



# SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

OENG1167: Engineering Capstone Project Part B

Assessment task 2:

Final report

for

An Improved Geolocator for Shorebird Monitoring

Supervisor: Dr Paul Beckett

Submission Deadline: 21st Oct 2019

---

*Student Name:*

---

Bashir Rawaaee: S3484534

Wilbur Lewis : S3529819

## Contents

|   |    |
|---|----|
| Executive summary.....                  | 3  |
| Statement of the problem.....           | 4  |
| Background literature review.....       | 5  |
| Methodology and engineering design..... | 6  |
| Project management and timeline.....    | 9  |
| Findings.....                           | 12 |
| Software.....                           | 12 |
| Hardware.....                           | 28 |
| Budget.....                             | 35 |
| Discussion of results.....              | 36 |
| Conclusion.....                         | 38 |
| References.....                         | 38 |

## Executive summary:

The purpose of this project is to develop an improved light based geolocator for shorebird monitoring. The device is to record the location of a bird using a light sensor to calculate its latitude and longitude coordinates.

As we dig into the research about this project, we found that there are shortcomings with the current technology.

Problems include battery life, accuracy of the location, and having to capture the bird to capture the data.

We came to the realisation that many of these issues can be solved, with the help of a low powered Bluetooth SOC chip.

Using the NXP QN9080 We can:

- Eliminate the need to recapture the bird.
- Prolong the battery life through the possible use of an energy harvesting module.
- Develop a more accurate algorithm to pinpoint the bird's location

The more research we conducted, we felt the drive to introduce improvements and changes to this technology.

To achieve this, we will start from scratch. We will drive this dream by developing its software and hardware. To design a printed circuit board and a developed algorithm to make it all work!



## Statement of the problem:

Shorebirds are birds which gather at intertidal areas or on the edges of freshwater wetlands. They commonly have long legs in relation to their body and some are quite small[1]. These birds often have long migration journeys between seasons for collecting food and mating. Tracking these birds is important as it allows us to monitor the effect of human civilisation on the population of these animals. It is vital for preventing endangerment and extinction of these animals.

Because of the small size of the birds and the long migration journeys to track these birds tracking devices attached to the bird must be very small and have a long lifetime of at least 6 months, ideally longer than a year. Current light reading geolocators use older technology and either have long lifetimes but weigh too much or not enough lifetime but fulfill the weight requirements. They also require the bird to be recaptured to collect the data and there is also no energy harvesting technology such as piezo harvesting or light energy harvesting[2].

The aim of this project is to design a device able to measure and calculate a subject's geolocation using light levels which is cheap, small and has a long lifetime. Using new, lightweight, ultra low powered devices a better product can be made. It also desirable to include bluetooth and possibly energy harvesting to increase lifespan. One design requirement is that we use the NXP QN9080 as our microcontroller for our geocator device which is a single chip SoC integrating Bluetooth Low-Energy, Arm Cortex-M4F, memory and peripherals[3]. Peripherals such as a real time clock and light sensor must be selected. The QN9080 was chosen as it is very small and lightweight, very low powered and includes bluetooth.

This project involves the design of a printed circuit board containing specific sensors and processing chip in a package that is small and light enough to not interfere with a bird's natural behaviour.

The device needs to be developed to regularly read ambient light levels and store the data in memory. The data can later be accessed by the user and an algorithm must be developed which will calculate the location the device was at each day.

It needs to have a long lifetime while not impacting performance which can possibly be achieved through energy harvesting such as a piezoelectric energy harvesting, ie harvesting energy through Solar or vibration.

Bluetooth connectivity is desirable but not entirely necessary. If time is available bluetooth connectivity between the device and a computer/smart phone can be established and light level reading data transferred.

## Background literature review:

To fully understand the problem we are solving, we need to understand the basics and research existing solutions.

Current methods of tracking animal migration is done with either satellite telemetry (such as GPS (Global Positioning System)), Very high frequency (VHF) radio transmitters or light based geolocation. Satellite telemetry uses satellites to get very accurate location estimates and the animal does need to be recaptured to access the data. This method of animal tracking however is often expensive and the devices are usually quite heavy and large which can be limiting for small animal tracking such as shorebirds[4]. Very high frequency (VHF) radio transmitters are an electronic tag which emit a very high radio frequency signal that can be used to locate the animal. The user can track the animal with a directional antenna. For this type of animal tracking it requires a receiver to be within a few kilometers of the animal[5]. This isn't a viable option for long distance shorebird migration as these birds have global migration routes. The most viable current option for shorebird tracking is light based geolocation. This works by periodically recording ambient light levels. These devices only need a light sensor so are lightweight, low powered and inexpensive. These are often used to research bird migration, their routes and identify important staging areas. [6]

These devices already exist on the market. They offer aquatic and terrestrial types and are suitable for species other than birds. They typically range from 0.39g to up to 2g in weight and manufactured by biotrack. [7] A meta analysis on 84 studies of avian species concluded that attaching devices to birds negatively affect their behaviour and ecology to some degree. The most substantial effect was the increase in energy expenditure of these birds and the decreased likelihood of making a nest [8]. Therefore keeping the devices light is of very high importance.

The light based geolocation works by calculating the latitude and longitude. The longitude can be determined by establishing time of either a local noon or a local midnight, If the time of sunrise and sunset are determined by the same light level, and the amount of solar light attenuation (e.g. shading) is equal at these times[9]. The latitude can be determined by examining the earth's spin axis in reference to the sun. It is calculated by determining the night or the day's length. This is the difference between sunrise to sunset or sunset to sunrise[9].

The problem with using sunset and sunrise information is that this method cannot be applied to certain locations and cannot be used all year around. Locations that would yield inaccurate results are around either poles. And latitude data are less accurate near the equator.[10] shading (including cloud, foliage and feathers) and light interference from non direct sunlight or artificial light can also create inaccurate results. Because of this the accuracy can vary depending on its use case/location.

Currently geolocators cost around \$200 to buy and so they are cheap to buy but the problem now in the market is that because they will need to be recaptured to collect the data, not only this is a challenging task but scientists say only about 1 bird is recaptured out of 5 birds sent out, which makes this an expensive research therefore the idea of using a bluetooth powered device helps us capture data without actually capturing the bird.[11]

Current bluetooth technology (version 5.1. QN9080 has a V5 bluetooth chip) allows a maximum range of 40m-400m depending on obstructions and quality of environment( air , rain , humidity). This allows a gap in the market to be filled, where a more costly unit is manufactured however the data can be collected and analysed without recapturing and interrupting the bird, this means the bird can continue its journey without interrupting its behaviour, yielding in more accurate real world results. Another gap in the research is the use of energy harvesting techniques to increase the lifetime of the the geolocator while not substantially increasing the weight of the device.

## Methodology and engineering design:

The outcome of this project is to develop a lightweight light based geolocator for shorebirds. The design process consists of 2 categories. Software and hardware.

The software part of this project focuses on getting the basic functionality working. Transfer of data, working with external clock, developing the algorithm to read sensor data and calculate coordinates. Other functions include storing the data and set up wake and sleep settings for the device as we need to optimise battery life. This is possible through borrowing the QN9080 development kit from our supervisor, as we do not need to develop the hardware before working on the software. This means we can work in parallel to get both developed at the same time.

The hardware part of this project is divided into two sections:

- Prototype design
- Finalised design

The prototype design focuses on helping us select our sensors and peripherals. The prototype printed circuit board we be designed to keep in mind aspects of diagnosis. We also focused on testing energy efficiency and methods to increase battery life.

A finalised PCB design is manufactured to iron out bugs and errors. It makes a compact design and fit all features and requirements.

Below is a breakdown of the list of tasks we undertook as we reach the completion of this project.

## **Task 1: Literature review/competing technology and project requirements**

The first step is reviewing literature on light based geolocation and looking at competing technology. Through this we will gain a fundamental understanding of this technology and its importance. Possible improvements can also be identified and clear outcomes and requirements of the project can be established. The two main requirements is the lifetime of the device which will be limited by its memory capacity and battery life and its low weight. Throughout the project these requirements must be considered for every decision.

## **Task 2: Project planning**

With clear outcomes identified the project is broken down into sub-tasks and were divided between the team members. A timeline is created with a deadline for the completion of each task to ensure the project is completed on time. Progress on the project must be steady throughout the year with weekly meetings with the supervisor with action points created to be completed by the following week.

## **Task 3: QN9080 geolocation algorithm from light-level readings**

### **3.1. Get familiar with Integrated Development Environment and QN9080 operation**

The QN9080 is programmed using the MCUXpresso Integrated Development Environment with C++. The student must familiarise themselves with this IDE and the QN9080 development board to learn how to program it and how it works. The power consumption of the device in different power modes and memory size must be identified.

### **3.2. Real time clock selection and establish communication with QN9080**

A clock must be selected which is accurate and low powered and able to communicate with the QN9080. With a clock selected communication between it and the QN9080 must be established. This will be done with the QN9080 development board and the MCUXpresso IDE.

### **3.3. QN9080 data storage**

Data readings must be stored in the onboard EEPROM. It must store light level readings with the time it was taken. The size of each location entry in the EEPROM inside the QN9080 must be figured out and calculate how long the geolocator tag can record data until it is full in memory. The memory of the QN9080 must be large enough to store cumulative geolocation data for at least six months.

### **3.3. QN9080 sleep settings**

For the device to have a long lifetime the device will be put in low power mode most of the time unless it needs to take a light level reading or communicate via bluetooth. How the device will sleep and wake needs to be decided and how it will utilize the real time clock.

### **3.3. Finalise QN9080 functionality**

A final program must be written for the QN9080 to ensure all requirements are met. How often it wakes and stores data will be based on memory size and power restrictions. There must be enough readings to allow us to calculate the time at which local noon occurs which is used to estimate the longitude and calculate the length of the day which will help us estimate latitude.

## **Task 4: Hardware development**

### **4.1 Choose peripherals**

The peripherals will include a suitable light sensor and a temperature sensor if possible (depends on weight and lifetime requirements). Different types of sensors are available to use. These include analogue sensors that require an analog to digital conversion. Others include digital phototransistor[14], cold junction thermocouples with SPI interface [13]options and even solar temperature sensors[15].

### **4.1 Prototype PCB**

To test and prove a working device and an algorithm, our supervisor has advised us to create a testing ground for this. This is to ensure that we have a platform we can work from rather than having to one chance to get everything working at once. We have decided to design and manufacture a test PCB which contains all the components required for this project. The design of this PCB incurs the following features.

- Test pins  
The test board is to be able to be worked on and easily diagnose problems. This would mean that between different points in the circuit we will place header pins for multimeter to read values from. And manually wire out components to the SoC.. We feel that this is required to ensure that we can work, fix and diagnose problems on the go without having to redesign a PCB because of a silly mistake.
- External clock
- Light sensor
- EEPROM
- Temperature sensor  
The board needs to have an accurate external clock to be able to work. The light based geolocator algorithm requires time to work out differences between sunrise and sunset to work out the location coordinates.
- For using the functionality of the bluetooth for this project we require an antenna to achieve higher distance connection. This will be achieved through implementing an RF circuit and antenna.
- Ground points and planes on the circuit board to decrease electrical noise and interferences
- USB connection  
The USB interface would be our method of pushing the program with the algorithm into the QN9080 chip.



### 4.3 Final Product

As we close into the second semester, wrapping up the project requires a finalised PCB and put all we have done and learnt along the way onto a small product. Along with the PCB we need package it up for use. This means we need to make sure the product can be mounted onto a bird and is waterproof for harsh conditions. We definitely do not want to look over this.

#### Task 5: Bluetooth (desired)

Once the final design has mostly been established this task can be started. Having a working bluetooth connection isn't as high priority as getting the device working. Bluetooth connectivity is very complicated. This involves magic of RF antennas which requires research and applying them to our PCB designs. By developing our device with the QN9080 if bluetooth data transfer is not established it is an option for further development.

#### Project management and timeline:

To ensure the required tasks were completed on time a project timeline was created. We set realistic goals towards achieving this with room for error for unexpected tasks or tasks that would take longer. As mentioned we divided these tasks into two categories (hardware and software). This allows for the project to be easily split between the two students with one student mostly working on hardware and the other on software however not exclusively.

Each team member can work independently towards achieving these tasks without having to wait for each others completion of individual tasks. We held weekly meetings to discuss progress and make decisions so both students were involved with and understood all aspects of the project. Biweekly meetings were also had with the project supervisor to discuss problems and bottlenecks.

Although we have achieved minimum requirements for this project, One major bottleneck we faced was that we did not have access to the SMD lab until the 25th of September (week 9). The project is heavily dependant on the availability of the lab as we are soldering SMD components. No other alternatives is suitable as the project required the use of the QN9080 SoC which is a SMD with 5mm x 5mm in profile and 48 pins. This meant we had to rush to design and manufacture PCBs quickly. This did not give us enough time to achieve the following:

1. Implement waterproofing solutions
2. Testing of final product
  - Algorithm accuracy
  - Bluetooth range
3. Implement the possibility of using a piezo harvester( optional )
4. Replenish other parts required.

