ABSTRACT


The increasing use of GPS measurements in interdisciplinary field like space weather, real-time spacecraft navigation, formation flying, three-axis attitude control, precise time synchronization and many more; users requires a good GPS signal measurements equipment or simulator. The burst of applications in these fields requires an accurate and elaborate GPS signal measurement for different environment and scenarios with different user trajectories. For terrestrial application, conventional receiver can be used to generate GPS measurements but for aerospace and space applications, researchers relies on simulator. The high cost of hardware based GPS signal simulator to simulate the desired environment and trajectory and capture the required signal measurements limits the usage of one of the most marvel tool in modern science. This work presents an object oriented system design and modeling (OOSDM) framework for reliable and flexible system design. Further OOSDM is used to design open-source software based simulation environment for GPS measurements. It evades the need of high cost GPS hardware based simulator and generates measurements that can be used in algorithm development for innovative applications of GPS. Open source nature of this tool will attract users and researchers in community to expand the capabilities of this simulator.
GPS-Gyan: An Open-Source GPS Software Simulator Using Object-Oriented System Design And Modeling Framework

by

Priyank Kumar

A thesis submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Master of Science

Electrical Engineering

Raleigh, North Carolina

2010

APPROVED BY:

__________________________
Dr. William Edmonson
Chair of Advisor Committee

__________________________
Dr. Winser Alexander

__________________________
Dr. Alexandra Duel-Hallen
Dedication

To my family, friends and my inspiration: Maa Gyan.
Biography

Priyank Kumar did his schooling in New Delhi, the capital city of India. He received his Bachelor of Engineering in Electrical & Communication from the KLS Gogte Institute of Technology at Belgaum. He then worked for four years as a senior systems engineer with Accord Software and Systems Pvt Ltd, Bangalore, India, before arriving at the North Carolina State University, where he is currently pursuing a Master of Science degree in Electrical Engineering. His interests lie in signal processing related to wireless communications and navigation systems like GPS and in reliable and robust systems design.
Acknowledgements

I thank Dr. William Edmonson, my advisor, for his constant motivation and inspiring thoughts and most of all for the visionary freedom he granted in allowing me to come up with the idea and cultivating it the meticulously to completion. I thank Dr. Winser Alexander and Dr. Alexandra Duel-Hallen for agreeing to be on my thesis committee. I thank all the members of the High Performance DSP lab at NCSU. I thank the Department of Electrical and Computer Engineering and NCSU for creating and maintaining such a beautiful and professional environment that is conducive to work and research. I thank my friends for their support in good and bad times we’ve had throughout this masters program. I thank Vikram Mulukutla, who is also pursuing the Master’s thesis and Arunesh Goswamy, who is pursuing PhD, for all the lively discussions during coffee and lunch breaks during the time when we were writing our thesis drafts and completing the research. I thank all those who have been involved with the creation and maintenance of Lyx, the LateX frontend and wholeheartedly thank Lara Pagano for creating the Lyx template for NCSU thesis that made writing this thesis a breeze.

I sincerely acknowledge the work of developer of GPSTk whose source is used extensively in this thesis.

I especially thank my family for their support and commitment to my education all the way to my Master’s degree.
# Table of Contents

List of Tables ................................................. x

List of Figures .............................................. xi

Chapter 1  Background ........................................... 1
  1.1  Introduction .............................................. 2
  1.2  Outcome and Impact ...................................... 5
  1.3  Approach and Outline .................................... 6

Chapter 2  Related Work ....................................... 8
  2.1  Hardware GNSS Simulators ................................. 8
      2.1.1  Flow Diagram ...................................... 9
      2.1.2  Existing Product Review ............................. 9
  2.2  Software GNSS Simulators ............................... 11
      2.2.1  Software Simulator Flow Diagram .................. 11
      2.2.2  Existing Product Review ............................. 11
  2.3  GNSS Post Processing Tools ............................. 13
      2.3.1  GPSTk ............................................. 13
      2.3.2  RTKLIB ............................................ 15
  2.4  Motivation ............................................... 16
      2.4.1  GPS in Science and Space Applications ............ 16
      2.4.2  Lack of an Open-Source Simulator ................ 18
      2.4.3  Need of a Software Architecture .................. 19
Chapter 3  GPS Signal Theory ................................................. 20
3.1 Time Based Systems ..................................................... 22
3.2 Signal Delay Models .................................................... 23
  3.2.1 Satellite Clock Errors .............................................. 25
  3.2.2 Antenna Phase Center Variation ($\phi_{pcv}$) .................... 25
  3.2.3 Polarization-Induced Carrier Phase Wind-Up ($\phi_{pwu}$) .... 25
  3.2.4 Free Space Propagation Delay ($\delta t_{trn}$) .................... 25
  3.2.5 Ionospheric Delay ($\delta t_{ion}$) ................................ 26
  3.2.6 Tropospheric Delay ($\delta t_{tropo}$) ............................. 27
  3.2.7 Multipath and Shadowing Effects ($\phi_{mp}$) .................... 27
  3.2.8 Line Bias ($\delta t_{ln}$) ............................................ 28
  3.2.9 Receiver Noise ($n_{\phi}$) ....................................... 29
3.3 Signal Delay Model Equation ......................................... 29

Chapter 4  Software Design Architecture ................................. 31
4.1 Philosophy .............................................................. 31
4.2 Object Oriented Programming (OOP) ................................ 33
  4.2.1 Object Orientated Paradigm: Users View ....................... 34
  4.2.2 Object Orientated Paradigm: Object’s View .................... 35
    4.2.2.1 Fetching Inputs: Public .................................... 36
    4.2.2.2 Fetching Controls: Public .................................. 36
    4.2.2.3 Prepare Inputs to Object: Protected ......................... 37
    4.2.2.4 Object Controller: Protected ................................ 37
    4.2.2.5 Methods of the Object: Protected and Private ............ 39
    4.2.2.6 Output from the Object: Public ............................ 39
4.3 Class, Objects and Processes ......................................... 39
  4.3.1 Standard Class Definition ....................................... 39
  4.3.2 OOSDM Class Definition ........................................... 40
4.4 Quality: New paradigm in OOSDM ........................................ 44
  4.4.1 Why Quality? .................................................. 44
  4.4.2 Errors in Software ............................................ 47
  4.4.3 Quality Process in OOSDM ................................. 49
    4.4.3.1 Verification ............................................. 49
    4.4.3.2 Validation ................................................ 50
    4.4.3.3 Exception Handler ..................................... 50

4.5 Health Monitoring and Debug Information .............................. 52
  4.5.1 Why Health Monitoring? .................................. 53
  4.5.2 Basic Health Monitoring ................................. 53
  4.5.3 Debug Port .................................................. 54

4.6 Complete Software Architecture ...................................... 54
  4.7 Chapter Summary ............................................. 57

Chapter 5 GPS Simulator Architecture .................................... 58
  5.1 GPS Simulator Design: Fundamental Assumption .................. 60
  5.2 Base Time Generator Engine ................................... 60
    5.2.1 Mathematical Model ..................................... 61
    5.2.2 Engine Controller ......................................... 62
    5.2.3 Exception Handling ....................................... 63
    5.2.4 Input Output Table ...................................... 63
    5.2.5 Sample Results .......................................... 63
  5.3 Satellite Section ............................................. 66
    5.3.1 Generation of Satellite Data and Clock .................. 66
      5.3.1.1 Mathematical Model ................................. 67
      5.3.1.2 Engine Controller ................................... 67
      5.3.1.3 Exception Handling ................................ 68
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>Comparison Chart</td>
<td>10</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Comparison Chart: Software Simulators</td>
<td>12</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Typical Power Spectral Density for Different Clock [38]</td>
<td>62</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Interfaces and Processes of Satellite Generator Engine</td>
<td>64</td>
</tr>
<tr>
<td>Table 5.3</td>
<td>Interfaces and Processes of Satellite Generator Engine</td>
<td>69</td>
</tr>
<tr>
<td>Table 5.4</td>
<td>Interfaces and Processes of Satellite Generator Engine</td>
<td>79</td>
</tr>
<tr>
<td>Table 5.5</td>
<td>Interfaces and Processes of Carrier Generator Engine</td>
<td>83</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Attitude determination ranging error sources (1σ)</td>
<td>93</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Flow Diagram of Hardware Simulator</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Flow Diagram of Software Simulator</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>GPS Signal Propagation Path</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>Signal Delay Models</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>User’s View of OOP Paradigm</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>Parameter Passing to an Object</td>
<td>37</td>
</tr>
<tr>
<td>4.3</td>
<td>State Machine: Object Controller</td>
<td>38</td>
</tr>
<tr>
<td>4.4</td>
<td>Object Based Functional Block Diagram</td>
<td>42</td>
</tr>
<tr>
<td>4.5</td>
<td>Flow Chart with Verification, Validation, and Exception Handling</td>
<td>52</td>
</tr>
<tr>
<td>4.6</td>
<td>Health Monitoring as WDT process</td>
<td>54</td>
</tr>
<tr>
<td>4.7</td>
<td>OOSDM of an Object</td>
<td>56</td>
</tr>
<tr>
<td>4.8</td>
<td>IO Block Diagram of an Engine in OOSDM</td>
<td>57</td>
</tr>
<tr>
<td>5.1</td>
<td>GPS Signal Propagation Path</td>
<td>59</td>
</tr>
<tr>
<td>5.2</td>
<td>Output of Base Time Generator Engine</td>
<td>65</td>
</tr>
<tr>
<td>5.3</td>
<td>Outputs of Satellite Data Generator Engine: Errors</td>
<td>70</td>
</tr>
<tr>
<td>5.4</td>
<td>Elevation and Azimuth Angle Distributions</td>
<td>71</td>
</tr>
<tr>
<td>5.5</td>
<td>Outputs of Satellite Data Generator Engine: Clock and Thermal Noise Errors</td>
<td>72</td>
</tr>
<tr>
<td>5.6</td>
<td>Outputs of Satellite Data Generator Engine: Range variation</td>
<td>73</td>
</tr>
<tr>
<td>5.7</td>
<td>Code Frequency and Phase</td>
<td>76</td>
</tr>
<tr>
<td>5.8</td>
<td>Carrier Frequency and Phase</td>
<td>81</td>
</tr>
<tr>
<td>5.9</td>
<td>GPS Engine Blocks of Simulator</td>
<td>84</td>
</tr>
</tbody>
</table>
Figure 6.1  Body Reference Frames ................................. 88
Figure 6.2  Rotation in a Coordinate Frame ......................... 89
Figure 6.3  Observation Geometry for Attitude Determination .......... 90
Figure 6.4  Using GPS-Gyan Blocks for Attitude Determination .......... 95

Figure 8.1  Signal and Slot Analogy in an OOSDM Engine ............... 101
Figure 8.2  Sample GUI using Qt .................................. 102
Figure 8.3  Proposed HW Simulator: Extension to Software Only Simulator .... 104
Chapter 1

Background

Midnight of May 1, 2000 marked a new era in global positioning system (GPS), with selective availability (SA) being off, GPS gave engineers and scientists a new tool to obtain reliable position, velocity and timing information (PVT) which is a key to many application of science and technology. Recent advancement in receiver technology has led GPS usage in applications from real time pedestrian navigation to high end application like precise aircraft landing, real-time spacecraft navigation, formation flying, three-axis attitude control, precise time synchronization, precision orbit determination and many more to be innovated. GPS characteristics are attracting many researchers to use GPS signals and measurements in different fields of science and engineering. Apart from using real GPS signals, researchers are also using GPS signal simulators to generate measurements for scenarios where measurements collection from real GPS satellites is difficult. With increasing use of GPS simulators for algorithm development, it has opened an area for an active design and development of better GPS/GNSS simulators. In this thesis we have made an effort to realize open-source GPS/GNSS simulator architecture and a basic simulator to generate GPS signals and measurements.
1.1 Introduction

Initially GPS was designed for military applications. GPS ability to give accurate position, velocity, and timing (PVT) solution attracted its civilian use. GPS satellite sends coarse acquisition (C/A) code which a civilian receiver can use to attain PVT solutions. PVT information is the fundamental characteristic of GPS signal that is being used in many terrestrial and civilian aerospace applications. Other useful characteristics of GPS signals are:

- **Multiple synchronized frequencies**: GPS satellite currently transmits ranging signal at two different frequencies, L1 (Link1) at 1575.42 MHz and L2C (Link2 Civilian) at 1227.60 MHz. This characteristic is heavily exploited in weather science, radio occultation, and areas where characteristics of a medium need to be measured.

