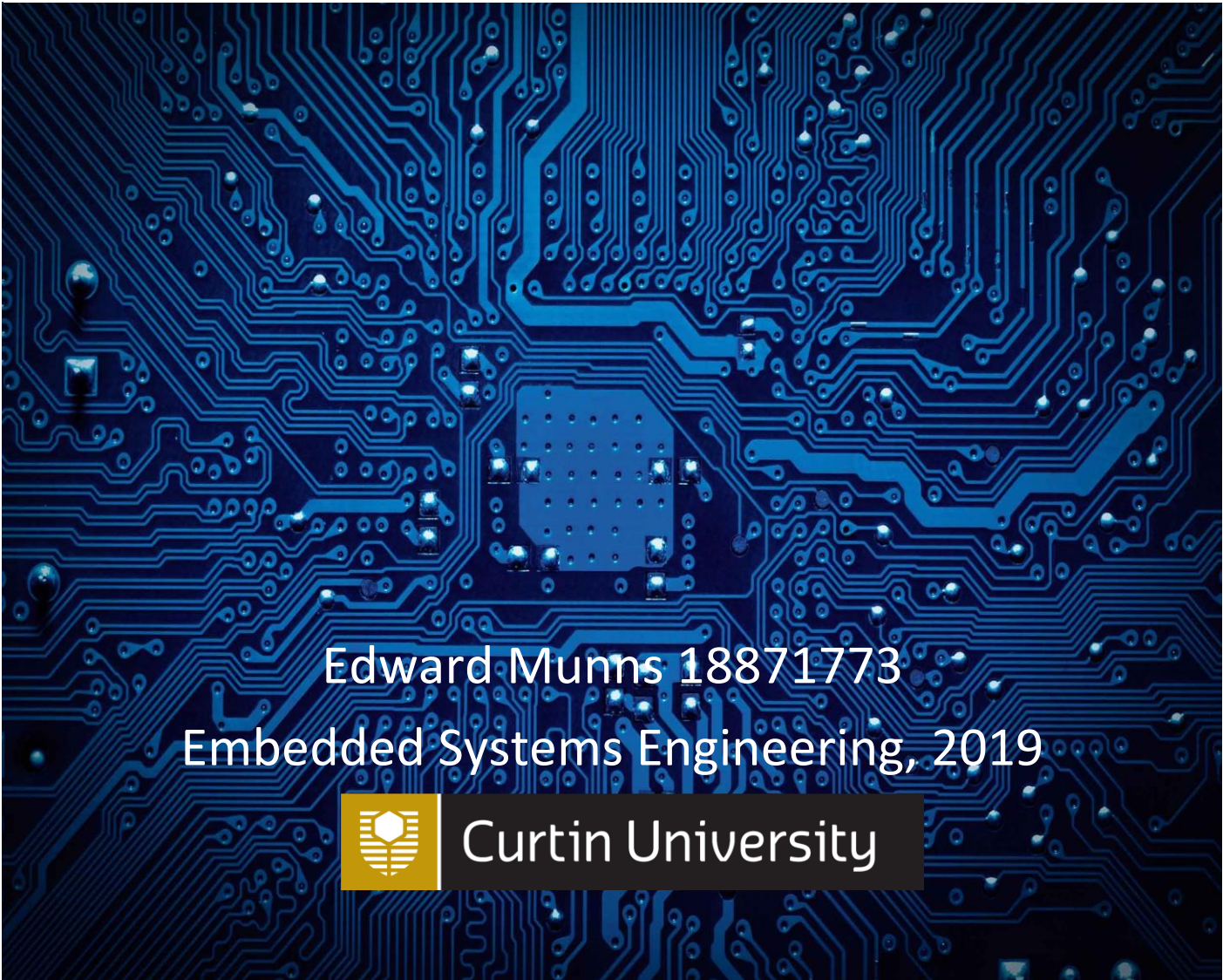




EMBEDDED SYSTEMS PROJECT REPORT.



Edward Munns 18871773
Embedded Systems Engineering, 2019



Curtin University

Contents

Table of Figures:	ii
1. Introduction:	1
2. Explanation of design and design requirements:	1
3. Proof of Concept Hardware Considerations:	3
3.1 Finding a Suitable Microcontroller:	3
3.2 The SIM5320A 3G module:	3
3.3 User Interface:	4
3.4 MAX6675 Temperature Sensing IC and K Type Thermocouple:	5
3.3 Schematic of Hardware Wiring:	5
4. Proof of Concept Software Considerations:	7
4.1 Microcontroller:	7
4.2 The SIM5320A 3G module:	8
4.2 Initial Flow Chart of Program:	9
4.4 Separation of Functions:	12
4.5 Pseudo-code of functions:	12
5. Proof of Concept Results:	15
6. Moving Beyond the Proof of Concept Phase:	15
7. Updated Software Considerations:	16
7.1 Microcontroller:	16
7.1.1 Watchdog timer:	16
7.1.2 Sleep Mode:	18
7.1.3 Further Power Saving Features:	19
7.1.4 Averaging Sample Size Calculation:	20
7.2 State Machine Implementation:	21
8. Updated Hardware Considerations:	24
8.1 Backup Battery Power circuit:	24
9. Fulfillment of Design Requirements and Constraints:	26
9.1 Power Requirements:	26
9.2 Weather Proofing:	29
9.3 Cost:	30
9.4 Maintainability:	30
10. Future development:	31
Conclusion:	31
Bibliography	32
Appendix:	33
Proof of concept Arduino IDE code:	33

Table of Figures:

Figure 1: Telstra 3G network coverage (Telstra 2019).	4
Figure 2: Schematic of proof of concept design.	6
Figure 3: AT commands for SIM5320A (AT command set 5320, SIMCOM 2014).....	8
Figure 4: Setup mode flow chart.	9
Figure 5: Main loop flow chart	10
Figure 6: Incident mode flow chart.	11
Figure 7: ATmega 2560 Watch dog timer register setup (ATmega 2560 datasheet page 65, Atmel 2019).....	16
Figure 8: Watchdog Timer modes (ATmega 2560 datasheet page 65, Atmel 2019).....	17
Figure 9: Watchdog Pre-scaler settings (ATmega 2560 datasheet page 66, Atmel 2019).	17
Figure 10: Available sleep modes on the ATmega 2560 (ATmega 2560 datasheet page 50, Atmel 2019).....	18
Figure 11: Sleep Mode Control Register (ATmega 2560 datasheet page 54, Atmel 2019).	18
Figure 12: Power reduction register 0 (ATmega 2560 datasheet page 55, Atmel 2019).	19
Figure 13: Power reduction register 1 (ATmega 2560 datasheet page 56, Atmel 2019).	20
Figure 14: Updated power supply system schematic.	25
Figure 15: ATmega 2560 power down current consumption (ATmega 2560 datasheet page 380, Atmel 2019). 26	
Figure 16: ATmega 2560 active current consumption (ATmega 2560 datasheet page 374, Atmel 2019).....	27
Figure 17: Max 6675 thermocouple IC electrical specifications (Max6675 datasheet, page 2, Maxim 2014).	28
Figure 18: NCP7800 specifications table (NCP7800 datasheet page 4, On semiconductor n.d.).	28
Figure 19: IP solids rating first digit (DSM&T 2019).....	29
Figure 20: IP liquids rating second digit (DSM&T 2019).	30
Figure 21: Proof of concept price list.....	30
Figure 22: Future development roadmap.	31

1. Introduction:

This document details the considerations made in the planning, design, proof of concept and prototyping phase of the “Cool Safe” embedded system. This report will first explain the design, design requirements and constraints. This will be followed by discussion of the first iteration of the design process, the “proof of concept” stage. This stage will then be analyzed with an intention to improve upon the satisfaction of design criteria and constraints in future iterations.

2. Explanation of design and design requirements:

This device is intended for wireless sensor monitoring in remote/regional areas with the only pre-requisite for operation being a connection to the 3G network. The prototype provided with this document is intended to monitor refrigeration systems and send text messages to stakeholders in the event of failure. The device first must be setup by the user with a temperature limit, number of contacts and phone numbers of these contacts to be stored in the EEPROM of the microcontroller for future access. While active, the device will monitor temperature and compare to the set limit. If this limit is breached the device will notify stakeholders via SMS messaging and inform them of the situation. The design is not limited to this application. There are many uses for remotely monitored sensors in regional Australia, perhaps a consequence of the extremely low population density. Examples of these include:

- Livestock feed and water monitoring
- Climate/Rainfall monitoring and datalogging
- Mechanical monitoring
- Feral animal monitoring
- Remote security applications

While this prototype deals with a simple temperature monitor, the idea behind this device is to implement an embedded system with remote area mobile networking capabilities for use in as wide a variety of applications as possible.

The first step in the design process is to clearly establish the design requirements and constraints of the project. This is an important step as the development of this design can best be implemented with an iterative design process with improvements made to systems and design outcomes with each iteration. Prioritized criterion and constraints are shown below.

Design Criterion:

1. Wireless connection through the available network in the target region.
2. The highest level of device reliability possible.
3. The lowest possible power consumption.
4. A high level of resistance to environmental conditions.
5. Ability to be implemented without the need for third party power supply.
6. Ability to be easily adapted for use in a diverse range of monitoring use cases.
7. An easy to operate user interface.

Design Constraints:

1. A prototyping build budget of under \$200 Australian.
2. A power consumption low enough to allow operation of at least 2 days in the event of power failure.
3. Must be able to withstand exposure to normal weather conditions (rain, dust, sunlight etc.) without malfunction.
4. Must be easily upgradeable/maintainable via USB connection to serial.

