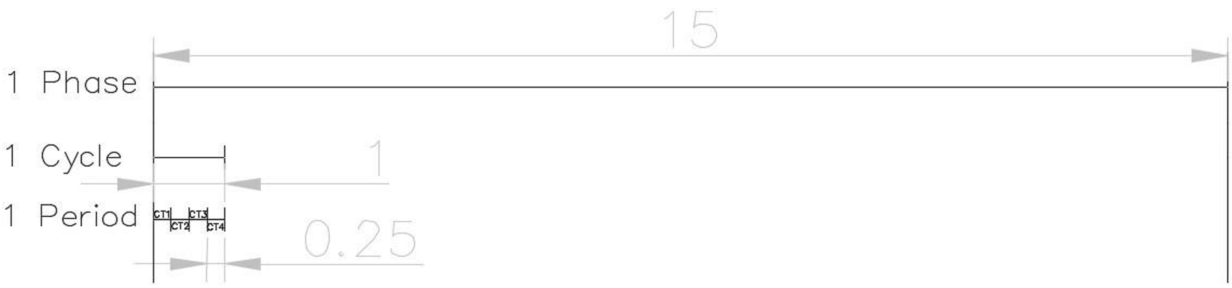


Figure 15 - Definition of Timing in Circuit (seconds)

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.

4.1.3. 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.

4.1.4. 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).

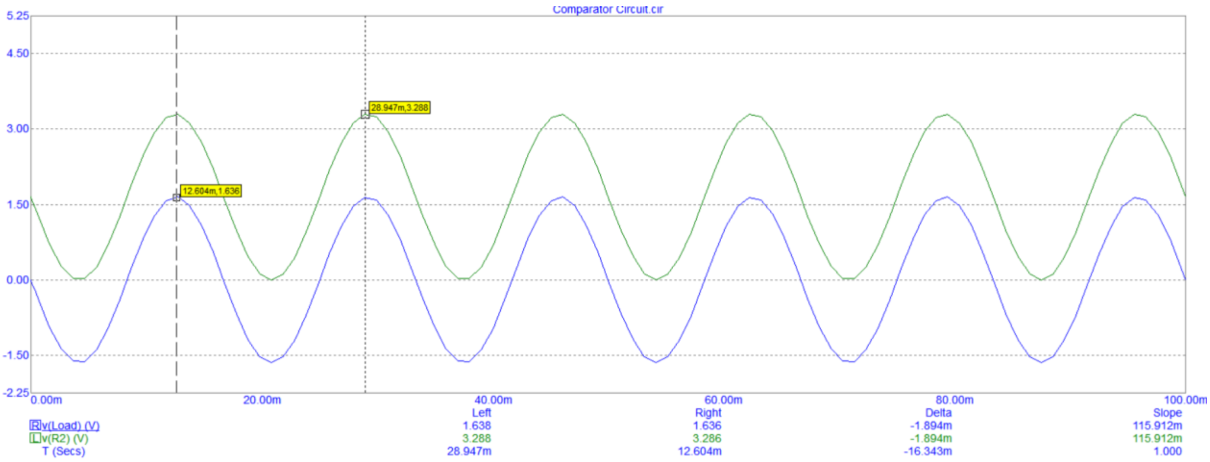


Figure 16 - Shifted Waveform from CT to ADC Input

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).

$$V_{rms} = \sqrt{\frac{1}{T}\int_0^T [v(t)]^2 dt} \tag{2}$$

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

$$x_{rms} = \sqrt{\frac{1}{n}({x_1}^2 + {x_2}^2 + \cdots + {x_n}^2)}$$

$$x_{rms} = \sqrt{\frac{1}{n}\sum_{i=0}^{n}({x_i}^2)} \tag{3}$$

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.

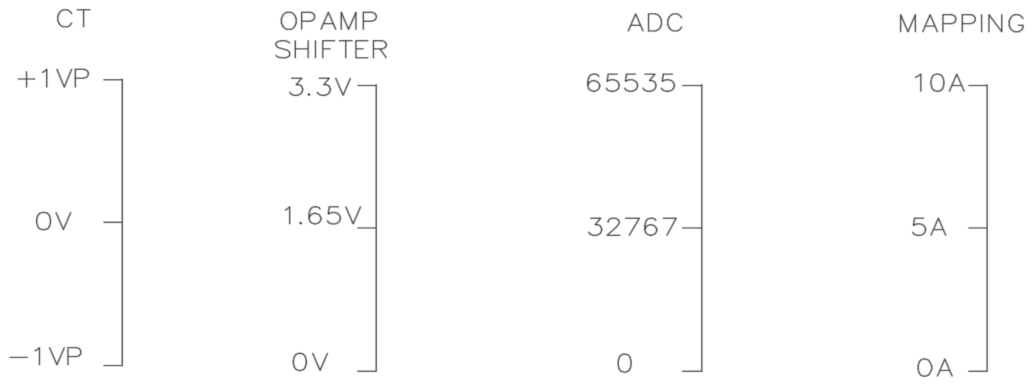


Figure 17 - Mapping Layout of Data

4.1.5. 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.