Our overall project timeline is outlined below in the task table.

| No: | Task                                     | Week |     |     |     |      |       |       |       |       |       |       |       |
|-----|--|------|-----|-----|-----|------|-------|-------|-------|-------|-------|-------|-------|
|     |  | 1-2  | 2-4 | 5-6 | 6-8 | 9-10 | 10-12 | 12-14 | 14-16 | 16-18 | 18-20 | 20-22 | 22-24 |
| 1   | Literature review                        |      |     |     |     |      |       |       |       |       |       |       |       |
| 2   | Project Proposal                         |      |     |     |     |      |       |       |       |       |       |       |       |
| 3   | QN9080 familiarisation                   |      |     |     |     |      |       |       |       |       |       |       |       |
| 4   | Geolocator algorithm research            |      |     |     |     |      |       |       |       |       |       |       |       |
| 5   | Choose peripherals                       |      |     |     |     |      |       |       |       |       |       |       |       |
| 6   | Clock communication                      |      |     |     |     |      |       |       |       |       |       |       |       |
| 7   | Semester 1 Presentation and Report       |      |     |     |     |      |       |       |       |       |       |       |       |
| 8   | Prototype 1 PCB Design                   |      |     |     |     |      |       |       |       |       |       |       |       |
| 9   | QN9080 Sleep settings                    |      |     |     |     |      |       |       |       |       |       |       |       |
| 10  | Prototype 2 PCB Design                   |      |     |     |     |      |       |       |       |       |       |       |       |
| 11  | Print/solder 2 Prototype PCB             |      |     |     |     |      |       |       |       |       |       |       |       |
| 12  | Completion Plan                          |      |     |     |     |      |       |       |       |       |       |       |       |
| 13  | Store Data In Memory                     |      |     |     |     |      |       |       |       |       |       |       |       |
| 14  | Light sensor reading                     |      |     |     |     |      |       |       |       |       |       |       |       |
| 15  | Finalise QN9080 functionality            |      |     |     |     |      |       |       |       |       |       |       |       |
| 16  | Final PCB Design, Printing and Soldering |      |     |     |     |      |       |       |       |       |       |       |       |
| 17  | Semester 2 Presentation and Report       |      |     |     |     |      |       |       |       |       |       |       |       |

#### Detailed description:

- Task 1 and 2 consisted of getting to know the project and understand what we are doing, why we are doing it and how it is done and what is expected from us.
- Task 3 , we looked at the NXP SOC chip. Its features and specs. In particular we looked at its operating voltage and current. We researched how to program the QN9080. To find the programming language, the IDE and manuals to help through it.
- Task 4 focuses on researching the workings and logic behind how a light based geolocator works, what we need, and how we will achieve this.
- Task 5 involved listing and selecting peripherals that were needed for our first prototype. We needed an real time clock, memory, light sensor and temperature sensor. Given that the peripherals selected were very small and surface mount. A prototype had to be developed (task 8) with breakout pins to enable us to start working on the communication with these devices.

- Task 6 involved us establishing SPI communication with the real time clock RV-2123-C2. This device had a development board which was purchased. This enabled us to start working with this device before developing the prototype. This was the first peripheral selected and was very important for our device. A low powered, accurate real time clock is very necessary for a long life and accurate light based geolocator.
- Task 7 presented the project to the audience, we have received very useful feedback from the assessors and the audience on what to implement to improve the design.
- Task 8 aimed to design a prototype with a light sensor, clock and memory with test pins. This was to enable us to start working on the communication with these peripherals.
- Task 9 involved working on the sleep and wake settings of the QN9080 to ensure low power consumption
- Task 10,11, these tasks focuses on developing a second prototype PCB as we have designed incorrect footprint for QN9080 chip.
- Task 12 focuses on solidifying our plan as we commit to finish the project on time and come up with action plan if anybody got sick and could not finish the project.
- Task 13 involved establishing communication with the EEPROM allowing data to be stored. We would have liked to have worked on this earlier however due to SMD lab being available so late and therefore the PCB being soldered late we only were able to work on it at this time. Preparation was made however as we waited for the PCB to be made. This was done by reading the datasheet and developing code.
- Task 14 involved taking light sensor readings using the ADC of the QN9080.
- Task 15 aims at finalising the functionality of the QN9080. This includes the reading of light sensor, storage of data into the eeprom and wake and sleep settings all into one program.
- Task 16 focused on finalising the hardware aspect of this project, as we near the end of the semester we have come up with a tiny version of our prototype and this finalised product irons out all the problems and issues we have had in the past and to test it out.
- Task 17 focused on the academic requirements side of this project, however this also solidifies anything we learnt during the year, as we recap them in the report and during the presentation. We develop a brochure and poster, this will help us reach into the marketing side of this project.

## Findings

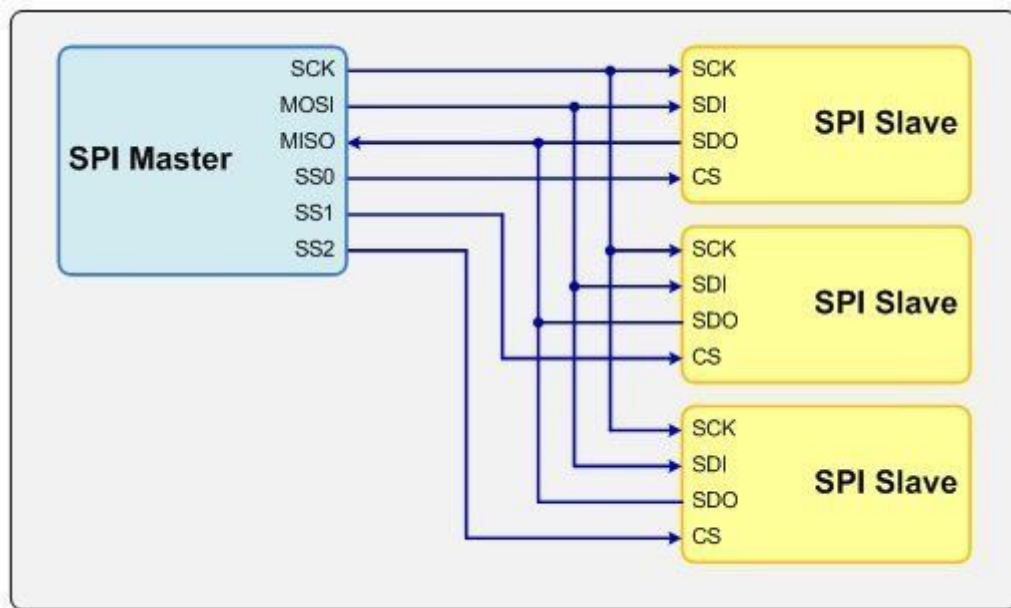
### Software:

The QN9080 development board has been acquired and examples provided by the manufactures of the QN9080 (NXP) have been run on the board using the MCUXpresso Integrated Development Environment (IDE). This was first to gain familiarity with the IDE as well as with the board.

### SPI:

We have chosen a clock (RV-2123-C2) as well as external memory (EEPROM M95M02-DR) and have established communication with them using the QN9080 development board.

Serial peripheral interface (SPI) is a communication protocol often used by microcontrollers to control peripherals such as sensors or analogue to digital converters. It is set up in a way where there is one master (the microcontroller) and the peripherals set up as slaves. The master will control the slave and can write to it to change it settings or write to it to ask for information. The slave will not do anything unless the master tells it to do something.



*Figure 1 - Master and three slaves connect via SPI [16]*

There is often three or four busses/wires that are used for this communication method.

| Master Pin                 | Purpose   | Slave Pin                            | Purpose   |
|----------------------------|---|--------------------------------------|---|
| Master Out Slave In (MOSI) | The master outputs data to the slave  | Slave Data In (SDI)                  | The slave receives data from the master   |
| Master In Slave Out (MISO) | The master receives data from the slave   | Slave Data Out (SDO)                 | The slave transmits data to the master  |
| Slave Select (SS)          | This pin selects the slave that will be used. Often multiple slaves are connected to the slave MOSI, MISO and SCK pins. This pin selects the slave that will interpret the data sent. | Chip Select (CS) or Chip Enable (CE) | The master will use this to select this chip to transfer and receive data from. |
| Serial Clock (SCK or SCL)  | SPI is synchronous, a clock is used to synchronise the transfer of data. The clock is generated by the master.  | Serial Clock (SCK)                   | Receive clock from master   |

*Table 1 - Master Pins and slave pins and their purpose*

To better explain SPI I will explain how we established SPI communication with the clock.

### **Clock:**

The clock we chose is the RV-2123-C2. It was chosen as it is very small (5.0 x 3.2 x 1.2 mm) and lightweight and also requires very little power (power consumption of 130nA at 3 V).

A typical communication stream with this clock is shown below. This device is slave select active high with data been output on the falling edge and captured on the rising edge.

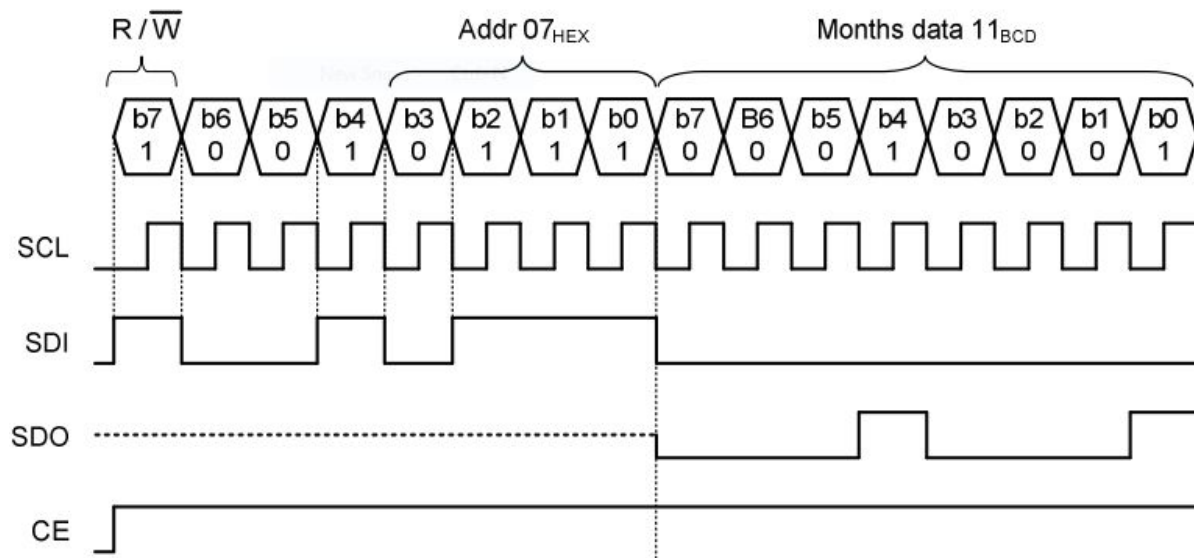


Figure 2 - Master reading month data from RV-2123-C2 taken from datasheet[17]

The chip enable (CE) pin must be set to high for this device to enable data transfer. With the CE pin enabled data will be transferred one bit at a time synchronising with the clock (with the most significant bit is sent first). The data will be read by the slave or master when the clock pin (SCL) is high and data will be changed on the falling edge. In this example the masters asks the slave for the month data. This is sent to the slave pin of Slave Data In (SDI) and the slave then replies with the pin of Slave Data Out (SDO) back to the master.

The first bit sent is the read or write bit. If the bit is 1 this means the master wants to read data from the slave. If the bit is 0 it wants to write. The next three bits (001) are the subaddress of the device. If not set correctly the device will ignore the data. The next four bits (Addr) is the address of the register. The device has multiple registers which control the configuration of the device and also specify where data is. In this example it wants to read the months data and so sends the address of the months register (0111 or 0x7). The slave then replies with the data of the month. This is in binary coded decimal format with the first four bits (in this example 0001) being the first digit of the month and the last four digits being the second digit of the month (0001). 0001 is 1 in decimal and so the month is 11 or November. The master has successfully accessed the month data from the slave.

