# THE DESIGN FOR AN AERIAL MONITORING SYSTEM FOR ENVIRONMENTAL SURVEILLANCE APPLICATIONS

AMIR FARHAN BIN MOHD RASIDI

BACHELOR OF ELECTRICAL ENGINEERING (ELECTRONICS) WITH HONOURS

UNIVERSITI MALAYSIA PAHANG

# UNIVERSITI MALAYSIA PAHANG

**DECLARATION OF THESIS AND COPYRIGHT**

| | | |
|---|---|---|
| Author's Full Name | : | AMIR FARHAN BIN MOHD RASIDI |
| Date of Birth | : | 9/10/1997 |
| Title | : | THE DESIGN FOR AN AERIAL MONITORING SYSTEM FOR ENVIRONMENTAL SURVEILLANCE APPLICATIONS |
| Academic Session | : | 2022 |

I declare that this thesis is classified as:

| | | |
|---|---|---|
| ☒ | CONFIDENTIAL | (Contains confidential information under the Official Secret Act 1997) * |
| ☒ | RESTRICTED | (Contains restricted information as specified by the organization where research was done) * |
| ☒ | OPEN ACCESS | I agree that my thesis to be published as online open access (Full Text) |

I acknowledge that Universiti Malaysia Pahang reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

| | |
|---|---|
| Amir | [illegible] |
| (Student's Signature) | (Supervisor's Signature) |
| 971009-43-5169 | Professor Madya Ir. Dr Nurul Hazlina Binti Noordin |
| New IC/Passport Number | Name of Supervisor |
| Date: 21/06/2022 | Date: 22/06/2022 |

NOTE: * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

Universiti Malaysia PAHANG
Engineering • Technology • Creativity

## SUPERVISOR'S DECLARATION

I hereby declare that I have checked this thesis. This thesis is adequate in terms of scope and quality for the award of Bachelor of Electrical Engineering (Electronics) with Honors.

[illegible]

(Supervisor's Signature)

| | |
|---|---|
| Full Name | : Professor Madya Ir. Dr Nurul Hazlina Binti Noordin |
| Position | : Associate Professor |
| Date | : 21/06/2022 |

## STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

Amir

(Student's Signature)

| | |
|---|---|
| Full Name | : AMIR FARHAN BIN MOHD RASIDI |
| I.D. Number | : EA18013 |
| Date | : 21/06/2022 |

# THE DESIGN FOR AN AERIAL MONITORING SYSTEM FOR ENVIRONMENTAL SURVEILLANCE APPLICATIONS

AMIR FARHAN BIN MOHD RASIDI

A thesis submitted in fulfilment of the requirements
for the award of the Bachelor of Electrical Engineering (Electronics) with Honors

Faculty of Electrical & Electronics Engineering

UNIVERSITI MALAYSIA PAHANG

JUNE 2022

## ACKNOWLEDGEMENTS

All praise be to Allah SWT, the only one who is the most merciful for providing the strength to complete this thesis. I want to give this token of appreciation to Universiti Malaysia Pahang (UMP) for financial support for this project.

From the bottom of my heart, I would want to thank my supervisor, Professor Madya Ir. Dr. Nurul Hazlina Binti Noordin, for her outstanding guidance and encouragement. As a supervisor, she always gives out ideas and responds to every question upon completing this project. I am beyond grateful for her unwavering support. I am also incredibly thankful for all the lecturers of Universiti Malaysia Pahang (UMP). They are willing to help whenever I reach for their help and guidance.

I am also expressing my appreciation to Intel Microelectronics (M) Sdn. Bhd. for providing the FPGA DE10-Lite Development Kit for this project and basic knowledge of using the FPGA board.

A special thanks to my beloved family and friends for their constant moral support and motivation. Their motivation keeps me going and brings out my potential to give my all for this project. Lastly, I would like to express my heartfelt gratitude to anyone who contributed to the process of succeeding in this project. I hope that all the research done on putting this project into words will be of great use and bring about great inspiration.

# ABSTRAK

Bahaya semula jadi berlaku pada kadar yang membimbangkan di seluruh dunia. Sebelum memulakan usaha menyelamat dan bantuan yang sesuai di kawasan yang dilanda bencana, perlu menjalankan siasatan. Pegawai tentera selalunya mesti meronda di lokasi berisiko untuk mencari sebarang potensi ancaman, aktiviti haram atau pencerobohan dalam sempadan negara yang boleh membahayakan nyawa penduduk. Lokasi sedemikian membawa risiko yang sangat ketara kepada kehidupan manusia.

Selain daripada UAV atau dron berteknologi tinggi, nanosatelit seperti CanSat boleh digunakan untuk pengawasan dan pemantauan udara dalam Sistem Udara Tanpa Pemandu (UAS). Ia juga boleh menyediakan data masa nyata ke bilik kawalan atau komputer. Jenis telemetri CanSat digunakan dalam projek ini untuk mengumpul dan menghantar data daripada penerbangan dan keadaan persekitaran dalam masa nyata untuk diproses oleh stesen pangkalan. Bahagian utama yang digunakan dalam reka bentuk CanSat ialah mikropengawal dan bekalan kuasa. Sebagai tambahan kepada yang dinyatakan di atas, misi mungkin menggabungkan elemen lain. FPGA dalam kapal angkasa ini membolehkan pemprosesan yang lebih cekap dengan memindahkan reka bentuk kepada perkakasan.

Terdapat banyak kesukaran dalam mereka bentuk nanosatelit seperti CanSat. Pertama, sukar untuk mereka bentuk nanotelit menggunakan teknologi FPGA dan mengintegrasikannya dengan penderia kerana FPGA terdiri daripada elemen logik mudah, yang boleh digabungkan untuk melaksanakan persisian yang dinyatakan sebelum ini, tetapi hanya berinteraksi dengan kebanyakan elektronik digital lain. Reka bentuk CanSat perlu menyesuaikan mikropengawal sebagai unit utama dengan semua subsistem utama yang terdapat dalam satelit, seperti kuasa, penderia dan sistem komunikasi, ke dalam volum minimum. Selain itu, penggunaan mikropengawal tersebut adalah kurang cekap untuk mendapatkan data masa nyata berkelajuan tinggi dalam aplikasi pengawasan udara

Matlamat utama kerja ini adalah untuk membangunkan telemetri CanSat menggunakan Papan Pembangunan FPGA DE10-Lite (Altera Max 10), Pengawal Mikro Arduino Uno R3, dan penderia yang sesuai (Penderia suhu dan kelembapan, Modul GPS, Penderia Barometrik, Penderia Accelerometer) dan sistem komunikasi tanpa wayar. CanSat kemudiannya akan memantau dan mengumpul data telemetri masa nyata dari stesen bumi. Akhir sekali, data telemetri dari pelbagai tempat dan ketinggian di bawah 1km akan dianalisis.

## ABSTRACT

Natural hazards are occurring at an alarming rate around the world. Before beginning suitable rescue and aid efforts in a disaster-stricken area, it is required to conduct an investigation. Military officials often must patrol risky locations to seek any potential threat, illegal activity, or intrusion within a country's borders that can put the lives of inhabitants in jeopardy. Such locations carry a very significant risk to human life.

Other than UAVs or high-tech drones, nanosatellites such as CanSat can be utilized for aerial surveillance and monitoring in Unmanned Aerial Systems (UAS). It can also provide real-time data to a control room or computer. The telemetry type of CanSat is utilized in this project to gather and send data from the flight and environmental conditions in real-time for processing by a base station. The major parts employed in the CanSat design are the microcontroller and power supply. In addition to those stated above, the mission may incorporate other elements. These FPGAs in spacecraft enable more efficient processing by transferring the design to hardware.

There are many difficulties in designing nanosatellites such as CanSat. Firstly, is difficult in designing nanosatellites using FPGAs technology and integrate them with the sensors because FPGAs consist of simple logic elements, which can be combined to implement the peripherals previously mentioned, but only interact with most other digital electronics. The design of CanSat needs to fit the microcontroller as the main unit with all the major subsystems found in a satellite, such as power, sensors, and a communication system, into the minimal volume. Besides, using those microcontrollers is less efficient to get the high-speed real-time data in an aerial surveillance application.

The primary goal of this work is to develop a telemetry CanSat utilizing an FPGA DE10-Lite Development Board (Altera Max 10), Arduino Uno R3 Microcontroller, and an appropriate sensor (Temperature and humidity sensor, GPS Module, Barometric Sensor, Accelerometer Sensor) and a wireless communication system. The CanSat will then monitor and collect real-time telemetry data from a ground station. Finally, the telemetry data from various places and altitudes below 1km will be analyzed.

# TABLE OF CONTENT

# LIST OF TABLES

## LIST OF FIGURES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| ADC | Analog-to-Digital Converter |
| ASIC | Application-Specific Integrated Circuit |
| CPLD | Complex Programmable Logic Device |
| CAD | Computer-Aided Design |
| DSP | Digital Signal Processing |
| FPGA | Field Programmable Gate Array |
| FTDI | Future Technology Devices International Limited |
| GPS | Global Positioning System |
| HDL | Hardware Description Language |
| IC | Integrated Circuit |
| IIR | Infinite impulse response |
| INS | Inertial Navigation System |
| IP | Intellectual Property |
| I2C | Inter-Integrated Circuit |
| LEs | Logic Elements |
| MISO | Master In Slave Out |
| MOSI | Master Out Slave In |
| NTC | Negative Temperature Coefficient |
| PC | Personal Computer |
| RF | Radio Frequency |
| RTL | Register-Transfer Level |
| SCL | Serial Clock line |
| SDA | Serial Data line |
| SPI | Serial Peripheral Interface |
| SoC | System On Chip |
| TNC | Terminal Node Controller |
| UART | Universal Asynchronous Receiver/Transmitter |
| UA | Unmanned Aerial |
| UAS | Unmanned Aerial System |
| VCC | Voltage Common Collector |
| VHDL | Verilog Hardware Description Language (VHDL) |

# LIST OF APPENDICES

# CHAPTER 1

# INTRODUCTION

## 1.1 Project Background

Natural hazards are occurring at an alarming rate around the world. Before beginning suitable rescue and aid efforts in a disaster-stricken area, it is required to conduct an investigation. Military officials often must patrol risky locations to seek any potential threat, illegal activity, or intrusion within a country's borders that can put the lives of inhabitants in jeopardy. Such locations carry a very significant risk to human life [1].

The Aerial Surveillance System can readily be employed to complete this task without endangering human lives. Furthermore, the speed of operation will be faster with drones because they may fly unfettered for days and days on end. An aerial Surveillance System is a flying machine that can be remotely controlled to deliver real-time data to a control room [1]. The name Unmanned Aerial System (UAS) was coined to emphasize that an Aerial Surveillance system is a component of a larger system that includes ground stations, launch mechanisms, and other components [1]. A UAS is a system that includes the necessary hardware, communication, and humans to operate an unmanned aircraft.

Furthermore, in the recent decade, developing technologies for space education have risen rapidly [2]. Nowadays, miniaturized satellites play an important role in academics and scientific projects for monitoring or imaging [2]. Small instruments can be integrated into the payload and operated remotely and telemetry for data acquisition [2].

Nano-satellites such as CanSat can be used for aerial surveillance and monitoring in Unmanned Aerial Systems (UAS), in addition to deploying Unmanned Aerial Vehicles (UAVs) or high-tech drones [2]. It can also transmit real-time data to a ground station or PC. The telemetry type in CanSat is utilized in this project to gather and transmit data from the flight and environmental conditions in real-time to a ground station for processing. In CanSat design, the Microcontroller and power supply are the main elements used in every CanSat. Aside from the elements described above, more may be added following the task assigned to it. The main unit used for this project is the FPGA DE10-Lite Development Board and Arduino Uno R3. Using these FPGAs in spacecraft will allow more efficient processing by moving the design onto hardware.

## 1.2 Problem Statement

There are many difficulties in designing nanosatellites such as CanSat. Firstly, is difficult in designing nanosatellites using FPGAs technology and integrate them with the sensors because FPGAs consist of simple logic elements, which can be combined to implement the peripherals previously mentioned, but only interact with most other digital electronics. CanSat's architecture must include the microcontroller as the core unit, as well as all of the essential subsystems present in a satellite, such as power, sensors, and a communication system, in the smallest possible size. Besides, using those microcontrollers is less efficient to get the high-speed real-time data in an aerial surveillance application.

## 1.3 Objective

This project aims to design a nanosatellite for applying ariel surveillance and ariel monitoring systems. The objectives include: -

i. To design telemetry CanSat using FPGA (Altera Max 10) and Arduino Uno R3 with the suitable sensors (Temperature and Humidity Sensor, Barometric Sensor, GPS, Accelerometer) and wireless communication system.

ii. To monitor and gather real-time telemetry data by the CanSat from a ground station.

iii. To analyze the telemetry data from different locations and altitudes below 1km.

## 1.4 Significance

According to previous studies, there are many types of Ariel monitoring and Surveillance system. In comparison to the writing project, these methods are essential. The author did some research for the last journal and research as a blogger.

The study on Angel Colin and Manuel Jimenez (2017) [2]. The use of CanSat technology to monitor the climate in tiny areas at altitudes below 1 km reveals that the project is based on the same concept as the previous one. The previous project used an Arduino Pro mini 328 microcontroller as the main unit [2], while the author used Arduino Uno R3 Microcontroller and FPGA Altera Max 10 to speed up the processing.

## 1.5 Project Scope

The focus of this study is the system development for ariel monitoring and surveillance by using the nanosatellite CanSat. The scope of this project is:

i. Using FPGA DE10-Lite Development board and Arduino Uno R3 as the main unit.

ii. Integration between FPGA DE10-Lite Development Board, Arduino Uno R3 with all the sensors and send the telemetry data using wireless communication to ground station.

iii. Real-time processing data.

iv. Collect and process telemetry data from different locations and altitudes below 1km.

## 1.6 Thesis Outline

The following is a summary of each of the study's chapters. The dissertation is organized into four chapters, which are as follows:

i. Chapter 2: Literature review

Provide an in-depth analysis of a similar subject on a previous researcher's project. Journals, books, and articles are used as a source of information. An overview of aerial surveillance as Unmanned Aerial System (UAS) is the first topic discussed. The introduction and comparison of FPGA and microprocessors are then discussed. The design approach includes an overview of the CanSat nanosatellites from previous projects or studies. The chapter ended with a review of the CanSat design with different methods.

ii. Chapter 3: Methodology

The methods used to design the CanSat nanosatellites The flow of methodology is the first topic discussed. Block diagrams for I/O are then discussed. Furthermore, briefly discussed the list of components and software used in this project. The flowchart of the system is then discussed with two flowcharts, one for the deployment of the CanSat flowchart and another one for the system flowchart. In addition, this chapter is will be discussed briefly about circuit development, and cost estimation and the chapter ended with a summary of the method used and development.

iii. Chapter 4: Result and discussion

The results and discussion of data collected from the CanSat and transmitted to the PC by the ground station. First, the hardware architecture is presented, followed by telemetry data collected at various altitudes and locations. Following that, the DE10-Lite Development Board's output is presented utilizing 6 seven segments for GPS coordinates and LEDs for Accelerometer data. Furthermore, the Arduino output is monitored using a serial monitor before being converted to Microsoft Excel using the PLX-DAQ Data Acquisition program. After that, the data will be examined.

iv. Chapter 5: Conclusion

This chapter aims to provide a high-level summary of the study's principal findings as well as an explanation of the study's contributions and limitations. In addition, as part of project management, this chapter will compute the entire cost of the project.

# CHAPTER 2

# LITERATURE REVIEW

## 2.1 Introduction

The journal review aims to provide an in-depth analysis of a similar subject on a previous researcher's project. Journals, books, and articles are used as a source of information. This development also involved the operation of circuits, software, and hardware. This chapter also gives insight into the project's device and hardware components. Based on the concept that has been applied, this project can be composed of the previous project to increase efficiency and operability.

## 2.2 Unmanned Aerial System (UAS)

In recent years, the name Unmanned Aerial System (UAS) has been superseded by (UA), which stands for Unmanned Aircraft. The term Unmanned Aircraft System (UAS) was used to emphasize that a UA is part of a broader system that includes ground stations, launch mechanisms, and other components [3]. A system that contains the necessary hardware, communications and humans to operate an unmanned aircraft is referred to as UAS [4].

Unmanned Aircraft Systems (UAS), sometimes known as Unmanned Aerial Systems (UAS), are components of aircraft and related equipment that operate without the presence of a human operator. It can fly autonomously or remotely, and all equipment, including the command, control, and communications (C3) system and personnel required to control the unmanned aircraft, must be evaluated in the context of the system. [5][6][7]. An unmanned aircraft system (UAS) is made up of three main components, as shown in Figure 2.1.

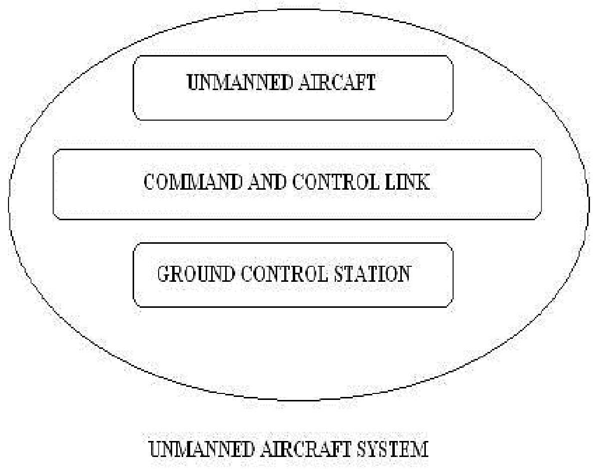


Figure 2. 1 Unmanned Aerial System Model[8]

Unmanned Aircraft Systems (UAS) have recently been deployed for military and civilian purposes. It has recently been acknowledged for its importance and benefits in search and rescue, real-time surveillance, reconnaissance operations, traffic monitoring, hazardous site inspection, and range extension. Furthermore, UAS is appropriate for circumstances when direct human surveillance is too dangerous. UAS is a rapidly expanding field in the unmanned aviation community. "UAS" is a broad phrase that refers to "the entire system, which includes aircraft, control stations, and data linkages." [9][10][11].

Besides the use of unmanned aerial systems (UAS) to collect data for environmental monitoring is becoming increasingly significant [12]. Unmanned Aerial Systems (UAS) have become a critical aspect of environmental monitoring as drone technology has advanced over the last decade, bridging the gap between traditional field research and satellite remote sensing [12]. UAS is a low-cost method of collecting visual data on a broad temporal scale over the electromagnetic spectrum, making it a great tool for monitoring dynamic environmental processes [12].

## 2.3 Field Programmable Gate Array (FPGA)

FPGAs (Field Programmable Gate Arrays) are programmable logic devices [13]. It has configurable connectivity and adjustable logic blocks [14]. FPGAs (Field

Programmable Gate Arrays) are programmable logic devices [13]. It has configurable connectivity and adjustable logic blocks [14]. Custom digital electrical circuits can be implemented on the device thanks to the programmable logic, which allows the circuit design to be modified for a specific application. A circuit can be designed, called a netlist, which can then be synthesized and programmed onto the FPGA. These circuits can be simple as glue logic: simple logic functions to interface chips together or as complex as an entire microprocessor. Before an FPGA can be programmed, the design must be described to a Computer-Aided Design (CAD) program, which takes it and translates it into bits to program the FPGA.

HDL is the language used to describe the design. The two HDL languages commonly used are Verilog and VHSIC Hardware Description Language (VHDL). Similar to software and software libraries, hardware descriptions can be reused and packaged in the form of Intellectual Property (IP) cores. These cores range from Digital Signal Processing (DSP) and arithmetic cores to chip interfaces such as Serial Peripheral Interface (SPI), to processor cores such as the 8051 [15] and OpenSPARC T1 processor [16]. FPGAs allow the custom design of a chip that can be similar in functionality to an ASIC, which would cost overwise cost millions to make [17]. One must take a design, let the software synthesize it, and then program it onto the FPGA. Synthesis is one of the steps done by the FPGA CAD software in implementing an FPGA design.

### 2.3.1 FPGA Workflow

This section will provide a brief description of the stages required in producing a design from the HDL input to programming the FPGA in order to describe how it is implemented on an FPGA [18]. Figure 2.2 illustrates the typical design workflow. The design steps are as follows:

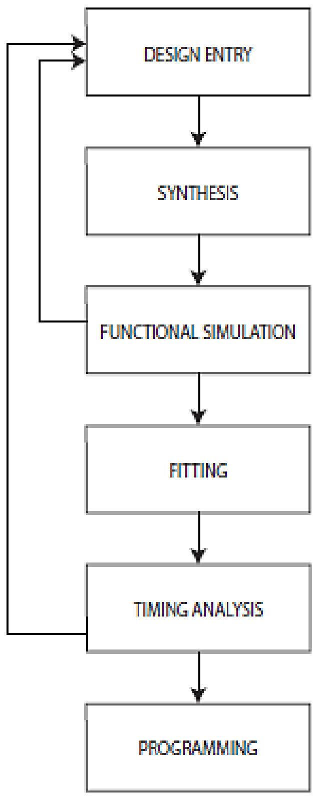


Figure 2. 2 FPGA Workflow[18]

Design Entry - The first step describes the hardware to be implemented using HDL such as Verilog or VHDL. This process is similar to a software programmer writing code. Depending on the CAD tools used, the design can also be created visually by connecting blocks in a schematic diagram [18].

Synthesis - The next step lets the CAD software interpret the HDL code and create a Register-Transfer Level (RTL) netlist. The lines of HDL code are translated into combinational Logic Elements (LEs), latches and registers [18]. The designer can also verify the design by visually examining the netlist to see the CAD software interpretation of the design described by the HDL code [18].

