DESIGN OF A SMALL SATELLITE UHF RADIO BEACON FOR IDENTIFICATION, TRACKING, TELEMETRY AND CONTROL

UNSW
THE UNIVERSITY OF NEW SOUTH WALES

CANBERRA

ORIGINATOR – PILOT OFFICER TRAVIS J MCKEE

SUPERVISORS – DR EDWIN PETERS AND DR CRAIG BENSON

THE UNIVERSITY OF NEW SOUTH WALES AT THE AUSTRALIAN DEFENCE FORCE ACADEMY

CONTENTS

QUICK OVERVIEW (ABSTRACT)

The reduced cost of designing, manufacturing, launching and operating small satellites has seen a significant increase in the number of objects deployed into the low earth orbit space environment. Small satellites operated by organizations with little space experience has resulted in a large failure rate, increasing the number of space debris. These two conditions have resulted in a greater reliance on resource-expensive space monitoring equipment to maintain space situational awareness.

The aim of this study is to provide a concept design for a self-sustained satellite radio beacon that can send a unique identification and telemetry data, independent of any satellite system failures, to multiple geographically dispersed ground stations which allows for tracking of the satellites position through time difference of arrival trilateration. The satellite radio beacon system allows for: 1) the collection of telemetry data for fault analysis which could decrease the current satellite failure rate; and, 2) a radio signal for tracking using cost-effective ground stations irrespective of satellite system failures to reduce the reliance on space monitoring equipment and increase SSA in the LEO environment. The study will utilize an agile, iterative design approach in which each component of the small satellite UHF radio beacon system (satellite radio beacon, communications link and ground receiving station) will be designed, tested and verified in sequential order.

This study has produced a solderless breadboard prototype design for the satellite radio beacon that is capable of self-sustained operation in a ground environment.

The communications link, using a LoRa radio module, is capable of transmitting data for the distances expected for a small satellite in a low earth orbit.

The ground receiving station has proven the capability to receive the identification data, telemetry data and transmit a command for satellite control. Further investigation to reduce the uncertainty in the time measurement techniques by the ground receiving station is required to create an accurate tracking capability which would allow the continued development of the satellite radio beacon system.

INTRODUCTION

The reduction in economic and resource cost of designing, manufacturing, and launching a small satellite has led to an increased number of small satellites being operated in the Low Earth Orbit (LEO) space environment. An increase in the number of satellites and space debris in the LEO environment has resulted in new challenges for space situational awareness (SSA), which leads to a larger reliance on resource expensive space station monitoring equipment. The reduced cost of launching a small satellite into the LEO environment is a result of releasing multiple small satellites from a single launch platform, this is commonly known as ride sharing. Ride sharing has resulted in upwards of 100 small satellites being released from the same launch vehicle in a small-time frame. This has resulted in a reduction in SSA immediately after the launch of small satellites and throughout their operational life cycle due to difficulties in identifying individual satellites. The difficulties of identifying small satellites in the LEO space environment results in a greater demand on the limited ground-based optical and radar monitoring resources to maintain an SSA capability. The standardisation of small satellite manufacture has reduced the cost of production allowing government, educational and commercial organisations with little to no space mission experience to create and produce small satellite. The unique development and design process to meet the organisation requirements for each satellite has resulted in a 55% failure rate for academic institutions and a 23% for commercial industry. The cause of a small satellite failure is difficult to determine as the failure can reduce the quantity and quality of satellite telemetry data available for fault-finding. A deficiency of telemetry data can result in the determination of satellite failure being contributed to several causes of possible failure (typically 5-10 possible causes). The ride sharing launch produces additional difficulties in identifying an individual satellite

immediately after release by the ground monitoring stations which leads to an increase in failure rates due to difficulties in creating the initial communication link with the individual satellite. A satellite failure can result in the satellite becoming uncontrollable and/or difficult to track causing it to become a space debris object leading to an increased risk of a collision with other objects within the LEO space environment.

To reduce the small satellite failure rate, increase SSA in the LEO environment and reduce the reliance on ground monitoring equipment a solution needs to be investigated that aims to provide better methods of obtaining satellite identification and telemetry data and provide a cost-effective method of tracking space debris, active and end of life satellites in the LEO space environment.

The purpose of this project will be to design a self-sufficient, independent satellite radio beacon system that can transmit satellite identification and telemetry data and receive control commands using a UHF radio signal that is capable of being tracked using multiple geographically dispersed, cost-effective ground receiving stations.

The satellite radio beacon shall primarily allow for the identification of an individual satellite after launch and its operational life cycle providing greater SSA which leads to a lower risk of collisions in the LEO environment. Multiple graphically dispersed ground mounting stations constructed using low-cost, commercially available components are to be utilised to track the satellite via the beacon radio signal allowing the existing ground-based monitoring systems to focus on other LEO space objects. The beacon will have a secondary function that can provide telemetry data for satellite on-orbit fault finding to facilitate the determination of causes of failure. Determining the actual cause of failure as opposed to having several possible causes of failure is expected to reduce the number of failures in future launches and operations.
This system can be extended to include an alternative communications pathway that can be used to provide limited control of the other satellite systems to offer a redundancy system to correct on-orbit failures. Correcting an on-orbit failure can result in regaining control of the small satellite reducing the number of space debris objects in the LEO environment.

AIM

The aim of this project is to design and produce a ground-tested satellite UHF radio beacon prototype and a cost-effective ground monitoring station prototype that can sustain a communications link for the distances required of a satellite in LEO. In order to achieve this aim, three aspects will be investigated, firstly the satellite radio beacon that is to be a self-sustained UHF communication system capable of operating independently of all other satellite systems for the duration of the satellites operation mission (until deorbit). Secondly, the UHF communications link which must be able to support the transmission of satellite identification and telemetry data up to 2000kms to support the operation and monitoring of satellites in the LEO space environment. Finally, the ground monitoring station which must be capable of capturing the satellites identification data, telemetry data, recording the precise time of arrival of the radio signal and determining its own global location. The ground station will be able to pass on the captured data and measurements to a peripheral device for post processing using the time difference of arrival (TDOA) to determine the position of the satellite through trilateration calculations.

BENEFITS AND VALUE

The increase in satellites launched into the LEO environment and high small satellite failure rate has resulted in additional challenges for monitoring and operating in the LEO environment. The problems to be addressed by this system is:

1. Difficulties in identifying small objects in the LEO environment
2. Increased demand on resource expensive space monitoring equipment to maintain Space situational awareness (SSA)
3. An increase in space debris due to a large failure rate of small satellites

The purpose of designing the satellite UHF radio beacon system is to address the identified problems by providing:

1. A better technique of uniquely identifying an object in LEO
2. A cost-effective system that can track active, End of Life (EoL) and failed satellites
3. A method of obtaining satellite telemetry or health data irrespective of a failure in any other satellite system
4. An alternative communications pathway for satellite control

The standardisation of small satellite design, construction and launch has led to the number of small satellite missions increasing from 20 missions in 2011 to 322 in 2018 with the number estimated to increase to 300 missions in 2019 with between 2000 to 2800 missions to be launched in the next 5 years. A small minority of these planned missions will be delivered to LEO using dedicated launch vehicles, but most are expected to utilize the current rideshare or piggyback launch vehicles. The Indian Space Research Organisation released 104 small satellites into LEO during a 12-minute cluster release from a single launch vessel on the 15th of February 2017, which provided a demonstration of the cluster launch but also presented the problem of identifying a singular satellite. Early identification and connection provide a good basis for mission success which can be achieved using the unique identification transmitted from the satellite radio beacon.

It is estimated that up to 2400 small satellites will launch into the LEO environment in the next five years by the 2020 Space Works market forecast. The high failure rate of small satellites, as discussed in the next paragraph, results in an increase of uncontrollable space debris objects in LEO. The increasing number of satellites and debris objects in the LEO environment result in a greater demand on the resource expensive space monitoring equipment to maintain SSA. The cost-effective tracking function provided by this systems ground receiving station can be used to reduce the reliance on the current space monitoring equipment. This will allow the reallocation of the space monitoring resources to monitor other difficult to track objects in the LEO environment.

When the satellite fails on launch or in orbit then the determination of the failure can be difficult to ascertain as the communications link providing telemetry data to the ground station is often non-functional. The provision of a system that provides telemetry and satellite health data to the ground station irrespective of a failure of any satellite system provides information that can be used to either determine the cause of failure or help reduce the number of possible causes of failure. When the cause of failure is identified or the number of possible causes of failure is reduced then this information can be used to reduce the number of full or partial failures in future small satellite missions. If the cause of failure is diagnosed whilst the satellite is on-orbit from the provided telemetry data, then there remains the possibility that an on-orbit rectification could be carried out and the mission can be partially or fully completed.

The on-orbit rectification will require a communication path to the satellite to be able to carry out any command to bring the affected system/s back to an operational state. If an emergency communication system has not been provided with the satellite system, then this radio beacon could be utilized to provide control commands for the satellite via an alternative communications link.

BACKGROUND

A review of the current systems available has shown that solutions are provided for individual aspects of SSA in the LEO environment, but they do not take a holistic approach to the key parameters of a satellite telemetry, tracking and control (TT&C) system which include satellite identification. There are several systems available that provide identification (CUBIT, SOARS, Passive RF tag, ELROI and LEDSAT) or telemetry (safety radio beacon) and some which provide identification and telemetry (HyELT, RILDOS and IRASAT1). The existing systems deliver solutions for one or two parameters of the TT&C system which address either, but not both problems highlighted in the introduction. The proposed satellite radio beacon system will implement solutions for all four key parameters of a TT&C system that will address both problems identified in the introduction.

The major orbital parameters that will be used for this project are based on a generic small satellite mission in the LEO environment. The most common orbits for a small satellite mission are sun-synchronous with an orbital height of 300-400kms and inclination of 52° or an orbital height of 500-800kms and inclination of 98° with the later parameters used for the testing of this system. The orbital parameters used for testing results in an orbital period of roughly 90 minutes with the view window of each pass being in the vicinity of 8-10 minutes in which the satellite and ground station have a maximum slant range of 2000kms.

THEORY OF OPERATION

The theory of operation of the satellite beacon system is that the system contains its own power, processing and radio subsystems to ensure that it is self-contained and independent for the satellite. Each radio beacon system contains an identifier in the form of a 16-bit address which allows for 65,536 unique addresses to be used simultaneously. Each radio beacon processor has the capability to be linked to other systems within the satellite to allow for the collection of satellite telemetry and health data or to pass a received control command to another system within the satellite. The 16-bit identification address and collected telemetry data are sent via the beacon radio to a ground receiving station to uniquely identify the small satellite and provide satellite telemetry data for tracking and fault-finding purposes. The radio beacon system can receive data from a ground receiving station to provide control of the beacon system and emergency control of the small satellite.

The ground station can receive and process the identification and telemetry data as well as providing a precise time of arrival of the received signal. The precise time of arrival for a unique small satellite beacon signal being received at three globally dispersed ground receiving stations allows the position of the satellite to be estimated. The global position of each ground station must be known with each station being synchronized to the same clock timing to allow for the Time Difference of Arrival (TDOA) calculation technique to approximate the satellites position as shown in Figure 1 (see below). The trilateration of the satellites position from three stations reduces the possible position of the satellite to two spatial locations with one being discarded as unfeasible due to it being below the surface of the earth.

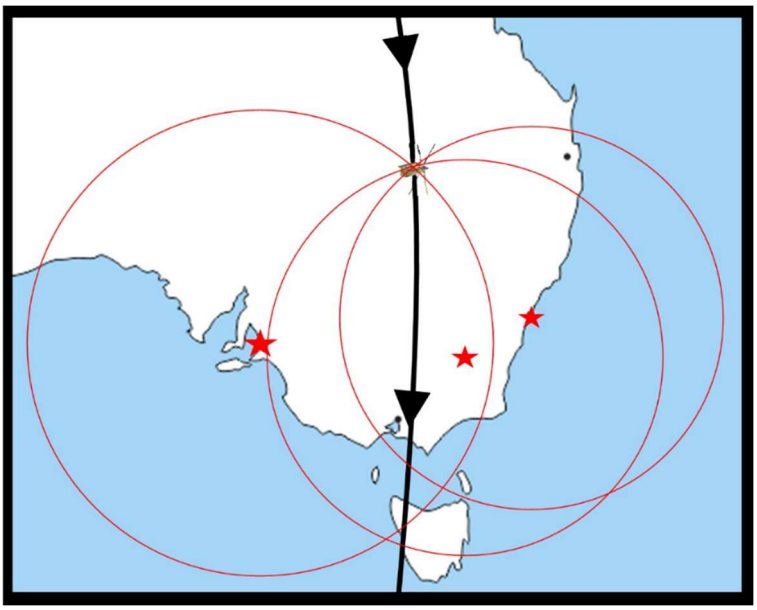
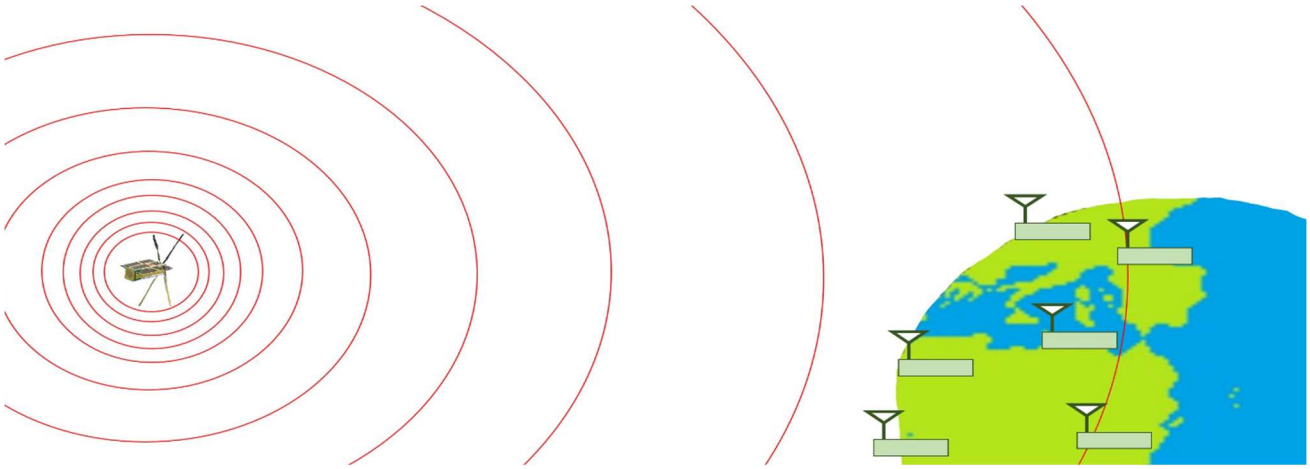


FIGURE 1 - VISUAL REPRESENTATION OF TDOA TRILATERATION WITH 3 STATIONS

A key goal is to ensure that the design of the ground monitoring station can be carried out such that the costs, difficulty of construction and the difficulty in operating the equipment is minimized. The result of minimizing these components of the design will allow a broader spectrum of the public, particularly the growing space enthusiast's community, to create their ground monitoring stations. This allows a larger number of ground stations that are more geographically dispersed which has a two-fold effect of increasing the footprint and tracking reliability of the ground stations and increasing the public awareness and engagement.

SYSTEM OVERVIEW



The design of the satellite UHF radio beacon system will be separated into two major components:

1) **The Satellite Radio Beacon** – The satellite beacon must be a self-sustaining system that can operate independent of all other satellite system. The beacon will need to be capable of transmitting data for a 2000km range.

   To achieve these aims the beacon contains its own:

   - **Processor** –Arduino Pro Mini (APM) module containing an ATMEGA328 processor
   - **Radio Transceiver** –RFM96 LoRa radio module, which uses an ultra-long range spread spectrum communication techniques
   - **Power Generation, Storage and Regulation** – 0.5W Silicon solar panel, super-capacitor storage system and a buck converter regulator

2) **The Ground Receiving Station** – The ground receiving station must be able to capture the transmitted data, record the precise time of arrival of the RF signals, determine its own global location, pass data to a peripheral device and transmit a control command to the beacon.

   To achieve these aims the ground receiving station contains:

   - **Processor** – Arduino Uno development board with an ATMEGA328P processor and 16MHz oscillator
   - **Radio Transceiver** –RFM96 LoRa radio module
   - **GPS Receiver** – U-Blox NEO-7M GNSS module which has a Pulse-per-Second signal synchronized to Coordinated Universal Time (UTC)

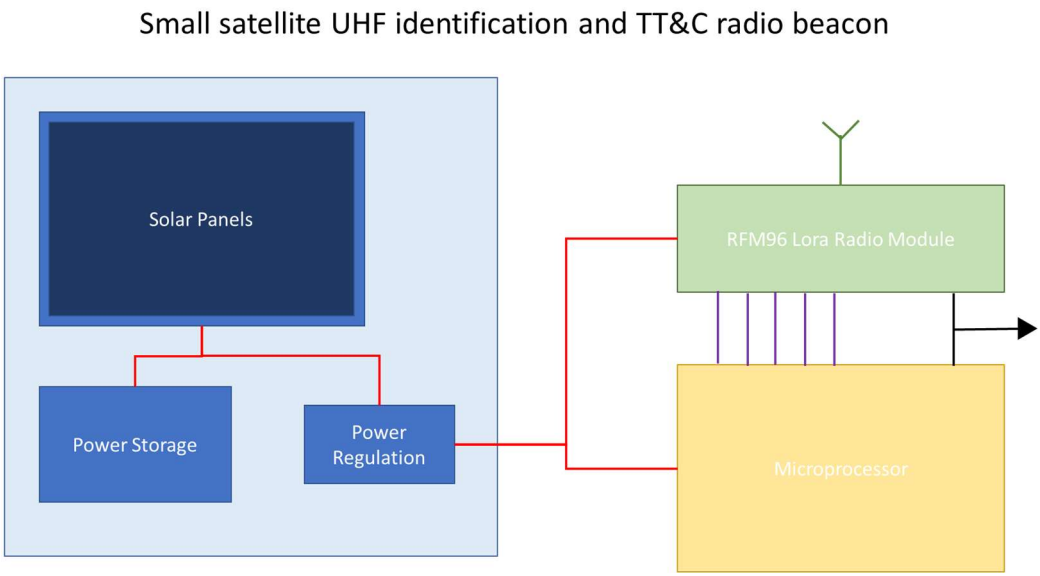


FIGURE 2 - RADIO BEACON SYSTEM BREAKDOWN

The satellite radio beacon will be broken up into a processor system that that contains a microcontroller (Arduino Pro Mini), the radio system (RFM96 LoRa UHF radio transceiver) and a power system (solar generation, regulation and storage) as shown in Figure 2. The first decision in the design process was to select the components for the initial design of the radio beacon with the focus being on components that have low power consumption and can meet the requirements of each sub-system.

The solutions researched for the processor were a Teensy based microcontroller, an Arduino based microcontroller, a raspberry Pi and using a FPGA board with the 3.3V, 8MHz Arduino Pro Mini module using a ATMEGA328P processor being selected for the initial design. The APM was selected as it fit all the requirements of the processor system: it has a proven space heritage, it is well resourced and can be operated with a low supply voltage and clock speed to reduce power consumption. The APM module contains an in-built power regulation system using a Low Dropout (LDO) voltage regulator which will be used in the initial design.

Three radio systems that are capable of long-range communications were found during the initial investigation for the radio design solution: The LoRa spread spectrum system, SIGFOX Low-Power Wide Area Network (LPWAN) system and the NB-IoT LPWAN system. The LoRa radio system was the selected medium for the initial design for it provided a superior point-to-point communications protocol, high immunity to noise and doppler shift, greater software support and can operate in the 70cm (430MHz, RFM96 module) and 33cm (915MHz, RFM95 module) band radio spectrum. A LoRa module breakout board designed by Boyan Nedkov (https://github.com/attexx/rfm9x_breakout_board)was used to allow for compatibility with a prototyping solderless breadboard.

Solar power was the only system considered for the power generation system due to the difficulties of operating other power sources (Lithium-ion batteries, hydrogen fuels cells, nuclear power, thermo-photovoltaic cells, etc.) in a space environment. The solar panel selected for ground testing is a 0.5W, monocrystalline silicon panel from Seeed (https://www.seeedstudio.com/0-5W-Solar-Panel-55x70.html) which can produces a typical output of 5.5V with a current of 100mA at 17% solar conversion efficiency. A capacitor-based energy storage system was selected for the initial design due to the launch isolation and ground testing requirements of a battery-based system.

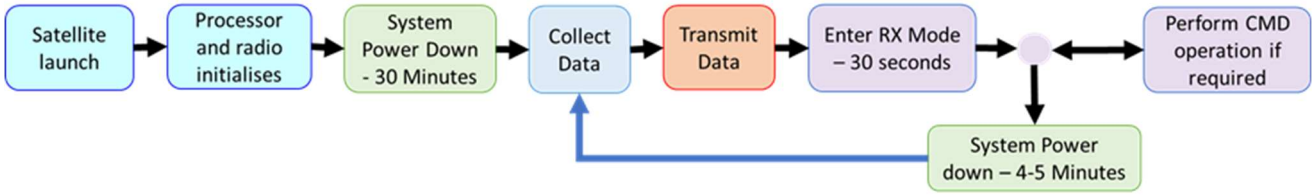


FIGURE 3 - SIMPLE SATELLITE RADIO BEACON SOFTWARE CYCLE

The radio beacon system will operate by the beacon operating on a variable length of time cyclic program in which the system will collect data, transmit the data, wait for a command for a small period then enter a power down mode. Each cycle will last between 4-6 minutes based upon the orbital parameters of a small satellite mission in low earth orbit. An example of a simple beacon cycle is detailed in Figure 3 for the initial development and testing of the system, but noting that it will change depending on the final mission parameters. The beacon cycle must maintain a radio silence for 30 minutes after release from the launch vehicle to satisfy the requirements of the launch provider with the length changing for each provider. The cycle must have a receive period to accept a command to cease radio transmissions to satisfy the requirements of ACMA and the ITU for operating in the RF spectrum. Finally, the length of power down period of each cycle must be varied to ensure that no synchronisations between transmission from sperate beacons.

The final Arduino code used for the initial prototype design for the satellite radio beacon is presented in the software section. The orbital view window of 8-10 minutes will allow at least 1 transmission to occur per satellite pass with 2 transmissions in that window being possible.