- **Precise time pulse**: GPS satellites clock are synchronized and stability of clock is of atomic precision. This characteristic is exploited in digital communication to synchronize the transmitter, for example 3G towers are synchronized using GPS timing signals.

- **Doppler measurement**: GPS signals can be exploited to measure Doppler associated with a moving target.

- **Predictable error budget**: GPS errors don’t grow with time; it always remains under the designed error budget. This characteristic is exploited for correction of errors in kinematics sensors like INS, gyroscope, and accelerometer.

- **Availability**: GPS signals are available on or above the earth surface till high earth orbits and under any weather conditions.

PVT and above characteristics of GPS signals makes it a powerful tool for scientific research in all three domains: terrestrial, aerospace, and space. A good receiver can provide measurements that can enable users to exploit all of above mentioned characteristics of GPS signals for scientific research. These measurements from the receiver are called **raw measurements** or **raw**
data in GPS receiver terminology. Thus, we require raw data in addition to PVT information from the GPS receiver to use GPS signals and measurements in different applications. For terrestrial domain; we can take a receiver in open sky to collect the raw measurements, but same is not the case for aerospace and space domain. Hence, research in different domains requires different measurements under different environmental and kinematic conditions. The cost and time involved for an active research that uses GPS signal in different domains can be accounted as:

- Terrestrial domain: Research that uses GPS signals received on earth
  - Cost of receiver is low, from $50 to $500
  - Raw data can be captured anytime
  - Algorithm can be modified real time
  - Less time and cost to acquire infrastructure for research

- Aerospace domain: Research that uses GPS signals received in air
  - Cost of receiver is high, from $500 to $5000
  - Receiver needs to be in air to capture the raw data (for real GPS signals)
  - Raw data is captured from simulation facility to avoid time and cost involved in collection of raw data from GPS satellites.
  - Cost of simulator is high
  - Overall high time and cost to acquire infrastructure for research

- Space domain: Research that uses GPS signals received in space
  - Cost of receiver is high, from $1000 to $10000
  - Receiver needs to be orbiting in space to capture the raw data (for real GPS signals)
- Data transmission from the satellite to the user takes time
- Raw data is captured from simulation facility to avoid time and cost involved in collection of raw data from GPS satellites
- Cost of simulator is high
- Overall high time and cost to acquire infrastructure for research

From above classification we can state that a good **simulation facility** is required to pursue an active research which uses GPS signal characteristics. There are three types of simulators:

- Hardware signal simulator: This generates close to real coded electromagnetic signal for receiver.
- Software signal simulator: This generates software intermediate frequency and measurement data, no hardware involved.
- Record and playback simulator: This mimics the real GPS signal by first recording it in an environment and then replaying it in lab environment.

GPS simulators currently available in the market are proprietary and expensive. Thus, there is a huge cost involved in setting up a good facility for research and development that uses GPS signal. For example, the best hardware simulator in market provides a fully functional 12 channel, three carrier GPS signal simulation environment at a cost of approximately half a million dollars. Low cost GPS software simulation environments available but they come with limited functionality. High cost of simulators and complexity involved in using the characteristics measurement makes one of the marvel tools in modern science being underutilized in its application. Some of the current bottlenecks that a researcher faces are:

- High cost of receiver for space and aerospace application
- High cost of simulation environment
• Ad hoc architecture of existing software simulators

• Receiver dependent messages

• Platform dependency

Efforts had been done by the research community to narrow down these bottlenecks. The International GNSS Service (IGS), formerly the International GPS Service, is a voluntary federation of more than 200 worldwide agencies that pool resources and permanent GPS and GLONASS station data to generate precise GPS and GLONASS measurements. The IGS provides the highest quality data and products as the standard for Global Navigation Satellite Systems (GNSS) in support of Earth science research, multidisciplinary applications, and education. Currently the IGS supports two systems, GPS and the Russian GLONASS, and intends to incorporate future GNSS like European Galileo and Japanese QZNSS [16]. IGS also provides guidelines for receiver manufacturers to output precise orbital and measurement information in a standard receiver independent file format called RINEX. This allows the user to post-process the received data to produce a more accurate solution [17].

IGS services have been proven useful to many researchers involved in using GPS measurement for terrestrial applications but IGS lacks in providing measurement of user’s desired trajectories. Researchers involved in using GPS signals for space and aerospace applications have to rely on GPS simulators for measurements. This calls for an urgent need of an open service or a low cost simulator or open source simulator which can generate GPS raw measurements for scenarios where real signal emulation is costly.

1.2 Outcome and Impact

The outcome of this thesis is an open-source GPS signals and measurements software simulator to generate GPS measurements at different receiver’s scenario desired by user. We propose an object oriented system design and modeling (OOSDM) architecture of simulator
that will be able to expand and incorporate future GPS and GNSS signals. Our designed architecture will also allow researchers and scientist to modify the existing models used for channel modeling and able to change parameters of simulation to their needs. The broader impact of our work would be the participation of community in this open source GPS signal simulator.

1.3 Approach and Outline

Approach taken in design and development of this open source GNSS simulator is described in following steps:

1. Review of related work: In Chapter 2, we reviewed the related work and highlight the merits and demerits of existing GPS simulators by different companies. We looked upon the work done by UT-Austin’s GPSTk library and other post-processing tools for GPS. We also made an effort to re-use the work done by various open-source libraries like RTKLIB [21] and GPSTk libraries maintained by University of Texas, Austin [20].

2. Understand the basic simulation needs: GPS simulator design requires a clear understanding of GPS signals and its mathematical relations. It also requires the understanding of all the models that need to be simulated to generate real time software based raw signal and measurements. This is covered in Chapter 3.

3. Identify the models: Our simulator doesn’t re-invent the wheel but uses the proven channel models provided by GPSTk libraries maintained by University of Texas, Austin [20].

4. Design the software architecture: Good software architecture is required to support future upgrade and expansion of this software. Good architecture is also necessary for the users to understand the models used, and allows them to easily change it. This is covered in Chapter 4.
5. Architect the simulator on designed architecture: This designs the simulator on the proposed object oriented system design and modeling (OOSDM) architecture. This is covered in Chapter 5.

6. Result and Performance: This section highlights the performance of the simulator. The performance of each block of the simulator is combined with the results described in separate sections given Chapter 5.

7. Case Study: The approach of using our simulator for attitude determination is done in Chapter 6.

8. Limitation and Future Work: Some of the current limitation of our designed simulator is stated in Chapter 7. We designed our software architecture (OOSDM) to support expandability, some of the future work that can be done to extend our work is described in Chapter 8.
Chapter 2

Related Work

A number of GPS tools for real-time measurements, signals processing, and post-processing of measurements are available. As discussed in Chapter 1, for aerospace and space application development, scientist and engineers rely mainly upon hardware signal simulators. Hardware signal simulator gives the analog RF signals at programmed L1, L2C or L5 frequencies which can be directly fed into the GPS receiver to validate the receiver algorithms and hardware. Software simulator also simulates the signal but without converting it to analog domain i.e. it provides the digital signal which can be used for only algorithm validation without the need of a receiver RF frontend. This chapter analyzes existing GPS hardware and software simulator and discusses merits and demerits of both to bring up our motivation behind the design of sustainable and extendable software architecture and a low cost GPS simulator for diverse applications.

2.1 Hardware GNSS Simulators

Hardware simulators are the best one can get for testing a GPS/GNSS receiver under different scenarios. Hardware simulators are designed by modeling GPS/GNSS satellites and the channel to provide near to real radio frequency (RF) signals. This section gives an insight to a basic flow diagram of hardware simulator and further compares some of the available
hardware simulators for their capabilities and cost involved in acquiring those capabilities.

2.1.1 Flow Diagram

Hardware simulator system consists of a user interactive section (mostly a workstation). User specifies the parameters like almanacs, trajectory, clock, and other simulation specific parameters for simulation. Workstation then uses those inputs, generates parameters for signal generations and passes it to the simulation hardware. Simulation hardware uses the simulation model generated by the workstation and produces the RF analog signals. A general flow diagram is shown in Figure 2.1.

Once the simulation environment is set, user has only limited control, like power and noise control, over the signal being simulated. This limits user capability to change the channel model during run time. Record and playback type of simulator falls in the category of hardware simulators that works independent of workstation. In such simulators, GPS signals for a particular scenario are recorded and then loaded to simulator hardware to playback the recorded signals [14, 15]. Next sub-section reviews some of the hardware simulators available in the market and their merits and de-merits.

2.1.2 Existing Product Review

From the airplane pilot operating a Boeing 747, to the everyday consumer using a GPS navigation system in his car, to the hobbyist searching for buried treasure in the forest, GPS technology is quickly becoming integrated in a wide variety of applications. As innovation drives GPS receivers to even better performance, techniques used to characterize performance are
Table 2.1: Comparison Chart

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Company</th>
<th>Simulator Type</th>
<th>Channels</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSS8000</td>
<td>Spirent Fedrel</td>
<td>Hardware</td>
<td>12, GPS</td>
<td>~$ 500,000</td>
</tr>
<tr>
<td>CAST Nav</td>
<td>CAST Navigation</td>
<td>Hardware</td>
<td>12, GPS</td>
<td>~$ 300,000</td>
</tr>
<tr>
<td>NI GPS Sim</td>
<td>National Instruments</td>
<td>Hardware</td>
<td>12, GPS-L1</td>
<td>~$100,000</td>
</tr>
<tr>
<td>GPS Toolbox</td>
<td>Navsys</td>
<td>Record &amp; Playback</td>
<td>12, GPS-L1</td>
<td>~$60,000</td>
</tr>
</tbody>
</table>

becoming increasingly sophisticated as well, thus many companies provide end to end simulation capabilities. This section lists and compares some of the popular product and their capabilities.

Table 2.1 highlights some of the most popular GNSS simulators. Some of the simulators like GSS8000 supports all GPS frequencies and have capabilities to introduce interference, multipath, and noise in the generated signals. These advanced features come with an additional cost which increases the cost of simulator. GPS receivers and hardware simulators are widely used to collect raw data and measurements for applications in aerospace and space applications, but they lack in providing the offline raw measurements and custom modeling of channel. Some of the missing measurements and processes of these simulators are:

- Capability to tap raw measurement without GPS receiver
- Upgrade to new signals without additional cost
- Support for Multiple GNSS system
- Open and accessible mathematical models

GNSS systems have been proven to be an important asset. Apart from US and Russia other developed and developing nations started making their own infrastructure and constellation of satellites. Some of them are:

- Galileo: European Union
- GLONASS: Russia
• QZNSS: Japan
• Compass: China
• IRNSS: India

With more GNSS systems in orbit, better and reliable measurements can be done. To use these additional capabilities, hardware simulators need to be upgraded accordingly which involves additional cost. Software simulators are beneficial in this aspect as any new upgrade in system can be easily deployed in software with less cost.

2.2 Software GNSS Simulators

Software simulators are flexible tradeoff one can get for testing a GPS/GNSS receiver. Software simulators are designed by modeling GPS/GNSS satellites and the channel in software using almanac data and then coding the electromagnetic signal in software. This section gives an insight to a basic flow diagram of software simulator and further analyzes some of the available software simulators and looks into their capabilities.

2.2.1 Software Simulator Flow Diagram

Software simulator system consists of a user interactive section (mostly a workstation) where user specifies the parameters like almanacs, trajectory and other parameters required for simulation. With this information, the software determines the range and timing information to generate PRN coded sinusoids in software. A general flow diagram is shown in Figure 2.2.