Functional Simulation - The functionality of the design is verified through simulation and test benches [18]. These test benches are written using HDL and specify the input waveforms for the design, and the simulation produces the corresponding output waveforms. This task uses specialized CAD software, such as ModelSim [18].

Fitting - The components of the RTL design are fitted into LEs in the FPGA through placement and routing [18]. The routing step determines the connections between individual LEs and, therefore, the length of each wire.

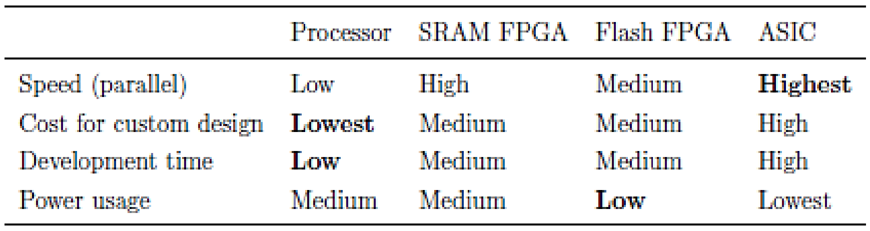
Timing Analysis - The length of each wire and the functionality of each LE determine the timing of the FPGA. The timing is important as it determines the fastest clock speed the design can handle. The timing simulation also helps determine if there might be any glitches that might happen due to the timing of various I/O changing states in the design [18].

Programming - The final step in the CAD workflow is to generate a bitstream and program the FPGA. The bitstream contains each routing switch and Logic Element (LE) configuration in an FPGA. The bits in the bitstream are then shifted into the FPGA, which programs the design onto the FPGA, implementing the HDL code in hardware [18].

### 2.3.2 Comparison between Microprocessors, FPGAs and ASICs

Another type of integrated circuit is the ASICs, a dedicated chip designed to perform a specific set of tasks. The advantages of ASICs over microprocessors are faster and much more efficient. Instead of fetching and following software instructions to complete a task, ASICs have predefined circuitry to complete the task. FPGAs provide a low-cost alternative to dedicated ASIC chips [17]. A design can be customized for a specific purpose and then programmed onto the FPGA, unlike an ASIC where the design is "hardware" and cannot be changed.

Progressing from the microcontroller to FPGA to ASIC, each gets more application-specific, therefore, more efficient [17]. A microcontroller can have many peripherals, such as an ADC and voltage comparators, allowing them to interact with other analog devices. These peripherals and software flexibility allows one kind of microcontroller to be used in many applications. An FPGA, on the other hand, consists of simple logic elements, which can be combined to implement the peripherals previously mentioned, but only interact with most other digital electronics [17]. With an ASIC, the logic elements of an FPGA are replaced with actual logic gates made of transistors, which are sometimes manually laid out. They represent the most optimized design and can be combined with analog electronics. Figure 2.3 shows the summary for the comparison between ASIC, Microprocessors and FPGA.



| | Processor | SRAM FPGA | Flash FPGA | ASIC |
|---|---|---|---|---|
| Speed (parallel) | Low | High | Medium | **Highest** |
| Cost for custom design | **Lowest** | Medium | Medium | High |
| Development time | **Low** | Medium | Medium | High |
| Power usage | Medium | Medium | **Low** | Lowest |

Figure 2. 3 Comparison of Technologies[18]

## 2.4 CanSat Nanosatellite

### 2.4.1 CanSat Concept

Prof. Robert Twiggs of Stanford University [19] first proposed the CanSat concept in 1999, demonstrating that the device's electronics could be designed to fit within a soda can.

A CanSat can be designed with commercial components that can be implemented to achieve its mission. For instance, the Terminal Node Controller (TNC) can be a commercial microcontroller. The ground station, a laptop computer with a connected antenna, acts as a transmitter for the up-link input signals to the TNC. On the other hand, the TNC also prepares the output data collected by the onboard sensors to be sent and downlinked by telemetry [2].

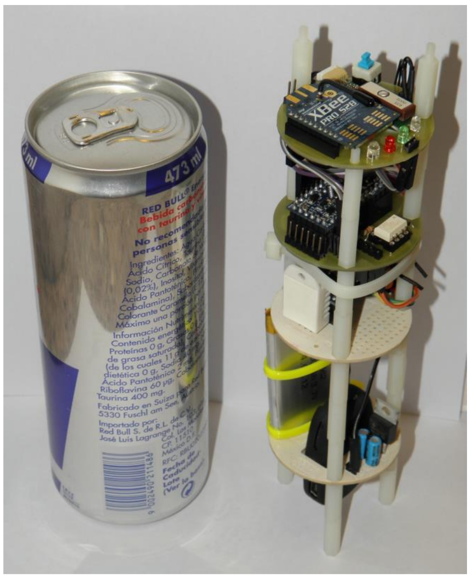


Figure 2. 4 Cansat[2]

### 2.4.2 CANSAT Design

In CanSat design, the microprocessor and power supply are the main elements used in every CanSat. The microprocessor is the robot's brain. It is in charge of receiving signals from external sensors and processing them so that they operate properly. The majority of microprocessors have or may have an internal memory for data storage, which is important for storing data from multiple sensors during flight. The Arduino Microcontroller, Mbed Microcontroller, and Raspberry Pi are common microprocessors that are used [19][2][20][21]. The power supply is critical for any robot or electronic system since it provides electricity for all robot systems to operate. Lithium polymer batteries are the most often utilized because of their performance and current-weight ratio (LiPo). Aside from the elements described, more may be added by the task with which it has been assigned, such as using a temperature sensor, GPS, accelerometer, and camera.

### 2.4.3 Type of CANSAT

There are mainly three types of CanSat, Telemetry CanSat, Comeback CanSat and Open Class CanSat. The major goal of Telemetry CanSat is to gather and send data from the flight and weather conditions in real-time to a ground station for processing. [20]. Because its objective is to collect data rather than land at a specific location, CanSat does not use a steering mechanism in this category. A barometric sensor, temperature and humidity sensor, GPS, and camera are the most frequent components utilized in the systems detailed in the previous sections [19].

Meanwhile, CanSat's main purpose is to land as close to a target indicated by GPS coordinates as possible in order to make a comeback. GPS or an Inertial Navigation System (INS) can guide these devices [21]. This information is sent to the microprocessor, which analyzes the target's location to process data to determine the angle at which it should turn to approach the target and delivers the necessary commands to the steering system. CanSat relies on two primary types of satellites: those with a parachute or glider, and those with a rotor and wings.

CanSat with a parachute or glider commonly has a steering system made up of asymmetrically moving threads that generate a difference in longitudinal axis lift, causing the CanSat to rotate in one direction or the other.  It makes use of rather simple mechanics. These devices are difficult to handle due to the slow rate of fall and the large surface area that elevates them [22]. Meanwhile, unlike CanSat with parachutes or gliders, CanSat has a rotor and wings, which are mechanically more sophisticated and less subject to weather conditions [23]. Because of their faster fall rate, these devices are significantly more difficult to manage and necessitate an electronic system capable of performing many more corrections per second.

Finally, an Open Class CanSat is any robot that does not fit into either of the preceding two categories and can be submitted. The majority of CanSat in this category are robots that are testing new technologies or designs that have never been tested before (technology demonstrators)[24].

## 2.5 Comparison between different approaches of CanSat

Table 2. 1 Comparison between different approaches

| The study | Title | Method | Results |
|---|---|---|---|
| M. Ostaszewski, K. Dzierżek and L. Magnuszewski (2018) | "Analysis of data collected while CanSat mission"[25] | Microcontroller with 2.4GHz transceiver STM32F103C8T6<br>The probe's main job is to transmit and retain data collected by sensors throughout the mission (Telemetry CanSat).<br>The ground control station, which uses a rocket as a launcher, has a laptop attached to a receiving module. | Almost all of CanSat's modules were functional. The GPS readings are simply too irregular to trust the data from the GPS during the flight. It may be beneficial to locate CanSat's landing zone after landing. |
| M. E. Umit, M. Tetlow, W. Cabanas, H. Akiyama, S. Yamaura, S. Olaleye (2011) | "Development of a fly-back CANSAT in 3 weeks"[21] | The microcontroller Mbed is the main onboard avionic. It has a microprocessor that runs at 100 MHz.<br>Use a rocket as the primary launch vehicle and a balloon for testing. Comeback CanSat mission – requires a complex algorithm to navigate<br>The ground station is communicated using an XBee pro modem. This modem is used to process all sensor data, bundle it in Mbed, and send it to the ground station. | The CanSat's first iteration was made of aluminum. However, the glider's weight was too much for it to raise itself.<br>Due to the earthquake, tests with the final design of the CanSat could not be performed. On March 11, 2011, something happened in Japan. |
| R. P. Ramadhan, A. R. Ramadhan, S. A. Putri, M. I. C. Latukolan, Edwa, Kusmadi (2019) | "Prototype of CanSat with Auto-gyro Payload for Small Satellite Education" [26] | The main unit is an Arduino Nano Microcontroller.<br>At an altitude of 670 to 725 meters, the CanSat payload will measure atmospheric parameters.<br>CanSat mission for telemetry<br>The sensor data will be collected and transferred to the Ground Control Station through XBEEPRO S1 (GCS). | When CanSat was launched, the data was gathered. The data from the pressure, altitude, temperature and tilt sensors is displayed in the GUI.<br>All sensors function well, but the descent time is slowed due to vibration. affected the sensors (not properly attached to the header) |

Table 2. 1 Comparison between different approaches (Continue)

| The study | Title | Method | Results |
|---|---|---|---|
| M. Çelebi, S. Ay, M. E. Aydemir, L. H. J. D. K. Fernando, M. K. Ibrahim, M. Bensaada, H Akiyama, S. Yamaura (2011) | **“**Design and Navigation Control of an Advanced Level CANSAT”[23] | Mbed Microcontroller, pressure, and temperature sensors; 3-axis accelerometer, 3-axis gyro, camera, GPS, infrared distance measurement sensor, and RF communication module for communication with the ground station PC [24].<br>Comeback The mission of the CanSat is to launch and land the satellite at a predetermined location.<br>As a launcher, an amateur rocket was used.<br>MATLAB was used to create the ground station software. | Strong winds induce failed recovery, necessitating a simple aerodynamics calculation to assure minimal drift.<br>During firmware development, pressure and GPS sensors are tested successfully.<br>Due to the earthquake that devastated Japan on March 11, 2011, and the resulting tsunami and nuclear leakage, the advanced level CanSat launch was canceled. |
| M. E. Aydemir, R. C. Dursun, M. Pehlevan (2013) | “Ground Station Design Procedures For CanSat”[27] | The core unit is an Arduino Uno microcontroller with GPS and a pressure sensor.<br>For a simple Arduino interface, use the Xbee and the Xbee-Shield.<br>The ground station GUI is designed using the C# programming language and Visual Studio as a development environment. | The device was tested outside, and the data flow was found to be smooth and accurate.<br>For the GUI to start, the serial port must be available. The program looks for all accessible serial ports, the user picks one, and the GUI appears. |
| E. B. Linares, E. A. M. Gonzales, M. H. Cortez, E. A. N. Martinez, J. M. Castillo | “Design of an Advanced Telemetry Mission using CanSat”[20] | As a flight computer, the Arduino Nano is used.<br>Advance telemetry mission - to measure the qualifiers of air pollution in the zone, such as carbon monoxide, methane, benzene, and butane, as well as the fundamental telemetry variables of the pollutants in the air that will be measured. | The CanSat type telemetry based on a Hardware and Software free to monitor pressure variables and temperature is the outcome of the prototype's lone success. |

Table 2. 1 Comparison between different approaches (Continue)

| The study | Title | Method | Results |
|---|---|---|---|
| M. A. Arais, M. E., A. Mohammed, M. E. M, B. Abdelmoteleb, A. Hatem, A.B. Mohamed, A. Ramy, K. Walid, M. E. Fiky, A. Darwish (2015) | "Approaching a nano-satellite using CANSAT systems."[22] | Along with three webcams utilized to live-stream the movies and captures during running time, the Arduino Uno V3, Raspberry Pi v.2, humidity, temperature, pressure, and GPS sensors were employed.<br>The Zigbee protocol and a 3G World-wide-web network make up the communication system.<br>CanSat mission for telemetry<br>The C# desktop application is used to create a ground station.<br>The launcher was a water rocket mechanism. | Humidity, temperature, pressure, GPS, and three cameras all capture and communicate movies, photographs, and sensor information to the ground station via communication protocols.<br>Live streaming of the communication system through the internet and 3G coverage built in the satellite. |
| Angel Colin, Manuel Jimenez-Lizárraga (2015) | "The CanSat Technology For Climate Monitoring In Small Regions At Altitudes Below 1 Km"[2] | The primary unit consists of an Arduino Pro mini 328 microcontroller, a photo/video camera, a 3.37 Volt LiPo battery, an accelerometer, a gyroscope, GPS, and temperature/humidity sensors, as well as an accelerometer, a gyroscope, and GPS.<br>Telemetry for the CanSat mission is transmitted via an XBee antenna mounted on the device's top.<br>The CanSat will be elevated and released using a hexacopter. | The CanSat is released from the center of the balloon's structure by a UAV at heights below 1km.<br>During the parachute drop, the CanSat captures and transmits data to the ground station through telemetry, which is then sent to a laptop for analysis. |

## 2.6 Summary

The project's past, previous, and current work or projects have all been collected in this chapter. The journals in this chapter were used as a guide for this project. They all included similar aerial monitoring systems and sensors to measure relevant parameters such as temperature, accelerometer, altitude, atmospheric pressure, and humidity of the environment. Even though the data processing method is different such as Arduino, Raspberry Pi and Mbed microcontrollers, the implemented concept operation can be used as a reference to improve the creation of aerial surveillance and monitoring applications using the CanSat nanosatellite. Refer to Figure 2.5 shows the summary of the literature review.

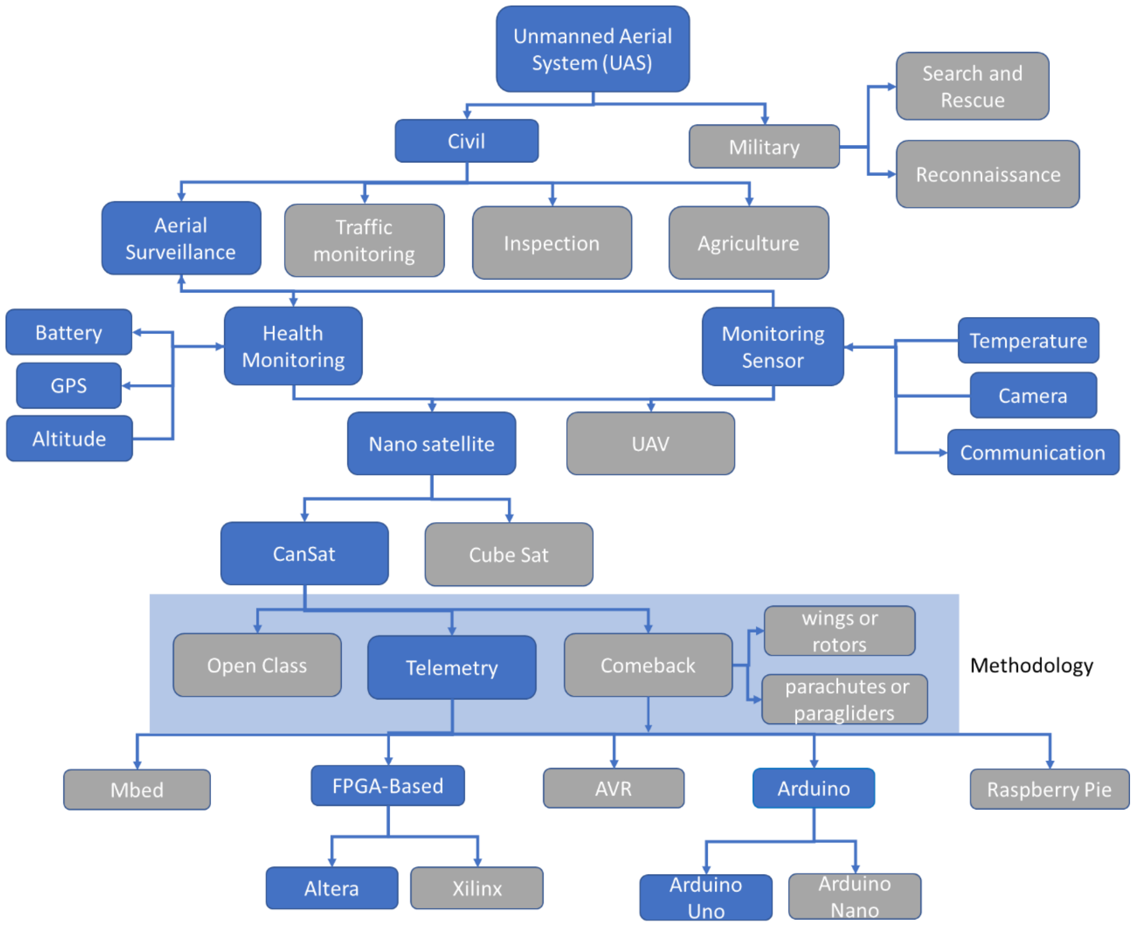


Figure 2. 5 Summary of literature review

# CHAPTER 3

# METHODOLOGY

## 3.1 Introduction

The third chapter will further discuss how the process in this project, especially in choosing the method of solution and design development. The whole CanSat nano-satellite system was divided into three parts: Ground-Station, Communication System, and Satellite body which is the CanSat [2]. Flowcharts are a helpful tool for streamlining procedures. Before beginning any project, the first thing to consider before beginning any project is planning, since the planning process will obtain a positive result. The block diagram, simulation software, hardware component, the flow chart, the connection, code development, the design layout and charges are all included after that. The rationale for each chosen component and hardware will be briefly explained.

## 3.2 Flow of Methodology

Refer to Figure 3.1 below, the project's proposed solution flow. Before starting any project, the Aerial Surveillance system and nanosatellite as the main ideas must be understood first, how they work, and the methods used to perform aerial monitoring and surveillance system on previous studies for references. Then, with the chosen method, the progress starts to design and program the systems, followed by choosing the suitable sensors for the task and objective in this project. The code for both the FPGA DE10-Lite Development board and the Arduino Uno R3 is then developed, and the chosen method is then tested. Lastly, the telemetry data gathered or collected was analyzed.

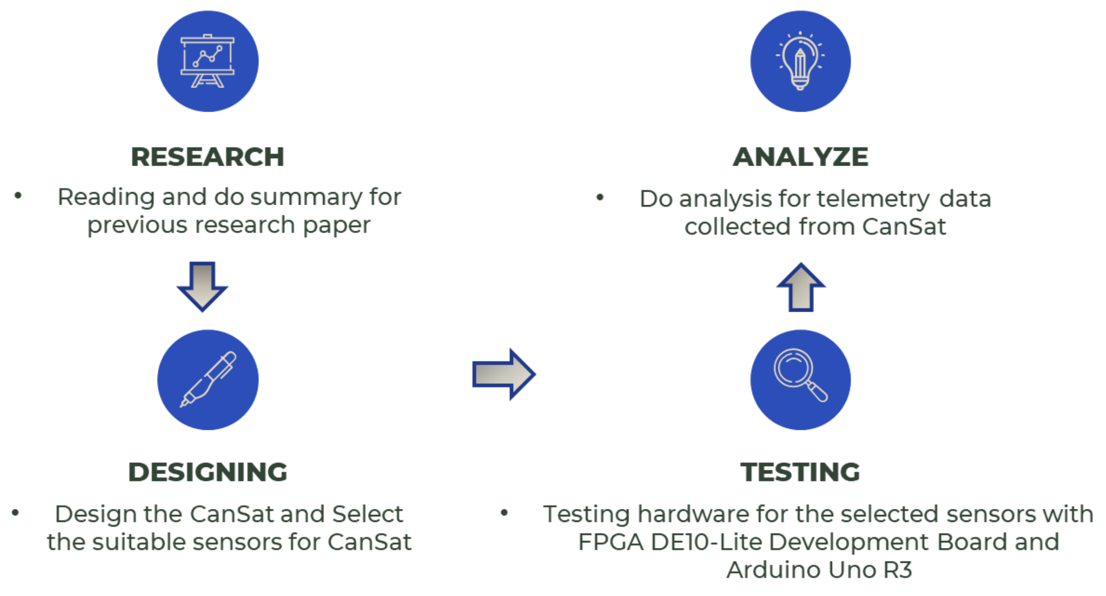

RESEARCH
• Reading and do summary for previous research paper
ANALYZE
• Do analysis for telemetry data collected from CanSat
DESIGNING
• Design the CanSat and Select the suitable sensors for CanSat
TESTING
• Testing hardware for the selected sensors with FPGA DE10-Lite Development Board and Arduino Uno R3

Figure 3. 1 Flow of Methodology

## 3.3 Block Diagram for Input/Output

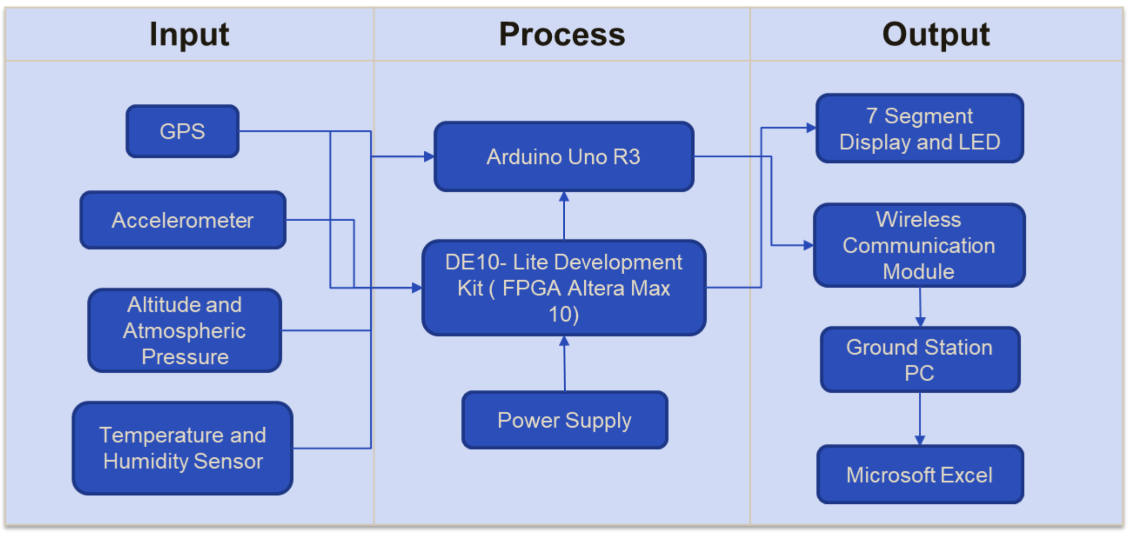

Input
Process
Output
GPS
Accelerometer
Altitude and Atmospheric Pressure
Temperature and Humidity Sensor
Arduino Uno R3
DE10- Lite Development Kit ( FPGA Altera Max 10)
Power Supply
7 Segment Display and LED
Wireless Communication Module
Ground Station PC
Microsoft Excel

Figure 3. 2 Block Diagram

A block diagram shown in Figure 3.2 is a visual illustration of how a project will operate. Readers will get a general idea of how the workflow works from input to procedure to output. The temperature and humidity sensor, altitude and atmospheric pressure, GPS and accelerometer are the types of sensors used in this project's input block

(Hardware List). This sensor will act as the device health monitoring and environmental monitoring system. The output from the sensor will be processed through the FPGA DE10-Lite Development Board and Arduino Uno R3 Microcontroller to determine if all sensors are working correctly. The FPGA DE10-Lite Development Board output will be displayed on the 7 segments and LED meanwhile Arduino Uno R3 Microcontroller output is transmitted to the ground station/PC for the data collected by the sensors through the wireless communication module. After that, the telemetry data that is collected will be converted to Microsoft Excel through serial communication.