The configuration, reading and writing to the clock (RV-2123-C2) by the QN9080 development board has been successful. Below is code written that resets the clock and then sets the 'seconds' to 2 seconds and then runs a loop that read the seconds from the clock and outputs it to the console approximately every one second. Some initialisations occurred before the main function are not shown. The code below however shows how data from the clock was accessed by the QN9080. Comments are used to explain what each statement does.

```

int main(void)
{
    spi_master_config_t userConfig = {0};
    uint32_t srcFreq = 0;
    uint32_t readSec = 0x92U;
    //uint32_t i = 0;
    //uint32_t err = 0;
    spi_transfer_t xfer = {0};

    BOARD_InitPins(); //initialise pins on board
    BOARD_BootClockHSRUN(); //set the clock speed that will be used.
    BOARD_InitDebugConsole(); //start debug console

    SPI_MasterGetDefaultConfig(&userConfig); //this will get default configurations for the SPI master such as baud rate and clock phase
    srcFreq = EXAMPLE_SPI_MASTER_CLK_FREQ; //gets frequency to run SPI functions
    userConfig.sselNum = (spi_ssel_t)EXAMPLE_SPI_SSEL; //this sets the SS pin to active high.
    userConfig.sselPol = (spi_spol_t)EXAMPLE_SPI_SPOL; //set the clock polarity
    SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality

    /* Init Buffer*/

    srcBuff[0] = 0x10U; //control register 1 selected to be written to
    srcBuff[1] = 0x58U; //data written to control register 1 (resets device)
    srcBuff[2] = 0x12U; // 'seconds' register to be written to.
    srcBuff[3] = 0x02U; //sets it to 2 seconds
    srcBuff[4] = 0x00U; //empty data

    /*Start Transfer*/
    // data is both transferred and read by the master at the same time.
    // for this case the received data is not relevant as we have only written to the slave. not read
    xfer.txData = srcBuff; //data to be transferred
    xfer.rxData = destBuff; //data to be received
    xfer.dataSize = sizeof(destBuff);
    SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //data sent

    for (int d = 1; d <= 20; d++){ //a loop is created as a delay
        for (int c = 1; c <= 32767; c++){
        }
    }
}

```

Figure 3 - Initialisation code resetting the clock and writing '2' to the 'seconds' register of the clock

```

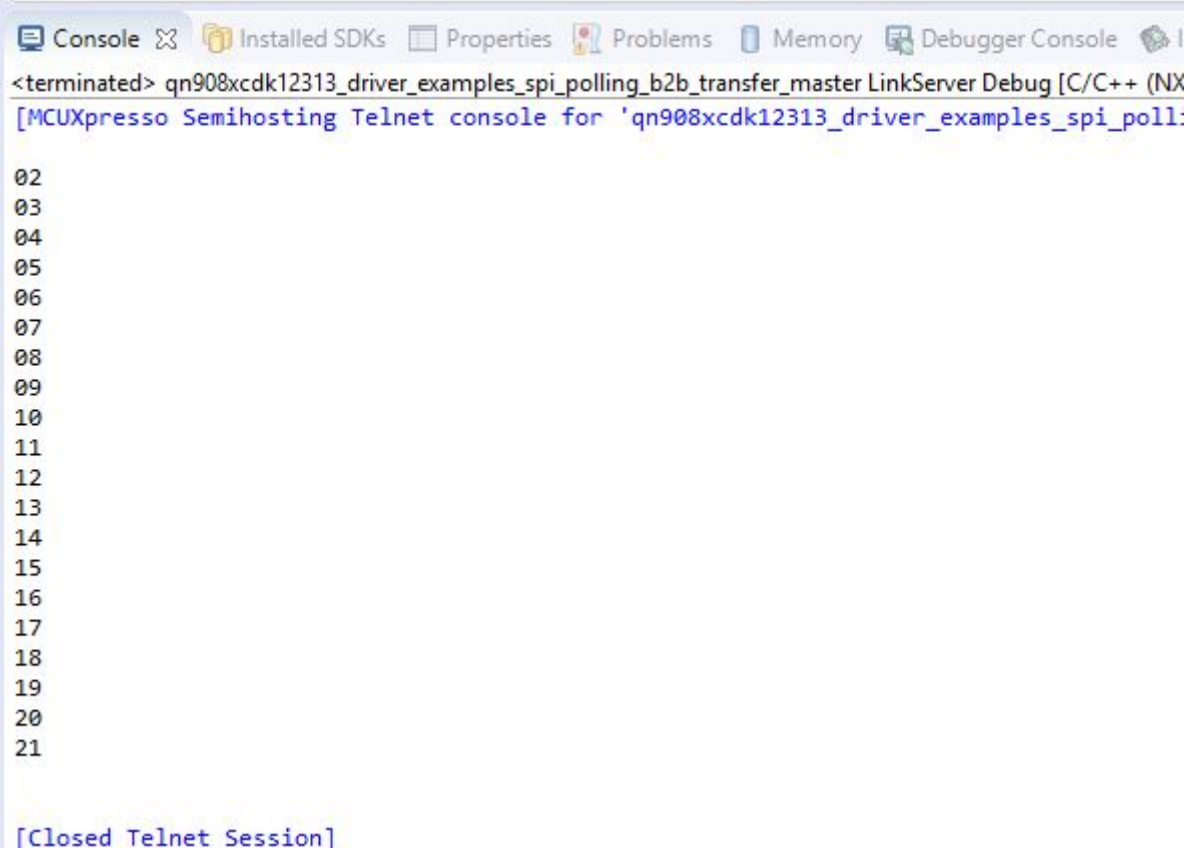
//next the seconds data is read from the slave that is output to the console on a loop
while (1)
{
    SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq);

    srcBuff2[0] = 0x92U; // selects the 'seconds' register and asks to read.
    srcBuff2[1] = 0x00U; // selects the 'seconds' register and asks to read.
    xfer.txData = srcBuff2;
    xfer.rxData = destBuff2; //data is read and received at the same time
    xfer.dataSize = sizeof(destBuff2);
    SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //'seconds' register selected

    /*Check if the data is right*/
    //output data to console
    test = destBuff2[1];
    digit1 = test & 0xF0; // select first 4 bits
    digit1 = digit1 >> 4; //shift bits by 4.this is the first digit
    digit2 = test & 0x0F; // this is the second digit
    PRINTF("%d%d \r\n", digit1, digit2); // digit 1 and two output to the console
    for (int d = 1; d <= 100; d++){ //delay of approximately one second
        for (int c = 1; c <= 32767; c++){
        }
    }
}
}

```

Figure 4 - Loop of code that reads the 'seconds' of the clock every 1 second and outputs to console



The screenshot shows a debugger's console window with the following content:

```
<terminated> qn908xcdk12313_driver_examples_spi_polling_b2b_transfer_master LinkServer Debug [C/C++ (NX
[MCUXpresso Semihosting Telnet console for 'qn908xcdk12313_driver_examples_spi_poll:

02
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20
21

[Closed Telnet Session]
```

*Figure 5 - Console with seconds data outputted*

The data has been successfully written and then read. As shown in figure 5 the clock begins at 2 seconds which is what it was set to and continues to count up.

The following is a block diagram of each of the inputs and outputs of the RV-2123-C2.



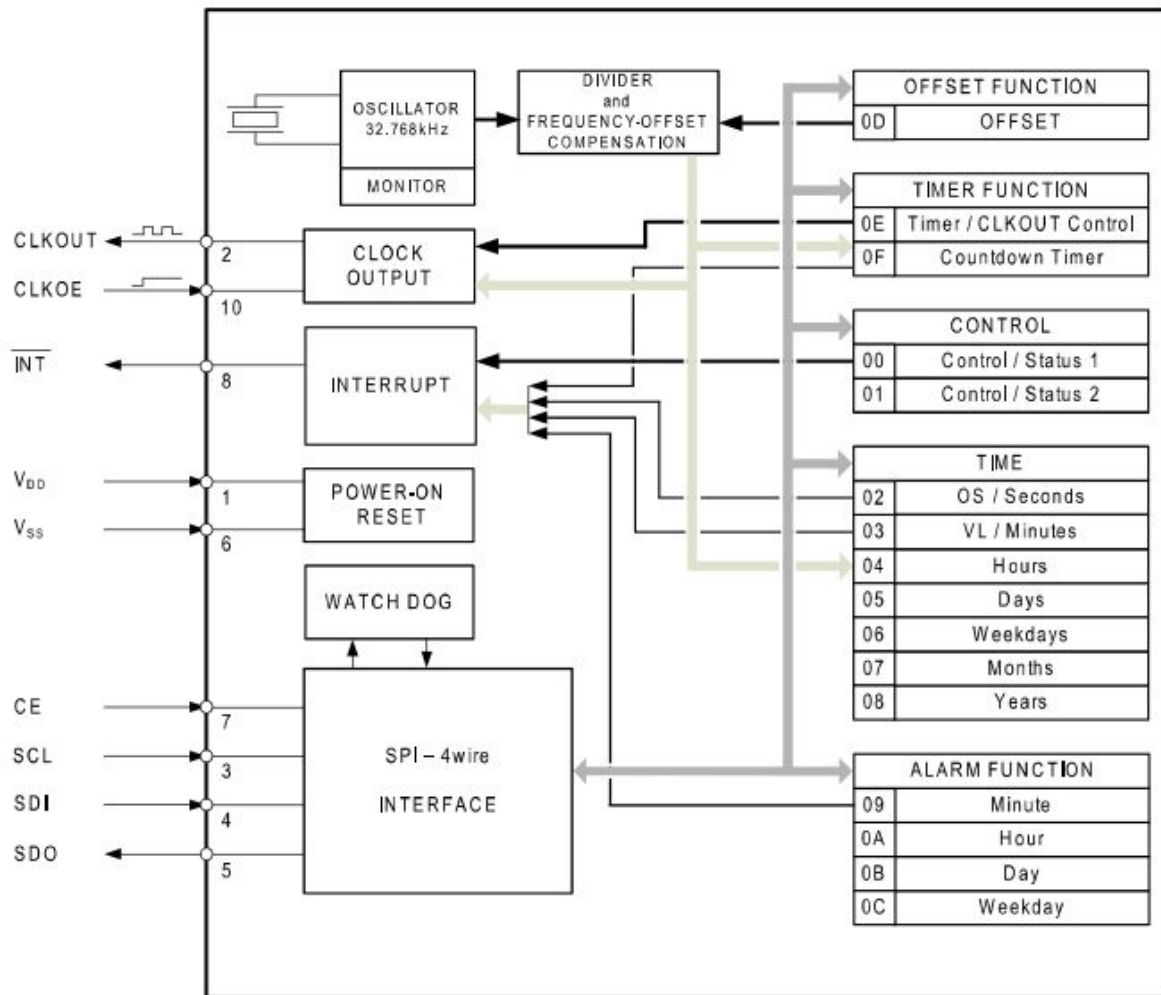


Figure 6 - RV-2123-C2 Clock block diagram

We have successfully been able to set the time registers as well as read them. This is what was required by this clock to ensure functionality of our device.

## EEPROM:

The eeprom we chose is the M95M02-DR. It was chosen as it also uses SPI communication, has a capacity of 256Kbytes and is small and lightweight.

Taken from the datasheet a typical read sequence is shown below.

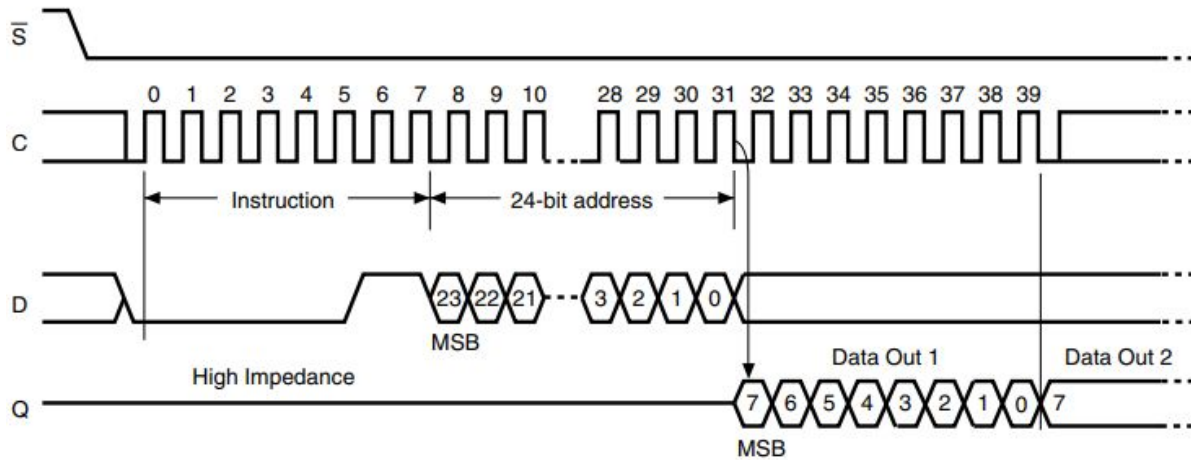


Figure 7 - M95M02-DR read sequence

This SPI communication has the slave select as active low. The address of the memory are 24 bits and data is stored at this memory location as a byte. The instruction for read as shown above is 0x03.

```
int main(void)
{
    spi_master_config_t userConfig = {0};
    uint32_t srcFreq = 0;
    spi_transfer_t xfer = {0};

    BOARD_InitPins(); //initialise pins on board
    BOARD_BootClockHSRUN(); //set the clock speed that will be used. high speed is better to ensure reliable SPI transfer.
    BOARD_InitDebugConsole(); //start debug console

    SPI_MasterGetDefaultConfig(&userConfig); //this will get default configurations for the SPI master such as baud rate and clock phase
    srcFreq = EXAMPLE_SPI_MASTER_CLK_FREQ; //gets frequency to run SPI functions
    userConfig.sselNum = (spi_ssel_t)EXAMPLE_SPI_SSEL; //this sets the SS pin to active low.
    userConfig.sselPol = (spi_spol_t)EXAMPLE_SPI_SPOL; //set the clock polarity
    userConfig.baudRate_Bps = (uint32_t)8000000;
    SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality

    /* Init Buffer*/

    srcBuff2[0] = 0x06U; //Write Enable register written to.

    /*Start Transfer*/
    // data is both transfered and read by the master at the same time.
    // for this case the received data isn't relevant as we have only written to the slave. not read
    xfer.txData = srcBuff2; //data to be transfered
    xfer.rxData = destBuff2; //data to be received
    xfer.dataSize = sizeof(destBuff2);
    SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //data sent
    SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality
}
```

Figure 8 - EEPROM initialise and write enable

```

SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality

srcBuff[0] = 0x02U; //write command
srcBuff[1] = 0x34U; //24 bit address
srcBuff[2] = 0x11U; //24 bit address
srcBuff[3] = 0x11U; //24 bit address
srcBuff[4] = 0x06U; //value to store

xfer.txData = srcBuff; //data to be transferred
xfer.rxData = destBuff; //data to be received
xfer.dataSize = sizeof(destBuff);
SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //data sent
SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality

srcBuff3[0] = 0x03U; //read command
srcBuff3[1] = 0x34U; //24 bit address
srcBuff3[2] = 0x11U; //24 bit address
srcBuff3[3] = 0x11U; //24 bit address
srcBuff3[4] = 0x00U; // receive reply from eeprom

xfer.txData = srcBuff3; //data to be transferred
xfer.rxData = destBuff3; //data to be received
xfer.dataSize = sizeof(destBuff3);
SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //data sent

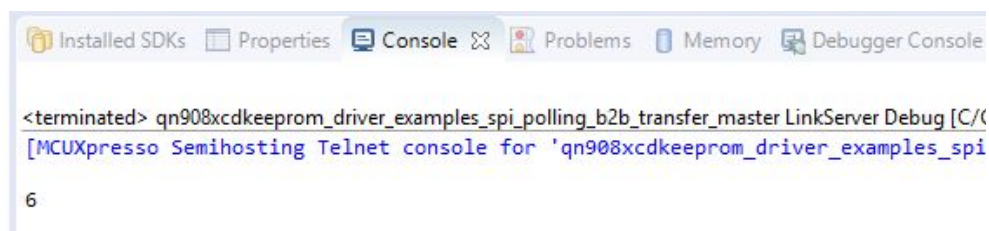
PRINTF("%d \r\n", destBuff3[4]); // output data to console

while (1)
{
}
}

```

Figure 9 - EEPROM write and read

As shown above. First write enable is written to the board which must be written too before writing to the board. Next the write command is sent with a 24 bit address and a value to store. In this case the value is 6. Then the read command is then sent with the 24 bit address and then the data is read and output to the console. The data read is found to be 6 as shown below.



```

<terminated> qn908xcdkeeprom_driver_examples_spi_polling_b2b_transfer_master LinkServer Debug [C/
[MCUXpresso Semihosting Telnet console for 'qn908xcdkeeprom_driver_examples_spi
6

```

Figure 10 - EEPROM code output (read and write successful)

## Light Sensor

The light sensor chosen is the APDS-9007. This is an analog light sensor which will give a current output based on lux. This was chosen as it is very small (0.8mm by 2.4mm by 2.0mm) and very simple to use and with a range of 3 lux to 70K lux. This is adequate for our design as anything less than 3lux will be night time.