3. Proof of Concept Hardware Considerations:

3.1 Finding a Suitable Microcontroller:

The prototype built for this project is based around the ATmega2560 micro-controller, however this module was only chosen for the proof of concept phase as development boards and compatible hardware are readily available. This microcontroller was chosen over other development kits for the following reasons:

1. It has a reliable USART baud rate of 115200 for communication to the SIM5320A. This is the native baud rate of the module. This can be altered through AT commands, however matching a microcontroller to the baud of the module is a more desirable solution, particularly during prototyping.
2. Provision of a second hardware serial port capable of running at the same baud rate for use in terminal control/debugging of hardware and software components. This requirement will also allow future maintenance of software and firmware.
3. The design requires
 - 8 digital pins and 1 interrupt pin for the 16 key keypad
 - Power, wake and 2 serial pins for the SIM5320A 3G module
 - power and 3 digital pins for the MAX6675 thermocouple IC module
 - power and 2 digital pins for the I2C bus for communication to the 20 by 4 LCD display
 - a USART serial line for connection to USB

for a total of 25 IO pins, some with special functions, required to be accessible simultaneously. The ATmega2560 satisfies these requirements.

3.2 The SIM5320A 3G module:

To establish wireless communication a module operating at the correct frequency had to be sourced. The mobile network coverage in regional Western Australia is predominantly provided through Telstra (see figure 1). A large portion of this network runs on 3G technology in a frequency band of 850Mhz so a wireless 3G modem operating in this frequency band as well as a compatible sim card were required. The SIM5320A module satisfies this requirement and was available as a pre-assembled development board compatible with the Arduino development environment. This module requires only Vcc and ground connections, a digital pin for wake/sleep mode and a transmit (Tx) and receive (Rx) serial connection. The SIM5320A has a maximum burst current draw of 2 amps (Sim5320 Hardware design page 20, SIMCOM 2012). The system will therefore be required to be powered through the DC input jack of the ATmega development board.

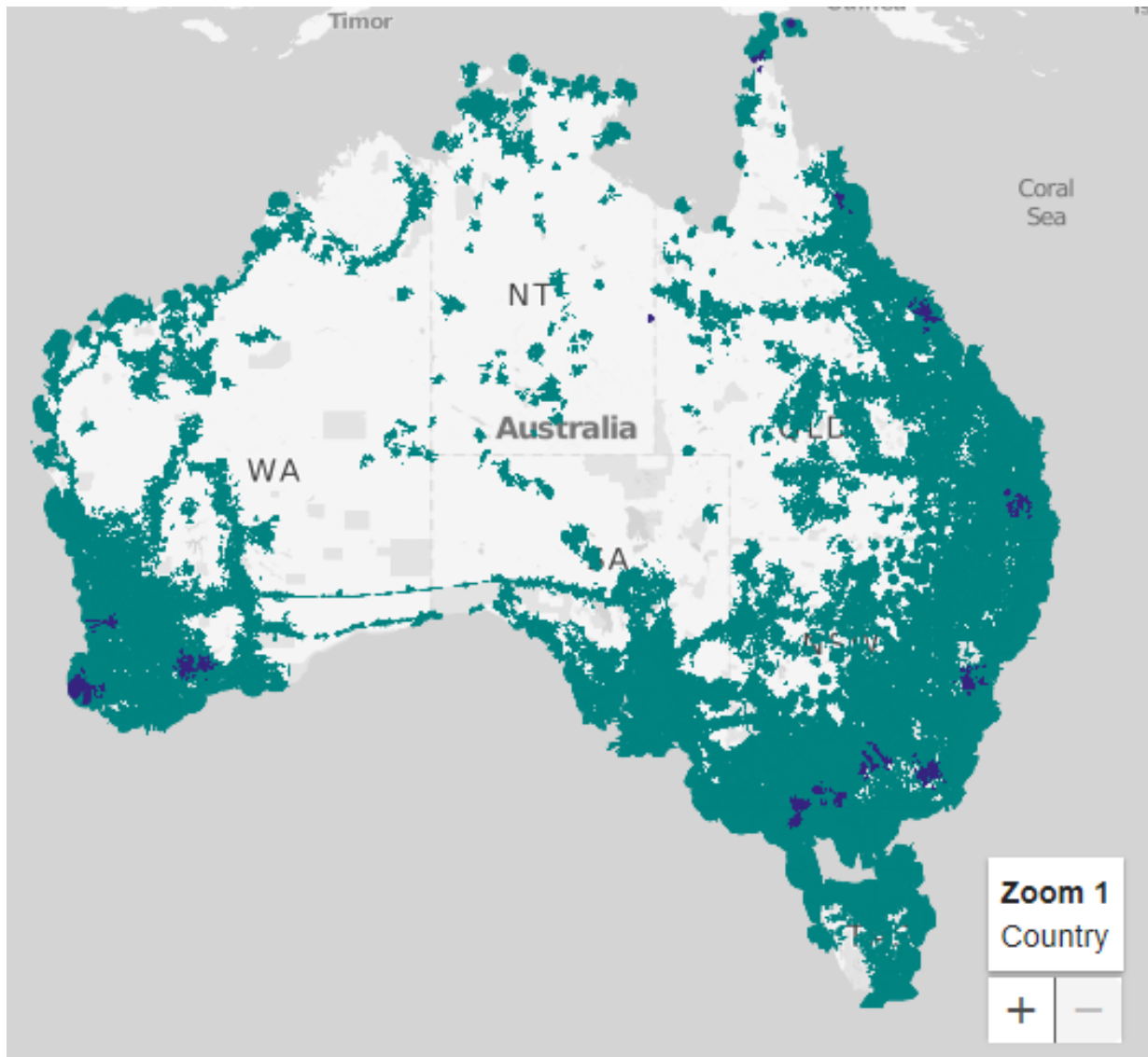


Figure 1: Telstra 3G network coverage (Telstra 2019).

3.3 User Interface:

To allow user input for setup of temperature limit and entering of contact phone numbers into the device a 16 key alphanumeric keypad was used. This device was chosen as user input requires digits from 0-9 for phone number input as well as auxiliary keys for acknowledge, delete and negative integer input in below zero applications. The alpha keys were required for these key functions.

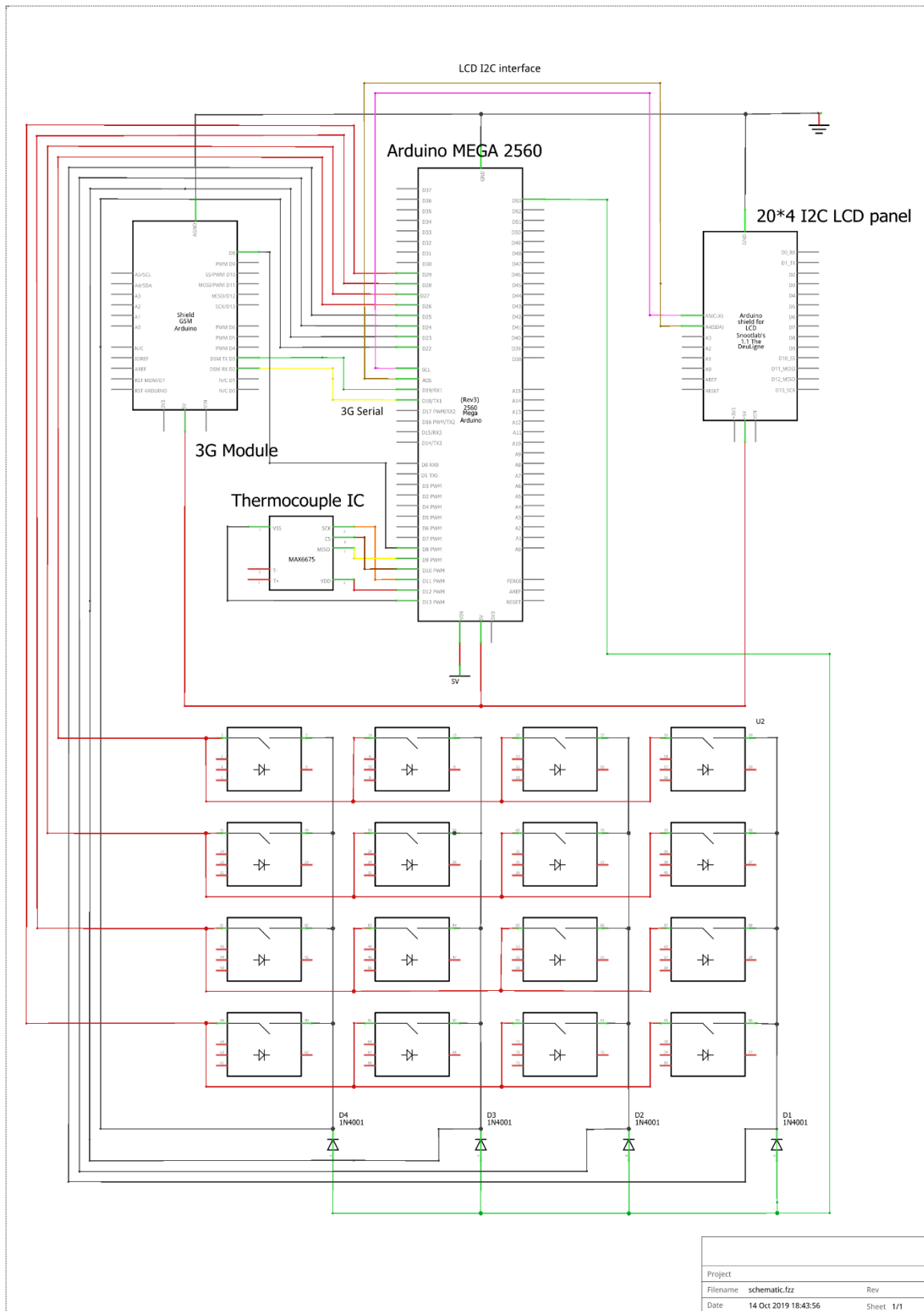
Output to the user was implemented via a 20 by 4 LCD panel. This size was chosen to allow lines of text to explain auxiliary key functions to the user during setup. This LCD panel was “piggy-backed” onto a PFC8574T I2C chip to operate through an I2C connection with the microcontroller. This interface type was chosen to reduce the required pin count for the LCD.