## 3.4 Lists of Software

### 3.4.1 Quartus Prime 18.1 Lite Edition

Figure 3. 3 Quartus Prime 18.1 Lite Edition Software Logo

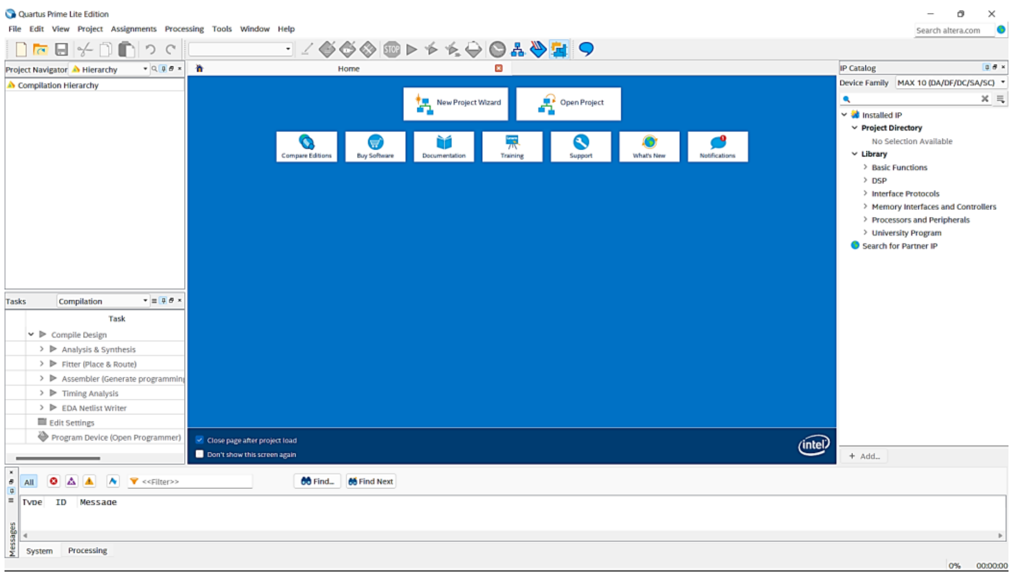


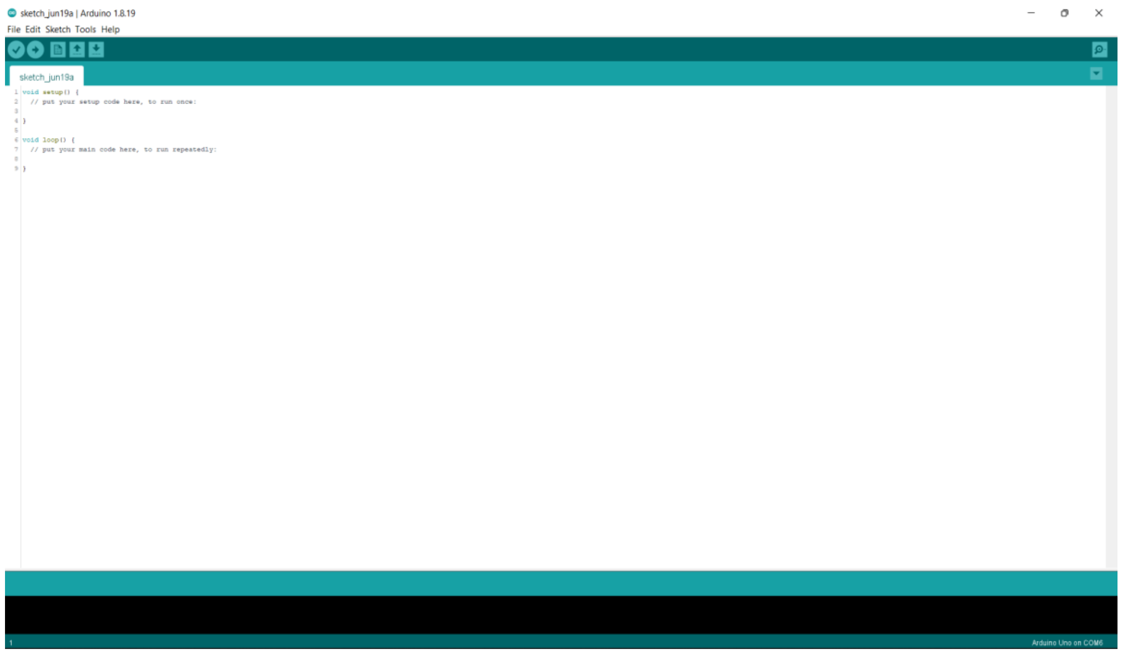
Figure 3. 4    Quartus Prime Design Software Interface[28]

Before Intel acquired Altera, the tool was Altera Quartus Prime, and before that, Altera Quartus II. Quartus Prime allows developers to compile their designs, perform timing analysis, analyze RTL diagrams, simulate a design's reaction to various stimuli, and configure the target device with the programmer. VHDL and Verilog implementations are included in Quartus Prime for hardware description, visual editing of logic circuits, and vector waveform simulation. [29]. The Intel Quartus Prime software for FPGA, CPLD, and SoC designs is groundbreaking in terms of performance and productivity, allowing you to quickly turn a concept into reality. Third-party synthesis tools, static timing analysis, board-level simulation, signal integrity analysis, and formal verification are all supported by the Intel Quartus Prime software [30]. It performs numerous tasks, including the construction of designs using HDL or schematics, the creation of systems using the Platform Designer graphical interface, and the generation and editing of constraints (timing, pin placements, physical location on die, and I/O voltage levels) [30].

It also performs synthesis of a high-level language into an FPGA netlist, FPGA place and route, formally known as fitting, generation of a design image used to program an FPGA, formally known as assembly, timing analysis, and programming of the design image into FPGA hardware, formally known as programming [30].