The initial design of the satellite radio beacon system was constructed on a prototyping breadboard using Arduino IDE software to operate the APM module The LoRa radio module software was driven by the Radiohead library developed by Airspayce (https://www.airspayce.com/mikem/arduino/RadioHead/).

SATELLITE RADIO BEACON

COMPUTER PROCESSOR SYSTEM

The major considerations for the selection of the computer processing sub-system are:

- The processing power, speed and memory (Flash and EEPROM) to support the tasks required
- Minimize the electrical power consumption of the sub-system to support low power operation
- Has a proven record of operation in the space environment with fast recovery after a power loss
- The processor and the operating system are well supported and resourced
- Ease of design, construction and initialization of the supporting hardware and software

The major systems investigated to meet the task required of the computer processing sub-system was the Raspberry PI, Arduino based microcontrollers, Teensy based microcontrollers (MC) and FPGA based solutions. The power consumption of the Raspberry PI is between 500mA and 1A which is too high for this application and the PCB space requirements meant that it is not suitable solution for this project. The difficulty and lack of support for programming a FPGA array means that this solution was not considered for the initial development of this system. The Teensy based microcontrollers (AT90USB1286, MK20DX128 and MK20DX256) were considered appropriate tools for the system as they meets the processor and memory considerations but the power consumption (approximately 20mA for low power and low clock rate Teensy2.0 and greater than 35mA for Teensy3.0 and above) is greater than other options available for use.

The Arduino based ATmega microcontrollers were selected as they are open source, have better support in terms of hardware and software design, and have a proven history of operation at low clock speeds and voltage reducing the power consumption of the processing sub-system. The ATmega microcontrollers were selected as the operating current is typically low when operating on 3.3V with an 8MHz external clock. The flexibility in the design of a processor system based upon an Arduino based ATmega microcontrollers and the existing resources available means that this solution is the best available for the initial design.

The APM module with the 328P microprocessor was chosen as the user interface (Arduino IDE) is a simple to use program for inexperienced users and has many resources available on the internet but it can also be programmed using a C language program such as Microsoft visual, Visual Source Code (VSC), eclipse or code::blocks for the experienced user to free up memory onboard the module. The initial testing of the system will be carried out using the Arduino IDE to show the proof of concept on the ground, but the full testing for satellite operations should be carried out using a C language program. The 328P processor contains 32K Bytes of flash memory, 1K Bytes of EEPROM and 2K Bytes of RAM available for use with the Arduino IDE bootloader taking up nearly 2KB of the flash memory.

The APM module used was the version that operates on 3.3V and contains an external oscillator for the processor that operates at 8MHz. The 328P processor has a 16MHz internal processor but the speed is lowered to 8MHz to decrease the required operating current. The speed of the processor can be lowered as the space segment will not require a large amount of processing, but the ground segment can be operated at 5V and 16MHz processing speed as it does not have the same power constraints. The RFM96 LoRa radio module and the 328P processor are currently being tested at 3.3V, but the voltage can be lowered to a minimum of 1.8V operating voltage as per the datasheet. More information about power saving for the 328P microprocessor can be found at http://www.gammon.com.au/power and the ATMEL AMEGA328P processor data sheet.

PROCESSOR MODULE POWER REQUIREMENTS

The Arduino Pro Mini module contains a built-in voltage regulator which requires two separate tests to be carried out. The first will test the electrical current consumption of the APM module using the built-in regulator and the second will test the APM consumption for bypassing the in-built regulator. To test the power consumption in both tests, the voltage difference over a known valued resistor placed in series between the power supply and APM module will be measured to determine the amount of current drawn by the APM module as shown below in Figure 4 . This method of calculating the current will be referred to throughout this document as using a 'tester resistor'.

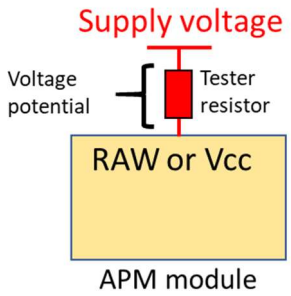


FIGURE 4 - MICROCONTROLLER TESTING SET-UP

Firstly, a 3.3V and 5V power supply will be applied to the RAW pin of the APM module which feeds directly into the built-in fixed Low Dropout (LDO) voltage regulator (p/n MIC5205). This will measure the change in current consumed by the voltage regulator when the input voltage is the same as the output voltage and when there is a large difference between the input and output voltage. The next step will apply a 3.3V regulated voltage to the Vcc pin which bypasses the in-built regulation to check the change in current consumption.

Secondly, the ATMEGA328P processor used in the APM module is capable of being operating in different power modes which consume different amounts of electrical current. The *lowpower.h* file developed by Rocketscream (https://github.com/rocketscream/Low-Power) is used to change the APM power operating mode. The test is carried out by cycling the APM module through the six modes of operation (*power on/normal mode, powerDown, powerSave, powerStandby, powerExtStandby* and *idle*) followed by a 5 second delay before the test cycle is repeated multiple times with each measurement averaged to increase the accuracy of the measurements.

TABLE 1 - AVERAGE CURRENT CONSUMPTION FOR APM MODULE PROCESSOR AND BUILT-IN REGULATOR

Average current consumption for APM module power modes					
Input Voltage and resistor	**RAW=5.0V,1Ω**	**RAW=3.3V,1Ω**	**Vcc=3.3V,1Ω**	**Vcc=3.3V,10Ω**	**Vcc=3.3V,20Ω**
power on current (mA)	50.67	13.78	13.51	14.66	14.40
powerDown current (mA)	42.25	10.00	10.00	11.43	11.94
powerSave current (mA)	42.30	10.00	10.00	11.43	12.10
powerStandby current (mA)	42.59	10.41	10.09	11.53	12.20
powerExtStandby (mA)	42.54	10.41	10.09	11.53	12.20
idle (mA)	46.15	10.86	10.58	11.94	12.50

The results of the tests are presented above with the key findings being:

1. The power consumption of the APM module was much larger than expected, with the cause of the consumption being a surface mount LED that consumed 10mA of current. If the LED is removed, then the current consumption matches the datasheets. The removal of unrequired electrical components minimising the current consumption of the APM module

2. If there is a large difference in the input and output voltage for the MIC5205 then the current consumption of the APM module increases significantly. An investigation into using different types of electrical voltage regulators should be carried out.
3. Placing the ATMEGA328P processor into *powerDown* or *powersave* mode wherever possible will reduce the most current consumption of the radio beacon but any low power mode will reduce the current.

The APM module with the ATMEGA328P processor when operated at 3.3V with an 8MHz processor has been tested and verified to consume a small amount of current. Further testing of the other components of the radio beacon system is required to be carried out to confirm if the total current consumption of the beacon is acceptable. If the total current consumption is required to be reduced further, then the operating voltage of the 328P processor can be reduced to 1.8V. Reducing the operating voltage of the processor requires the bootloader and programming file for the processor to be updated for the Arduino IDE.

The components that are used on the APM module (supporting components) are not verified for space operations, so an investigation into selecting space-ready components and producing a PCB design is required before mission readiness testing can commence.

RADIO COMMUNICATIONS SYSTEM

The main considerations for the selection of the radio sub-system are…

- Long-range communications (nominally 1000-1200kms, but up to 2000kms)
- Low power consumption during all phases of operation
- Noise tolerance and reduction of interference including RF spectrum considerations
- Minimizing Antenna requirement
- Ease of integration with the APM module and Arduino IDE software
- Well supported system with a proven space heritage

The major available low power, long range communications systems that were considered for this application were SIGFOX, LoRa and NB-IoT. The SIGFOX and LoRa systems utilize the unlicensed ISM bands, whereas the NB-IoT system requires the use of the licensed LTE bands so the NB-IoT will not be considered a solution for this project. The main difference between the SIGFOX and LoRa systems is that the LoRa system can use point-to-point (P2P) communication whereas the SIGFOX uses a Low-Power Wide Area Network (LPWAN) to collect and distribute the data. The P2P communications protocol is considered a better option then the LPWAN as there will be no reliance or cost on utilizing a third-party infrastructure to collect the data and there remains greater flexibility in the communication system design and operation.

The LoRa radio module can operate in the 433MHz band (RFM96 model) and in the 915MHz ISM band (RFM95 model) with the former being selected to be used as this part of the spectrum can be used in more regions around the world. There is a portion of the spectrum (435-438MHz) which is dedicated from amateur satellite use that can be utilised. The ACMA and ITU require a license before this part of the spectrum can be used in space but it can be used on the ground for testing. The LoRa radio module is controlled using the RadioHead library for use with the Arduino IDE or Visual Studio Code programs which allows for rapid development of software supporting the communications link. The HopeRF RFM96 radio module was selected as the data sheet indicates that the module is cable of operating at 3.3V whilst consuming between 20mA and 120mA during a transmission (for a TX power of +5dBm and +20dBm respectively) with a maximum link budget of 168dBm and a receive sensitivity down to -148dBm . The LoRa has a demonstrated history of operations in a space environment and can operate over the distances required which will be verified through ground testing to ensure that it will meet the link budget.

LoRa MODULE POWER REQUIREMENTS

The testing configuration for all three RFM96 LoRa module tests are the same with the current consumption of the module being calculated using a tester resistor in series between the power supply and the 3.3V pin on the LoRa module. The RFM96 software is configured such that the radio operates at 437Mhz in the LoRa packet mode with the (0) default radio settings are used [Bandwidth = 125 kHz, Coding rate = 4/5, Spreading factor = 7 (128 chips/symbol) and Cyclic Redundancy Check (CRC) on]. It is noted that the (0) default radio settings is for medium range, medium data rate applications but it will allow the current consumption of the radio to be characterized with the final radio settings being determined by the testing of the communications link. The configuration of the APM processor and LoRa radio module used for all LoRa consumption testing is shown on the next page in Figure 5.

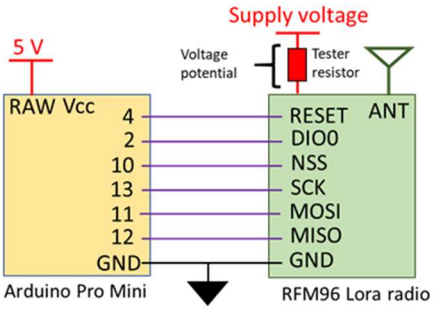


FIGURE 5 - LORA RADIO TESTING CONFIGURATION AND CONNECTIONS

The first test will determine the current consumption of the LoRa radio module during each mode of radio operation (*sleep, receive, transmit and idle*) available using the RadioHead library developed by Airspayce (https://www.airspayce.com/mikem/arduino/RadioHead/). Each mode of radio operation will be activated in sequence during one testing cycle which is repeated several times with different values of tester resistor to find the average current consumption. When the radio is set to *transmit* mode there is no data being transmitted from the LoRa module with the idle consumption of the transmit mode being checked and not the active mode which will be checked in the next test. The active *transmit* mode current consumption is expected to much higher than the *transmit* mode with no transmission occurring.

The second test will check the difference in current consumption of the LoRa module transmitting 30 bytes of data when the transmit power is increased from 5dBm to 23dBm. The transmit power is increased in 1dBm increments over several transmit cycles using different values of tester resistors to determine the average current consumption of the radio module. During this test, the beacon receives a command from a ground receiver station during the *receive* phase and in response to receiving a command the beacon displays a message on the Arduino IDE serial monitor.

The last test will measure the transmission time when the size of the transmitted radio packet is decreased from 250 bytes to 5 bytes in 5 byte increments when using a variety of transmit powers (5, 10, 15 and 20dBm). The current consumption will be measured using different tester resistors to average the results which are compared against the LoRa modem calculator tool supplied by the chip manufacturer, Semtech.

TABLE 2 - RFM96 LORA MODULE AVERAGE CURRENT CONSUMPTION FOR EACH MODE OF OPERATION

RFM96 LoRa module current measurements for each mode			
Series resistor value (Ω)	**1**	**10**	**20**
Sleep current (mA)	0.70	0.62	-0.09
Receive current (mA)	11.00	10.00	11.67
No data transmit current (mA)	2.00	2.00	4.40
Idle current (mA)	2.00	2.00	2.00

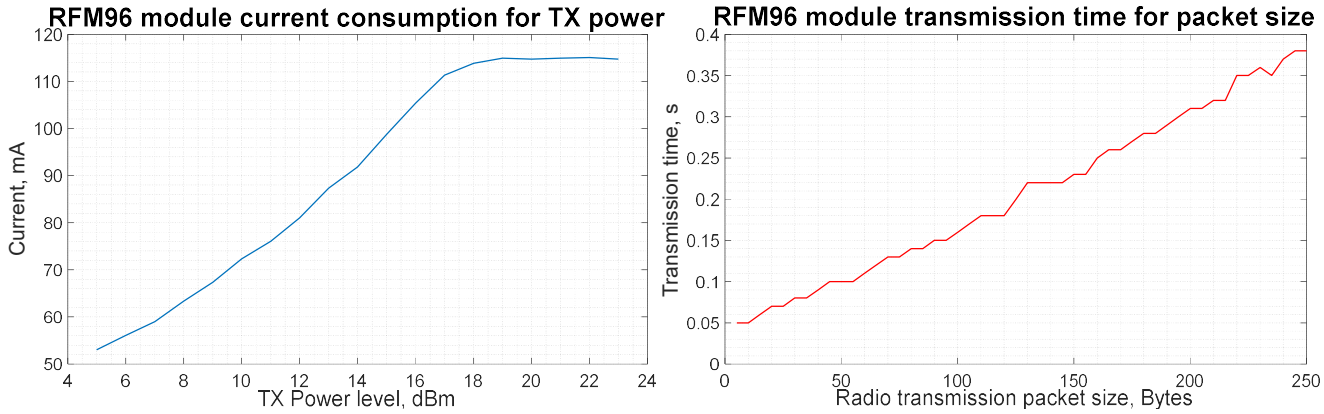


FIGURE 6 - RFM96 MODULE CURRENT CONSUMPTION FOR TX POWER LEVEL AND TRANSMISSION TIME FOR RADIO PACKET SIZE RESULTS

The results from the testing are shown on the previous page in Table 2 and Figure 6 and show that the current consumption of the RFM96 LoRa radio module can be minimised by:

1. Placing the radio into *sleep* mode where possible as it consumes the smallest amount of current
2. Reducing the receive period to as small as possible as it consumes the most current, though it is noted that the radio is placed into transmit mode with no transmissions being carried out and this would consume the most current
3. Use the lowest transmit (TX) power to achieve the successful transmit of data to a ground station
4. Minimising the size of radio packets to effectively communication the information

The power consumption of the LoRa module has been characterised but further testing of the other components is required to ensure the beacon operation can be supported. The radio settings and TX power for the LoRa radio module will be determined using the results of the communication link testing.

During the second test it was verified that the LoRa module can receive and act on a correct command received from a second LoRa radio. It was also shown during the testing that if an incorrect command was received then the program ignored the received transmission and the radio beacon software cycle continued. This system will require further investigation to ensure that it cannot be exploited by nefarious people performing clandestine operations. To prevent this a secure receive function is required to be developed in the Radio beacon software with a good start would be to look into using the open source tiny AES library.

LORA MODULE DROPPED/MISSING PACKETS

The initial testing of the RFM95/96 radio module connection with the APM module indicated that some packets of information were not received by a second LoRa transceiver. A software defined radio (RTL-SDR USB dongle) was used with *Qprx* software on a Linux computer to verify that the radio packets were transmitted. This confirmed that the receiving LoRa module was responsible for dropping or missing the transmitted packets. The testing was carried out by sending 10,000 radio packets containing one randomly generated Byte (a Hexadecimal character) in 100 batches each containing 100 packets. The radio was initiated using the default settings with the coding rate changed between 4/5 and 4/8 with the results showing that the number of packets that were not received decreasing as the coding rate increases (as expected). When the coding rate is at 4/8 then the number of packets dropped by the receiver was 8 in every 10,000 packets approximately. When the coding rate is dropped to 4/5 then the rate increased to between 20-30 dropped packets per 10,000 packets sent. The LoRa radio receiver drops the incoming data due to multiple reasons with the two most probable reasons for not reading the data is if there is an error in the message preamble (*ValidHeader*) or if the cyclic redundancy check (CRC) fails due to an error in the message (*PayloadCRCError*).

Using the LoRa datasheet, it was determined that the CRC error check may be able to be turned off using the Radiohead library function setPayloadCRC() which is used in the receiver only. This function sets the receiver *RegModemConfig – RxPayloadCrcOn* such that the CRC information is not extracted from the received packet header. When the CRC check was turned on or off then there was not a significant change in the number of radio packets that were dropped. The same test as above (100 batches of 100 radio packets) was carried out where each Byte that was received by the LoRa receiver was compared against the originally generated random data. The results show that no Bytes received by the Lora radio contained any errors which indicate that the CRC checks is carried out regardless of the setting the CRC check on or off and that the packets being dropped are due to an invalid header in the preamble. Further testing was carried out where each radio packet was increased to contain 10 Hexadecimal Bytes, with each batch contain 10 radio packets and 1,000 batches sent in total. The results of this testing corresponded with the previously obtained results in which the determining factor of the dropped packets was inconclusive and no Bytes with errors were processed by the receiver.

The functions txGood(), rxGood() and rxBad() from the RHGenericDriver class library were used to determine if the transmitting radio fails to send data or if the receiving radio fails to process the data due to errors in the preamble or payload data. The txGood() function returns the number of packets successfully transmitted by maintaining a running total. The rxGood() function returns a running total of the number of good packets received and the rxBad() function returns a running total of the number of bad received packets which were rejected and not delivered to the application (APM module). For the data to be passed from the FIFO register to the APM module then the following flags are checked in the *RegIrqFlags* register…

- *ValidHeader*
- *PayloadCrcError*
- *RxDone*
- *RxTimeout*

If any of the flags are set to on, then the received packet is dumped from the FIFO register and the Arduino module does not receive the sent data. To test this, the 1 byte per radio packet, 100 packets per batch and 100 batches test program was used where a running total of the good/bad received packets and good transmitted packets are monitored. When the SetPayloadCRC() is set to true then 10,000 good packets were transmitted, with 9915 packets received with 85 packets not being seen by the Arduino module. Of the 85 dropped packets, 79 were dropped due to an error flag in the *RegIrqFlags* register and 6 not being received at all due to a failure to detect the preamble (preamble CRC check). When the SetPayloadCRC() is set to false then number of dropped packets decreases to 12 with 3 packets received with an error flag and 9 not received at all (preamble failure). The testing of the LoRa radio settings to reduce the number of packets being dropped or missed could not produce any consistent results for a set combination of settings.

When investigating the LoRa radio datasheet, it was determined that when the radio operates in LoRa mode then the radio registers/address cannot be changed to prevent the FIFO from clearing if the radio receives a packet containing a CRC with an error. The proprietary nature of the LoRa software prevented further investigation of the reason why the packets were being dropped and prevented access to methods that could bypass the LoRa receive process. It was decided at this point to not pursue this fault any further at this point and accept that there will be a Packet Error Rate (PER) up to 1% by the LoRa module as stated in the data sheet. To mitigate the risk of not receiving the satellite ID it will be transmitted in multiple packets so that the loss of 1 to 3 packets for every 4 sent will still allow the data to be useful and usable in this application.

BEACON TRANSMITTED DATA FORMAT

The data packets will be broken up into 2 elements, the first containing the satellite unique identification number and the second containing the satellite telemetry information. The satellite ID will be represented by 4 hexadecimal characters which allows 65,536 combinations for the address which has enough combinations to cover the predicted number of LEO satellite launches for the next 20 years. Each satellite ID will consist of 2 Bytes representing 4 hexadecimal characters (0xffff). The first byte within a satellite ID radio packet will be a unique packet identifier followed by two sets of satellite ID. The satellite ID packet will then be repeated four times to reduce the probability of the ground station not receiving the satellite ID to 0.00001% ($1\%^4$). These packets will then be preceded by a separate radio packet containing 50 Bytes of satellite telemetry data which can be changed for each satellite application. The format of the transmission of data from the satellite radio beacon is shown below in Figure 7.

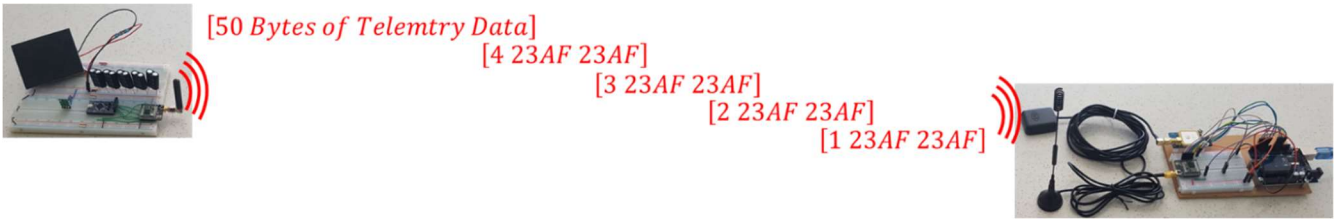


FIGURE 7 - SATELLITE BEACON RADIO PACKET TRANSMISSION FORMAT

The satellite ID will be sent eight times which increases the bit energy of the ID data to minimise bit errors and maximise the likelihood of receiving the satellite ID. The ground receiving station will be able to process each bit of the hexadecimal data using a majority polling to provide an additional error checking process for the satellite ID data.

COMMUNICATIONS LINK BUDGET

The LoRa radio module must be verified that it can transmit data from the radio beacon in LEO to a ground receiving station. The orbital parameters show the expected maximum slant range is 2000km which equates to a Free-Space Path Loss (FSPL) of 151.3dB with these values used for testing.