It is recommended that the output current be put over a 27kohm resistor for the most accurate conversion. The output current corresponds to the lux as outlined in figure 11.

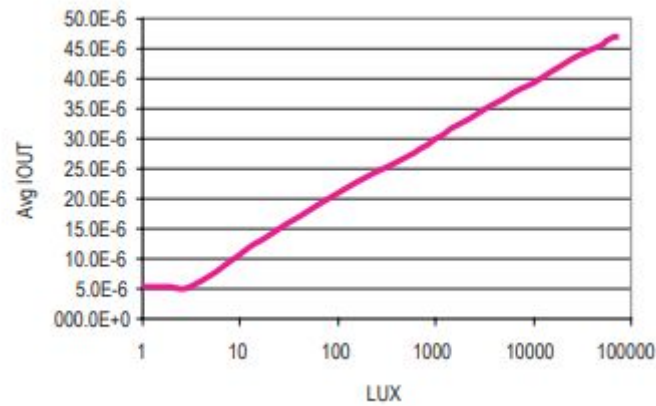


Figure 11 - Current vs Lux for apds-9007 [20]

This was read using the ADCs of the QN9080. A burst method was used to take a number of readings and then get the average. This ensures an accurate light reading. Code was written to configure the ADC and then wait for user input. When the user inputs to the console the ADC will take a reading.

```
int main(void)
{
    uint32_t i;
    uint32_t data[ADC_BURST_EXAMPLE_NUMBER];

    /* Power on ADC */
    POWER_EnableADC(true);

    /* Initialize board hardware. */
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();

    /* Configure the converter and work mode. */
    ADC_Configuration();
    PRINTF("Configuration Done.\r\n");
    PRINTF("Press any key to get demo channel's ADC value ...\r\n"); //wait for user input
```

Figure 12 - Initialise ADC code

```

float fresult;
while (1)
{
    GETCHAR();

    /* Start burst */
    ADC_Enable(DEMO_ADC_BASE, true);
    /* Do software trigger */
    ADC_DoSoftwareTrigger(DEMO_ADC_BASE);
    for (i = 0; i < ADC_BURST_EXAMPLE_NUMBER; i++)
    {
        /* Wait for convert complete */
        while (!(ADC_GetStatusFlags(DEMO_ADC_BASE) & kADC_DataReadyFlag))
        {
        }
        /* Get the result */
        data[i] = ADC_GetConversionResult(DEMO_ADC_BASE);
    }
    /* Stop burst */
    ADC_Enable(DEMO_ADC_BASE, false);

    PRINTF("Sample %d numbers: \r\n", ADC_BURST_EXAMPLE_NUMBER);
    for (i = 0; i < ADC_BURST_EXAMPLE_NUMBER; i++)
    {
        fresult = ADC_ConversionResult2Mv(DEMO_ADC_BASE, DEMO_ADC_CHANNEL, DEMO_ADC_CFG_IDX, g_AdcBandgap,
                                           g_AdcVinn, data[i]); // convert from number to mV.
        PRINTF("Original: 0x%x\t Ch: %d\t Result: %f(mv)\r\n", data[i], DEMO_ADC_CHANNEL, fresult);
    }
}
}

```

Figure 13 - Read ADC pin as burst of 10

```

qn908xcdk_driver_examples_adc_burst LinkServer Debug [C/C++ (NXP Semiconduc
[MCUXpresso Semihosting Telnet console for 'qn908xcdk_driver_e

Configuration Done.
Press any key to get demo channel's ADC value ...
j
Sample 10 numbers:
Original: 0x47e86889      Ch: 8      Result: 220.770935(mv)
Original: 0x47e89135      Ch: 8      Result: 225.426285(mv)
Original: 0x47e8826f      Ch: 8      Result: 223.735275(mv)
Original: 0x47e86de1      Ch: 8      Result: 221.382584(mv)
Original: 0x47e86d4f      Ch: 8      Result: 221.317337(mv)
Original: 0x47e878b7      Ch: 8      Result: 222.622864(mv)
Original: 0x47e88a63      Ch: 8      Result: 224.645584(mv)
Original: 0x47e8875d      Ch: 8      Result: 224.299561(mv)
Original: 0x47e87aad      Ch: 8      Result: 222.847351(mv)
Original: 0x47e87d59      Ch: 8      Result: 223.153137(mv)

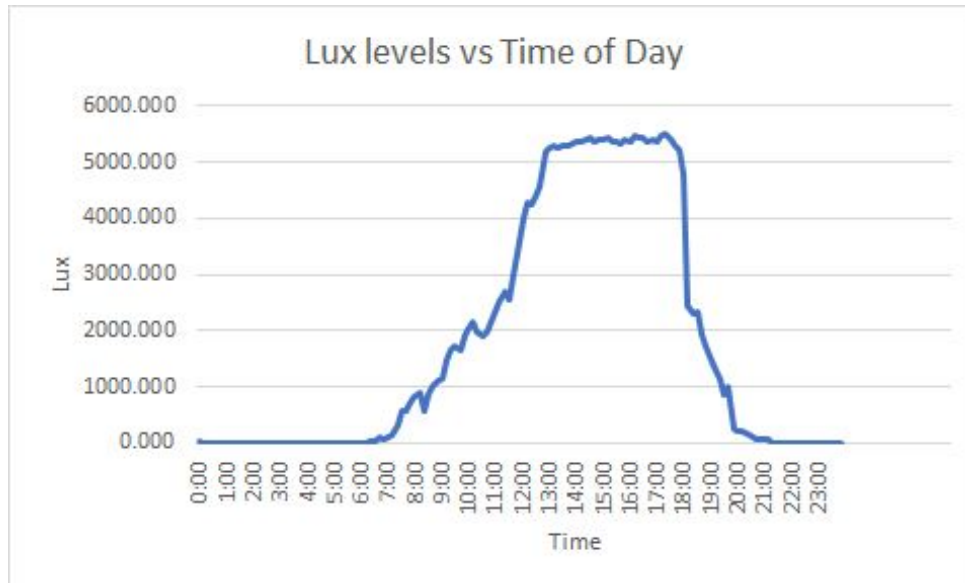
```

Figure 14 - ADC reading output to console

As shown a value of approximately 224mV was taken. This is equivalent to 8.3uA (224mv/27kohms). This is equivalent to 6.7 Lux according to the datasheet. This reading was taken at night with a desk lamp so the lux reading is what is expected.



With this working a simple program was written to take light readings every 10 minutes for a day (14/10/2019). With this we developed the following graph.



*Figure 15 - Lux readings of a full day*

As shown the light level readings increase at approximately 6:30am (sunrise) and decreased at approximately 8pm (sunset). This will be adequate for calculating the location of the bird as longitude is determined by establishing time of local noon and latitude is determined by the day length (sunrise to sunset) as well as by examining the earth's spin axis in reference to the sun. We only need to find the time of local sunrise and sunset and this was done successfully.

### **Finalise QN9080 Functionality**

For our final prototype we will have the device wake every 10 minutes to take and store light level readings. To explain how this program works I will first explain how we organise storing data in EEPROM so data can be easily accessed and data isn't overwritten. To do this the first three bytes of the EEPROM are used to store the memory address of where to store data when the device wakes up (memory address is 24 bits long). Then before the device goes back to sleep the three bytes of data are updated so that when the device wakes against it will store data in the next successive memory address.

The data that will be stored in memory is first the time/date (2 bytes) and then the light sensor reading (1 byte). Given that the device is reading every 10 minutes we have managed to minimize the amount of bytes needed to store the time/date. This is done by giving each 10 minutes of the year a value of 1. I.e. January 1st at 00:10 is coded as 1 (the value of 1 is stored in the two bytes used to store time/date readings) and January 1st at 00:20 will be stored as 2. With this format we will have 52560 ( $365 \times 24 \times 6$ ) readings a year.

December 31st at 23:50 will have a code of 52559. This number is less than the max number able to be stored in two bytes ( $2^{16} = 65536$ ). This allows us to store data using only two bytes which saves us a lot of space. A single byte can be used to store the start year of the device if the device runs for multiple years.

The light readings will be stored in 8 bytes. The light readings will be between 5-45uA as per figure 11. This will give us an accuracy of  $\pm 0.15625\text{uA}$  ( $40/2^8$ ). Given that 5-45uA is exponential the readings with low lux (i.e 10) will be more accurate than readings of larger lux. We need low lux readings to be more accurate as this is how we tell when sunrise and sunset occurs so this is adequate. At approximately 10 lux we will have an accuracy of  $\pm 0.15625$  lux.

With storing 3 bytes of data for each reading and with an EEPROM of 256 Kbytes the lifetime of the device can be calculated. The number of days until the memory fills up can be approximated by dividing the total storage amount (in bytes) by the number of bytes used per day. This gives us a lifetime of 592.59 days ( $256,000/(24*6*3)$ ). With the inclusion of a temperature sensor reading this would decrease to 444.44 days. A larger memory could be selected if a larger lifetime is required or if readings are requested to be taken more often. This would not increase the size or weight of the device by a substantial amount as the current EEPROM is of 3x3x0.5mm.

With the functionality defined a simplified version of this was created and run on the QN9080DK connecting to the peripherals (light sensor, clock and EEPROM). The program is set up to wake up, read the clock, read the EEPROM to get the memory address to store data in, take a light reading and then store the clock and light reading into memory. We do this every 20 seconds using power mode 1 and waking up using a GPIO pin. Before this program is run the clock is set up to send an interrupt to this pin every 20 seconds. This will allow us to wake up and take a reading every 20 seconds. This is done using a separate program which sets the COUNTDOWN TIMER register of the clock. This interrupt output is active low and the GPIO wake-up we use (PA19) is also set-up as active low so can just be directly connected. Another separate program is also run to set the memory address to 000003 to begin with. For this program the memory address is 1 byte long instead of 3. This program is only run a few times to verify the functionality so we only need to store the last byte of the memory address (i.e 03 is the starting byte). We also only read the seconds register and store this data as our time data. The light reading is also converted to an integer of uA. This means there is only an accuracy of 1uA. We would like to have completed a more complex algorithm as mentioned above however due to getting a working PCB so late and therefore access to the EEPROM and light sensor late this is all we were able to complete. This however verifies the functionality of how the device should work.