2.2.2 Existing Product Review

Large numbers of software simulators are based on MATLAB. IGS site [18] provides links to popular software toolboxes available for use. These toolboxes are useful but were made for dedicated applications without proper documentations to use it. Some of the companies also provides dedicated software tools but again are dedicated to generate single GPS L1 frequency.
Table 2.2: Comparison Chart : Software Simulators

<table>
<thead>
<tr>
<th>Product Name</th>
<th>Company</th>
<th>Simulator Type</th>
<th>Channels &amp; Frequency</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPSlab</td>
<td>Accord Software</td>
<td>Software</td>
<td>single channel, GPS</td>
<td>$4,000</td>
</tr>
<tr>
<td>GPS Toolbox</td>
<td>L3 Nav</td>
<td>Matlab</td>
<td>12 Channels, GPS</td>
<td>$3,000</td>
</tr>
<tr>
<td>GNSS Sim</td>
<td>CRS</td>
<td>Software</td>
<td>12 channels, GPS</td>
<td>$10,000</td>
</tr>
</tbody>
</table>

Table 2.2 summarizes some of the popular software toolboxes for GPS simulation.

According to Table 2.2 popular GNSS software simulators are again tailored only towards GPS and do not support other GNSS Systems. Some of the simulator like GNSS Sim [19] is a good choice for generating measurements but manufacturer restricts complete features to a non-military customer. Some of the envisaged limitations of these software simulators are:

- Lacks new frequencies support, e.g. L2C and L5
- Lacks multiple GNSS system support, e.g. GPS and GLONASS
- Architecture is not designed to support future GNSS systems, e.g. Galileo, IRNSS, Compass, and QZNSS
- Ad-hoc software architecture which limits expansion
- Huge amount of data handling
• Accuracy of modeling

Software models are always more flexible. The major limitation is the accuracy of modeling real environment like dynamics, ionosphere and other errors. Closed source doesn’t allow developers to tweak the model for their applications. With growing applications of GPS, this field attracts developers to make an open source flexible simulator.

2.3 GNSS Post Processing Tools

This category of tools are not exactly signal simulators, but tools that does post processing of data collected from a receiver. Most of the survey grade receiver manufacturer provides such tools to process the data collected from their GPS receivers. These tools are designed to process proprietary receiver’s messages and also the IGS standard RINEX files. Some of the open source tools like GPSTk [20] and RTKLIB [21] provide useful packages for GPS post processing and data analysis tools. GPSTk comes as an open source library and have range of libraries from space geodesy to RINEX file processing. These tools and library though provides good insight to collected GPS measurement, but works in offline mode or needs a GPS receiver to feed the measurements.

With richness of modeling that has been done by community, these libraries can be very useful in designing a simulator that can generate the signals and measurements. Some of the benefits of GPSTk and RTKLIB are mentioned in next subsection.

2.3.1 GPSTk

The GPSTk [22] distribution contains a library, a collection of applications, and a test suite. The library consists of the base functionality used to develop applications that read, process, and write GPS data. The applications are standalone utilities that are valuable for examining and processing GPS data and for providing more detailed programming examples. The test
suite includes programs that test parts of the library. The test programs provide not only validation of a new installation but also additional examples of the how to use the library in standalone programs. The GPSTk library supports a wide range of functionality, including, but not limited to following libraries:

- **RINEX utilities**: The RINEX utilities provide a set of applications that can be used to examine, manipulate, and plot RINEX observation files.

- **Positioning**: The positioning applications include two different applications that perform standard pseudorange-based positioning and two that implement differential phase-based solutions.

- **Residual analysis**: A residual analysis application computes two types of measurement residuals using a single receiver or two receivers in a zero baseline configuration.

- **Ionospheric modeling**: The ionospheric modeling applications utilize the two frequency TEC estimate from the RINEX utilities and compute a model of the ionosphere.

- **Signal Tracking Simulation**: These utilities simulate the tracking of GPS C/A and P-code.

- **Basic transformations**: Conversions of time and coordinate systems.

- **Observation data collection and conversion**: Translating receiver specific data formats to RINEX.

- **File comparison and validation**: Differing observations files against a truth source.

- **Data editing**: Simple editing like systematic removal of observations by satellite, type or time and more advanced editing like cycle slip detection and correction.

- **Autonomous and relative positioning**: Navigation and surveying applications.
These libraries are useful in modeling various subsection of signal simulator like ionosphere, troposphere. Open source nature of this library allow user to modify the models and re-use the design code.

2.3.2 RTKLIB

RTKLIB is an open source program package for RTK-GPS/GNSS. RTKLIB consists of a simple and portable program library and several application programs (APs) utilizing the library. The program library of RTKLIB provides:

- Matrix and vector functions
- Time and string functions
- Coordinates transformation and geoid model
- Navigation processing
- Positioning models (troposphere, ionosphere, and antenna phase center variation)
- Single point positioning
- Carrier-based and code-based relative positioning
- On the fly (OTF) integer ambiguity resolution
- Receiver raw binary data input
- Positioning solution and National Marine Electronics Association (NMEA) input and output.
- RINEX observation data/navigation message input/output
• Precise ephemeris input

RTKLIB is implemented in ANSI C (99) and supports GUI that can be used to process data from receivers. The library model is accurate but architecture is not object oriented and hence was not chosen for the need of our simulator which required a design based on object oriented approach.

Library models are always more flexible as many applications can be built using these libraries. GPSTk provides all useful features but requires learning curve and programming to use the libraries for required application. User community using GPSTk has developed some utilities for signal generation but are tailored to the designer needs and submitted to the repositories.

2.4 Motivation

This work involves the design of an open source GPS simulator. Factors that inspired this work are:

• Increasing use of GPS in science and space application.

• Lack of an open-source simulator to generate measurements for user defined scenarios.

• Need of a software architecture that supports a reliable system and software development.

2.4.1 GPS in Science and Space Applications

It is not surprising that there is interest in expanding the utility of GPS to a wider range of space missions for a new era of small satellites which include cubesats. With improved navigation performance, LEO and HEO space vehicles will enable new engineering and science innovations, including improved Earth and Space weather prediction, space vehicle formation
flying, and Earth and Space science research, exploration missions to the Moon and beyond, as well as military applications.

Low cost small satellite design is a big challenge as it is constrained by small size and constrained power. GPS receivers find a huge application in small satellite as a multipurpose sensor because of its low cost, miniaturization and low power. A single GPS receiver can be used for orbit determination, attitude control, spacecraft timing autonomy and useful mission payload for scientific exploration as mentioned above, hence would serve as a useful multipurpose subsystem. Since the envisaged role of GPS in small satellite is critical to the functionality of satellite and mission specific goals, a robust design and development of GPS receiver is needed to cater to all applications. Some other applications of GPS measurements apart from standard applications of determining position, velocity, and timing for space domain can be listed as follows:

- Mission specific applications: These applications use GPS signals and measurements in different space science and physics experiments.
  - Probing of Earth’s magnetosphere
  - Formation Flying
  - Study of celestial object using optical interferometry
  - Radio Occultation
  - Space weather and many more to be innovated

- Spacecraft Related Applications: These are the applications of GPS signal and measurements that helps user to control and navigate a satellite in more reliable manner.
  - Three axis attitude determination
  - Precise orbit determination
In Chapter 1, we discussed the huge cost associated with GPS/GNSS simulators and complexity involved in using the raw measurements in different applications. For example, an attitude determination system (see Chapter 6) requires a minimum of two GPS sensor’s carrier phase data to design and analyze the feasibility of attitude determination algorithm. Conventional way to make this research setup is to get two space qualified receivers that produce carrier phase data and a GPS signal simulator. This setup would cost more than half a million dollars. An alternative way is to get GPS measurements at earth using low cost receivers as research setup. The latter approach, although more realistic, misses the gist of environment where spacecraft’s dynamics, channel and other physical attributes are missing and can be modeled by a simulator. This highlights a definite need of an open-source and low cost for scientific community using GPS signals and measurement in their applications.

2.4.2 Lack of an Open-Source Simulator

Sections 2.1, 2.2, and 2.3 lists various tools available to user, but none of them completely caters to the need of application developers and researchers. Three major limitations of these tools are:

- High cost for hardware simulators
- Ad-hoc implementations of software simulators
- Closed source

Models presented in some of the open source libraries (refer Section 2.3) can be used for signal simulation also. Huge processing power is required to generate signals in software. The huge number crunching and nonlinear modeling can be achieved in real time due to increase in computational performance of a personal workstation.
2.4.3 Need of a Software Architecture

Software simulators discussed in Section 2.2 clearly reflect their failure due to ad-hoc software architecture. Usage of software simulator is limited because they are:

- Less flexible
- No verification and validation of model used
- No information on quality of modeling
- Limits expandability
- Limits compatibility

Above listed limitation are all requirement of good software architecture. Since software bugs can corrupt the measurement there is a need for quality processes like verification and validation on the data generated.

Our work is motivated by the needs mentioned in each subsections 2.4.1, 2.4.2, and 2.4.3. We gave main emphasis in design of OOSDM architecture, as this holds the key for successful expansion and modification to our simulator: GPS-Gyan. In next chapter we will discuss some of the standard terminology of GPS, study the different segments of a GPS system, and identify sections in each segment that needs to be modeled.
Chapter 3

GPS Signal Theory

The Global Positioning System (GPS), though a generic term, also referred as US version of space-based Global Navigation Satellite System (GNSS). Many believe it as an outcome of cold war as almost in parallel Russia (formerly USSR) also designed their GNSS named GLONASS (GLObal'naya NAvigatsionnaya Sputnikovaya Sistema; "GLObal NAvigation Satellite System" in English). Purpose of both is to provide reliable positioning, navigation, and timing services to worldwide users on a continuous basis in all weather, day and night, anywhere on or near the Earth in clear line of sight. Since the class of application is same and GPS being widely used, the acronym GPS and GNSS would be used interchangeably throughout this material unless the difference is clearly stated. GPS is made up of three segments.

- Space
- Control
- User

The Space Segment is composed of 24 to 32 satellites in Medium Earth Orbit and also includes the boosters required to launch them into orbit. The Control Segment is composed of
a Master Control Station, an Alternate Master Control Station, and a host of dedicated and shared Ground Antennas and Monitor Stations. The User Segment is composed of hundreds of thousands of U.S. and allied military users of the secure GPS Precise Positioning Service and tens of millions of civil, commercial and scientific users of the Standard Positioning Service (see GPS navigation devices). GPS satellites broadcast signals from space that GPS receivers use to provide three-dimensional location (latitude, longitude, and altitude) plus precise time. This chapter discuss basics of GPS signal theory and errors involved in GPS signal measurement. Interested readers can look on to references [51, 41] for the details on GPS functionality and how a GPS receiver works. The intent of this chapter is to develop the mathematical notation for errors involved in GPS signal. These errors will be modeled in the simulator to generate a pseudo signal near to the real signal.

GPS Signal Model

This section develops a signal model of a general GNSS satellite with focus on existing GPS satellites. First a simple communication link model is developed which later will be convolved w.r.t navigation signal models as shown in 3.1. Two major navigation measurement signal model are carrier phase measurements model and code phase (also known as pseudo-measurement model). The model equations are based on works done by Mark L. Psiaki and Shan Mohiuddin, Cornell University [1] with some modification done to make it for general for GNSS satellite. As GNSS system is a time based system so the section starts with modeling the time then continues on modeling the communication, channel, and other signal attributes.
3.1 Time Based Systems

Time based system (clocks) categorization is one of the first and most important model parameter in the modeling equation. To sustain the time synchronization for the GNSS system we need to start with a reference time which is in turn referred to earth’s common reference GMT time. This reference is called the true time of the GNSS satellite system and in GPS system it’s referred to “GPS” time. The second time is the user equipment clock called as receiver time. Receiver equipment depending upon the application incorporates different clocks in terms of stability and drift. The uncertainties in receiver clocks are measured by its bias from the base frequency (it was originally tuned by manufacturer) and drift rate at which the base frequency is drifts. These clock errors are modeled using white Gaussian noise process. True 'GNSS' time is denoted by the variable $t$ and w.r.t the true GNSS time $t$, the receiver clock time $t_R$ at receiver has relationship:
\[ t = t_R - \delta t_R \]  \hspace{1cm} (3.1)

where \( \delta t_R \) is the receiver clock error.

