CUBESAT TELEMETRY DECODER SOFTWARE

A thesis submitted in partial fulfilment of the requirement for the degree of
B.Sc. (HONS) Electrical and Electronic Engineering
(ELECTRONICS AND COMPUTER SYSTEMS ENGINEERING)

By:

Rayan Mohamed Elmubarak Ali Imam

INDEX NO. 064028

Supervisor:

Dr. Tahani Abdalla Attia

TO
Department of Electrical and Electronic Engineering
Faculty of Engineering
University of Khartoum

July 2011
DECLARATION OF ORIGINALITY

I declare that this thesis entitled “Cube Satellite Telemetry Decoder Software” is my own work except as cited in the references. The thesis has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature: _________________________

Name: ____________________________

Date: _____________________________
ACKNOWLEDGMENT

I wish to express my gratitude to my supervisor, Dr. Tahani Abdalla Attia, for her constant insight guidance, support and encouragement. I’m much indebted to her; she took my hand through this project and thesis writing and had been always there for me.

I would also like to express my gratitude to Dr. Nadir Omer, U of K CubeSat Team Manager, and his team. They have always been caring and willing to help.

I’m grateful to all my teachers in the department and throughout my life.

I specially wish to express my gratitude to Dr. Sharief F. Babikir, Head of Electronics and Computer Systems Engineering Department, for supporting and advising me. I have benefited tremendously from studying under such able and knowledgeable teacher.

I wish to express my gratitude to my partner Sara Yousif for her generous and valuable discussions, excellent and authoritative comments and freely sharing her insights and information. We lived the happy times and overcame the difficulties together during this project.

My thanks are extended to my parents and my siblings for their unstinting support and encouragement.
ABSTRACT

Satellites are expensive to develop in both time and money. They cost millions of dollars and years of time. That made them monopolized by rich countries. Also, universities were far from this important industry. By introducing small satellites known as CubeSats, this monopolization was broken and universities and developing countries could participate in space industry.

In the past space missions never used up to date technology in their spacecraft due to the long development time; the used technology becomes old when the satellite is launched, CubeSats opened a wide door for testing new technologies in space missions before the technology becomes obsolete.

Any spacecraft must communicate with its ground station. It sends the data of the mission that it was made to do and data about its health. This data is usually sent encoded. The data is decoded at the ground station using decoder software. Such a decoder was built in this project.

First, the encoding used must be known. Since there was not defined encoding format, an encoding was developed taking into consideration all requirements and constraints of CubeSats. Then the decoder software was built based on this encoding.

The design was done using Object Oriented Design. The system was divided into six subsystems: Decoding, Alarm, storage, options Chart Diagrams and GUI Display.

The software was written using Java™. There were problems with libraries that draw charts but they were solved.

The project fulfilled the objectives states for it. The software is working well and with good performance.
المستخلص

تعتبر الأقمار الصناعية مجالًا مطلقة من حيث المال والوقت. فهي تشمل مليارات الشركات وسنين طويلة من الإعداد، مما يجعلها تحت خيّرًا على الدول الفنية، حيث أن الفاعلية أيضًا تزداد بعيدة عن هذه الصناعة. لكن بظهور نوع من الأقمار الصناعية - يسمى الأقمار المتنقلة أو الطائرات - ظهر هذا الإحتجاز، وأصبح بإمكان الجامعات والدول النامية المشاركة في صناعة الفضاء.

في الماضي وبسبب الزمن العديد الذي تستغرقه مراحل إعداد الأقمار الصناعية، فإن التكنولوجيا المستخدمة في صناعة الفضاء غيرت فعلً إقامة فعالة عند إعلان القمر. لكن وينزل تقنية الأقمار المتنقلة فأنه أصبح بالإمكان اختيار التقنيات الجديدة في مصانع فضاء فلسطين قبل أن تشير تقنيات غريبة.

أي مركبة فضائية إيلاءها وجود اتصال ببعضها الأرضية. فهي يجب أن ترسل البيانات التي ارسلها لتجميعها، بدلاً من أن ترسل بيانات عن سمعتها الوظيفية. هذه البيانات غالباً يتم ترتيبها ومن ثم إرسالها.

وفي الحقيقة الأرضية يتم من هذه الترمينز، هذه الترمينز يتم وظيفة بواسطة برنامج. في هذا المشروع، تم بناء برنامج يقوم بذلك.

لضمان ذلك الترمينز يجب أن يكون الترمينز معروفًا. على في هذا المشروع لم يكن هناك ترمينز معروف. لذلك تم أولاً تطوير ترمينز برامجي قوي ومتطلبات الأقمار المتنقلة. ومن ثم بنى برنامج على أساس هذا الترميز.

تم تصميم البرنامج باتباع منهجية البرمجة الموجهة نحو الغرض. فتم النطاق لستة نظم فرعية مع: تلك الترمينز، الخرائط، الإحصاء، رسم الخرائط، وواجهة المستخدم.

تم التصميم باستخدام لغة جافا. خذتهم هناك مشاكل في المختبرات التي ترسم الخطط، وفقًا تم حلها.

البرنامج استوفي أهدافه المحددة له ويعمل ويعمل جيد.
# TABLE OF CONTENTS

1. Introduction .................................................................................................................. 1  
   1.1. Overview ................................................................................................................. 1  
   1.2. Problem Definition ................................................................................................. 1  
   1.3. Literature Review ................................................................................................. 3  
   1.4. Objectives ............................................................................................................. 3  
   1.5. Methodology ......................................................................................................... 4  
   1.6. Thesis layout ......................................................................................................... 4  

2. Cubesat History and theoretical Background ................................................................ 5  
   2.1. CubeSat Constraints ............................................................................................ 5  
   2.1.1 Limited Communication Window .................................................................... 5  
   2.1.2 Limited Power Budget .................................................................................... 6  
   2.1.3 Data Size ........................................................................................................... 6  
   2.2. Encoding/decoding Requirements ....................................................................... 6  
   2.2.1 Short Reliable Communications Bursts must be used ..................................... 6  
   2.2.2 A Simple, Easily Recognizable Beacon Implementation ............................... 7  
   2.3. Cubesat and Universities ..................................................................................... 7  
   2.3.1 PolySat .............................................................................................................. 7  
   2.3.2 AAU1 CubeSat .................................................................................................. 7  
   2.3.3 CUTE-1 ............................................................................................................ 8  
   2.3.4 QuakeSat-1 ...................................................................................................... 8  
   2.3.5 CAPE1 ................................................................................................................ 8  
   2.4. Cubesat Communication System ......................................................................... 9  
   2.4.1 Telemetry Subsystem ...................................................................................... 10  
   2.5. Cubesat Ground Stations ..................................................................................... 10  
   2.5.1 Ground Station Hardware ............................................................................... 11  
   2.5.2 Ground Station Software ................................................................................ 11  

vi
4. Results and Discussion........................................................................................................36
   4.1. System Developed........................................................................................................36
   4.2. How to use the software: ..........................................................................................38
       4.2.1 Exceptions and Error Handling: .......................................................................45
   4.3. Software Testing: ......................................................................................................47
       4.3.1 Discussion: ........................................................................................................49

5. Conclusion, comments and recommendations ..............................................................52
   5.1. Conclusion ................................................................................................................52
   5.2. Future Work ..............................................................................................................53
       5.2.1 Interface with other ground station software ..................................................53
       5.2.2 A web server to allow users to upload packets on it .....................................53

References ..........................................................................................................................54

Appendix A: Guide to the Software Engineering Body of Knowledge (SWEBOK) ...... A-1
Appendix B: Packet Format................................................................................................. B-1
Appendix C: Code ............................................................................................................... C-1
List of Tables

Table 2-1 Low Earth Orbit Characteristics ................................................................. 5
Table 2-2 Morse Code .................................................................................................. 14
Table 2-3 NanoSail-D Beacon Packet Definition ...................................................... 17
Table 2-4 OOREOS Beacon Packet Definition ......................................................... 17
Table 3-1 Packet Format .............................................................................................. 24
Table 4-1 Unit Test Results ......................................................................................... 47
Table B-1 Beacon Parts Details .................................................................................. B-1
Table B-2 Payload Request Lookup Table ................................................................. B-2
Table B-3 Cubesat Status Lookup Table ..................................................................... B-2
Table B-4 Transmission Modes Lookup Table ........................................................... B-2
Table B-5 Battery Voltage Lookup Table .................................................................... B-2
Table B-6 Temperature and Bus Voltage Lookup Table ............................................. B-3
LIST OF FIGURES