### 3.4.2 Arduino IDE 1.8.19

Figure 3. 5 Arduino IDE 1.8.19 Software Logo



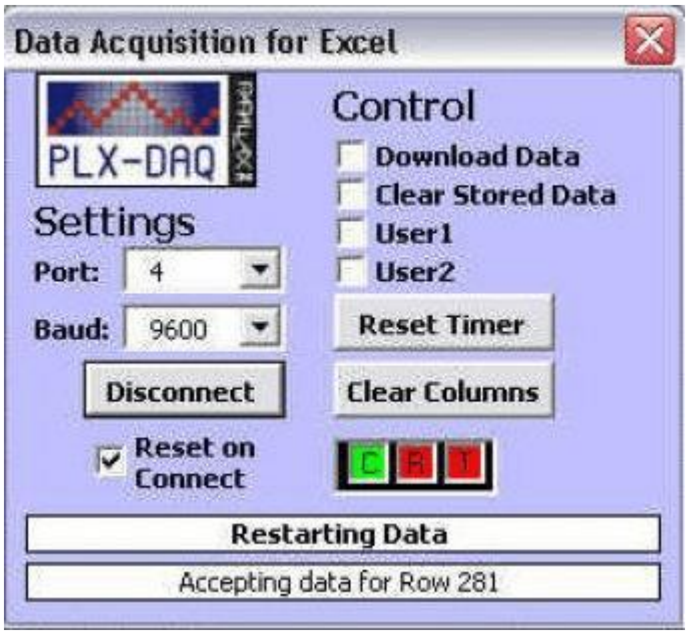
```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Figure 3. 6 Arduino IDE 1.8.19 Software Interface[31]

The IDE (Integrated Development Environment) is an official Arduino.cc software that is primarily used for editing, compiling, and uploading code to the Arduino device. The Arduino IDE is free software that allows to write and compile code for the Arduino Module. Almost all Arduino modules are compatible with this open-source software, which is simple to install and begin compiling code on the go. It is official Arduino software that makes code compilation so simple that even a non-technical person may get their feet wet with the learning process. It operates on the Java Platform and is compatible with operating systems such as MAC, Windows, and Linux [31]. Besides, it includes built-in functions and commands that aid in debugging, modifying, and compiling code in the environment.

Arduino Uno, Arduino Mega, Arduino Leonardo, Arduino Micro, and many other Arduino modules are available [31]. On the board of each of them is a microcontroller that has been programmed and accepts data in the form of code. The core code, also known as a sketch, written on the IDE platform will eventually generate a Hex File, which will be copied and uploaded to the board's controller [31]. The IDE environment is made up of two parts: an editor and a compiler. The editor is used to write the required code, while the compiler is used to compile and upload the code to the Arduino Module. Both C and C++ languages are supported in this environment [31].

### 3.4.3 Parallax Data Acquisition tool (PLX-DAQ) Software



Port: 4
Baud: 9600
Reset on Connect ✓
Restarting Data
Accepting data for Row 281

Figure 3. 7 PLX-DAQ Software Interface[32]

The Parallax Data Acquisition Tool (PLX-DAQ) software add-in for Microsoft Excel collects up to 26 channels of data from any Parallax microcontroller and sorts it into columns [32]. PLX-DAQ allows for simple spreadsheet analysis of field data, laboratory sensor analysis, and real-time equipment monitoring. Any microcontroller linked to a sensor and connected to a PC's serial connection can now send data directly to Excel [32]. PLX-DAQ includes the following features: plot or graph data in real-time using Microsoft Excel, record up to 26 columns of data, and more [32]. Read/write any cell on a worksheet, read/set any of four checkboxes on the control the interface, Baud rates up to 128K, and supports Com1-15[32].

## 3.5 Lists of Component

### 3.5.1 DE10-Lite Development Board (Altera FPGA MAX10)

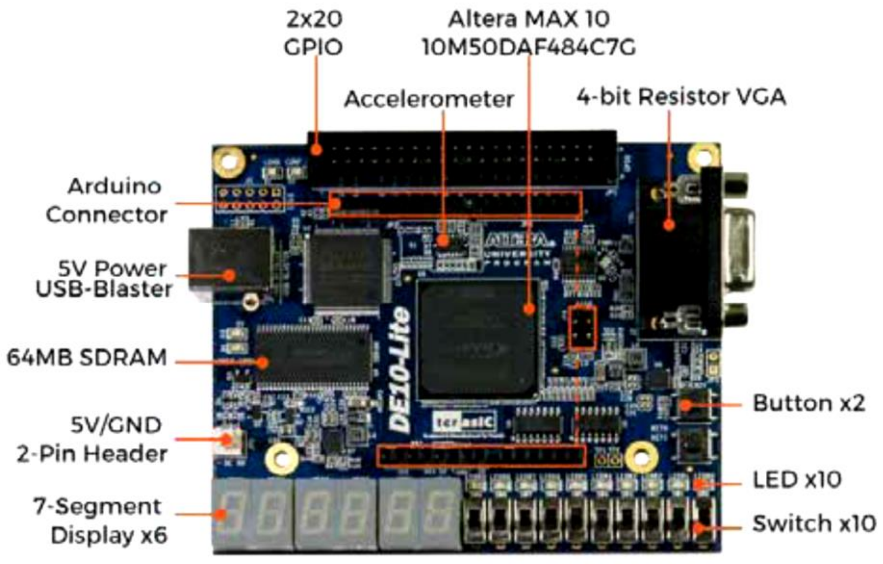


Figure 3. 8 DE10-Lite Development Board

Based on the Altera MAX 10 FPGA, the DE10-Lite is a reliable hardware design platform. The MAX 10 FPGA is well-suited for low-cost, single-chip control plane or data path applications, as well as industry-leading programmable logic for maximum design flexibility. The MAX 10 FPGA can reduce power consumption and costs while increasing performance. [33]. Protocol bridging, motor control drive, analog-to-digital conversion, image processing, and handheld devices are all good candidates for the MAX 10 FPGA.

The DE10-Lite development board includes hardware such as an onboard USB Blaster, a 3-axis accelerometer, video support, and more. Exploiting all of these features is ideal for demonstrating, assessing, and prototyping the Altera MAX 10 FPGA's true potential and is sufficient for this project [33]. The configuration of the board is shown in Figure 3.8, along with the positioning of the connectors and critical components. This board has several features that allow users to build a variety of designed circuits, ranging from simple circuits to various multimedia applications.

Dual ADCs are included in the FPGA device. One dedicated analog input, eight dual function pins, 50K programmable logic elements, 1,638 Kbits of M9K memory, 5,888 Kbits of user flash memory, 144 18 18 multipliers, and four PLLs are all included in each ADC. The board includes a USB blaster with a standard type B USB connector. The board also has a memory device which is 64MB SDRAM, x16 bits data bus. Besides, the board consists of a 2x20 GPIO Header and Arduino Uno R3 Connector, including six ADC channels. It also has 10 LEDs, ten slide switches, two pushbuttons with debounced and 6 seven-segments. The board's power supply is a 5V DC input from a USB or an external power socket [33].

### 3.5.2 Arduino Uno R3 Microcontroller

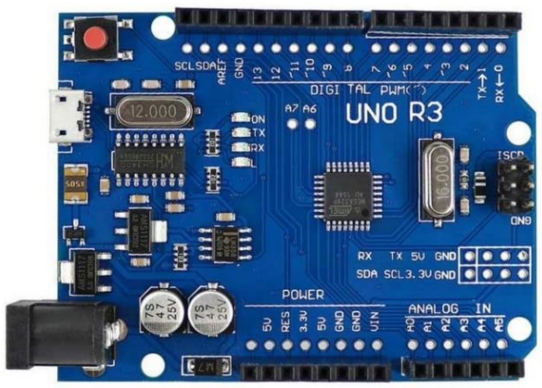


Figure 3. 9 Arduino Uno R3 Microcontroller Board[34]

The Arduino Uno is an ATmega328-based microcontroller board. A 16 MHz resonator, a USB connection, a power jack, an in-circuit system programming (ICSP) header, and a reset button are among the 20 digital input/output pins (of which 6 can be used as PWM outputs and 6 can be used as analog inputs) [34].

The Uno is unique in that it does not employ the FTDI USB-to-serial driver chip found on previous boards. Instead, it uses an ATmega16U2 that has been designed to function as a USB-to-serial converter [34]. The USB bootloader on this supplementary microcontroller allows advanced users to modify it [34]. The Arduino Uno R3 is the third and most recent iteration.

### 3.5.3 DHT22 Temperature and Humidity Sensor

Figure 3. 10 DHT22 Temperature and Humidity Sensor[35]

The DHT22 is the more expensive version when compared to the DHT11, but it has better specifications. It has a temperature range of -40 to +125°C with +-0.5°C accuracy [35], whereas the DHT11 has a temperature range of 0 to 50°C with +-2°C accuracy [36]. In addition, the DHT22 sensor has a wider humidity measuring range, ranging from 0 to 100% with 2-5% accuracy [35], compared to 20 to 80% with 5% accuracy for the DHT11 [36].

DHT22 is a low-cost humidity and temperature measuring sensor. The fact that it can get data in two seconds is a feature that makes it more valuable than other sensors. It is divided into two sections. One is used to measure temperature, while the other is used to measure humidity. It also has an IC that allows data to be sent to a microcontroller [35]. For humidity measurement, it uses the humidity measurement component, which consists of two electrodes separated by a moisture-holding substrate [35]. Meanwhile, this sensor uses an NTC temperature sensor or a Thermistor to measure temperature [35].

### 3.5.4 The built-in accelerometer in DE10-Lite Development Board (ADXL345)

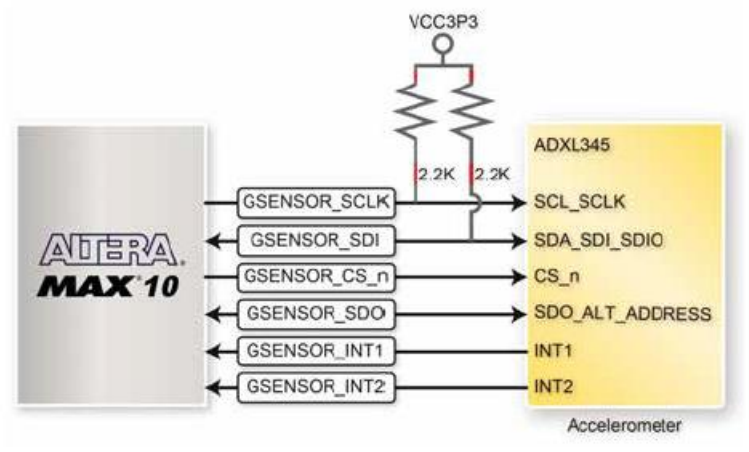


Figure 3. 11 Connection between the accelerometer sensor and FPGA

The ADXL345 is a three-axis accelerometer with a high resolution (13-bit) that can measure up to 16g in a small, thin, ultralow-power package. [37]. The 16-bit two's complement digital output data is encoded and can be accessed through an I2C or SPI (3- or 4-wire) interface [37]. For mobile device applications, the ADXL345 is a great option. Tilt-sensing applications monitor the performance of both gravity's static acceleration and the dynamic acceleration produced by motion or shock. It can measure inclination fluctuations of less than 1.0° due to its high resolution (3.9 mg/LSB) [37].

The ADXL345 is in I2C mode when the GSENSOR_CS_N signal is high. The device's 7-bit I2C address is 0x1D, followed by the R/W bit when the GSENSOR_SDO signal is high [33]. This corresponds to a write of 0x3A and a read of 0x3B. Give a low signal to the GSENSOR_SDO will select an alternate I2C address of 0x53 (followed by the R/W bit). This corresponds to a write of 0xA6 and a read of 0xA7 [37].

### 3.5.5 GPS NEO-7M Module.

Figure 3. 12 GPS NEO-7M Module[38]

The GPS-NEO-7M module is a high-performance GPS module with an active antenna that enables UART communication. Any microcontroller may simply interface with it. This module comes with a rechargeable battery and can be directly connected to a computer using a USB to TTL converter [38].

This module can receive data and calculate the geographical position quickly and accurately. The module contains internal memory to save settings and supports BeiDou, Galileo, GLONASS, and GPS / QZSS [38].

### 3.5.6 BMP280 Barometric Pressure Sensor

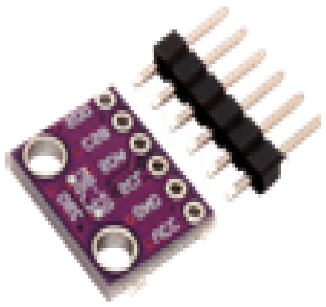


Figure 3. 13 BMP280 Barometric Pressure Sensor [39]

Bosch Sensortec's BMP280 is a pressure, humidity, temperature, and approximate altitude sensor. It is best suited for environmental applications, but it can also be utilized in Prosthetics applications where pressure is a key parameter to work with. It's also useful in drones, where pressure, temperature, and altitude may be monitored and used to make further observations. The BMP280 is made up of a Piezo-resistive sensing element that is connected to an A/D converter [39]. This converter delivers conversion results with appropriate sensor compensation, as well as a digital interface and a built-in IIR filter to reduce output data disruptions. [39].

I2C and SPI are the two protocols that can be used to connect to the BMP280. It can monitor pressure with a 1% accuracy from 300 to 1100 hPa, temperature with a 1% accuracy from -40 to +85°C, and approximate altitude with some calculations [39]. The sensor's operational voltage ranges from 1.2V to 3.6V with current consumption of 0.1uA in sleep mode, making it ideal for low-power applications in embedded devices [39].

### 3.5.7 RF transceiver module (2.4GHz, NRF24L01+ with PA and LNA)

Figure 3. 14 RF NRF24L01+ transceiver module[40]

This module is based on the Nordic nRF24L01+, which has a 1,000-meter range and contains a built-in Power Amplifier (PA) and Low-Noise Amplifier (LNA) [40]. This module can use 125 different channels which gives a possibility to have a network of 125 independently working modems in one place. Each channel can have up to 6 addresses, or each unit can communicate with up to 6 other units at the same time. Keep an 8-pin interface on this module and use the SPI interface to control it while transferring data [41][40]. Three of these pins have SPI communication and they need to be connected to the SPI pins.

The nRF24L01+ module transmits data using GFSK modulation and operates in the 2.4 GHz global ISM frequency range. 250kbps, 1Mbps, and 2Mbps data transfer rates are all available [40]. All settings, including frequency channel (125 selectable channels), output power (0 dBm, -6 dBm, -12 dBm, or -18 dBm), and data rate, can be configured using the SPI interface (250kbps, 1Mbps, or 2Mbps).

## 3.6 Flowchart of the system

A flowchart is a diagram that displays the flow of information, steps and decisions that must be made to complete a procedure. There will be two flowcharts for this project: the deployment flowchart and the testing process flowchart.

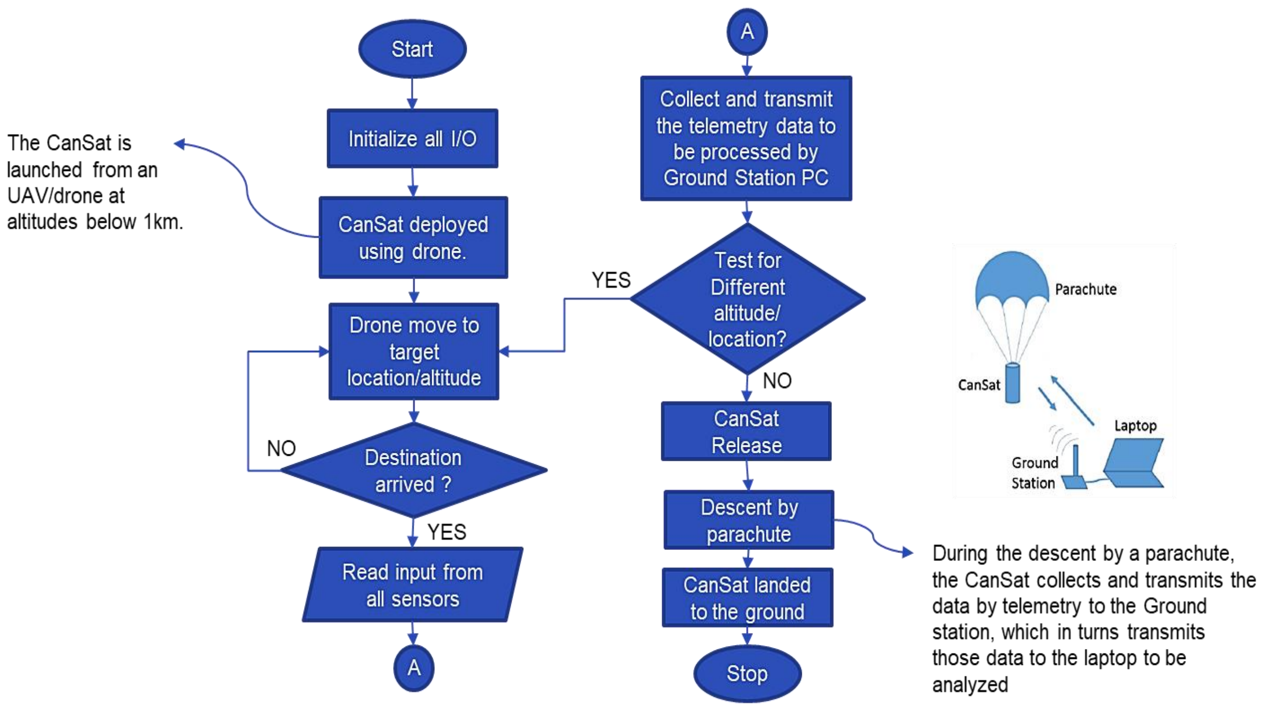
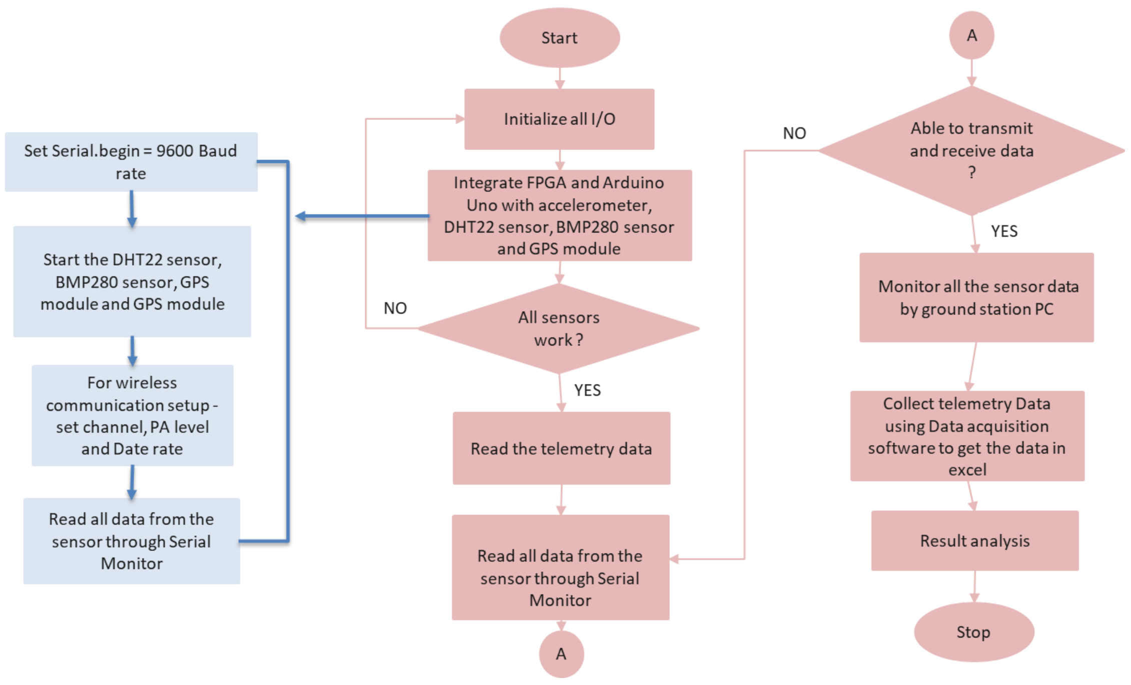

Start
Initialize all I/O
The CanSat is launched from an UAV/drone at altitudes below 1km.
CanSat deployed using drone.
Drone move to target location/altitude
NO
Destination arrived ?
YES
Read input from all sensors
A
A
Collect and transmit the telemetry data to be processed by Ground Station PC
Test for Different altitude/ location?
YES
NO
CanSat Release
Descent by parachute
CanSat landed to the ground
Stop
Parachute
CanSat
Laptop
Ground Station
During the descent by a parachute, the CanSat collects and transmits the data by telemetry to the Ground station, which in turns transmits those data to the laptop to be analyzed

Figure 3. 15 Deployment flowchart

A flowchart is a diagram that depicts the movement of data. Figure 3.11 depicts the deployment of CanSat in this project. The flow starts and all sensors read the initiate all I/O commands. The CanSat will be deployed using a drone and moved to the target locations or altitude for this project for the next flow. If the drone arrived at the target locations, the CanSat would start reading the input from all the sensors. The collected data will transmit the telemetry data to be processed by the Ground station PC. The CanSat will move to other locations or altitudes to collect more data and compare it to another. If the CanSat collects enough data, the CanSat will descend to the ground. During the descent by a parachute, the CanSat collects and transmits data by telemetry to the ground station, transmitting those data to the laptop to be analyzed. After the whole phase has been completed, the flow will end.



Start
Initialize all I/O
Integrate FPGA and Arduino Uno with accelerometer, DHT22 sensor, BMP280 sensor and GPS module
All sensors work ?
NO
YES
Read the telemetry data
Read all data from the sensor through Serial Monitor
A
Set Serial.begin = 9600 Baud rate
Start the DHT22 sensor, BMP280 sensor, GPS module and GPS module
For wireless communication setup - set channel, PA level and Date rate
Read all data from the sensor through Serial Monitor
A
Able to transmit and receive data ?
NO
YES
Monitor all the sensor data by ground station PC
Collect telemetry Data using Data acquisition software to get the data in excel
Result analysis
Stop

Figure 3. 16 System flowchart

Refer Figure 3.12 depicts the deployment of CanSat in this project. The flow begins with initializing all input and output, and the system design integrates the DE10-Lite development board and Arduino Uno with all the sensors (barometric sensor, temperature and humidity sensor, GPS and accelerometer). The serial baud rate is set to 9600 and starts with all the sensors. Also, set up the channel, PA level and data rate for the wireless communication module. In the decision stage, the sensor's output will be processed by a DE10-Lite Development board, testing the output data gathered from the sensors.

The flow will return to system design if the sensors are not working correctly. If yes, the process will proceed to the next flow which is to read the telemetry data through serial communication and test the system's capability in wireless communication. When the system can transmit and receive the data, the next flow monitors all the data gathered and processes it by ground station to the laptop. The next step of the procedure is that with all the data collected from the CanSat, the ground station will process the data to the laptop using data acquisition software to receive the data in excel format. The telemetry data then will be analyzed and compared to the expected results. After the whole phase has been completed, the flow will end.

## 3.7 Design Development

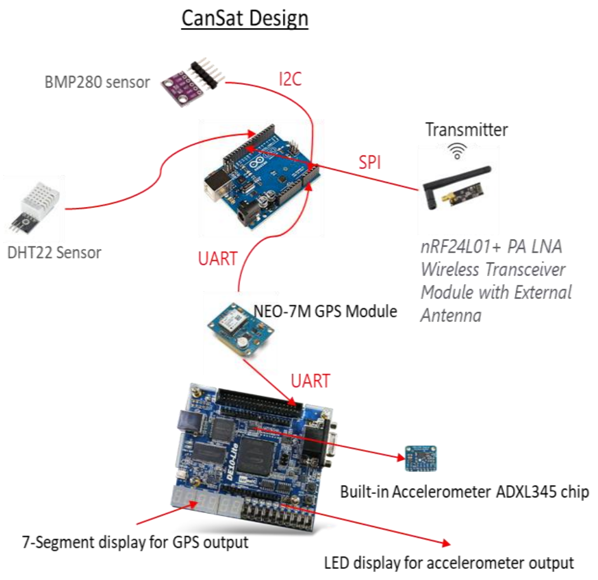

CanSat Design
BMP280 sensor
I2C
Transmitter
SPI
DHT22 Sensor
UART
nRF24L01+ PA LNA Wireless Transceiver Module with External Antenna
NEO-7M GPS Module
UART
Built-in Accelerometer ADXL345 chip
7-Segment display for GPS output
LED display for accelerometer output

Figure 3. 17 CanSat Circuit Design

Figure 3.17 shows the circuit design for the CanSat. For the BMP280 sensor connection, the I2C protocol involves using two lines to send and receive data: a serial clock pin (SCL) that the Arduino Controller board pulses at a regular interval, and a serial data pin (SDA) over which data is sent between the two devices. Next, the NEO-7M GPS module uses a UART interface for the data transfer on the both DE10-Lite Development Board and Arduino Uno R3 Microcontroller. UART stands for Universal Asynchronous Receiver/Transmitter. It is a hardware device (or circuit) used for serial communication between two devices. The nRF24L01+ communicates over a 4-pin SPI (Serial Peripheral Interface) with a maximum data rate of 10Mbps. The SPI bus uses the concept of a master and a slave. In this project, the Arduino is the master and the nRF24L01+ module is the slave. Unlike the I2C bus, the SPI bus has a limited number of slaves. The details pinout for each connection will be brief in the next subtopic.

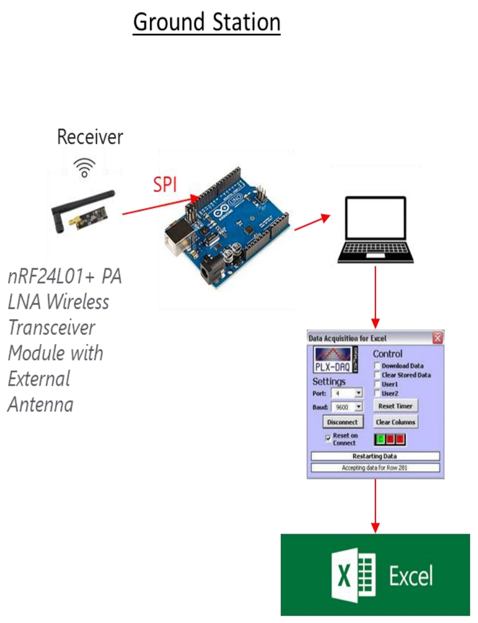


Figure 3. 18 Circuit Design on Ground Station

Figure 3.18 shows the circuit design for the Ground station. The nRF24L01+ communicates over a 4-pin SPI (Serial Peripheral Interface) with a maximum data rate of 10Mbps. The SPI bus uses the concept of a master and a slave. In this project, the Arduino is the master and the nRF24L01+ module is the slave. Unlike the I2C bus, the SPI bus has a limited number of slaves. After that, the telemetry data are collected through serial communication and converted to Microsoft Excel using PLX-DAQ Data Acquisition Software to easily analyze the data.

### 3.7.1 FPGA DE10-lite Development Board Connection

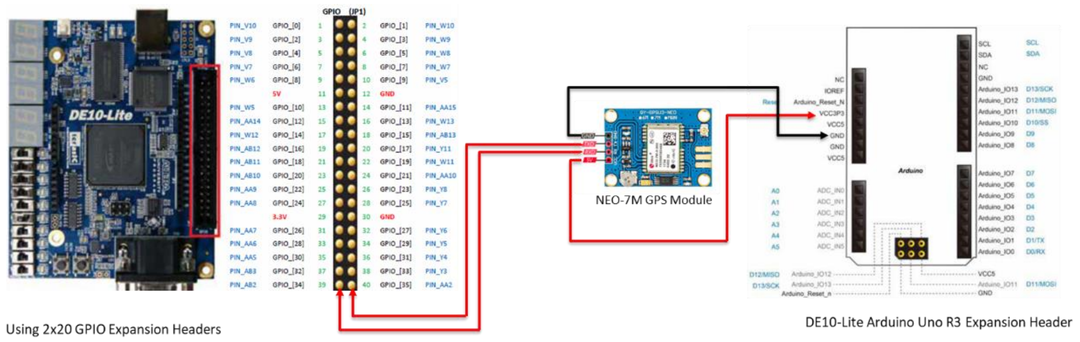

GPIO (JP1)
DE10-Lite
NEO-7M GPS Module
Arduino
Using 2x20 GPIO Expansion Headers
DE10-Lite Arduino Uno R3 Expansion Header

Figure 3. 19 DE10-Lite Development Board circuit connection

Refer to Figure 3.19 shows the connection between the DE10-Lite Development Board and GPS NEO-7M. The VCC and GND of the GPS module are connected to the 3.3V and GND of the DE10-Lite Development Board. The Tx pin on the GPS module is connected to the GPIO_[35] and the Rx pin is connected to the GPIO_[34]. The NMEA protocol and the UART interface were combined for this project, and just one NMEA message, GPGLL (Geographic Position Latitude/Longitude), was processed. The most widely used text-based data transmission protocol is NMEA (8-bit ASCII cricking). Only six seven-segment indicators were utilized on the board, three of which represented the latitude coordinate and the other three the longitude coordinate. NMEA 0183 is a standard defining a text-based communication protocol for navigation equipment, especially popular in GPS receiver modules. NMEA 0183 message format:

1. “$" - message start marker
2. A 5-letter message identifier that identifies the type of message being transmitted:

   i. The first two letters are the identifier of the source of the message, in the case of the GPS module it is always "GP"

   ii. Last three letters - message type

3. A list of message data. The amount and format of this data depend on the message, but the following characteristics are required for the information transmitted:

   i. All data is separated by commas - ","

   ii. If there is no data in the message, commas are still put (possible situations like ",,,") - the number of commas in a certain type of message is constant and does not depend on the data, this is done to simplify the process of processing the message

4. "*" - checksum start marker

5. 2 checksum symbols - the checksum is counted as the XOR sum of all symbols between "$" and "*" not inclusive. The resulting amount is represented in hexadecimal form in two ASCII characters (letters are transmitted in uppercase)

6. "<CR> <LF>" - 2 characters marking the end of the message:

   i. <CR> - ASCII carriage return, hexadecimal value - 0D

   ii. < LF> - ASCII line feed, hexadecimal value - 0A

### 3.7.2 Development Of a Message Processing Module from A GPS Module

To process messages from the GPS module, a simple state machine was developed.

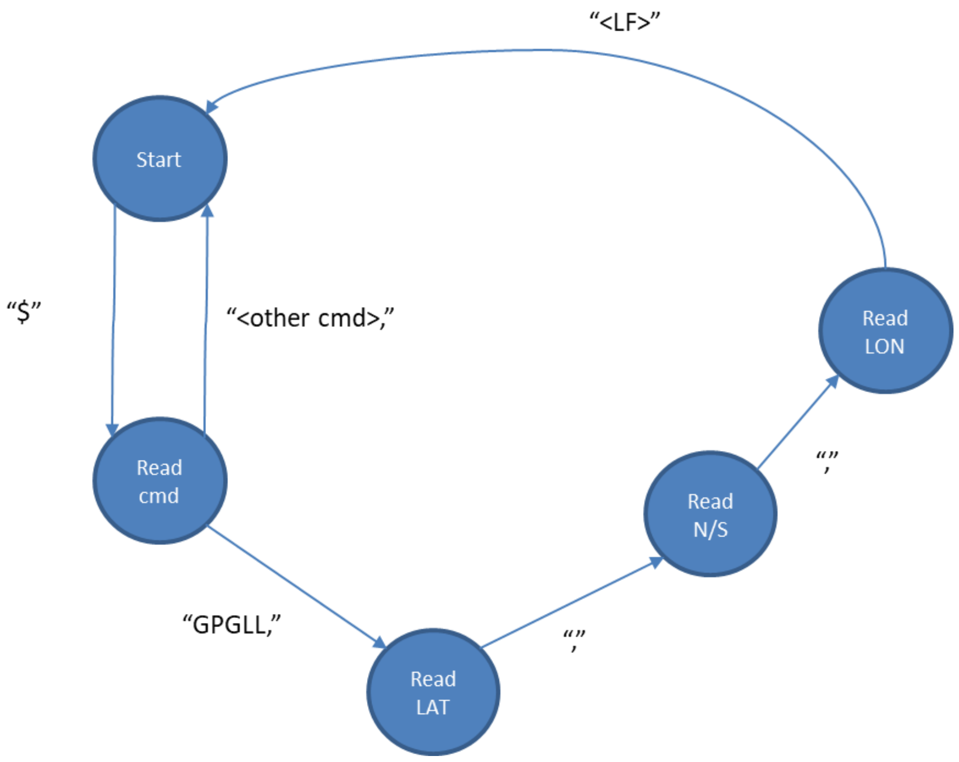


Figure 3. 20 State Diagram for Message Processing Module

Starting state (rest state) - the machine waits for a signal from the UART module, reporting that the data has been received and upon the arrival of this signal checks the data, if the "$" has come - goes into the following state:

```
0: begin //Search for '$'
    if(rxReady && rxData=="$")begin// '$': Start of frame
        STATE<=STATE+1;
    end
end
```

The state saves the data received from the UART to an NMEA array [23:0] until it encounters ",". When the machine encounters a comma, it checks the NMEA condition == "GLL", i.e., whether the necessary command has arrived and if the condition is met, it goes to the next state. If another command came from the GPS module, the machine returns to its starting state. only the last three bytes received from the UART are stored in the NMEA array because they determine the message type, and the rest of the information is not required.

```
1: begin //Check Tag, if not the proper tag go back
    if(rxReady) begin
        if(rxData==",")begin
            if(NMEA=="GLL")begin
                STATE<=STATE+1;
            end
            else begin
                STATE<=0;
            end
        end
        else begin //Grab the NMEA Message Type
            NMEA[23:8]<=NMEA[15:0];
            NMEA[7:0]<=rxData; //Grab the tag info
        end
    end
end
```

This state processes the latitude coordinate. It writes the first three ASCII bytes with coordinates to the LAT array [23:0]. As soon as this state is met with a comma character, it moves to the next state.

```
2: begin //Parse the latitude
    if(rxReady) begin
        if(rxData==",")begin
            STATE<=STATE+1;
            prec_cnt = 0;
        end
        else if (prec_cnt < precision) begin //grab only 3 first LAT values, because we have only 3 HEX available
            prec_cnt = prec_cnt + 1;
            LAT[23:8]<=LAT[15:0];
            LAT[7:0]<=rxData;
        end
    end
end
```

In this state, you can handle the north/south latitude value. The board used has only 6 seven-segment indicators, 3 of which displayed the latitude coordinate, and the other 3 - the longitude coordinate - so the values of north/ south latitude and east/ west of longitude were not processed, because there is nowhere to display them. For this case, this state just waited for "," and went into the following state:

```
3: begin //Parse letter
    if(rxReady) begin
        if(rxData==",")begin
            STATE<=STATE+1;
        end
    end
end
```

This state handles the longitude coordinate-the principle of operation is the same as that of the latitude coordinate handler. As soon as there is a line break symbol - there is a transition to the starting state - the data that goes further we are not interested in.