The first step in testing the communication link is to obtain the radio settings for the 0, 2 and 3 default radio settings in the RadioHead Library. These settings were then used to estimate the receiver sensitivity and link budget using the LoRa modem calculator tool available from Semtech (https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R000000HUhK/6T9Vdb3_ldnElA8drIbPYjs1wBbhlWUXej8ZMXtZXOM). The radio was initiated using the RadioHead default settings results in the following settings and estimates from the LoRa calculator tool:

Bw125Cr45Sf128 - Bandwidth = 125kHz, Coding Rate = 4/5, Spreading factor = 7 (128 chips/symbol)
Resultant data rate is 3125bps and time-on-air is 57.86 mS (obtained using the Lora calculator)

TABLE 3 - LORA CALCULATOR ESTIMATES FOR THE (0) RADIOHEAD DEFAULT SETTINGS

4 Byte identification radio packet					
TX Power (dBm)	Bit Rate (bps)	Time on Air (mS)	Link Budget (dB)	Receiver sensitivity (dBm)	Transmit current (mA)
5	3125	59.9	132	-127	25
10	3125	59.9	137	-127	31
15	3125	59.9	142	-127	82
20	3125	59.9	147	-127	125
50 Byte telemetry radio packet					
TX Power (dBm)	Bit Rate (bps)	Time on Air (mS)	Link Budget (dB)	Receiver sensitivity (dBm)	Transmit current (mA)
5	3125	182.78	132	-127	25
10	3125	182.78	137	-127	31
15	3125	182.78	142	-127	82
20	3125	182.78	147	-127	125
				Total Transmission Time (mS)	422.38
				Total Transmission Time (S)	0.42238

To get longer range for the communications, then the modem configuration can be changed to the modem settings Bw31_25Cr48Sf512 or Bw125Cr48Sf4096 where the settings are….

Bw31_25Cr48Sf512 - Bandwidth = 31.25kHz, Coding Rate = 4/8, Spreading factor = 10 (512 chips/symbol) Resultant data rate is 152.59 bps and time-on-air is 1122.3 mS

TABLE 4 - LORA CALCULATOR ESTIMATES FOR THE (2) RADIOHEAD DEFAULT SETTINGS

4 Byte identification radio packet					
TX Power (dBm)	Bit Rate (bps)	Time on Air (mS)	Link Budget (dB)	Receiver sensitivity (dBm)	Transmit current (mA)
5	152.59	1056.77	144.4	-139.4	25
10	152.59	1056.77	149.4	-139.4	31
15	152.59	1056.77	154.4	-139.4	82
20	152.59	1056.77	159.4	-139.4	125
50 Byte telemetry radio packet					
TX Power (dBm)	Bit Rate (bps)	Time on Air (mS)	Link Budget (dB)	Receiver sensitivity (dBm)	Transmit current (mA)
5	152.59	3416.06	144.4	-139.4	25
10	152.59	3416.06	149.4	-139.4	31
15	152.59	3416.06	154.4	-139.4	82
20	152.59	3416.06	159.4	-139.4	125
				Total Transmission Time (mS)	7643.14
				Total Transmission Time (S)	7.64314

Bw125Cr48Sf4096 - Bandwidth = 125kHz, Coding Rate = 4/8, Spreading factor = 12 (4096 chips/symbol) Resultant data rate is 183.11 bps and time-on-air is 860.16 mS

TABLE 5 - LORA CALCULATOR ESTIMATES FOR THE (3) RADIOHEAD DEFAULT SETTINGS

4 Byte identification radio packet					
TX Power (dBm)	Bit Rate (bps)	Time on Air (mS)	Link Budget (dB)	Receiver sensitivity (dBm)	Transmit current (mA)
5	183.11	1056.77	143	-138	25
10	183.11	1056.77	148	-138	31
15	183.11	1056.77	153	-138	82
20	183.11	1056.77	158	-138	125
50 Byte telemetry radio packet					
TX Power (dBm)	Bit Rate (bps)	Time on Air (mS)	Link Budget (dB)	Receiver sensitivity (dBm)	Transmit current (mA)
5	183.11	2891.78	143	-138	25
10	183.11	2891.78	148	-138	31
15	183.11	2891.78	153	-138	82
20	183.11	2891.78	158	-138	125
				Total Transmission Time (mS)	7118.86
				Total Transmission Time (S)	7.11886

TABLE 6 - LORA MODULE RADIOHEAD SETTINGS AND CALCULATOR TOOL ESTIMATES

RadioHead default setting	LoRa module radio configuration	Bandwidth	Coding rate	Spreading factor	Preamble length	Estimated Bit rate	Estimated Link Budget with 5dBm	Estimated receiver sensitivity (dBm)
(0)	Medium range & data rate	125 kHz	4/5	7 (128 chips/symbol)	12	5469 bps	129	-124
(2)	Long range, slow data rate	31.25 kHz	4/8	10 (512 chips/symbol)	12	152.59 bps	144.4	-139.4
(3)	Long range, slow data rate	125 kHz	4/8	12 (4096 chips/symbol)	12	183.11 bps	143	-138

The results from the LoRa calculator estimate are shown above in Table 6 and show that the (2) settings produces the lowest estimated receiver sensitivity. A test was carried out where the transmission time of each setting was measured for the LoRa module and compared against the LoRa calculator estimates, The results are presented below in Table 7. The results show that the LoRa calculator is relatively accurate but was out a little bit for the (2) setting transmission times. The results from the LoRa calculator tool indicate that the (2) settings will be the better selection for the beacon radio as the receiver sensitivity is lower (higher link budget), the data rate are similar and smaller transmission time (less Time-on-Air).

TABLE 7 - LORA CALCULATOR TRANSMISSION TIME ESTIMATES AND TESTING MEASUREMENTS FOR THE RADIOHEAD DEFAULT SETTINGS

RFM96 LoRa radio transmit times for RadioHead default settings			
RadioHead default setting	**(0) Bw125Cr45Sf128**	**(2) Bw31_25Cr48Sf512**	**(3) Bw125Cr48Sf4096**
Estimated Ident TX time (s)	0.06	1.057	1.057
Estimated telemetry TX time (s)	0.183	3.416	2.892
Estimated total TX time (s)	0.423	7.643	7.119
Measured ident TX time (s)	0.03	0.59	1.01
Measured telemetry TX time (s)	0.09	2.02	3.04
Measured total TX time (s)	0.23	4.39	7.1

The next step was to use the radio settings and communication system values to calculate the Link Budget for each RadioHead setting with various TX powers (5, 10, 15 and 20dBm). The Link budget for the communications link was broken up into 2 groups of calculations with the second group of calculations being carried out using two different methods. The first group of calculations was determining the link budget for the system hardware from radio transmitter to where the signal enters the radio receiver and the second group calculating from the receiver onwards including any software gains. A graphical representation of the Link budget is displayed below in Figure 8 followed by the equations used in the calculation.

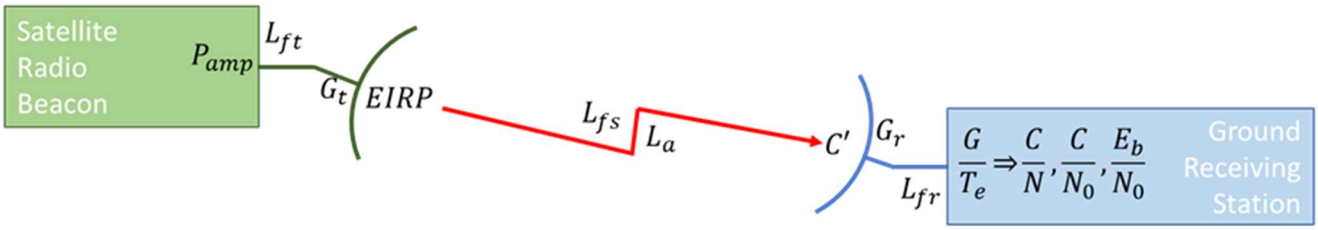


FIGURE 8 - SATELLITE RADIO BEACON COMMUNICATIONS LINK BUDGET REPRESENTATION

The first group of calculations from the transmitter amplifier to the receiver amplifier is as follows…

Transmitter EIRP (EIRP) [dB] – $EIRP = P_{amp} - L_{bo} - L_{ft} + G_t$

Free Space Loss (L_{FS}) [dB] – L_{FS} = 92.44+20*log($d_s.f$)

Carrier Power Density (C') [dBW] – C' = $EIRP - L_{FS} - L_A$

Equivalent Noise Temperature (Te) [dBK] – T_e = 10*log($T^*(NF-1)$)

Receiver G/T_e (G/T_e) [dBK^{-1}] – G/T_e = G_r - T_e

The second set of calculations was performed in two ways, with the first method using the maximum bit rate figure obtained from the LoRa modem calculator tool. The bit energy to noise density ratio was calculated using the power to noise density (C/N_o) value with the equations used being…

C/N0 ratio [dB] – C/No = $C' - 10.\log(k) + \frac{G}{Te} - L_{fr}$

Eb/No Ratio [dB] – Eb/No = C/No – 10log(f_b)

The second method manually calculates the maximum bit rate from the bandwidth efficiency while using the Carrier to Noise (C/N) to calculate the bit energy to noise density ratio.

Noise Density [dBW/Hz] – $N_0 = \frac{G}{T_e} - 10 * \log(k) - L_{rf}$

Raw bit rate [bit/s] – $R_b = SF * \frac{BW}{2*SF}$

Effective bit rate [bit/s] – $R_{b\,eff} = R_b * FEC$

Bandwidth efficiency [bit/Hz] – $\eta = \frac{R_b}{BW}$

Total Noise Power [dBw] – $N = N_0 + 10\log_{10}(BW)$

Carrier to Noise [dB] – $\frac{C}{N} = C' - N$

Eb/No ratio [dB] – $\frac{E_b}{N_0} = \frac{C}{N} + 10\log_{10}\left(\frac{BW}{R_{b\,eff}}\right)$

P_{amp} – Transmitter Power (dBW)
L_{bo} – Back-off Loss (earth station only – for this assignment assume to be 0)
L_{ft} – Transmitter feeder Loss (dB)
G_t – Transmitter antenna Gain (dB)
d_s – slant range (kms)
f – frequency (GHz)
EIRP – Effective Isotropic Radiated Power (dB)
L_{FS} – Free space Loss (dB)
L_A– Atmospheric Loss (dB)
T – Receiver Environmental temperature (K)
NF – Receiver Noise Figure
G_r – Receiver Antenna Gain (dB)
T_e – Equivalent Noise Temperature (dBK)
f_b – maximum bit rate (bit/s)
C' – Carrier Power Density (dBW)
L_{fr} – Receiver feeder Loss (dB)
k – Boltzmann's constant (1.36*$10^{-23}\ Jk^{-1}$)
SF – Spreading factor
BW – Bandwidth (Hz)
R_b – Raw bit rate (bit/s)
FEC – Forward error correction code rate
N_0- Noise density (dBw/Hz)
N – Total noise power (dBw)
$\frac{C}{N}$ – Carrier to Noise (dB)
$R_{b\,eff}$- Effective bit Rate (bit/s)

TABLE 8 - LINK BUDGET CALCULATION AND RESULTS FOR THE RADIO (0) DEFAULT SETTINGS

Link Budget for LoRa RFM96 with (0) default settings					
Parameter	**Symbol**	**Downlink**	**Downlink**	**Downlink**	**Downlink**
Transmitter Power (dBm)	**Pamp (dBm)**	5	10	15	20
Transmitter Power (dB)	**Pamp (dB)**	-25	-20	-15	-10
Frequency (GHz)	**f**	0.437	0.437	0.437	0.437
Transmitter Antenna Gain (dBi)	**Gt (dB)**	3.5	3.5	3.5	3.5
Transmitter Feeder Loss (dB)	**Lft**	0.5	0.5	0.5	0.5
Transmitter EIRP (dB)	**EIRP**	-22	-17	-12	-7
Slant Range (km)	**ds**	2000	2000	2000	2000
Free Space Loss (dB)	**Lfs**	151.2702287	151.2702287	151.2702287	151.2702287
Atmospheric Loss (dB)	**La**	0.5	0.5	0.5	0.5
Carrier Power Density (dBW)	**C'**	-173.770229	-168.770229	-163.770229	-158.770229
Receiver Noise Figure	**NF**	6	6	6	6
Receiver Environmental Temperature (K)	**T**	290	290	290	290
Equivalent Noise Temperature (K)	**Te**	1450	1450	1450	1450
Equivalent Noise Temperature (dBK)	**Te (dBK)**	31.61368002	31.61368002	31.61368002	31.61368002
Receiver Antenna Gain (dBi)	**Gr (dB)**	3.5	3.5	3.5	3.5
Receiver G/Te (dBK-1)	**G/Te**	-28.11368	-28.11368	-28.11368	-28.11368
Receiver Feeder Loss (dB)	**Lfr**	0.5	0.5	0.5	0.5
Boltzmann's Constant (JK-1)	**k**	1.38E-23	1.38E-23	1.38E-23	1.38E-23
Boltzmann's Constant (dBJK-1)	**10*log(k)**	-228.601209	-228.601209	-228.601209	-228.601209
Method of calculation using the manually calculated maximum bit rate					
Noise Density (dBW/Hz)	**N0**	-199.987529	-199.987529	-199.987529	-199.987529
bandwidth (Hz)	**BW**	125000	125000	125000	125000
FEC code rate	**CR**	0.8	0.8	0.8	0.8
spreading factor	**SF**	7	7	7	7
raw bit rate (bit/s)	**Rb**	6835.9375	6835.9375	6835.9375	6835.9375
effective bit rate (bit/s)	**Rb eff**	5468.75	5468.75	5468.75	5468.75
Bandwidth efficiency (bit/Hz)	**η**	0.0546875	0.0546875	0.0546875	0.0546875
Total Noise Power (dBW)	**N**	-149.018429	-149.018429	-149.018429	-149.018429
Carrier to Noise (dB)	**C/N**	-24.7517997	-19.7517997	-14.7517997	-9.75179967
Bit energy to carrier noise ratio	**Eb/N0**	-11.1615802	-6.16158024	-1.16158024	3.838419757
Method of calculation using the LoRa calulcator maximum bit rate					
Carrier to Noise Density (dB)	**C/No**	26.21730046	31.21730046	36.21730046	41.21730046
Maximum bit-rate (bit/s)	**fb**	5469	5469	5469	5469
Bit energy to carrier noise ratio	**Eb/No**	-11.1617788	-6.16177877	-1.16177877	3.838221227

Red are assumed values

Blue are values from the datasheet

Black are calculated

Green values are constants

TABLE 9 - LINK BUDGET CALCULATION AND RESULTS FOR THE RADIO (2) DEFAULT SETTINGS

Link Budget for LoRa RFM96 with (2) long range settings					
Parameter	**Symbol**	**Downlink**	**Downlink**	**Downlink**	**Downlink**
Transmitter Power (dBm)	**Pamp (dBm)**	5	10	15	20
Transmitter Power (dB)	**Pamp (dB)**	-25	-20	-15	-10
Frequency (GHz)	**f**	0.437	0.437	0.437	0.437
Transmitter Antenna Gain (dBi)	**Gt (dB)**	3.5	3.5	3.5	3.5
Transmitter Feeder Loss (dB)	**Lft**	0.5	0.5	0.5	0.5
Transmitter EIRP (dB)	**EIRP**	-22	-17	-12	-7
Slant Range (km)	**ds**	2000	2000	2000	2000
Free Space Loss (dB)	**Lfs**	151.270229	151.270229	151.270229	151.270229
Atmospheric Loss (dB)	**La**	0.5	0.5	0.5	0.5
Carrier Power Density (dBW)	**C'**	-173.770229	-168.770229	-163.770229	-158.770229
Receiver Noise Figure	**NF**	6	6	6	6
Receiver Environmental Temperature (K)	**T**	290	290	290	290
Equivalent Noise Temperature (K)	**Te**	1450	1450	1450	1450
Equivalent Noise Temperature (dBK)	**Te (dBK)**	31.61368	31.61368	31.61368	31.61368
Receiver Antenna Gain (dBi)	**Gr (dB)**	3.5	3.5	3.5	3.5
Receiver G/Te (dBK-1)	**G/Te**	-28.11368	-28.11368	-28.11368	-28.11368
Receiver Feeder Loss (dB)	**Lfr**	0.5	0.5	0.5	0.5
Boltzmann's Constant (JK-1)	**k**	1.38E-23	1.38E-23	1.38E-23	1.38E-23
Boltzmann's Constant (dBJK-1)	**10*log(k)**	-228.601209	-228.601209	-228.601209	-228.601209
Method of calculation using the manually calculated maximum bit rate					
Noise Density (dBW/Hz)	**N0**	-199.987529	-199.987529	-199.987529	-199.987529
bandwidth (Hz)	**BW**	31250	31250	31250	31250
Fec code rate	**CR**	0.5	0.5	0.5	0.5
spreading factor	**SF**	10	10	10	10
raw bit rate (bit/s)	**Rb**	305.175781	305.175781	305.175781	305.175781
effective bit rate (bit/s)	**Rb eff**	152.587891	152.587891	152.587891	152.587891
Bandwidth efficiency (bit/Hz)	**η**	0.00976563	0.00976563	0.00976563	0.00976563
Total Noise Power (dBW)	**N**	-155.039029	-155.039029	-155.039029	-155.039029
Carrier to Noise (dB)	**C/N**	-18.7311998	-13.7311998	-8.73119976	-3.73119976
Bit energy to carrier noise ratio	**Eb/N0**	4.38209977	9.38209977	14.3820998	19.3820998
Method of calculation using the LoRa calulcator maximum bit rate					
Carrier to Noise Density (dB)	**C/No**	26.2173005	31.2173005	36.2173005	41.2173005
Maximum bit-rate (bit/s)	**fb**	152.6	152.6	152.6	152.6
Bit energy to carrier noise ratio	**Eb/No**	4.38175512	9.38175512	14.3817551	19.3817551

TABLE 10 - LINK BUDGET CALCULATION AND RESULTS FOR THE RADIO (3) DEFAULT SETTINGS

Link Budget for LoRa RFM96 with (3) long range settings					
Parameter	**Symbol**	**Downlink**	**Downlink**	**Downlink**	**Downlink**
Transmitter Power (dBm)	**Pamp (dBm)**	5	10	15	20
Transmitter Power (dB)	**Pamp (dB)**	-25	-20	-15	-10
Frequency (GHz)	**f**	0.437	0.437	0.437	0.437
Transmitter Antenna Gain (dBi)	**Gt (dB)**	3.5	3.5	3.5	3.5
Transmitter Feeder Loss (dB)	**Lft**	0.5	0.5	0.5	0.5
Transmitter EIRP (dB)	**EIRP**	-22	-17	-12	-7
Slant Range (km)	**ds**	2000	2000	2000	2000
Free Space Loss (dB)	**Lfs**	151.2702287	151.2702287	151.2702287	151.2702287
Atmospheric Loss (dB)	**La**	0.5	0.5	0.5	0.5
Carrier Power Density (dBW)	**C'**	-173.770229	-168.770229	-163.770229	-158.770229
Receiver Noise Figure	**NF**	6	6	6	6
Receiver Environmental Temperature (K)	**T**	290	290	290	290
Equivalent Noise Temperature (K)	**Te**	1450	1450	1450	1450
Equivalent Noise Temperature (dBK)	**Te (dBK)**	31.61368002	31.61368002	31.61368002	31.61368002
Receiver Antenna Gain (dBi)	**Gr (dB)**	3.5	3.5	3.5	3.5
Receiver G/Te (dBK-1)	**G/Te**	-28.11368	-28.11368	-28.11368	-28.11368
Receiver Feeder Loss (dB)	**Lfr**	0.5	0.5	0.5	0.5
Boltzmann's Constant (JK-1)	**k**	1.38E-23	1.38E-23	1.38E-23	1.38E-23
Boltzmann's Constant (dBJK-1)	**10*log(k)**	-228.601209	-228.601209	-228.601209	-228.601209
Method of calculation using the manually calculated maximum bit rate					
Noise Density (dBW/Hz)	**N0**	-199.987529	-199.987529	-199.987529	-199.987529
bandwidth (Hz)	**BW**	1.25E+05	1.25E+05	1.25E+05	1.25E+05
Fec code rate	**CR**	0.5	0.5	0.5	0.5
spreading factor	**SF**	12	12	12	12
raw bit rate (bit/s)	**Rb**	366.2109375	366.2138672	366.2167969	366.2197266
effective bit rate (bit/s)	**Rb eff**	183.1054688	183.1069336	183.1083984	183.1098633
Bandwidth efficiency (bit/Hz)	**η**	0.002929688	0.002929688	0.002929688	0.002929688
Total Noise Power (dBW)	**N**	-149.018429	-149.018394	-149.018359	-149.018325
Carrier to Noise (dB)	**C/N**	-24.7517997	-19.7518344	-14.7518692	-9.7519039
Bit energy to carrier noise ratio	**Eb/N0**	3.590287307	8.590252563	13.59021782	18.59018308
Method of calculation using the LoRa calulcator maximum bit rate					
Carrier to Noise Density (dB)	**C/No**	26.21730046	31.21730046	36.21730046	41.21730046
Maximum bit-rate (bit/s)	**fb**	183.1	183.1	183.1	183.1
Bit energy to carrier noise ratio	**Eb/No**	3.590417018	8.590417018	13.59041702	18.59041702

TABLE 11 - SUMMARY OF THE LINK BUDGET FOR EACH RADIOHEAD SETTING AND TX POWER

Link Budget calculations for the satellite radio beacon communications link at 2000kms												
RadioHead default settings	**(0) Medium Range Settings**				**(2) Long Range Settings**				**(3) Long Range Settings**			
Parameter	**Downlink**				**Downlink**				**Downlink**			
Transmitter Power, Pamp (dBm)	5	10	15	20	5	10	15	20	5	10	15	20
Bit Enegy/Noise Density Ratio, Eb/No	-8.73	-3.73	1.27	6.27	4.38	9.38	14.38	20.38	3.59	8.59	13.59	18.59

The results from calculating the Link budget are presented above in Table 11 and show that the (0) RadioHead default setting does not achieve an Eb/N0 above 10 for the maximum TX power level. If the TX power is set to 15dBm then an Eb/N0 above 10 for the communications link can be achieved using the (2) and (3) RadioHead default settings. A received Eb/No greater than 10 is generally considered a reliable communications link, with

the (2) default settings providing a better Eb/N0 value per Transmit power. The (2) RadioHead was selected over the (3) settings for the satellite beacon radio as it:

1. Provides a higher received Eb/N0
2. It has a smaller bandwidth and uses a smaller portion of the RF spectrum
3. It has a smaller transmit time, less Time-on-Air
4. It has 20% higher energy per Bit

If a 15dBm transmit power is used with (2) default settings then a received Eb/N0 of 14.38 is achieved which provides a safety margin in the received signal allowing for unaccounted losses in the communication link (other atmospheric loss, doppler shift loss, component degradation, etc.)