Figure 1.1 Problem Definition Diagram .................................................................................. 2
Figure 2.1 Communication Window In Terms of AOS and LOS ............................................. 5
Figure 2.2 The Constraints Affecting the Encoding and Sending of Beacons ......................... 6
Figure 2.3 CP1 ............................................................................................................................. 7
Figure 2.4 Time Available Per Day To Conduct a CubeSat Before (Left) and After (Right) GENSO ........................................................................................................................................ 10
Figure 2.5 The Main Elements of a Telemetry Subsystem ....................................................... 10
Figure 2.6 Ground Segment Components ............................................................................. 11
Figure 2.7 Morse Code Tree ..................................................................................................... 14
Figure 2.8 AX.25 U And S Frame Construction ...................................................................... 15
Figure 2.9 AX.25 Information Frame Construction .................................................................. 15
Figure 2.10 MixW Screenshot .................................................................................................. 18
Figure 2.11 MULTIPSK Screenshot ........................................................................................ 19
Figure 2.12 The Telemetry Decoder Software For OOREOS Beacon ...................................... 19
Figure 2.13 The Telemetry Decoder Software For NanoSail-D Beacon .................................. 20
Figure 3.1 Software Development Methodology ...................................................................... 22
Figure 3.2 Packet Format ......................................................................................................... 22
Figure 3.3 Subsystems of the Ground Station Decoding Software ........................................... 26
Figure 3.4 Sound Decoder and Telemetry Decoder System’s Flow Chart ................................ 27
Figure 3.5 Telemetry Decoder Software Context Diagram ...................................................... 28
Figure 3.6 Use Case diagram (1) ............................................................................................. 29
Figure 3.7 Use Case Diagram (2) ............................................................................................. 29
Figure 3.8 Class Diagram ......................................................................................................... 31
Figure 3.9 Sequence Diagram .................................................................................................. 31
Figure 3.10 Activity Diagram ................................................................................................... 32
Figure 4.1 Start Tap .................................................................................................................. 36
Figure 4.2 Beacon Tap ............................................................................................................. 37
Figure 4.3 Options Tap ............................................................................................................. 38
Figure 4.4 How to Decode a File ............................................................................................. 39
Figure 4.5 Save As Dialog Box ............................................................................................... 40
Figure 4.6 The Encoded (File Upper) and the Decoded File saved as “Save As.txt” (Lower) ... 40
Figure 4.7 Chart Display: In Window (Upper) In Separate Window (Lower) ......................... 41
Figure 4.8 Chart Actions: Copy, Save As, Properties, Zoom and Print ................................. 42
Figure 4.9 Multiple Files Import ......................................................................................... 42
Figure 4.10 Three Files Imported (Upper) and The Database After Saving Them ............... 43
Figure 4.11 Some Help Tips ................................................................................................. 43
Figure 4.12 Alarm Display ................................................................................................. 44
Figure 4.13 Temperature Unit: Celsius (Left) Fahrenheit (Right) ....................................... 44
Figure 4.14 Trying to Save Without Importing a File ......................................................... 45
Figure 4.15 Trying to Import a Non-text File ..................................................................... 46
Figure 4.16 Trying to Import a Text Tile That Doesn't Belong to The CubeSat .................. 46
Figure 4.17 The Encoded Packet (Upper) and The Database After Decoding The Packets (Lower) .................................................................................................................. 48
Figure 4.18 UofK Ground Station and Cubesat Prototype ................................................ 48
Figure 4.19 Encoding Options Balance Chart ................................................................. 50
Figure 4.20 Text Files Can Easily be Further Processed .................................................... 51
Figure A-1 SWEBOK Knowledge Areas ......................................................................... A-2
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3U</td>
<td>Three units CubeSat. Where a unit is 10<em>10</em>10 cm</td>
</tr>
<tr>
<td>AAU1</td>
<td>Aalborg University first CubeSat</td>
</tr>
<tr>
<td>ADC</td>
<td>Analogue to Digital Converter</td>
</tr>
<tr>
<td>AOS</td>
<td>Acquisition of signal</td>
</tr>
<tr>
<td>ARRL</td>
<td>American Radio Relay League</td>
</tr>
<tr>
<td>CP1</td>
<td>Cal Poly’s first satellite</td>
</tr>
<tr>
<td>CubeSat</td>
<td>Cube Satellite</td>
</tr>
<tr>
<td>CUTE-1</td>
<td>Cubical Titech Engineering CubeSat</td>
</tr>
<tr>
<td>CW</td>
<td>Continuous Wave or Continuous Waveform</td>
</tr>
<tr>
<td>DTMF</td>
<td>Dual Tone Multi-Frequency</td>
</tr>
<tr>
<td>FM</td>
<td>Frequency Modulation</td>
</tr>
<tr>
<td>FSK</td>
<td>Frequency Shift Keying</td>
</tr>
<tr>
<td>GENSO</td>
<td>Global Educational Network for Satellite Operators</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>kB</td>
<td>Kilo Byte</td>
</tr>
<tr>
<td>km</td>
<td>Kilo meter</td>
</tr>
<tr>
<td>Km/h</td>
<td>Kilo meter per hour</td>
</tr>
<tr>
<td>LEO</td>
<td>Low Earth Orbit</td>
</tr>
<tr>
<td>LOS</td>
<td>loss of signal</td>
</tr>
<tr>
<td>LST</td>
<td>Low-Speed Telemetry</td>
</tr>
<tr>
<td>MHz</td>
<td>Mega Hertz</td>
</tr>
<tr>
<td>Mile/h</td>
<td>Mile per hour</td>
</tr>
<tr>
<td>mV</td>
<td>Milli Watt</td>
</tr>
<tr>
<td>OOD</td>
<td>Object-Oriented Design</td>
</tr>
<tr>
<td>OPAL</td>
<td>Orbiting Picosatellites Automated Launcher</td>
</tr>
<tr>
<td>PolySat</td>
<td>Cal Poly’s Cubesat Project</td>
</tr>
<tr>
<td>P-POD</td>
<td>Poly Picosatellite Orbital Deployer</td>
</tr>
<tr>
<td>STCP</td>
<td>Stream Control Transmission Protocol</td>
</tr>
<tr>
<td>SWEBOK</td>
<td>Software Engineering Body of Knowledge</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TNC</td>
<td>Terminal Node Controller</td>
</tr>
<tr>
<td>VLF</td>
<td>Very Low Frequency</td>
</tr>
</tbody>
</table>
Chapter 1

1. Introduction

1.1. Overview

Cube satellite is a cubic satellite that has a volume of 10*10*10 cm$^3$ and maximum of 1.33 kg mass. It is commonly referred to as CubeSat. CubeSat idea started at the late 1990s. The standard of CubeSats was started as a joint project between Cal Poly University and Stanford University in 1999. The first CubeSat launch was in 2003.

Telemetry is a technology that allows remote measurement and reporting of information. The word was derived from the Greek roots tele (remote) and metron (measure).

A decoder is a device or software which does the reverse of an encoder, undoing the encoding so that the original information can be retrieved.

A communication subsystem between the CubeSat and the ground station is needed. The primary goal of it is to provide a link to transfer data to ground station from the CubeSat. Data is collected by the payload and sensors on the satellite and then sent to the ground station. There, it will be handled and displayed to users in order to make use of them. This handling of data at the ground station is done using telemetry decoder software.

1.2. Problem Definition

After the CubeSat is launched in space, data of the CubeSat mission, also known as payload, must be collected and sent to earth. Also, operators on earth must know something about the CubeSat status, which is known as housekeeping data, to know whether the CubeSat is healthy or not. Actually, the CubeSat processor accumulates these data and converts them into a stream of binary numbers. This numerical string is encoded. A transmitter then sends the signal to the ground station when it is in a position that allows communication. The ground station receives the data signal and demodulates it. Then it is entered to the ground station computer. Software is needed to decode the signal and display its data to the users in the ground station. The user may want to view reports about the satellite and display it in some format. If the satellite is not healthy or there is data that needs extra attention from the ground station operators, a system must alarm the operators for this data. For all that, and more, computer software is needed in the ground station to handle these jobs Figure 1.1 shows the Problem Definition as a diagram.
Without telemetry the satellite is just an object in space that operators can’t make use of and can’t know anything about it.

Figure 1.1 Problem Definition Diagram
1.3. Literature Review

The CubeSat standard started as a joint project between Cal Poly State University and Stanford University in 1999. Cal Poly Professor Dr. Jordi Puig-Suari and Stanford Professor Bob Twiggs imagined multiple 10 cm cubes in a jack-in-the-box type launcher after their experience building and deploying Picosatellites from the Orbiting Picosatellites Automated Launcher (OPAL). While many criticized this standard as being “too small to do anything,” universities and industry have shown that a lot of science and data collection is possible with these Picosatellites [1].

The main advantage of CubeSats is the dramatically reduced development and launch cost as well as short development time. Projects usually take 5 months to 2.5 years from idea to launch. An implication of the reduced development time frames is that CubeSats are more likely to integrate up-to-date technology. The costs for such spacecrafts are in the range of tens to hundreds of thousands of Dollars, due to low cost of launch and hardware. Launch cost is effectively reduced by sharing the launch vehicle with a primary payload. This enables universities, small companies and developing countries to participate in the international business of space technology [2].

Presently, the CubeSat Project is an international collaboration of over 40 universities, high schools and private firms developing Picosatellites containing scientific, private and government payloads [3].

Since each CubeSat sends its data encoded, then each CubeSat must have its telemetry decoder software.

1.4. Objectives

The objectives of this project were:

- To decode the data accumulated by the CubeSat and was sent to the ground station.
- Display the data to the user using a Graphical User Interface.
- Doing some processing to data if needed.
- Save data properly so that users can view them any time.
- Some data may need to be displayed in graphs or charts. The software is to be capable of doing so.
- Alarm users if there is something wrong.
1.5. Methodology

The methodology followed in doing the project was:

- The signal was assumed to be encoded using an encoding that was developed by us.
- This data was received in the ground station and entered to a computer through the sound card.
- A sound decoder program was used to convert the sound to text.
- Telemetry decoder software was developed to decode the encoded text and then display it.
- This program was constructed using Java programming language.
- The data decoded was saved in a text file that serves as a database.

1.6. Thesis layout

The thesis was organized in five chapters. This is the introduction chapter. It gives general idea about the project, its objectives, methodology and introduces the next chapters. The rest of the thesis is organized as follows:

Chapter two covers the theory behind this project. It also contains literature review about some universities that launched CubeSats and their CubeSats communication system.

Chapter three shows all steps of the software development, starting from the assumptions made, requirements, design, coding and ending with the test plan.

Chapter four is devoted to the software developed. Also the testing of the software is shown. The software and its capabilities are discussed too.

Chapter five concludes the thesis by summarizing the project and the work that was done. Then the recommended future work is listed.

References are listed after chapter five concluding by that the body of the thesis.

The appendices are at the end of the thesis.
CHAPTER 2

2. Cubesat History and theoretical Background

2.1. CubeSat Constraints

CubeSats have many constraints due to its small size and the nature of launch. These constraints affect all subsystems of the CubeSat. Here the ones affecting encoding and decoding are shown.

2.1.1 Limited Communication Window

The geometry of a satellite’s orbit defines the schedule of when, and for how long, the satellite is able to communicate with a fixed ground station. CubeSats are typically launched in what is called a low-earth orbit (LEO). LEOs are characterized by their short range, high orbital velocity and non-geosynchronous nature [4]. Table 2-1 shows LEO characteristics.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value (US)</th>
<th>Value (metric)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elevation</td>
<td>400-435 mile</td>
<td>650-700 km</td>
</tr>
<tr>
<td>Orbital Velocity</td>
<td>≈17,000 mile/h</td>
<td>27,000 km/h</td>
</tr>
</tbody>
</table>

The communication window for a satellite is the amount of time that a fixed ground station can transmit to and receive signals from a satellite. The duration of this window is determined by the orbital parameters, and is defined as the length of time between AOS and LOS as in Figure 2.1 [4].

![Figure 2.1 Communication Window In Terms of AOS and LOS](image)
2.1.2 Limited Power Budget

The primary power source of the satellite is a pack of batteries that are charged by solar panels. These batteries must support the full functionality of the satellite, and thus power consumption must be strictly budgeted between satellite subsystems.