```
4: begin //Parse the longitude
   if(rxReady) begin
      if(rxData==10)begin //LF: Line Feed, end of message, go to idle state
         STATE<=0;
         prec_cnt = 0;
      end
      else if (prec_cnt < precision)begin //grab only 3 first LON values, because we have only 3 HEX available
         prec_cnt = prec_cnt + 1;
         LON[23:8]<=LON[15:0];
         LON[7:0]<=rxData;
      end
   end
end
```

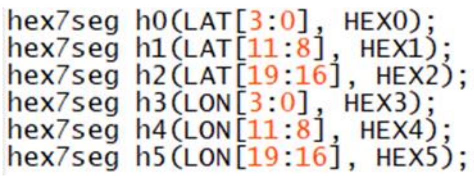
In this case, only the first three bytes of latitude and longitude coordinates were processed for technical reasons, but it is quite simple to increase the accuracy of processing - for this, it is necessary to increase the variable "precision" to the required accuracy value. Increase the dimension of the LAT and LON arrays -you need one byte for each accuracy digit. And fix the filling of these arrays in the states "Read LAT" and "Read LON". The full Verilog code with the module used is attached in Appendix C.

### 3.7.3 Module for displaying information on a seven-segment indicator

To display hexadecimal values on the seven-segment indicator, the "hex7seg" module was used:

```
always @ (hex)
case (hex)
4'h0: display = 7'b1000000;
4'h1: display = 7'b1111001;
4'h2: display = 7'b0100100;
4'h3: display = 7'b0110000;
4'h4: display = 7'b0011001;
4'h5: display = 7'b0010010;
4'h6: display = 7'b0000010;
4'h7: display = 7'b1111000;
4'h8: display = 7'b0000000;
4'h9: display = 7'b0011000;
4'hA: display = 7'b0001000;
4'hb: display = 7'b0000011;
4'hC: display = 7'b1000110;
4'hd: display = 7'b0100001;
4'hE: display = 7'b0000110;
4'hF: display = 7'b0001110;
endcase
```

Therefore, we transmitted only the second half of each byte of the coordinate to the seven-segment indicator, since it was the second half that contained the information necessary for output, and the first half of the byte was always stored 0x3 (in the case when the byte represented a digit):



```
hex7seg h0(LAT[3:0], HEX0);
hex7seg h1(LAT[11:8], HEX1);
hex7seg h2(LAT[19:16], HEX2);
hex7seg h3(LON[3:0], HEX3);
hex7seg h4(LON[11:8], HEX4);
hex7seg h5(LON[19:16], HEX5);
```

### 3.7.4 Built-in DE10-lite Accelerometer Sensor

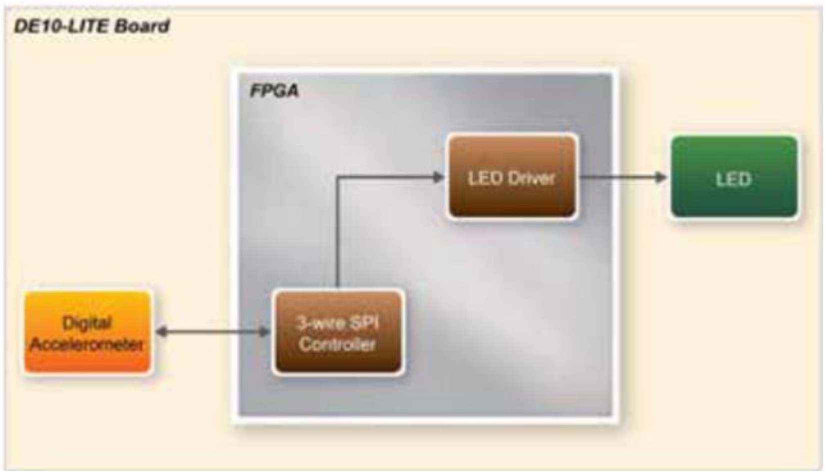


Figure 3. 21 Accelerometer Connection with LEDs

The accelerometer is controlled via a 3-wire SPI in this project. The controller sets 1 on the SPI bit in the Register 0x31 –DATA FORMAT register before reading any data from the accelerometer [33]. To calculate the board's tilt, the 3-wire SPI Controller block reads the digital accelerometer's X-axis value. The LEDs are lit up and float to the top of the board like a bubble. The digital accelerometer detects and shows tilting movement on the LEDs as the board is tilted from left to right and right to left.

### 3.7.5 Arduino Uno R3 Microcontroller connection

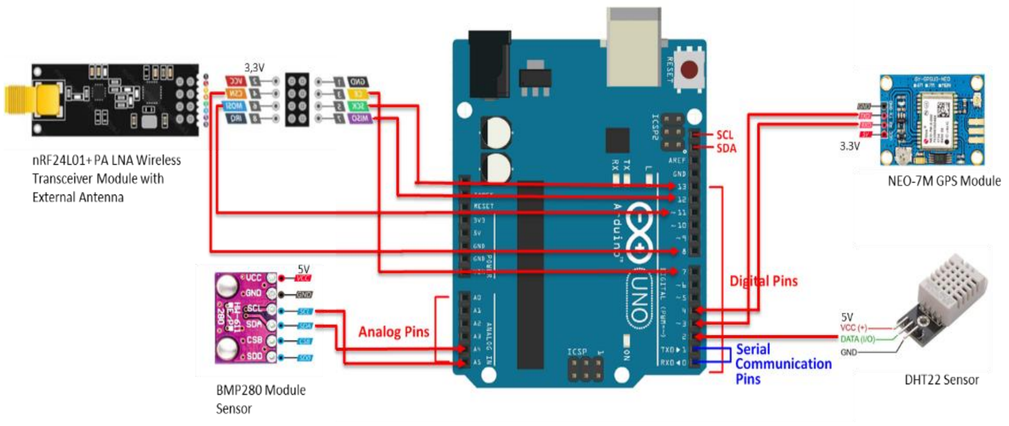


Figure 3. 22 Arduino Uno R3 Circuit on CanSat

Refer to Figure 3.22 shows the circuit connection on the CanSat. For the nRF24L01+ module, the VCC pin on the module is connected to 3.3V, and the GND pin is to ground on the Arduino. The Chip Enable (CE) and Chip Select Not (CSN) are connected to digital pins #7 and #8 respectively. MOSI, MISO, and Serial Clock (SCK) are connected to the SPI pins #11, #12, and #13 respectively. After the data from all the sensors is collected, the data package is sent through the nRF24L01+ module to another nRF24L01+ module on the Ground station.

Furthermore, the VCC pin on the BMP280 module is connected to 5V and the GND pin to the ground on the Arduino. The SDA and SCL pins are connected to I2C pins #A4 and #A5 respectively. Next, for the GPS NEO-7M module, the VCC pin on the module is connected to 3.3V and the GND pin is to ground on the Arduino. The TX and RX pins are connected to digital pins #3 and #4 respectively. Lastly, the VCC pin on the module is connected to 5V and the GND pin to the ground on the Arduino meanwhile the DATA pin is connected to the digital pin #2.

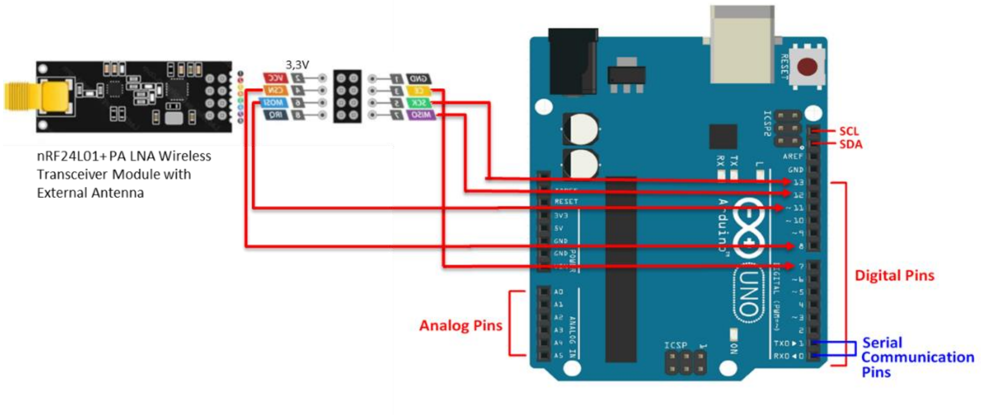

nRF24L01+ PA LNA Wireless Transceiver Module with External Antenna
3,3V
SCL
SDA
Digital Pins
Analog Pins
Serial Communication Pins

Figure 3. 23 Arduino Uno R3 Circuit Connection on Ground Station

Refer to Figure 3.23 shows the circuit connection on the ground station. For the nRF24L01+ module, the VCC pin on the module is connected to 3.3V and the GND pin is to ground on the Arduino. The CE and CSK are connected to digital pins #7 and #8 respectively. MOSI, MISO and SCK are connected to the SPI pins #11, #12 and #13 respectively. Lastly, the data package from the CanSat is received through the nRF24L01+ module.

## 3.8 Chapter Summary

For this chapter, all subtopics were explained briefly, and each chosen rationale was explained. Each component chosen for this project is well studied to ensure it is suitable. A reader can relate to the whole system from this topic's block diagram, flow chart and in-circuit development.

# CHAPTER 4

# RESULTS AND DISCUSSION

## 4.1 Introduction

The planned project outcome of the aerial surveillance and monitoring system devices will be addressed in this section. It will explain the system's working system from the start to the end. The result outcome and discussion for this project were presented in this chapter. Throughout the stage, the result will be implemented based on the project's objective.

## 4.2 Hardware Design

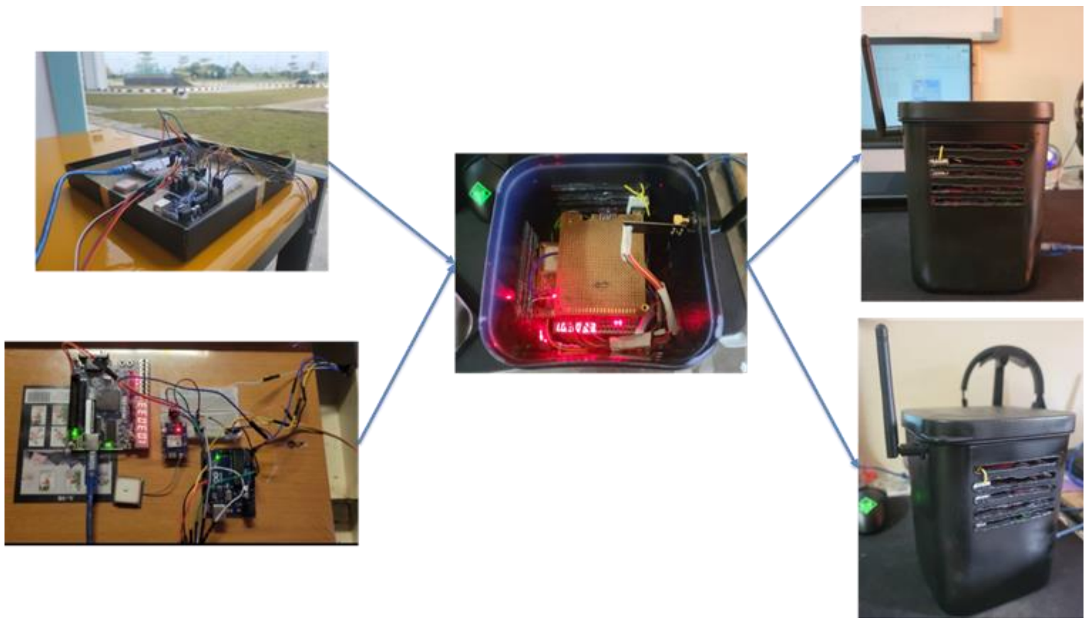


Figure 4. 1 Hardware Design

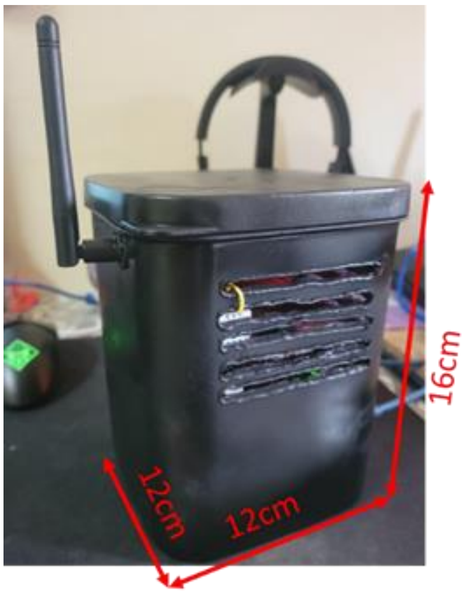


Figure 4. 2 Dimension of the enclosure

Refer to Figure 4.1 depicts the CanSat design, which is made up of a circuit that uses a breadboard and a jumper wire to link all the sensors. Before the circuit is adjusted and put into the shell, the data is obtained simply using the circuit. The objective of this is to ensure that all the sensors and systems are functioning properly. For this project's data collecting and analysis, several sites and altitudes are taken. The DE10-Lite Development Board, Arduino Uno R3 Microcontroller, and all the sensors required in this project are then placed in the plastic cuboid enclosure. The purpose of using a plastic shell is to make the CanSat as light as feasible, as the drone will be carrying the CanSat for an environmental surveillance application in specific areas. As indicated in figure 4.2, the case's dimensions are 12cm x 12cm x 16cm.

## 4.3 Telemetry data collection from different Altitude and locations

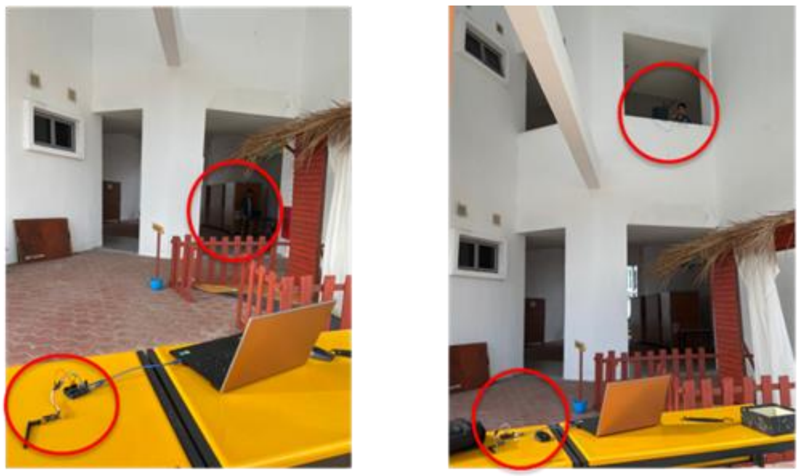


Figure 4. 3 Faculty of Computer 1st and 2nd floor

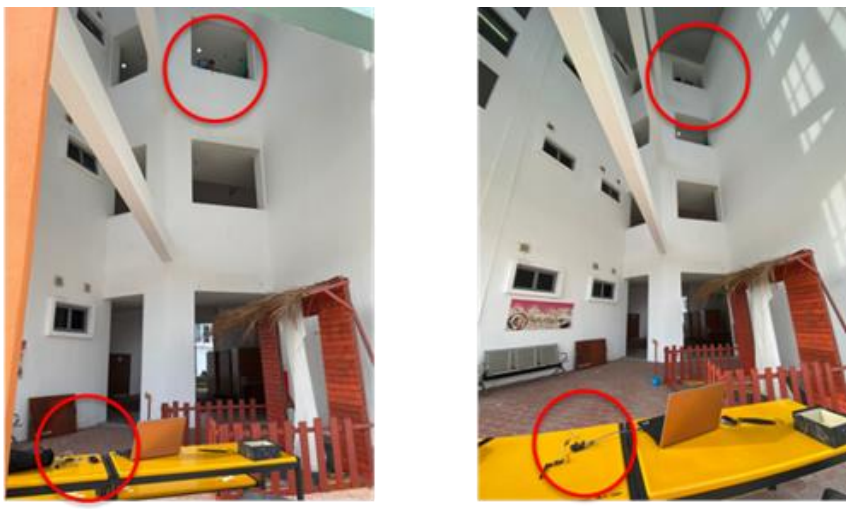


Figure 4. 4 Faculty of Computer 3rd and 4th floor

To test the CanSat whole system, the telemetry data are collected from the different altitudes and locations. For the first test, the data is collected in the Faculty of Computers at Universiti Malaysia Pahang. Refer to Figure 4.3 and Figure 4.4 shows the telemetry data collected from the 1st floor, 2nd floor, 3rd floor, and 4th floor respectively in the Faculty of Computers. Each floor collected 100 rows package of telemetry data. All data collected are converted to Microsoft Excel to make it easy for data analysis. After testing in the open area, data is collected from the indoor area to test the GPS and wireless communication.

## 4.4 GPS reading and Accelerometer Output

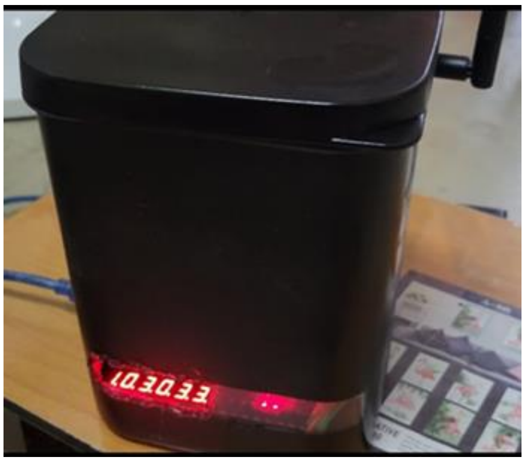
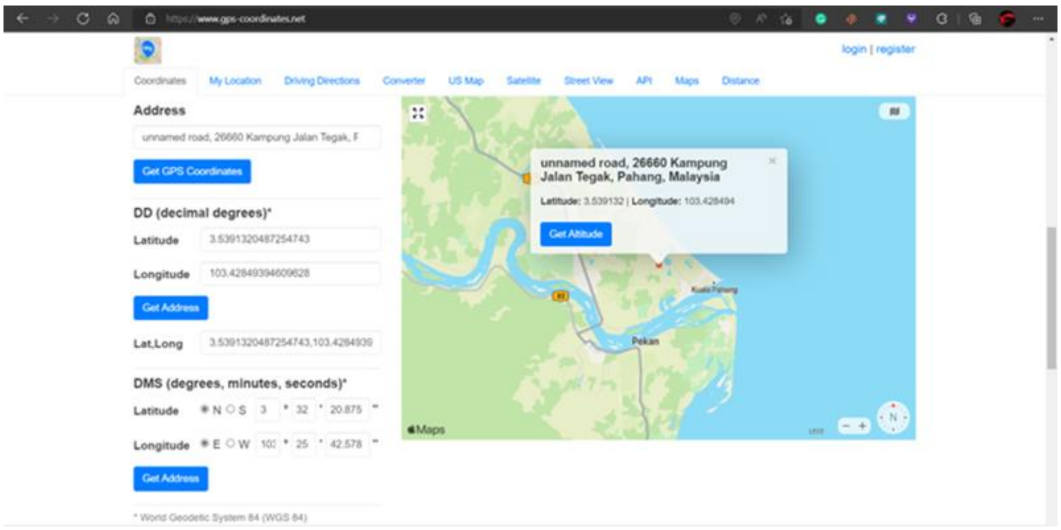


Figure 4. 5 GPS and Accelerometer output displayed



Address

unnamed road, 26660 Kampung Jalan Tegak, F

DD (decimal degrees)*

Latitude 3.5391320487254743

Longitude 103.42849394609628

Lat,Long 3.5391320487254743,103.4284939

DMS (degrees, minutes, seconds)*

Latitude ● N ○ S 3 ° 32 ' 20.875 "

Longitude ● E ○ W 103 ° 25 ' 42.578 "

* World Geodetic System 84 (WGS 84)

unnamed road, 26660 Kampung Jalan Tegak, Pahang, Malaysia

Latitude: 3.539132 | Longitude: 103.428494

Figure 4. 6 GPS coordinate website

The reading for a GPS coordinate is shown in Figure 4.5 using six seven-segment displays. Only six seven-segment indicators were utilized on the board, three of which indicated the latitude coordinate and three of which displayed the longitude coordinate. The first three seven segments show the longitude value of 103, whereas the other three seven segments show the latitude value of 3.3. The ground station will also receive the whole GPS coordinate via a wireless connection. This project also compared the GPS reading with the GPS coordinator online on gpscoordinate.net in Figure 4.6 to verify the GPS coordinate. The LEDs represented the beginning point of the accelerometer reading when the surface was flat or when the CanSat was in balance.

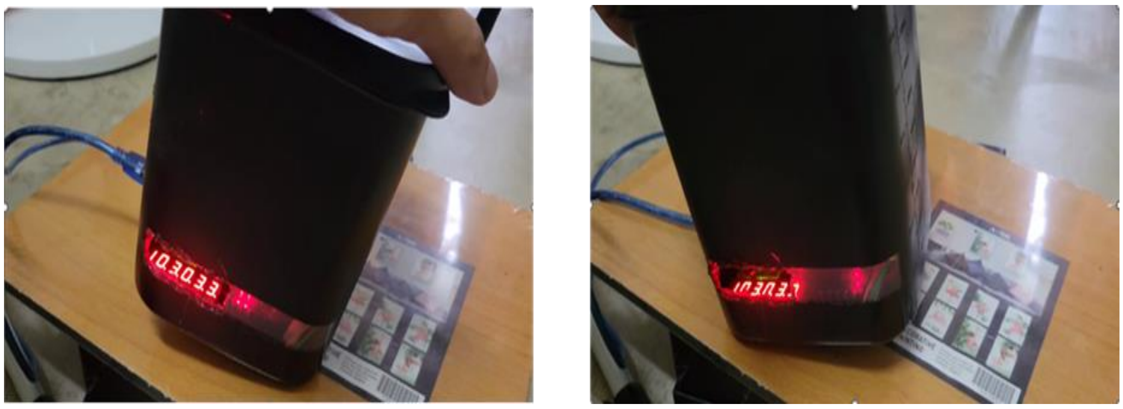
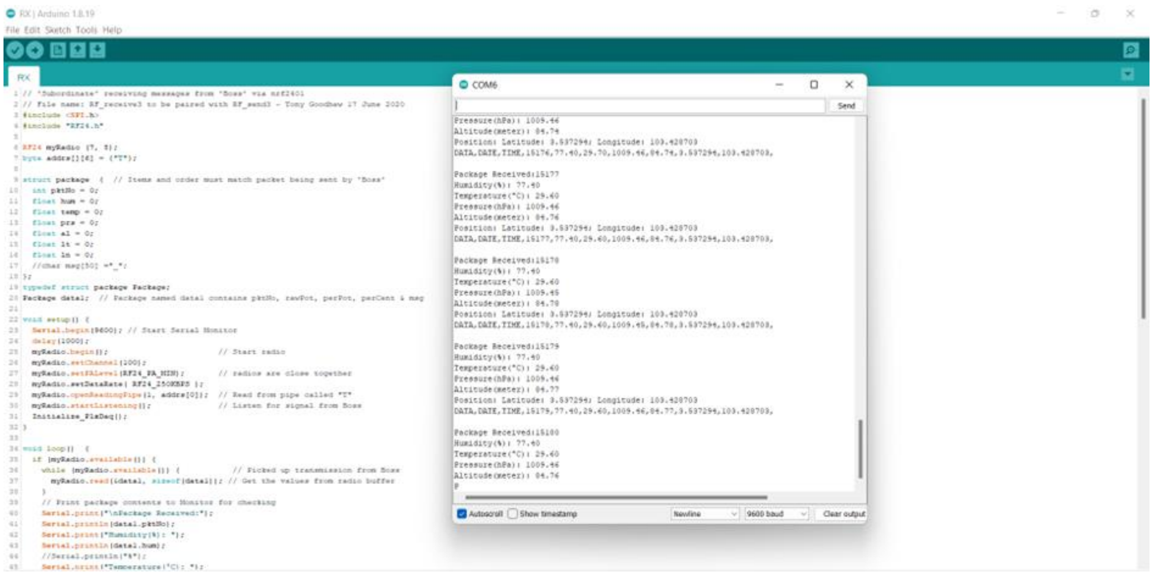


Figure 4. 7 Accelerometer output display on the LEDs

Refer to Figure 4.7 shows how the LEDs light up like a bubble and float above the board. As the board tilts from left to right, the digital accelerometer detects and displays tilting movement on the LEDs. To see the effect on the LEDs, tilt the DE10-Lite board from side - to - side. The tilt of the board is determined by reading the digital accelerometer's X-axis value.

## 4.5 Arduino Uno Serial monitor on the Ground Station



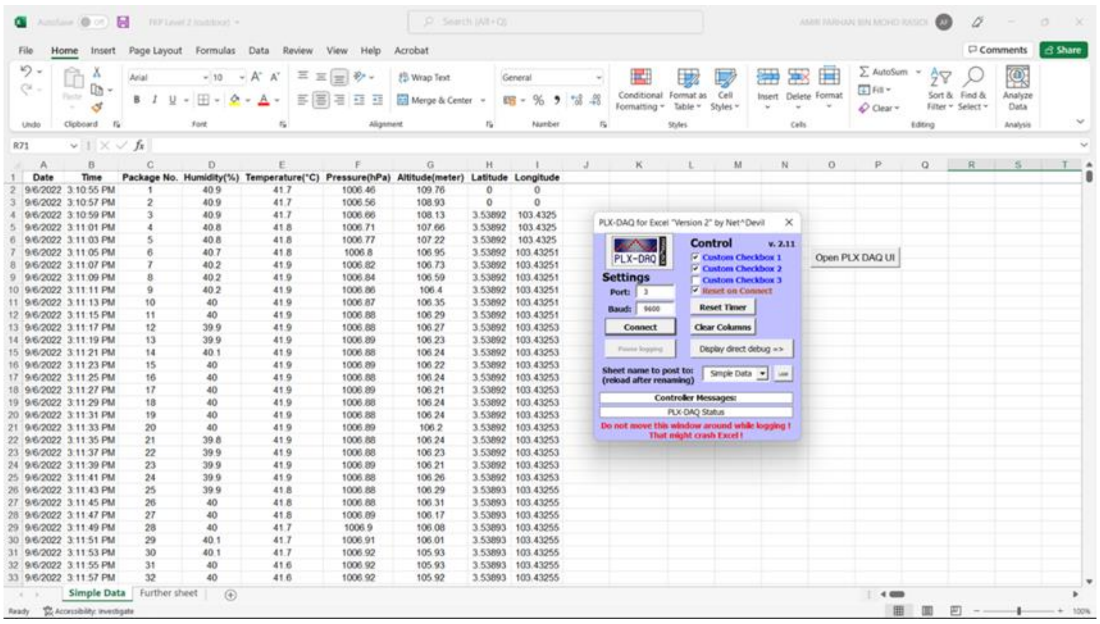
```
// "Subordinate" receiving messages from "Boss" via nrf2401
// File name: RF_receive3 to be paired with RF_send3 - Tony Goodhew 17 June 2020
#include <SPI.h>
#include "RF24.h"