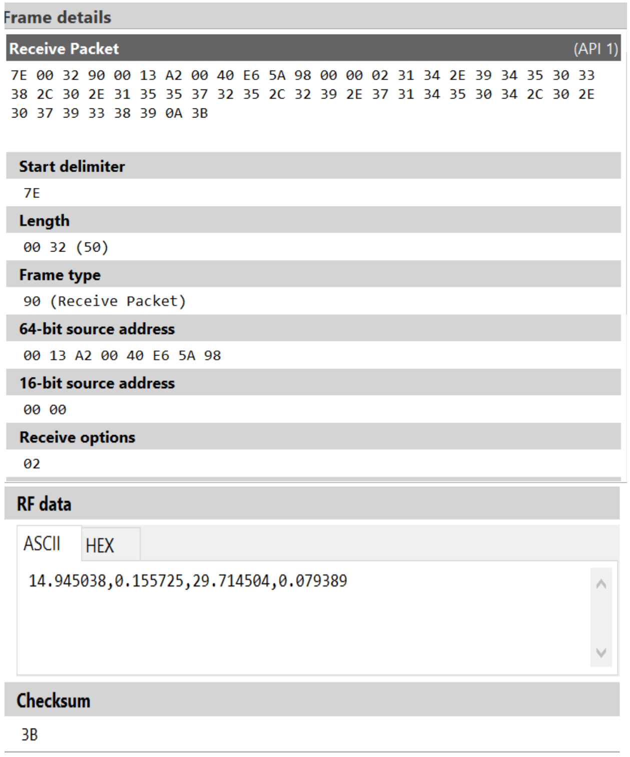


Figure 18 - Frame Details

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

```
// 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
```

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.

```
// Place message into frame packet
for (int i = 0; i < message_size; i++) {
packet[17 + i] = (byte)message[i];
}
```

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.

```
// 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) {}
```

Figure 21 - Checksum and AS Transmission

4.1.6. 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

4.1.7. 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".

4.2. C++ Code

4.2.1. 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).

```
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;
```

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;
```

4.2.2. 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).

```
// 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);
```

Figure 23 - Libcurl Code

4.2.3. 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.

```
// 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=JYCQY04Q24PQWZPR&field1=%f&field2=%f&
field3=%f&field4=%f",CT_Current0,CT_Current1,CT_Current2,CT_Current3);

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

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

4.3. Webpage Code

4.3.1. 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.

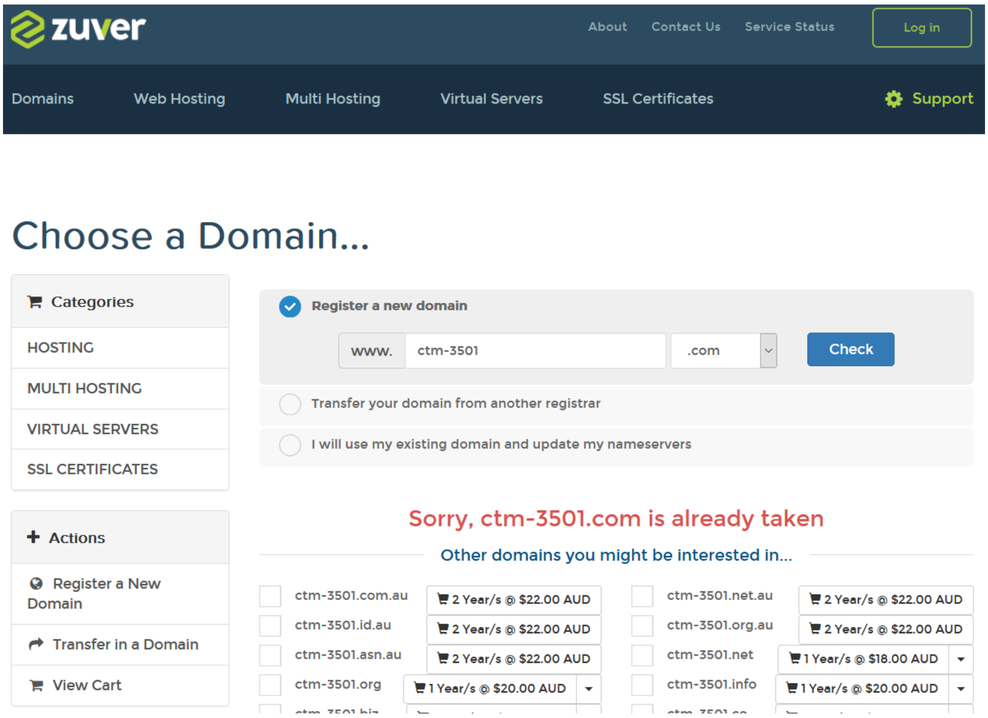


Figure 25 - Hosting Website

4.3.2. 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).

```
<!DOCTYPE html>
<html class="gr__ctm-3501_com" lang="en"><head>
<meta http-equiv="content-type" content="text/html; charset=UTF-8">

    <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>
```

Figure 26 - HTML Initial Setup

4.3.3. 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.

```css
#background {
    width: 100%;
    height: 400%;
    position: absolute;
    left: 0px;
    top: 0px;
    z-index: 0;
}

.stretch {
    min-width:100%; min-height:100%; width:auto; height:auto;
    z-index: 0;
}
```

Figure 27 - Background Style Sheet

4.3.4. 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.