Communications is responsible for consuming 75-88% of available power budget. Thus every effort should be made to reduce the power consumption of this subsystem [4].

2.1.3 Data Size

Another constraint is the size of data that is to be sent. The amount of data that should be sent can’t be reduces, but suitable encoding can make the packet sent the minimum in size.

Figure 2.2 shows the constraints affecting the encoding and sending of data.

![Figure 2.2 The Constraints Affecting the Encoding and Sending of Beacons](image)

2.2. Encoding/decoding Requirements

In order to encode and decode data within the constraints above, the following derived requirements must be followed to encode and send data

2.2.1 Short Reliable Communications Bursts must be used

The communications window restricts telemetry activities to many short bursts over the course of one day. Due to the short duration of availability, a high data rate is necessary. The maximum data size, together with the limited number of passes and variable pass length, will determine a minimum baud rate required for data transmission. Higher data rates are acceptable as long as they are reliable [4].
2.2.2 A Simple, Easily Recognizable Beacon Implementation

It was found, from previous missions of other universities, that CubeSats with audible beacons were much easier to locate and contact. After initial deployment, it would be extremely difficult to locate a satellite using only short bursts, less than one second, of encoded satellite telemetry. Thus, an audible beacon is required [4].

2.3. Cubesat and Universities

Many universities succeeded to build and launch CubeSats. Some of them are shown in this subsection concentrating on the communications subsystem.

2.3.1 PolySat

In 1999, a team designed, constructed, and tested CP1, Cal Poly’s first satellite (Figure 2.3). CP1 communicated on amateur radio frequencies using a combination of Morse Code and Dual Tone Multi-Frequency (DTMF) to encode data. Morse Code is used to identify transmissions while allowing operators to tune to the correct frequency, and DTMF data is sent at 15 characters per second. Despite its relative simplicity, the CP1 communication system was highly efficient [5].

![Figure 2.3 CP1](image)

2.3.2 AAU1 CubeSat

It was developed by students of Aalborg University in Denmark. The system used a 9600 baud rate for communications. This satellite used a Mobitex packet encoding scheme underneath standard AX.25. These packets contained telemetry data, but could not be decoded by regular amateur radio operators due to the Mobitex packet encoding. This satellite beaconed every two minutes if the on-board computer was not functioning, and every four minutes in a low battery situation. Ground stations only received about 1 kB of data [2] [1] [6].
2.3.3 CUTE-1

CUTE-1 is a Japanese CubeSat developed by sixteen graduate and undergraduate students at Tokyo Institute of Technology Laboratory for Space Systems. One of the primary mission goals was to test two different implementations of downlink communication protocols [2] [7].

The communications experiment consisted of changing the modulation schemes between standard AX.25 and SRL, a new protocol developed for the project. SRL includes error correction and can correct for up to 3 erroneous bits per 32 byte packet [8].

The communications subsystem included a 1200 baud FM transmitter and a CW transmitter that operates almost continuously, making it easy to track this satellite. The spacecraft is operational since its launch in June 2003.

2.3.4 QuakeSat-1

Stanford University and QuakeFinder collaborated on this 3U CubeSat designed to measure signal amplitudes in the VLF range. This satellite beamed a short 200 byte packet every 10 seconds. The downlink protocol uses a derivative of the Pacsat protocol, especially well suited for satellite communications [9].

QuakeSat-1 used two ground stations linked via the internet to download more data. The first ground station, located at Stanford, started a data downlink session, and the other ground station in Alaska continued receiving the data after the satellite went below the horizon at Stanford. This configuration allowed 423 MB of data downloaded from this triple CubeSat, the most from any CubeSat in space as of June 2011 [1].

2.3.5 CAPE1

It was built by the University of Louisiana. This satellite transmitted two beacons, a 30 second CW beacon followed by a short 9600 baud packet, repeating once per minute. Nobody has ever decoded a 9600 baud packet, including the CAPE1 ground station, leading the team to surmise that there was some problem with the packet format. Luckily, most of the data contained in the packet also existed in the CW portion of the beacon, so the loss of the packet did not affect the satellite health knowledge. This satellite died 4 months after launch, but recently revived itself in March 2008. It beacons intermittently [1].
2.4. Cubesat Communication System

The communications subsystem, also known as telemetry, tracking and command subsystem, is the interface between the satellite and the earth, or the satellite and other satellites. With its help data can be send from and towards the spacecraft [10].

Communication with the spacecraft takes place over two types of links:

- The uplink carries commands from a ground station to the spacecraft.
- The downlink carries data from the spacecraft to a ground station. It consists of two different types of information. One is the data generated by the payload. The other data is information about the spacecrafts vital characteristics, which is called housekeeping data. This is the total of all information for a specific time gathered by various sensors, e.g. temperature, voltage etc. [11].

An amateur-band communications system is chosen by almost all CubeSats. Amateur radio (also, ham radio) is the licensed and private use of designated radio bands, for purposes of private recreation, non-commercial exchange of messages, experimentation, self-training, and emergency communication. Amateur radio, like other regulated radio services, operates under rules that limit the maximum power and the technical and operational characteristics of transmissions.

Amateur-band is chosen for three reasons. First, there is a lack of regulatory constraints in attaining the frequency bands. Second, commercial off the shelf parts can be used for the flight system. Third, which is important to us as telemetry decoder software, other ground stations can be coordinated such that there is more ground station access [12].

While the current CubeSat operations situation does not require automated operations, the Global Educational Network for Satellite Operators (GENSO) will soon greatly increase the potential operations time. GENSO is a project which promises increased educational satellite (i.e. CubeSats) communication time by connecting earth stations all over the world through the internet [13].

Figure 2.4 shows the contact Time Available Per Day To Conduct a CubeSat Before (Left) and After (Right) GENSO.
2.4.1 Telemetry Subsystem

The function of the Telemetry subsystem is to monitor various spacecraft parameters such as voltage, temperature and equipment status, and to transmit the measured values to ground. The telemetry data is analyzed and used for routine operational and failure diagnostic purposes [11]. Figure 2.5 shows The Main Elements of a Telemetry Subsystem.

![Figure 2.5 The Main Elements of a Telemetry Subsystem](image)

2.5. Cubesat Ground Stations

A ground system supports the space segment. To support spacecrafts and their payloads, the ground system must monitor their health and track them. The ground system controls the spacecraft and its instruments or payloads by transmitting command data to the spacecraft. The ground system uses spacecraft housekeeping telemetry and mission data to carry out these functions [14].
The main functions performed at a ground station in support of an operational spacecraft are highly complex functions, usually involving the following tasks [15]:

i. Tracking to determine the position of the satellite in orbit.
ii. Telemetry operations to acquire and record satellite data and status.
iii. Data processing operations to present all the engineering and scientific data in the formats required for the successful progress of the mission.
iv. Commanding operations to interrogate and control the various functions of the satellite.
v. Controlling operations to determine orbital parameters, to schedule all satellite passes and to monitor and load the on-board computer.
vi. Voice and data links to other worldwide ground stations and processing centers.

2.5.1 Ground Station Hardware

The main hardware components of a ground station are an antenna, a transmit-receive system, data recorder(s), computer(s) and their peripherals and control consoles. This is a basic list that does not significantly change with the type of spacecraft being controlled, and therefore applies equally to Space Science, Earth Observation, Remote Sensing and Telecommunications. It is also as applicable to low earth orbits as it is to geostationary orbits or interplanetary orbits [15].

The principle layout of the ground station is given in Figure 2.6.

Figure 2.6 Ground Segment Components
2.6. Ground Station Software

There are three major areas of software that run at a ground station. These are the pre-pass, real-time and post-pass software [15].

2.6.1 Pre-pass software

As its name implies, pre-pass software is required to run in advance of the pass of an active spacecraft over a ground station.

2.6.2 Real-time software

Real-time software operates during the period when the spacecraft is visible from the associated ground station. It includes computer control of the antenna tracking, command uplink and verification, data reception and status checking of all critical system parameters.

Data reception

During a satellite's ground-contact, data is transmitted to the ground over a telemetry link. The contents of the computer memory form part of the data that is transmitted continuously over a low-speed telemetry (LST) channel, usually at the rate of a few kbps. The rest of the frame contains command verification data and satellite housekeeping data. All the LST data transmitted during the pass is received and processed in real time.

Status checking

Both before and after the uplink of commands to the satellite, real-time data from the housekeeping telemetry frames are received at the ground station. These data contain parameters that are critical to both the health and operation of the satellite. In general, these data are a mixture of status flags, e.g. Subsystem on/off, and engineering parameter values. This data must be checked and analyzed immediately. The status checking software trips this critical data out of the main telemetry stream, converts it to engineering values, compares it with in-limits criteria and displays it on control consoles for immediate notification and if necessary, action, by subsystem engineers.

2.6.3 Post-pass software

The immediate post-pass tasks include extraction of housekeeping and science/technology data for quality control and health assessment, data processing, orbit determination and data analysis [15].
**Health assessment**

After a pass, the ground software extracts and processes the engineering data in order to assess the success of the observations' or experiments' schedule on the satellite, and it supplies the instrument and spacecraft support teams with the information they need to monitor the spacecraft's health and performance.

**Data processing**

The task of data processing, other than that contained within the Status Checking and Health Assessment described above, includes two other processings. The first is further processing of housekeeping data to provide subsystem engineers with all their products, such as the full range of temperatures, pressures, Voltage/current levels. The second is processing of science/technology data from the raw telemetry. In both cases telemetry data are demultiplexed, telemetry is quality checked and flagged, time-tagged and converted. The data are then passed on to the specialist engineers and scientists for further detailed analysis.

**Data analysis**

Data analysis software often accounts for the largest share of the budget for software development. It encompasses the software for analysis of trends in the engineering data as well as the complete analysis of all the science/technical data. It includes all the analysis and graphics facilities, as well as the often worldwide distribution of data to the users.

**2.7. Cubesat Telemetry Encoding**

Whether the beacon was CW or packet, the encoding goes through two levels. First the data is ordered as text. This text is either converted directly to Morse Code then sent as CW, or packetized using AX.25 protocol in Packets then sent by one of the frequency Shift keying methods.