3.4 MAX6675 Temperature Sensing IC and K Type Thermocouple:

To measure temperatures a Maxim MAX6675 temperature sensing IC and a k type thermocouple were used. This solution was chosen over a standard analog temperature sensor to negate the need for calibration and for the large range of temperature ranges thermocouples are available to operate over.

3.3 Schematic of Hardware Wiring:

A schematic was drawn up in software to outline all hardware connections required for this phase of the project. This schematic can be seen below in figure 1.



fritzing

Figure 2: Schematic of proof of concept design.

4. Proof of Concept Software Considerations:

4.1 Microcontroller:

As this phase of the project is mainly concerned with integration of design components it was decided to program the microcontroller with the Arduino IDE. This IDE provides many part specific libraries and allows for fast prototyping. Pre-built Arduino libraries were sourced for the MAX6675 thermocouple module, the I2C lcd, the keypad and the EEPROM of the microcontroller.

4.2 The SIM5320A 3G module:

While there are many mobile communications modules available for embedded systems prototyping, many of these are GSM 2G based devices and are not supported by network providers in Australia. As the SIM5320A is a somewhat specialized device no reliable pre-made library could be sourced, so the implementation is briefly explained here. The module is controlled via a command set known as AT commands. This command set is designed specifically for use in mobile telecommunications devices. The command set is not standardized between all module models, so the specific command set document for the SIM5320A was sourced for interfacing between hardware. The commands used for this project are outlined below:

Command	Description	Response
"AT"	Ping modem	"OK"
"AT+CMGF?"	Get current value of SMS mode	"0"
"AT+CMGF=?"	Get possible values of SMS mode	"(0,1)"
"AT+CMGF=1"	Change SMS mode from PDU to text	"OK"
"AT+CROF"	Power off the module	"OK"
"AT+CSQ"	Get signal strength	"+CSQ:22,0"
"AT+IPREX=9600"	Sets serial baud rate to 9600 permanently	"OK"
"AT+CMGS<phone number>"	Prepares modem to send message to number	":"

Figure 3: AT commands for SIM5320A (AT command set 5320, SIMCOM 2014)

The first command in the table is the standard ping. This is used as a method to ensure the device is active. The next three commands give an example of the standard command syntax used with most variables, with a single "?" returning the current value of the variable, the "=?" returns all valid values the variable can be set to and "=1" setting the variable to 1. The +CMGF example was used as this must be changed from factory settings for the modem to function in this application. +IPREX changes the baud rate in the modem firmware permanently (SIMCOM 2014). This command was tested during prototyping to test if a lower baud rate could be used in future development. The last line is the send SMS command, this is a five-line process as follows:

1. Terminal: "AT+CMGS=<phone number>"<CR>
2. SIM5320: ":"
3. Terminal: "This is a message."<CTRL-Z>
4. SIM5320: "+CMGS: <message number> "
5. SIM5320: "OK"

Note in this process the <CTRL-Z> command refers to the substitute ASCII command (0x1A). As these commands can be implemented with string manipulation and sent over serial communication a basic library function set was created to establish networking capabilities via a serial connection (SIMCOM 2014).

To wake the SIM5320A from standby mode a pulse of 180ms must be applied to pin D8 of the development board (Sim5320 Hardware design page 25, SIMCOM 2012).

4.2 Initial Flow Chart of Program:

The flow chart of the program was split up into three sections, the user setup mode, the main loop mode and the incident mode. Note some functionality in these flow charts was omitted from the proof of concept phase for simplicity. Flow charts of these modes can be seen in the diagrams below.

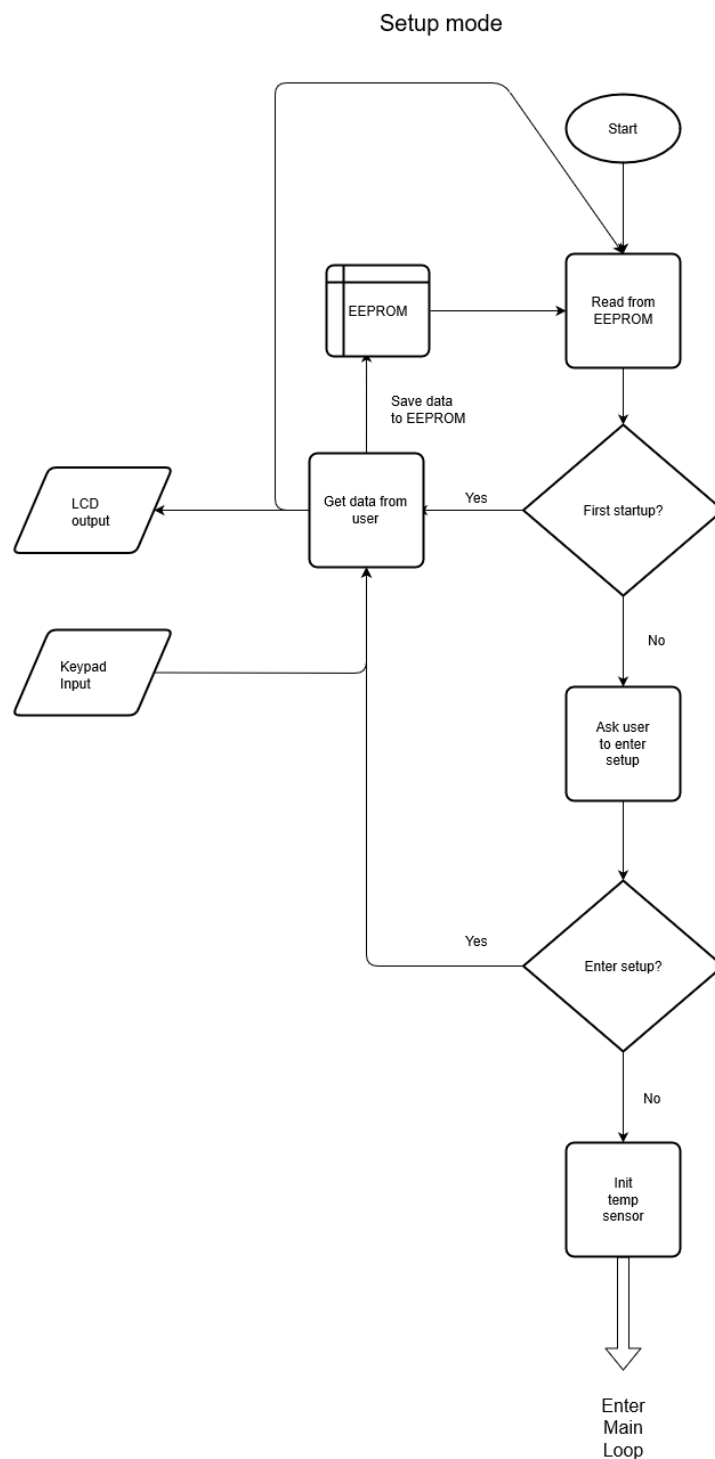


Figure 4: Setup mode flow chart.