The final test was to perform a ground-based check that will verify the results from the LoRa calculator and Link budget to prove that the radio can operate over the distance. The test was performed by passing a signal between 2 LoRa modules that are connected by a series of attenuators and cables. The total attenuation of the cables and attenuator represent the FSPL and can be used to calculate the distance the represented by the attenuation. The cables used for the testing is RG-174/U coaxial cable which has an attenuation of 0.62dB/m (obtained from multiple datasheets). The program used for the testing is the generic "Hello world" program obtained from the Adafruit website which utilises the Radiohead library for the RFM96 module (https://learn.adafruit.com/adafruit-feather-32u4-radio-with-lora-radio-module/using-the-rfm-9x-radio).

FIGURE 9 - COMMUNICATIONS LINK GROUND TESTING SETUP

The ground testing of the communications link was achieved by adjusting the values for the A and B attenuator such that the total attenuation increases slowly. The "hello world" data that was received by the LoRa radio was monitored to observe when the data started to be inconsistently received. The last value for which the data was consistently being received is observed to be the maximum FSPL path loss for the reliable transmission of data. This value was then used to calculate the maximum distance for reliable transmitted data used in the following equation:

$$FSPL\ (dB) = 20\log_{10}(d) + 20\log_{10}(f) + 20\log_{10}\left(\frac{4\pi}{c}\right)$$

Where,

$$d = distance\ (m)$$
$$f = transmited\ frequency\ (Hz)$$
$$c = speed\ of\ light\ (3 * 10^8 m/s)$$

TABLE 12 - TOTAL MEASURE FSPL AND CALCULATED DISTANCE FOR THE COMMUNICATION LINK GROUND BASED TESTING

Largest measured FSPL attenuation and distance for consistent data reception									
Transmit power (dBm)		5		10		15		20	
RadioHead default setting	Reciever sensitivity (dB)	Total attenuation (dB)	Distance (kms)	Total attenuation (dB)	Distance (kms)	Total attenuation (dB)	Distance (kms)	Total attenuation (dB)	Distance (kms)
(0)	-124	131.14	195	137.14	390	141.14	620	145.14	980
(2)	-139.2	146.14	1100	151.14	1950	155.14	3100	161.14	6200
(3)	-138	144.14	880	150.14	1750	155.14	3100	160.14	5500

The radio settings for the LoRa module are based on the (2) RadioHead default settings for long range communications and slow data rate. These radio settings with a TX power of 15dBm combine to allow a transmit range of 3300kms while maintaining an Eb/N0 over 10. If the range of the communications link is 2000kms then an Eb/N0 of 14.38 is achieved for the received signal. The results from the LoRa calculator

estimates are congruent with the Link Budget calculation and the results of the ground testing correspond very closely with the link budget calculations. The LoRa UHF communications link from a small satellite in LEO to a ground receiving station has been verified to transmit data for 2000km through calculations and ground testing with the operation only being validated by on-orbit operation.

The connection of the LoRa radio module with the APM processor module (shown previously in Figure 5) results more than 50% of the total APM connection remaining for interconnection with the other satellite systems. The APM connection remaining are a combination of:

- Up to 8 analogue connections
- Up to 12 digital connections
- 1 UART Serial connection
- 1 Two wire Interface (TWI)
- An FTDI header
- 4 PWM channels
- 1 external interrupt and 2 analogue comparators

The software program developed in Arduino for the APM module to drive the LoRa radio module use:

- 8208 Bytes (28%) of the flash memory for the program, with 22512 Bytes remaining
- 1004 Bytes (50%) of the static RAM for the global variables, with 2044 Bytes remaining
- 1 Byte (0%) of EEPROM for variables, with 1023 Bytes remaining

The software program for using the LoRa modules uses less than 50% of the APM module memory, with the rest being able to be used for the development of the functions that collect telemetry data and perform operations based a command received from the ground station. The way the receive function is performed will require an investigation into some cyber-security protocols to ensure a secure environment such that this cannot be exploited by a third party.

ELECTRICAL POWER REGULATION, GENERATION AND STORAGE

POWER REGULATION SYSTEM

The testing of the APM module revealed that the onboard LDO voltage regulator (MIC5205) can consume a large quiescent current when the input voltage (5V) was much larger than the output voltage (3.3V). The Silicon solar panel (next section) to be used in the ground testing is expected to have an output voltage of between 5-6V which determine that an investigation into alternatives voltage regulators be undertaken. The investigation will be carried out by testing the current consumption of different regulators for the electrical power system. The testing will be carried out by using a 5V, 1.5A wall power supply being applied to a variety of regulators and running the satellite beacon through a shortened software cycle. The APM module contains a surface mount LED which can consume up to 10mA (more information in the solar panel section) which has been removed for this testing. The regulators that will be utilized for testing are:

1. MIC5205 – APM module in-built Low-Noise LDO voltage regulator
2. LM1086 – LDO voltage positive regulator
3. LM3671 – Step-Down DC-DC converter (Buck converter)
4. TS2904CZ – Ultra LDO linear voltage regulator

The LoRa radio will use a TX power setting of 15dBm and will use the (2) RadioHead default settings [slow & long-range settings – BW = 31.25kHz, CR= 4/8, SF = 9 (512chips/symbol) and CRC on]. A 1Ω and 2Ω tester resistor will be placed in series with the power supply and regulator to measure the supply current with the testing configuration of the satellite beacon power regulator testing shown below in Figure 10.

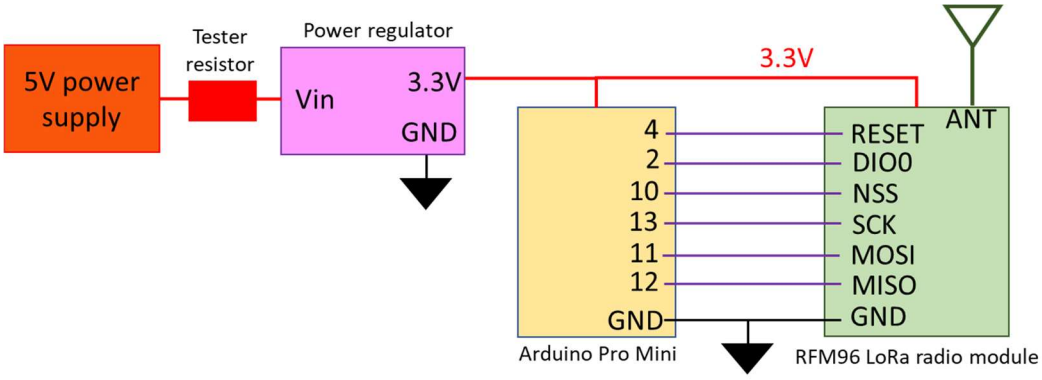


FIGURE 10 - SATELLITE BEACON POWER REGULATOR TESTING CONFIGURATION

TABLE 13 - BEACON TOTAL CURRENT CONSUMPTION RESULTS UTILISING DIFFERENT VOLTAGE REGULATORS

Satellite beacon current consumption using different regulators									
MC5205 in-built regulator - 1 Ohm resistor									
Cycle number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Avg**
Launch (mA)	1.51								**1.51**
Collect Data (mA)	8.48	8.15	8.15	7.82	7.82	8.15	8.07	8.14	**8.10**
Transmit data (mA)	116.70	116.00	116.00	116.00	116.00	115.70	116.00	116.00	**116.05**
Receive mode (mA)	17.20	16.87	16.86	16.86	16.86	16.86	16.87	16.86	**16.91**
Idle mode (mA)	2.13	2.13	1.79	2.13	2.13	2.14	1.80	2.13	**2.05**
MC5205 in-built regulator - 2 Ohm resistor									
Cycle number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Avg**
Launch (mA)	0..4								**0.00**
Collect Data (mA)	7.43	7.26	7.26	7.26	7.26	7.43	7.26	7.44	**7.33**
Transmit data (mA)	115.40	115.40	115.40	115.40	115.40	115.40	115.40	115.40	**115.40**
Receive mode (mA)	16.32	16.31	16.32	16.32	16.48	16.32	16.32	16.32	**16.34**
Idle mode (mA)	1.73	1.57	1.57	1.40	1.74	1.57	1.57	1.57	**1.59**
LM1086 external regulator - 1 Ohm resistor									
Cycle number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Avg**
Launch (mA)	6.48								**6.48**
Collect Data (mA)	11.85	12.17	11.84	12.18	12.18	12.18	12.18	12.18	**12.10**
Transmit data (mA)	126.70	126.70	126.70	126.70	126.70	126.70	126.70	126.40	**126.66**
Receive mode (mA)	20.89	20.88	20.89	20.89	20.89	20.89	20.88	20.89	**20.89**
Idle mode (mA)	6.81	7.15	7.15	7.15	6.82	6.82	7.15	7.15	**7.02**
LM1086 external regulator - 2 Ohm resistor									
Cycle number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Avg**
Launch (mA)	5.75								**5.75**
Collect Data (mA)	11.46	11.29	11.29	11.46	11.46	11.61	11.46	11.29	**11.42**
Transmit data (mA)	124.90	124.90	124.80	124.80	124.80	124.90	124.90	124.80	**124.85**
Receive mode (mA)	20.34	20.34	20.17	20.34	20.34	20.34	20.18	20.17	**20.28**
Idle mode (mA)	6.43	6.26	6.25	6.26	6.43	6.26	6.43	6.26	**6.32**
LM3671 external regulator - 1 Ohm resistor									
Cycle number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Avg**
Launch (mA)	1.51								**1.51**
Collect Data (mA)	5.47	5.45	5.48	5.14	5.46	5.13	5.12	5.15	**5.30**
Transmit data (mA)	88.83	88.83	88.49	88.83	88.83	88.83	88.83	88.82	**88.79**
Receive mode (mA)	11.83	11.83	11.83	11.83	11.84	11.83	11.83	11.84	**11.83**
Idle mode (mA)	1.48	1.49	1.14	1.13	1.15	1.15	1.48	1.15	**1.27**
LM3671 external regulator - 2 Ohm resistor									
Cycle number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Avg**
Launch (mA)	0.75								**0.75**
Collect Data (mA)	4.91	4.92	4.91	4.91	5.08	4.91	4.75	4.91	**4.91**
Transmit data (mA)	88.73	88.73	88.73	88.73	88.73	88.73	88.56	88.73	**88.71**
Receive mode (mA)	11.46	11.46	11.46	11.62	11.63	11.46	11.46	11.46	**11.50**
Idle mode (mA)	1.08	1.08	1.08	1.41	1.08	0.91	1.08	1.08	**1.10**
TS2940CZ external regulator - 1 Ohm resistor									
Cycle number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Avg**
Launch (mA)	3.47								**3.47**
Collect Data (mA)	9.16	9.17	9.17	9.17	9.50	9.49	9.16	9.16	**9.25**
Transmit data (mA)	127.40	127.40	127.00	127.40	127.40	127.40	127.40	127.40	**127.35**
Receive mode (mA)	17.87	18.21	18.20	18.21	18.21	18.20	18.21	18.21	**18.17**
Idle mode (mA)	3.81	3.80	5.08	4.14	4.13	4.14	3.81	3.80	**4.09**
TS2940CZ external regulator - 2 Ohm resistor									
Cycle number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Avg**
Launch (mA)	2.58								**2.58**
Collect Data (mA)	7.42	7.43	7.26	7.26	7.43	7.43	7.26	7.29	**7.35**
Transmit data (mA)	125.10	125.30	125.10	124.90	125.10	125.10	125.30	125.30	**125.15**
Receive mode (mA)	16.31	18.32	16.48	16.32	16.32	16.32	16.32	16.32	**16.59**
Idle mode (mA)	3.24	3.25	3.24	3.25	3.42	3.41	3.24	3.25	**3.29**

The results for the regulator testing is detailed above in Table 13 and reveal that the LM1086 and TS2940CZ LDO regulators have a much higher current consumption than the MC5205 LDO regulator and LM3671 Step-Down DC-DC converter (buck converter) during all phases of the software cycle. The inbuilt regulator and buck converter have similar quiescent currents throughout the software cycle except for during the *transmit* phase in which the LM3671 buck converter consumes 27mA less current.

The LM3671 DC-DC buck converter has been selected to be used in the Satellite radio beacon as the expected solar panel voltage is nearly twice the regulated output. This will minimise the current consumed by the power regulation system, but further research will be required to ensure that the components used in the buck converter are space ready.

SOLAR POWER GENERATION SYSTEM

The largest constraint on the solar power generation system will be the amount of space available on the outside of the satellite for the solar panels and the amount of power required by the computer and radio modules. The major types of solar panels available are Gallium-Arsenide (GaAs) or a silicon-based (Si) with the GaAs being more efficient (producing higher power values for surface area) but it also comes with a higher purchase cost. There have been improvements in the construction of monocrystalline structure Silicon solar panels that have resulted in more robust and efficient Silicon panels that would be worth investigation (Check on the research at UNINSW Kensington). A full investigation into a space ready solar power generation system is yet to be carried out but will be required to be carried out prior to commencing mission readiness testing.

The initial ground testing for the solar power generation system was carried out using a 0.5W monocrystalline Silicon solar panel from Seeed studio (https://www.seeedstudio.com/0-5W-Solar-Panel-55x70.html). The solar panels are 55x70mm and perform at a conversion rate of up to 17% producing approx. 5V at 100mA in full sun per panel. To determine the current drawn from the solar panels by the complete satellite radio beacon system (external LM3671 regulator, computer processor, LoRa radio module and attached components) during each phase of the software cycle, a tester resistor was placed between the solar panels and the breadboard positive power rail which supplies the power for all the other sub-systems components. The transmit power of the RFM96 LoRa radio module was set to 15dBm with the radio using the (2) RadioHead default settings (long range settings). The testing program used was a shortened version of the software cycle where the launch lasts for 5 seconds, the receive mode is 3 seconds and the idle/low power mode is 2 seconds.

The results from testing the current consumption of the satellite beacon provided evidence that the radio beacon was unable to be operated with one solar panel connected if the transmission power was greater than 10dBm as the current required for continuous operation was larger than the current being supplies by one solar panel. This prompted an investigation to find a solution where the transmit power can be increased while operating the beacon using one solar panel. This led to including five 2.2mF electrolytic capacitors in parallel with the solar panels which stores approximately 0.17 joules of energy that could be used during the transmit phase to stabilize the power system and provide enough energy for the transit current spike. The configuration of the satellite radio beacon used for the total power requirement and generation testing is presented below in Figure 11.

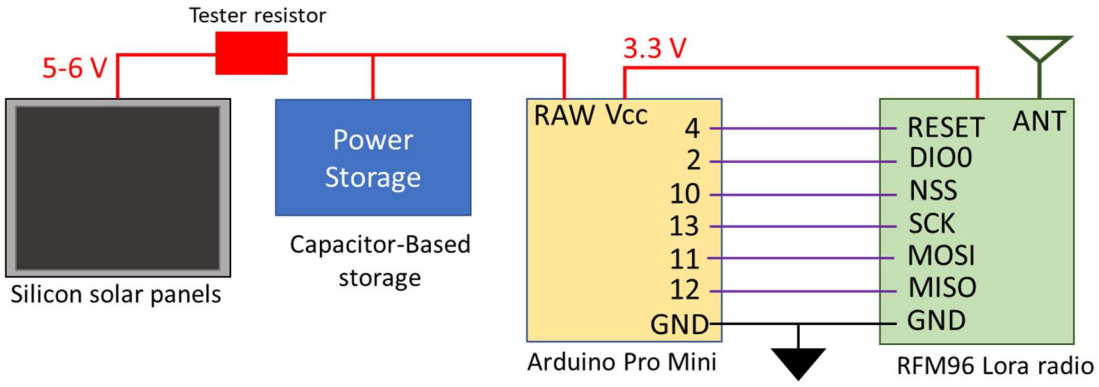


FIGURE 11 - SATELLITE RADIO BEACON CONFIGURATION FOR POWER REQUIREMENT AND GENERATION TESTING

TABLE 14 - SATELLITE BEACON TOTAL CURRENT CONSUMPTION MEASUREMENTS DURING EACH SOFTWARE CYCLE PHASE

Satellite beacon total current measurements								
1Ω tester resistor								
Cycle number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Avg**
Initialisation (mA)	**55.58**							**55.58**
Launch (mA)	1.50							**1.50**
Collect Data (mA)	5.44	5.44	5.14	5.46	5.46	5.46	5.46	**5.41**
Transmit data (mA)	88.83	88.88	88.70	88.56	88.56	88.83	88.88	**88.75**
Receive mode (mA)	12.17	12.17	11.83	11.83	11.84	11.84	11.87	**11.94**
low-power mode (mA)	1.48	1.48	1.48	1.48	1.48	1.48	1.48	**1.48**
2Ω tester resistor								
Cycle number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Avg**
Initialisation (mA)	**55.4**							**55.4**
Launch (mA)	0.75							**0.75**
Collect Data (mA)	4.91	4.90	4.91	4.90	4.92	4.92	4.75	**4.89**
Transmit data (mA)	88.56	88.73	88.73	88.56	88.73	88.56	88.73	**88.66**
Receive mode (mA)	11.46	11.29	11.46	11.62	`11.46	11.46	11.46	**9.82**
low-power mode (mA)	1.08	1.08	1.08	1.08	1.24	1.25	1.00	**1.12**

The testing of including 11mF of electrolytic capacitance to store energy for the transmit phase was conducted by running the beacon through the shortened software cycle with one solar panel connected while increasing the transmit power from 5dBm to 23dBm in 1dBm increments. The shortened beacon software cycle with the same settings as the previous test was used to test the addition of supporting capacitors in the power system. At each power level, the software cycle was performed 10 times with operation of the beacon being sustained for all transmit power levels when connected to a single solar panel and five 2.2mF capacitors. The weather conditions for the day were clear and sunny with the orientation of the solar panels perpendicular to the sun. It is noted that solar panels, in general, can generate approximately 20% more power in a space environment as the sunlight does not have to penetrate the Earth's atmosphere. This will result in the beacon system having more power available when deployed in LEO with the excess providing a safety margin in the power generation system in lower irradiance conditions.

The ground testing carried out on the solar generation system has shown that the operation of the beacons processor and radio (using max TX power of 23dBm) can be supported by one ground-based silicon solar panel if a supplementary energy source is provided. The supplementary energy source used for the testing was 11mF of capacitance that stores electrical energy that can be utilised during the high current *transmit* phase of the radio beacon software cycle. The next section will investigate an energy storage system based upon super-capacitors (also known as ultracapacitors) which can be used to replace the 11mF of electrolytic capacitance.

Further research is required into selecting a solar panel that can withstand the LEO space environment as well as providing enough energy for the whole duration of a small satellite mission until it deorbits. An energy storage system is required to be investigated to ensure that there is enough energy to support the beacon operation in low irradiance conditions.

ELECTRICAL POWER STORAGE SYSTEM

The final investigation for the satellite radio beacon was to research an electrical power storage system that will support operation of the beacon in low irradiance conditions when the generation of electrical power is reduced and can provide additional energy during the *transmit* phase of the beacon. There were several options explored for the storage of electrical power (Lithium-Ion batteries, hydrogens fuels cells, nuclear power, etc) with these solutions all requiring expansive additional testing from the launch providers to ensure isolation and multiple fail-safes in the system. An electrical storage system based on super-capacitors will alleviate the need for this testing as it can be launched with no stored electrical power with energy only been stored once the solar panels are deployed from the satellite and generating electricity.

The last set of tests for the satellite beacon begins by determining if the inclusion of a super capacitor in parallel with the solar panel can sustain the transmit phase of the software cycle when using one solar panel. Secondly, a measurement of the voltage level and charge time of the super-capacitor storage system, consisting of five 1F capacitors, when connected to no load and 1 or 2 solar panels. The capacitor storage system is then connected to the Satellite radio beacon (with the solar panels are disconnected) to determine the length of time that software cycle can be run using only the energy stored in the capacitors. The full satellite beacon software cycle will be used without the 30-minute launch cycle, a transmit power of 15dBm and the (2) RadioHead default radio settings. The final step for the super-capacitor storage test will have it connected, with no electrical energy in the capacitors, to one solar panel and the satellite radio beacon which has the same radio settings as the previous test but will operate the full satellite radio beacon software cycle. The electrical potential of the capacitors will be monitored to measure the charging characteristics after a simulated launch and through the first cycles of the software program as well as to measure the time it takes for the system to contain enough energy to initialise post launch. The final configuration of the satellite radio beacon for ground testing is shown below in Figure 12.

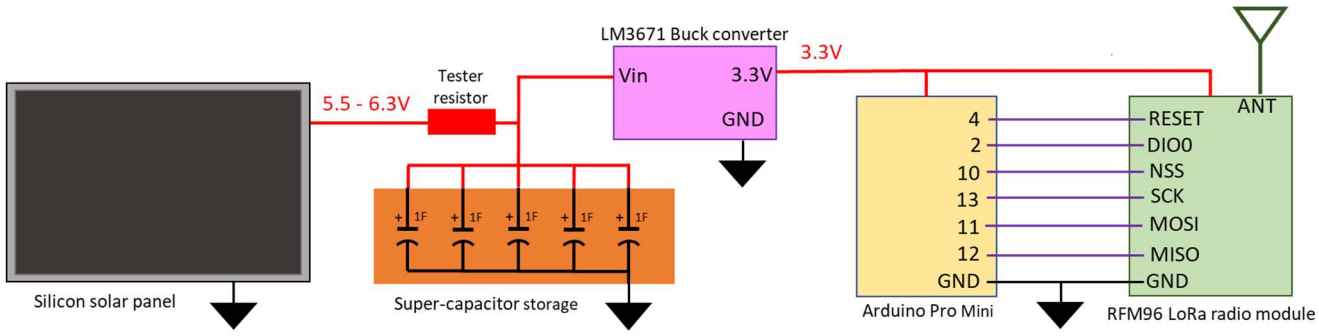


FIGURE 12 - SATELLITE RADIO BEACON FINAL CONFIGURATION FOR GROUND TESTING

When a singular 5.5V, 1F super-capacitor replaced the 11mF electrolytic capacitor as the energy storage medium, then the electrical storage system was able to support the radio transmit phase. The inclusion of the super-capacitor in the power system causes a delay for the beacon software initialisation to begin once power has begun being generated. When one solar panel is used to charge a single super-capacitor then it takes 1 minute for power to be applied to the system, and when the number of super-capacitors is increased to 5 then the time increases to 7 minutes when there is no load on the capacitors.