**2.7.1 Morse Code**

Morse code is a method of transmitting textual information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer. The International Morse Code encodes the Roman alphabet, the Arabic numerals and a small set of punctuation and procedural signals as standardized sequences of short and long signals called "dots" and "dashes" respectively, or "dits" and "dahs". Each character is represented by a unique sequence of dots and dashes. The duration of a dash is three times the duration of a dot. Each dot or dash is followed by a short silence, equal to the dot duration [16]. Table 2-2 and Figure 2.7 show Morse
code as a table and as a tree respectively. In CubeSats encoding it is preferred to use the upper
letters of the tree because they consume less power in transmission.

Table 2-2 Morse Code

<table>
<thead>
<tr>
<th>A</th>
<th>..-</th>
<th>M</th>
<th>--</th>
<th>Y</th>
<th>..-</th>
<th>6</th>
<th>..-..</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>...</td>
<td>N</td>
<td>.-</td>
<td>Z</td>
<td>--.</td>
<td>7</td>
<td>--.</td>
</tr>
<tr>
<td>C</td>
<td>..-.</td>
<td>O</td>
<td>---</td>
<td>Ä</td>
<td>..-</td>
<td>8</td>
<td>----</td>
</tr>
<tr>
<td>D</td>
<td>..</td>
<td>P</td>
<td>--</td>
<td>Ö</td>
<td>--</td>
<td>9</td>
<td>----</td>
</tr>
<tr>
<td>E</td>
<td>.</td>
<td>Q</td>
<td>--</td>
<td>Ü</td>
<td>..-</td>
<td>.</td>
<td>..-.</td>
</tr>
<tr>
<td>F</td>
<td>..-</td>
<td>R</td>
<td>--</td>
<td>Ch</td>
<td>----</td>
<td>,</td>
<td>..--</td>
</tr>
<tr>
<td>G</td>
<td>..</td>
<td>S</td>
<td>..</td>
<td>0</td>
<td>----</td>
<td>?</td>
<td>..--</td>
</tr>
<tr>
<td>H</td>
<td>...</td>
<td>T</td>
<td>--</td>
<td>1</td>
<td>----</td>
<td>!</td>
<td>..-</td>
</tr>
<tr>
<td>I</td>
<td>..</td>
<td>U</td>
<td>..</td>
<td>2</td>
<td>----</td>
<td>:</td>
<td>----</td>
</tr>
<tr>
<td>J</td>
<td>---</td>
<td>V</td>
<td>..</td>
<td>3</td>
<td>----</td>
<td>“</td>
<td>..-</td>
</tr>
<tr>
<td>K</td>
<td>..</td>
<td>W</td>
<td>..</td>
<td>4</td>
<td>----</td>
<td>‘</td>
<td>----</td>
</tr>
<tr>
<td>L</td>
<td>..-</td>
<td>X</td>
<td>..</td>
<td>5</td>
<td>.....</td>
<td>=</td>
<td>..-</td>
</tr>
</tbody>
</table>

Figure 2.7 Morse Code Tree

2.7.2 Continuous Wave

A continuous wave or continuous waveform (CW) is an electromagnetic wave of a constant
amplitude and frequency. Continuous wave is also the name given to a method of radio
transmission, in which a carrier wave is switched on and off. Information is carried in the
varying duration of the on and off periods. In CubeSats CW is used to send Morse code.
2.7.3 AX.25 Protocol

AX.25 is the Link Access Protocol for Amateur Packet Radio. It’s used by amateur radio operators to send packets. The amateur radio community needed to define a protocol that can accept and reliably deliver data over a variety of communications links between two signaling terminals [17].

Frame Structure

Link layer packet radio transmissions are sent in small blocks of data, called frames. There are three general types of AX.25 frames:

a) Information frame (I frame);

b) Supervisory frame (S frame); and

c) Unnumbered frame (U frame).

Each frame is made up of several smaller groups, called fields. Figure 2.8 and Figure 2.9 illustrate the three basic types of frames.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Address</th>
<th>Control</th>
<th>Info</th>
<th>FCS</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111110</td>
<td>112/244 Bits</td>
<td>8/16 Bits</td>
<td>N*8 Bits</td>
<td>16 Bits</td>
<td>01111110</td>
</tr>
</tbody>
</table>

*Figure 2.8 AX.25 U And S Frame Construction*

<table>
<thead>
<tr>
<th>Flag</th>
<th>Address</th>
<th>Control</th>
<th>PID</th>
<th>Info</th>
<th>FCS</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111110</td>
<td>112/244 Bits</td>
<td>8/16 Bits</td>
<td>8 Bits</td>
<td>N*8 Bits</td>
<td>16 Bits</td>
<td>01111110</td>
</tr>
</tbody>
</table>

*Figure 2.9 AX.25 Information Frame Construction*

- The Info field exists only in certain frames
- FCS is the Frame Check Sequence field
- PID is the Protocol Identifier field

Each field is made up of an integral number of octets (8-bit byte of binary data) and serves a specific function.
2.7.4 Frequency Shift keying

Frequency-shift keying (FSK) is a frequency modulation scheme in which digital information is transmitted through discrete frequency changes of a carrier wave. It was used in the past for telephones. Now it’s used in some countries for emergency calls. It’s used extensively by CubeSats.

2.7.5 Encoding Examples:

In this subsection three CubeSats encodings are shown as examples.

UIUC-ION Beacon Format:

There are three types of beacon. They are sent using AX.25 protocol. The format is shown below (%i is replaced by 2-digit numbers) [18]:

Power up beacon:
“UIUC-ION: University of Illinois Cubesat Satellite - C[%i %i %i] - Greetings Earthlings!”

Main beacon:
“UIUC-ION: C[%i %i %i] T[%i] B1[%i %i] B2[%i %i] U[%i %i] SB[%i %i] S[%i %i %i %i %i] M[%i %i %i %i]”

Power down beacon
“UIUC-ION: University of Illinois Cubesat Satellite - C[%i %i %i] - See you on the flip side!”

NanoSail-D Beacon Format

The NanoSail-D sends an AX.25 packet every 10 seconds. The data portion of a standard AX.25 packet is 64 bytes long. The beacon packet contains only standard ASCII characters. Each character represents a HEX value. The 64 byte long data packet is divided in fields with fixed sizes as described in Table 2-3. The voltage and temperature are calculated by substituting the received values in some equations [19]

O/OREOS Beacon Encoding:

O/OREOS beacon sends an AX.25 packet every 5 seconds. The data portion of a standard AX.25 packet is 64 bytes long, although only the first 62 bytes are considered for the O/OREOS packet. The beacon packet contains only standard ASCII characters. Each character represents a HEX. The 62 byte long data packet is divided in fields as described in Table 2-4. The voltage and temperatures are calculated by substituting the received values in equations [20]
# Chapter 2

## Table 2-3 NanoSail-D Beacon Packet Definition

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (Bytes)</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Website</td>
<td>13</td>
<td>NanoSailD.org</td>
<td>Characters</td>
</tr>
<tr>
<td>Reserved</td>
<td>1</td>
<td>&lt;space&gt;</td>
<td>none</td>
</tr>
<tr>
<td>Software Version</td>
<td>2</td>
<td>Software version</td>
<td>140</td>
</tr>
<tr>
<td>Burn Time</td>
<td>2</td>
<td>Burn time</td>
<td>15</td>
</tr>
<tr>
<td>Start Phase</td>
<td>4</td>
<td>Start phase</td>
<td>0</td>
</tr>
<tr>
<td>Current Phase</td>
<td>4</td>
<td>Current phase</td>
<td>30 second increments</td>
</tr>
<tr>
<td>SwitchState</td>
<td>2</td>
<td>Real-time state of switch</td>
<td>boolean</td>
</tr>
<tr>
<td>PanelDeployStatus</td>
<td>2</td>
<td>Status of panel deployment</td>
<td>boolean</td>
</tr>
<tr>
<td>SwitchDeployStatus</td>
<td>2</td>
<td>Status of switch deployment</td>
<td>boolean</td>
</tr>
<tr>
<td>StartupCounter</td>
<td>4</td>
<td>Startup counter</td>
<td>Integer</td>
</tr>
<tr>
<td>PowerPort</td>
<td>2</td>
<td>Power port</td>
<td>Integer</td>
</tr>
<tr>
<td>BatteryV</td>
<td>4</td>
<td>Battery voltage</td>
<td>Integer</td>
</tr>
<tr>
<td>Temp</td>
<td>4</td>
<td>Temperature</td>
<td>ADC counts</td>
</tr>
<tr>
<td>Timestamp</td>
<td>6</td>
<td>Bus time</td>
<td>Integer</td>
</tr>
<tr>
<td>Ejection time</td>
<td>6</td>
<td>Time of last reset</td>
<td>Integer</td>
</tr>
<tr>
<td>Deployment time</td>
<td>6</td>
<td>Time for start of burn sequence</td>
<td>Integer</td>
</tr>
<tr>
<td><strong>Total Bytes</strong></td>
<td><strong>64</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Table 2-4 OOREOS Beacon Packet Definition

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (Bytes)</th>
<th>Offset (Bytes)</th>
<th>Description</th>
<th>Valid for Packet Type</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Website</td>
<td>10</td>
<td>0</td>
<td>OOREOS.org</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>Reserved</td>
<td>4</td>
<td>10</td>
<td>&lt;white space&gt;</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>BusTime</td>
<td>6</td>
<td>14</td>
<td>Bus Time</td>
<td>All, Seconds</td>
<td>Integer</td>
</tr>
<tr>
<td>PacketType</td>
<td>2</td>
<td>20</td>
<td>Beacon packet type (0,1,2,or 3)</td>
<td>All</td>
<td>Integer</td>
</tr>
<tr>
<td>Solar1</td>
<td>4</td>
<td>22</td>
<td>Solar Panel 1 Electrical Current</td>
<td>0</td>
<td>ADC counts</td>
</tr>
<tr>
<td>Solar2</td>
<td>4</td>
<td>26</td>
<td>Solar Panel 2 Electrical Current</td>
<td>1</td>
<td>ADC counts</td>
</tr>
<tr>
<td>Solar3</td>
<td>4</td>
<td>33</td>
<td>Solar Panel 3 Electrical Current</td>
<td>2</td>
<td>ADC counts</td>
</tr>
<tr>
<td>Solar4</td>
<td>4</td>
<td>40</td>
<td>Solar Panel 4 Electrical Current</td>
<td>3</td>
<td>ADC counts</td>
</tr>
<tr>
<td>Health0</td>
<td>2</td>
<td>30</td>
<td>Bus Power Port Status</td>
<td>0</td>
<td>Bit field</td>
</tr>
<tr>
<td>Health1</td>
<td>4</td>
<td>32</td>
<td>R1 Current Phase</td>
<td>1</td>
<td>Integer</td>
</tr>
<tr>
<td>Health2</td>
<td>4</td>
<td>36</td>
<td>Battery Voltage</td>
<td>0</td>
<td>ADC counts</td>
</tr>
<tr>
<td>Health3</td>
<td>4</td>
<td>40</td>
<td>PayLoad Electric Current</td>
<td>1</td>
<td>ADC counts</td>
</tr>
<tr>
<td>Reserved</td>
<td>14</td>
<td>48</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Total Bytes 62**
2.8. Cubesat Telemetry decoding

In decoding the reverse operation for those done in encoding are performed. Those operations are sound decoding and text decoding. Since the Morse code or FSK used by a CubeSat is not unique to the CubeSat and it has been, actually, existing before CubeSats used it, Sound decoding doesn’t differ from CubeSat to other. There are many software that are available for free or paid. Two examples of sound decoder software are MixW and MULTIPSK.