Main Loop

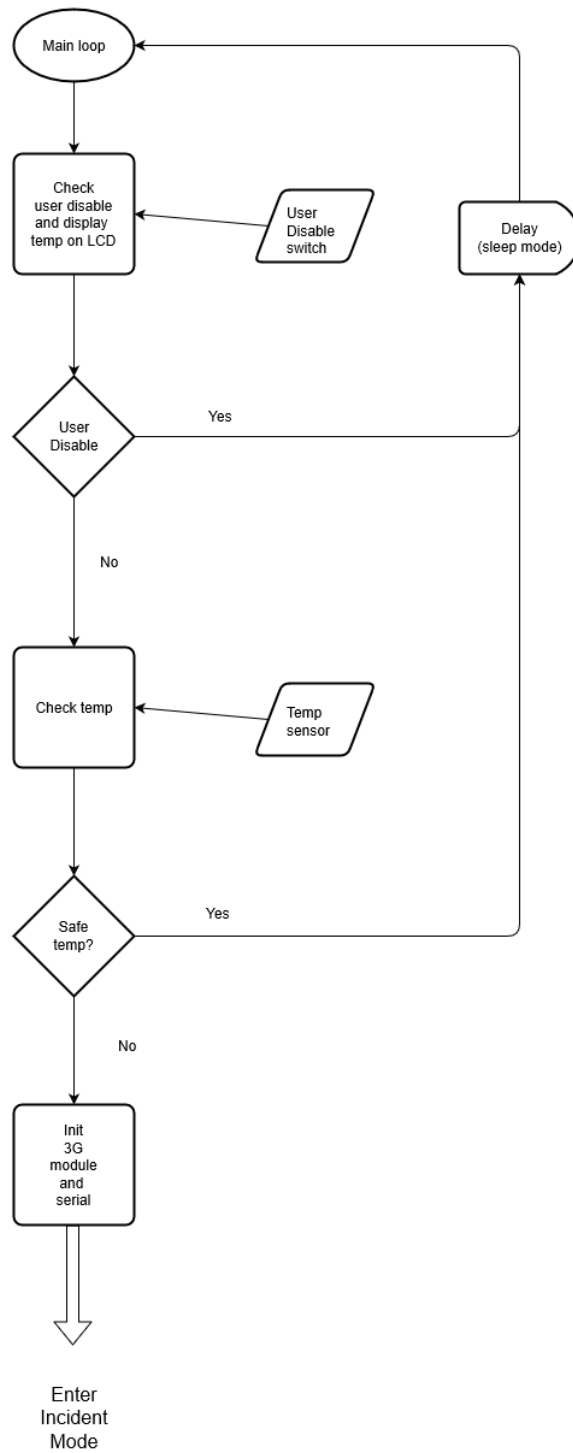


Figure 5: Main loop flow chart

Incident mode

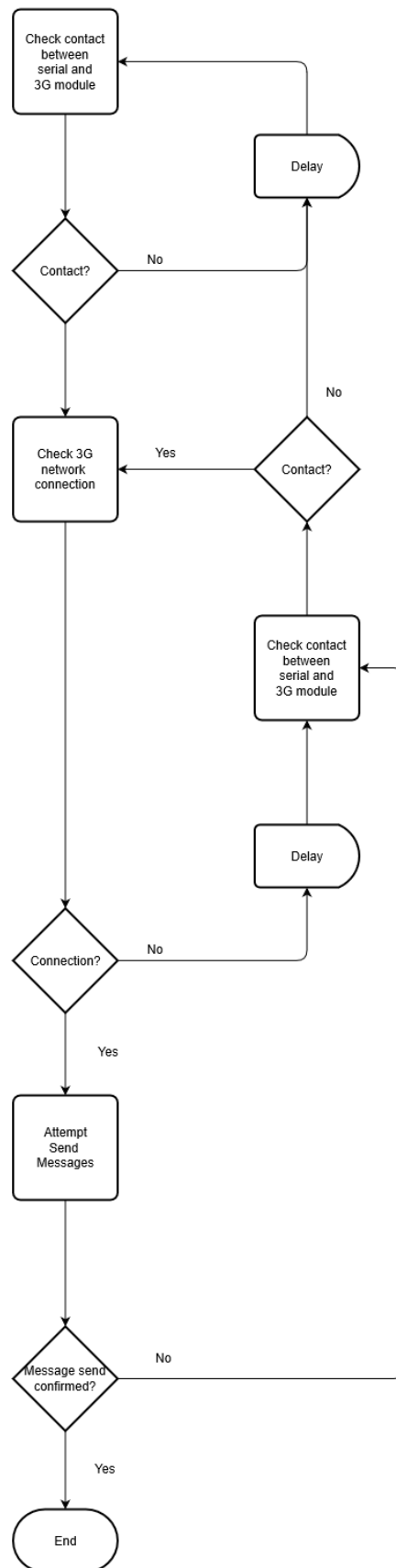


Figure 6: Incident mode flow chart.

4.4 Separation of Functions:

It was decided to separate the program into the following functions for this stage of the project:

1. Void setup(), contains all the setup required to initialize systems.
2. Void loop(), the endless loop to run the main section of the program.
3. Void firstTimeSetup(), allows the user to enter temperature limit, number of contacts and the contact phone numbers. This information is then stored in the EEPROM of the microcontroller. A struct was created in code to allow single get and put calls to EEPROM to streamline this process. This struct is a global variable to allow a single EEPROM read instruction in the setup function for use in the rest of the program.
4. Void simInit(), wakes the sim5320A from standby mode via a 180ms pulse on arduino pin D8.
5. Void sendTempWarnings(), sends messages to stored contacts in the event of an incident.

4.5 Pseudo-code of functions:

Void setup:

```
INITIALIZE serial
INITIALIZE lcd
INITIALIZE keypad
INITIALIZE thermocouple
OUTPUT enter setup prompt
i = 100
WHILE i > 0 AND NOT done DO
    INPUT user input
    IF INPUT IS '#' DO
        RUN firstTimeSetup
        done = TRUE
    END IF
    DELAY 100ms
    i = i - 1
END WHILE
GET settings from EEPROM
END setup
```

Void loop:

```
READ temperature from thermocouple
IF temperature > settings[temperature limit] AND NOT incident DO
    RUN sendTempWarnings
END IF
OUTPUT temperature
DELAY 200ms
END loop
```

Void simInit:

```
SET Sim module pin to HIGH
DELAY 180ms
SET Sim module pin to LOW
DELAY 3000ms
END simInit
```

Void sendTempWarnings:

```
RUN simInit
FOR i in settings[number of contacts] DO:
    FOR j in phone number length DO:
        phoneNumber[j] = settings[ phone_numbers[i][j] ]
    END FOR
    SEND message to phoneNumber
END FOR
Incident = TRUE
END sendTempWarnings
```

Void firstTimeSetup:

// get data for temperature limit from user and set in settings structure

OUTPUT temperature limit prompt

INPUT user input

SET settings[temperature limit] to INPUT

// get data for number of contacts from user and set in settings structure

OUTPUT number of contacts prompt

INPUT user input

SET settings[number of contacts] to INPUT

// get phone numbers from user and set in settings 2D array of phone numbers

FOR i in settings[number of contacts]

 OUTPUT phone number prompt

 WHILE j < phone number length DO

 INPUT user input

 IF INPUT is a digit DO

 SET settings[phone numbers[i][j]] to INPUT

 END IF

 IF INPUT IS 'D' DO

 j = j - 1

 DELETE settings[phone numbers[i][j]]

 END IF

 PREPEND Australian country code to settings[phone number]

 END WHILE

END FOR

PUT settings in EEPROM

END firstTimeSetup

Note, Arduino code of this pseudo-code can be seen in the appendix of this report.

5. Proof of Concept Results:

The proof of concept phase of this design successfully proved that integration of the ATmega 2560, Sim5320A 3G module, user interface and software could achieve the design brief. (note, this implementation was demonstrated in the microcomputers lab on Thursday 24th between 12 and 2pm with 100% awarded for the practical part of this project)

6. Moving Beyond the Proof of Concept Phase:

The initial proof of concept phase highlighted some areas of improvement that could be implemented to better satisfy the design requirements and constraints of the project. These factors will be discussed here to explain the changes in hardware and program flow implemented in the next iteration of the design. Once a functional prototype was built it was realized that all user input is carried out in the setup phase of program flow. This section of the program deals with storing data in the microcontroller EEPROM and it could be assumed that changes to this data would not regularly be required. For this reason, it was decided that the interrupt on the keypad input seen implemented with the diode array to the microcontroller in the schematic (figure 1) was not necessary and could be implemented better with a “halt program for key” design implemented within the user setup function.

Some thought was also given to the application itself and how the device could further be tuned to better satisfy the design requirements. As the device is essentially a safety device reliability must be measured both in the success of identifying incident cases and in distinguishing between these and “false positives”. An example of a false positive in this situation would be if the temperature of the refrigeration device went above the set limit for a short period of time. This situation is a common occurrence in use cases such as in a shop, where the refrigeration device can be opened continuously during busy sales periods only to return to a safe temperature level minutes later. This situation presents no harm to stock, therefore the constant sending of messages in these situations would constitute a failure of device reliability. To distinguish between these positive and false positive situations and to offer a layer of protection against erratic sensor readings it was decided to implement an averaging system based on a timer and an interrupt. Design of this system started with an estimate of a suitable time period of approximately 15 minutes. As this time period would require real world testing for final evaluation, the system was implemented to be easily tunable by simply adjusting the averaging sample size. Note this form of frequency manipulation may also make the device easier to implement in different use cases requiring longer measurement periods as it may not require timer registers to be adjusted.

Another important factor to examine was the situation of mains power loss as this could be a major cause of refrigeration failure. As the initial prototype was implemented for establishing basic “proof of concept” the addition of a rechargeable backup battery power system was incorporated into the design. An important factor to consider here is that of the worst-case scenario. This could be represented as a situation where an unattended below zero refrigeration device suffers an extended period of power loss. In this case the thermal mass of the frozen stock within, along with the refrigeration device insulation, could possibly hold a below zero temperature for several days. Taking this case, it was decided to implement a power saving mode on the microcontroller to extend the period of operating time should this situation occur. Another consideration made to improve the operating time period was the displaying of the temperature on the LCD display during operation.

While this function is a nice feature to incorporate this functionality could be a major factor on power draw. As reliability is the primary design requirement, it was decided that power draw should be of higher consideration in this application. To solve this issue a pin interrupt was incorporated to trigger on mains power loss and disable the LCD panel though the period of battery operation. This battery backup system could also be used in solar powered applications in cases where mains power connections are not available, and so this functionality may vastly improve the possible “off-grid” applications of the device going forward.

7. Updated Software Considerations:

7.1 Microcontroller:

7.1.1 Watchdog timer:

The watchdog timer was chosen to be used in this implementation because it allows the best-case power saving features of the ATmega 2560 when compared to other available timers (see 7.1.2, Sleep Mode). On the ATmega 2560 the WDT is setup with an 8-bit register named WDTCSR. A table of this register is shown below:

WDTCSR – Watchdog Timer Control Register

Bit (0x60)	7	6	5	4	3	2	1	0	
	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Figure 7: ATmega 2560 Watch dog timer register setup (ATmega 2560 datasheet page 65, Atmel 2019).