The third time that we need to consider is the satellite clock. Satellite uses highly stable clocks and its bias and drift parameters modeled by ground station and sent to the user. Satellite clock is referenced to true time and related by

\[ t^j = t + a^j_{f0} + a^j_{f1}(t - t^j_{oc}) + \delta t^j_{rel} \]  \hspace{1cm} (3.2)

where \( a^j_{f0} \) relate to the bias of the satellite clock and \( a^j_{f1} \) gives the clock drift value. These values are sent to the user in the navigation data message by satellite. The drift value is the predicted value from the epoch time \( t^j_{oc} \) which is again provided in navigation data message. \( \delta t^j_{rel} \) is a relativistic correction which depends on eccentricity, semi-major axis, and eccentric anomaly of GPS satellite orbit \[6\]. GPS clocks prior to launch are set to run at GPS time with an offset for relativity so that while in orbit it is assumed that the satellite transmits modulated electromagnetic signal at time instant determined by the actual designed satellite atomic clock frequency. In order to correctly model the navigation measurement equation we need to have a track of all these clock models. Satellite clock modeling is used in generation of carrier phase error and code phase error originated from the modulator and code generator at satellite which internally is clocked by satellite clock. Receiver clock will be used to generate the local replica of the carrier and code frequency.

### 3.2 Signal Delay Models

GPS is a time based system. The time taken for an electromagnetic (EM) wave transmitted from the GPS satellite transmitter to reach the receiver section is called the propagation delay of signal, this is shown in Figure 3.1. The accuracy of all the measurements are directly related to accuracy of this delay measurements which get affected by many physical phenomenon.
Figure 3.2: Signal Delay Models
Figure 3.2 shows the major sources of errors that add to the propagation delay of the EM wave. This section explains all these errors and establishes a final propagation delay equation that accounts for all the delays introduced by these errors.

### 3.2.1 Satellite Clock Errors

GPS satellites use highly stable atomic clocks. These clock errors are transmitted as a part of navigation messages by GPS satellites. Equation 3.2 is used to model the satellite time and the errors due to the bias and drift of satellite clock.

### 3.2.2 Antenna Phase Center Variation ($\phi_{pcv}$)

Antenna phase center variations also produce a carrier phase error that is a function of the arrival signal direction and the user spacecraft attitude. This error source may yield a carrier phase perturbation as large as $(0.01 \text{ m})/\lambda$ cycles [13].

### 3.2.3 Polarization-Induced Carrier Phase Wind-Up ($\phi_{pwu}$)

The circular polarization of the GPS signal causes a carrier phase perturbation that is a function of the relative geometries of the transmitting and receiving antennas. This geometry is characterized by the unit direction vector that points from GPS satellite $j$ to user receiver. The periodic phase variations can be different for different GPS satellite signals and for different receivers [1].

### 3.2.4 Free Space Propagation Delay ($\delta t_{trn}$)

Propagation delay is the time taken by the coded electromagnetic signal to reach the receiver antenna. At a given time $t$ and in a given coordinate frame if we know the satellite position vector $\mathbf{r}_j$ for the $j^{th}$ satellite and user antenna position vector $\mathbf{r}_{rx}$, the nominal propagation delay is the geometric range between the receiver and transmitter divided by the speed of the
\[ \delta t_{trn}^j = \frac{p_j^i}{c} = \frac{\sqrt{[r_{rx}(t) - r_j(t - \delta t_{trn}^j)]^T[r_{rx}(t) - r_j(t - \delta t_{trn}^j)]}}{c}, \]

where \( c \) is the speed of light. This relationship currently does not include the errors and delay in the transmission path. Also, the position of satellite should be computed at the transmit time \( \delta t_{trn}^j \), which is implicitly receive time minus the transmission delay, hence it appears on both side of the relationship.

### 3.2.5 Ionospheric Delay (\( \delta t_{\text{ion}}^j \))

The ionosphere is a dispersive medium located primarily in the region of the atmosphere between about 70 km and 1,000 km above the Earth’s surface. Within this region, ultraviolet rays from the sun ionize a portion of gas molecules and release free electrons. These free electrons influence electromagnetic wave propagation, including the GPS satellite signal broadcasts. Refraction is the main physical phenomenon that governs the delay caused by ionosphere and it depends upon:

- Refractive index of the medium
- Angle of incidence or elevation angle of satellite from receiver
- Frequency of the incident wave in the dispersive medium

Depending on the frequency of the incident wave the ionosphere advances or deters the signal. Refractive index of phase propagation in the ionosphere can be approximated as Taylor series. By neglecting the higher order terms the path delay due to the refraction and the contribution in the transmission delay for the \( j^{th} \) satellite is given by [1, 12]
\[
\delta t_{ion}^j = \frac{40.3 TEC^j}{cf^2},
\]

\[
\delta t_{trn}^j = \sqrt{[r_{rx}(t - \delta t_{ln}) - r_j(t - \delta t_{trn})]^T [r_{rx}(t - \delta t_{ln}) - r_j(t - \delta t_{trn})]} + \delta t_{ln} + \delta t_{ion}^j,
\]

where \( f \) is the transmitted carrier frequency of GPS and \( TEC^j \) is the total electron content integrated along the line-of-sight vector from receiver to \( j^{th} \) GPS satellite.

### 3.2.6 Tropospheric Delay (\( \delta t_{tropo} \))

The troposphere is the lower part of the atmosphere that is non-dispersive for frequencies up to 15 GHz\[12\]. Within this medium, the phase and group velocities associated with the GPS carrier and signal information (PRN code and navigation data) on both L1 and L2 are equally delayed with respect to free-space propagation. This delay is a function of the tropospheric refractive index, which is dependent on the local temperature, pressure, and relative humidity. Left uncompensated, the range equivalent of this delay can vary from about 2.4m for a satellite at the zenith and the user at sea level to about 25m for a satellite at an elevation angle of approximately 5º\[12\]. GPSTk libraries are used in modeling this delay.

### 3.2.7 Multipath and Shadowing Effects (\( \phi_{mp} \))

One of the most significant errors incurred in the receiver measurement process is multipath. Multipath errors affect both pseudorange and carrier-phase measurements. Multipath errors can vary significantly in magnitude depending on the environment within which the receiver is located, satellite elevation angle, receiver signal processing, antenna gain pattern, and signal characteristics\[12\].
3.2.8 Line Bias ($\delta t_{ln}$)

GPS signal travel through the different propagation medium till it reaches the receiver antenna. After reaching antenna, it propagates through the wired medium in receiver electronics. Receivers have some propagation delay associated with the cables from antenna to the point signal is processed. This delay depends on the type of component used in the RF section and frequency of the signal passed on those components. Major contributors on these delay includes:

- Line delay of antenna cables
- Receiver RF section analog components group delays
- Receiver RF wires, connections and joints

Line bias and other delays for a receiver at a given temperature and for a single GPS frequency [23] are modeled as a constant common-mode receiver dependent error and are denoted by a single term $\delta t_{ln}$. Group delays of the RF components vary with frequency and have different values for different GPS L1, L2C, and L5 frequencies. In equation formulation we assumed single frequency (L1) receiver. For a dual or multiple frequencies receiver appropriate values of line bias for each frequency should be added in the Equation 3.6. As mentioned, GPS is a timing based system, line bias delay also contributes to the transmission delay hence Equation 3.3 becomes:

$$\delta t_{trn}^j = \rho^j = \frac{\sqrt{[r_{rx}(t-\delta t_{ln}) - r^j(t-\delta t_{trn})]^T[r_{rx}(t-\delta t_{ln}) - r^j(t-\delta t_{trn})]}}{c} + \delta t_{ln}, \quad (3.6)$$

where the line bias term $\delta t_{ln}$ is added to the delay. This delay also is required to measure range precisely for position and carrier phase determination.
3.2.9 Receiver Noise \( (n_{\phi}^j) \)

Thermal noise is one of the common sources of error in any digital receiver. This are the result of random electrical noise received by antenna and generated by RF component. This depends on the noise figure of the RF section [4]. Measurement errors and noise are also induced by the receiver tracking loops software and the bandwidth associated with it. In terms of the DLL (delay lock loop) used for carrier synchronization and tracking the code, the dominant sources of code phase measurement error (excluding multipath) are thermal noise jitter and the effects of interference. For a typical modern receiver under nominal conditions (i.e., without external interference), 1\( \sigma \) error of noise and resolution are in the order of a decimeter or less. This is negligible compared to errors induced by multipath but the receiver noise and resolution errors affect carrier phase measurements made by a PLL [9] for carrier phase positioning. PLL measurements errors in nominal conditions are in the order of 1.2 mm (1\( \sigma \)) when tracking the C/A code and 1.6 mm (1\( \sigma \)) when tracking the P(Y) code [12]. In total, the receiver noise error mainly depends upon the bandwidth of the receiver’s phase-lock loops (PLL) and carrier-to-noise ratio, \( C/N_0 \). The standard deviation of this error is given by [1]:

\[
\sigma_{n_{\phi}}^j = \frac{1}{2\pi} \sqrt{\frac{B_{PLL}}{2C/N_0}} \text{cycles.} \tag{3.7}
\]

3.3 Signal Delay Model Equation

GPS signal propagates through different mediums before reaching the receiver antenna. Some of the physical phenomenon associated with each medium has tangible effect on the signal propagation path of the GPS signals. These physical phenomenon includes sun’s position, moon position, sagnac, special relativistic effect, satellite antenna pattern, satellite attitude and many more. It’s hard to model all of these physical phenomenon but some of them which can be modeled are described in Section 3.2. The complete delay model for the signal can be written as:
\[ \delta t_{trn}^j = \frac{\sqrt{\left|r_{rs}(t-\delta t_{tn})-r^j(t-\delta t_{tn})\right|^2|\left|r_{rs}(t-\delta t_{tn})-r^j(t-\delta t_{tn})\right|}}{c} + \delta t_{ln} + \delta t_{ion}^j + \delta t_{tropo}^j + f_{L1}(n_j^i + \phi_{pcv} + \phi_{pwu} + \phi_{mp}). \] (3.8)

Equation 3.8 assumes L1 (Link 1 at 1575.42 MHz) as the frequency of the signal transmission. For L2C and L5 frequencies the term and delay associated with that frequency should be used as described in Sections 3.2.5 and 3.2.8.

This chapter highlighted the major sources of errors that get introduced in a pure signal generated from the satellite. Error budget and modeling is the major task in a simulator design to mimic the real noisy GPS measurement. We have used GPSTk library [20] to model errors like ionosphere, troposphere, and phase center variation in our GPS-Gyan simulator. Chapter 5 focuses on the design of the error models described in this chapter.
Chapter 4

Software Design Architecture

4.1 Philosophy

Design of a GPS simulator not only requires an adequate knowledge of the GPS signals and its processing, but it also involves a good amount of software design and architecture. The proposed simulator encapsulates following software requirements:

- Design for re-usability
- Design for portability
- Design for verification
- Design for validation
- Design for testability
- Design for platform independence
- Design to interface, and inputs to the module.
These requirements dictate design patterns of a software design cycle. There are adequate amount of literature on good design patterns that should be followed in a software design [24]. In addition to good software architecture, the simulator should support expandability. Some of the additional GPS related modules and desired feature in the base software can be:

- Ability to add noise models
- Ability to integrate with INS, Gyro, and Accelerometer
- Ability to use already designed subsection independently
- Ability to add user designed modules and plug-ins

Requirements listed above can be fulfilled by a well designed architecture and a software development kit (SDK), so that developers can extend the capability of the existing simulator. This chapter architects a scalable software architecture which incorporates all listed design and functional requirements. The chapter starts with the philosophy behind object oriented programming (OOP) and goes on extending that philosophy by adding some additional procedures which maps the entire design requirement. This extension is named as object oriented system design and modeling architecture (OOSDM). Object oriented approach treats each individual subsystem as an independent object and complete simulator is a system based on collection of such objects [24]. An attempt is made to make the software architecture generic and discipline independent so that it not only is limited to this simulator, but can be used in modeling designs across fields.
4.2 Object Oriented Programming (OOP)

Object-oriented programming (OOP) represents an attempt to make programs more closely model the way people think about and deal with the real world procedures to perform actions. Normally to perform a task we need some input, parameters and the knowledge of method to perform the task. A user demanding a task from a task performer doesn’t necessarily know how the task is performed, given it is performed as demanded by the user under the given constraints and requirements. This is the basic approach used in OOP which lays out the methodology to write programs that closely models the real world.