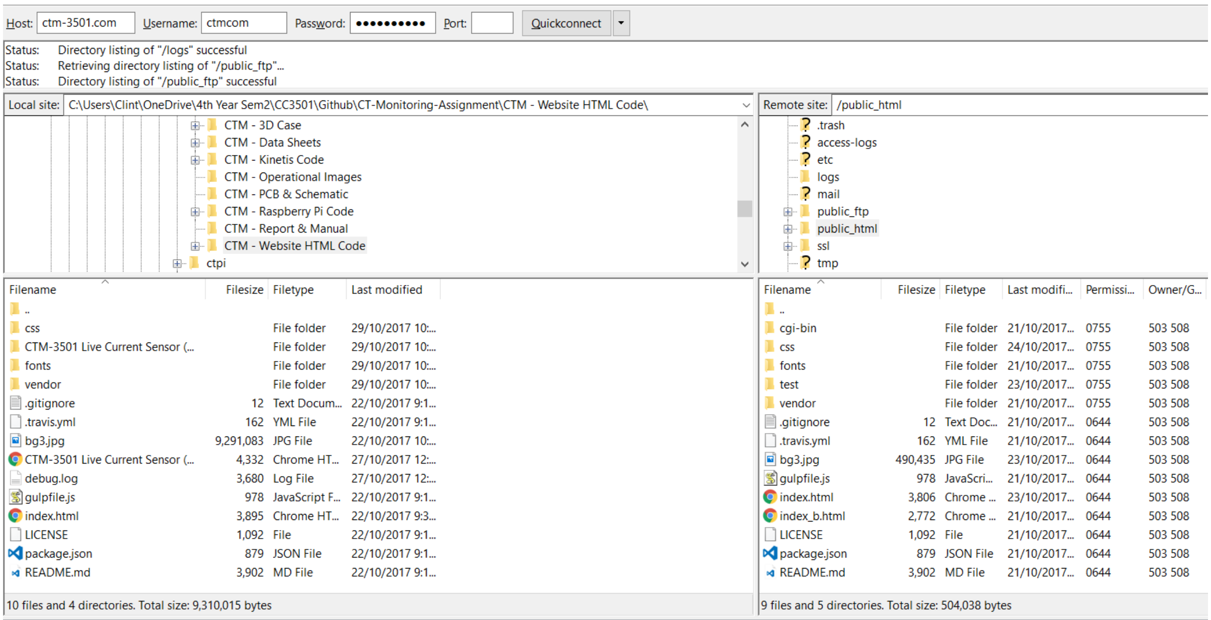


Figure 28 - Transferring Files to the Host

5. TESTING & RESULTS

5.1. 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.

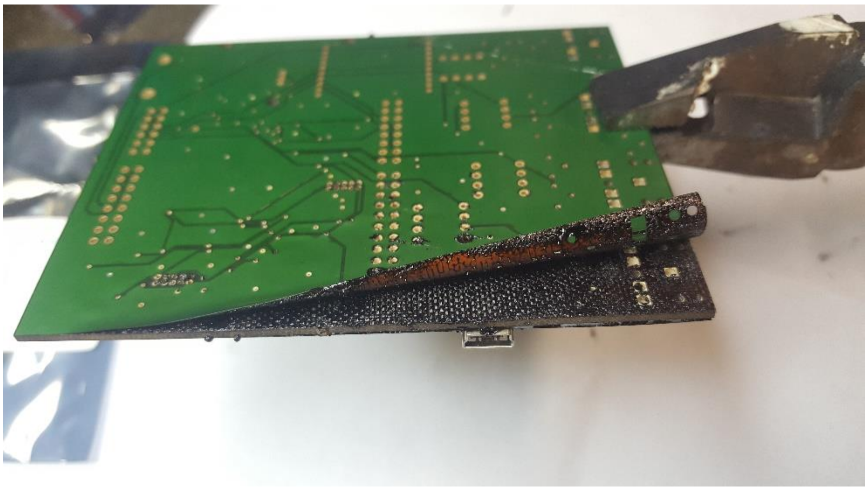


Figure 29 - Burnt PCB

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.