The following is code snippets with comments explaining the workings of the code.

```

BOARD_InitPins(); // set up the function of pins that will be used.
BOARD_BootClockRUN(); // this different than used before.
BOARD_InitDebugConsole(); //start debug console

//SPI read clock
spi_master_config_t userConfig = {0};
uint32_t srcFreq = 0;
spi_transfer_t xfer = {0};

SPI_MasterGetDefaultConfig(&userConfig); //this will get default configurations for the SPI master such as baud rate and clock phase
srcFreq = EXAMPLE_SPI_MASTER_CLK_FREQ; //gets frequency to run SPI functions
userConfig.sselNum = (spi_ssel_t)EXAMPLE_SPI_SSEL; //selects which slave to talk to
userConfig.sselPol = (spi_spol_t)EXAMPLE_SPI_SPOL; //set the slave select polarity
userConfig.baudRate_Bps = (uint32_t)8000000;
SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality

srcBuff2[0] = 0x92U; // selects the 'seconds' register and asks to read.
srcBuff2[1] = 0x00U; // selects the 'seconds' register and asks to read.
xfer.txData = srcBuff2;
xfer.rxData = destBuff2; //data is read and received at the same time
xfer.dataSize = sizeof(destBuff2);
SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //seconds' register selected

/*Check if the data is right*/
//output data to console
seconds = destBuff2[1];
digit1 = seconds & 0xF0; // select first 4 bits
digit1 = digit1 >> 4; //shift bits by 4.this is the first digit
digit2 = seconds & 0x0F; // this is the second digit
PRINTF("Seconds: %d%d \r\n", digit1, digit2); // digit 1 and two output to the console

```

Figure 16 - Initialise board, read clock and output seconds to console

```

//EEPROM memory address read
userConfig.sselNum = (spi_ssel_t)0; //changes which slave to talk to
SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality

read memory address. this has been simplified to 1 byte for this prototype
srcBuff3[0] = 0x03U; //read command
srcBuff3[1] = 0x00U; //24 bit address
srcBuff3[2] = 0x00U; //24 bit address
srcBuff3[3] = 0x01U; //24 bit address
srcBuff3[4] = 0x00U; // receive reply from eeprom

xfer.txData = srcBuff3; //data to be transferred
xfer.rxData = destBuff3; //data to be received
xfer.dataSize = sizeof(destBuff3);
SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //data sent

PRINTF("Memory Location: %d \r\n",srcBuff3[4]); // output memory location to the console
Mem_loc = srcBuff3[4];

```

Figure 17 - memory address read



```

//light reading
// configure ADC
uint32_t i;
uint32_t data[ADC_BURST_EXAMPLE_NUMBER];
ADC_Configuration();

// get ADC readings
/* Power on ADC */
POWER_EnableADC(true);
/* Start burst */
ADC_Enable(DEMO_ADC_BASE, true);
/* Do software trigger */
ADC_DoSoftwareTrigger(DEMO_ADC_BASE);
for (i = 0; i < ADC_BURST_EXAMPLE_NUMBER; i++)
{
    /* Wait for convert complete */
    while (!(ADC_GetStatusFlags(DEMO_ADC_BASE) & kADC_DataReadyFlag))
    {
    }
    /* Get the result */
    data[i] = ADC_GetConversionResult(DEMO_ADC_BASE);
}
/* Stop burst */
ADC_Enable(DEMO_ADC_BASE, false);

//PRINTF("Sample %d numbers: \r\n", ADC_BURST_EXAMPLE_NUMBER);
for (i = 0; i < ADC_BURST_EXAMPLE_NUMBER; i++)
{
    fresult = ADC_ConversionResult2Mv(DEMO_ADC_BASE, DEMO_ADC_CHANNEL, DEMO_ADC_CFG_IDX, g_AdcBandgap,
    g_AdcVinn, data[i]); // convert from number to mV.
    //PRINTF("Original: 0x%x\t Ch: %d\t Result: %f(mV)\r\n", data[i], DEMO_ADC_CHANNEL, fresult);
}
light1 = fresult[5]; // light reading is in mV.
light1 = light1/27000; //convert to mA
light1 = light1/1000; // convert to uA.
light = (int) light1; // convert to integer;
PRINTF("Light reading: %d \r\n", light); // output light reading to the console.

```

Figure 18 - Light read and output to console.

```

//store time and light reading in EEPROM
/* Init Buffer*/
SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality

srcBuff2[0] = 0x06U; //Write Enable register written to.

xfer.txData = srcBuff2; //data to be transfered
xfer.rxData = destBuff2; //data to be received
xfer.dataSize = sizeof(destBuff2);
SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //data sent
SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality

srcBuff[0] = 0x02U; //write command
srcBuff[1] = 0x00U; //24 bit address
srcBuff[2] = 0x00U; //24 bit address
srcBuff[3] = Mem_loc; //memory location to store in
srcBuff[4] = seconds; //store seconds data
srcBuff[5] = light; //store light data

xfer.txData = srcBuff; //data to be transfered
xfer.rxData = destBuff; //data to be received
xfer.dataSize = sizeof(destBuff);
SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //data sent

```

Figure 19 - store time and light reading in EEPROM

```

//EEPROM memory address updated
Mem_loc = Mem_loc+2; //incremented by 2 as two bytes of data was written to.
SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality

srcBuff2[0] = 0x06U; //Write Enable register written to.

xfer.txData = srcBuff2; //data to be transfered
xfer.rxData = destBuff2; //data to be received
xfer.dataSize = sizeof(destBuff2);
SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //data sent
SPI_MasterInit(EXAMPLE_SPI_MASTER, &userConfig, srcFreq); //initialises SPI master functionality
|
srcBuff[0] = 0x02U; //write command
srcBuff[1] = 0x00U; //24 bit address
srcBuff[2] = 0x00U; //24 bit address
srcBuff[3] = 0x01U; //register holding current memory address
srcBuff[4] = Mem_loc; //store seconds data

xfer.txData = srcBuff; //data to be transfered
xfer.rxData = destBuff; //data to be received
xfer.dataSize = sizeof(destBuff);
SPI_MasterTransferBlocking(EXAMPLE_SPI_MASTER, &xfer); //data sent

```

Figure 20 - EEPROM memory address updated

```

//turn off device
/* Shut down higher 120K RAM for lower power consumption */// turn this off
POWER_EnablePD(kPDRUNCFG_PD_MEM9);
POWER_EnablePD(kPDRUNCFG_PD_MEM8);
POWER_EnablePD(kPDRUNCFG_PD_MEM7);
POWER_EnablePD(kPDRUNCFG_PD_MEM6);
POWER_EnablePD(kPDRUNCFG_PD_MEM5);
POWER_EnablePD(kPDRUNCFG_PD_MEM4);
POWER_EnablePD(kPDRUNCFG_PD_MEM3);
POWER_EnablePD(kPDRUNCFG_PD_MEM2);

POWER_Init();

POWER_EnablePD(kPDRUNCFG_PD_FIR); //powers down additional stuff
POWER_EnablePD(kPDRUNCFG_PD_FSP); //Fusion Signal Processor
POWER_EnablePD(kPDRUNCFG_PD_OSC32M);

/* Enable OSC_EN as interrupt and wakeup source */
// SYSCON->PMU_CTRL0 = SYSCON->PMU_CTRL0 | SYSCON_PMU_CTRL0_OSC_INT_MSK_MASK;
/* Power control 1 */
msk = SYSCON_PMU_CTRL1_XTAL32K_PDM_DIS_MASK | SYSCON_PMU_CTRL1_RC032K_PDM_DIS_MASK |
      SYSCON_PMU_CTRL1_XTAL32K_DIS_MASK | SYSCON_PMU_CTRL1_RC032K_DIS_MASK;

#if (defined(BOARD_XTAL1_CLK_HZ) && (BOARD_XTAL1_CLK_HZ == 32768U))
    val = SYSCON_PMU_CTRL1_XTAL32K_PDM_DIS(0U) /* switch on XTAL32K during power down */
        | SYSCON_PMU_CTRL1_RC032K_PDM_DIS(1U) /* switch off RC032K during power down */
        | SYSCON_PMU_CTRL1_XTAL32K_DIS(0U) /* switch on XTAL32K */
        | SYSCON_PMU_CTRL1_RC032K_DIS(1U); /* switch off RC032K at all time */
#else
    val = SYSCON_PMU_CTRL1_XTAL32K_PDM_DIS(1U) /* switch off XTAL32K during power down */
        | SYSCON_PMU_CTRL1_RC032K_PDM_DIS(0U) /* switch on RC032K during power down */
        | SYSCON_PMU_CTRL1_XTAL32K_DIS(1U) /* switch off XTAL32K at all time */
        | SYSCON_PMU_CTRL1_RC032K_DIS(0U); /* switch on RC032K */
#endif

```

Figure 21 - Initialise power down

```

PRINTF(" Power Down.\r\n");

while (1)
{
    /* Enable GPIO wakeup */
    NVIC_ClearPendingIRQ(EXT_GPIO_WAKEUP_IRQn);
    NVIC_EnableIRQ(EXT_GPIO_WAKEUP_IRQn);
    __disable_irq();
    switch_to_OSC32M();
    #if (defined(BOARD_XTAL1_CLK_HZ) && (BOARD_XTAL1_CLK_HZ == CLK_RCO_32KHZ))
        POWER_WritePmuCtrl1(SYSCON, SYSCON_PMU_CTRL1_RCO32K_DIS_MASK, SYSCON_PMU_CTRL1_RCO32K_DIS(1U));
    #elif(defined(BOARD_XTAL1_CLK_HZ) && (BOARD_XTAL1_CLK_HZ == CLK_XTAL_32KHZ))
        POWER_WritePmuCtrl1(SYSCON, SYSCON_PMU_CTRL1_XTAL32K_DIS_MASK, SYSCON_PMU_CTRL1_XTAL32K_DIS(1U));
    #endif

    POWER_LatchIO();
    POWER_EnterPowerDown(0);
    POWER_RestoreIO();
    switch_to_XTAL();

    NVIC_DisableIRQ(EXT_GPIO_WAKEUP_IRQn);
    NVIC_ClearPendingIRQ(EXT_GPIO_WAKEUP_IRQn);

    #if (defined(BOARD_XTAL1_CLK_HZ) && (BOARD_XTAL1_CLK_HZ == CLK_RCO_32KHZ))
        POWER_WritePmuCtrl1(SYSCON, SYSCON_PMU_CTRL1_RCO32K_DIS_MASK, SYSCON_PMU_CTRL1_RCO32K_DIS(0U));
    #elif(defined(BOARD_XTAL1_CLK_HZ) && (BOARD_XTAL1_CLK_HZ == CLK_XTAL_32KHZ))
        POWER_WritePmuCtrl1(SYSCON, SYSCON_PMU_CTRL1_XTAL32K_DIS_MASK, SYSCON_PMU_CTRL1_XTAL32K_DIS(0U));
    #endif

    /* after waking up from power down, usart config is lost, recover it */
    BOARD_WakeupRestore();
    __enable_irq();

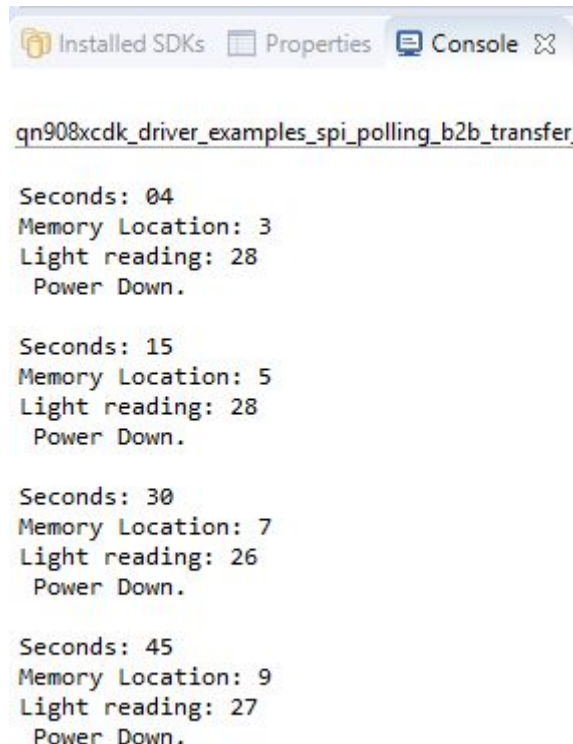
    RTC_FreeRunningEnable(RTC, false);

    delay(1000000);
}

```

Figure 22 - powered down to power mode 1. Wake up with GPIO.

This code ran successfully as shown below.



```

qn908xcdk_driver_examples_spi_polling_b2b_transfer

Seconds: 04
Memory Location: 3
Light reading: 28
Power Down.

Seconds: 15
Memory Location: 5
Light reading: 28
Power Down.

Seconds: 30
Memory Location: 7
Light reading: 26
Power Down.

Seconds: 45
Memory Location: 9
Light reading: 27
Power Down.

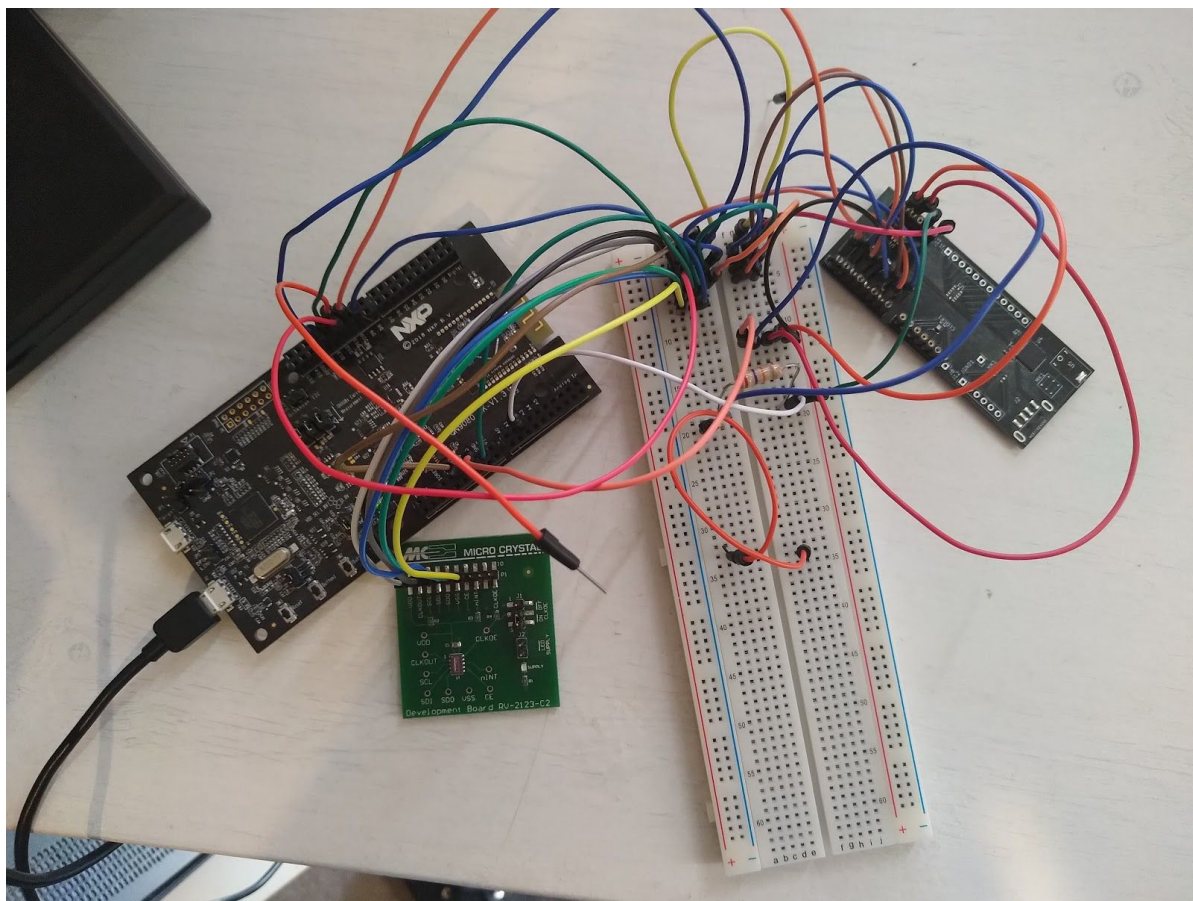
```