MixW is one of the most famous sound decoder software in HAM radio. Its first version was released in 1999. Figure 2.10 shows a screen shot of the program.

![MixW Screenshot](image)

Figure 2.10 MixW Screenshot

MULTIPSK is widely used software. It’s free and more powerful than MixW, but it is more difficult to use. Figure 2.11 shows a screen shot for it.

In case of AX.25 packetized Beacons, these software extract the info field from the packet.
Figure 2.11 MULTIPSK Screenshot

Beacon, in the text form, decoding on the other hand is unique for each CubeSat. Software is used to decode the beacon and display it. Figure 2.12 shows the telemetry decoder software for OOREOS beacon showed in Section 2.7 and Figure 2.13 shows NanoSail-D’s.

Figure 2.12 The Telemetry Decoder Software For OOREOS Beacon
Figure 2.13 The Telemetry Decoder Software For NanoSail-D Beacon
CHAPTER 3

3. Design of Cubesat Telemetry decoder software

3.1. Software Standard Followed

This project is a software project; at the end of the project software that is able to decode a packet must be submitted. For that reason standards of software projects were reviewed. From among them "IEEE Software Engineering Standards" was chosen for many reasons as shown in Appendix A. Their body of knowledge is "Guide to the Software Engineering Body of Knowledge (SWEBOK)". Part of it is shown in Appendix A.

3.2. Software Development Methodology

In software engineering, software development methodology or system development methodology is a framework that is used to structure, plan, and control the process of developing an information system.

The software development methodology used in this project was Agile. In this model the system is divided into small units, each unit is developed and tested till completion. After that another unit is developed and tested and then integrated with the previous units and so on. This model allows developers to bring out useful software in short time, then adding new features to this software. Also, dividing the system into small pieces converts the problems to smaller ones. The small units their selves were developed using a model known as V-Model. V-Model development is like waterfall, i.e. requirements then design then coding then testing, but testing is taken into consideration since the very first steps. Test cases are developed concurrently with requirements, design and coding. Since the V-Model demonstrates the relationship between each phase of the development life cycle and its associated phase of testing, it is very helpful in testing the system. Figure 3.1 shows the software development methodology.

3.3. Assumptions and Constraints

The first assumption is that the CubeSat is going to send a packet every minute. The software is to be able to handle a file containing all data sent during the day.

Since this software is the decoder software, the data must have been encoded somehow. Unfortunately, there was not a given encoding format. So the encoding must have been assumed.
To do that, deep investigation about most CubeSats that are/were successfully transmitting was done. From that, the ways used to encode data and also types of data sent were collected. The following constraints about encoding were driven:

i. The encoding used must take into consideration the power limitation of the CubeSat. The CubeSat power is very limited. Therefore an encoding that requires the minimum power must be chosen.

ii. Also, the encoding assumed must chase for simplicity, since it’s not preferred that the encoding unit performs complicated job.

iii. Software that is as flexible as possible must be done. The software must be flexible so that no much change in the software will be needed if the encoding format changed.

iv. Also, the encoding used must cover all kinds of data that may possibly be sent.

From the investigation and taking into consideration the above constraints, the following encoding format as shown in Figure 3.2 was assumed.

<table>
<thead>
<tr>
<th>KN0</th>
<th>KN1</th>
<th>KN2</th>
<th>KN3</th>
<th>KN4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL SIGN</td>
<td>TIME</td>
<td>STATUS</td>
<td>VOLTAGES</td>
<td>TEMERATURES</td>
</tr>
</tbody>
</table>

Figure 3.2 Packet Format
Each beacon consists of five parts:

**KN0 CALL SIGN:** the call sign of the CubeSat. Each satellite must have a call sign that identifies it. The call sign must be sent in each packet. It is what indicates that this data is sent from the specific satellite. KNSAT is the call sign of our CubeSat. It stands for Khartoum Nile SATellite.

**KN1 Cubesat Time:** the second part of the packet is the CubeSat time. The CubeSat has an internal clock. This clock time is used by the tracking software to calculate some values. For us as telemetry decoder software it was used as key since no two packets can have the same time. It was also used as index for sorting the packets.

**KN2 STATUS:** this part holds the statuses of the subsystems, i.e. ON or OFF. Each subsystem status is indicated by a letter. It was possible that each subsystem status to be indicated by a bit, 1 or 0. But this would have required 32 different combinations ($2^5$ combinations) which are more than the available letters in Morse Code. If numbers were used in addition to letters this problem would have been solved, but the flexibility objective would have be ruined because no more statuses could be added without complicating the system which is also unwanted. Also, in Morse Code the numbers require more power to be sent than letters which is not good with the limited power budget.

**KN3 VOLTAGES:** the voltages measures are sent in this part. These measures are entered to the encoder using an ADC. They consist of eight bits. There were two options for encoding these 8 bits; either to send each 4 bits as a letter, or number, or to send each 8 bits as a letter where each letter represents a range of values. It was found that in CubeSat telemetry accurate values are not critical. It is not critical if 5.9V for example was sent as 5.0V. So the second option was chosen, i.e. each letter represents a range of values. For Battery voltage the precision is:

$$12V / 25 \text{ letters} = 480 \text{ mV}$$

Where 12 V is the maximum Battery Voltage and 25 letters are the alphabetic letters except the ‘e’. ‘E’ was excluded because in Morse it is represented by a dot ‘.’ which is easily generated by any random noise source.

For Bus 5V the precision is:

$$5V / 25 = 200 \text{ mV}$$

For Bus 3.3V the precision is:
3.3 / 26 = 127 mV ≈ 130 mV

Which are acceptable values.

**KN4 TEMPERATURES:** here the temperature values except the battery temperature are put. Like in Voltages part, a decision must have been taken between sending each 4 bits as letter or each 8 bits as a letter. And for the same reasons of Voltages, range of values was chosen. Here the precision is:

\[(85 - (-40)) ^\circ C / 25 \text{ letters} = 5 ^\circ C\]

Where 85°C is the maximum expected temperature and -40°C is the minimum. A 5°C precision is acceptable.

Each part is assumed to be separated from the other by a space. Table 3-1 gives the details of these parts and Appendix B gives detailed information about the encoding and the values corresponding to each letter.

<table>
<thead>
<tr>
<th>Part Name</th>
<th>No. of chars</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KN0</td>
<td>5 chars</td>
<td>Call sign (KNSAT)</td>
</tr>
<tr>
<td>KN1</td>
<td>5 chars</td>
<td>Time in seconds</td>
</tr>
<tr>
<td>KN2</td>
<td>5 chars</td>
<td>Mode + status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Transmission mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Payload request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Status (On/Off)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• OBC status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Payload status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Transmitter status</td>
</tr>
<tr>
<td>KN3</td>
<td>4 chars</td>
<td>Bus + battery</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Battery:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Temperature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bus 3.3 Voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Bus 5 Voltage</td>
</tr>
<tr>
<td>KN4</td>
<td>9 chars</td>
<td>Temperatures:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• OBC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Antenna</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Cells (6 cells)</td>
</tr>
</tbody>
</table>

Total 28 bytes
Another important thing is the nature of data when it is received. Actually all beacons whether they were CW or packets are received as sound signals. This sound needs to be converted to text. It couldn’t be assumed a certain format for this sound and work on it then modify it in the future, as done with encoding, neither try all possible ways of formatting. Fortunately there are many programs that convert from sound to text for all types of format. These programs are available and they are used extensively by amateur radio community to decode sound. Many programs were checked and the best two programs that are recommended by us are: mixW and MULTIPS K Shown in section 2.8.

### 3.4. System Requirements

The requirements of the system were derived directly from the project objectives on Chapter one, and CubeSats requirements and the literature review in chapter two.

CubeSats send two typed of beacon:

1. CW: which normally hold housekeeping data, but nothing prevents it from holding payload data,
2. Packets: which hold payload data in addition to housekeeping data. They are encoded using AX.25 protocol shown in Chapter 2. Fortunately the sound decoders used in HAM radio community extract the data from the frame. After that they look like a CW beacon and they can be decoded the same way.

The following are the requirements of telemetry decoder software.

#### 3.4.1 Functional Requirements

i. The beacon is saved as a text file. The software is to decode this text file to its corresponding values.

ii. No Currents or power will be sent from the spacecraft. They are to be calculated using voltages.

iii. After decoding and processing data they are to be displayed to the user using GUI.

iv. Voltages, currents, power and temperatures are to be displayed using charts.

v. Data must be saved so that user can view them anytime.

vi. If the data decoded indicated that the CubeSat may be in danger, the software is to alarm the user.

vii. If all data received during a day is saved in one text file, the software is to be able to decode this file with good performance and without unacceptable delays.
3.4.2 Non-Functional Requirements

i. Regarding temperature, some users may prefer Celsius, others may prefer Fahrenheit. The software is to allow user to choose the unit of temperature.

ii. For charts, sometimes it would be good if they were displayed in the same software window. Other times it may be better to display them in separate windows. The software is to be able to handle this.

iii. For saving, sometimes the user may want to save in a separate place rather than the database. The software is to allow doing this.

iv. Also, the colors of the software must be comfortable.

3.5. System Design

The method followed in the design of the project was Object-Oriented Design (OOD). The problem is divided into classes and objects of these classes are used.