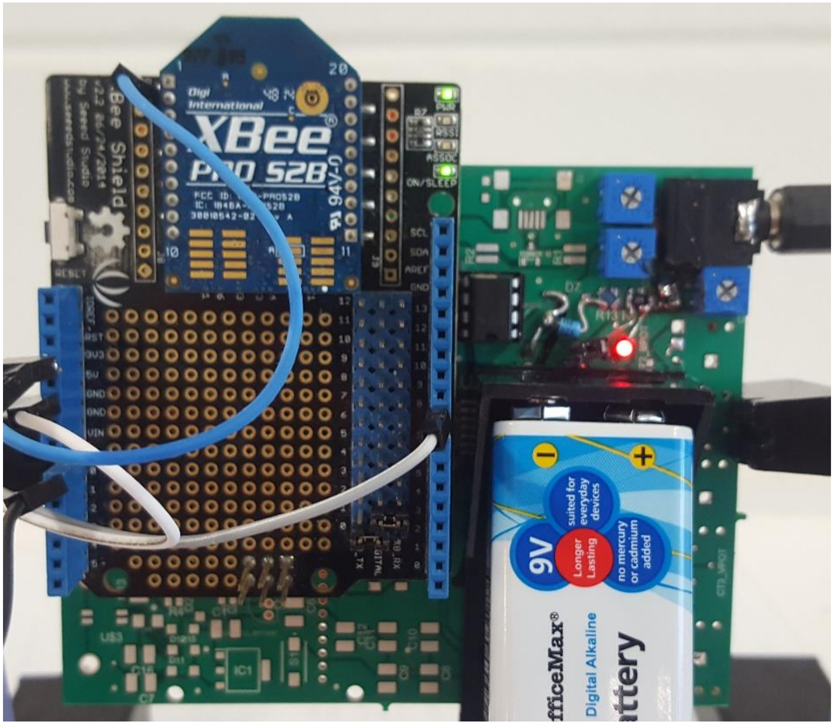


Figure 30 - Finished PCB and One CT Connected

5.2. 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

5.3. Webpage Display

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

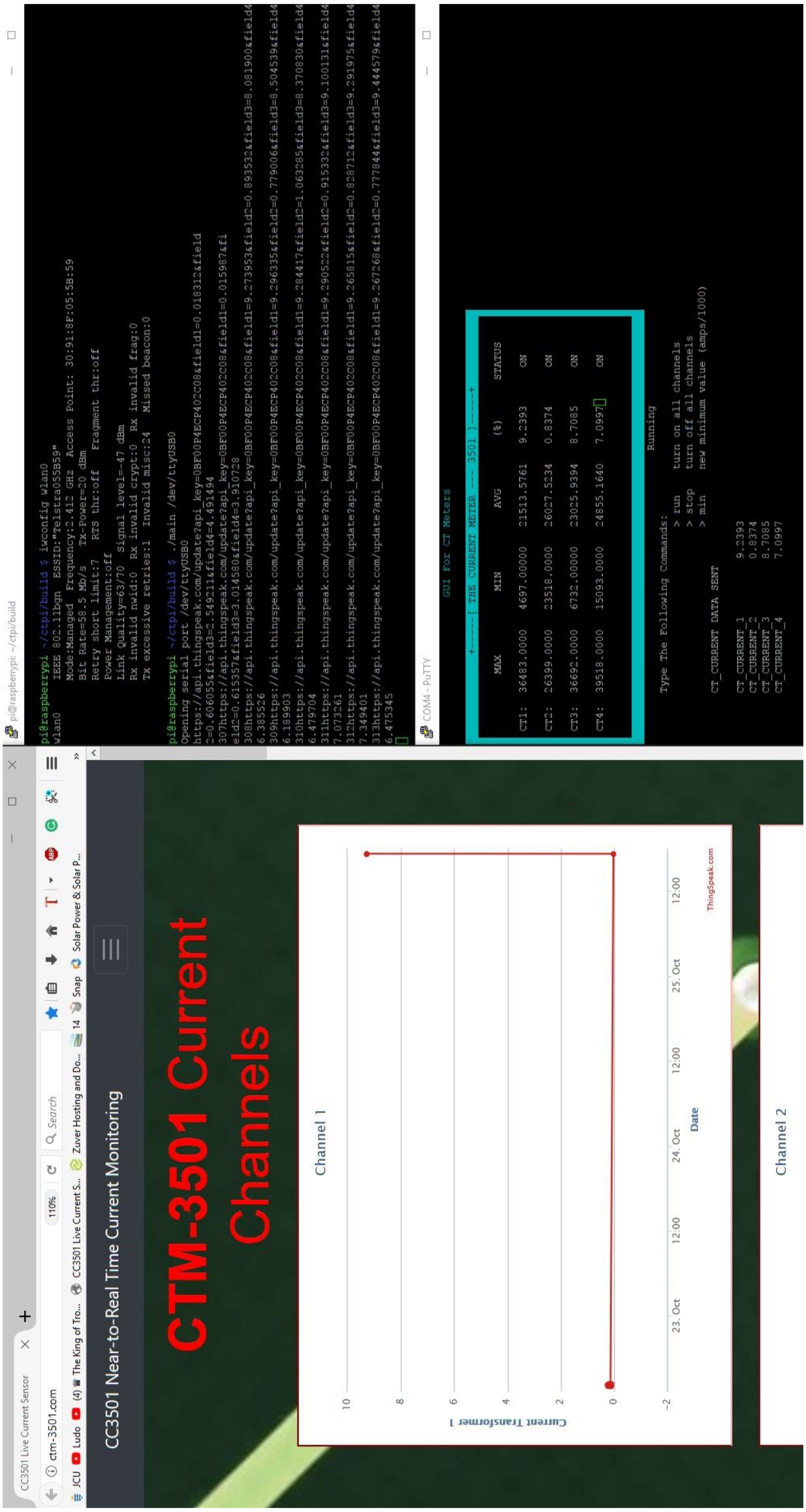


Figure 32 - The Checking Of Data Being Sent By CTM-3501 And Being Displayed On The Pi And Internet Webpage