Figure 23 - Final code output to console



The first reading is at 4 seconds as this reading was done just based on when the code was uploaded not based on when an interrupt occurred by the external clock. The next reading is 11 seconds later at 15 seconds when the interrupt occurred. Anytime after this the readings were initiated by the interrupt waking the device and as shown this occurred every 15 seconds. The memory location increased by 2 each time which was expected. The light reading was found to be at approximately 27uA which is equivalent to a lux of approximately just below 1000. This is expected as this reading was taken in a room beside a window at 11am. The storage in memory was also verified by running code reading the memory addresses used in this program. It was found to contain the data expected.

Below is an image of the devices all connected for this program to run correctly.



*Figure 24 - QN9080DK (left), clock (middle), eeprom and light sensor(right)*

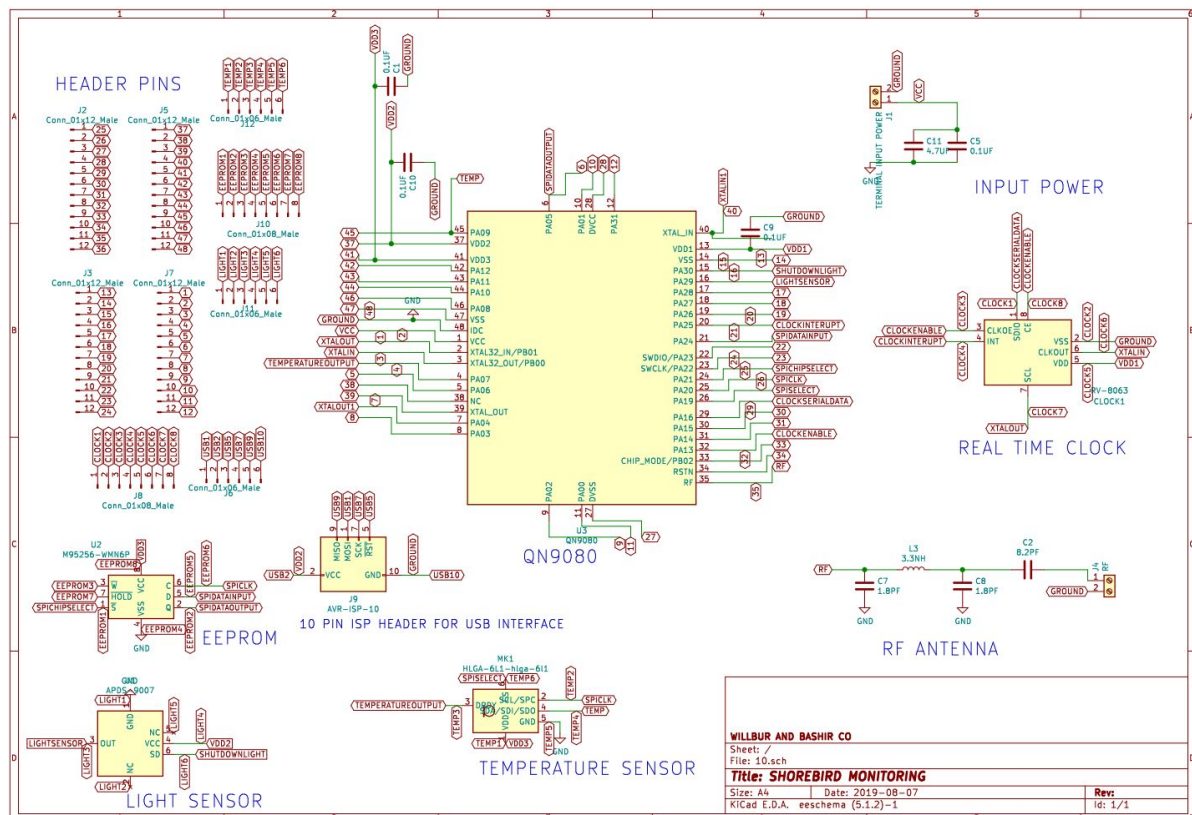
## **Hardware:**

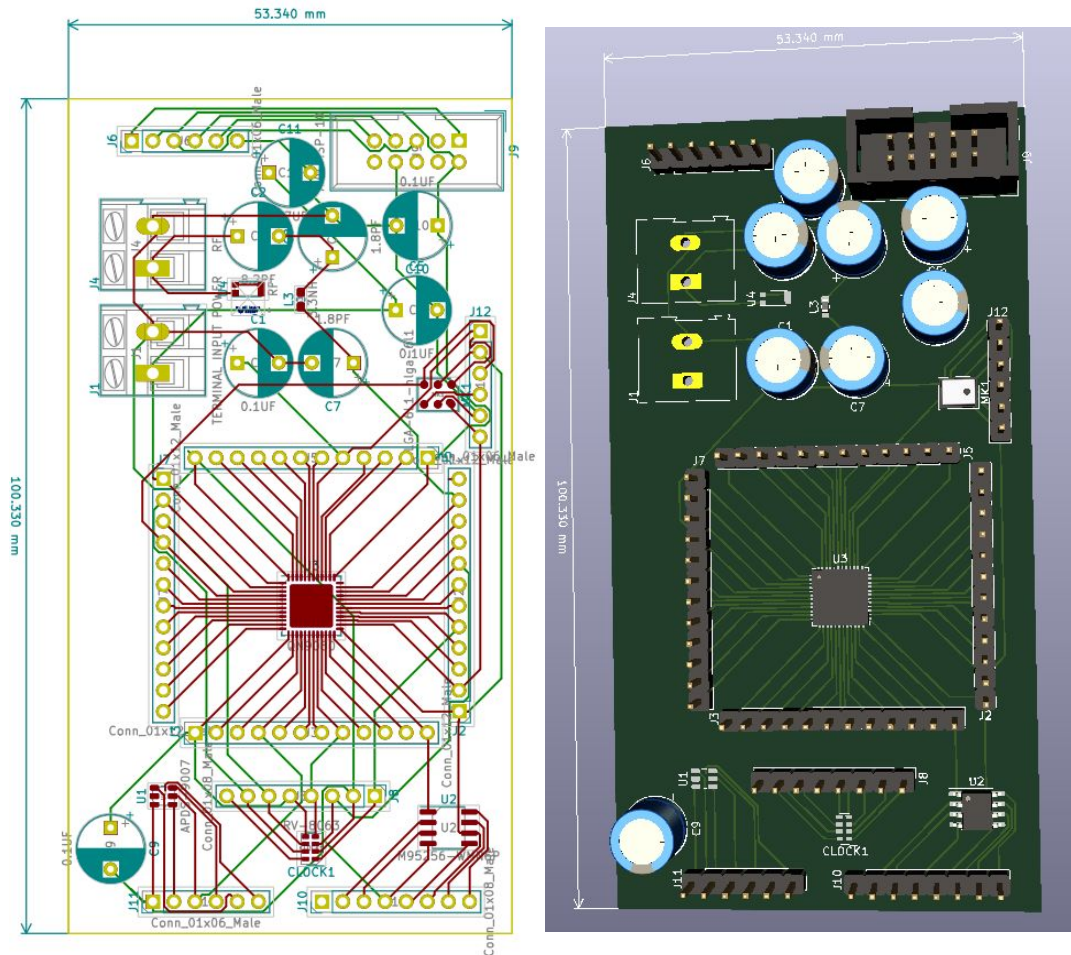
The hardware section of this project proved to be a challenging task as we have designed 2 different PCBs before a final version. We have run across different difficulties during the design process.

The initial prototype design was aimed at achieving a standalone product but on a bigger space to allow for diagnosis and testing purposes. The first design included the use of IO

headers for each component to enable us to manually wire components in case of errors in the schematics. We have also used screw terminals connect our own 5V power supply. As upon inspection and delivery of parts we quickly realise a few mistakes explained below.

1. We have used incorrect footprint for the SoC QN9080. This has occurred as there were multiple footprints under the same QFN-48 package, and unknowingly we have selected the incorrect one.
2. Use of large sized capacitors. We have quickly came to the realisation that when designing small electronics and dealing with small components, we cannot use large capacitors or resistors as they will produce large noise and also small amounts of inductance. This could lead to complications in the PCB and function of the SoC chip.
3. We cannot use screw terminals as a placement for an RF antenna. There is magic science around how RF antennas works and their performance under certain conditions. Some of these are as follows.
  - RF components are to be very close to each other.
  - RF components trace are to be on the same layer with no vias.
  - RF trace must keep a  $50\Omega$  impedance
4. VDD capacitors are too far away from their source and they need to be close to the SoC as much as possible.
5. We did not order a PCB stencil and soldering these components are next to impossible with hands.



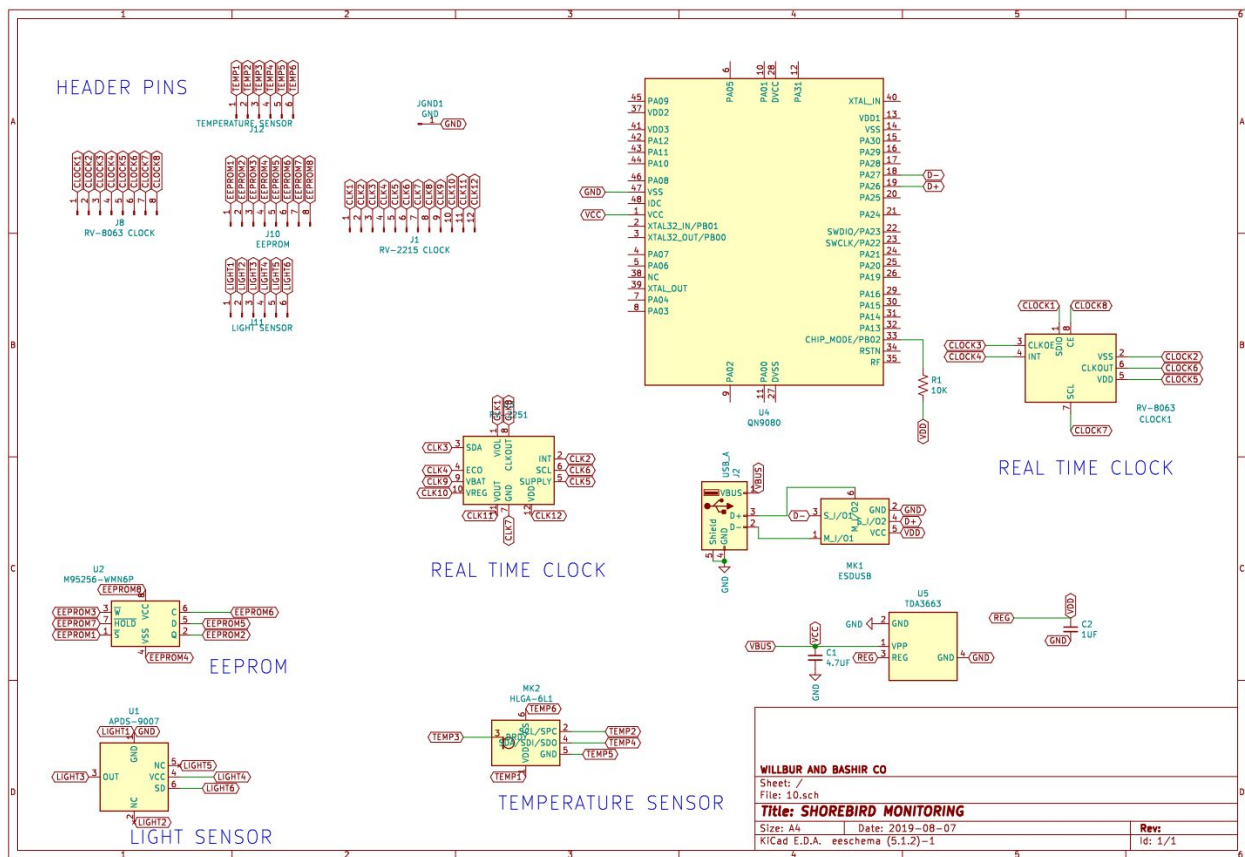


In the next adventure of designing a prototype PCB we have taken a different approach to our problems with minimal risks taken. Our supervisor Paul Beckett has kindly lend us his QN9080 development kit Board. This means we can use the development kit to program all of our requirements for this project, however we still need to connect our required sensors and components( EEPROM, light sensor, temperature sensor, external clock). So we have decided to design a bridging PCB to mount our components and wire these manually to the development kit. This meant we didn't need to worry about RF antenna , power supply and their related components.