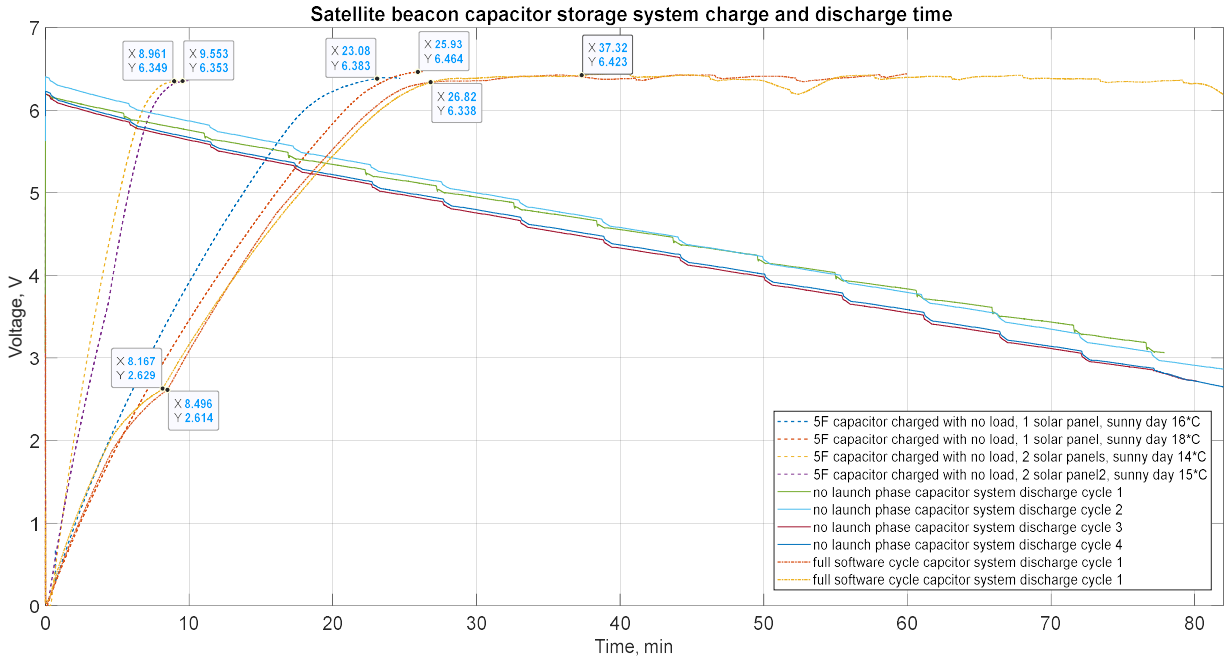


FIGURE 13 - SATELLITE BEACON POWER STORAGE CHARGE AND DISCHARGE VOLTAGE WITH 5F SUPER-CAPACITOR SYSTEM

The no-load charging time testing of the super-capacitors using one solar panel is detailed above in Figure 13, with the results showing that the average time taken for five 1F super-capacitors to charge to full capacity when no load is connected is 24 minutes and 30 seconds using one solar panel. If two solar panel are connected, then the charging time decreases to 9 minutes and 20 second.

The average voltage potential of the super-capacitor storage system after a charging cycle was 6.3V which equals 101 Joules of energy stored in the five capacitors. The average operating time before the super-capacitor could not support the beacon operation is 1 hour, 13 minutes and 43 seconds for one charge of the super-capacitor storage system. Discontinuities in the power supply would cause the satellite beacon software to reset when the voltage potential of the super-capacitor storage system reduced below 3V which typically occurred during the transmit phase of the software cycle. The results show that there is an 8 minute and 20 seconds delay from when the solar panel first start to generate electrical power until there is enough energy to initialize the radio beacon hardware and software in which the voltage potential of the capacitors measures 2.6V. The super-capacitor storage system reaches it full electrical potential after 27 minutes of operation when the software cycle is approximately 19 minutes into the low power launch phase.

The ground testing carried out on a super-capacitor energy storage system has confirmed that it can be used as a supplementary energy source to support the satellite beacon operations when a single Silicon solar panel is used for power generation. The super-capacitors used for ground testing contains a paste electrolyte which is not suitable for space operation, but there has been research and testing carried out on other types of super-capacitors capable of operating in space. The prohibitive cost of obtaining these prevented further investigation into these solutions will be needed before mission readiness tests can be carries out.

SATELITE RADIO BEACON SYSTEM SUMMARY

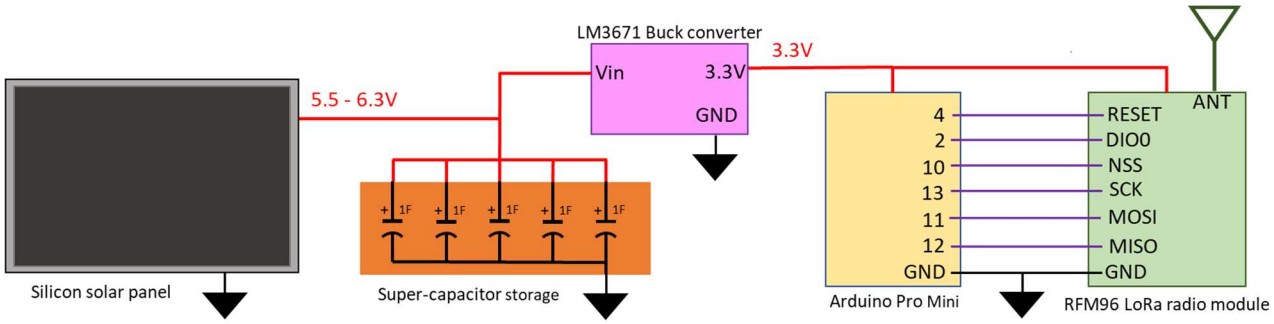


FIGURE 14 - INITIAL PROTOTYPE SATELLITE RADIO BEACON DESIGN

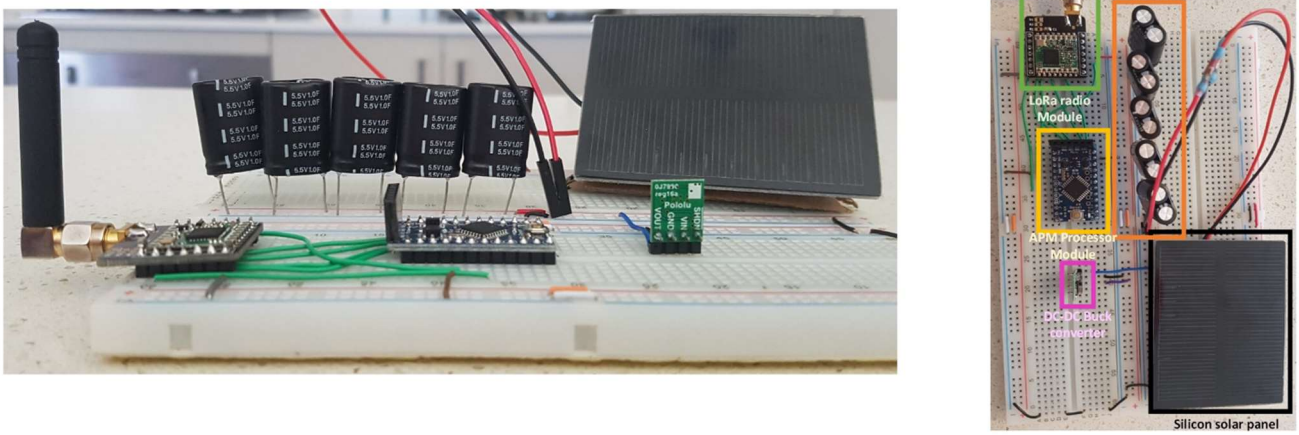


FIGURE 15 - BREADBOARD CONSTRUCTION OF THE SATELLITE BEACON PROTOTYPE DESIGN

The initial prototype design for the satellite radio beacon is shown above in Figure 14 with the prototype model that was constructed for the ground testing of the beacon being presented in Figure 15. The results of the ground-based testing of the satellite beacon has shown that the system operation can be self-sustained with one solar panel and the super-capacitor storage system. The energy storage system retains enough energy for approximately 74 minutes of beacon operation. The processor module ensures that the beacon operates independently of the other satellite systems and can be interconnected to collect telemetry data or to pass a received control command. The LoRa radio module used with the RadioHead (2) default settings with a TX power of 15dBm has been shown to successfully transmit data for a range of 2000km which is the maximum expected slant range for a satellite in LEO.

The next step in the development process is to convert the initial design from a prototyping breadboard to a Printed Circuit Board (PCB). The components that are used within each system with the beacon should be reviewed and converted to space ready components (if able). This will ground testing of the system to commence to verify its readiness for space missions such as tumbling tests on solar generation and radio transmissions, solar panel angle of incidence checks, high and low temperature component checks, etc.

GROUND RECEIVER STATION SUBSYSTEM

THEORY OF OPERATION

The ground receiving station will have 2 purposes, the first is to receive the satellite identification and telemetry data and the second is to provide a precise time of arrival for tracking purposes.

The satellite identification data is the satellite address represented as 4 hexadecimal characters that uniquely identify the satellite that the RF signal originates from. This requires that a database of all addresses that have been assigned to a unique satellite be kept and maintained to ensure the integrity of the system. There are a possible 65,536 addresses using the 4 hexadecimal characters, which would allow for a minimum of 22 years of operation (using the nano-microsatellite forecast figures from Spaceworks) without a clash in addresses. The first byte in each Identification radio packet sent by the satellite beacon contains a unique numeric identifier which shows which packet has been received (the first packet starts with 1, the second with 2 and so on). The telemetry data that is collected by the ground receiving station will be available to be read by any operator, but the contents of the radio packet will not be meaningful with knowing the structure of the data in the radio packet. This ensures that the operator of each individual satellite has control over the protection of that data, so if they would like to share the structure and meaning of the data in the radio packet then they can choose to do so. It will be encouraged to provide the details of the telemetry data to allow the amateur operator of the ground receiving station to collect the data and send to a centralised storage and processing facility to allow inclusiveness amongst the amateur, commercial, academic and governmental organisations in the LEO space operations and monitoring.

The precise Time of Arrival (TOA) and satellite identification will be recorded by each ground receiving station for each signal sent by a satellite beacon to allow trilateration of the signals using Time Difference of Arrival (TDOA). Using the TOA of a beacon RF signal being received at three ground receiving stations will allow calculations to be carried out that will identify the estimated position of the satellite. The information required to perform the calculation will be the Latitude, longitude and altitude of each ground receiving station (this can be obtained using a GPS receiver) and the time of arrival of the signal at each ground station (ideally, down to nano seconds precision where 1ns represents a distance of 0.3m and 100ns represents a distance of 30m). The data that is sent from the beacon is broken up into 5 separate radio packets (4 identification packets and 1 telemetry packets) with each packet allowing a separate time stamp to be provided for time of arrival calculations (averaging will help with precision). A PPS signal can be provided by a GPS receiver which is synchronised to the Universal Coordinated Time (UTC) second function. The PPS signal and UTC data obtained from the GNSS satellites allows each globally dispersed ground receiving station to be synchronised with the same timing source with a tolerance of 30ns (9m).

The collected identification & telemetry data, the ground stations lat/long/alt position, the recorded GNSS UTC and the radio packet and PPS signal time stamps are sent from the ground receiving station to a peripheral device through its serial connection. The peripheral device (computer or laptop) with then collect this data from multiple ground stations to estimate the position of the satellite using a TDOA trilateration technique.

INITIAL CONFIGURATION

The initial configuration of the ground RX station is on the next page in Figure 16 and was chosen as the components are readily available from commercial suppliers and eBay, are affordable to purchase and can easily be constructed and programmed by inexperienced users.

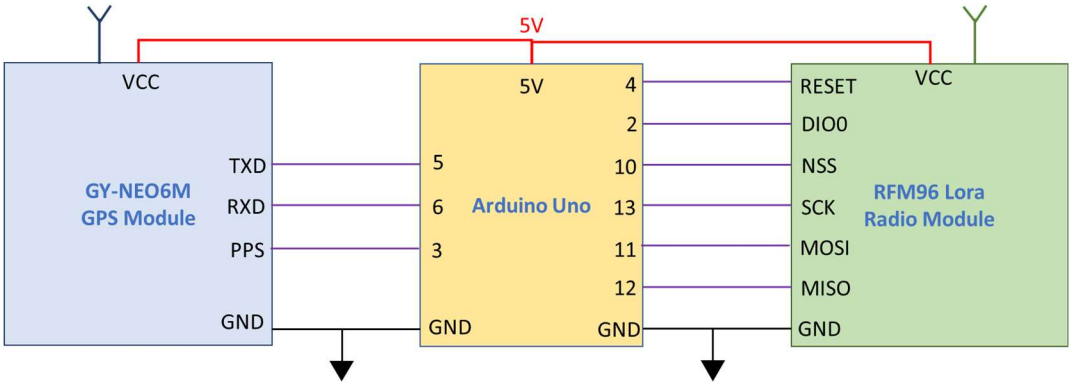


FIGURE 16 - BLOCK DIAGRAM OF THE INITIAL CONFIGURATION OF THE GROUND RECEIVING STATION

The RFM96 Lora radio is mounted onto a breakout PCB which allows for direct integration with a standard breadboard (2.54mm hole-to-hole spacing), which can be made by any PCB manufacturer (PCBWay, PCBs.io) using files from GITHUB (https://github.com/attexx/rfm9x_breakout_board). The antenna used for the initial testing was a simple SMA 433MHz omni-directional antenna with a 3.5dBi gain (https://www.ebay.com.au/itm/1PC-433Mhz-3-5dbi-wireless-module-antenna-OMNI-radio-aerial-SMA-male-connector/184268947966?hash=item2ae74901fe:g:q8oAAOSwx79epdGA) but there are other antennas available that will provide a higher gain to obtain a better Eb/N0 value for the communications link such as this 12dBi antenna. (https://www.ebay.com.au/c/19036536885#oid153679223262)

The GPS unit used on the breadboard is the U-Blox NEO7-MV GNSS receiver module which was purchased from Ebay for a cost of approx. $15 (https://www.ebay.com.au/itm/Positioning-Module-Development-Board-NEO-7M-Replace-NEO-6M-Smart-Phone-PC-GPS/324119426785?ssPageName=STRK%3AMEBIDX%3AIT&_trksid=p2060353.m2749.l2649). This model was selected as it contained a built-in passive antenna that has a gain of 28dBi which is sufficient to capture a usable GPS signal if the ground receiving station is located outside. This GPS unit has a provision to solder an alternative SMA plug onto the board which allows for a second antenna to be fixed to the GPS chip. This allows a better antenna (https://www.ebay.com.au/itm/CB58-3-M-Car-Auto-GPS-SMA-Connector-Cable-Antenna-Signal-Booster-For-Car/273759947089?epid=9030380923&hash=item3fbd5d4951:g:0LgAAOSwupBcibrJ&frcectupt=true) which places the antenna outside with less interference and allows the ground station to be utilised indoors. This GNSS module also contains a breakout pin for obtaining the GPS PPS signal which is a low-to-high pulse that repeats every second with a tolerance of 30nS. This signal, in conjunction with the UTC provided by the GPS signal will be used to synchronise the timing between the individual ground receiving stations.

The Arduino unit selected for the initial testing and development of the software was the Arduino Uno as it has a higher processing power and more pins for use compared to the Arduino Pro Mini. The Arduino Uno contains a ceramic oscillator (CSTCE16MOV53-RO) which has a large tolerance for frequency stability and tolerance. This component has a drift of 0.8% which can equate to a drift of up 128,000 clock cycles over one second which will need to be considered for time measurement in the ground receiving station.

Other alternatives that can be used for the processing are the Arduino Mega2560 (utilises a crystal oscillator with smaller tolerances), the Arduino Due (crystal oscillator) or a Teensy based board. The Arduino Uno model was selected as it has more resources available to help create the software program for the ground receiving station. If it is found that the Arduino Uno is not a suitable board for this application then the Arduino Due should be considered next as is still an affordable option but it has much more processing power, memory, available connections and oscillator frequency (84MHz).

SOFTWARE PROGRAM DEVELOPMENT

The initial development of the ground receiving software cycle was used to get a baseline program that allows for reception of the LoRa signal, recording a LoRa signal time stamp, obtaining time stamp of several GNSS PPS signal, obtaining GNSS data (ground station lat/long/alt and the UTC value) and displaying these on the serial monitor.

The program ran such that when the Arduino Uno is powered then the LoRa and GNSS modules are initialised, and the system waits for a valid signal from the LoRa unit. When a valid signal is received by the LoRa radio (contains valid header and packet CRC checks, etc.), then a time stamp is taken and saved with the data from the radio packet. When the fifth packet is received (5^{th} time stamp taken, and telemetry data saved) then the ground receiver waits for the PPS signal from the GPS unit which triggers an interrupt service routine (ISR).

During this ISR a time stamp is immediately taken which allows for the time between the PPS signal (which is representative of the UTC time with a one second resolution as received in the GNSS data packet) and the time stamp of each of the 5 packets of data to be calculated. There will be a delay in time between receiving the PPS signal, creating an interrupt and measuring the current time stamp, but this delay will be the same for each ground station and will not produce an error in the Time Difference of Arrival. The time stamp for the next three PPS signals are saved and averaged to determine the number of clock cycles or time between each pulse to determine the instantaneous (or true) oscillator frequency of that ground station at the time the measurements were made.

All the collected data is then sent through the Arduino Uno serial connection to a peripheral device for post processing where the satellites positions is determined using the TDOA calculation technique. A laptop using MATLAB or C/Python script can be utilised to collect and process the sent data or a C or python program could be written that does the same function. The initial design of the software program in the Arduino IDE is based upon the software flow chart that is presented on the next page in Figure 17.

Consideration will need to be given to creating a server based system which can collect data from globally dispersed ground stations, process and calculation the satellite position and then push this data back out to the computer connected to the servers. This will also require a GUI to be created for the user to display the position of all satellites being tracked.

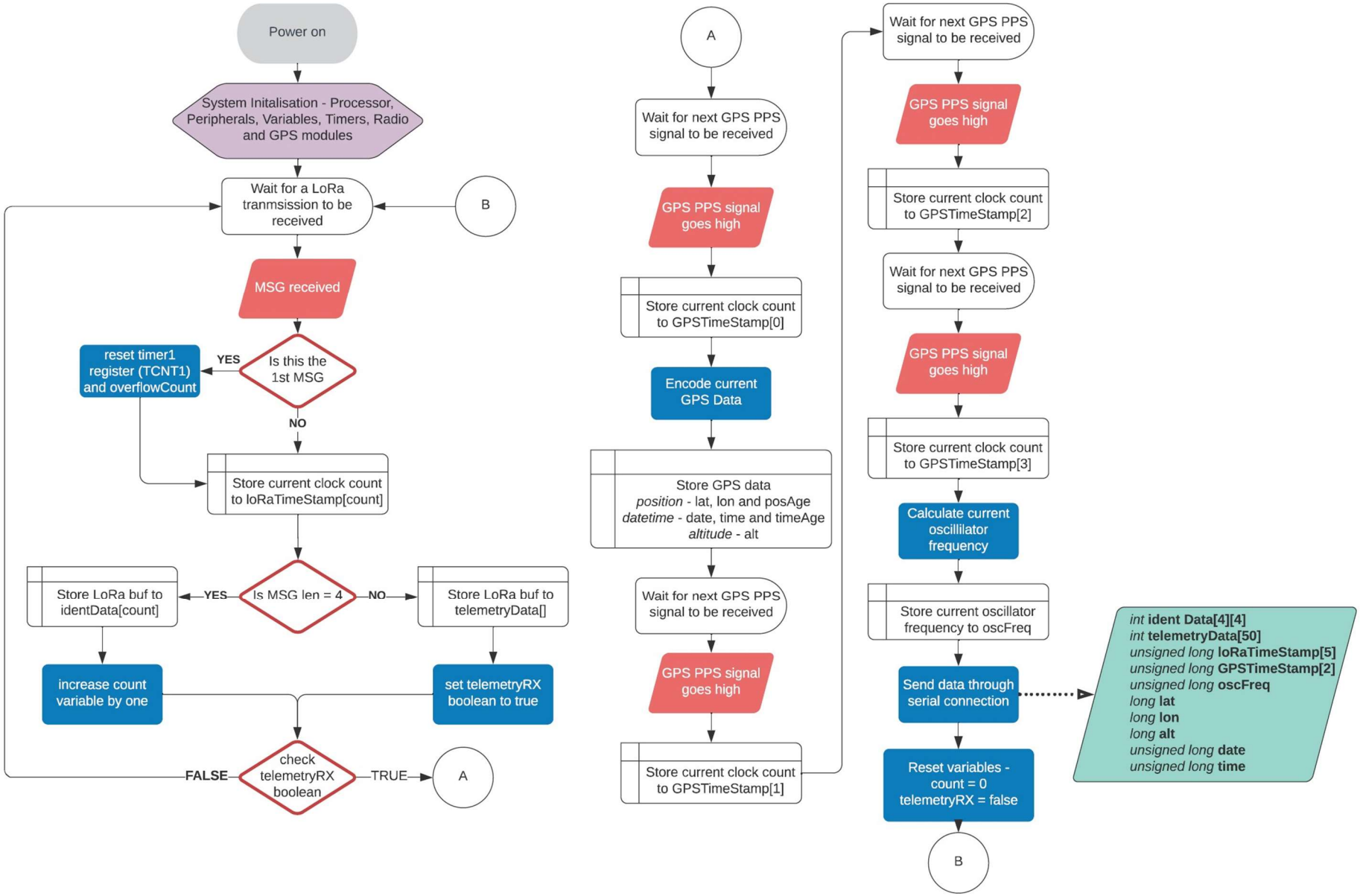


FIGURE 17 - GROUND RECEIVING STATION SOFTWARE FLOW DIAGRAM

Initial timing testing

The first test carried out was to see if the PPS signal from the GPS unit is a valid signal that can be used to signpost breaks in time every second for the Arduino Uno. Each time the signal transitions from low to high, an interrupt is carried out by the microcontroller where a time stamp is captured using the *micros()* function in Arduino. When the next PPS signal arrives then a 2nd time stamp is captured in which the difference represents the time between PPS signals as determined by the microcontroller clock (obtained from measuring the frequency of the oscillator). The results show that the there was a time difference of -252 or -256 microseconds every time the PPS signal was received which shows that as room temperature (approx. 25*C) the time drifted by 252-256µs which equates to a drift of approx. 3906 clock cycles. This clock drift is comparable to a crystal oscillator and if the drift is measured directly after the time stamps are taken then the drift caused by local conditions for that ground station can be compensated for. This testing also showed that the maximum resolution that can be obtained using the *micros()* function is 4mS which equates to a distance tolerance of 1.2km which initially may be too large for this application. This requires an alternative method to measuring time using the Arduino Uno (i.e. directly reading the clock registers, etc) to increase the resolution of the timing and to decrease the uncertainty in measuring the time at the ground station. This prompted an investigation to identify all the possible source of uncertainty in the measuring of time and distance that may be present in the ground receiving station.