Object-orientation is a set of tools and methods that enable designers to build reliable, user friendly, maintainable, well documented, reusable software or systems that fulfills the requirements of its users. An object-oriented programming is based on following concept:

- Objects and Classes
- Inheritance
- Polymorphism and Dynamic binding

Software’s based on OOP follows a software design cycle that uses OOP concepts to achieve functionality. Simulator designed in this work also models a real world system (GPS) in this case. GPS system is made of various subsystems which performs dedicated operations. OOP paradigm is suitable for the GPS simulator modeling, but it doesn’t layout all the processes that will ensure a reliable system design. One such process is verification and validation of inputs and outputs. This is an important process that both user and designer engineer desire as it makes software and systems more reliable. Next sub-section explains users and designers view of an object oriented design and layout how the concepts of OOP can be used to their advantages. Section 4.3 will then extend the existing designers methodology of a standard OOP design to incorporate processes like verification, validation, and health monitoring.
4.2.1 Object Orientated Paradigm: Users View

User here is an abstract entity that requests or requires services. In the OOP paradigm, services are performed by entity named as objects. Each object provides some services or performs actions or runs some algorithms that can be used by a user. For a user to use a particular service offered by an object, it has to perform following actions in sequence:

1. Identify the service required
2. Identify the correct object or objects which offer that service
3. Feed the information and the requirements to the object as per guidelines defined by the object
4. Wait and/or acknowledge the results produced upon the completion of the service.

As long as services performed by the object are acceptable to the user, it doesn’t need to know how the service was performed. Figure 4.1 shows how a user interacts with an object. This section highlights following important processes which enables OOP to model the real world.

- Encapsulation: This process is shields a user from the internal methods used by an object to perform the requested service. This is achieved by information hiding [34].

Figure 4.1: User’s View of OOP Paradigm
• Public Port: User has only access to the public services offered by object.

• Get and Set Mode: This is the process performed by user to set some public accessible parameters of an object.

In the next subsection we discuss the procedures followed by an object to compliment the user actions under OOP paradigm.

4.2.2 Object Orientated Paradigm: Object’s View

In OOP, objects are responsible for perform actions. To perform the requested service it follows OOP dictated processes (a defined step to perform execution). Processes are categorized in different access levels. OOP lists three kinds of access levels [34]:

• Public: Those services and utilities which are intended to be open to public are kept in this level. Any outside entity (a user or an object) have access to this level.

• Protected: This is the level hidden from the public level but accessible to some of the special objects which assists the main object to perform the service. This mimics the real world example. A service provider to perform requested service needs assistance of other service providers. In the process of this assistance, group of service providers share resources. Protected level allows access to all such shared services and resources.

• Private: This level hosts all the processes which are internal to an object. This level is neither open to the public nor accessible to the allied service providers (allied objects).

We can further classify the processes in mutually exclusive abstract processes, which an object follows at each of these levels. Some of the processes are broken down into independent sub-processes.
4.2.2.1 Fetching Inputs: Public

For an object to start performing service, it needs inputs from external world. These inputs are divided in two types depending on their nature.

- Passive inputs: These are the inputs which are fed to the object by the user before the object execution starts.

- Active inputs: These are the inputs which are applied to the object (algorithm) when object is in service mode. Active inputs are sometime missing. For example, consider an object that performs compression of a file. This object requires only the file’s address and performs the operation of compression. But an object computing cyclic redundancy check (CRC) on the data packets at a router keep on receiving new packet and performs same operation without changing any parameters. Thus active inputs are the varying inputs that are fed to an object to get varying outputs.

4.2.2.2 Fetching Controls: Public

These are the parameters and controls which are set and modified by the users. These are offered by an object to the users to help them select how they want their requested services to be performed. For example, a CRC object can perform CRC-16, CRC-32 bit check depending on the user given parameter it will select which to perform. These handy controls are again user accessible (public) and can be of two types.

- Passive parameters: These are the parameters which are applied to the methods (algorithm) by default. This helps in setting up the initial state (mode) of an object.

- Active parameters: These are the parameters which are applied to the methods (algorithm) during the run mode. The active parameters help the controller of the object to decide its actions in an active (dynamic) state.

Figure 4.2 explains how active parameters are useful in a CRC object. Depending on the parameters passed by user at different time, the multiplexer chooses the appropriate CRC
algorithm.

4.2.2.3 Prepare Inputs to Object: Protected

Philosophy of OOP is to make the easy guideline for users to use an object’s services. Main emphasis is on “Ease of use” and user “convenience”. Thus sometimes it’s required to prepare the input before starting the service routine. Normally objects re-format the information fed by user, for optimization and compatibility to its internal services. For an example, to a CRC object, user feed data packet in parallel bits but internally the CRC object routine is optimized for serial data. So this process helps in re-formatting the data internally for compatibility.

4.2.2.4 Object Controller: Protected

This process acts as a manager of the object. It manages all the processes that an object follows to perform the services. This is basically a state machine, which changes its state to properly execute the actions needed to perform the service. This should not be confused with the methods which perform actual actions; it just schedules and manages tasks. Object controller manages following things for an object.
Initialize the object

Sequentially allow input modules to take inputs and parameters

Felicitate input preparation

Feed and manages allied objects

Ask action performer (algorithm) to execute the assigned tasks

Acknowledge the user about the service completion

Ask output port to generate the result

Above tasks resembles timing and control path in a processor design. Thus it can be modeled in a finite state machine. State machine of an object controller is shown in Figure 4.2. Whenever there is a user interrupts or there are some exceptions, the object changes the state under some intelligence from the object controller.
4.2.2.5 Methods of the Object: Protected and Private

These processes contain the main functional blocks of the objects. We define procedures and algorithms that run and produce outputs for the given inputs and parameters. There may be many methods in an object, each for different purposes. Methods are categorized in their respective (public, protected, and private) levels as per their accessibility. The unique methods which are totally independent from the outside world are declared in private levels. It is the role of the object controller to give correct inputs and controls to the methods. Engine controller also schedules the sequence in which the methods perform execution.

4.2.2.6 Output from the Object: Public

This process is responsible for delivering the outputs to the user. It takes the output generated by different methods, package it according to the user requirement and deliver it to the user. Once the user accepts the result, this process relinquishes the link between the object and the user. Normally in OOP, a process called “destructor” is called to do this process.

4.3 Class, Objects and Processes

OOP is now a matured field in computer programming. Lists of high level languages like C++, Java, Perl, Python support object-oriented programming. In these languages, the processes mentioned in Section 4.2, are listed in a template called “Class”. This section tries to relate the class definition to the above mentioned processes. An attempt is made to have a language independent analogy. This section also recommends some of the enhancement to the existing process of OOP, which if added, can make class more accessible and manageable.

4.3.1 Standard Class Definition

In the real world, we will often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an instance of the class of objects known as
bicycles. A class is the blueprint from which individual objects are created [34]. In an OOP language a class definition looks like:

Class <Class name> : <Access level>
{
public:
    /* Input sections */
    /* All public/ user accessible methods and variables (parameter and control) are defined here*/
    Constructor(); // Helps in setting up initial value
    Public SetFunctions();
    InputParameters;
    /* Output sections */
    Public GetFunctions();
    Public GetResults();
protected:
    SharedFunctions();
private:
    CoreFunctions();
};

Above code listing is just an abstract, basic, and language independent class definition. This may vary in implementations. Class definition for an object defines the guidelines and blueprint that an object should follow. These guidelines are condensed and generic as described in Section 4.2.2 on page 35. Some of the processes which are mentioned in Section 4.2.2 like Object controller and Prepare input for an object, are not explicitly present in standard class definition of OOP based software. Normally such processes are convoluted within the methods of a standard object. To have a close resemblance to real world, those explicit processes are included in the proposed extension of class which is defined next.

4.3.2 OOSDM Class Definition

Standard OOP class definition is very generic and can be extended in more refined way as stated in Section 4.2.2. This extended class definition classifies some of the mutually exclusive
processes and declare them at different access levels. The extended class definition looks like:

```cpp
Class <Extended Class name> : <Inheritance level>
{
  public:
    /* Input sections */
    /* All public/user accessible variables (parameter and control) are defined here*/
    Constructor(); // Helps in setting up initial value
    InputServiceRequestToObject();
    InputParamToObject();
    /* Method Operating mode sections */
    Object();
    /* Output sections */
    ObjectOutput();
    /* Exit Process */
    ShutDownObject();
    Destructor();

    protected: /* Defines protected processes*/
    PrepareInputForObject();
    ObjectController();
    SharedFunctions();
    PrepareOutputFromObject();

    private:
    CoreFunctions();
};
```

Above class definition extended a generic class definition by assigning explicit process that needs to be performed at each level of the abstraction. From this class definition, the architecture of an object can be described by Figure 4.4. It resembles a two inputs and one output system.
A brief description of the ports and blocks are given below:

- **Input Port** (Public): Comprises of active and passive input port of the object. A user of this object put service requests or feed input data on these ports. This hosts routines to provide access to these ports.

- **Parameter Port** (Public): Comprises of active and passive parameter ports of the object. A user of this object modifies the accessible parameters and control of the object through these ports. This hosts routines and procedures to provide access to these ports.

- **Prepare Input** (Protected): This is an optional functional block which takes various inputs and combine it together, so as to make it compatible with inner processes and methods. All internal processes and methods take data and parameters from this block. This block act as a plain buffer if the input data and parameters are compatible with methods specified inside the object. This block hence encapsulates the data from outside world to the inner blocks of an object.

- **Object Controller** (Protected): This functional block is the manager of the object and hosts the intelligence and control routines to control the algorithm, inputs and outputs of an object. It also does the scheduling of processes as shown in the state diagram in the Figure 4.3.
- **Method of the Object** (Private): This functional block hosts the main algorithm of the object. The routines inside of this block operate on given inputs and parameters to produce the desired outputs or functionality.

- **Prepare Output** (Protected): This is again an optional functional block. This block packages results produced by “method” block in user desired format.

- **Output** (Public): This functional block hosts processes and methods to exhibit the results to the user.

This section described an extended class and highlighted the need of some of the extended process. This extended class definition is application independent, and can be easily used by an object designer to construct classes on this template. Most of the software designer uses the standard class definition to make their application based on OOP concepts. Programs made using OOP have been proven to be manageable, reusable, and scalable. The only bottleneck that has plagued the software programs are the software bugs. Software bugs can be intentional, un-intentional, machine specific, and other unknown upsets. There are separate software testing tools to detect software bugs, but they also can’t cover all the bugs.

Most of the software bugs are introduced at design phase, and it’s always desirable if the bug can be identified in design phase [31]. As OOP methodology is now a standard in software design, so effort should be done to extend OOP to handle bugs. OOP’s philosophy is to model software to real-world objects and procedures. Most of the OOP languages implements this philosophy very well but didn’t explicitly mimic some of the core procedures of real world, which if included, would make the design more robust and reliable. One of such procedure is “Quality”. Every product and object in real-world is driven by its quality. The better the quality, the better is the product. Another important process apart from the “Quality” is the “Status”. Status is useful for both the user of the service and the service provider. One real world example where the “Status” information helps is the “Shipping Industry”. Shipping companies now provides point to point tracking of the shipment to a user, which is counted as
one of the measure of quality of service (QoS). Next section will extend the existing software architecture and incorporate these two procedures “Quality” and “Status” in the architecture.