RF24 myRadio (7, 8);
byte addrs[][6] = {"T"};

struct package  {  // Items and order must match packet being sent by "Boss"
  int pktNo = 0;
  float hum = 0;
  float temp = 0;
  float prs = 0;
  float al = 0;
  float lt = 0;
  float ln = 0;
  //char msg[50] ="_";
};
typedef struct package Package;
Package data1;  // Package named data1 contains pktNo, rawPot, perPot, perCent & msg

void setup() {
  Serial.begin(9600); // Start Serial Monitor
  delay(1000);
  myRadio.begin();                          // Start radio
  myRadio.setChannel(100);
  myRadio.setPALevel(RF24_PA_MIN);          // radios are close together
  myRadio.setDataRate( RF24_250KBPS );
  myRadio.openReadingPipe(1, addrs[0]);  // Read from pipe called "T"
  myRadio.startListening();                 // Listen for signal from Boss
  Initialize_PlxDaq();
}

void loop()  {
  if (myRadio.available()) {
    while (myRadio.available()) {           // Picked up transmission from Boss
      myRadio.read(&data1, sizeof(data1)); // Get the values from radio buffer
    }
    // Print package contents to Monitor for checking
    Serial.print("\nPackage Received:");
    Serial.println(data1.pktNo);
    Serial.print("Humidity(%): ");
    Serial.println(data1.hum);
    //Serial.println("%");
    Serial.print("Temperature(°C): ");
```

```
Pressure(hPa): 1009.46
Altitude(meter): 84.74
Position: Latitude: 3.537294; Longitude: 103.428703
DATA,DATE,TIME,15176,77.40,29.70,1009.46,84.74,3.537294,103.428703,

Package Received:15177
Humidity(%): 77.40
Temperature(°C): 29.40
Pressure(hPa): 1009.46
Altitude(meter): 84.76
Position: Latitude: 3.537294; Longitude: 103.428703
DATA,DATE,TIME,15177,77.40,29.40,1009.46,84.76,3.537294,103.428703,

Package Received:15178
Humidity(%): 77.40
Temperature(°C): 29.40
Pressure(hPa): 1009.45
Altitude(meter): 84.78
Position: Latitude: 3.537294; Longitude: 103.428703
DATA,DATE,TIME,15178,77.40,29.40,1009.45,84.78,3.537294,103.428703,

Package Received:15179
Humidity(%): 77.40
Temperature(°C): 29.40
Pressure(hPa): 1009.46
Altitude(meter): 84.77
Position: Latitude: 3.537294; Longitude: 103.428703
DATA,DATE,TIME,15179,77.40,29.40,1009.46,84.77,3.537294,103.428703,

Package Received:15180
Humidity(%): 77.40
Temperature(°C): 29.40
Pressure(hPa): 1009.46
Altitude(meter): 84.76
P
```

Figure 4. 8 Arduino IDE Serial Monitor

Refer Figure 4.8 displays the output from an Arduino IDE serial monitor, with data from the CanSat received via package. The value of humidity (percent), temperature (°C), atmospheric pressure (hPa), altitude (meter), and GPS coordinates for longitude and altitude are all included in each package. For this project, the Baud Rate is 9600. In a communication channel, the baud rate is the pace at which data is conveyed. When discussing serial communication circuits, the term "baud rate" is frequently used. "9600 baud" in the context of a serial port signifies that the serial port can send a maximum of 9600 bits per second, which is sufficient for this project.

## 4.6 Telemetry data in Microsoft Excel on the Ground Station



| Date | Time | Package No. | Humidity(%) | Temperature(°C) | Pressure(hPa) | Altitude(meter) | Latitude | Longitude |
|---|---|---|---|---|---|---|---|---|
| 9/6/2022 | 3:10:55 PM | 1 | 40.9 | 41.7 | 1006.46 | 109.76 | 0 | 0 |
| 9/6/2022 | 3:10:57 PM | 2 | 40.9 | 41.7 | 1006.56 | 108.93 | 0 | 0 |
| 9/6/2022 | 3:10:59 PM | 3 | 40.9 | 41.7 | 1006.66 | 108.13 | 3.53892 | 103.4325 |
| 9/6/2022 | 3:11:01 PM | 4 | 40.8 | 41.8 | 1006.71 | 107.66 | 3.53892 | 103.4325 |
| 9/6/2022 | 3:11:03 PM | 5 | 40.8 | 41.8 | 1006.77 | 107.22 | 3.53892 | 103.4325 |
| 9/6/2022 | 3:11:05 PM | 6 | 40.7 | 41.8 | 1006.8 | 106.95 | 3.53892 | 103.43251 |
| 9/6/2022 | 3:11:07 PM | 7 | 40.2 | 41.9 | 1006.82 | 106.73 | 3.53892 | 103.43251 |
| 9/6/2022 | 3:11:09 PM | 8 | 40.2 | 41.9 | 1006.84 | 106.59 | 3.53892 | 103.43251 |
| 9/6/2022 | 3:11:11 PM | 9 | 40.2 | 41.9 | 1006.86 | 106.4 | 3.53892 | 103.43251 |
| 9/6/2022 | 3:11:13 PM | 10 | 40 | 41.9 | 1006.87 | 106.35 | 3.53892 | 103.43251 |
| 9/6/2022 | 3:11:15 PM | 11 | 40 | 41.9 | 1006.88 | 106.29 | 3.53892 | 103.43251 |
| 9/6/2022 | 3:11:17 PM | 12 | 39.9 | 41.9 | 1006.88 | 106.27 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:19 PM | 13 | 39.9 | 41.9 | 1006.89 | 106.23 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:21 PM | 14 | 40.1 | 41.9 | 1006.88 | 106.24 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:23 PM | 15 | 40 | 41.9 | 1006.89 | 106.22 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:25 PM | 16 | 40 | 41.9 | 1006.88 | 106.24 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:27 PM | 17 | 40 | 41.9 | 1006.89 | 106.21 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:29 PM | 18 | 40 | 41.9 | 1006.88 | 106.24 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:31 PM | 19 | 40 | 41.9 | 1006.88 | 106.24 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:33 PM | 20 | 40 | 41.9 | 1006.89 | 106.2 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:35 PM | 21 | 39.8 | 41.9 | 1006.88 | 106.24 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:37 PM | 22 | 39.9 | 41.9 | 1006.88 | 106.23 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:39 PM | 23 | 39.9 | 41.9 | 1006.89 | 106.21 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:41 PM | 24 | 39.9 | 41.9 | 1006.88 | 106.26 | 3.53892 | 103.43253 |
| 9/6/2022 | 3:11:43 PM | 25 | 39.9 | 41.8 | 1006.88 | 106.29 | 3.53893 | 103.43255 |
| 9/6/2022 | 3:11:45 PM | 26 | 40 | 41.8 | 1006.88 | 106.31 | 3.53893 | 103.43255 |
| 9/6/2022 | 3:11:47 PM | 27 | 40 | 41.8 | 1006.89 | 106.17 | 3.53893 | 103.43255 |
| 9/6/2022 | 3:11:49 PM | 28 | 40 | 41.7 | 1006.9 | 106.08 | 3.53893 | 103.43255 |
| 9/6/2022 | 3:11:51 PM | 29 | 40.1 | 41.7 | 1006.91 | 106.01 | 3.53893 | 103.43255 |
| 9/6/2022 | 3:11:53 PM | 30 | 40.1 | 41.7 | 1006.92 | 105.93 | 3.53893 | 103.43255 |
| 9/6/2022 | 3:11:55 PM | 31 | 40 | 41.6 | 1006.92 | 105.93 | 3.53893 | 103.43255 |
| 9/6/2022 | 3:11:57 PM | 32 | 40 | 41.6 | 1006.92 | 105.92 | 3.53893 | 103.43255 |

Figure 4. 9 Telemetry data in Microsoft Excel

Refer to Figure 4.9 illustrates the data generated from the PLX-DAQ Data Acquisition Software in Microsoft Excel. PLX-DAQ can now send data directly to Excel from any microcontroller connected to a sensor and connected to a PC's serial port. The PLX-DAQ specifications are covered in the Methodology chapter. The first and second columns show the date of the observation and the time each data was received, respectively. The humidity value in percent and the temperature of the environment in degrees Celsius are displayed in the third and fourth columns from the DHT22 temperature and humidity sensor. The BMP280 barometer sensor displays the air pressure in hectopascals (hPa) and altitude in meters in the fifth and sixth columns, respectively. Finally, in the 8th and 9th columns, the GPS coordinates from the GPS-7M-NEO module are displayed. The data collected from various heights and locations will then be processed, and a graph will be created.

## 4.7 Data analysis in data collection

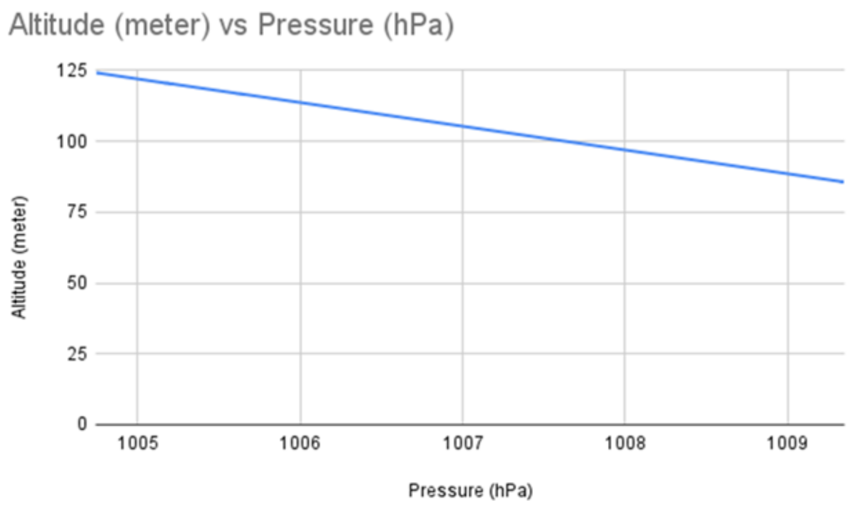


Figure 4. 10 Altitude vs Atmospheric pressure graph

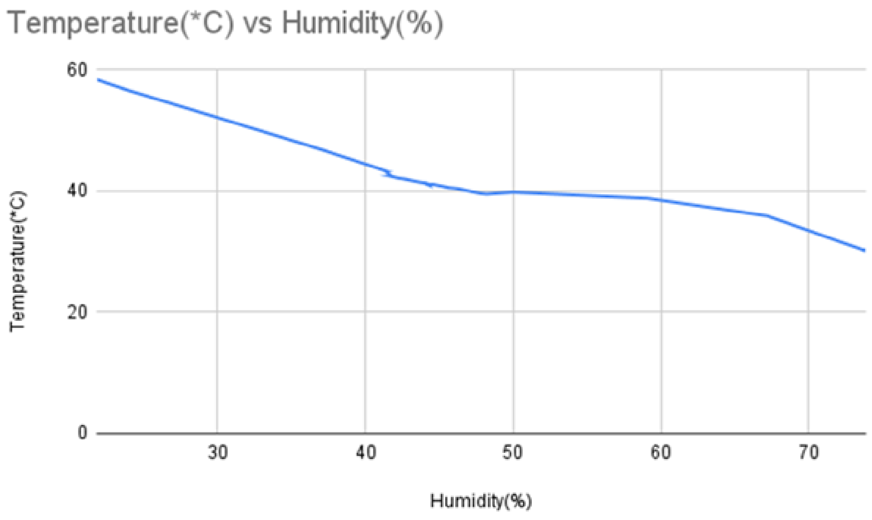


Figure 4. 11 Temperature vs Humidity graph

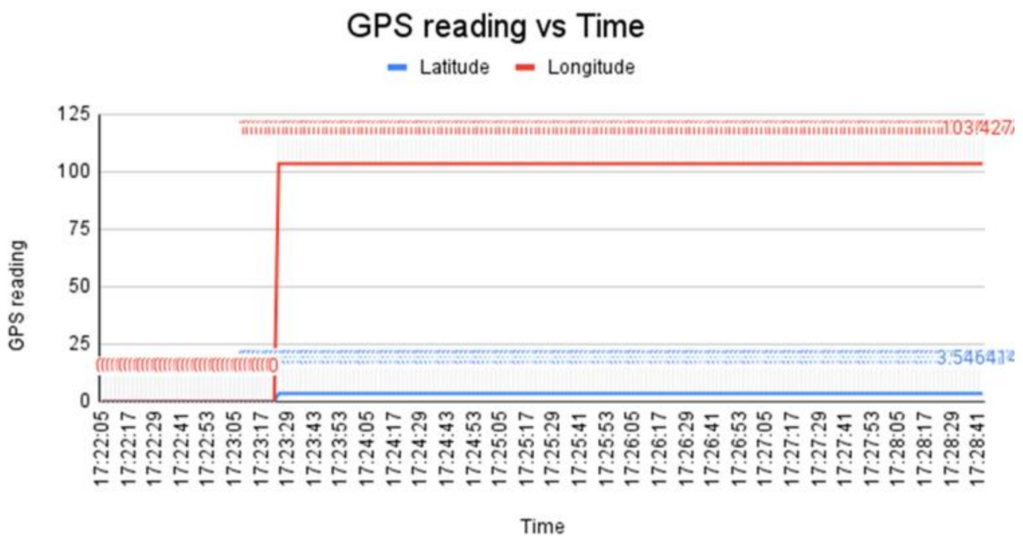


Figure 4. 12 GPS time taken graph

Refer to Figures 4.10 show pressure and altitude readings, Figures 4.11 show temperature and humidity readings, and Figure 4.12 shows GPS Latitude and Longitude readings. All sensors function properly, but the effect of shaking when traveling to each level has harmed the sensors (not properly attached to the header). Another reason for not being able to send all the telemetry data is a poor antenna pointing from the Ground station. Because of the low visibility, binoculars and sunglasses are required to overcome the visual limits, especially during the day. The GPS has a rechargeable battery that helps retain clock data, the latest position data (GNSS orbit data), and module configuration in RAM. This allows much faster position locks. Without the battery the GPS always cold-start and needs to acquire information from the satellites, so the initial GPS lock takes more time and will take more time in the indoor area.

# CHAPTER 5

# CONCLUSION

## 5.1 Introduction

This chapter focused primarily on the conclusions drawn from this research. Furthermore, it suggests future research in aerial surveillance and monitoring systems.

## 5.2 Conclusion

The Telemetry CanSat was successfully designed utilizing FPGA and Arduino Uno in this project. Furthermore, all essential subsystems contained in CanSat, such as the main unit, sensors, and communication system module, were successfully fitted into the smallest possible volume. Because several sensors used in this project require a module to utilize in the FPGA alone, Telemetry CanSat is created utilizing an FPGA DE10-Lite Development board and Arduino Uno as the primary unit.

CanSat technology is a foundational topic in space engineering education. It has only been utilized for educational purposes until now, but its incorporation in a scientific endeavor will show that it can be used for both scientific and technological goals. CanSat is well suited for satellite modeling in education and research since it can simply reflect how the satellite's systems work. For example, telemetry data collected by sensors were utilized to monitor atmospheric conditions. Due to a technical issue with obtaining a drone to attach to the CanSat, the final design could not be tested.

## 5.3 Project Management

### 5.3.1 Cost of Project

For the project management of this project, I need to use hardware component for this project and their prices are listed in table 5.1 below. I spent RM162.30 on this study because of Intel Microelectronics (M) Sdn. Bhd. sponsored the DE10-Lite Development Board during my internship with this company.

Table 5. 1 Cost of Project

| No | Item | Unit | Price/Unit |
|---|---|---|---|
| 1 | DE10-Lite Development Kit | 1 | RM 448.56 |
| 2 | Arduino Uno R3 Microcontroller | 2 | RM 59.80 |
| 3 | DHT22 Module | 1 | RM 14.90 |
| 4 | GPS NEO-7M Module | 1 | RM 41.50 |
| 5 | 40 ways Male to Male Jumper | 1 | RM 6.00 |
| 6 | BMP280 Barometric Sensor Module | 1 | RM 9.50 |
| 7 | NRF24L01+ with PA and LNA Module | 2 | RM 17.60 |
| 8 | Cable Sleeve (5 meter) | 1 | RM 5.00 |
| 9 | Case | 1 | RM 8.00 |
|  | **Total** |  | RM 610.86 |

## 5.4 Future Recommendation

The module for each sensor can be constructed in the future so that this project can solely utilize FPGA because FPGA has several advantages over microcontrollers, one of which is the ability to execute On-Board Processing. For the next upgraded version of CanSat, it is recommended that you use a 3D printed machine with advanced features or choose a new technique of construction.

# REFERENCES

[1] Z. Zaheer, A. Usmani, E. Khan, and M. A. Qadeer, “Aerial surveillance system using UAV,” *IFIP Int. Conf. Wirel. Opt. Commun. Networks, WOCN*, vol. 2016-Novem, 2016, doi: 10.1109/WOCN.2016.7759885.

[2] A. Colin and M. Jimenez-Lizárraga, “The Cansat Technology for Climate Monitoring in Small Regions at Altitudes Below 1km,” *IAAA Clim. Chang. Disaster Manag. Conf.*, no. September 2015, pp. 1–10, 2017.

[3] P. Rudol, *Increasing Autonomy of Unmanned Aircraft Systems Through the Use of Imaging Sensors*, no. 1510. 2011.

[4] S. W. Walker, U. S. A. F. Academy, and C. Springs, “INTEGRATING DEPARTMENT OF DEFENSE UNMANNED AERIAL SYSTEMS INTO THE NATIONAL AIRSPACE STRUCTURE A thesis presented to the Faculty of the U . S . Army Command and General Staff College in partial fulfillment of the requirements for the degree by Master ’ s ,” *Methodology*, 2010.

[5] sezer çoban and tuğrul oktay, “Legal and Ethical Issues of Unmanned Aerial Vehicles,” *J. Aviat.*, no. June, 2018, doi: 10.30518/jav.421644.

[6] U.S. Army, “Unmanned Aircraft Systems Roadmap 2010-2035,” *Fed. Am. Sci.*, pp. 1–140, 2010.

[7] J. Oliver, *Autonomus Flying Robot*, vol. 53, no. 9. 2013.

[8] S. G. Gupta, M. Ghonge, and P. M. Jawandhiya, “Review of Unmanned Aircraft System (UAS),” *SSRN Electron. J.*, no. April, 2019, doi: 10.2139/ssrn.3451039.

[9] H. Chao, Y. Cao, and Y. Chen, “Autopilots for small unmanned aerial vehicles: A survey,” *Int. J. Control. Autom. Syst.*, vol. 8, no. 1, pp. 36–44, 2010, doi: 10.1007/s12555-010-0105-z.

[10] J. a. Winnefeld and F. Kendall, “Unmanned Systems integrated roadmap: FY 2011-2036,” pp. 1–74, 2010.

[11] D. D. Weatherington, “UAS Planning Task Force,” 2007.

[12] U. A. Systems, “UAS-based Environmental Monitoring.”

[13] P. Catalog, “Intel ® Cover TBD Product.”

[14] Xilinx Inc., “Introduction to FPGA Design with Vivado High-Level Synthesis UG998,” *Ug998*, vol. UG998, pp. 1–89, 2019, [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf.

[15] M. H. Mickle, “Product Summary,” *IEEE Micro*, vol. 7, no. 2, pp. 100–101, 1987, doi: 10.1109/MM.1987.304862.

[16] S. Microsystems, “OpenSPARC T1 Microarchitecture Specification,” no. 819, 2009.

[17] I. Kuon and J. Rose, *Quantifying and exploring the gap between FPGAs and ASICs: Measuring and exploring*. 2010.

[18] T. Thai, “Applications for FPGAs on Nanosatellites,” no. April, 2014.

[19] S. Yamaura, H. Akiyama, and R. Kawashima, “Report of CanSat leader training program,” *RAST 2011 - Proc. 5th Int. Conf. Recent Adv. Sp. Technol.*, pp. 856–860, 2011, doi: 10.1109/RAST.2011.5966964.

[20] E. Bautista-linares *et al.*, “CanSat,” pp. 0–3, 2015.

[21] M. E. Umit, W. Cabanas, M. Tetlow, H. Akiyama, S. Yamaura, and S. Olaleye, “Development of a fly-back CANSAT in 3 weeks,” *RAST 2011 - Proc. 5th Int. Conf. Recent Adv. Sp. Technol.*, pp. 804–807, 2011, doi: 10.1109/RAST.2011.5966953.

[22] M. Abo-arais *et al.*, “Approaching a nano-satellite using CAN-SAT systems,” pp. 813–817, 2015.

[23] M. Celebi *et al.*, “Design and navigation control of an advanced level CANSAT,” *RAST 2011 - Proc. 5th Int. Conf. Recent Adv. Sp. Technol.*, pp. 752–757, 2011, doi: 10.1109/RAST.2011.5966942.

[24] Y. Miyazaki and M. Yamazaki, “A practical education of space engineering by using CanSat and pico-satellite - Fruitful collaboration with UNISEC for success of student satellite program - Fruitful c,” *RAST 2013 - Proc. 6th Int. Conf. Recent Adv. Sp. Technol.*, pp. 1081–1086, 2013, doi: 10.1109/RAST.2013.6581163.

[25] M. Ostaszewski, K. Dzierzek, and Ł. Magnuszewski, “Analysis of data collected while CanSat mission,” *Proc. 2018 19th Int. Carpathian Control Conf. ICCC 2018*, pp. 1–4, 2018, doi: 10.1109/CarpathianCC.2018.8399591.

[26] R. P. Ramadhan, A. R. Ramadhan, S. A. Putri, M. I. C. Latukolan, Edwar, and Kusmadi, “Prototype of CanSat with Auto-gyro Payload for Small Satellite Education,” *TSSA 2019 - 13th Int. Conf. Telecommun. Syst. Serv. Appl. Proc.*, pp. 243–248, 2019, doi: 10.1109/TSSA48701.2019.8985514.

[27] M. E. Aydemir, R. C. Dursun, and M. Pehlevan, “Ground station design procedures for CANSAT,” *RAST 2013 - Proc. 6th Int. Conf. Recent Adv. Sp. Technol.*, no. 1, pp. 909–912, 2013, doi: 10.1109/RAST.2013.6581343.

[28] Q. P. Lite, “Introduction to FPGA Simulation and Debug,” 2018.

[29] S. Feedback and S. Jose, “Quartus Prime Standard Edition Handbook Volume 3: Verification,” vol. 3, 2016.

[30] T. Office, “INTRO TO INTEL ® FPGAS AND INTEL ® QUARTUS ® PRIME,” 2018.

[31] M. Fezari and A. Al Dahoud, “Integrated Development Environment ‘ IDE ’ For Arduino,” *ResearchGate*, no. October, pp. 1–12, 2018, [Online]. Available: https://www.researchgate.net/publication/328615543%0AIntegrated.

[32] T. E. Ui *et al.*, “Beginners Guide to PLX DAQ v2 by Net Devil The Excel UI part,” no.

Revision 1, pp. 1–12, 2016.

[33] E. Dist and H. City, “DE10-Lite Board,” no. 176.

[34] P. R. Manual, “Arduino ® UNO R3 Target areas : Arduino ® UNO R3 Features,” pp. 1–13, 2021.

[35] T. Liu, “Digital-output relative humidity & temperature sensor/module DHT22,” *New York Aosong Electron.*, vol. 22, pp. 1–10, 2015, [Online]. Available: https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf.

[36] D. Srivastava, A. Kesarwani, and S. Dubey, “Measurement of Temperature and Humidity by using Arduino Tool and DHT11,” *Int. Res. J. Eng. Technol.*, vol. 05, no. 12, pp. 876–878, 2018.

[37] T. Mmaq, “Digital Accelerometer,” pp. 1–47, 2011.

[38] U-blox, “NEO-7 - Data Sheet Document,” pp. 1–26, 2014, [Online]. Available: https://www.u-blox.com/sites/default/files/products/documents/NEO-7_DataSheet_%28UBX-13003830%29.pdf.

[39] Bosch, “BMP280: Datasheet,” *Digit. Presusure Sens.*, p. 49, 2015, [Online]. Available: https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf.

[40] N. Simicondutor, “Preliminary Product Specification v1.0,” no. March, pp. 1–75, 2008, [Online]. Available: https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf.

[41] R. Rittenberry, “Hands-on technology.,” *Occup. Health Saf.*, vol. 74, no. 2, p. 24, 2005.

# APPENDICES

## Appendix A Gantt Chart

| | PSM1 | | | | | | | | | | | | | | PSM2 | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Week/Task | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Meeting with a subject coordinator -PSM briefing session | 🟧 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PSM Titles realaease | 🟧 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ApprovAL FYP title | 🟧 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PSM Register Title | 🟧 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Meeting with supervisor | 🟧 | 🟧 | | 🟧 | | | | 🟧 | 🟧 | 🟧 | | | 🟧 | | | | | | | | | | | | | | | |
| Study of problem | | 🟧 | 🟧 | 🟧 | | | | | | | | | | | | | | | | | | | | | | | | |
| Identify the project objecttive and expected results | | | | 🟧 | 🟧 | | | | | | | | | | | | | | | | | | | | | | | |
| Selection of software for simulation | | | | 🟧 | 🟧 | | | | | | | | | | | | | | | | | | | | | | | |
| Hardware component selection | | | | | 🟧 | 🟧 | | | | | | | | | | | | | | | | | | | | | | |
| Determine the best method for project | | | | | 🟧 | 🟧 | 🟧 | 🟧 | 🟧 | | | | | | | | | | | | | | | | | | | |
| Progress presentation to SV | | | | 🟧 | | | | 🟧 | 🟧 | 🟧 | | | | | | | | | | | | | | | | | | |
| Analytical review | | | | | | | | | 🟧 | 🟧 | | | | | | | | | | | | | | | | | | |
| Presentation slide submission to panel/supervisor | | | | | | | | | | | | | 🟧 | | | | | | | | | | | | | | | |
| Final presentation | | | | | | | | | | | | | 🟧 | | | | | | | | | | | | | | | |
| Report and logbook submission to supervisor | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Submission to PSM1 Cordinantor | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Learn basic concept of FPGA board | | | | | | | | | | | 🟧 | 🟧 | | | | | | | | | | | | | | | | |
| Report and logbook submission to supervisor | | | | | | | | | | | | | 🟧 | | | | | | | | | | | | | | | |
| Submission to PSM1 Cordinantor | | | | | | | | | | | | | | 🟧 | | | | | | | | | | | | | | |
| Learn basic concept of FPGA board | | | | | | | | | | | | | | | 🟧 | 🟧 | | | | | | | | | | | | |
| Interface connection between FPGA board amd other component | | | | | | | | | | | | | | | | 🟧 | 🟧 | 🟧 | | 🟧 | 🟧 | | | | | | | |
| Project Coding | | | | | | | | | | | | | | | | 🟧 | 🟧 | 🟧 | | | 🟧 | 🟧 | 🟧 | | 🟧 | | | |
| Software simulation in Intel Quartus Prime Software | | | | | | | | | | | | | | | | 🟧 | 🟧 | 🟧 | | | | | 🟧 | | 🟧 | | | |
| Hardware tesing | | | | | | | | | | | | | | | | | | 🟧 | 🟧 | 🟧 | 🟧 | 🟧 | 🟧 | 🟧 | 🟧 | | | |
| Progress presentation to SV 2 | | | | | | | | | | | | | | | | | | | | | | | | | | 🟧 | 🟧 | |
| Presentation slide submission to panel/supervisor 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | 🟧 | |
| Final presentation 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | 🟧 |
| Report and logbook submission to supervisor 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | 🟧 |
| Submission to PSM2 Cordinantor | | | | | | | | | | | | | | | | | | | | | | | | | | | | 🟧 |

# Appendix B    Arduino IDE coding

## Arduino coding for Ground Station