SOURCES OF UNCERTAINITY IN MEASURING THE TIME/DISTANCE

The identified possible sources of uncertainty in measuring time or distance are:

1. Resolution of the inbuilt Arduino *micros()* function timing – 4uS
2. The number of clock cycles required to execute an Interrupt Service Routine (ISR)
3. Drift in external oscillator frequency due to tolerances, temperature, and other sources of error.
4. Tolerance in the GNSS module PPS signal (typically 30nS)
5. Accuracy of GPS latitude/longitude/altitude measurement (typically within 10ms but is dependent on GPS signal strength and number of GPS satellites obtained)
6. The time taken for the LoRa module software to process the received RF signal and make it available to the processor
7. Resolution of the oscillator – 16MHz equates to a clock cycle every 62.5nS (or 18.75ms distance travelled by an EM wave).

Timing Resolution of *micros()* function (4µS)

When the in-built Arduino *micros()* function is used to capture the time, the measurement is made in 4µS increments which equates to a distance measuring error of 1.2kms which may be too large for this application if this uncertainty compounds. If the Timer1 clock of the ATMEGA328P can be accessed, then theoretically the resolution of timing measurement could be reduced to 62.5nS which has a distance error of 18.75ms.

The Timer1 register in the ATEMEGA328P is a 16-Bit counter which has a resolution of 62.5nS for the values held in the TCNT1 counter (number of clock pulses) which can hold 65536 values if the pre-scaler is set to 1 or 0x01 in the TCCR1B register. The Timer1 can be set-up such that once the TCNT1 counter overflows (reaches 65535 clock pulses and starts at 0) then an ISR can be setup to count the number of overflows and the TCNT1 resets back to 0 to begin again. This method uses the compare function of the Timer1 (*TIMER1_COMPA*_vect) where the compare value is set to 65535 (*OCR1A = 65535;*) such that the ISR will be called when the Timer1 count value is equal to 65535. To set this up then the CTC mode must be turned on in the TCCR1B register (*TCCR1B |= (1<< WGM12);*) and the interrupt must be turned on (*TIMSK1 |= (1 << OCIE1A);*). When the ISR is

called a variable that keeps track of the number of overflows is updated while Timer1 resets to 0 and continues counting.

If the number of clock counts that have occurred between events is required to be determined then the value for TCNT1 is read and added to the number of overflows that have occurred multiplied by 65536. This will give the total amount of clock cycles that have occurred and if multiple by the length of the clock cycle (approx. 62.5ns), will give the time between events with a 62.5ns resolution.

****Important Note – When the Arduino Uno is initiated using the Arduino IDE, then the init() function for the microcontroller is carried out in the background where all the pins are initially set-up for PWM operation. This causes the Timer1 to revert to an 8-Bit timer where the low register (TCNT1L) holds the 8-Bit value, but the high register (TCNT1H) is turned off. Before using the Timer1 it must be reset to revert it back to a 16-Bit timer which can be done by resetting the TCCR1A and TCCR1B registers to 0 or 0x00 (*TCCR1A=0;* and *TCCR1B-=0;*)****

****Important Note – If using the Radiohead library to drive the LoRa radio, then the RH_ASK.c and RH_ASK.hpp utilise the Timer1 counter. The Arduino compiler does not work with this conflict and as these files are not required in this application then they should be removed from the radio library****

To use the time measurement function in this application, we can reset the Timer1 register (TCNT1) when the first LoRa signal arrives, then we can mark the time between the first signal against the next 4 LoRa signals. After the timestamps of all 5 Lora signals are captured then we can wait for the next PPS signal from the GPS to mark the final time stamp and get the current GPS Universal time Coordinated (UTC) time stamp which has a 1 second resolution. We can then subtract each LoRa timestamp from the GPS UTC time stamp to get the signals time of arrival with a 62.5nS resolution with a reference to a common clock for all ground stations.

To test the variances in the clock counting technique, an Arduino Uno was used to measure the amount of clock cycles between PPS signals from the NEO-7M GNSS module on the Uno external interrupt pin 3, in which there should be no or very little difference between one PPS pulse interval to the next. Over time there may be an overall drift due to factors such as a difference in temperature effecting the oscillator, etc but this will have little effect between 2 PPS pulse intervals. To measure if there is a variance in the number of clock cycles between PPS signal then the number of clocks cycles between each PPS pulse will be measured over a large period to allow a large sample of data. The difference in the number of cycles between two PPS signals is determined to find the immediate average differences in clock cycles. This will allow an uncertainty in counting clock cycles for time measurements to be determined and used in the distance calculation.

FIGURE 18 - DIFFERENCE IN CLOCK CYCLE MEASUREMENTS BETWEEN SUCCESSIVE GNSS PPS PULSES

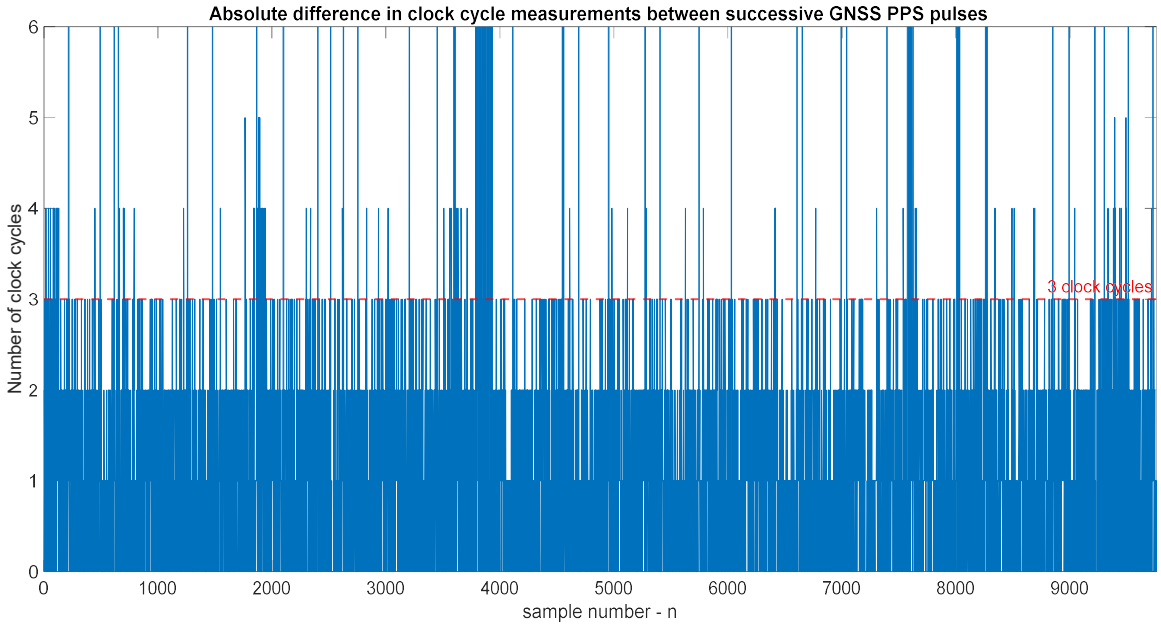


FIGURE 19 - HIGHER RESOLUTION ABSOLUTE DIFFERENCE IN CLOCK CYCLES AT THE 3-CYCLE THRESHOLD

The graph displayed above in Figure 18 shows that most of the measurements are below 3 clock cycles which is amplified in Figure 19. The results show that when 9757 samples are tested for the clock cycle differences then 260 of the samples are above the 3-cycle threshold, which equates to an 2.7% error rate. If the threshold is increased to 4 clock cycles, then the error rate reduces to 1.8% and if the threshold is decreased to 2 clock-cycles then the error rate increases to 9.2%. The resultant statistics, including the spikes, has mean of 2.2404 and a standard deviation of 10.6291 which indicates that 95% of the absolute measurements are between 0 and 23.4986 clock cycles.

The cause of the spikes in the measurements is a known phenomenon (discussed in the next paragraph) and if removed from the statistics calculation then the absolute statistics has a mean of 0.9134 and standard deviation of 0.7213 which has 95% of absolute measurements between 0 and 2.356 clock cycles. If the statistics were determined on the measured values, then the mean is 0.0266 with a standard deviation of 1.1636 which results in 95% of the measurements being between -2.3006 and 2.3538 clock cycles. If the uncertainty is taken to be ±3 clock cycles, then this equals a time measurement uncertainty of 187.5ns or distance uncertainty of 56.25m every time the clock counting technique is carried out.

The spikes present in Figure 18 (previous page) show an increase in the error of clock cycles between 1-second uniform events by up to 180 clock cycles and can be contributed to the implementation of the clock counting ISR in the software. The Arduino IDE is based upon C programming in which the processor will complete the execution of the current list of commands before an ISR is carried out. The number of clock cycles it takes to execute the current list of commands is variable which is shown in the differing values of the spikes. The software program developed for testing the number of clock cycles between the GNSS PPS signal uses two interrupt routines, an external interrupt and a clock compare interrupt, with the external interrupt occurring on the PPS signal and the clock compare used to count how many time the clock overflows. The ISR hierarchy dictates that the external interrupt occurs first before carrying out the clock compare ISR causing a delay in incrementing the clock overflow and significantly changing the number of clock cycles being counted between PPS signals. This large spike in uncertainty due to the delay in entering the ISR may occur multiple times in the ground receiving station software due to the increased complexity of the software cycle. These errors could compound to produce an error that is larger than the ±3 clock cycles expected and may need to change the method in which the clock counting technique is implemented.

Number of clock cycles to execute an ISR

The clock counting technique, the method to timestamp the LoRa signal and GNSS PPS signals are events that require an interrupt to initiate the ISR. There are several clock cycles required between when the event occurs and when the ISR begins which will introduce another uncertainty into the time measurement at the ground station. The number of clock cycles that occur for each different type of interrupt to initiate a ISR is a fixed amount (http://www.gammon.com.au/forum/?id=11488) and will be exactly the same for each ground receiving station. There will be a fixed time uncertainty for using the interrupts, but the interrupts are all performed in the same way between ground stations so there will be no uncertainty difference between ground stations. As there will be no uncertainty difference then this will not affect the TDOA calculation technique and this uncertainty can be ignored. The only variable uncertainty will be if the ground station is performing an operation as the interrupts occurs and does not carry out the ISR until the current set of commands are executed as discussed in the previous section.

Oscillator Drift

The Arduino Uno contains a 16MHz ceramic resonator (CSTCE16M0V53-R0) which has a frequency tolerance of 0.5% and a frequency stability of 0.3%. This results in the oscillator having a frequency drift of up to 128kHz due to temperature, aging or tolerances in manufacture and the drift will not be the same for each oscillator in each ground station. This means that the true, instantaneous frequency of the oscillator at the time the measurement is taken must be known to determine the exact length of time for each clock cycle.

The GNSS PPS signal occurs every one second with a tolerance of 30nS which can then be used to measure the number of clock cycles in a second. The number of Timer1 clock cycles that occur between PPS pulses will determine the instantaneous frequency of the oscillator and the time of each clock cycle. To help reduce uncertainty in determining the oscillator frequency, then the number of clock cycles per PPS pulse should be done over multiple PPS pulse cycles and averaged. This will be implemented in the ground station after all the

time stamps of the LoRa radio and GNSS PPS are taken. When the GNSS PPS time stamp is taken then the clock cycles are counted for the next 3 GNSS PPS pulses, summed together and then divided by 3 to determine the instantaneous frequency of the oscillator. This value is then used to calculate the exact time of one clock pulse which is then used to determine the amount of time that has elapsed between each LoRa timestamp with reference to the first GNSS PPS timestamp (the one used to get the current UTC value).

The clock counting method has a uncertainty of 3 clock cycles which is multiple by 4 (number of PPS pulses used) and divided by three (the averaging factor) which equals a total uncertainty of 4 clock cycles when determining the instantaneous oscillator frequency. For a 16MHz oscillator frequency, this represents an uncertainty of 2fs for calculating the length of a clock cycle which is negligible and can be ignored in the ground stations total uncertainty of time measurement.

PPS Signal Tolerance

The GPS receiving module used in the ground station utilises the U-Blox NEO-7M GNSS module which has a PPS time tolerance of ±30nS that equates to a distance calculation uncertainty of 9m. This tolerance is within an acceptable limit and no further investigation will be carried out

Accuracy of the GPS position

The data sheet for the U-Blox NEO-7M states that the accuracy of the GNSS positioning is 2.5m, my own tests utilising U-Center software to monitor the position shows that the positioning tolerance is larger than 2.5m but never greater than 10m. The error in position is largest during the initialisation of the GNSS module and reduces over time (generally a minute or two) to the expected 2.5m. This tolerance in determining the GNSS position of the ground receive station is acceptable and no further tests will be carried out on the GNSS module.

The biggest determinant of the tolerance in position is the antenna that is used and the weather of the day. The external antenna used in the ground station can lock onto more GPS satellites than the built-in antenna which results in less positional uncertainty. Using the built-in antenna or a cheap patch antenna could produce an uncertainty of up to 30m as less GPS satellites are able to be locked onto the ground station. If there is a lot of bad weather present in the local environment such as clouds or rains, then less GPS satellites can be picked up resulting in a greater positional tolerance. The only way to counter this error due to weather is to use the best antenna available for the GPS module with the one utilised has shown acceptable performance for all but the worst weather when the antenna is placed outside in an unobstructed area.

RFM96 module processing time (preamble and message)

The RFM96 LoRa module contains its own software in which the received data is error checked and corrected if required when an RF signal is received. There are a series of checks carried out on the LoRa radio packet preamble and payload data that is described on page 35-37 of the RFM96 documentation. The amount of time that is taken to carry out the checks, perform any error corrections and then make the data available to the processor is unknown and adds additional uncertainty to the measurement of time by the ground station.

The general process for the LoRa module receiving an RF signal is that the preamble of the radio packet will be checked for integrity first and if the preamble is correct then the payload data integrity will then be assessed. The integrity of the payload is checked (checksum errors, payload length, Cyclic Redundancy Checks (CRC) checks, etc.) and if any error correction is required then this is carried out. If the received payload data is verified, then the *RXDone* interrupt is set on the RFM96 module (the DIO0 pin) to signal that the data is ready to be passed to the processor. The DIO0 pin on the LoRa module is connected to pin 2 on the Arduino Uno module which uses an external ISR to process the received data from the SPI connection to the LoRa unit.

An investigation is required to determine the amount of uncertainty in the time difference between two ground station to process the LoRa RF signal and make the data available to the processor. To achieve this, two ground stations are connected directly to the Satellite radio beacon to ensure that the transmitted LoRa signal will arrive at the exact same time to the LoRa radio receiver (see below in Figure 20). The DIO0 pin on each LoRa module on the ground stations will be connected to an oscilloscope which will capture when the signal goes from low to high. This will show when the LoRa module processing has been completed and the *RXDone* interrupt has been set. The LoRa module that is connected to channel one of the oscilloscopes with be the reference signal with the time measured between the second channel represent the difference in time taken to process the signal between two modules. This test will be repeated multiple times to analyse the resultant statistics and determine the uncertainty in LoRa processing time.

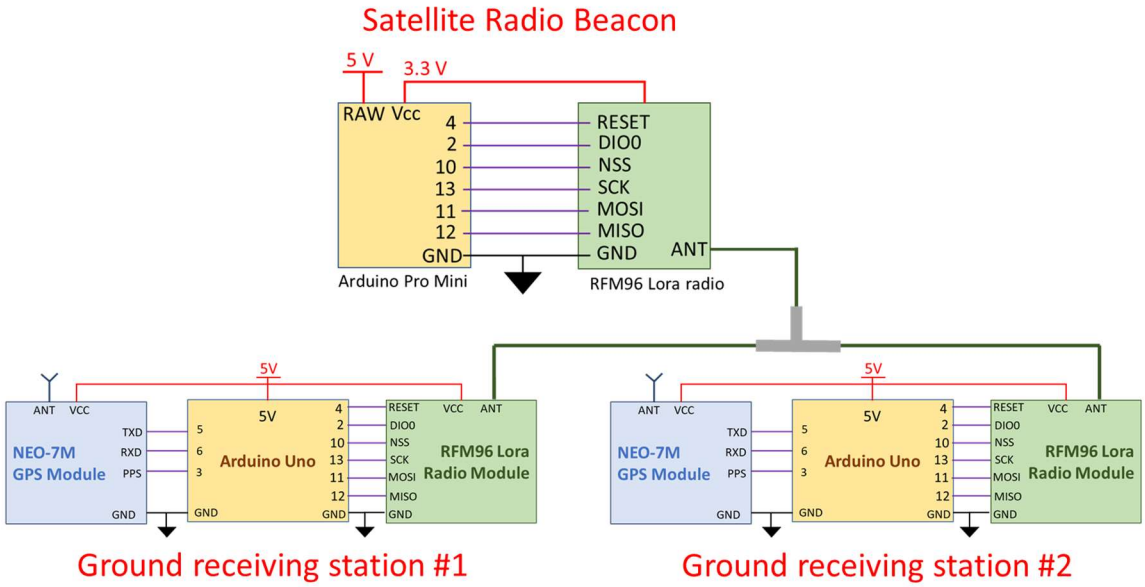


FIGURE 20 - UNCERTAINTY IN LORA PROCESSING TIME TEST SET-UP

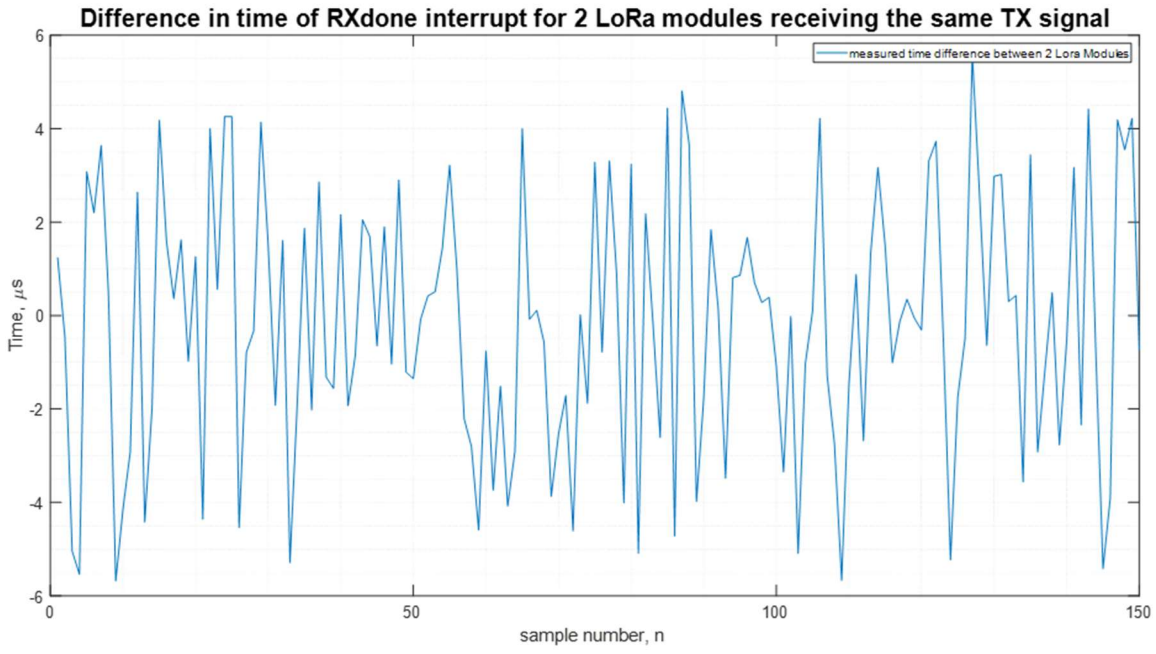


FIGURE 21 - TIME DIFFERENCES FOR THE RXDONE INTERRUPT BETWEEN 2 LORA MODULES FROM A SINGLE TX SOURCE

The results in Figure 21 (previous page) show the values for the time difference measured between the *RXdone* interrupts for 2 ground stations when receiving a LoRa signal from a single transmit source with the channel 1 of the oscilloscope being the reference. The resultant statistics taken from the absolute values of the collected data show that the difference in time between the LoRa modules has a mean of 2.2901µs, a standard deviation of 1.6042µs and a maximum difference of 5.68µs with 95% of the data being between 0 and 5.4985µs. When the statistical calculations are performed on the measured data the resultant mean is -0.1803µs with a standard deviation of 2.7966µs which results in 95% of the measured data being between -5.7735µs and 5.4129µs. Using the Cumulative Distribution Function of the measured data has found that 92.56% of the data lies between ±5µs, 96.56% of the data lies between ±5.5µs and 96.77% lies between ±6µs. Using an uncertainty of 5µs for the difference in LoRa processing time between 2 stations results in an uncertainty of 1.5kms when calculating the position.

Oscillator Resolution

The Arduino Uno contains an 16MHz oscillator which results in a clock cycle occurring every 62.5nS which is the maximum resolution that could be achieved when measuring time. This results a resolution of 18.75ms when using the measured time to calculate the distance from the ground station to the satellite. The only way to improve this will be to either install a oscillator with a larger frequency, which would be a non-preferred option as it would require specialist equipment to perform this operation and would cause the ground station to be unable to be made by an amateur operator. A more preferred option would be to investigate other existing Arduino boards that contain a higher oscillator frequency with the following boards being research and found to be acceptable to be using in this application…

- Arduino Due ($80) – 84MHz = 12nS clock cycles or 3.6ms distance resolution
- Arduino M0 (PRO) ($50) – 48MHz = 20.8nS or 6.22ms *may be difficult to get one
- Arduino Zero ($45) – 48MHz = 20.8nS or 6.22ms
- Arduino MKRZero ($60) – 48MHz = 20.8nS or 6.22ms *will need to check if enough pins)

This will only be pursued if the total uncertainty of the time measurement is required to be reduced. As the error in the RFM96 processing time is much larger than any other error, then this is not worth pursuing as the timers may not be available to be utilised for the clock counting technique.