The system consists of two software: the first decodes the sound signal and the second decodes the text resulting from the first software. Figure 3.3 shows these subsystems and Figure 3.4 shows the flow charts of them.

As seen in Figure 3.4, when the satellite is in the ground station range, the sound decoder listens to the port and decodes the sound it receives. The output is saved to a text file. This text file is entered to the telemetry decoder software. The software applies the decoding algorithm. The output of the algorithm is then saved and displayed in a GUI.

The telemetry decoder software itself was divided into subsystems. These subsystems are shown in the context diagram of Figure 3.5.
Figure 3.4 Sound Decoder and Telemetry Decoder System’s Flow Chart
Figure 3.5 Telemetry Decoder Software Context Diagram

The subsystems shown in Figure 3.5 do the following jobs:

The decoding subsystem applies the decoding algorithm. It decodes the packet to the corresponding values. It is the basic subsystem. All other subsystems are added to it.

The storage subsystem is responsible of storing the decoded packets in the database. It is also responsible of storing them in any file that a user specifies.

The alarm subsystem is responsible of producing alarms. It examines the decoded data. If they indicate that some values need attention, alarms are issued at those values.

The options subsystem is responsible of the options of display. The chart subsystem is responsible of drawing charts using the decoded data. The display subsystem displays the decoded data on a GUI. It also displays the alarms and charts.

The way a user can interact with the software is shown in the use case diagrams shown in Figure 3.6 and Figure 3.7. The user can import a file to be decoded, save the decoded data to the database, save it to a specific file, draw charts, change the way charts are displayed, change the temperature unit and change the theme. In Figure 3.7 the options details are not shown. Options are not strategic details in this project. Save and save as have the same details.
Figure 3.6 Use Case diagram (1)

Figure 3.7 Use Case Diagram (2)
Since object oriented design is used, classes were developed to represent the problem solution. Figure 3.8 shows the class diagram of the software. The frame class is the main class. The execution starts from it. Most objects are declared here. It is related to other classes as follows:

- Packet class: frame class can call any number of objects of packet file. This means the user can import more than one file; each file is of class pack.
- Options: since the option is unique at any moment, the Frame class is connected to options class by a one-to-one relationship.
- There is no direct connection between the Frame class and the beacon class.

The Packet class deals with any imported file as an object. Operations on this object are independent of other objects. The Packet file consists of many beacons. Its relation to the beacon class is many-to-many.

The beacon class deals with beacons. Each object of it is one beacon. The decoding algorithm is applied to the beacon. Also the alarm check is performed on beacons. The many-to-many relationship between beacon and packet classes means that there can be many packets objects. Each object can have many beacons. And there can be many beacons. Each of them can belong to different packet object.

The option class gets the different options from the user. It has only one object since there cannot be many options at the same time.

The sequence by which each class calls the other is shown in Figure 3.9. Files are imported from the frame class. Each file is an object of class Packet. Packet is responsible of decoding the file. Packet sees the packet as a collection of objects of class beacons. Beacon decodes each object and returns the result to packet. Packet then returns the whole packet object to class Frame to display it. Before displaying it, Frame calls the object of class Options to know how the user wants it to be displayed. Then Frame displays it.

The sequence by which functions of those classes are called is shown in the activity diagram of the system shown in Figure 3.10.
Figure 3.8 Class Diagram

Figure 3.9 Sequence Diagram
Figure 3.10 Activity Diagram
From Figure 3.10:

- First `import()` function is called.
- It calls the `chooseFile()` function which returns the directory of the file.
- Then it calls the `checkValid()` function. This function checks if the file is valid for decoding or not. It checks in two steps. First is the file a text file or not? If it is not it throws an exception and exits the function back to `import()` function. If the file is text, it checks if the beacons start with KNSAT or not. If it doesn’t it throws an exception and exits the function back to `import()` function.
- If it starts with KNSAT then this file is valid and `createPacket()` is called.
- `createPacket()` first calls `incrementNoOfPackets()` which increments the number of packets.
- Then `createBeacon()` function is called which creates beacon objects as many as the Packet file contains.
- `Devideandconvert()` function is called next which devides the packet parts and decode each part.
- The decoded values are checked to see if alarm must be displayed. If there is alarm, `DisplayAlarm()` is called. If not nothing happens.
- Then temperature check is done. If the temperature must be displayed in Fahrenheit, the `ConvertTemp()` is called. If it must be displayed in Celsius nothing happens.
- Finally `DisplayValues()` function is called which displays the values on the GUI.

3.6. System Coding

The code of the system was developed based on the design above and using Java programming language the reasons for choosing Java are shown in the following section.

3.7. Programming Language used

The programming language chosen for coding was java™. Java was chosen because:

- It is Object oriented Programming Language.
- Java is the best language that deals with strings. Since the beacons to be decoded are in text format, Java was the best choice.
- Java is a good choice when designing systems that require security. Of course satellites data are secure in their nature.
- Programs written in Java can be run under any operating system.
3.8. System Testing
The following levels of testing must be done to the software:

1. Unit Testing
2. Integration Testing
3. System Testing

3.8.1 Unit Testing
Here each unit must be tested independently. The main purpose of this test is to make sure that there are no syntax errors or logical errors. Since system units development was done using V-Model, this test must have been done regularly and unit by unit. The following functions must to be tested extensively.

**Import Function:**
This function must pass two tests:

1. If a non text file was imported.
2. If a text file that contains data that doesn’t belong to KNSAT was selected.

**Save and Save As Functions:**
These functions must pass the following tests:

1. If the database is not found.
2. If the database file cannot be opened.
3. If in “Save As” a file that is already exits was specified.

**Convert Function:**
This function must pass two tests:

1. If the string passed to it is not complete (couldn’t be received by the sound decoder for example).
2. If the string passed to it contains undefined symbols.

3.8.2 Integration testing
Since each unit is developed independently, a test must be performed when integrating those units. The model used in developing the system, Agile Model, helps with this. Because unit by unit are finished and then new unit is added to the existing system, the integration testing is done
by the end of each unit, and each time only one unit is added to the integration. This simplified the testing very much.

### 3.8.3 System Testing

System testing is done when development of the system finishes. Here the software is tested as a black box which must be provided with inputs and brings out the outputs. In this testing, help from U of K CubeSat team is needed. Their ground station and CubeSat prototype are needed to act as the receiver and the sender respectively.
CHAPTER 4

4. Results and Discussion

4.1. System Developed

The code of the software was written based on the design shown in chapter 3. Since the code is huge, part of it is shown in appendix C and the complete code is attached in the CD with the thesis.

The system developed was Telemetry decoder software that has a GUI. The GUI consists of three taps:

1. Start,
2. Beacon, and
3. Options.

Figure 4.1 shows the Start tap. When the program is opened this tap is displayed. It holds the name of the project and the Logo developed for the project.

Figure 4.1 Start Tap
The Beacon tap is the main tap in the GUI. From it, the user can import the text files. Decoding results are displayed on this tap. Also from this tap the user can “Save” and “Save As” the decoded files. Wherever the mouse is moved, a suitable help will be displayed on the lower panel of the tap. Figure 4.2 illustrates this tap.

The Beacon Tap consists of the following buttons:

- **Import**: to select the file to be decoded and decode it (if it is a valid file).
- **Save**: to save the decoded data to the local database.
- **Save As**: to save the data to a text file in the directory that the user specifies.
- "<<" and ">>" buttons: the user can import more than one file. To move from file to another those buttons are used.
- "Previous" and "Next" buttons: the selected file may contain more than one beacon. To move from beacon to another those buttons are used.
- "X" on the upper right corner: is used to close the current file. But if there is only one file opened, the button is designed not to close it.

In addition to those buttons, there are many buttons to draw charts. Those are the buttons of voltages, currents, power and temperatures.

The Options tap allow user to choose temperature Unit, Theme and the ways charts are displayed. Figure 4.3 shows this tap.

![Figure 4.3 Options Tap](image)

**4.2. How to use the software:**

To decode a file:

- The user selects the file using the Import button which displays a file chooser dialogue box.
- The file chooser by default displays text files only.
- Beacons will be decoded and displayed on the software.
- The user can draw charts by selecting the button of the data wanted.
Figure 4.4 shows that. A file named “test.txt” was imported. “Cell 1 temp” chart was drawn.
User can save data in the database by clicking “Save” button.

User can save data in any other file by clicking “Save As” button. A dialogue box asking the user the location of the file and the file name will be displayed as shown in Figure 4.5. Then a text file holding that name will be created at the specified location. Figure 4.6 shows the file before encoding and after decoding and saving it in a file named “Save As.txt”
Charts can be viewed in the same window or in a separate window by changing the corresponding setting from the option tap. Figure 4.7 shows the two ways for displaying charts.

The chart can be dealt with as an image that can be saved, copied, zoomed, printed and changing its properties as in Figure 4.8.

Also, the user can Import more than one file. Each file is treated independently. In Figure 4.9 three files were imported and the user was viewing the first file of them. The three files and the database after saving them is showed in Figure 4.10.

As the mouse is moved on a button, a help tip will be displayed on the help panel at the bottom of the software. Figure 4.11 shows some examples of help.

Alarms are displayed by filling the text box of the worrying values in red as in Figure 4.12.
Results and Discussion

Chapter 4

Figure 4.8 Chart Actions: Copy, Save As, Properties, Zoom and Print

Figure 4.9 Multiple Files Import
Figure 4.10 Three Files Imported (Upper) and The Database After Saving Them

Figure 4.11 Some Help Tips

When the mouse is not pointing to a button

"<<" button

"Import" button

"Save" button
The temperature unit can be Celsius or Fahrenheit as decided from the options tap. Figure 4.13 shows a file decoding result; temperatures were displayed in Celsius once and in Fahrenheit another time. Also two different themes were used to display them.

Figure 4.12 Alarm Display

Figure 4.13 Temperature Unit: Celsius (Left) Fahrenheit (Right)
4.2.1 Exceptions and Error Handling:

This is an important part that must be carefully handled when developing software. If it is not well handled the system becomes exposed to failure and thus the performance of the system degrades. The system can handle the following actions:

1. If the user tried to save before importing any file: the system will ask the user to import a file first as in Figure 4.14.
2. If the user tried to import a non text file: by default the file chooser dialog box will display text files only. If the user changed the type of displayed files to show all types and selected a non text file the system will tell the user that there is something wrong as in Figure 4.15.
3. If the user selected a text file containing data that doesn’t belong to the CubeSat: the software will display error message saying that this is a wrong file as in Figure 4.16.
4. If the database, where the program saves decoded beacons by default, was deleted: the software will automatically create a new database.
5. If the user tried to save in a directory containing a file with the same name user selects: the file will be overwritten.