5.4. Game-Day Performance

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

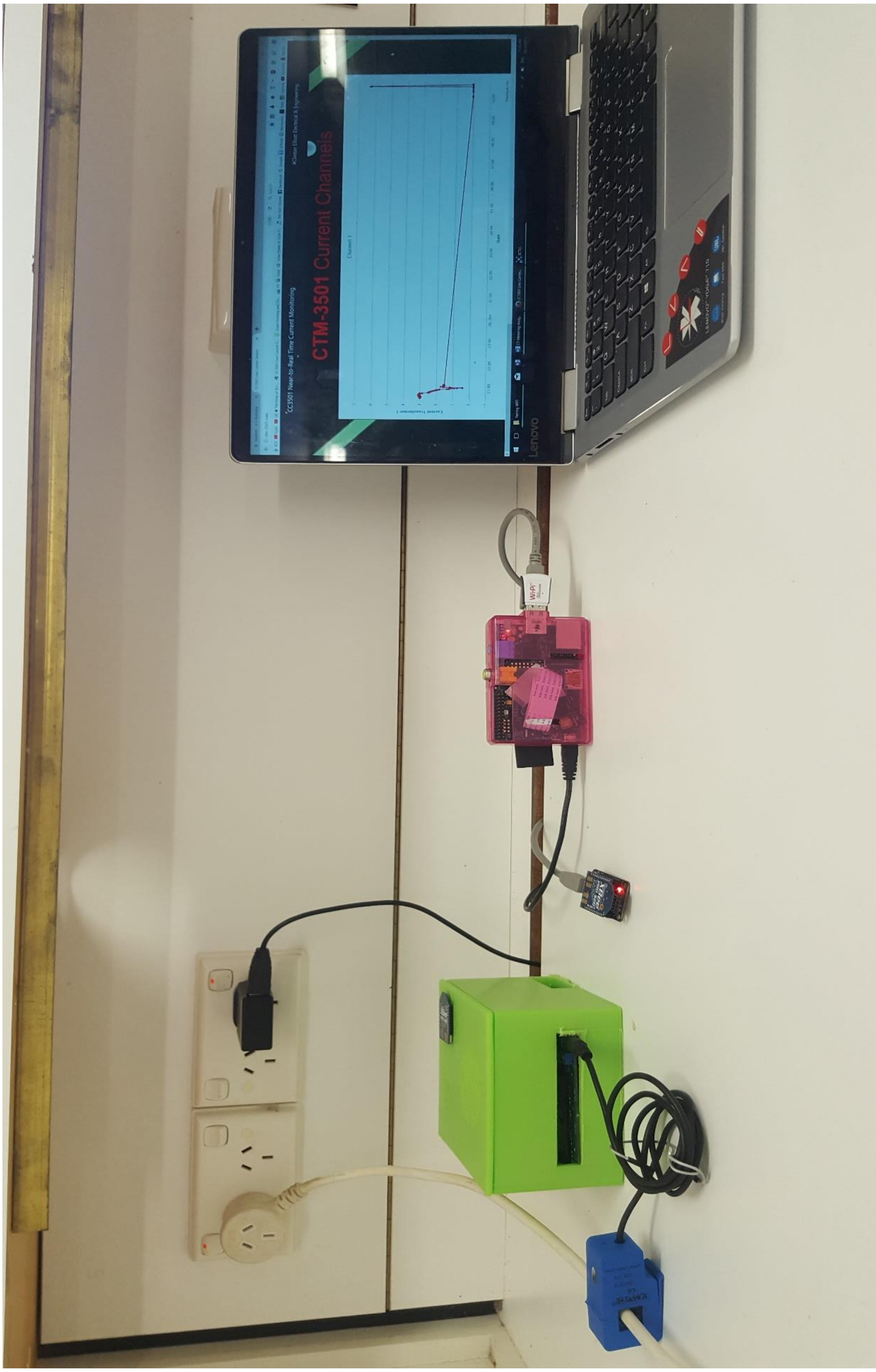


Figure 33 - The CTM-3501 Setup

5.5. 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.

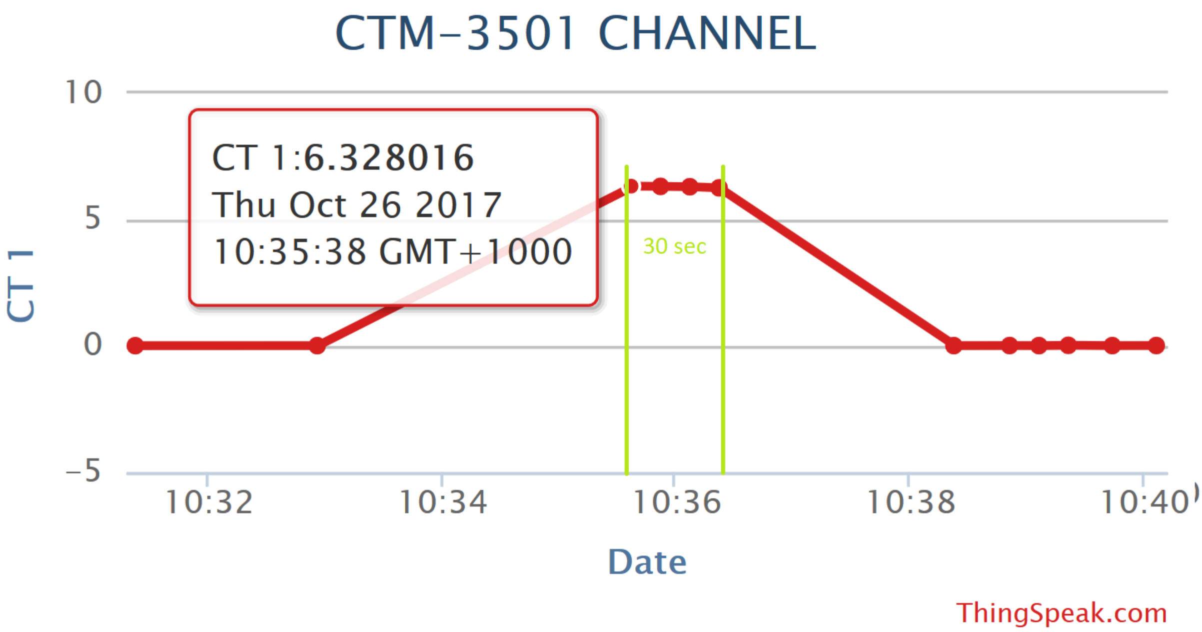


Figure 34 - Current Draw of Vacuum Cleaner

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.

6. LIMITATIONS & RECOMMENDATIONS

6.1. 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.

6.2. 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.