TOTAL UNCERTAINITY IN MEASURING THE TIME/DISTANCE

The investigation and testing of the identified source of uncertainty of measuring time or distance resulted in the following individual uncertainties:

1. 4µs Resolution of the Arduino timing function, *micros()* – led to using the clock counting technique
2. Error in *TIMER1* processor clock counting technique, ±3 clock cycles ~187.5ns (equates to 56.5m)
3. Number of clock cycles taken for an Interrupt Service Routine (ISR), no error time difference
4. External oscillator tolerances and drift, < 2fs (negligible)
5. GNSS position tolerance, 10m
6. GNSS PPS signal tolerance, 30ns (9m)
7. Difference in LoRa software receive processing time, 5µs (1.5km)

These values indicate that the total estimated uncertainty in the measured time difference of arrival between two ground stations is 5.25µs. This equates to an estimated uncertainty of 1.58km when calcualting the satellite position using the TDOA technique.

Final verification testing

The testing method used to verify the total uncertainty in measurements utilises the same test setup used to verify the difference in LoRa processing time (shown previously in Figure 20) where the satellite radio beacon is connected directly to two ground receiving stations via a coaxial cable and T-Piece. This direct connection ensures that the transmitted RF signal will arrive at each ground stations simultaneously in which no time difference of arrival should be observed in the measured timing data. The collected timestamps from the ground receiving station will not be the same due the uncertainties identified during the previous testing with the measured difference being the total uncertainty of the time measurement between two ground stations.

A large sample of timestamps from the radio packet transmissions were collected by the ground receiving station and sent through a serial peripheral connection to be processed by MATLAB. The format of the data that is sent from the Arduino to MATLAB in order is: four address timestamps, one telemetry time stamp, four GNSS PPS timestamps, the instantaneous oscillator frequency as calculated in Arduino, the UTC date and UTC time of arrival of the first GNSS PPS timestamp. The number of clock cycle between the first GNSS timestamp to the address and telemetry time stamps are multiple by the inverse instantaneous oscillator frequency to determine the amount of time that has passed since the PPS signal where the GNSS data was collected. The amount of time between each timestamp for a single set of transmissions are compared against the results of the other ground receiver station to find the difference which represents the total uncertainty in time measurement. The difference in the timestamps for each packet will be indicative of the total sum of all errors in measuring the difference time of arrival. The only uncertainty that is not included in this final total error is the positional tolerance of the GNSS which is much, much smaller than all other errors.

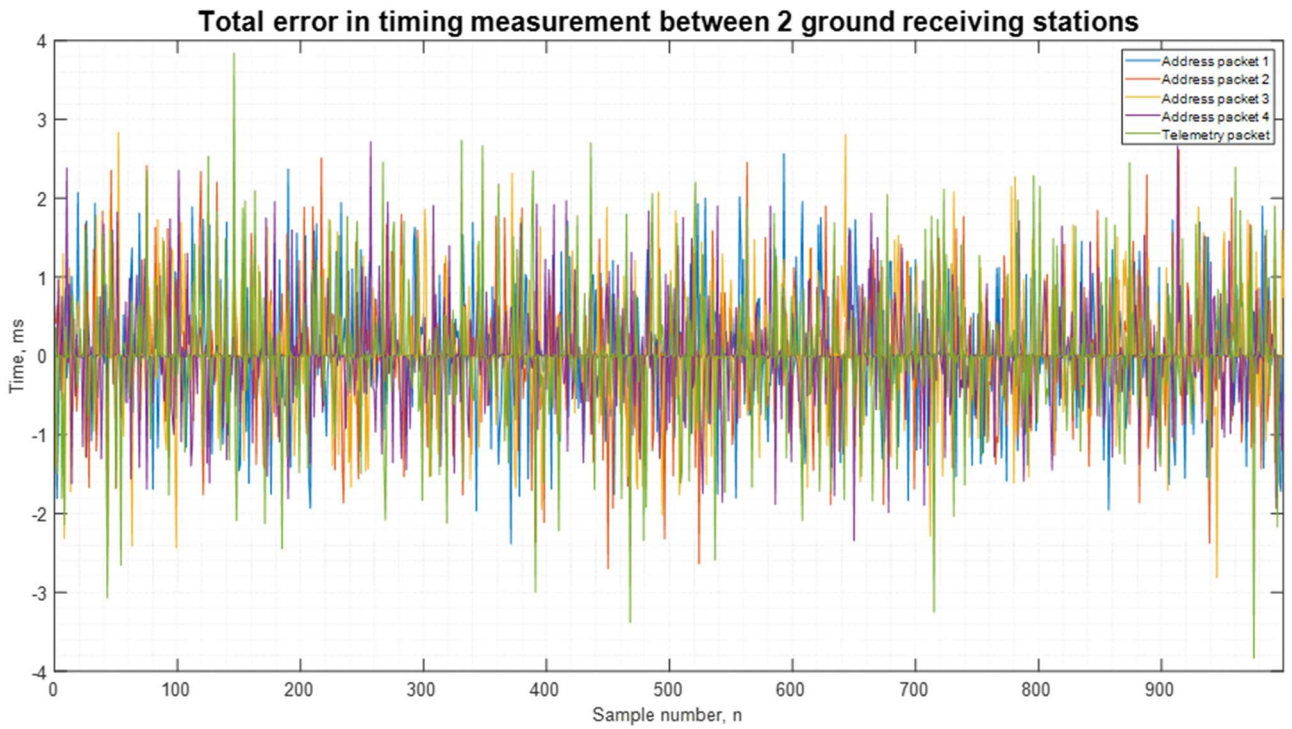


FIGURE 22 - TOTAL ERROR IN TIMING MEASUREMENT BETWEEN TWO GROUND RECEIVING STATIONS

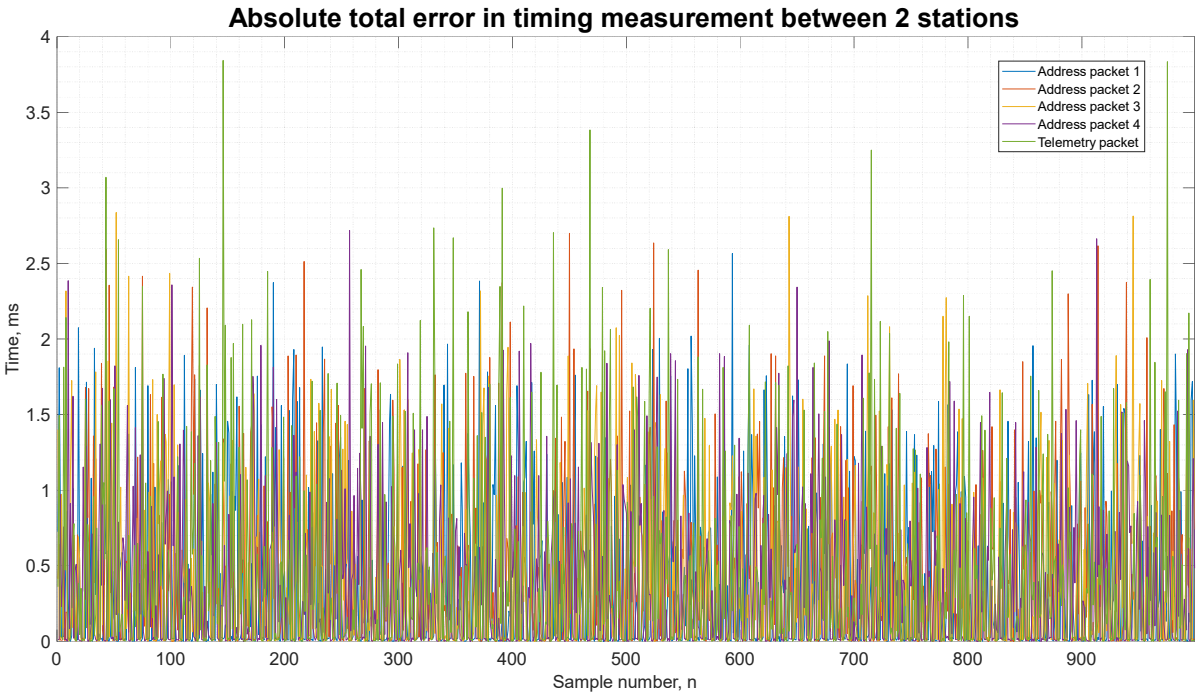


FIGURE 23 - ABSOLUTE TOTAL ERROR IN TIMING MEASUREMENT BETWEEN TWO GROUND RECEIVING STATIONS

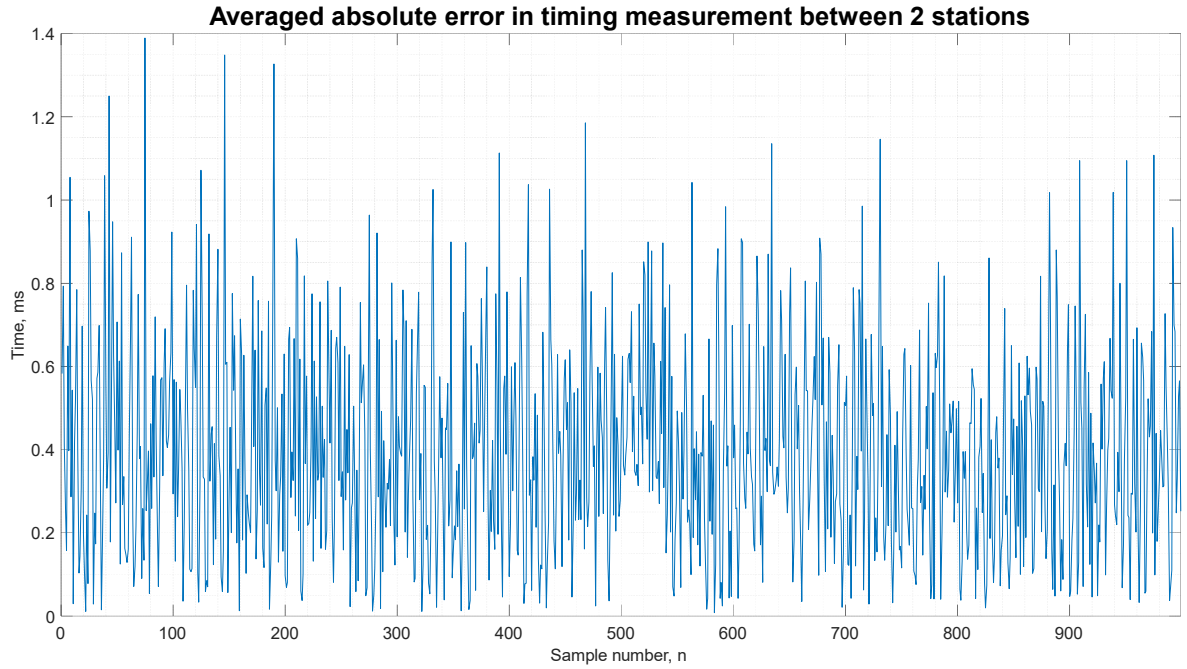


FIGURE 24 - AVERAGED ABSOLUTE TOTAL ERROR IN TIMING MEASUREMENT BETWEEN TWO GROUND RECEIVING STATIONS

TABLE 15 - TOTAL UNCERTAINTY IN TIME MEASUREMENT STATISTICS

	Measured values				Absolute values			
	mean (µs)	Std Dev (µs)	95% of data interval (ms)		mean (µs)	Std Dev (µs)	95% of data interval (ms)	
Address packet 1	19.816	638.83	-1.257844	1.297476	381.48	512.66	0	1.4068
Address packet 2	17.482	681.87	-1.346258	1.381222	409.35	545.46	0	1.50027
Address packet 3	38.799	652.44	-1.266081	1.343679	389.8	524.49	0	1.43878
Address packet 4	51.243	624.07	-1.196897	1.299383	379.93	498.02	0	1.37597
Telemetry packet	24.433	759.46	-1.494487	1.543353	419.14	633.66	0	1.68646
				Averaged values	395.14	250.2	0	0.89554

The results of the final verification testing are summarized above in Table 15 and show that the total absolute uncertainty in timing measurement has a mean of 380µs with a standard deviation of 520µs for the address packets and a mean of 419µs with a standard deviation of 634µs for the telemetry packet. This equates to 95% of the absolute measurement uncertainty of the address packets being less than 1.5ms and 95% of the telemetry uncertainty being less than 1.7ms which equate to a distance calculation error of 450km and 510km, respectively. If the absolute uncertainty in timing measurement is averaged between all five packets, then the mean is 395µs with a standard deviation of 150.2µs and 95% of the measured values below 0.9ms which is a distance calculation error of 270kms. This level of uncertainty in the measurement of the RF signals time of arrival is much larger than the expected level from the testing and is too large to develop a reliable tracking function from TDOA calculations.

The initial investigation for the large uncertainty in timing measurement tested the two external interrupts (the GNSS PPS signal and LoRa module *RXDone* interrupt) with the PPS signal found to be within tolerance while the LoRa module was found to have a processing time difference of up to 30µs (most were between 5-15µs) which is 25µs greater that then the values found during previous testing. This increase of the LoRa processing time is equal to a distance calculation error of 9km which is much less than the error found in the final verification testing.

The large spike in clock counting error found during testing of the clock cycle counting algorithm was never greater than 190 cycles which is equal to on uncertainty of 12µs or a distance calculation error of 3.5km which is much less than the final verification testing error but gives an indication to the cause of the error. The increased complexity of the ground receiver software and the two external interrupts used in the program taking precedence over the clock counting ISR causes a large increase in the number of clock cycles occurring before an ISR is entered. There seems to be large periods in the software cycle where the clock counting ISR breaks down. The mostly likely cause being when it must wait for the current series of codes being executed or if the external ISR are called (which takes priority) while waiting for the current code to be executed. This prevents the *TIMER1_COMPA_vect* ISR from updating the *overflowCount* variable and can cause up to hundreds and thousands of clock cycles being missed. The external ISR is called 9 times (5 for the LoRa radio and 4 for the PPS signals) during the ground station software cycle which can compound the error. The *TIMER1_COMPA_vect* ISR is called every 4.096µs which provides 244140 chances every second to be delayed by an external ISR or waiting for code to be executed.

To determine the reason for the large increase in error in counting clock cycles (1.5ms is equal to 24,000 clock cycles), an investigation into the method of implementing and calculating the clock cycles between events for the ground receiver station will be required with a focus on using timer capture modes for the ISR. The investigation should start by researching the method in which the *micros()* function is implemented by the Arduino IDE, as this method can operate successfully in the background without any effect on the executed program and without losing track of the count. This investigation to reduce the final error in timing measurement will be required to be carried out before the data acquired by the ground receiving station can be reliably used in calculating the estimated position of the satellite using TDOA.

SYSTEM SOFTWARE

SHORTENED BEACON SOFTWARE CYCLE FOR GROUND TESTING

```
/*
 * Beacon_Transmitter - Ground Station testing - Shortened Cycles
 *
 * This program is designed to carry out the Shortened software cycle for the Satellite Beacon for Ground
 * testing purposes. The software cycle comprises of five phases...
 *
 * 1) Initialisation/Launch - At launch the radio and microcontroller intitialises once power is applied and the both are
 * put to sleep for 5 seconds (for testing purposes), with the shutdown after launch time being 30-45 minutes for
 * actual space operations
 * 2) Data collection - The microcontroller does a series of commands such as check voltage, poll light sensing diodes etc.
 * to simulate the collection of satellite telemetry data.
 * 3) Data Transmission - 5 packets of data are transmitted. The first four packets contain the satellite address (packet number,23,AF,23,AF)
 * and the fifth packet containing 50 Bytes of random data representing the satellite telemetry data.
 * 4) Receive mode - The satellite radio enters receive mode for 5 seconds where it could accept a command
 * 5) Low power idle mode - The satellite enters a low power mode for 2 seconds where the radio is turned off and the
 * microcontroller is set to idle
 *
 * Author:  Travis McKee
 * Date:    09 Jun 2020
 *
 */

#include <SPI.h>
#include <RH_RF95.h>
#include <LowPower.h>
#include <EEPROM.h>

// for use with the Arduino Pro mini on the red PCB or Breadboard (Space Beacon) and the Arduino Uno (ground receiving station)
#define RFM95_CS 10
#define RFM95_RST 4
#define RFM95_INT 2 //This is the interrupt for the TXDone/RXDone signal from the LoRa module
#define RX_TIME 5000 //This sets how long the receive phase is for the beacon cycle

// Set the Transmit/Receive frequency
#define RF95_FREQ 437.0

// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);

int satBatVoltage = 0;
int solarPanelVoltage = 0;
int photoDiode1 = 0;
int photoDiode2 = 0;
int photoDiode3 = 0;

char radiopacket[50];
uint8_t satTelemetry[50];
uint8_t satAddress[5] = {0,35,175,35,175};
boolean launched = false;

void setup()
{
  Serial.begin(115200);

  Serial.println("Initialisation Begins");
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);

  // manual reset
  digitalWrite(RFM95_RST, LOW);
  delay(10);
```

```
 digitalWrite(RFM95_RST, HIGH);
 delay(10);

 EEPROM.write(0,0); //This uses the EEPROM to store if the satellite has just been launched (0) or if it is in orbit (1)

 while (!rf95.init()) {
  Serial.println("LoRa radio init failed");
  Serial.println("Uncomment '#define SERIAL_DEBUG' in RH_RF95.cpp for detailed debug info");
  while (1);
 }
 Serial.println("LoRa radio init OK!");

  if (!rf95.setFrequency(RF95_FREQ)) {
  Serial.println("setFrequency failed");
  while (1);
 }
 Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

 rf95.setTxPower(15, false);

 //Defaults after init are 437.0MHz, 15dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on
 //Changing the default modem settings to one of the RadioHead default settings
 while(!rf95.setModemConfig(2))
 {
  Serial.println("LoRa radio modem settings change failed");
 }
 //Setting 0 (Bw125Cr45Sf128) - Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on. Default medium range
 //Setting 1 (Bw500Cr45Sf128) - Bw = 500 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on. Fast+short range.
 //Setting 2 (Bw31_25Cr48Sf512 ) - Bw = 31.25 kHz, Cr = 4/8, Sf = 512chips/symbol, CRC on. Slow+long range.
 //Setting 3 (Bw125Cr48Sf4096) - Bw = 125 kHz, Cr = 4/8, Sf = 4096chips/symbol, CRC on. Slow+long range.

 Serial.println("Initialisation finished");
}


void loop()
{
 if (EEPROM.read(0) == 0)// This only occurs when the satellite first launches from the satellite
 {
  Serial.println("satellite launches");
  launch(); //shuts down microcontroller for 4 seconds to represent 45 minutes after launch
  EEPROM.update(0,1); //updates that the satellite has been launched only after the launch cycle is complete
  Serial.begin(115200);
  delay(100);
  satTelemetry[0] = 1;
  for (int i = 1; i< sizeof(satTelemetry); i++) //fills the telemetry packet with random data for testing purposes
  {
    satTelemetry[i] = random(255);
  }
 }
 Serial.begin(115200);
 delay(50);

 Serial.println("Begin collecting data");
 delay(10);
 collectData(); //collecting data - battery checks, check light sensing diode, current check
 Serial.println("Finish collecting data");

 Serial.println("begin Transmit");
 delay(10);
 transmit(); //transmit 4 identification packets followed by the telemetry packet
 Serial.println("Transmission complete");


 Serial.println("enter receive cycle");
 delay(10);
 receive(); //places the LoRa into receive mode so it can receive a control command
```

```
  Serial.println("finish receive cycle");

  delay(10);
  Serial.println("enter idle cycle");
  delay(10);
  idleBeacon(); //enters a variable lenght low power mode to conserve electrical power
  Serial.println("finish idle cycle");

}

void launch() //This function is for when the satellite 1st launches
{ //if this cycle completes then the EEPROM variable is set to true and it will not be called again if power is disconnected
  rf95.sleep(); //places the radio into sleep mode to prevent radio transmissions
  Serial.flush();
  delay(100);
  for (int i = 0; i < 5; i++)
  {
    LowPower.powerSave(SLEEP_1S, ADC_OFF, BOD_OFF, TIMER2_OFF);
    // the first part "SLEEP_1S" can also be changed to SLEEP_2S, SLEEP_4S or SLEEP_8S for 2, 4 or 8 seconds
  }
  delay(100);
  rf95.setModeIdle(); // returns the radio to idle mode ready for the next cycle which is data collection and transmit
  delay(100);
}

void transmit()
{
  for (int i=1; i<5; i++)
  {
  Serial.print("Sending Ident "); Serial.println(i);
  satAddress[0]=i; //this is the unique identifier for each identification radio packet, sequential count from 1 to 4
  rf95.send((uint8_t *)satAddress, 5); //send 5 Bytes of data (8 bit unsigned integer)
  delay(10);
  rf95.waitPacketSent(); //waits for the transmission to complete
  delay(100);
  }

  delay(100);
  Serial.println("Sending Telemetry ");
  rf95.send((uint8_t *)satTelemetry, 50);
  delay(10);
  rf95.waitPacketSent();
  satTelemetry[0] = satTelemetry[0]+1; //Increases the first byte of the telemetry packet by 1 for testing purposes
  delay(100);
}

void receive()
{
  unsigned long timeStamp = millis();
  delay(10);
  do
  {
    if (rf95.available()) //this acts on the interrupt on pin 2 (DIO0 from the LoRa module)
    {
      uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
      uint8_t len = sizeof(buf);

      if (rf95.recv(buf, &len)) //this is where the beacon collects data and functions can be written dependant on RX'ed command
      {
        Serial.print("Got: ");
        Serial.println((char*)buf);
        Serial.println("Doing something");
      }
      else
      {
        Serial.println("Receive failed");
      }
```

```
   }
  } while (millis() < timeStamp + RX_TIME); //while the elapsed time is less than the RX_TIME
}

void idleBeacon()
{
  rf95.sleep();
  delay(100);
  for (int i = 0; i < 2; i++) //powers down beacon for 2 seconds
  {
    //these are the other options available for the power down phase of the beacon software cycle
    //LowPower.powerSave(SLEEP_1S, ADC_OFF, BOD_OFF, TIMER2_OFF);
    LowPower.idle(SLEEP_1S, ADC_OFF,TIMER2_OFF, TIMER1_OFF, TIMER0_OFF, SPI_OFF, USART0_OFF, TWI_OFF);
    //LowPower.powerStandby(SLEEP_1S, ADC_OFF, BOD_OFF);

  }
  delay(100);
  rf95.setModeIdle();
}

void collectData()
{
  // The following are a series of commands that were used during the VIVA presentation and were
  // kept in the code so that something happens during the collect data phase and the current
  // consumption can be measured and approximated for normal beacon operation. This phase can be
  // filled with functions that collect data from other satellite systems.
  digitalWrite(2,HIGH); //power for light sensing diodes
  delay(1000);

  int batRead=0;
  for(int i = 0;i<50;i++)
  {
  batRead = batRead + analogRead(A3);
  }
  satBatVoltage = batRead/50;
  Serial.print("Battery voltage is: "); Serial.println(satBatVoltage);

  solarPanelVoltage =  solarPanelVoltage+1;
  if ( solarPanelVoltage == 100)
  {
    solarPanelVoltage = 0;
  }
  Serial.print("Solar Panel Voltage: ");
  Serial.println(solarPanelVoltage);

  int diode1=0;
  int diode2=0;
  int diode3=0;
  for (int j=0;j<50;j++)
  {
  diode1 = diode1 + analogRead(A0);
  diode2 = diode2 + analogRead(A1);
  diode3 = diode3 + analogRead(A2);
  }
  photoDiode1 = diode1/50;
  photoDiode2 = diode2/50;
  photoDiode3 = diode3/50;

   Serial.print("Diode 1: ");
  Serial.print(photoDiode1);
   Serial.print(", Diode 2: ");
  Serial.print(photoDiode2);
   Serial.print(", Diode 3: ");
  Serial.println(photoDiode3);
  digitalWrite(2,LOW);
  delay(500);
}
```