Bit explanation:

- WDIF: watchdog interrupt flag, set to 1 when watchdog timer times out.
- WDIE: watchdog interrupt enable, interrupt enabled on logic 1
- WDP3, WDP2, WDP1, WDP0: watchdog pre-scaler setting bits (note these bits are not consecutive)
- WDCE: watchdog change enable
- WDE: watchdog reset enable

For this implementation it was decided to operate the watchdog timer with the highest pre-scaler value of 1024 kilocycles (pre-scaler settings 1001). According to the 2560 documentation this should give a timeout length of approximately 8 seconds (see figure 7). The watchdog timer in this project is implemented in interrupt only mode with an interrupt service routine designed to wake the ATmega2560 from sleep.

$$\therefore \text{WDTSC} = (1 \ll \text{WDIE}) | (1 \ll \text{WDP3}) | (1 \ll \text{WDP0})$$

Table 12-1. Watchdog Timer Configuration

WDTON ⁽¹⁾	WDE	WDIE	Mode	Action on Time-out
1	0	0	Stopped	None
1	0	1	Interrupt Mode	Interrupt
1	1	0	System Reset Mode	Reset
1	1	1	Interrupt and System Reset Mode	Interrupt, then go to System Reset Mode
0	x	x	System Reset Mode	Reset

Figure 8: Watchdog Timer modes (ATMega 2560 datasheet page 65, Atmel 2019).

Table 12-2. Watchdog Timer Prescale Select

WDP3	WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V _{CC} = 5.0V
0	0	0	0	2K (2048) cycles	16ms
0	0	0	1	4K (4096) cycles	32ms
0	0	1	0	8K (8192) cycles	64ms
0	0	1	1	16K (16384) cycles	0.125s
0	1	0	0	32K (32768) cycles	0.25s
0	1	0	1	64K (65536) cycles	0.5s
0	1	1	0	128K (131072) cycles	1.0s
0	1	1	1	256K (262144) cycles	2.0s
1	0	0	0	512K (524288) cycles	4.0s
1	0	0	1	1024K (1048576) cycles	8.0s
1	0	1	0	Reserved	
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Figure 9: Watchdog Pre-scaler settings (ATMega 2560 datasheet page 66, Atmel 2019).

7.1.2 Sleep Mode:

The chart below outlines the available sleep modes and wake-up sources on the ATmega 2560:

Table 11-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

Sleep Mode	Active Clock Domains					Oscillators		Wake-up Sources						
	clk _{CPU}	clk _{FLASH}	clk _{IO}	clk _{ADC}	clk _{ASY}	Main Clock Source Enabled	Timer Osc Enabled	INT7:0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT Interrupt	Other I/O
Idle			X	X	X	X	X ⁽²⁾	X	X	X	X	X	X	X
ADCNRM				X	X	X	X ⁽²⁾	X ⁽³⁾	X	X ⁽²⁾	X	X	X	
Power-down								X ⁽³⁾	X				X	
Power-save					X		X ⁽²⁾	X ⁽³⁾	X	X			X	
Standby ⁽¹⁾						X		X ⁽³⁾	X				X	
Extended Standby					X ⁽²⁾	X	X ⁽²⁾	X ⁽³⁾	X	X			X	

Figure 10: Available sleep modes on the ATmega 2560 (ATmega 2560 datasheet page 50, Atmel 2019)

For implementation of sleep mode for the state when mains power is disconnected it was decided to use Power-down mode. This mode can be woken with a pin interrupt, TWI address match or the watchdog timer. The watchdog timer was used in this instance because it allows the disabling of all other timers for added power saving (see section 7.1.3).

11.10.1 SMCR – Sleep Mode Control Register

The Sleep Mode Control Register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	–	–	–	–	SM2	SM1	SM0	SE	SMCR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bits 3, 2, 1 – SM2:0: Sleep Mode Select Bits 2, 1, and 0

These bits select between the five available sleep modes as shown in Table 11-2.

Table 11-2. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby ⁽¹⁾
1	1	1	Extended Standby ⁽¹⁾

Figure 11: Sleep Mode Control Register (ATmega 2560 datasheet page 54, Atmel 2019).

∴ SMCR |= (1<<SM1)

7.1.3 Further Power Saving Features:

The ATmega 2560 microcontroller allows the disabling of unused features for power saving purposes via two power reduction registers, PRR0 and PRR1. The function of these registers is shown below.

PRR0 – Power Reduction Register 0

Bit (0x64)	7	6	5	4	3	2	1	0	
	PRTWI	PRTIM2	PRTIM0	–	PRTIM1	PRSPI	PRUSART0	PRADC	PRR0
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 12: Power reduction register 0 (ATmega 2560 datasheet page 55, Atmel 2019).

Bit explanation:

#	Code	Function	Setting
0	PRADC	Power off ADC	1
1	PRUSART0	Power off USART 0	0
2	PRSPI	Power off SPI bus	1
3	PRTIM1	Power off timer 1	1
4	Reserved		0
5	PRTIM0	Power off timer 0	1
6	PRTIM2	Power off timer 2	1
7	PRTWI	Power off TWI	1

$$\therefore \text{PRR0} = (1 \ll \text{PRTWI}) | (1 \ll \text{PRTIM2}) | (1 \ll \text{PRTIM0}) | (1 \ll \text{PRTIM1}) | (1 \ll \text{PRSPI}) | (1 \ll \text{PRADC})$$

PRR1 – Power Reduction Register 1

Bit	7	6	5	4	3	2	1	0	
(0x65)	–	–	PRTIM5	PRTIM4	PRTIM3	PRUSART3	PRUSART2	PRUSART1	PRR1
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 13: Power reduction register 1 (ATMega 2560 datasheet page 56, Atmel 2019).

Bit explanation:

#	Code	Function	Setting
0	PRUSART1	Power off USART 1	1
1	PRUSART2	Power off USART 2	0
2	PRUSART3	Power off USART 3	1
3	PRTIM3	Power off timer 3	1
4	PRTIM4	Power off timer 4	1
5	PRTIM5	Power off timer 5	1
6	Reserved		0
7	Reserved		0

$$\therefore \text{PRR1} = (1 \ll \text{PRTIM5}) | (1 \ll \text{PRTIM4}) | (1 \ll \text{PRTIM3}) | (1 \ll \text{PRUSART3}) | (1 \ll \text{PRUSART1})$$

7.1.4 Averaging Sample Size Calculation:

To calculate an appropriate sample size for the averaging system:

Estimated appropriate time = 15 minutes = 900 seconds

Single cycle time = 1 second for reading + 8 seconds for delay/sleep = 9 seconds

$$\therefore \text{Estimated sample size} = 900/9 = 100 \text{ samples.}$$

7.2 State Machine Implementation:

To implement the changes discussed in section 6 of this report it was decided that a state machine should be used in the main loop. The operation of this state machine is demonstrated with the pseudo-code below.

```
STATES = [lcd_flip, read_temp, calc_average, delay, sleep_on_power_loss, incident]
```

```
state = STATES[read_temp]
```

```
power_loss_flag = false
```

```
sum = 0
```

```
sampleSize = 100
```

```
Void loop
```

```
ENABLE_INTERRUPTS
```

```
SWITCH based on state
```

```
// Power pin state has just been changed, invert the power status of the LCD
```

```
    CASE lcd_flip:
```

```
        IF power_loss_flag DO
```

```
            POWER_OFF lcd
```

```
        ELSE
```

```
            POWER_ON lcd
```

```
        END IF
```

```
        state = read_temp
```

```
    END CASE
```

```
// beginning of a cycle, read the temperature and decide which state to go to next
```

```
CASE read_temp:
    POWER_ON max6675
    READ temperature
    POWER_OFF max6675
    sum = sum + temperature
    sampleNumber = sampleNumber + 1
    IF sampleNumber = sampleSize DO
        state = calc_average
    ELSE
        IF power_loss_flag DO
            state = sleep_on_power_loss
        ELSE
            state = delay
        END IF
    END IF
END CASE
```

```
// sample size has been reached, calculate average and compare to temp_limit then decide on next state
```

```
CASE calc_average:
    average = sum/sampleSize
    sampleNumber = 0
    sum = 0
    IF average > settings[temperature_limit] DO
        DISABLE_INTERRUPTS
        state = incident
    ELSE
        IF power_loss_flag DO
            state = sleep_on_power_loss
        ELSE
            state = delay
        END IF
    END IF
END CASE
```

```
// power in on, start watchdog timer and wait until timeout
```

```
    CASE delay:
```

```
        START watchdog_timer
```

```
        WHILE infinite
```

```
    END CASE
```

```
// power is off, start watchdog timer, enter sleep mode and wait until watchdog timer timeout
```

```
    CASE sleep_on_power_loss:
```

```
        START watchdog_timer
```

```
        SLEEP
```

```
        WHILE infinite
```

```
    END CASE
```

```
// average is greater than temp_limit, send out messages.
```

```
    CASE incident:
```

```
        RUN sendTempWarnings
```

```
    END CASE
```

```
END SWITCH CASE
```

```
INTERRUPT based on watch_dog_timer (timeout)
```

```
    STOP watchdog_timer
```

```
    RESET watchdog_timer
```

```
    state = read_temp
```

```
END INTERRUPT
```

```
INTERRUPT based on power_interrupt_pin (change)
```

```
    state = lcd_flip
```