7. 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 😊

8. REFERENCES

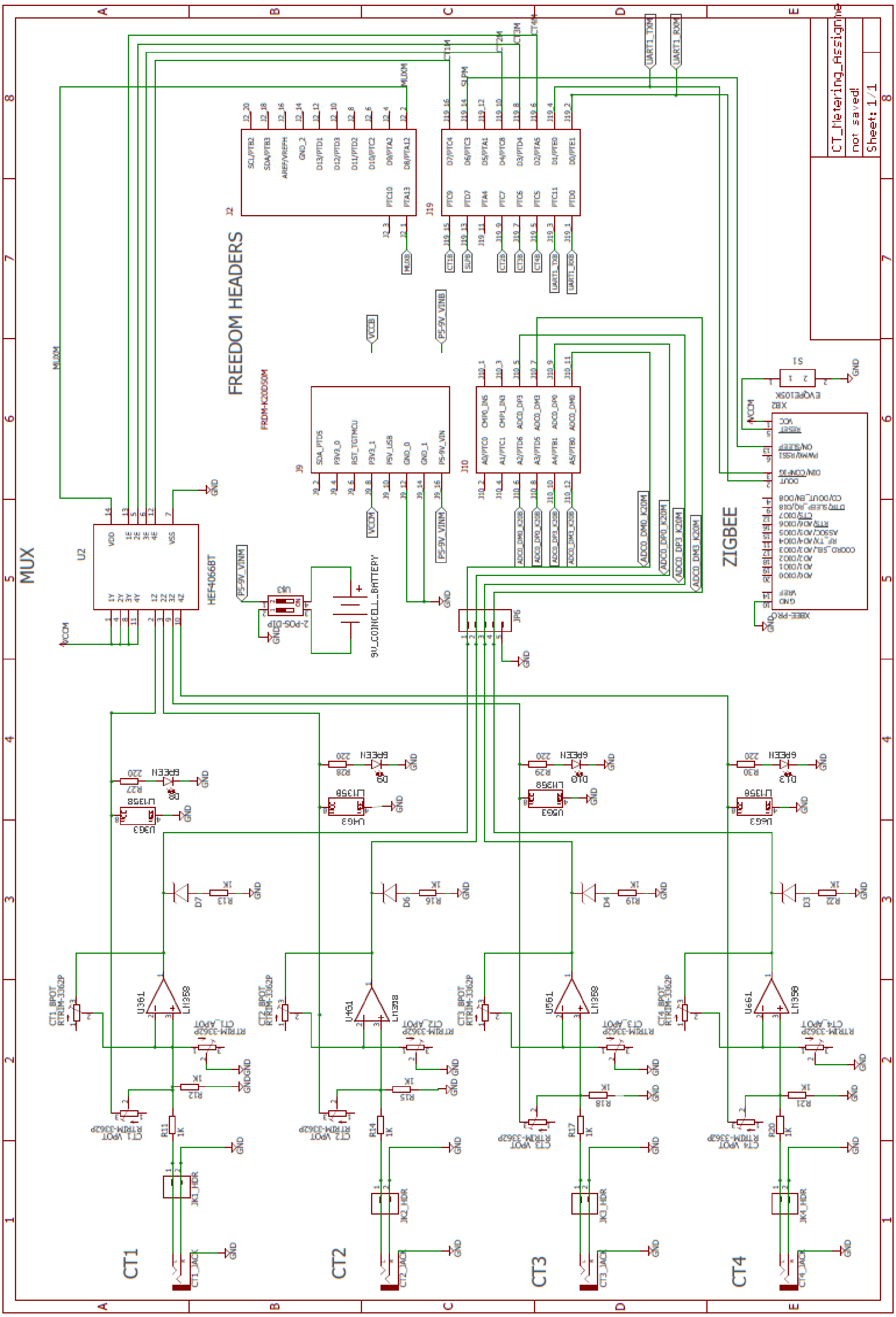
[1] E. Hub. (2017, 17/10). *Current Transformer*. Available: http://www.electronicshub.org/current-transformer/

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

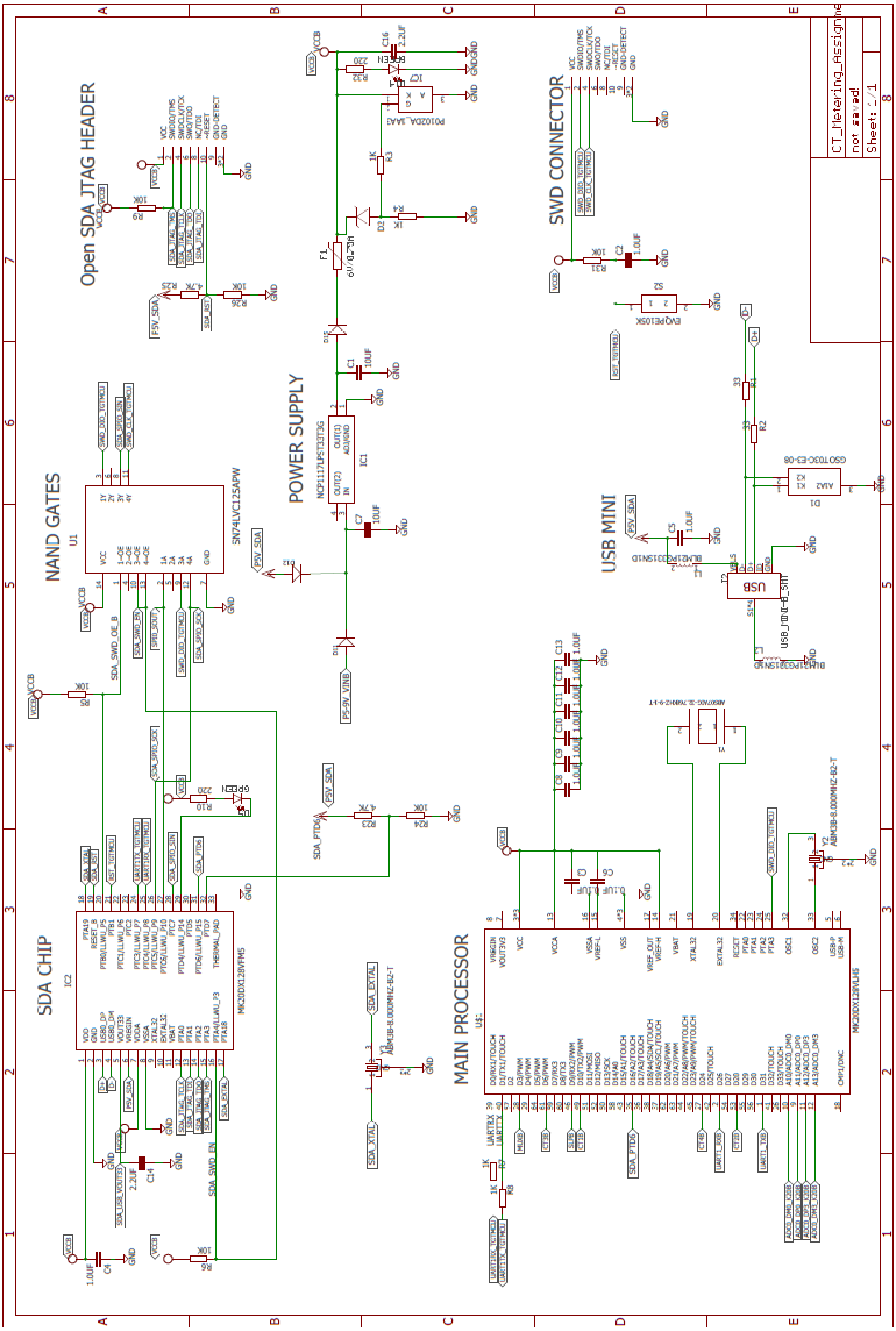
[3] Data sheets, “GitHub” – Various data sheets, 2017, https://github.com/clintonelliott23/CT-Monitoring-Assignment.