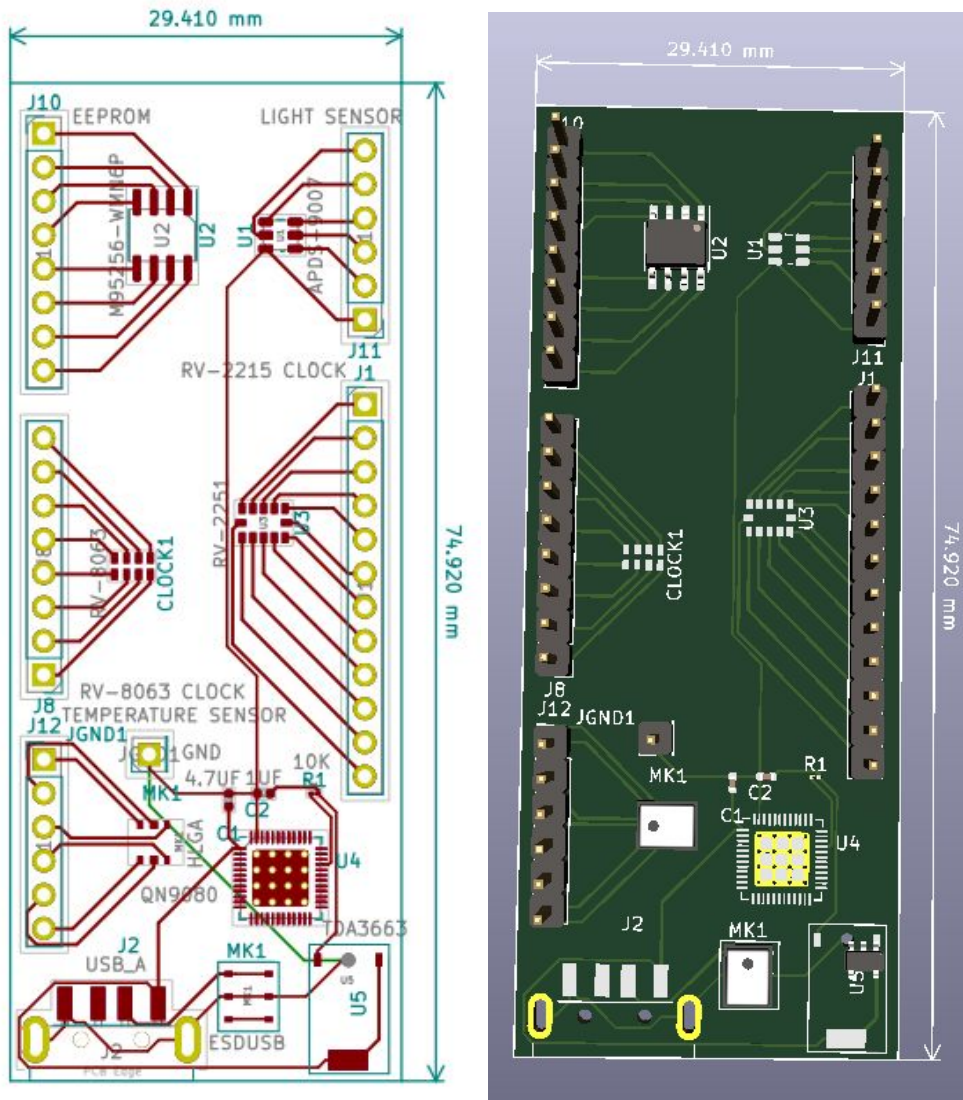
During this PCB design we have also decided to draw the footprint of each component from scratch using the datasheets provided in the supplier's website (element14). This ensures 100% success rate to avoid redesigning a new PCB as the clock is ticking fast.

We have also made an attempt to design a USB port to allow for programming of the QN9080 chip as this PCB will be used to program the chip before it is soldered onto the finalised product.





Using this bridging PCB was proved to be a successful device as we have used it to program, test and diagnose the project.



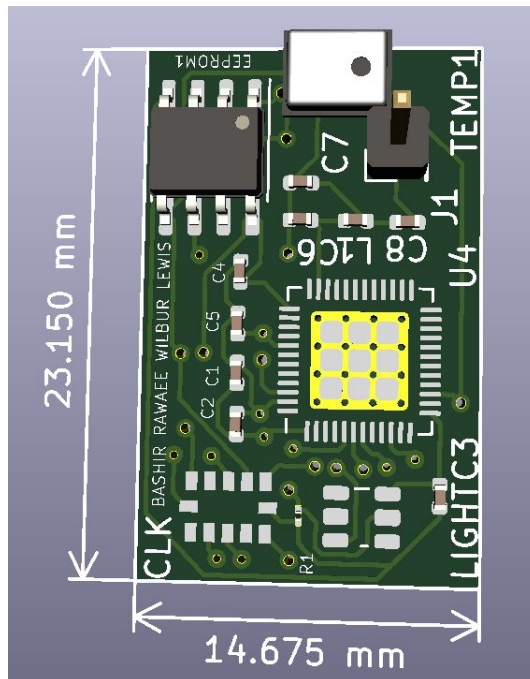
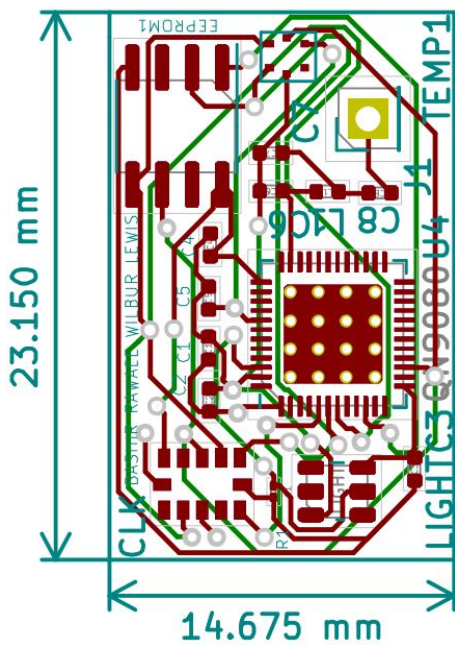
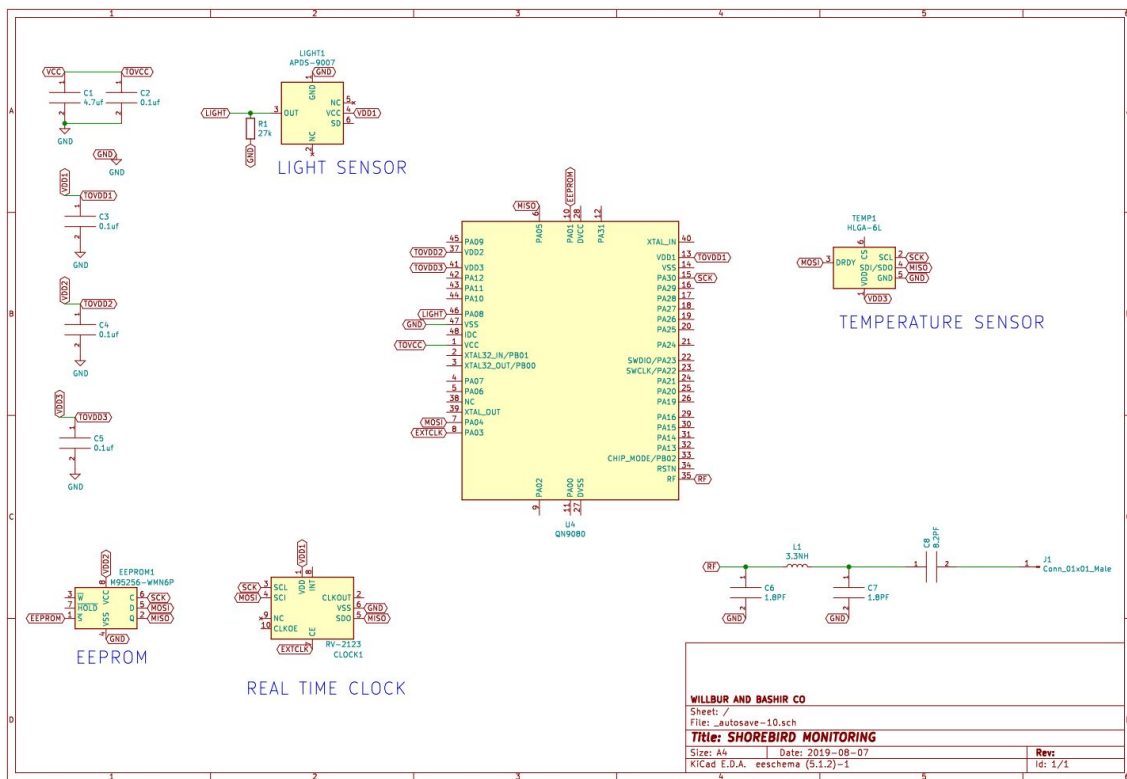
The last and finalised PCB was designed to put together all the components in a tiny package. We ensured that we corrected all technical mistakes and followed all the guidelines and advises of our supervisor. The finalised PCB is a 2 layer board measuring at 23.15mm x 14.675mm x 1.6mm. A two layer PCB was made possible by using a large number of vias to switch from top and bottom layers. We have fulfilled the following requirements for a working product.

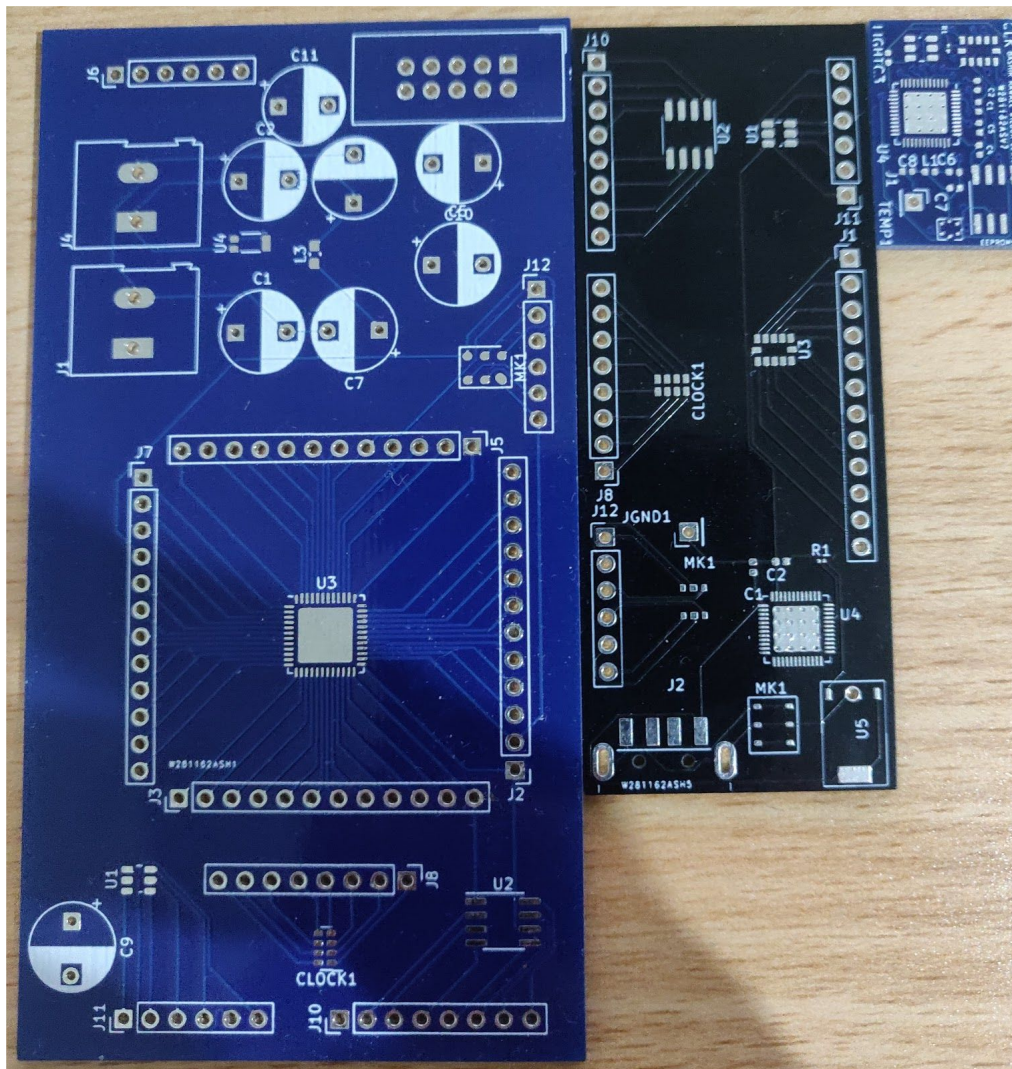
Make sure the product is made as small as possible compared to other competitors for example Lotec's LAT2800S size comes to 36mm x 11mm. This is very crucial as shorebirds cannot carry large items on them and so making these geotags as small as possible allows more accurately natural data.

We have used very small components but we have ensured to select components that are possible to place on PCB by hand. We have decided to use 0401 SMD capacitors and resistors.

We ensured to use a two layered PCB instead of 4 to minimise cost.