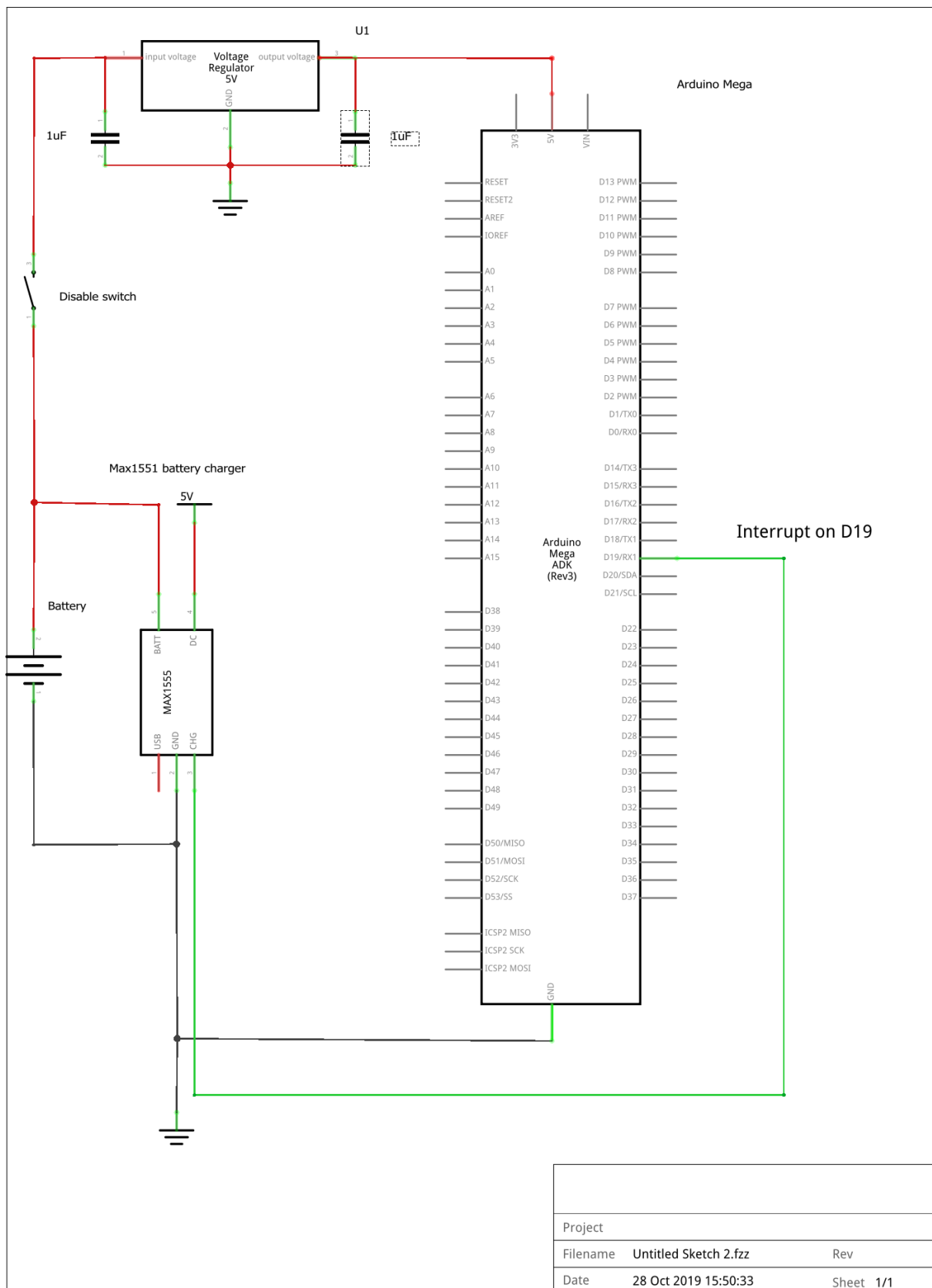
```
    power_loss_flag = NOT power_loss_flag
```

```
END INTERRUPT
```


8. Updated Hardware Considerations:

8.1 Backup Battery Power circuit:

To incorporate battery backup and allow sensing of loss of mains power a selection of a max 1551 or a max 1555 lithium battery charger can be paired with a 7.4-volt lithium ion battery with a capacity of 3.45-amp hours. The max 1551 battery charging IC incorporates a logic circuit that is driven low upon disconnection of the DC input power while the max 1555 offers a logic circuit based on charging on or off. The max 1551 would be suitable for sensing mains power loss while the 1555 model may be applied to solar applications for power saving periods while no sunlight is available. This logic circuit was setup on an interrupt on Arduino pin number D19 of the ATmega 2560 (Atmel pin number PD3). This pin has a hardware interrupt setup on INT3 to allow the microcontroller to sense power loss. This interrupt will then trigger the program to enter LCD off and sleep mode to extend the operational period under these circumstances. This will require a swap of the SIM5320A serial channel from USART1 to USART2 on the microcontroller, lowering the interrupt priority of this serial line. While not an ideal situation, it should be noted that priority of serial communications will switch from Terminal to SIM5320A as development continues. As the selected battery for trial in this iteration of the design operates at 7.4 volts a 5-volt NCP7800 voltage regulator and supply circuit was required to distribute voltage to all onboard components. This voltage regulator has a peak output current of 2.4 amps (see figure 16). A disable switch was also incorporated to allow the system to be manually isolated from power by the user. This provides a simpler and more effective implementation of the user disable interrupt shown in the main loop flow chart (see figure 3). A schematic of this system can be seen in figure 7 of this report.



fritzing

Figure 14: Updated power supply system schematic.

9. Fulfillment of Design Requirements and Constraints:

9.1 Power Requirements:

With the known battery capacity and an estimate of systems power usage during the loss of mains power operating state it is possible to estimate a time period for the systems involved in this state of the program. Note this value is an estimate only and will require real world testing to accurately evaluate, however this estimation will give a rough idea of the suitability of the chosen components.

Battery amp hours = 3.45Ah = 3450 milliamp hours

The following table gives an estimate of the current draw of the ATmega 2560 in power-down mode with the watchdog timer enabled. Assuming 5 volts and a 25-degree temperature a value of 7 micro amps was taken from the chart for this estimate.

Figure 32-12. Power-down Supply Current vs. V_{CC} (Watchdog Timer Enabled)

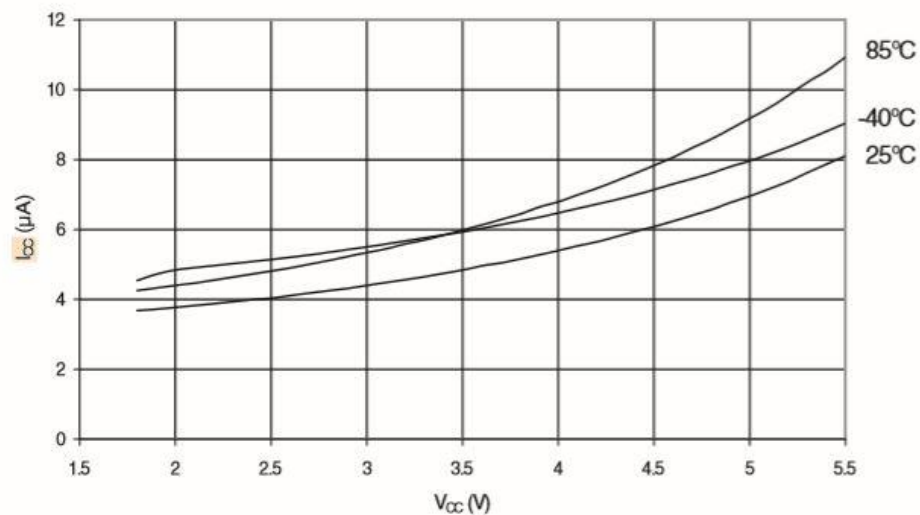


Figure 15: ATmega 2560 power down current consumption (ATmega 2560 datasheet page 380, Atmel 2019).

The next table gives an estimate of the current draw of the ATmega2560 in active mode. Assuming a frequency of 16Mhz and a voltage of 5-volts an estimate of 21-milliamps was taken from this chart.

Figure 32-2. Active Supply Current vs. Frequency (1MHz - 16MHz)

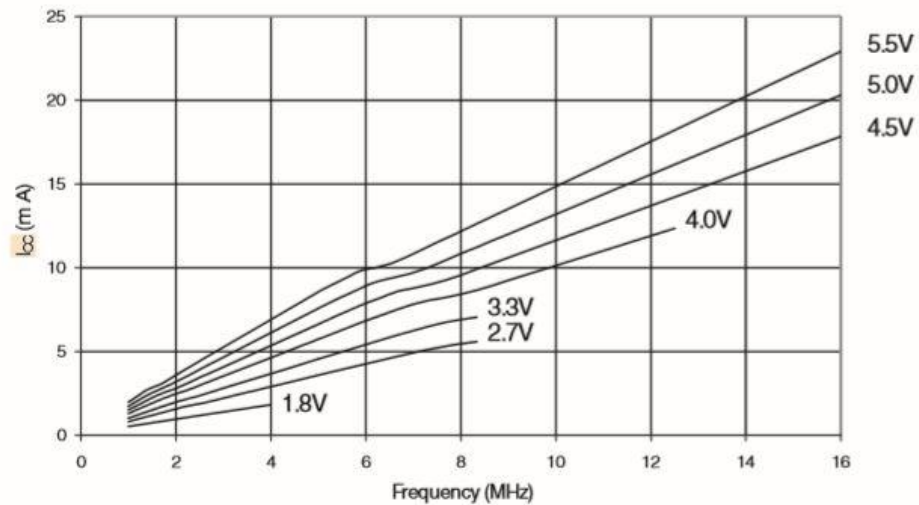


Figure 16: ATmega 2560 active current consumption (ATmega 2560 datasheet page 374, Atmel 2019).