4.4 Quality: New paradigm in OOSDM

Previous section partitioned the processes at different levels. This helped in defining class very close to real-world processes. This section will include two more procedures to the class definition to closely mimic real-world processes in software. Since, the standard definitions in OOP paradigm are very common and widely used, new terminologies are coined here to make a clear distinction between existing terms in OOP and proposed designed extension to OOP. These new terms would be used in the remaining literature to distinguish between the OOP and the proposed systematic extension to the OOP.

- Object-Oriented System Design Model (OOSDM): This term replaces the term “OOP”. OOSDM inherits all the design philosophy of OOP and adds the processes described in Section 4.2.2.

- Engine: This term replaces the “Object” of OOP. Engine has same feature and architecture as an object of OOP, just the name is different to make a clear distinction.

- Engine architecture: This term replaces the “Classes” in OOP. This contains the process lists, blueprint, and levels of OOSDM, like the classes in OOPS.

Adopting these new terms, OOSDM is extended by adding “Quality” and “Status” procedures in the Engine Architecture.

4.4.1 Why Quality?

Quality is one of the necessary procedures followed in real-world processes, but the question arises, why it is so important in software. The importance of quality is apparent from the
definition of quality in software perspective. Some of the standard definitions of quality in software cycle and software architectures related to software quality are [31]:

- **Pressman’s definition of "Software Quality":** Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

- **IEEE-928 Definition:**
  
  - A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.
  
  - A set of activities designed to evaluate the process by which the products are developed or manufactured. Contrast with quality control.

- **CMM [32]:** The Capability Maturity Model (CMM) for Software developed by the Software Engineering Institute (SEI) is a framework that describes the key elements of an effective software process. The CMM describes an evolutionary improvement path for software organizations from an ad hoc, immature process to a mature, disciplined one. The CMM covers practices for planning, engineering, and managing software development and maintenance. When followed, these practices improve the ability of organizations to meet goals for cost, schedule, functionality, and product quality.

According to above definitions, quality can be stated as a process which reduces or removes the error, thereby helps in minimizing software bugs. There are many other standards available that if followed can also save software errors. For example, CMM-Based Software Process Improvement developed by Carnegie Mellon university provide the guidelines, which if followed, can reduce software errors [33]. Software quality management (SQM) applies to all perspectives of software processes, products, and resources. It defines processes, process owners, and requirements for those processes, measurements of the process and its outputs, and feedback channels. Some of the specific SQM processes are defined in standard (IEEE12207.0-96) [50]:

45
• Software Quality Assurance (SQA): This processes provide assurance that the software products and processes in the project life cycle conform to their specified requirements by planning, enacting, and performing a set of activities to provide adequate confidence that quality is being built into the software. This means ensuring that the problem is clearly and adequately stated and that the solution’s requirements are properly defined and expressed [50, 48, 49].

• Verification and Validation process (V&V): Software V&V is a disciplined approach to assessing software products throughout the product life cycle. A V&V effort strives to ensure that quality is built into the software and that the software satisfies user requirements (IEEE1059-93). The V&V process determines whether or not products of a given development or maintenance activity conform to the requirement of that activity, and whether or not the final software product fulfills its intended purpose and meets user requirements. Verification is an attempt to ensure that the product is built correctly, in the sense that the output products of an activity meet the specifications imposed on them in previous activities. Validation is an attempt to ensure that the right product is built, that is, the product fulfills its specific intended purpose. Both the verification process and the validation process begin early in the development or maintenance phase [50, 48, 49].

• Review and Audit process: Review and Audit processes extend out of the reach a designer. Review and Audit helps in removing the bugs by peer review and review by some external agencies. It can be further divided in:
  
  – Management reviews: “The purpose of a management review is to monitor progress, determine the status of plans and schedules, confirm requirements and their system allocation, or evaluate the effectiveness of management approaches used to achieve fitness for purpose” (IEEE1028-97).
  
  – Technical reviews: “The purpose of a technical review is to evaluate a software
product to determine its suitability for its intended use. The objective is to identify discrepancies from approved specifications and standards. The results should provide management with evidence confirming (or not) that the product meets the specifications and adheres to standards, and that changes are controlled” (IEEE1028-97).

– Inspections: “The purpose of an inspection is to detect and identify software product anomalies” (IEEE1028-97).

– Walk-throughs: “The purpose of a walk-through is to evaluate a software product. A walk-through may be conducted for the purpose of educating an audience regarding a software product” (IEEE1028-97).

– Audits: “The purpose of a software audit is to provide an independent evaluation of the conformance of software products and processes to applicable regulations, standards, guidelines, plans, and procedures” (IEEE1028-97).

All of the above quality processes cannot be followed by a designer, but designer can follow some of the quality processes during design phase to minimize the bugs and improve the level of SQA of a software product. We propose to include some of these quality procedures in design phase itself to handle errors that may occur in software. We included VnV and Exception handling in our software architecture based on the types of errors.

### 4.4.2 Errors in Software

Errors in general can be categorized as 'software error', 'software fault', and 'software failure' and types of error can be in general:

- Code error
- Procedure error
- Documentation error
• Software data error

It’s hard to find the cause of an error in software. Some of the most common causes of software errors are [31]:

• Faulty requirements definition

• Client-developer communication failures

• Deliberate deviations from software requirements

• Logical design errors

• Coding errors

• Non-compliance with documentation and coding instructions

• Shortcomings of the testing process

• Procedure errors

• Documentation errors

Cost of these errors is huge. According to a statement released by US Department of Commerce in June 2002 - "Software bugs, or errors, are so prevalent and so detrimental that they cost the U.S. economy an estimated $59.5 billion annually, or about 0.6 percent of the gross domestic product. Although all errors cannot be removed, more than a third of these costs, or an estimated $22.2 billion, could be eliminated by an improved testing infrastructure that enables earlier and more effective identification and removal of software defects. These are the savings associated with finding an increased percentage (but not 100 percent) of errors closer to the development stages in which they are introduced. Currently, over half of all errors are not found until "downstream" in the development process or during post-sale software use.'

This work focuses on how to minimize some of the errors in design phase. Real world processes for quality like “six sigma” have been proved to minimize the defect. “Six Sigma”
is a quality process which seeks to improve the quality of process outputs by identifying and removing the causes of defects (errors) and minimizing variability in manufacturing and business processes [35]. Also some of quality processes defined in Section 4.4.1, we adopted some of the quality process defined under SQM [50, 48, 49] that can be added in design phase.

4.4.3 Quality Process in OOSDM

The types of error in Section 4.4.2 suggest that verification and validation in design phase only can reduce huge percentage of bugs. The OOSDM architecture is extended to add following quality processes.

4.4.3.1 Verification

Software data error is considered to be one of the major types of error. This type of errors is caused mainly due to: “Faulty inputs and control to the software (engine)”. Faulty inputs can be from external world or some other part of the software. In OOP philosophy, an object knows its capabilities and limitations. Similarly in OOSDM, as an engine or systems of engines, knows its functional capabilities and limitations. This knowledge helps the designer of the engine to filter out any inputs or controls, which if accepted without any objection, can put the engine in trouble. Verification is the process of verifying inputs and parameters (both active and passive) before it goes to engine’s main routine. Basic verifications methods could be:

- Checking the inputs for the data range. This avoids overflow and underflow errors.
- Checking data and parameter for compatibility.
- Checking for missing data or incomplete data.
- Checking based on basic application specific sanity checks.

Verification ensures that no wrong information (data) or parameter is fed inside the engine unless forced into it by the designer. Under OOSDM architecture, engine controller manages
the verification procedure. Verification process is protected from the user and in the case of a verification error; it notifies the engine controller about the error and pass the cause of error to it. This process doesn’t resolve the errors found in inputs and parameter; it only reports it to the engine manager (controller). Engine controller then routes the information received by the verification process to the “Exception Handler” process of the software architecture. Role of exception handler is described in Section 4.4.3.3.

4.4.3.2 Validation

Another source of errors in system design is the errors and defects produced at the output of services and methods. This error falls in the category of “deviation from the desired output”. Engine knows the quality of the output desired by the user and based on the verified inputs and parameters. Hence, it can validate the produce output to the desired output and report to the engine controller in case of a mismatch. Basic validation methods could be:

- Checking the outputs for the data range. This avoids overflow and underflow errors.
- Checking logical data value w.r.t threshold kept by designer.
- Checking the output data format mismatch.
- Checking the timing of output to avoid conflict.

Again, validation process does not correct the faulty output, it only reports the fault occurrence to the engine controller. Engine controller again treats it as an exception (of different type) and directs the information to “Exception Handler” to handle it.

4.4.3.3 Exception Handler

In OOSDM philosophy, role of an exception handler is same as the “Quality manager” in a real-world product company. In case of an exception reported by verification, validation or
some internal methods, the exception handler identifies the type of exceptions. Exception types are broadly defined in following categories:

- **Recoverable**: These are the types of exceptions which can be recovered within the engine. A real-world analogy to this type of exception can be: wrong zip code, but correct address and house number for a shipment. Shipping company can recover from this exception as it can solve for a unique destination from the provided information and delivers the shipment.

- **Resolvable**: These are the types of exceptions which can be resolved and problem can be identified, but are not recoverable by the engine controller. This may require external aiding to complete resolve the exception without restarting the process all over again. A real-world analogy to this type of exception can be: wrong address and apartment number, but correct zip code for a shipment. Shipping company cannot recover from this exception as it cannot solve for a unique destination from the provided information. Here the problem is located and resolvable if customer corrects the address.

- **Identifiable but not resolvable**: These are the types of exceptions which cannot be resolved but cause of exception can be identified. A real-world analogy to this type of exception can be: wrong address and apartment number, correct zip code for a shipment, but no information about the sender. Shipping can identify the cause but cannot resolve it due to lack of sender’s information.

- **Unknown**: All other types of error where exception handler cannot locate the cause of exception are categorized here.

In an event of an exception, exception handler categorizes the exception in these categories, makes a report and sends the report back to the engine controller. Engine controller, upon receiving the report changes the state as designed by designer. State change and control path change are usually application specific.
The philosophy behind these three procedures (Verification, Validation, and Exception Handling) is, “I will take only correct inputs and will produce correct only healthy outputs”, the degree of correctness here is directly proportional to how closely it matched the requirements of that engine. Figure 4.5 gives the flow chart of software architecture involving these three processes of OOSDM.

### 4.5 Health Monitoring and Debug Information

Inclusion of verification and validation process in the work flow modified the back bone of the Engine Architecture to sustain and handle errors and exceptions. These sections include the health management and debug information generation to the architecture of OOSDM. Status of each process is one of the important information, that if provided to the engine controller, it can manage engine more reliably. Status information is generated within the engine and its purpose of health monitoring process is to update the information to the engine controller. This philosophy again mimics the example of project management and project planning [32].
4.5.1 Why Health Monitoring?

As in case of quality, absence of health monitoring can also results in software or engine failure. These failures are in general result of a "procedural error". They mainly arises due to faulty behavior of some methods or procedures (infinite loop is one such example) or total failure of some internal methods. This type of error can be catastrophic. As an example, On February 25, 1991, “Patriot Missile System” at Dhahran, Saudi Arabia had been in operation for 100 hours, by which time the system’s internal clock had drifted by one third of a second. This was equivalent to a position error of 600 meters. The radar system had successfully detected the Scud and predicted where to look for it next, but because of the time error, looked in the wrong part of the sky and found no missile. With no missile, the initial detection was assumed to be a spurious track and the missile was removed from the system. No interception was attempted, and the missile impacted on a barracks killing 28 soldiers [31].

Had there been a fault detection system or internal clock health monitoring system, it could have indicated the fault to the user and appropriate action could have been taken. In this type of failure the engine can never recover automatically but it can avoid the wrong output data by the assistance of health monitoring system.