FULL BEACON SOFTWARE CYCLE FOR GROUND TESTING

```
/*
 * Beacon_Transmitter - Ground Station testing - Full beacon cycle
 *
 * This program is designed to carry out the full software cycle for the Satellite Beacon for Ground
 * testing purposes. The software cycle comprises of five phase...
 *
 * 1) Initialisation/Launch - At launch the radio and microcontroller initialises once power is applied and the
 * both are put to sleep with the shutdown after launch time being for 30 minutes for launch requirements
 * 2) Data collection - The microcontroller does collect any telemetry data but uses 50 Bytes of random data
 * 3) Data Transmission - 5 radio packets are transmitted. The first 4 packets contain the satellite identification
 * (packet number,23,AF,23,AF) and the fifth packet containing 50 Bytes of random telemetry data
 * 4) Receive mode - The satellite radio enters receive mode for 60 seconds where it can accept a control command
 * 5) Low power idle mode - The satellite enters a low power mode for between 4-6 minutes where the radio is
 * turned off and the microcontroller is set to idle
 *
 * Author:  Travis McKee
 * Date:    9 June 2020
 *
 */

#include <SPI.h>
#include <RH_RF95.h>
#include <LowPower.h>
#include <EEPROM.h>

#define RFM95_CS 10
#define RFM95_RST 4
#define RFM95_INT 2
#define RX_TIME 60000

// Set the Transmit/Receive frequency to 437MHz
#define RF95_FREQ 437.0

// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);

char radiopacket[50];
uint8_t satTelemetry[50];
uint8_t satAddress[5] = {0,35,175,35,175}; //unique satellite identification - Hexadecimal = 0x23AF
boolean launched = false;

void setup()
{
 Serial.begin(115200);

 Serial.println("Initialisation Begins");
 pinMode(RFM95_RST, OUTPUT);
 digitalWrite(RFM95_RST, HIGH);

 // manual reset
 digitalWrite(RFM95_RST, LOW);
 delay(10);
 digitalWrite(RFM95_RST, HIGH);
 delay(10);

 EEPROM.write(0,0); //This uses the EEPROM to store if the satellite has just been launched (0) or if it is in orbit (1)

 while (!rf95.init()) {
  Serial.println("LoRa radio init failed");
  Serial.println("Uncomment '#define SERIAL_DEBUG' in RH_RF95.cpp for detailed debug info");
  while (1);
 }
 Serial.println("LoRa radio init OK!");

 // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM
```

```
  if (!rf95.setFrequency(RF95_FREQ)) {
    Serial.println("setFrequency failed");
    while (1);
  }
  Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

  rf95.setTxPower(15, false);

  //Defaults after init are 437.0MHz, 15dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on
  //Changing the default modem settings to one of the RadioHead default settings
  while(!rf95.setModemConfig(2))
  {
    Serial.println("LoRa radio modem settings change failed");
  }
  //Setting 0 (Bw125Cr45Sf128) - Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on. Default medium range
  //Setting 1 (Bw500Cr45Sf128) - Bw = 500 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on. Fast+short range.
  //Setting 2 (Bw31_25Cr48Sf512 ) - Bw = 31.25 kHz, Cr = 4/8, Sf = 512chips/symbol, CRC on. Slow+long range.
  //Setting 3 (Bw125Cr48Sf4096) - Bw = 125 kHz, Cr = 4/8, Sf = 4096chips/symbol, CRC on. Slow+long range.

  Serial.println("Initalisation finished");
}


void loop()
{
  if (EEPROM.read(0) == 0) //carried out the following after release from launch vehicle
  {
    Serial.println("satellite launches");
    launch(); //shuts down radio and processor for 30 minutes
    EEPROM.update(0,1); //updates EEPROM variable only after launch function is complete
    Serial.begin(115200);
    delay(100);
    satTelemetry[0] = 1;
    for (int i = 1; i< sizeof(satTelemetry); i++) //random data for the telemetry radio packet
    {
      satTelemetry[i] = random(255);
    }
  }
  Serial.begin(115200);
  delay(50);

  Serial.println("Begin collecting data");
  collectData();
  Serial.println("Finish collecting data");
  delay(10);

  Serial.println("begin Transmit");
  transmit();
  Serial.println("Transmission complete");
  delay(10);


  Serial.println("enter receive cycle");
  receive();
  Serial.println("finish receive cycle");
  delay(10);

  Serial.println("enter idle cycle");
  idleBeacon();
  Serial.println("finish idle cycle");
  delay(10);
}

void launch()
{
  rf95.sleep(); // puts radio into sleep mode
  Serial.flush();
```

```
 delay(100);
 for (int i = 0; i < 225; i++) // puts processor into power save mode for 30 minutes
 {
  LowPower.powerSave(SLEEP_8S, ADC_OFF, BOD_OFF, TIMER2_OFF);
 }
 delay(100);
 rf95.setModeIdle(); // puts radio into idle mode ready for operation
 delay(100);
}

void transmit()
{
 for (int i=1; i<5; i++) // send 4 identification radio packets
 {
 Serial.print("Sending Ident "); Serial.println(i);
 satAddress[0]=i; //unique packet identifier
 rf95.send((uint8_t *)satAddress, 5); //send 5 Bytes of data (identifier, address, address)
 delay(10);
 rf95.waitPacketSent(); // wait until transmission is complete
 delay(100);
 }

 delay(100);
 Serial.println("Sending Telemetry ");
 rf95.send((uint8_t *)satTelemetry, 50); //send 50 Bytes of telemetry data
 delay(10);
 rf95.waitPacketSent(); // wait until transmission is complete
 satTelemetry[0] = satTelemetry[0]+1; //increases the first Bytes by 1 to identify each telemetry packet
 delay(100);
}

void receive()
{
 unsigned long timeStamp = millis();
 delay(10);
 do
 {
  if (rf95.available()) //This starts if an external interrupt is RX'ed on the 2 pin (from LoRa module DIO0 pin)
  {
   uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
   uint8_t len = sizeof(buf);

   if (rf95.recv(buf, &len)) //this performs an oepration based on the received data
   {
    Serial.print("Perform control command function"); //this is where the control functions should be placed
   }
   else
   {
    Serial.println("Receive failed");
   }
  }
 } while (millis() < timeStamp + RX_TIME);
}

void idleBeacon()
{
 rf95.sleep(); // Puts the radio in sleep mode
 delay(100);
  for (int i = 0; i < 30; i++) // Puts the processor in idle mode for 4 minutes
 {
  LowPower.idle(SLEEP_8S, ADC_OFF,TIMER2_OFF, TIMER1_OFF, TIMER0_OFF, SPI_OFF, USART0_OFF, TWI_OFF); //for arduino pro mini
 }
 for (int i = 0; i < random(120); i++) // Puts the processor into idle mode for between 1 and 120 seconds
 {
  LowPower.idle(SLEEP_1S, ADC_OFF,TIMER2_OFF, TIMER1_OFF, TIMER0_OFF, SPI_OFF, USART0_OFF, TWI_OFF); //for arduino pro mini
 }
 delay(100);
```

```
  rf95.setModeIdle(); // Puts the radio into the idle mode ready for operation
}

void collectData()
{
 Serial.println("*Some telemetry data is collected*");
 // This is where the function should be placed to collect telemetry data from other satellite systems
}
```

INITIAL SOFTWARE CYCLE FOR GROUND RECEIVING STATION

```
/*
 * Ground_Receiver station - Original clock counting technique
 *
 * This program is used for the initial development of the ground receiving station. The program will
 * initialise the radio and GPS units when power is applied and wait for a LoRa signal to be received.
 * When a valid signal is received (header and packet data CRC checks c/o successfully by the radios
 * in-built software) then a timestamp is taken by counting clock cycles, which is saved with the msg data.
 * The first received LoRa signal resets the clock count so that it begins at 0 for the time measurements
 * Once the telemetry data is received (timestamp and data saved) then the system will wait for the next
 * GPS PPS signal to arrive in which a GPS timestamp will be taken, and the current GPS data is read from
 * the GPS unit. The positional data (latitude/longitude/altitude) and time data (UTC time) is saved. The
 * program will then save the time stamp of the next 3 PPS pulses and use this to determine the average
 * number of clock cycles in one second which will get the instantaneous frequency of the oscillator. All
 * the saved data (messages, timestamps, freq drift data and GPS data) is then displayed on the serial monitor.
 * The data can also be sent through the serial connection to a computer for processing the data
 *
 *
 * Author:  Travis McKee
 * Date:    9 June 2020
 *
 */

#include <SPI.h>
#include <RH_RF95.h>
#include <SoftwareSerial.h>
#include <TinyGPS.h>

#define RFM95_CS 10
#define RFM95_RST 4
#define RFM95_INT 2 //receive the external interrupt LoRa RXDone/TXDone signal on pin 3
#define PPS_pin 3 //receive the external interrupt PPS signal on pin 3

// Set the Transmit/Receive frequency to 437MHz
#define RF95_FREQ 437.0

// Single instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);

// set up the GPS receiver connections, RXD(pin 5) & TXD(pin 6)
SoftwareSerial mySerial(5, 6);
TinyGPS gps;

unsigned long loRaTimeStamp[5], oscFreq;
int identData[4][5],telemetryData[50];
long lat, lon, alt;
unsigned long posAge, timeAge, date, time;
volatile unsigned long overflowCount;
boolean telemetryRX = false;
int count = 0;
volatile int count1 = 0;
```

```
volatile unsigned long GPSTimeStamp[4];

void setup()
{
 //Sets up serial 0 to print on the serial monitor
 Serial.begin(115200);
 while (!Serial) {
  delay(1);
 }
 delay(100);
 Serial.println("** Initialisation begins **");

 // set the data rate for the SoftwareSerial port for the GPS module
 mySerial.begin(9600);

 //sets up the Timer1 to be used as clock cycle counter will resolution 1/f = 62.5nSec
 noInterrupts();
 TCCR1A = 0;// set entire TCCR1A register to 0 to reset the Timer1 settings
 TCCR1B = 0;// same for TCCR1B
 TCNT1 = 0; //initialise counter value to 0
 OCR1A = 65535; //sets the top value for when the TImer 1 counter resets and the overflowCount increments by 1
 TCCR1B |= (1 << WGM12); //turn on CTC mode
 TCCR1B |= (1 << CS10); //prescaler set to 1 (62.5nS resolution)
 TIMSK1 |= (1 << OCIE1A); //sets up the compare ISR for timer1 counter reaches the OCR1A value
 interrupts();

 pinMode(RFM95_RST, OUTPUT);
 pinMode(PPS_pin, INPUT);
 digitalWrite(RFM95_RST, HIGH);

 // manual reset
 digitalWrite(RFM95_RST, LOW);
  delay(10);
 digitalWrite(RFM95_RST, HIGH);
 delay(10);

 while (!rf95.init()) {
  Serial.println("LoRa radio init failed");
  Serial.println("Uncomment '#define SERIAL_DEBUG' in RH_RF95.cpp for detailed debug info");
  while (1);
 }
 Serial.println("LoRa radio init OK!");

 // Defaults after init are 434.0MHz, modulation GFSK_Rb250Fd250, +13dbM
 if (!rf95.setFrequency(RF95_FREQ)) {
  Serial.println("setFrequency failed");
  while (1);
 }
 Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

 // you can set transmitter powers from 5 to 23 dBm:
 rf95.setTxPower(15, false);

 //Defaults after init are 437.0MHz, 15dBm, Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on
 //Changing the default modem settings to one of the RadioHead default settings
 while(!rf95.setModemConfig(2))
 {
  Serial.println("LoRa radio modem settings change failed");
 }
 //Setting 0 (Bw125Cr45Sf128) - Bw = 125 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on. Default medium range
 //Setting 1 (Bw500Cr45Sf128) - Bw = 500 kHz, Cr = 4/5, Sf = 128chips/symbol, CRC on. Fast+short range.
 //Setting 2 (Bw31_25Cr48Sf512) - Bw = 31.25 kHz, Cr = 4/8, Sf = 512chips/symbol, CRC on. Slow+long range.
 //Setting 3 (Bw125Cr48Sf4096) - Bw = 125 kHz, Cr = 4/8, Sf = 4096chips/symbol, CRC on. Slow+long range.
```

```
 Serial.println("** Initialisation complete **");
}

void loop()
{
 // This waits until an external interrupt is received on the Uno 2 pin from the DIO0 pin on the LoRa module
 // The signal that arrives is the LoRa RXDone signal that tells the processor there is a MSG waiting that
 // has passed the preamble and data payload integrity checks
 if (rf95.available())
 {
  uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);

  if (rf95.recv(buf, &len))
  {
   if (count == 0) //reset the clock counter variables if this is the first received data packet
   {
     overflowCount = 0; //reset clock overflow counter
     TCNT1 = 0; //initialise counter value to 0
   }
   loRaTimeStamp[count] = ((overflowCount*65535)+TCNT1); //Time stamp the radio packet that has arrived
    if (len == 5) //if the radio packet is an identification packet
    {
     for (int i=0;i<len;i++)
     {identData[count][i] = buf[i];} // save the RX'ed data into the identification data array
     count++;
    }
    else //if the radio packet is a telemetry packet
    {
     for (int i=0;i<len;i++)
     {telemetryData[i] = buf[i];} // save the RX'ed into the telemetry data array
     telemetryRX = true; //used to tell the processor the radio packets have been RX'ed and to collect GPS data
    }


   //look at GPS data after receiving  all the radio packets - will get the current UTC to the second
   if (telemetryRX)
   {
    Serial.println("**** ALL MESSAGES NOW RECEIVED ****");
    Serial.println();

    //The next command turns on the external interrupt on pin 3 which is the PPS signal from the GPS Module
    noInterrupts();
    attachInterrupt(digitalPinToInterrupt(3), gpsTimeStamp, RISING);
    EIFR |= 3; // clear ext intrupt flag
    interrupts();

    while(count1<1) {}; //tells the processor to wait for the next PPS signal, may be a better way for this

     unsigned long start = millis();
     bool newdata = false;

     //this reads in the GPS data - may need to consider if the data is not ready (not enough GPS satellites locked)
     while (millis() - start < 5000) //look to see if this can be done better, i could not bypass it
     {
      if (mySerial.available()) // If there is a connection with the GPS module
      {
       char c = mySerial.read(); // read all the current data from the GPS module.
                                 // It has the current UTC from the last PPS Pulse
       //Serial.print(c);  // uncomment to see raw GPS data
       if (gps.encode(c)) //feeds all the NEMA data from the GPS module to the gps object in Arduino
       {
```

```
            newdata = true; //tells the processor the GPS data is ready to collect
            break;  // uncomment to print new data immediately!
          }
        }
      }
    if (newdata) {gpsdump(gps);} //call a function that collects the required GNSS information from the GPS object

    while(count1!=4){}; //This lets the processor wait until it receives its fourth GNSS PPS timestamp
    detachInterrupt(digitalPinToInterrupt(3));

    oscFreq =(GPSTimeStamp[3]-GPSTimeStamp[0])/3; //calculates the current frequency of the Unos oscillator

    //print the time stamp and msg data for each message and the GPS data saved in each variable
    printData();

    //reset all variables ready for the next lot of RX'ed radio packets
    count = 0;
    count1 = 0;
    telemetryRX= false;
    }

  }
 }
}

ISR(TIMER1_COMPA_vect) //if using RadioHead library, the RH_ASK.h and RH_ASH.cpp need to be deleted so that this works
{
overflowCount++; //perform interupt when timer1 hits 65535 (top of register)
}

void gpsTimeStamp()
{
 if (count1 < 4)
 {
 GPSTimeStamp[count1] = ((overflowCount * 65535) + TCNT1); //save the PPS timestamp into an array
 count1++;
 }
}


void gpsdump(TinyGPS &gps) // collects the data required by the ground station from the GPS object with the latest GNSS data
{
 gps.get_position(&lat, &lon, &posAge);
 gps.get_datetime(&date, &time, &timeAge);
 alt = gps.altitude(); //takes a while to obtain after startup, the GPS module must be run for a while for reliable Altitude
 //gps.crack_datetime(&year, &month, &day, &hour, &minute, &second, &hundredths, &crackTimeAge);
}


void printData() //this prints all the data on the Arduino serial monitor
{
 for (int i=0; i<4; i++)
 {
   Serial.print("identification msg number "); Serial.print(i+1); Serial.print(" timestamp is: "); Serial.println(loRaTimeStamp[i]);
   Serial.print("and the data in the msg was: ");
    for(int j=0;j<5; j++)
    { Serial.print(identData[i][j],HEX); Serial.print(" ");}
    Serial.println();
 }Serial.println();

 //print the time srtamp and msg data for the telemtry message
 Serial.print("the timestamp for the telemetry msg is: "); Serial.println(loRaTimeStamp[4]);
 Serial.print("and the data in the msg was: ");
```

```
   for(int j=0;j<50; j++)
   { Serial.print(telemetryData[j],HEX); Serial.print(" ");}
   Serial.println(); Serial.println();

  Serial.print("the GPS Time stamps are: ");
  for(int i=0;i<4; i++)
  {Serial.print(GPSTimeStamp[i]);; Serial.print(" , ");}
  Serial.println(); Serial.println();

  Serial.print("The current frequency of the oscillator is "); Serial.println(oscFreq);
  Serial.println();

  Serial.print("Lat/Long(10^-5 deg): "); Serial.print(lat); Serial.print(", "); Serial.print(lon);
  Serial.print(" Fix age: "); Serial.print(posAge); Serial.println("ms.");

  Serial.print("Date(ddmmyy): "); Serial.print(date); Serial.print(" Time(hhmmsscc): ");
  Serial.print(time);
  Serial.print(" Fix age: "); Serial.print(timeAge); Serial.println("ms.");
  Serial.print("Alt(cm): "); Serial.println(alt); Serial.println();
}
```

FUTURE WORK AND SYSTEM EXTENSIONS

REQUIRED WORK FOR CONTINUED DEVELOPMENT

A reduction of the total uncertainty in the time measurement technique employed by the ground station is required before further development of this system can continue. The large total uncertainty (1.5-1.7ms) can be reduced by looking at two aspects of the time measurement technique:

1. The clock counting method using the *TIMER1_COMPA_vect* ISR

There are many (hundred to thousands) clock cycles missed due to implementing the clock counting method using the *TIMER1_COMPA_Vect* ISR to update an *overflowCount* variable. This is mostly likely the cause of the large increase in uncertainty. An alternative method of counting clock cycles needs to be investigated, with the Arduino *Micros()* function a good place to start the research as well as researching timer capture mode

2. The LoRa module processing time uncertainty

The timestamps for the LoRa signal arrival are taken when the *RXDone* interrupt is sent from the DIO0 pin of the LoRa module which occurs after all the LoRa software processing is done. This has an uncertainty of up to 30µs when implemented in the ground receiving station software. Capturing a timestamp when the LoRa module first receives the RF signal rather than after the processing would eliminate this uncertainty. The LoRa module allows for a Channel Activity Detection (CAD) mode which could timestamp when the signal first arrives, but this mode is not implemented correctly with the RadioHead library. To implement this mode an alternative library is required to be investigated that can implement the CAD mode or this can be added to the RadioHead manually. There are a few available resources in which the LoRa module can be controlled directly by manually adjusting the registers and could be used as a template

If the total uncertainty in time measurement is reduced down to an acceptable level, then the next steps in the development of the UHF radio beacon system would be to:

- Develop an algorithm that can estimate the satellites position based on the LoRa signals time difference of arrival at three ground stations that have a Lat/Long position (can be programmed using C, MATLAB, Python etc)
- Review the current ground station software for improvements such as performing regular adjustment of the true oscillator frequency rather than after each time it receives a LoRa signal (this will make it more efficient if there are many satellite beacons transmitting)
- Perform ground testing with a single transmitter and multiple ground stations at a known distance to verify and validate the time measurement techniques and the TDOA calculation of the transmitter position
- Investigation and selection of space ready components for the satellite beacon system
- Full PCB design for satellite radio beacon
- Investigate space-ready antennas for the satellite beacon
- Ground testing carried out on PCB design with space ready components
- Develop a secure receive function for the beacon software
- Develop a server system for collecting data from the ground stations and providing all calculated positions to connected users
- A GUI software program for displaying positions of satellites for all connected users
- Develop a website that provides real-time updates of all satellites
- Begin analysis of a ground testing regime that will carry out all mission readiness testing

OPTIONAL EXTENSIONS FOR CONSIDERATION

Some optional extras that may be worthwhile pursuing but are not essential

- Creating a Full PCB design for ground receiving station
- Reduction of the satellite voltage supply to 1.8V (need to develop the associated Arduino support and test a new regulator with the right output)
- Transfer the beacon software from the Arduino IDE to a programme that use the C or python language directly
- Incorporating the LoRa module components into the satellite beacon PCB design (as opposed to soldering the RFM96 LoRa module PCB onto the satellite beacon PCB)
- Begin a website/blog for spreading information about this system
- Campaign to government agencies and defence entities to present information about this system