```
#include <SPI.h>
#include "RF24.h"

RF24 myRadio (7, 8);
byte addrs[][6] = {"T"};

struct package  {  // Items and order must match packet being sent by
'Boss'
  int pktNo = 0;
  float hum = 0;
  float temp = 0;
  float prs = 0;
  float al = 0;
  float lt = 0;
  float ln = 0;
  //char msg[50] ="_";
};
typedef struct package Package;
Package data1;  // Package named data1 contains pktNo, rawPot, perPot,
perCent & msg

void setup() {
  Serial.begin(9600); // Start Serial Monitor
  delay(1000);
  myRadio.begin();                         // Start radio
  myRadio.setChannel(100);
  myRadio.setPALevel(RF24_PA_MIN);         // radios are close together
  myRadio.setDataRate( RF24_250KBPS );
  myRadio.openReadingPipe(1, addrs[0]);    // Read from pipe called "T"
  myRadio.startListening();                // Listen for signal from
Boss
  Initialize_PlxDaq();
}

void loop()  {
  if (myRadio.available()) {
    while (myRadio.available()) {            // Picked up transmission
from Boss
      myRadio.read(&data1, sizeof(data1)); // Get the values from
radio buffer
    }
    // Print package contents to Monitor for checking
    Serial.print("\nPackage Received:");
    Serial.println(data1.pktNo);
    Serial.print("Humidity(%): ");
    Serial.println(data1.hum);
    Serial.print("Temperature(°C): ");
    Serial.println(data1.temp);
    Serial.print("Pressure(hPa): ");
    Serial.println(data1.prs);
    Serial.print("Altitude(meter): ");
    Serial.println(data1.al);
    Serial.print("Position: ");
    Serial.print("Latitude: ");
    Serial.print(data1.lt,6);
    Serial.print("; ");
    Serial.print("Longitude: ");
    Serial.println(data1.ln,6);
    Write_PlxDaq();
    delay(800);
  }
}
void Initialize_PlxDaq()
{
Serial.println("CLEARDATA"); //clears up any data left from previous
projects
Serial.println("LABEL,Date,Time,Package
No.,Humidity(%),Temperature(*C),Pressure(hPa),Altitude(meter),Latitude
,Longitude"); //always write LABEL, to indicate it as first line
}
void Write_PlxDaq()
  {
    Serial.print("DATA"); //always write "DATA" to Inidicate the
following as Data
    Serial.print(","); //Move to next column using a ","
    Serial.print("DATE"); //Store date on Excel
    Serial.print(","); //Move to next column using a ","
    Serial.print("TIME"); //Store date on Excel
    Serial.print(","); //Move to next column using a ","
    Serial.print(data1.pktNo); //Store date on Excel
    Serial.print(","); //Move to next column using a ","
    Serial.print(data1.hum); //Store date on Excel
    Serial.print(","); //Move to next column using a ","
    Serial.print(data1.temp); //Store date on Excel
    Serial.print(","); //Move to next column using a ","
    Serial.print(data1.prs); //Store date on Excel
    Serial.print(","); //Move to next column using a ","
    Serial.print(data1.al); //Store date on Excel
    Serial.print(","); //Move to next column using a ","
    Serial.print(data1.lt,6); //Store date on Excel
    Serial.print(","); //Move to next column using a ","
    Serial.print(data1.ln,6); //Store date on Excel
    Serial.print(","); //Move to next column using a ","
    Serial.println(); //End of Row move to next row
  }
```

Arduino coding for CanSat

```
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
//#include <Adafruit_SSD1306.h>
#include <Adafruit_BMP280.h>
#include "DHT.h"
#include "RF24.h"
#include <SoftwareSerial.h>
#include <TinyGPS.h>

Adafruit_BMP280 bmp; // I2C

float lat = 0.00,lon = 0.00; //create variable for latitude and
longitude object
SoftwareSerial gpsSerial(3,4);//rx,tx
TinyGPS gps; // create gps object

#define DHTPIN 2
#define DHTTYPE DHT22   // DHT 22  (AM2302), AM2321
DHT dht(DHTPIN, DHTTYPE);
float humi;
float tempC;
float tempF;
float pres;
float alt;

RF24 myRadio (7, 8);      // These can be changed CE CSN
byte addrs[][6] = {"T"};  // Single data1 pipe to send on

struct package {         // Define the data1 packet contents
  int pktNo = 0;         // Count and identify packets sent
  float hum = 0;        // 0 .. 1023 from 10K Ohm potentiometer
  float temp = 0.0;   // floating point value
  float prs = 0;
  float al = 0;
  float lt = 0;
  float ln = 0;
  //char msg[50] = "";  // Text string - Not changing
};
typedef struct package Package;
Package data1;   // Package named data1 contains pktNo, hum, temp,
prs, al, lt, ln, msg

void setup() {
  Serial.begin(9600);                  // Start Serial Monitor
  dht.begin();
  bmp.begin();
  gpsSerial.begin(9600);                  // connect gps sensor
  myRadio.begin();                       // Start radio
  myRadio.setChannel(100);               // Could be changed 1 ... 125
  myRadio.setPALevel(RF24_PA_MIN);       // Power level minimum -
radios are close together
  myRadio.setDataRate( RF24_250KBPS );
  myRadio.openWritingPipe(addrs[0]);    // Write to pipe called "T"
  delay(750);

    /* Default settings from datasheet. */
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,     /* Operating Mode.
*/
                  Adafruit_BMP280::SAMPLING_X2,     /* Temp.
oversampling */
                  Adafruit_BMP280::SAMPLING_X16,    /* Pressure
oversampling */
                  Adafruit_BMP280::FILTER_X16,      /* Filtering. */
                  Adafruit_BMP280::STANDBY_MS_500); /* Standby time.
*/
}

void loop() {
  // Wait a few seconds between measurements.
  delay(1000);
  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very
slow sensor)
  humi = dht.readHumidity();
  // Read temperature as Celsius (the default)
  tempC = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  tempF = dht.readTemperature(true);

  pres = bmp.readPressure()/100;
  alt = bmp.readAltitude(1019.66);

    while(gpsSerial.available()){ // check for gps data
    if(gps.encode(gpsSerial.read()))// encode gps data
    {
   gps.f_get_position(&lat,&lon); // get latitude and longitude
    }
    }

  data1.hum = humi;
  data1.temp = tempC;
  data1.prs = pres;
  data1.al = alt;
  data1.lt = lat;
  data1.ln = lon;
  data1.pktNo = data1.pktNo + 1;

  // Send the packet of data called data1 to Subordinate
  myRadio.write(&data1, sizeof(data1));

  // Report sent values to Monitor
  Serial.print("\nPackage Sent:");
  Serial.println(data1.pktNo);
  Serial.print("Humidity(%): ");
  Serial.println(data1.hum);
  Serial.print("Temperature(°C): ");
  Serial.println(data1.temp);
  Serial.print("Pressure(hPa): ");
  Serial.println(data1.prs);
  Serial.print("Altitude(meter): ");
  Serial.println(data1.al);
  Serial.print("Position: ");
  Serial.print("Latitude: ");
  Serial.print(data1.lt,6);
  Serial.print("; ");
  Serial.print("Longitude: ");
  Serial.println(data1.ln,6);
  delay(800);
}
```

Appendix C Quartus Prime Code (Verilog)

DE10_LITE_Golden_Top,v

```
`define ENABLE_ADC_CLOCK
`define ENABLE_CLOCK1
`define ENABLE_CLOCK2
`define ENABLE_SDRAM
`define ENABLE_HEX0
`define ENABLE_HEX1
`define ENABLE_HEX2
`define ENABLE_HEX3
`define ENABLE_HEX4
`define ENABLE_HEX5
`define ENABLE_KEY
`define ENABLE_LED
`define ENABLE_SW
`define ENABLE_VGA
`define ENABLE_ACCELEROMETER
`define ENABLE_ARDUINO
`define ENABLE_GPIO
module DE10_LITE_Golden_Top(
      //////////// ADC CLOCK: 3.3-V LVTTL //////////
`ifdef ENABLE_ADC_CLOCK
      input                                ADC_CLK_10,
`endif
      //////////// CLOCK 1: 3.3-V LVTTL //////////
`ifdef ENABLE_CLOCK1
      input                                MAX10_CLK1_50,
`endif
      //////////// CLOCK 2: 3.3-V LVTTL //////////
`ifdef ENABLE_CLOCK2
      input                                MAX10_CLK2_50,
`endif
      //////////// SDRAM: 3.3-V LVTTL //////////
`ifdef ENABLE_SDRAM
      output                [12:0]         DRAM_ADDR,
      output                 [1:0]         DRAM_BA,
      output                               DRAM_CAS_N,
      output                               DRAM_CKE,
      output                               DRAM_CLK,
      output                               DRAM_CS_N,
      inout                 [15:0]         DRAM_DQ,
      output                               DRAM_LDQM,
      output                               DRAM_RAS_N,
      output                               DRAM_UDQM,
      output                               DRAM_WE_N,
`endif
      //////////// SEG7: 3.3-V LVTTL //////////
`ifdef ENABLE_HEX0
```

```
      output                  [7:0]        HEX0,
`endif
`ifdef ENABLE_HEX1
      output                  [7:0]        HEX1,
`endif
`ifdef ENABLE_HEX2
      output                  [7:0]        HEX2,
`endif
`ifdef ENABLE_HEX3
      output                  [7:0]        HEX3,
`endif
`ifdef ENABLE_HEX4
      output                  [7:0]        HEX4,
`endif
`ifdef ENABLE_HEX5
      output                  [7:0]        HEX5,
`endif
      //////////// KEY: 3.3 V SCHMITT TRIGGER //////////
`ifdef ENABLE_KEY
      input                   [1:0]        KEY,
`endif
      //////////// LED: 3.3-V LVTTL //////////
`ifdef ENABLE_LED
      output                  [9:0]        LEDR,
`endif
      //////////// SW: 3.3-V LVTTL //////////
`ifdef ENABLE_SW
      input                   [9:0]        SW,
`endif
      //////////// VGA: 3.3-V LVTTL //////////
`ifdef ENABLE_VGA
      output                  [3:0]        VGA_B,
      output                  [3:0]        VGA_G,
      output                               VGA_HS,
      output                  [3:0]        VGA_R,
      output                               VGA_VS,
`endif
      //////////// Accelerometer: 3.3-V LVTTL //////////
`ifdef ENABLE_ACCELEROMETER
      output                               GSENSOR_CS_N,
      input                   [2:1]        GSENSOR_INT,
      output                               GSENSOR_SCLK,
      inout                                GSENSOR_SDI,
      inout                                GSENSOR_SDO,
`endif
      //////////// Arduino: 3.3-V LVTTL //////////
`ifdef ENABLE_ARDUINO
      inout                  [15:0]        ARDUINO_IO,
      inout                                ARDUINO_RESET_N,
`endif
      //////////// GPIO, GPIO connect to GPIO Default: 3.3-V LVTTL
//////////
`ifdef ENABLE_GPIO
      inout                  [35:0]        GPIO
`endif
```