Figure 4.14 Trying to Save Without Importing a File
Figure 4.15 Trying to Import a Non-text File

Figure 4.16 Trying to Import a Text Tile That Doesn't Belong to The CubeSat
4.3. **Software Testing:**

The test cases specified in chapter three were done and the software passed all of them. Table 4-1 shows the unit test results.

**Table 4-1 Unit Test Results**

<table>
<thead>
<tr>
<th>No.</th>
<th>Test</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>If a non text file was imported</td>
<td>Error message was displayed as in Figure 4.15.</td>
</tr>
<tr>
<td>2</td>
<td>a text file that contains data that doesn’t belong to KNSAT was selected</td>
<td>Error message was displayed as in Figure 4.16.</td>
</tr>
<tr>
<td>3</td>
<td>If the database was not found</td>
<td>New database was created</td>
</tr>
<tr>
<td>4</td>
<td>The database file cannot be opened</td>
<td>Error message was displayed.</td>
</tr>
<tr>
<td>5</td>
<td>If in “Save As” a file that is already exits was specified</td>
<td>The file was overwritten</td>
</tr>
<tr>
<td>6</td>
<td>If the string passed to it is not complete</td>
<td>Missing fields were filled with spaces in the decoded file</td>
</tr>
<tr>
<td>7</td>
<td>If the string passed to it contains undefined symbols</td>
<td>Undefined symbols were filled with spaces in the decoded file</td>
</tr>
</tbody>
</table>

The integration testing was done by the end of each unit and it was successful. To do the System Test, a file containing sample packets was made. Help from U of K ground station was needed. Their CubeSat prototype was used to send the sample packets. It sent them from the 4\textsuperscript{th} floor of the Engineering Tower. The ground station was configured at the Distance Learning Hall. The packets were received using an “HP Pavilion dv4” laptop using the sound card. MULTIPSK program decoded this sound to text. This text was saved to a file named “test.txt”. This file was decoded and saved to the database. Figure 4.17 shows “test.txt” file and the database after decoding and saving the file. Figure 4.18 shows the ground station configuration.

Also the software was given to UofK ground station team to test it. No problem in using the software has been reported until this moment.
Results and Discussion

Figure 4.17 The Encoded Packet (Upper) and The Database After Decoding The Packets (Lower)

Figure 4.18 UofK Ground Station and Cubesat Prototype
**4.3.1 Discussion:**

The software is simple and can be run on any computer that have java run time environment. Java run environment can be installed on any computer with any operating system.

The power and currents displayed are calculated from the voltages received knowing the resistances of the CubeSat. This simplifies the CubeSat encoder hardware and minimizes the amount of data sent.

There are two versions of this software: One for users around the world who will help in receiving the telemetry. The other is for the CubeSat ground station. The only difference is that the first version doesn’t need the database; it only needs a save as button. This version is almost complete and needs little improvement, but the ground station version will need more improvement as will be shown in the future work in chapter 5.

The performance of the software is very good. Since the software deals with the file line by line (beacon by beacon), it doesn’t matter if the file contains 10 or 1000 lines. It matters only when charts are to be displayed. But since a text file is used as database, and the chart library deals with text files, the performance is the maximum. If another type of database was used, e.g. mySQL, the system will spend more time copying from database to a temporary text file then display it.

The system was tested with a file containing 1500 messages (which is more than the expected packets per day). It gave satisfactory performance and no unacceptable delays were recorded.

When encoding was being, developed, a choice between representing each 8 bits by a character or each 4 bits by a character must have been done. The following balance chart (Figure 4.19 ) shows the comparison and the winner.
The GUI was built using JFrame class (library) which was a good choice for dealing with the GUI. Components of the GUI can be chosen from a side panel and dropped to the tap. This method was easy compared to what should have been done without this class. Each component position should have been stated in terms of pixels. Also colors would have been defined using RGB where each color is defined using its color components: Red, Green and Blue.

The charts were drawn using a class named JFreeChart. This class draws charts from a text file. It is free to download but it is restricted from Sudan. A proxy server was used to solve this problem, but there was another problem. The documentation of this class is not free. Fortunately, a company that uses this class to develop its software was found and a copy of the documentation was given to us.

The charts are displayed using Bars because the values contained in the beacons are discrete. Samples are taken every minute and then sent.

Displaying the alarm in red color attracts the eye to the warning. Red is not used anywhere in the software except the alarm.
One of the useful things about text is files that they can be further processed in many ways. For example they can be copied to an SQL database using a small interface program. For example they also can be copied to a spread sheet like Microsoft Excel as in Figure 4.20 where the decoded file of Figure 4.17 was pasted in an Excel sheet and some charts were drawn.

![Excel Spread Sheet](image)

**Figure 4.20 Text Files Can Easily be Further Processed**

This software has some features that are not found in any other CubeSat telemetry decoder software until now. These features are:

1. The alarm display.
2. The multiple import feature.
3. The ability to do actions with the charts, i.e. save, copy, zoom, print and change properties.
4. The software is platform independent. It was written using Java which can be run on any computer.
5. Developing the decoder software in the university. All universities assign the decoder software to a software engineer, named Mike Rupprecht\(^1\), who is a Ham radio member and a software engineer too.

---

\(^1\) His web site: [http://www.dk3wn.info/dk3wn.shtml](http://www.dk3wn.info/dk3wn.shtml)
CHAPTER 5

5. Conclusion, comments and recommendations

5.1. Conclusion

In this project a decoder software that decodes Beacons sent from a special type of satellites, called CubeSat, is developed.

But it is a decoder, how was encoding done? Because there was no encoding format, encoding was investigated and an encoding format was proposed and used. This required deep literature review about CubeSats communication system, type of data sent by CubeSat’s telemetry system, duration of contact between the CubeSat and the ground station and ranges of values sent (Voltages and temperature).

It was found that CubeSats are launched in Low Earth Orbit (LEO). In LEO the contact duration with the ground station is limited to less than thirty minutes a day. There were two solutions for this problem. Either to build more than one ground station so as to increase the contact duration or to use HAM radio community frequency so that many users all around the world contribute in receiving. The second choice was more economic and reasonable. This added another literature review task to investigate how data is encoded and sent in ham radio community.

CubeSat data is sent as Beacons holding housekeeping data (data about the CubeSat health) in addition to payload data (data of the mission which the CubeSat was sent to do). Beacons are sent by modulating a carrier using Morse code where the Morse code corresponding to each letter of the beacon is sent. In Morse some letters require less power when sent than other letters. This was taken into consideration when encoding was developed. Beacons are also sent by modulating a carrier using FSK in most CubeSats. Therefore data is received as sound in the ground station. Fortunately, ham radio community has many sound decoders software. These software were used to decode the sound signal to text. This text is to be decoded knowing the format of the encoding. The software developed in this project performs this task.

Comparison was done between many programming languages to choose the best one for this project. Java was the best choice and it was learnt and used.
5.2. Future Work

The telemetry decoder is only one of the software set found in ground stations. These software are related to each other and work together. So, there are many things that can be done in the future.

5.2.1 Interface with other ground station software

The telemetry decoder software can be connected to the command software. The data analyzed by the decoder is useful to the command software. If there is an alarm, the command software must try to rescue the CubeSat. Also, if the CubeSat is trying to handshake with the ground station, there must be corporation between the decoder software and the command software.

Also, the decoder software must be connected to the tracking software. Till this point the software is so far from being an automatic system. The port will be opened and closed by the user. By connecting the decoder and the tracking software this can be done automatically.

The decoder software must be connected to the sound decoder software. Till now, the decoded packet by the sound decoder is copied and pasted it in a text file. Then this file is imported manually to the telemetry decoder software. By integrating these software the system will become real time.

5.2.2 A web server to allow users to upload packets on it

Most CubeSats receive packets from the ham radio members around the world by email. This method is somehow tiresome to the receivers and to the ground station operators. It will be perfect if the receivers upload the packets directly to a server. An Upload button can be added to the software. This will reduce the receivers task to just a button click. Also the software can automatically import the packets from the server, decode them and display them to the ground station operators.
References


2. Gießelmann, J., Development of an Active Magnetic Attitude Determination and Control System for Picosatellites on highly inclined circular Low Earth Orbits, in School of Aerospace, Mechanical and Manufacturing Engineering. 2006, RMIT University.


6. Alminade, L., Bisgaard, M., Binther, D., Viscor, T., and Østergaard, K., Robustness of Radio Link Between AAU-Cubesat and Ground Station.


13. Anderson, J.L., AUTONOMOUS SATELLITE OPERATIONS FOR CUBESAT SATELLITES. March 2010, the Faculty of California Polytechnic State University: San Luis Obispo.


Appendix A: Guide to the Software Engineering Body of Knowledge (SWEBOK)

Standards are document established by consensus and approved by a recognized body that provides guidelines or characteristics for activities or their results, aimed at the achievement of the optimum degree of order in a given context.

A.1. Background

1993: Foundation of the Joint IEEE-CS and ACM Steering Committee.
1998: Foundation of the Software Engineering Coordinating Committee (SWECC)

A.2. Objectives

1. Promote a consistent view of SE worldwide
2. Clarify the place and boundaries of SE relative to other related disciplines
3. Define/characterize the contents of the SE discipline
4. Provide access to the various components of the SWEBOK
5. Provide a foundation for curriculum development and individual certification and licensing

A.3. Project Background:

International participation from industry, professional societies, standards bodies, academia, authors Over 500 hundred software engineering professionals have been involved

A.4. Project Team:

1. Editorial Team of the Guide
2. Industrial Advisory Board
3. Associate Editors of the Knowledge Areas
4. Reviewers (573 persons, 55 countries)

A.5 Knowledge Areas

1. Software Requirements
2. Software Design
3. Software Construction
4. Software Testing
5. Software Maintenance
Figure A-1 shows the Primary and Supporting Knowledge Areas.

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Design</th>
<th>Construction</th>
<th>Testing</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software Configuration Management</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Software Engineering Management</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Software Engineering Process</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Software Engineering Tools and Methods</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Software Quality</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### A.5.1. Software Requirements

**Definition:**

A requirement is defined as a property that must be exhibited in order to solve some real-world problem.