Weighing our latest PCB we found it to be 1.2g which is below the weight requirement of 2g. This could be reduced even further as a large amount of the weight is due to the PCB and not the components. Miniaturising will take a large amount of time than we had for this course and requires much planning to ensure signals do no interfere with each other and circuit errors do not occur.

## Budget:

As we have a maximum budget of \$400 per person for a total spending of \$800, we have purchased the following components. The cost of PCB exponentially increased over each purchase. Reasons are as per following:

1. No stencil was purchased for first Prototype PCB.
2. Second PCB we purchased a frameless stencil.
3. Third PCB was purchased using a framed stencil.
4. Paid more for urgent delivery using DHL.

| Component list  | Price    |
|---|----------|
| M95M02, EEPROM, 2 Mbit, 256K x 8bit, Serial SPI, 5 MHz, SOIC, 8 Pins                                      | \$4.40   |
| HTS221TR, Humidity/Temperature Sensor, 0% to 100% RH/-40°C to 120°C Range, I2C, SPI, 1.7V to 3.6V, HLGA-6 | \$4.64   |
| APDS-9007, Ambient Light Photosensor, Miniature, SMD, chipLED, Logarithmic Current, 2 V to 3.6 V          | \$2.17   |
| TERMINAL BLOCK, PCB, 2 POSITION, 30-16AWG   | \$6.36   |
| QN9080C microcontroller, QN908X Family QN908X Series Microcontrollers, ARM Cortex-M4F, 32bit, 32 MHz      | \$81.20  |
| High Frequency Inductor, 3.3 nH, LQG15HN Series, 300 mA, 0402 [1005 Metric], Multilayer, 0.19 ohm         | \$0.92   |
| RV-2123-C2 Real Time Clock 32.768 kHz +/-20 PPM SPI -40/+85C  | \$3.03   |
| SMD Multilayer Ceramic Capacitor, 0.1 µF, 16 V, 0402 [1005 Metric], ± 10%, X7R, C Series KEMET            | \$0.88   |
| SMD Multilayer Ceramic Capacitor, 4.7 µF, 10 V, 0402 [1005 Metric], ± 20%, X5R, C Series TDK              | \$0.99   |
| SMD Chip Resistor, 0402 [1005 Metric], 33 ohm, CRCW e3 Series, 50 V, Thick Film, 63 mW                    | \$0.60   |
| 3.7V 50mAh battery CR1616 Lithium Manganese Dioxide, 50 mAh, Pressure Contact, 16 mm                      | \$1.77   |
| Total   | \$105.19 |

| PCB List      | Cost + Delivery |
|---------------|-----------------|
| Prototype PCB | \$32.57         |
| Bridging PCB  | \$72.54         |
| Final PCB     | \$105.67        |
| Total         | \$210.78        |

|                    |          |
|--------------------|----------|
| Total Project Cost | \$315.97 |
|--------------------|----------|

The cost of making this product would be a lot cheaper once the product is finalised and a BOM( Bill of Material ) was constructed, we can estimate that we can make more of for less than \$50 each. If these are made in larger numbers, it will drive the cost down to a fraction.

### Discussion of results:

When creating a device that is designed to strap onto a bird, there are few critical areas that are to be fulfilled to make sure we have not only a working device but one that captures realistic data. The device needs to be very light in weight, that is it needs to be less than 2g. This isn't because the bird cannot carry this, but this device needs to be light enough to not trigger any unnatural behaviour from the bird.

The second area of focus in the design of the bird geolocator is the size of the device. This in our case would refer to the size of the printed circuit board. All components need to fit in a compact area. This would mean more complicated and multilayered PCBs and the use of surface mounted components.

So with coming up with a concept design we have run into a few problems. Knowing we have to design a compact , light , weatherproof device , it meant more complication of fitting the components. Researched methods include multi layer boards, surface mount devices and reflow ovens come to light.

Although we can increase the number of layers on a PCB to create a more compact, the cost associated with the manufacturing cost of this are much higher. Using surface mount devices which are more compact from their normal through hole versions, need special tools and techniques to complete. Being very lucky, this semester the new SMD labs opened just in time for us to use the brand new solder irons, close up cameras and the new reflow oven.

The device also needs to have enough battery life as well as memory storage to last at least 6 months. Shorebirds often do not return to the same place after leaving until 6 months or even a year. The memory we chose allows us to last greater than a year. Additional memory can also be added if necessary which will increase the size and weight of the device however small modules can be found to be as small as 3x3x0.5 mm and isn't likely to increase the weight by a substantial amount.

It is difficult to calculate how much the battery will last as the program needs to be loaded to the microcontroller and physically test this, however theoretically we can calculate how much the battery will last under optimal conditions.

| Component list                   | Power usage  |
|----------------------------------|--|
| PCB                              | N/A  |
| EEPROM                           | 5 $\mu$ A at standby, 3mA at read/write[18]          |
| Humidity/Temperature Sensor      | 2 $\mu$ A [19]                                       |
| Ambient Light Photosensor        | 230 $\mu$ A [20]                                     |
| Microcontroller, QN9080          | 1.0 $\mu$ A at power down, 806.59 $\mu$ A active[21] |
| High Frequency Inductor          | -  |
| Real Time Clock                  | 130nA [17]   |
| SMD Multilayer Ceramic Capacitor | -  |
| SMD Chip Resistor                | -  |

Assuming the device is in active mode for 1 second before going back to sleep. If it takes a reading every 10 minutes this means it will be in active mode for 144 seconds a day ( $24 \times 6$ ). This means a total of 0.11615 Coulombs are consumed each day by the active mode. In a year this is 42.4 Coulombs ( $0.11615 \times 365$ ). Per year this consumes 11.78mAh ( $42.4/3.6$ ). The real time clock as well as the power consumed during sleep mode can also be calculated.

Seconds in a day =  $60 \times 60 \times 24 = 86400$

Sleep mode power + real time clock =  $1 \times 10^{-6} + 130 \times 10^{-9} = 1130 \times 10^{-9}$ A.

Coulombs per day =  $86400 \times 1130 \times 10^{-9} = 0.097632$ C

Coulombs per year =  $0.097632 \times 365 = 35.64$ C

mAh per year =  $35.64/3.6 = 9.9$ mAh

A total of 21.68mAh is consumed per year when the device takes readings every 10 minutes being in active mode for 1 second. If the device was in active mode for longer such as 3 seconds this would result in a total power consumed of 45.24mAh. The battery we chose for our prototype is of 50mAh. The consumption of the peripherals is hard to calculate as they will be on for such a small amount of time however we believe will not substantially add to the power consumption of our device as they have all been chosen for low power operation.

## Conclusion

In conclusion, during this capstone project we have hit a large learning curve, from PCB design, learning to solder SMD components, working with SPI, analogue devices and designing an algorithm to calculate coordinates. The project proved to be a success as we have successfully achieved the requirements of the course although we had bottlenecks along the way. The functionality of the software was mostly completed using the QN9080-DK. Further development requires us to have more time working with the PCB which we only were able to solder near the end of our project. The project has a lot of room for improvements such as running the program on our printed PCB, establishing the functionality of bluetooth connectivity, miniaturising the device even further, creating a program with an interface to analyse and plot bird data as well as investing into piezoelectric harvesting mechanism for increased battery life.

## Group work member contribution table:

| <b>Section</b>                     | <b>Person(s) responsible and percentage</b><br>(e.g. Person A 70% Person B 20% Person C 10% OR All members contributed equally) |
|------------------------------------|---|
| Executive Summary                  | Bashir (100%)   |
| Statement of problem               | Bashir (60%) - Wilbur (40%)   |
| Background and literature review   | Bashir (60%) - Wilbur (40%)   |
| Methodology and Engineering Design | Bashir (30%) - Wilbur (70%)   |
| Project management and timeline    | Bashir (60%) - Wilbur (40%)   |
| Findings                           | Bashir (40%) - Wilbur (60%)   |
| Discussion                         | Bashir (70%) - Wilbur (30%)   |
| Budget                             | Wilbur (100%)   |
| Conclusion                         | Bashir (50%) - Wilbur (50%)   |

## References

- [1].Department of Environment and Science. (2019). *Shorebirds (Department of Environment and Heritage Protection)*. [online] Available at: <https://environment.des.qld.gov.au/wildlife/threatened-species/shorebirds/>
- [2]E. Bridge, J. Kelly, A. Contina, R. Gabrielson, R. MacCurdy and D. Winkler, "Advances in tracking small migratory birds: a technical review of light-level geolocation", *Journal of Field Ornithology*, vol. 84, no. 2, pp. 121-137, 2013. Available: 10.1111/jof.12011.

- [3]Nxp.com. (2019). *QN9080 DATASHEET*. [online] Available at: <https://www.nxp.com/docs/en/nxp/data-sheets/QN908x.pdf>
- [4]B. Thomas, J. Holland and E. Minot, "Wildlife tracking technology options and cost considerations", *Wildlife Research*, vol. 38, no. 8, p. 653, 2011. Available: 10.1071/wr10211
- [5]"What is animal tracking? | Movebank", *Movebank.org*, 2019. [Online]. Available: <https://www.movebank.org/node/857>. .
- [6]S. Lisovski, C. Hewson, R. Klaassen, F. Korner-Nievergelt, M. Kristensen and S. Hahn, "Geolocation by light: accuracy and precision affected by environmental factors", *Methods in Ecology and Evolution*, vol. 3, no. 3, pp. 603-612, 2012. Available: 10.1111/j.2041-210x.2012.00185.x.
- [7].Birdtracker.co.uk. (2019). *Geolocator models*. [online] Available at: [http://www.birdtracker.co.uk/geolocators\\_2.html](http://www.birdtracker.co.uk/geolocators_2.html)
- [8].Barron, D., Brawn, J. and Weatherhead, P. (2010). Meta-analysis of transmitter effects on avian behaviour and ecology. *Methods in Ecology and Evolution*, 1(2), pp.180-187.
- [9].Migratetech.co.uk. (2019). *Light level geolocation theory*. [online] Available at: [http://www.migratetech.co.uk/geolocation\\_9.html](http://www.migratetech.co.uk/geolocation_9.html)
- [10]S. Lisovski, C. Hewson, R. Klaassen, F. Korner-Nievergelt, M. Kristensen and S. Hahn, "Geolocation by light: accuracy and precision affected by environmental factors", *Methods in Ecology and Evolution*, vol. 3, no. 3, pp. 603-612, 2012. Available: 10.1111/j.2041-210x.2012.00185.x [Accessed 28 June 2019].
- [11].The Migratory Connectivity Project. (2019). *Geolocators - The Migratory Connectivity Project*. [online] Available at: <http://www.migratoryconnectivityproject.org/geolocators/>
- [12]Breakout, E. (2019). *Energy Harvester - LTC3588 Breakout*. [online] Core Electronics. Available at: [https://core-electronics.com.au/energy-harvester-ltc3588-breakout.html?utm\\_source=google\\_shopping&gclid=Cj0KCQjwxMjnBRCTARIsAGWwNBPkp36yWFHxnvKm9ITXEpF-6Y3nDjSAmJI3a79FbILmHKLnuSjkwkAakTJEALw\\_wcB](https://core-electronics.com.au/energy-harvester-ltc3588-breakout.html?utm_source=google_shopping&gclid=Cj0KCQjwxMjnBRCTARIsAGWwNBPkp36yWFHxnvKm9ITXEpF-6Y3nDjSAmJI3a79FbILmHKLnuSjkwkAakTJEALw_wcB) .
- [13]Broadcom.com. (2019). *APDS-9306*. [online] Available at: <https://www.broadcom.com/products/optical-sensors/ambient-light-photo-sensors/apds-9306>
- [14]B-kainka.de. (2019). [online] Available at: <http://www.b-kainka.de/Daten/Sensor/bp103bf.pdf>
- [15]Farnell.com. (2019). [online] Available at: <http://www.farnell.com/datasheets/64618.pdf>



[16]"SPI Tutorial – Serial Peripheral Interface Bus Protocol Basics", *JTAG Boundary-Scan, In-System Programming, & Bus Analyzers - Corelis*, 2019. [Online]. Available: <https://www.corelis.com/education/tutorials/spi-tutorial/>

[17]*Microcrystal.com*, 2019. [Online]. Available: [https://www.microcrystal.com/fileadmin/Media/Products/RTC/App.Manual/RV-2123-C2\\_App-Manual.pdf](https://www.microcrystal.com/fileadmin/Media/Products/RTC/App.Manual/RV-2123-C2_App-Manual.pdf)

[18]"M95M02-DR", *St.com*, 2019. [Online]. Available: <https://www.st.com/resource/en/datasheet/m95m02-dr.pdf>. [Accessed: 10- Oct- 2019].

[19]"HTS221", *Farnell.com*, 2019. [Online]. Available: [http://www.farnell.com/datasheets/2046114.pdf?\\_ga=2.153192575.672711408.1571630993-647593797.1571484475](http://www.farnell.com/datasheets/2046114.pdf?_ga=2.153192575.672711408.1571630993-647593797.1571484475). [Accessed: 11- Oct- 2019].

[20]"Apds-9007", *Broadcom.com*, 2019. [Online]. Available: <https://www.broadcom.com/products/optical-sensors/ambient-light-photo-sensors/apds-9007>. [Accessed: 13- Oct- 2019].

[21]"QN908x Power Consumption Analysis", *Nxp.com*, 2019. [Online]. Available: <https://www.nxp.com/docs/en/nxp/application-notes/AN01786.pdf>. [Accessed: 16- Oct- 2019].