```
);
//    GPS
reg [7:0] txData;
reg txLoad  = 1'b0;
wire [7:0] rxData;
reg flag = 1'b1;
wire txReset = 1'b1;
wire rxReset = 1'b1;
reg [3:0] hex_info [5:0];
wire txIdle;
wire txReady;
wire rxIdle;
wire rxReady;
parameter ClkFrequency = 50000000;
parameter Baud = 9600;
wire RxD_data_ready;
wire [7:0] RxD_data;
//    Accelerometer - SPI
wire        dly_rst;
wire        spi_clk, spi_clk_out;
wire  [15:0]  data_x;
//    GPS
async_receiver GPS(    .clk(MAX10_CLK1_50),
                                         .rst(1'b0),
                                         .RxD(GPIO[35]),

                                         .RxD_data_ready(rxReady),
                                         .RxD_data(rxData));
//    Accelerometer
//    Reset
reset_delay u_reset_delay    (
             .iRSTN(KEY[0]),
             .iCLK(MAX10_CLK1_50),
             .oRST(dly_rst));
//  PLL
spi_pll     u_spi_pll  (
             .areset(dly_rst),
             .inclk0(MAX10_CLK1_50),
             .c0(spi_clk),      // 2MHz
             .c1(spi_clk_out)); // 2MHz phase shift
//  Initial Setting and Data Read Back
spi_ee_config u_spi_ee_config (
                                   .iRSTN(!dly_rst),


                                   .iSPI_CLK(spi_clk),

                                   .iSPI_CLK_OUT(spi_clk_out),

                                   .iG_INT2(GSENSOR_INT[1]),
                                   .oDATA_L(data_x[7:0]),
                                   .oDATA_H(data_x[15:8]),
                                   .SPI_SDIO(GSENSOR_SDI),
                                   .oSPI_CSN(GSENSOR_CS_N),
                                   .oSPI_CLK(GSENSOR_SCLK));
```

```
//    LED
led_driver u_led_driver      (
                                    .iRSTN(!dly_rst),
                                    .iCLK(MAX10_CLK1_50),
                                    .iDIG(data_x[9:0]),
                                    .iG_INT2(GSENSOR_INT[1]),
                                    .oLED(LEDR));
//    GPS
defparam GPS.ClkFrequency=ClkFrequency;
defparam GPS.Baud=Baud;

hex7seg h0(LAT[3:0], HEX0);
hex7seg h1(LAT[11:8], HEX1);
hex7seg h2(LAT[19:16], HEX2);
hex7seg h3(LON[3:0], HEX3);
hex7seg h4(LON[11:8], HEX4);
hex7seg h5(LON[19:16], HEX5);
integer precision = 3;
reg [2:0] STATE=0;
reg [23:0] NMEA=0;
reg [23:0] LAT=0;
reg [23:0] LON=0;
integer prec_cnt = 0;
always @(posedge MAX10_CLK1_50) begin
      case(STATE)
            0: begin //Search for '$'
                  if(rxReady && rxData=="$")begin// '$': Start of
frame
                        STATE<=STATE+1;
                  end
            end
            1: begin //Check Tag, if not the proper tag go back
                  if(rxReady) begin
                        if(rxData==",")begin
                              if(NMEA=="GLL")begin
                                    STATE<=STATE+1;
                              end
                              else begin
                                    STATE<=0;
                              end
                        end
                        else begin //Grab the NMEA Message Type

                              NMEA[23:8]<=NMEA[15:0];
                              NMEA[7:0]<=rxData; //Grab the tag info

                        end
                  end
            end
            2: begin //Parse the latitude
                  if(rxReady) begin
                        if(rxData==",")begin
                              STATE<=STATE+1;
                              prec_cnt = 0;
```

```
                    end
                    else if (prec_cnt < precision) begin //grab
only 3 first LAT values, because we have only 3 HEX available
                         prec_cnt = prec_cnt + 1;
                         LAT[23:8]<=LAT[15:0];
                         LAT[7:0]<=rxData;
                    end
               end
          end
          3: begin //Parse letter
               if(rxReady) begin
                    if(rxData==",")begin
                         STATE<=STATE+1;
                    end
               end
          end
          4: begin //Parse the longitude
               if(rxReady) begin
                    if(rxData==10)begin //LF: Line Feed, end of
message, go to idle state
                         STATE<=0;
                         prec_cnt = 0;
                    end
                    else if (prec_cnt < precision)begin //grab
only 3 first LON values, because we have only 3 HEX available
                         prec_cnt = prec_cnt + 1;
                         LON[23:8]<=LON[15:0];
                         LON[7:0]<=rxData;

                    end
               end
          end
          default: begin
               STATE<=0;
          end
     endcase
end
endmodule
```

## Quartus Prime module file

### hex7seg.v

```
module hex7seg (hex, display);
input [3:0] hex;
output [0:6] display;
reg [0:6] display;
/*
* - 0 -
* 5 | | 1
* - 6 -
* 4 | | 2
* - 3 -
*/
always @ (hex)
case (hex)
4'h0: display = 7'b1000000;
4'h1: display = 7'b1111001;
4'h2: display = 7'b0100100;
4'h3: display = 7'b0110000;
4'h4: display = 7'b0011001;
4'h5: display = 7'b0010010;
4'h6: display = 7'b0000010;
4'h7: display = 7'b1111000;
4'h8: display = 7'b0000000;
4'h9: display = 7'b0011000;
4'hA: display = 7'b0001000;
4'hb: display = 7'b0000011;
4'hC: display = 7'b1000110;
4'hd: display = 7'b0100001;
4'hE: display = 7'b0000110;
4'hF: display = 7'b0001110;
endcase
endmodule
```

async_receiver.v

```
module async_receiver(clk,
                               rst,
                               RxD, //rx pin
                               RxD_data_ready,
                               RxD_data,
                               RxD_endofpacket,
                               RxD_idle);
input clk, rst, RxD;
output RxD_data_ready;  // one clock pulse when RxD_data is valid
output [7:0] RxD_data;
parameter ClkFrequency = 24000000; // 24MHz
parameter Baud = 115200;
// We also detect if a gap occurs in the received stream of
characters
// That can be useful if multiple characters are sent in a burst
//  so that multiple characters can be treated as a "packet"
output RxD_endofpacket;  // one clock pulse, when no more data is
received (RxD_idle is going high)
output RxD_idle;  // no data is being received
// Baud generator (we use 8 times oversampling)
parameter Baud8 = Baud*8;
parameter Baud8GeneratorAccWidth = 16;
wire [Baud8GeneratorAccWidth:0] Baud8GeneratorInc =
((Baud8<<(Baud8GeneratorAccWidth-
7))+(ClkFrequency>>8))/(ClkFrequency>>7);
reg [Baud8GeneratorAccWidth:0] Baud8GeneratorAcc;
always @(posedge clk or posedge rst) if(rst) Baud8GeneratorAcc<=0;
else Baud8GeneratorAcc <= Baud8GeneratorAcc[Baud8GeneratorAccWidth-
1:0] + Baud8GeneratorInc;
wire Baud8Tick = Baud8GeneratorAcc[Baud8GeneratorAccWidth];
////////////////////////////
reg [1:0] RxD_sync_inv;
always @(posedge clk or posedge rst) if(rst) RxD_sync_inv<=0; else
if(Baud8Tick) RxD_sync_inv <= {RxD_sync_inv[0], ~RxD};
// we invert RxD, so that the idle becomes "0", to prevent a
phantom character to be received at startup
reg [1:0] RxD_cnt_inv;
reg RxD_bit_inv;
always @(posedge clk or posedge rst)
if(rst)
      RxD_cnt_inv <= 0;
else
if(Baud8Tick)
begin
      if( RxD_sync_inv[1] && RxD_cnt_inv!=2'b11) RxD_cnt_inv <=
RxD_cnt_inv + 2'h1;
      else
      if(~RxD_sync_inv[1] && RxD_cnt_inv!=2'b00) RxD_cnt_inv <=
RxD_cnt_inv - 2'h1;
      if(RxD_cnt_inv==2'b00) RxD_bit_inv <= 1'b0;
      else
      if(RxD_cnt_inv==2'b11) RxD_bit_inv <= 1'b1;
end
```

```
reg [3:0] state;
reg [3:0] bit_spacing;
// "next_bit" controls when the data sampling occurs
// depending on how noisy the RxD is, different values might work
better
// with a clean connection, values from 8 to 11 work
wire next_bit = (bit_spacing==4'd10);
always @(posedge clk or posedge rst)
if(rst)
      bit_spacing <= 4'b0000;
else
if(state==0)
      bit_spacing <= 4'b0000;
else
if(Baud8Tick)
      bit_spacing <= {bit_spacing[2:0] + 4'b0001} |
{bit_spacing[3], 3'b000};
always @(posedge clk or posedge rst)
if(rst)
      state <= 4'b0000;
else
if(Baud8Tick)
case(state)
      4'b0000: if(RxD_bit_inv) state <= 4'b1000;  // start bit
found?
      4'b1000: if(next_bit) state <= 4'b1001;  // bit 0
      4'b1001: if(next_bit) state <= 4'b1010;  // bit 1
      4'b1010: if(next_bit) state <= 4'b1011;  // bit 2
      4'b1011: if(next_bit) state <= 4'b1100;  // bit 3
      4'b1100: if(next_bit) state <= 4'b1101;  // bit 4
      4'b1101: if(next_bit) state <= 4'b1110;  // bit 5
      4'b1110: if(next_bit) state <= 4'b1111;  // bit 6
      4'b1111: if(next_bit) state <= 4'b0001;  // bit 7
      4'b0001: if(next_bit) state <= 4'b0000;  // stop bit
      default: state <= 4'b0000;
endcase
reg [7:0] RxD_data;
always @(posedge clk or posedge rst)
if(rst)
      RxD_data <= 0;
else
if(Baud8Tick && next_bit && state[3])
      RxD_data <= {~RxD_bit_inv, RxD_data[7:1]};
reg RxD_data_ready, RxD_data_error;
always @(posedge clk or posedge rst)
if(rst)
begin
      RxD_data_ready <= 0;
      RxD_data_error <= 0;
end
else
begin
      RxD_data_ready <= (Baud8Tick && next_bit && state==4'b0001 &&
~RxD_bit_inv);  // ready only if the stop bit is received
```

```
      RxD_data_error <= (Baud8Tick && next_bit && state==4'b0001 &&
RxD_bit_inv);  // error if the stop bit is not received
end
reg [4:0] gap_count;
always @(posedge clk or posedge rst) if(rst) gap_count<=0; else if
(state!=0) gap_count<=5'h00; else if(Baud8Tick & ~gap_count[4])
gap_count <= gap_count + 5'h01;
assign RxD_idle = gap_count[4];
reg RxD_endofpacket;
always @(posedge clk or posedge rst) if(rst) RxD_endofpacket<=0;
else RxD_endofpacket <= Baud8Tick & (gap_count==5'h0F);
endmodule
```

led_driver.v

```
module led_driver (iRSTN, iCLK, iDIG, iG_INT2, oLED);
input                        iRSTN;
input                        iCLK;
input          [9:0]  iDIG;
input                 iG_INT2;
output         [9:0]  oLED;

//=======================================================
//  REG/WIRE declarations
//=======================================================
wire                    [4:0]  select_data;
wire                signed_bit;
wire                    [3:0]  abs_select_high;
reg                       [1:0]  int2_d;
reg            [23:0] int2_count;
reg                   int2_count_en;

//=======================================================
//  Structural coding
//=======================================================
assign select_data = iG_INT2 ? iDIG[9:5] :  // +-2g resolution :
10-bit

(iDIG[9]?(iDIG[8]?iDIG[8:4]:5'h10):(iDIG[8]?5'hf:iDIG[8:4])); // +-
g resolution : 9-bit
assign signed_bit = select_data[4];
assign abs_select_high = signed_bit ? ~select_data[3:0] :
select_data[3:0];//the negitive number here is the 2's complement-1

assign oLED = int2_count[23] ? ((abs_select_high[3:0] == 3'h0) ?
10'h030 :

(abs_select_high[3:0] == 3'h1) ? (signed_bit?10'h020:10'h010) :

(abs_select_high[3:0] == 3'h2) ? (signed_bit?10'h060:10'h018) :

(abs_select_high[3:0] == 3'h3) ? (signed_bit?10'h040:10'h8) :

(abs_select_high[3:0] == 3'h4) ? (signed_bit?10'h0C0:10'hC) :

(abs_select_high[3:0] == 3'h5) ? (signed_bit?10'h080:10'h4) :

(abs_select_high[3:0] == 3'h6) ? (signed_bit?10'h180:10'h6) :

(abs_select_high[3:0] == 3'h7) ? (signed_bit?10'h100:10'h2) :

(abs_select_high[3:0] == 3'h8) ? (signed_bit?10'h300:10'h3) :

(signed_bit?10'h200:10'h1)):
                                                (int2_count[20] ?
10'h0 : 10'h3ff); // Activity
```

```
always@(posedge iCLK or negedge iRSTN)
      if (!iRSTN)
  begin
    int2_count_en<= 1'b0;
    int2_count <= 24'h800000;
  end
      else
      begin
            int2_d <= {int2_d[0], iG_INT2};

            if (!int2_d[1] && int2_d[0])
    begin
      int2_count_en    <= 1'b1;
          int2_count <= 24'h0;
        end
        else if (int2_count[23])
            int2_count_en    <= 1'b0;
    else
            int2_count <= int2_count + 1;
      end

endmodule
```

reset_delay.v

```
module      reset_delay(iRSTN, iCLK, oRST);
input           iRSTN;
input           iCLK;
output reg  oRST;

reg  [20:0] cont;

always @(posedge iCLK or negedge iRSTN)
  if (!iRSTN)
  begin
    cont     <= 21'b0;
    oRST     <= 1'b1;
  end
  else if (!cont[20])
  begin
    cont <= cont + 21'b1;
    oRST <= 1'b1;
  end
  else
    oRST <= 1'b0;

endmodule
```

<u>spi_controller.v</u>

```
module spi_controller (
                                        iRSTN,
                                        iSPI_CLK,
                                        iSPI_CLK_OUT,
                                        iP2S_DATA,
                                        iSPI_GO,
                                        oSPI_END,

                                        oS2P_DATA,

                                        SPI_SDIO,
                                        oSPI_CSN,

                                        oSPI_CLK);

`include "spi_param.h"

//=======================================================
//  PORT declarations
//=======================================================
//    Host Side
input                                iRSTN;
input                                iSPI_CLK;
input                                iSPI_CLK_OUT;
input       [SI_DataL:0]  iP2S_DATA;
input                                iSPI_GO;
output                                 oSPI_END;
output      reg [SO_DataL:0] oS2P_DATA;
//    SPI Side
inout                                SPI_SDIO;
output                                 oSPI_CSN;
output                                   oSPI_CLK;

//=======================================================
//  REG/WIRE declarations
//=======================================================
wire          read_mode, write_address;
reg           spi_count_en;
reg   [3:0]       spi_count;

//=======================================================
//  Structural coding
//=======================================================
assign read_mode = iP2S_DATA[SI_DataL];
assign write_address = spi_count[3];
assign oSPI_END = ~|spi_count;
assign oSPI_CSN = ~iSPI_GO;
assign oSPI_CLK = spi_count_en ? iSPI_CLK_OUT : 1'b1;
assign SPI_SDIO = spi_count_en && (!read_mode || write_address) ?
iP2S_DATA[spi_count] : 1'bz;

always @ (posedge iSPI_CLK or negedge iRSTN)
      if (!iRSTN)
      begin
            spi_count_en <= 1'b0;
            spi_count <= 4'hf;
      end
      else
      begin
            if (oSPI_END)
                  spi_count_en <= 1'b0;
            else if (iSPI_GO)
                  spi_count_en <= 1'b1;

            if (!spi_count_en)
            spi_count <= 4'hf;
            else
                  spi_count   <= spi_count - 4'b1;

    if (read_mode && !write_address)
              oS2P_DATA <= {oS2P_DATA[SO_DataL-1:0], SPI_SDIO};
      end

endmodule
```

spi_ee_config.v

```
module spi_ee_config (
                                              iRSTN,


                                              iSPI_CLK,

                                              iSPI_CLK_OUT,

                                              iG_INT2,
                                              oDATA_L,
                                              oDATA_H,
                                              SPI_SDIO,
                                              oSPI_CSN,
                                              oSPI_CLK);


`include "spi_param.h"

//=========================================================
//  PORT declarations
//=========================================================
//    Host Side
input                                  iRSTN;
input                                  iSPI_CLK, iSPI_CLK_OUT;
input                                  iG_INT2;
output reg [SO_DataL:0] oDATA_L;
output reg [SO_DataL:0] oDATA_H;
//    SPI Side
inout                                  SPI_SDIO;
output                                     oSPI_CSN;
output                                     oSPI_CLK;

//=========================================================
//  REG/WIRE declarations
//=========================================================
reg       [3:0]          ini_index;
reg           [SI_DataL-2:0] write_data;
reg           [SI_DataL:0]    p2s_data;
reg                     spi_go;
wire                    spi_end;
wire    [SO_DataL:0]     s2p_data;
reg     [SO_DataL:0]     low_byte_data;
reg                                 spi_state;
reg                     high_byte; // indicate to read the high or
low byte
reg                     read_back; // indicate to read back data
reg                     clear_status, read_ready;
reg     [3:0]           clear_status_d;
reg                     high_byte_d, read_back_d;
reg       [IDLE_MSB:0]   read_idle_count; // reducing the reading
rate

//=========================================================
//  Sub-module
//=======================================================
spi_controller u_spi_controller (
                                        .iRSTN(iRSTN),
                                        .iSPI_CLK(iSPI_CLK),

      .iSPI_CLK_OUT(iSPI_CLK_OUT),
                                        .iP2S_DATA(p2s_data),
                                        .iSPI_GO(spi_go),
                                        .oSPI_END(spi_end),

                                        .oS2P_DATA(s2p_data),

                                        .SPI_SDIO(SPI_SDIO),
                                        .oSPI_CSN(oSPI_CSN),

                                        .oSPI_CLK(oSPI_CLK));

//=======================================================
//  Structural coding
//=======================================================
// Initial Setting Table
always @ (ini_index)
      case (ini_index)
    0       : write_data = {THRESH_ACT,8'h20};
    1       : write_data = {THRESH_INACT,8'h03};
    2       : write_data = {TIME_INACT,8'h01};
    3       : write_data = {ACT_INACT_CTL,8'h7f};
    4       : write_data = {THRESH_FF,8'h09};
    5       : write_data = {TIME_FF,8'h46};
    6       : write_data = {BW_RATE,8'h09}; // output data rate : 50
Hz
    7       : write_data = {INT_ENABLE,8'h10};
    8       : write_data = {INT_MAP,8'h10};
    9       : write_data = {DATA_FORMAT,8'h40};
        default: write_data = {POWER_CONTROL,8'h08};
      endcase

always@(posedge iSPI_CLK or negedge iRSTN)
      if(!iRSTN)
      begin
            ini_index   <= 4'b0;
            spi_go            <= 1'b0;
            spi_state   <= IDLE;
            read_idle_count <= 0; // read mode only
            high_byte <= 1'b0; // read mode only
            read_back <= 1'b0; // read mode only
    clear_status <= 1'b0;
      end
      // initial setting (write mode)
      else if(ini_index < INI_NUMBER)
            case(spi_state)
                  IDLE : begin
                              p2s_data  <= {WRITE_MODE, write_data};
                              spi_go            <= 1'b1;
                              spi_state   <= TRANSFER;
                    end
                    TRANSFER : begin
                              if (spi_end)
                              begin
                        ini_index<= ini_index + 4'b1;
                                    spi_go           <= 1'b0;
                                    spi_state   <= IDLE;

                              end
                    end
                endcase
  // read data and clear interrupt (read mode)
  else
                case(spi_state)
                    IDLE : begin
                           read_idle_count <= read_idle_count + 1;

                              if (high_byte) // multiple-byte read
                           begin
                                p2s_data[15:8] <= {READ_MODE, X_HB};

                                read_back      <= 1'b1;
                              end
                           else if (read_ready)
                           begin
                                p2s_data[15:8] <= {READ_MODE, X_LB};

                                read_back      <= 1'b1;
                              end
                           else if (!clear_status_d[3]&&iG_INT2 ||
read_idle_count[IDLE_MSB])
                           begin
                                p2s_data[15:8] <= {READ_MODE,
INT_SOURCE};
                                clear_status   <= 1'b1;
          end

          if (high_byte || read_ready || read_idle_count[IDLE_MSB]
|| !clear_status_d[3]&&iG_INT2)
          begin
                                spi_go         <= 1'b1;
                                spi_state <= TRANSFER;
                              end

                           if (read_back_d) // update the read back
data
                           begin
                              if (high_byte_d)
                              begin
                                oDATA_H <= s2p_data;
                                oDATA_L <= low_byte_data;

                              end
                              else
```

```
                                   low_byte_data <= s2p_data;
                         end
                 end
                 TRANSFER : begin
                             if (spi_end)
                             begin
                                   spi_go           <= 1'b0;
                                   spi_state   <= IDLE;

                                   if (read_back)
                                   begin
                                         read_back <= 1'b0;
                                 high_byte <= !high_byte;
                                 read_ready <= 1'b0;

                               end
                               else
                               begin
              clear_status <= 1'b0;
              read_ready <= s2p_data[6];

                                 read_idle_count <= 0;
            end
                             end
                 end
           endcase

always@(posedge iSPI_CLK or negedge iRSTN)
      if(!iRSTN)
      begin
            high_byte_d <= 1'b0;
            read_back_d <= 1'b0;
            clear_status_d <= 4'b0;
      end
      else
      begin
            high_byte_d <= high_byte;
            read_back_d <= read_back;
            clear_status_d <= {clear_status_d[2:0], clear_status};
      end
endmodule
```