The Software Requirements Knowledge Area (KA) is concerned with:

1. Requirements Elicitation
2. Requirements Analysis
3. Requirements Specification
4. Requirements Validation
A.5.2. Software Design

Definition:
Design is both the process of defining the architecture, components, interfaces, and other characteristics of a system or component and the result of that process.

A.5.3. Software Construction

Definition:
Software construction refers to the detailed creation of working, meaningful software through a combination of coding, verification, unit testing, integration testing, and debugging.

A.5.4. Software Testing

Definition:
Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems. Software testing consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior.
Appendix B: Packet Format

The packet is assumed to have the following format:

<table>
<thead>
<tr>
<th>Kn0</th>
<th>Kn1</th>
<th>Kn2</th>
<th>Kn3</th>
<th>Kn4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL SIGN</td>
<td>TIME</td>
<td>STATUS</td>
<td>VOLTAGES</td>
<td>TEMPERATURES</td>
</tr>
</tbody>
</table>

Figure B-1 Beacon Parts

Where the details of each part are given in the table below.

Table B-1 Beacon Parts Details

<table>
<thead>
<tr>
<th>Part Name</th>
<th>No. of chars</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KN0</td>
<td>5 chars</td>
<td>Call sign (KNSAT)</td>
</tr>
<tr>
<td>KN1</td>
<td>5 chars</td>
<td>Time in seconds</td>
</tr>
<tr>
<td>KN2</td>
<td>5 chars</td>
<td>Mode + status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Transmission mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Payload request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Status (On/Off)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- OBC status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Payload status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Transmitter status</td>
</tr>
<tr>
<td>KN3</td>
<td>4 chars</td>
<td>Bus + battery</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Battery:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Temperature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Bus 3.3 Voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Bus 5 Voltage</td>
</tr>
<tr>
<td>KN4</td>
<td>9 chars</td>
<td>Temperatures:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Payload</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- OBC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Antenna</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Cells (6 cells)</td>
</tr>
</tbody>
</table>

Total 28 bytes

The lookup tables for the corresponding values for the alphabetic letters are given in the tables below. Also the corresponding codes in Morse are shown.
### Table B-2 Payload Request Lookup Table

<table>
<thead>
<tr>
<th>Value</th>
<th>In Morse</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>-</td>
<td>No payload data to be sent</td>
</tr>
<tr>
<td>I</td>
<td>..</td>
<td>Payload data ready to be sent</td>
</tr>
</tbody>
</table>

### Table B-3 Cubesat Status Lookup Table

<table>
<thead>
<tr>
<th>Value</th>
<th>In Morse</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>-</td>
<td>Is ON</td>
</tr>
<tr>
<td>I</td>
<td>..</td>
<td>Is Of</td>
</tr>
</tbody>
</table>

### Table B-4 Transmission Modes Lookup Table

<table>
<thead>
<tr>
<th>Mode</th>
<th>Value</th>
<th>In Morse</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>B</td>
<td>I</td>
<td>..</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>.-</td>
</tr>
<tr>
<td>D</td>
<td>N</td>
<td>-.</td>
</tr>
<tr>
<td>E</td>
<td>U</td>
<td>..-</td>
</tr>
<tr>
<td>F</td>
<td>D</td>
<td>..-</td>
</tr>
</tbody>
</table>

### Table B-5 Battery Voltage Lookup Table

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Value</th>
<th>In Morse</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>E</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>I</td>
<td>..</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>.-</td>
</tr>
<tr>
<td>5</td>
<td>N</td>
<td>-.</td>
</tr>
<tr>
<td>Code</td>
<td>Temperature</td>
<td>Bus Voltage</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>A</td>
<td>&lt;40</td>
<td>&lt;0.00</td>
</tr>
<tr>
<td>B</td>
<td>-35</td>
<td>0.00</td>
</tr>
<tr>
<td>C</td>
<td>-30</td>
<td>0.25</td>
</tr>
<tr>
<td>D</td>
<td>-25</td>
<td>0.50</td>
</tr>
<tr>
<td>E</td>
<td>-20</td>
<td>0.75</td>
</tr>
<tr>
<td>F</td>
<td>-15</td>
<td>1.00</td>
</tr>
<tr>
<td>G</td>
<td>-10</td>
<td>1.25</td>
</tr>
<tr>
<td>H</td>
<td>-5</td>
<td>1.50</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>1.75</td>
</tr>
<tr>
<td>J</td>
<td>5</td>
<td>2.00</td>
</tr>
<tr>
<td>K</td>
<td>10</td>
<td>2.25</td>
</tr>
<tr>
<td>L</td>
<td>15</td>
<td>2.50</td>
</tr>
<tr>
<td>M</td>
<td>20</td>
<td>2.75</td>
</tr>
<tr>
<td>N</td>
<td>25</td>
<td>3.00</td>
</tr>
<tr>
<td>O</td>
<td>30</td>
<td>3.25</td>
</tr>
<tr>
<td>P</td>
<td>35</td>
<td>3.50</td>
</tr>
<tr>
<td>Q</td>
<td>40</td>
<td>3.75</td>
</tr>
<tr>
<td>R</td>
<td>45</td>
<td>4.00</td>
</tr>
<tr>
<td>S</td>
<td>50</td>
<td>4.25</td>
</tr>
<tr>
<td>T</td>
<td>55</td>
<td>4.50</td>
</tr>
<tr>
<td>U</td>
<td>60</td>
<td>4.75</td>
</tr>
<tr>
<td>V</td>
<td>65</td>
<td>5.00</td>
</tr>
<tr>
<td>W</td>
<td>70</td>
<td>5.25</td>
</tr>
<tr>
<td>X</td>
<td>75</td>
<td>5.50</td>
</tr>
<tr>
<td>Y</td>
<td>80</td>
<td>5.75</td>
</tr>
<tr>
<td>Z</td>
<td>&gt;85</td>
<td>&gt;6.00</td>
</tr>
</tbody>
</table>

Table B-6 Temperature and Bus Voltage Lookup Table
The alarm is given at the following values:

If the mode is F

If the payload request is I “ready packet” or unknown value was received.

If the payload is off

If the onboard computer is off

If the transmitter is off.

If the battery Voltage is < 11

If the battery Temp is < 0

If bus3.3 Voltage is < 3

If bus5 Voltage is < 4

If temperature, except for Battery, is < 0 or > 60
Appendix C: Code

Only the functions that convert to the corresponding values are shown here. The complete code is attached in the CD with the thesis.

First: the variables declarations, only the needed ones are mentioned here:

```java
private String line; // the beacon

    // the beacon parts
    private String KN1;
    private String KN2;
    private String KN3;
    private String KN4;

    //KN0:
    private String siteName; // the call sign

    // KN2: 5 digits
    private String mode;
    private String payloadRequest;
    private String statusOBC;
    private String statusPayload;
    private String statusTx;

    // KN3: 6 digits
    private int batteryVoltage;
    private double batteryTemp;
    private double bus3Voltage;
    private double bus3Current;
    private double bus5Voltage;
    private double bus5Current;
    private double pwrBus3;
    private double pwrBus5;

    //KN4: 9 digits
    private double tempPayload;
    private double tempOBC;
    private double tempAntenna;
    private double tempCell1;
    private double tempCell2;
    private double tempCell3;
    private double tempCell4;
    private double tempCell5;
    private double tempCell6;
```
The following method, function, separates the beacon to its parts. Then it calls the functions that convert each part. It calls the alarm function. It checks the temperature unit and calls changetemperature function if needed.

```java
public convert(String stream,options tUnit){
    tempUnit=tUnit; // Calices of Fahrenheit
    try{
        line=stream;
        String [] KN =stream.split(" ");
        int k=0;
        siteName=KN[k++];
        KN1=KN[k++];
        KN2= KN[k++];
        KN3= KN[k++];
        KN4=KN[k++];
        toKN2(KN2);
        toKN3(KN3);
        toKN4(KN4);

        alarm();
        if(tempUnit.getTempUnit()=='f') changeTemp(tempUnit);
    }catch(Exception e){}
}
```

The following functions convert the different parts of the beacon.

Convert KN2:

```java
private void toKN2(String kn2){
    try{
        int k2=0;
        mode= toMode(kn2.charAt(k2++));
        payloadRequest= toPayloadRequest(kn2.charAt(k2++));
        statusPayload= toStatus(kn2.charAt(k2++));
        statusOBC= toStatus(kn2.charAt(k2++)) ;
        statusTx= toStatus(kn2.charAt(k2++));
        status = toStatus(kn2.charAt(k2++));
    } catch(Exception e){}
}
```

Convert KN3

```java
private void toKN3(String kn3){
    try{
        int k3=0;
```
batteryVoltage = toBatteryVoltage(kn3.charAt(k3++));

batteryTemp = toTemp(kn3.charAt(k3++));
bus3Voltage = toBusVoltage(kn3.charAt(k3++));
bus3Current = toBusCurrent(bus3Voltage);
    bus5Voltage = toBusVoltage(kn3.charAt(k3++));
bus5Current = toBusCurrent(bus5Voltage);
pwrBus3= power(bus3Voltage,bus3Current);
pwrBus5= power(bus5Voltage,bus3Current);
} catch(Exception e){}

Convert KN4

private void toKN4(String kn4){
    try{
        int k4=0;
        tempPayload = toTemp(kn4.charAt(k4++));
        tempAntenna = toTemp(kn4.charAt(k4++));
        tempOBC = toTemp(kn4.charAt(k4++));
        tempCell1 = toTemp(kn4.charAt(k4++));
        tempCell2 = toTemp(kn4.charAt(k4++));
        tempCell3 = toTemp(kn4.charAt(k4++));
        tempCell4 = toTemp(kn4.charAt(k4++));
        tempCell5 = toTemp(kn4.charAt(k4++));
        tempCell6 = toTemp(kn4.charAt(k4++));
    } catch(Exception e) {}}

The above functions are calling the lookup tables’ functions. Lookup tables compare the values given to them to the values they have and return the result. Only one function is shown below. No need to put more than example. The function is the lookup table for the status.

private String toStatus(char m){
    String s= " ";
    switch(m) {
        case 'T' : s= "ON"; break;
        case 'I' : s= "OFF"; break;
        default : s= "error";
    } return s;
}