4.5.2 Basic Health Monitoring

Health monitoring system resembles the watch dog timer (WDT) in hardware system design. An easy way to know the status of a sub-process is by message passing. This can be easily explained with the flowchart in Figure 4.6. With assistance of exception handler and engine controller, health monitoring system can assist engine controller to better control the engine. For errors, like the one in example of clock drift in patriot missile system, engine designer can couple some of the critical sub-section or sub-process in engine with validation process. With this coupling and health monitoring procedure, such errors can be detected and reported to the user.
4.5.3 Debug Port

This is the additional output port that is added in OOSDM. This port is intended to produce the basic health information to user and detailed debug information to the designer. At any time user can read this port and can monitors the health of internal process running inside the engine. This is read only port, a user can interpret the health information and can perform appropriate actions if required. Debug port also helps designer and tester of engine to read the details of an exception if it occurs. With both health and debug information, a designer can easy debug the engine.

4.6 Complete Software Architecture

Previous sections extended an OOP object to an OOSDM engine by adding some of new abstract procedures and processes. Following processes are added to OOP class definition to make an OOSDM architecture.

1. Active & Passive Input Port: This is modified version of set functions in a generic class definition in OOP. This clearly distinguishes between types of inputs fed to an engine in
the architecture.

2. Active & Passive Parameter Port: This is again modified version of set functions but it clearly differentiates between an input and a parameter.

3. Prepare Input to Engine: This process is added to maintain the compatibility between input ports and inner processes.

4. Verification: This is a quality assurance process to minimize the errors due to wrong data input.

5. Validation: This is again a quality assurance process to minimize the data and control errors produce by the engine at the output.

6. Health Monitoring: This process acts as watchdog and adds fault tolerance capability in engine.

7. Engine Controller: This acts as process manager and decision making entity in case of some exception.

8. Exception handler: This is the quality manager process which identifies the type of fault in the engine and reports it to the engine controller.

Advantages of modeling an engine using OOSDM are:

- It inherits OOP philosophy, it retains all the advantages of OOP. Thus software and systems built on OOSDM are:
  
  - Easy to maintain and modify existing code
  - Robust and Secure
  - Allows re-usability
  - Easy to extend
They are modular

- It supports quality. Verification and Validation are two quality processes that are added to OOSDM
- It supports error handling.
- It supports health monitoring.
- It clearly divides code in data pathway and control pathway by introducing manager or controller.

Figure 4.7 shows the complete architecture of an engine in OOSDM. Input-output block diagram of an engine is shown in Figure 4.8.
4.7 Chapter Summary

This chapter started with the need of an organized software architecture. The purpose was to make GPS simulator expandable, reusable and easily maintainable. OOP philosophy of software architecture supports the requirements but lacks in providing verification, validation, and health monitoring. Section 4.2, defined the philosophy behind OOP and Section 4.3 extended the process definition in OOP by classifying processes in a mutually exclusive way at each level of abstraction. Section 4.4 highlighted the need of quality in a class defined OOP paradigm. Addition of quality processes like verification and validation extended OOP processes and re-named it to object-oriented system design model (OOSDM). Section 4.5 adds the health monitoring as another quality and fault tolerance process to OOSDM. Finally, the Section 4.6 puts all the pieces together and built a complete architecture of an engine. Engine retains OOP dictated processes like abstraction level, inheritance and polymorphism and adds verification, validation, and health monitoring to it. Next chapter will use this software architecture to model sections and sub-sections of a GPS simulator using OOSDM philosophy.
Chapter 5

GPS Simulator Architecture

The simulator discussed in this work is based on Object Oriented System Design & Modeling (OOSDM) architecture. The designed simulator is partitioned in a modular way to support OOSDM.

As shown in Figure 5.1, a GPS signal simulator can be partitioned based signal generation and propagation in four main modules:

1. Base time generator: This generates the base time for both satellite and receiver section.

2. Satellite section: This is responsible for generation of coded GPS signal at the transmission frequency.

3. Channel: This section models the channel through which the signal is propagated.

4. Receiver Section: This hosts the receiver section responsible for down conversion and demodulation of noisy signal.

This chapter explains the architecture and algorithm of: base time generator, transmission satellite, and channel model of GPS as described in Figure 3.2. The receiver section will not be developed in the work, but will be developed as a part of future expansion of this software. The explanation of each section is further divided in five sections:
• Mathematical model: This explains the mathematics involved and concepts in each engine.

• Engine controller: This lays out the steps followed by each engine to achieve the functionality.

• Exception Handler: This provides the guideline that the engine controller will follow in an event of an exception.

• IO Table: This is a tabular representation of all the inputs and outputs related to an engine.

• Sample Results: This will plot the results achieved by simulation of engine.

Above sections covers all aspect of an engine. In this work, the results are presented at each engine output section.
5.1 GPS Simulator Design: Fundamental Assumption

GPS simulator simulates GPS signals with respect to the position of GPS receiver. Thus, the kinematics (position, velocity, and time) of a user should be known to the simulator to generate the GPS signal with reference to the current user (receiver) position. It's practically impossible for a simulator to produce generic GPS signal that can be used by any receiver to determine its location. This is the fundamental assumption that governs the simulator design. For multiple receivers, synchronized GPS signal generation is possible by calling multiple instances of simulator in synchronized way. By use of OOSDM, this simulator supports multiple receiver location for same set of GPS measurements. Next section starts with satellite data and signal generation at the satellite section.

5.2 Base Time Generator Engine

GPS is a time based system, i.e. the precise ranging depends upon the stable clocks to which all satellites are synchronized. This requires a proper modeling of the clocks used in the simulator. GPS system involves three types of time:

- Receiver time: This is the local time of the receiver that is generated by the receiver clock. This time is not synchronized with GPS satellite time. Receiver clock has bias and drift associated with it and need to be corrected for precise range measurements.

- GPS true time. This is the time without error and is reference to GMT time. This can be referred as the reference time to which every measurements are made.

- Satellite time: This is the time at which satellites generates the data. GPS satellites have high stable atomic clocks and have a small drift and bias associated with it. These corrections are sent by GPS satellites in their navigation messages.
Referring to Section 3.1, all three times are related by Equations 3.1 and 3.2. The simulation uses receiver clock error models and GPS satellite clock error models. The receiver clock error models are double-integrators that are driven by two white-noise terms.

5.2.1 Mathematical Model

GPS true time in Equation 3.1 is modeled in the simulator as a counter which is incremented by one on each tick of the simulator workstation clock. The counter ticks are exact one second (increment by one) after initialization. Simulator workstation here is considered to be the receiver clock of Equation 3.1. It is assumed that the drift of the workstation clock is low. The receiver clock errors are modeled as a 2-dimensional Gaussian discrete-time white-noise process with a mean of zero and a covariance equal to the identity matrix. The random number generator used to generate the discrete-time white-noise process takes the feed from the workstation clock [46].

Advantage of using the workstation clock is that it closely models the noise process given by [38]:

\[
\begin{bmatrix}
\delta t_{R(k+1)} \\
\delta f_{R(k+1)}
\end{bmatrix} = \begin{bmatrix} 1 & \Delta t_k \\ 0 & 1 \end{bmatrix} \begin{bmatrix}
\delta t_{Rk} \\
\delta f_{Rk}
\end{bmatrix} + \begin{bmatrix} \sqrt{S_f \Delta t_k + S_g \Delta t_k^2/3} & \sqrt{S_g \Delta t_k^2/2} \\ 0 & \sqrt{S_g \Delta t_k} \end{bmatrix} w_k, \quad (5.1)
\]

where \( \Delta t_k = t_{k+1} - t_k \) is the interval between the true GPS times given by Equation 3.1 and \( \Delta t_{k+1} \) is the next predicted interval value of Equation 3.1 and where \( w_k \) is a 2-dimensional Gaussian discrete-time white-noise process with a mean of zero and a covariance equal to the identity matrix. Appropriate values for the clock’s phase drift intensity \( S_f \) and frequency drift intensity \( S_g \) for various oscillator types can be found from Table 5.1 for different types of timing standards. Additional receiver clock error model has the same form, though it may involve different drift intensities.
Table 5.1: Typical Power Spectral Density for Different Clock [38]

<table>
<thead>
<tr>
<th>Timing Standard</th>
<th>$h_0$</th>
<th>$h_{-1}$</th>
<th>$h_{-2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Temperature Controlled Oscillator (TCXO)</td>
<td>$2 \times 10^{-19}$</td>
<td>$7 \times 10^{-21}$</td>
<td>$2 \times 10^{-20}$</td>
</tr>
<tr>
<td>Oven Controlled Oscillator (OCXO)</td>
<td>$8 \times 10^{-20}$</td>
<td>$2 \times 10^{-21}$</td>
<td>$4 \times 10^{-23}$</td>
</tr>
<tr>
<td>Rubidium</td>
<td>$2 \times 10^{-20}$</td>
<td>$7 \times 10^{-24}$</td>
<td>$4 \times 10^{-29}$</td>
</tr>
</tbody>
</table>

\[ S_f \sim h_0/2, \ S_g \sim 2\pi^2 h_{-2} \] (5.2)

5.2.2 Engine Controller

Engine controller schedules the algorithms that need to be followed in sequence to generate the base clock. The steps followed by base Time Generator’s engine are:

1. Initialize the Engine: Initialize the white noise Gaussian process.
2. Take passive input: None
3. Take active input: Take work station tick as the base clock with interval update as 1 second.
4. Take passive control: Choose the clock from Table 5.1, Take the value of clock’s phase drift intensity $S_f$ and frequency drift intensity $S_g$.
5. Take active control: None.
6. Verify input: As a quality process, input time range is always checked for their consistency.
   In addition to active inputs, the passive parameters like clock’s phase drift and frequency drift are checked during the initialization phase.
7. Method of Engine: This follows the model given by Equation 5.1 to generate the receiver time.
8. Validate Output: This validates the data consistency generated GPS true time and Rx clock time.

9. Output: This outputs GPS true time and Rx clock time and errors $\delta t_r$ of Equation 3.1.

5.2.3 Exception Handling

In case of any verification and validation anomaly this throws out the detail report. Time is the most critical data for signal generation, an anomaly can be catastrophic. In this engine, engine controller aborts the simulator in case of time anomaly.

5.2.4 Input Output Table

Table 5.2 provides interfaces and processes to this engine.

5.2.5 Sample Results

Outputs of base time generator are true GPS time and receiver clock error. A simulation run of one day has been done to capture the results. From plot in Figure 5.2 we can state the receiver clock has drifted by $0.16406 \mu sec$ (deviation from the true value), this is the value of $\delta t_R$ in Equation 5.1. This is an acceptable value as we care not correcting it. The Gaussian distribution shown in the plot follows additive white Gaussian distribution $w_k$ mentioned in Equation 5.1.
Table 5.2: Interfaces and Processes of Satellite Generator Engine

<table>
<thead>
<tr>
<th>Access level</th>
<th>Process</th>
<th>Data/ Sub methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected</td>
<td>Initialization</td>
<td>Initialize Gaussian process using gaussian random number generator.</td>
</tr>
<tr>
<td>Public</td>
<td>Take Passive Input</td>
<td>None</td>
</tr>
<tr>
<td>Public</td>
<td>Take Active Input</td>
<td>Simulator Workstation local clock</td>
</tr>
<tr>
<td>Public</td>
<td>Take Passive Parameter</td>
<td>None</td>
</tr>
<tr>
<td>Public</td>
<td>Take Active Parameter</td>
<td>Type of Clock standard Value of $S_f$, $S_g$</td>
</tr>
<tr>
<td>Protected</td>
<td>Verify Inputs</td>
<td>Consistency of Workstation local clock</td>
</tr>
<tr>
<td>Protected</td>
<td>Engine Controller</td>
<td>Steps in Section 5.2.2</td>
</tr>
<tr>
<td>Protected</td>
<td>Exception Handler</td>
<td>Steps in Section 5.2.3</td>
</tr>
<tr>
<td>Private</td>
<td>Method of Engine</td>
<td>Generate True GPS time as counter Generate modeled receiver clock from Equation 5.1</td>
</tr>
<tr>
<td>Protected</td>
<td>Validate Output</td>
<td>Check threshold of output clock.</td>
</tr>
<tr>
<td>Public</td>
<td>Output of Engine</td>
<td>• GPS True time $t_k$ • Receiver modeled time $t_r$</td>
</tr>
<tr>
<td>Public</td>
<td>Health Information</td>
<td>Status of the clock drift (if its more user should restart the simulator)</td>
</tr>
</tbody>
</table>
Figure 5.2: Output of Base Time Generator Engine
5.3 Satellite Section

Satellite section of GPS is also known as satellite vehicle (SV). These SV’s broadcasts synchronized ranging signals at GPS L-band frequencies and orbital information as a part of the navigation message. This section of the simulator is responsible for:

- Generation of satellite data & satellite clock
- Generation of satellite code
- Generation of satellite carrier
- Modulation of coded data: Carrier is modulated by the code. This is done by carrier generation engine only.