To calculate a value for the total current consumption of the ATmega 2560 a ratio of active to power-down time must be calculated. To estimate this average an active on-time period of 1 second was used. Note this time period could be far beyond the time period the microcontroller is in active mode for each program loop cycle, however this over-estimate is applied to allow for re-initialization of the Max6675 thermocouple module as this may require a delay and will add a layer of safety in ensuring the chosen battery capacity is adequate for the task.

$$\text{Average } I_{cc \text{ ATmega}} = \frac{7\mu A \times 8 \text{ seconds} + 21mA \times 1 \text{ second}}{9 \text{ seconds}} = 2.339mA$$

SIM5320A 850Mhz in sleep mode max current consumption = 4.5mA (Sim5320 hardware design page 59, SIMCOM 2012).

The following table shows the specifications of the Max6675 thermocouple IC. Note the SO Current of this device is 50mA, this is a high value and will affect power saving features significantly. An investigation into alternatives should be conducted in future iterations.

Absolute Maximum Ratings

Supply Voltage (V _{CC} to GND)	-0.3V to +6V	Storage Temperature Range	-65°C to +150°C
SO, SCK, CS, T-, T+ to GND	-0.3V to V _{CC} + 0.3V	Junction Temperature	+150°C
SO Current	50mA	SO Package	
ESD Protection (Human Body Model)	±2000V	Vapor Phase (60s)	+215°C
Continuous Power Dissipation (T _A = +70°C)		Infrared (15s)	+220°C
8-Pin SO (derate 5.88mW/°C above +70°C)	471mW	Lead Temperature (soldering, 10s)	+300°C
Operating Temperature Range	-20°C to +85°C		

Figure 17: Max 6675 thermocouple IC electrical specifications (Max6675 datasheet, page 2, Maxim 2014).

$$\text{Max 6675 average current consumption} = \frac{50\text{mA} \times 1\text{ second}}{9\text{seconds}} = 5.55\text{mA}$$

The NCP7800 voltage regulator has Quiescent maximum (worst-case) current consumption of 8 mA, this is the no-load current consumption of the unit and should give an estimate of current usage for this situation.

NCP7800

Table 5. ELECTRICAL CHARACTERISTICS (V_{in} = 10 V, I_O = 500 mA, T_J = 0°C to 125°C, unless otherwise noted) (Note 7)

Characteristic	Symbol	NCP7805			Unit
		Min	Typ	Max	
Output Voltage (T _J = 25°C)	V _O	4.8	5.0	5.2	Vdc
Output Voltage (5.0 mA ≤ I _O ≤ 1.0 A, P _D ≤ 0.5 W) 7.0 Vdc ≤ V _{in} ≤ 20 Vdc	V _O	4.75	5.0	5.25	Vdc
Line Regulation (T _J = 25°C) 7.5 Vdc ≤ V _{in} ≤ 20 Vdc 8.0 Vdc ≤ V _{in} ≤ 12 Vdc	Reg _{line}	–	1.1 0.6	100 50	mV
Load Regulation (T _J = 25°C) 5.0 mA ≤ I _O ≤ 1.0 A 5.0 mA ≤ I _O ≤ 1.5 A	Reg _{load}	–	1.5 2.9	50 100	mV
Quiescent Current (T _J = 25°C)	I _B	–	3.0	8	mA
Quiescent Current Change 7.0 Vdc ≤ V _{in} ≤ 25 Vdc 5.0 mA ≤ I _O ≤ 1.0 A	ΔI _B	–	0.28 0.07	1.3 0.5	mA
Ripple Rejection (Note 8) 8.0 Vdc ≤ V _{in} ≤ 18 Vdc, f = 120 Hz	RR	62	75	–	dB
Dropout Voltage (I _O = 1.0 A, T _J = 25°C) (Note 8)	V _I – V _O	–	2.0	–	Vdc
Output Noise Voltage (T _J = 25°C) (Note 8) 10 Hz ≤ f ≤ 100 kHz	V _n	–	6.8	–	μV/V _O
Output Resistance f = 1.0 kHz (Note 8)	r _O	–	2.2	–	mΩ
Short Circuit Current Limit (T _J = 25°C) (Note 8) V _{in} = 35 Vdc	I _{sc}	–	0.3	–	A
Peak Output Current (T _J = 25°C) (Note 8)	I _{max}	–	2.4	–	A
Average Temperature Coefficient of Output Voltage (Note 8)	TCV _O	–	0.13	–	mV/°C

7. Performance guaranteed over the indicated operating temperature range by design and/or characterization, production tested at T_J = 25°C. Low duty cycle pulse techniques are used during testing to maintain the junction temperature as close to ambient as possible.
8. Value based on design and/or characterization.

Figure 18: NCP7800 specifications table (NCP7800 datasheet page 4, On semiconductor n.d.).

$$\sum I_{average} = I_{ATMega} + I_{SIM5320A} + I_{Max6675} + I_{vr} = 2.339mA + 4.5mA + 5.55mA + 8mA \approx 20.4mA$$

$$\frac{3450mAh}{20.4mA} = 169 \text{ hours} \approx 7 \text{ days.}$$

This value gives an indication that the battery selected may be oversized for the purpose, however this situation is preferred during development as real-world testing is required to accurately evaluate this figure with all sub-systems in operation. There are many untested factors involved in this calculation including the power draw of uncalculated auxiliary components and the accuracy of the stated battery capacity. Battery capacity may also decrease with age so this battery should be a good starting point.

9.2 Weather Proofing:

In order to weather-proof the design to allow for open air operation in varying weather conditions a suitable enclosure would need to be sourced. Electronics enclosures are rated for this factor with a grading system known as an IP rating. This rating is given as a 2-digit number with the first digit designating the level of protection against solid materials such as dust and the second digit rating the protective capacity of the enclosure against liquids. The following charts outline this ratings system:

First Digit, solids rating:

Level	Object size protected against
0	No protection
1	>50mm
2	>12.5mm
3	>2.5mm
4	>1mm
5	Dust protected
6	Dust Proof

Figure 19: IP solids rating first digit (DSM&T 2019).

Second Digit, liquids rating:

Level	Object size protected against
0	No protection
1	Dripping water
2	Dripping water when tilted 15 degrees
3	Spraying water
4	Splashing water
5	Water jets
6	Powerful water jets
7	Immersion up to 1 meter
8	Immersion beyond 1 meter

Figure 20: IP liquids rating second digit (DSM&T 2019).

After review of this information it was decided an enclosure of an IP rating of 54 or greater would be suitable for this project as this should provide protection against heavy rain and reasonable protection against dust. An Aluminium enclosure would also be preferable over a plastic one to avoid long term damage from UV degradation of plastic.

9.3 Cost:

One of the main constraints of the project was that the budget for the proof of concept unit be manufacturable for under \$200 Australian. In order to evaluate this constraint a price list of required parts was constructed with various sources evaluated. This price list is shown below.

Part #	Quantity	Description	Supplier	Price
1	1	Arduino ATmega 2560 clone	EBAY supplier	\$14.99
2	1	SIMCOM 5320A Arduino shield	tinyosshop	\$86.69
3	1	20 × 4 I2C LCD panel	EBAY supplier	\$5.99
4	1	4 × 4 matrix keypad	EBAY supplier	\$5.29
5	1	MAX 6675 IC and thermocouple	EBAY supplier	\$4.99
6	1	145 × 121 × 39 IP54 aluminium enclosure	EBAY supplier	\$21.29
TOTAL				\$139.24

Figure 21: Proof of concept price list.

9.4 Maintainability:

As the device is designed to be easily implemented to different use cases it was decided to retain a USB connection to the terminal serial line of the ATmega 2560. This will allow reprogramming of the device for use with alternative sensors and provide a communication line to both the ATmega 2560 and the SIM 5320A module should future firmware upgrades be required.

10. Future development:

In order to plan for the project going forward the following roadmap of future design iterations and the goals of these iterations was created. This roadmap is a basic plan only but sets some goals for future development of the project. This roadmap can be seen in figure 20.

Iteration	Purpose of this iteration
1	Proof of concept, testing of software for improvements to program function
2	Transfer software to Atmel studio, Implement battery backup system
3	Real world testing of hardware and software systems
4	Move hardware to purpose-built PCB, create final parts list and pricing, final testing
5	Evaluate manufacturing options, continue maintenance

Figure 22: Future development roadmap.

Conclusion:

The iterations of the design presented in this report have been successful in satisfying the initial design requirements. The proof of concept phase has been completed and analysis of this phase has been used to define changes that will be made in the second iteration of the design process. These changes include the transferring of software programming from the Arduino IDE to Atmel studio, the discussed implementation of the backup battery system, the requirements of the device enclosure and the averaging system implemented in software to increase the reliability of the design. These changes should improve the device function to the point where real world testing can start to be carried out. This testing should highlight areas of the design that require further improvements going forward. The constraints highlighted in section 2 of this report have been met or planned for in iteration two. Improvements to the criterion of the design will continue to be carried out through iterations two, three and four with the ability to test the device in final application environments being the main driver for design improvements going forward.