LIST OF APPENDICES

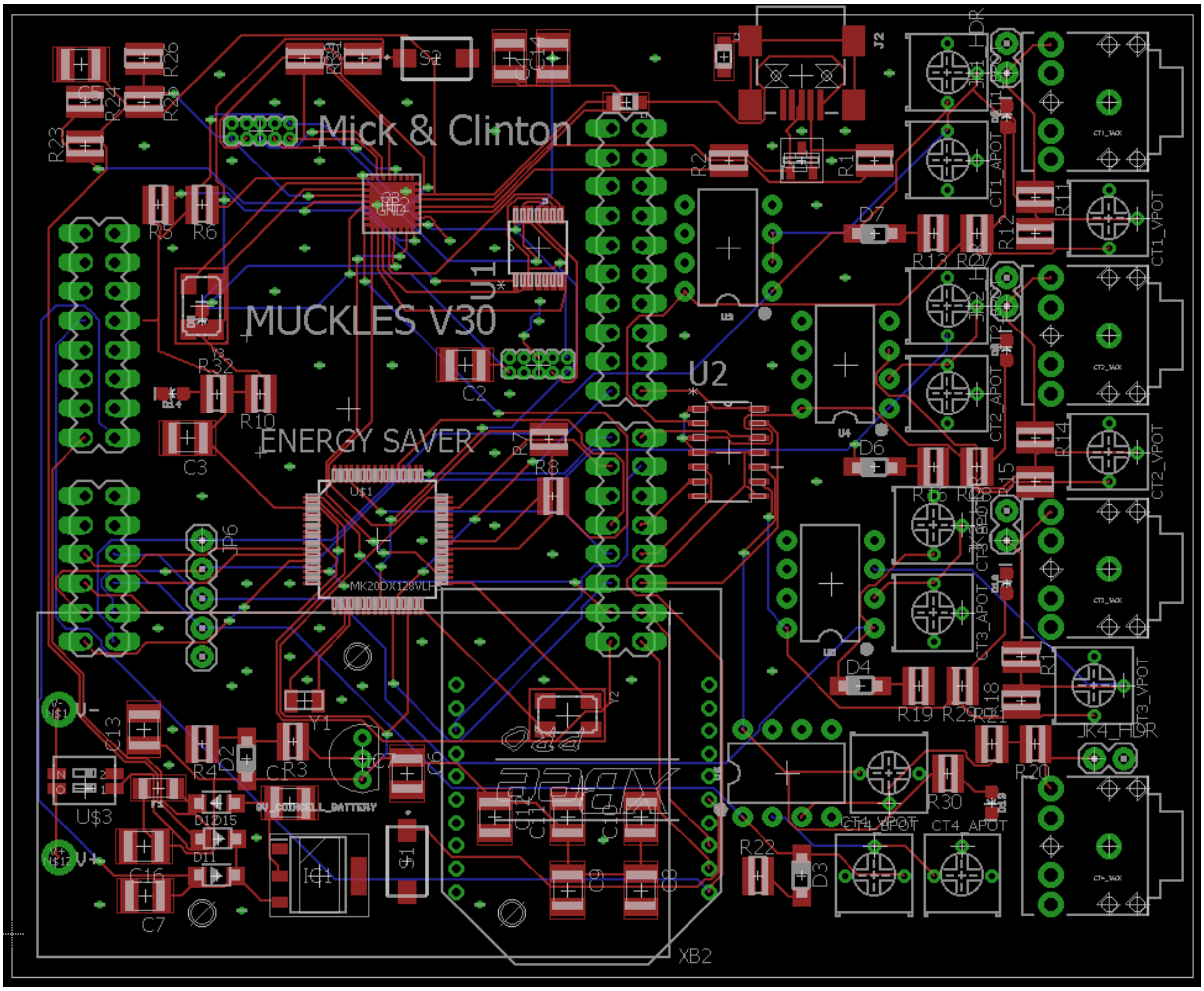
APPENDIX A.1 – SCHEMATIC



APPENDIX A.2 – SCHEMATIC



APPENDIX B – PCB LAYOUT



APPENDIX C – BILL OF MATERIALS

Order Code	MuF#	Quantity	Part	Description	Size	Mount	Max Voltage	lax Curren	Power Draw	Unit price	Total price
1887096	MM3Z3V3ST1G	4	Zener	ZENER DIODE, 200MW, 3.3V, SOD-323	Axial	SMD				$0.22	0.864
1902429	BZT52C15S	1	Zener	Zener Single Diode, 15 V, 200 mW, SOD-323		SMD				$0.54	0.535
8737592	PESD3V3S2UT	1	ESD	ESD Protection Device, TVS, 20 V, SOT-23, 3 Pins		SMD	20			$0.13	0.133
2409013	LM358AN	4	Op-Amp	OP AMP, DIP-8		DIP	16	0.8ma		$0.31	1.256
2336656	HEF4066BT,653	1	Mux	Analogue Switch, Quad Channel, SPST, 4 Channels, 175 ohm, 3V to 15V, SOIC, 14 Pins		SOIC	18	10ma	100mw	$0.18	0.177
											0
2329885	LR0204F1K0	14	Resistors	Through Hole Resistor, 1 kohm, 200 V, Axial Leaded, 250 mW			200				14
2690182	MCW0612MC1001FP100	2	Resistors	SMD Chip Resistor, 1 kohm, 75 V, 0612 [1632 Metric], 1 W,		SMD				$0.30	0.598
2706091	MCW0612MC2209FP100	1	Resistors	SMD Chip Resistor, 22 ohm, 75 V, 0612 [1632 Metric], 1 W						$0.81	0.81
2706096	MCW0612MC2000FP100	1	Resistors	SMD Chip Resistor, 200 ohm, 75 V, 0612 [1632 Metric], 1 W						$0.88	0.88
2706092	MCW0612MC3309FP100	2	Resistors	SMD Chip Resistor, 33 ohm, 75 V, 0612 [1632 Metric], 1 W,		SMD				$0.81	1.62
2706102	MCW0612MC4701FP100	2	Resistors	SMD Chip Resistor, 4.7 kohm, 75 V, 0612 [1632 Metric], 1 W,		SMD				$0.88	1.76
2706103	MCW0612MC1002FP100	5	Resistors	SMD Chip Resistor, 10 kohm, 75 V, 0612 [1632 Metric], 1 W,		SMD				$0.88	4.4
2466724	885012209012	2	Capacitors	SMD Multilayer Ceramic Capacitor, 1210 [3225 Metric], 2.2 µF, 16 V		SMD				$0.55	1.102
2466731	885012209019	2	Capacitors	SMD Multilayer Ceramic Capacitor, 1210 [3225 Metric], 0.1 µF, 25 V,		SMD				$0.30	0.6
1650910	C1210C105K5RAC-TU	8	Capacitors	SMD Multilayer Ceramic Capacitor, 1210 [3225 Metric], 1 µF, 50 V		SMD				$0.48	3.84