Bibliography

- Atmel. 2019. "ATmega 2560 datasheet." *www.microchip.com*. Accessed 10 29, 2019.
ww1.microchip.com/downloads/en/deviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf.
- DSM&T. 2019. "IP Rating Chart ." *www.dsmst.com*. Accessed 20 29, 2019.
www.dsmst.com/resources/ip-ratings-chart/.
- Maxim. 2014. "Max6675 datasheet." *www.farnell.com*. Accessed 10 29, 2019.
www.farnell.com/datasheets/2079272.pdf?_ga=2.83046549.20991621.1572243983-497523684.1488250959.
- On semiconductor. n.d. "NCP7800 datasheet." *www.farnell.com*. Accessed 10 29, 2019.
www.farnell.com/datasheets/2118309.pdf?_ga=2.8977718.20991621.1572243983-497523684.1488250958.
- SIMCOM. 2014. "AT command set SIMCOM_SIM5320_ATC_EN_V2.02." *tinyosshop*. Accessed 11 2, 2019. https://www.tinyosshop.com/datasheet/SIMCOM_SIM5320_ATC_EN_V2.02.pdf.
- . 2012. "SIM5320A hardware design." *tinyosshop*. Accessed 20 29, 2019.
www.tinyosshop.com/datasheet/SIM5320_Hardware+Design_V1.07.pdf.
- Telstra. 2019. *Telstra network services*. Accessed 11 2, 2019. <https://www.telstra.com.au/coverage-networks/our-coverage>.

Appendix:

Proof of concept Arduino IDE code:

```
#include <Key.h>

#include <Keypad.h>

#include <EventBuffer.h>

#include <max6675.h>

#include <LiquidCrystal_I2C.h>

#include <Wire.h>

#include <EEPROM.h>


#define SIM5320A_ON 8 // arduino pin to wake/sleep SIM unit

#define MAX_NUM_CONTACTS 5


// Settings struct definition:

typedef struct Settings
{
    int temp_limit;

    int number_contacts;

    char phone_numbers[MAX_NUM_CONTACTS][11];
}Settings;


// Global variables:

Settings settings;

HardwareSerial & Terminal = Serial;

HardwareSerial & SimSerial = Serial1;

volatile byte eepromIndex = 0;

volatile bool incident = false;

volatile char key = ' ';
```

```

// Max 6675 thermocouple IC pin setup:

int thermo_gnd = 46;

int thermo_vcc = 44;

int thermo_sck = 42;

int thermo_cs = 40;

int thermo_so = 38;

MAX6675 thermocouple(thermo_sck, thermo_cs, thermo_so);


// initialize LCD, I2C IC is a PCF8574T, these use I2C address 0x27 as standard, this panel is 20*4:

static LiquidCrystal_I2C lcd(0x27,20,4);


// keypad setup:

const byte ROWS = 4;

const byte COLS = 4;


// define the keypad matrix

char keys[ROWS][COLS] = {{ '1','2','3','A'},
                           { '4','5','6','B'},
                           { '7','8','9','C'},
                           { '*', '0', '#', 'D' } };


// define keypad pins

byte rowPins[ROWS] = {26, 27, 28, 29};

byte colPins[COLS] = {22, 23, 24, 25};


// create keypad object

Keypad myKeypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS);


void setup()
{
    // Thermocouple IC:

    pinMode(thermo_vcc, OUTPUT);

    pinMode(thermo_gnd, OUTPUT);

    digitalWrite(thermo_vcc, HIGH);

    digitalWrite(thermo_gnd, LOW);

    delay(500);

```

```

char line3[20];

// initalize serial:
Terminal.begin(115200);
SimSerial.begin(115200);

lcd.begin();

lcd.backlight();

lcd.setCursor(0,0);

lcd.print("**** Cool Safe ****");

lcd.setCursor(0,1);

lcd.print("*   Press # key   *");

lcd.setCursor(0,2);

lcd.print("*   to enter setup *");


// loop for 10 seconds and wait for user input
int i = 100;

bool done = false;

while(i > 0 && !done)
{
    key = myKeypad.getKey();

    if(key == '#')
    {
        firstTimeSetup();

        done = true;
    }

    lcd.setCursor(0,3);

    sprintf(line3, "****   %2d   ****", (int)(i/10));

    lcd.print(line3);

    delay(100);

    i--;
}

lcd.clear();

lcd.setCursor(0,0);

delay(200);
}

```

```

void loop()
{
    char line3[20];

    settings = EEPROM.get(eepromIndex, settings);

    // the fuction readCelcius in the MAX6675 library returns double,
    // put this in integer format:
    int temp = thermocouple.readCelsius()*100;
    int deg = temp/100;
    int dp = temp%100;

    // If statement for initiation of incident code.
    if(deg > settings.temp_limit && !incident)
    {
        sendTempWarnings();
    }

    lcd.setCursor(0,0);
    lcd.print("**** Cool Safe ****");
    lcd.setCursor(0,1);
    lcd.print("* Current temp: *");
    lcd.setCursor(0,2);
    lcd.print("*          *");
    lcd.setCursor(0,3);
    sprintf(line3, "**** %d.%02d ****", deg, dp);
    lcd.print(line3);
    delay(200);
}

```

```

void firstTimeSetup()
{
    // initalize blank settings struct
    settings = {30, 1, {{'0','0','0','0','0','0','0','0','0','0','0'},
                        {'0','0','0','0','0','0','0','0','0','0','0'},
                        {'0','0','0','0','0','0','0','0','0','0','0'},
                        {'0','0','0','0','0','0','0','0','0','0','0'},
                        {'0','0','0','0','0','0','0','0','0','0','0'}}};
}

```

```

// setup temperature limit

signed int scaler = 1;

char temp[3] = "";

int ii = 0;

key = ' ';

lcd.clear();

lcd.setCursor(0,0);

lcd.print("Please enter temp");

lcd.setCursor(0,1);

lcd.print("Press A when done");

lcd.setCursor(0,2);

lcd.print("* = neg, D = del");

lcd.setCursor(0,3);


while(key != 'A')
{
    key = myKeypad.waitForKey();

    if(key == '*' && ii == 0)
    {
        scaler = -1;

        lcd.print("-");
    }

    if(key == 'D')
    {
        temp[ii] = ' ';

        lcd.setCursor(ii,3);

        lcd.print(" ");

        lcd.setCursor(ii,3);

        if(ii > 0)
        {
            ii--;
        }
    }
}

```



```

        if(isDigit(key))
        {
            temp[ii] = key;
            lcd.print(key);
            ii++;
        }
    }

    // change the char to int with atoi then multiply by scaler for negative values.
    settings.temp_limit = atoi(temp) * scaler;
    Terminal.print(settings.temp_limit);
    Terminal.print("\n");

    // setup number of contacts
    int num_contacts = 0;
    key = ' ';
    char nums[1] = "";
    ii = 0;
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Enter # contacts");
    lcd.setCursor(0,1);
    lcd.print("Press A when done");
    lcd.setCursor(0,2);
    lcd.print("Max = 5, D = del"); // TODO sprintf statement to show max defined contacts
    lcd.setCursor(0,3);

    // while loop for user input
    while(key != 'A')
    {
        key = myKeypad.waitForKey();

        if(key == 'D')
        {
            lcd.setCursor(ii,3);
            lcd.print(" ");

```

```

        lcd.setCursor(ii,3);

        if(ii > 0)
        {
            ii--;
        }
    }

    if(isDigit(key))
    {
        nums[ii] = key;
        lcd.print(key);
        ii++;
    }
}

settings.number_contacts = atoi(nums);
Terminal.print(settings.number_contacts);
Terminal.print("\n");

// setup phone numbers
for(int ii = 0; ii < settings.number_contacts; ii++)
{
    char number[11];

    lcd.clear();

    lcd.setCursor(0,0);
    lcd.print("Enter phone number");

    lcd.setCursor(0,1);
    lcd.print("Press A when done");

    lcd.setCursor(0,2);
    lcd.print("D = del");

    lcd.setCursor(0,3);

    // get data from user
    //for(int jj = 0; jj < 11; jj++)
        int jj = 0;

        while(jj < 11)
        {

```

```

// this function pauses program until a key is pressed.
key = myKeypad.waitForKey(); // wait for key entry

// if the key pressed is a digit
if(isDigit(key))
{
    // add digit to number
    number[jj] = key;

    // print digit to lcd
    lcd.print(key);

    jj++;
}

// delete last digit
else if(key == 'D')
{
    jj -= 1;

    lcd.setCursor(jj,3);

    lcd.print(' ');

    lcd.setCursor(jj,3);
}

// acknowlege number done
else if(key == 'A')
{
    // insert Australian country code
    settings.phone_numbers[ii][0] = '6';
    settings.phone_numbers[ii][1] = '1';
    Terminal.print(settings.phone_numbers[ii][0]);
    Terminal.print(settings.phone_numbers[ii][1]);

    // set this number to number char array
    for(int kk = 2; kk < 11; kk++)
    {
        settings.phone_numbers[ii][kk] = number[kk-1];
        Terminal.print(settings.phone_numbers[ii][kk]);
    }

    Terminal.print("\n");
}

```



```

{

    // fetch all numbers from eeprom and message...
    for( int ii = 0; ii < settings.number_contacts; ii++)
    {

        for(int jj = 0; jj < 11; jj++)
        {

            phoneNumber[jj] = settings.phone_numbers[ii][jj];

        }

        // send message here...

        SimSerial.println("AT");

        Terminal.println("AT");

        delay(5000); // delay 5 seconds to allow init of sim module.

        sprintf(AT_Command,"AT+CMGS=\"%s\"",phoneNumber);

        SimSerial.println(AT_Command);

        Terminal.println(AT_Command);

        delay(200);

        Terminal.write(message);

        SimSerial.write(message);

        Terminal.print(endMessage);

        SimSerial.write(endMessage);

        Terminal.println();

        SimSerial.println();

        Terminal.flush();

        SimSerial.flush();

        delay(5000);

    }

    eepromIndex = 0;

    msgSent = true;

}

incident = true;

// turn off SIM module


// enter sleep mode on arduino
}

```