2112728	C3225X7S1H106K250AB	2	Capacitors	SMD Multilayer Ceramic Capacitor, 1210 [3225 Metric], 10 µF, 50 V		SMD				$1.39	2.78
9354301	3362P-1-103LF	12	Potentiometer	Trimmer Potentiometer, 10 kohm, 500 mW						$1.88	22.56
2437102	SN74LVC125APWR	1	NAND Gates	Buffer / Line Driver, 74LVC125, 1.65 V to 3.6 V, TSSOP-14	SOI	SMD	6.5	100ma	500mw	$0.19	0.185
2534283	NCP1117LPST33T3G	1	Voltage Reg	Fixed LDO Voltage Regulator, 18V in, 1.3V Dropout, 3.3V/1A out, SOT-223-3	SOT−223	SMD	18	1		$0.60	0.604
2533227	MBR120VLSFT3G	2	Schottky Diodes	Surface Mount Schottky Power Rectifier	SOD-123	SMD	20	1		$0.61	1.22
1822203	1206L035/16YR	1	Fuse	Resettable PTCs, Surface Mount > 1206L Series	1206	SMD		0.5		$0.54	0.542
9802541	P0102DA 1AA3	1	SCR (thyristor)	Thyristor for Crowbar protection	TO-92	T	600	5 to 200uA		$0.69	0.69
2133567	MK20DX128VFM5	1	Processor Main	ARM Microcontroller, K2 USB Series, Kinetis ARM Cortex-M4 Microcontrollers, 32bit, 50 MHz, 128 KB			3.6	155ma		$6.94	6.94
2133572	MK20DX128VLH5	1	Processor SDA	ARM Microcontroller, K2 USB Series, Kinetis ARM Cortex-M4 Microcontrollers, 32bit, 50 MHz, 128 KB			3.6	155ma		$7.56	7.56
2751667	1554266-1	1	USB Connector	USB Connector, Micro USB Type B, USB 2.0, Receptacle, 5 Ways, Surface Mount, Right Angle		SMD				$1.77	1.77
1515663	BLM21PG331SN1D	2	Ferrite Bead	Ferrite Bead, 330 ohm, 0805 [2012 Metric], BLM21P Series, 1.5 A,		SMD				$0.08	0.158
										0.0405	0.0405
9471898	DTSM-32S-B	1	switch	Tactile Switch, Non Illuminated, 12 V, 50 mA							1
											0
8554609	HSMG-C190.	5	LED	LED, Green, SMD, 0.8mm x 1.6mm, 25 mA, 2.2 V						$ 0.70	3.48
1208855		2		LED, Red, Through Hole						$ 0.77	1.536
Total Items:		83					Total Price:				$ 83.64

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=0BF00P4ECP402C0
8&field1=%f&field2=%f&field3=%f&field4=%f",CT_Current0,CT_Current1,CT_Curre
nt2,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>
```