5.3.1 Generation of Satellite Data and Clock

GPS satellites transmit its orbital information. The orbital information is structured in precise orbital information, also known as “Ephemeris”, and coarse orbital information known as “Almanac”. This Almanac data is not very precise and is considered valid for up to several months. Ephemeris data by comparison is very precise orbital and clock correction for each SV and is necessary for precise positioning. Each SV broadcasts only its own Ephemeris data. The validity of this data is dictated by the particular satellite and may be valid up to 4 to 6 hours. Each set of ephemeris data gives a "fit" indication which tells how long the particular Ephemeris data is valid. The Ephemeris data is broadcast by each SV every 30 seconds so GPS receivers have frequent opportunities to receive and log this essential information. Ephemeris data differs from almanac only in precision. In case of simulation environment, ephemeris can be generated from the almanac data using modeling. Another approach could be taking the known collected log from the National Geodetic Survey (NGS) [36] and simulate the data.

This simulator will use coarse orbital information from GPS almanac file provided by navigation center website [37] to generate ranging signal by computing satellite position and
velocity. Support for using ephemeris information can be implemented in future. Satellite
data generator engine is created using OOSDM. This engine generates SV position, velocity,
and clock information for a given time. Next section describes architecture of satellite data
generator.

5.3.1.1 Mathematical Model

This engine is based on the mathematical model developed in Chapter 3 and mainly focused
on Equation 3.5. The routines for satellite position computation, sagnac, ionosphere, elevation
and azimuth angle computation are taken from GPSTk library and the source for this can be
found at [40].

5.3.1.2 Engine Controller

As described in Chapter 4, engine controller schedules the algorithms that need to be fol-
lowed in sequence to achieve the functionality. The steps required by satellite data generator’s
(SatDGen) engine controllers are:

1. Take passive input: The almanac file is the passive input. This doesn’t change during
run time of the engine.

2. Take active input: Active input to this engine as this keep changing.

   (a) Time to compute satellite data

   (b) Satellite ID

   (c) User Position

3. Take passive control: None

4. Take active control: None
5. Verify input: As a quality process, input time range and satellite id is always checked for their consistency. In addition to active inputs, the passive input is checked during the initialization phase.

6. Method of Engine: This hosts the routines to compute satellite position, velocity and clock parameters. This simulator uses GPSTk routines to compute these parameters which actually are based on routines provided in interface control document (ICD) of GPS [ICD].

7. Validate Output: This validates the data consistency of GPS satellite position, velocity, and other parameters

8. Output: This outputs all the SV parameters to the output port of the engine

5.3.1.3 Exception Handling

In case of any verification and validation anomaly this throws out the detail report. As these are the critical data for signal generation, an anomaly can be catastrophic. In this engine, engine controller notifies the user or other engines about the exception by notifying the health of data as unhealthy.

5.3.1.4 Input Output Table

Table 5.3 provides interfaces and processes to this engine.

5.3.1.5 Sample Results

This section gives the plot of GPS satellite PRN 4, which was visible for approximately 4 hours. Log has been generated and plotted.

Plots for satellite dependent errors as described in mathematical model are shown in Figure 5.3 and Figure 5.4 shows the histogram of elevation and azimuth angle for PRN-4. Errors due
### Table 5.3: Interfaces and Processes of Satellite Generator Engine

<table>
<thead>
<tr>
<th>Access level</th>
<th>Process</th>
<th>Data/ Sub methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td>Take Passive Input</td>
<td>Almanac file</td>
</tr>
<tr>
<td>Public</td>
<td>TakeActive Input</td>
<td>• GPS Time,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Satellite ID,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• User Position</td>
</tr>
<tr>
<td>Public</td>
<td>Take Passive Parameter</td>
<td>None</td>
</tr>
<tr>
<td>Public</td>
<td>Take Active Parameter</td>
<td>None</td>
</tr>
<tr>
<td>Protected</td>
<td>Verify Inputs</td>
<td>• Consistency of Almanac File</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Range of Satellite ID (Max =32)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Validity of GPS time</td>
</tr>
<tr>
<td>Protected</td>
<td>Engine Controller</td>
<td>Steps in Section 5.3.1.2</td>
</tr>
<tr>
<td>Protected</td>
<td>Exception Handler</td>
<td>Steps in Section 5.3.1.3</td>
</tr>
<tr>
<td>Private</td>
<td>Method of Engine</td>
<td>Compute Satellite Position</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute Satellite Velocity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute Satellite Clock Errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute pseudo range</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute satellite related parameters</td>
</tr>
<tr>
<td>Protected</td>
<td>Validate Output</td>
<td>Check Satellite Position &amp; velocity range</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check threshold for other outputs.</td>
</tr>
<tr>
<td>Public</td>
<td>Output of Engine</td>
<td>• Satellite position, velocity &amp; clock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Relativity and Relativity corrections</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Sagnac Correction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Elevation and Azimuth angle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Satellite time</td>
</tr>
<tr>
<td>Public</td>
<td>Health Information</td>
<td>Satellite data health flag</td>
</tr>
</tbody>
</table>
5.3.2 Generation of Satellite Code

GPS is a CDMA system and uses gold codes to spread the signal. GPS uses GOLD-codes (named after a mathematician) because of its good cross correlation capabilities. GPS signals, thus inherits all the advantages of a CDMA system like, multiple access, resistance to jamming, better multipath resolution and others [4]. GOLD-codes are also used to produce ranging signals. The 50 Hz navigation data of a GPS satellite is modulated by three different binary codes: first there is the C/A code (coarse acquisition). This code is a 1023 “chip” long code,
Figure 5.4: Elevation and Azimuth Angle Distributions
Figure 5.5: Outputs of Satellite Data Generator Engine: Clock and Thermal Noise Errors
Figure 5.6: Outputs of Satellite Data Generator Engine: Range variation
being transmitted with a frequency of 1.023 MHz. A “chip” is the same as a “bit”, and is described by the numbers “one” or “zero”. The name “chip” is used instead of “bit” because no information is carried by the signal. By this code, the carrier signals are modulated and the bandwidth of the main frequency band is spread from 2 MHz to 20 MHz (spread spectrum). Thus, the interference liability is reduced. The C/A code is a pseudo random code (PRN) which looks like a random code but is clearly defined for each satellite. It is repeated every 1023 bits or every millisecond. Therefore, each second 1023000 chips are generated. Taking into account the speed of light the length of one chip can be calculated to be approximately 300 m.

The C/A code is the base for all civil GPS receivers. The P code (P = precise) modulates the L1 as well as the L2 carrier frequency and is a very long 10.23 MHz pseudo random code. The code would be 266 days long, but only 7 days are used. For protection against interfering signals transmitted by an possible enemy, the P-code can be transmitted encrypted. During this anti-spoofing (AS) mode the P-code is encrypted in a Y-code. The encrypted code needs a special AS-module for each receiving channel and is only accessible for authorized personnel in possession of a special key.

This simulator generates C/A code. Satellite code generator engine (CodeGen) is created using OOSDM. This engine generates periodic gold codes for different satellites.

5.3.2.1 Mathematical Models

Gold codes are formed by product of two code equal period 1023 bit PN codes $G_1(t)$ and $G_2(t)$ [4]. Thus, this product is also of 1023 bit period and is represented as follows:

$$XG(t) = G_1(t)G_2[t + N_i(10T_c)], \quad (5.3)$$

where $N_i$ determines the phase offset in chips $G_1$ and $G_2$. Each GPS satellites have unique code that is generated by the generator polynomial of two codes given by:
\begin{equation}
G_1 : G_1(X) = 1 + X^3 + X^{10} \tag{5.4}
\end{equation}

\begin{equation}
G_2 : G_2(X) = 1 + X^2 + X^3 + X^6 + X^8 + X^9 + X^{10}.
\end{equation}

GPS code is periodic with period 1ms. Simulator provides simulated signals every one second, in one second 1000 × 1023 bits of code is generated if there is no noise and doppler. This huge data is practically impossible to generate in real time for a workstation. For simulation purpose, only sufficient statistics for the code is generated. Noise in the signal and propagation path delays the code. Due to the Doppler associated with code and the noise in the signal; number of complete code period varies with time. Let \( f_{sat} \) be the true oscillator frequency, which in case of GPS is 10.23 MHz, this oscillator has some bias and drift. So the oscillator frequency satellite \( j \) is:

\begin{equation}
f_{sat} = f_{sat}(1 + corr^j), \tag{5.5}
\end{equation}

where \( corr^j \) is the clock correction term of \( j^{th} \) satellite as stated in Equation 3.2. The code frequency for C/A code is \( f_{sat}/10 \). As code is modulated by carrier, carrier doppler due to the satellite motion also affects the code. Carrier frequency transmitted from the satellite is constant multiple of satellite oscillator frequency and for different carrier it is:

\[
\begin{bmatrix}
 f_{carr-L1} \\
 f_{carr-L2} \\
 f_{carr-L5}
\end{bmatrix} = f_{sat} \begin{bmatrix}
 154 \\
 120 \\
 115
\end{bmatrix}.
\tag{5.6}
\]

Furthermore, carrier of the signal has delays due to the propagation medium like ionosphere, troposphere etc. Thus, received carrier frequency is:

\begin{equation}
f_{rx-carr} = f_{carr}(1 + \delta\phi_{prop}), \tag{5.7}
\end{equation}

where \( \delta\phi_{prop} \) is the accumulated phase delay due to all error sources in channel. Hence, the
received code frequency for C/A code embedded in L1 carrier would be:

\[
f_{rx-code} = \frac{f_{rx-code}-L1}{1540}.
\]  \hspace{1cm} (5.8)

The \(f_{rx-code}\) is definitely different than the original code frequency of C/A code (1.023 MHz). Using the periodic nature of C/A code, at any measurement epoch at receiver, there are \(k_{int} = \lfloor f_{rx-code}/1.023e6 \rfloor\) completed code blocks and \(k_{frac} = f_{rx-code} - k_{int}\) fractional blocks of code. The fractional block (< 1ms) at the epoch contain \(S\) number of completed chips called slew and \(S_p\) amount of fractional chip (< 977.517 ns) as shown in Figure 5.7.

From Figure 5.7, the sufficient amount of information from which all chips of PRN code can
be generated from epoch to epoch are:

- First Completed block-A (1023 bits)
- Last incomplete block-B (< 1023 bits)
- Total number of completed code blocks $k_{int}$
- Residue slew number $S_k$ of last slew in block B ($k < 1023$)
- Residue slew phase $S_p$ at epoch $e_{t+1}$ in block B (duration < 1 chip delay)

CodeGen engine take the satellite number and generate this information every second.

5.3.2.2 Engine controller

The steps required by CodeGen engine controllers are:

1. Take passive input: The generator polynomial is the passive input.

2. Take active input: Active input to this engine is the satellite number as the taps of the
gold code changes with satellite number

3. Take passive control: None

4. Take active control: None

5. Verify input: As a quality process, satellite number is verified here, as engine cannot
generate code above the satellite number 32.

6. Method of Engine: This hosts the routines to generate the sufficient information of the
code as described in Section 5.3.2.1

7. Validate Output: This validates the data consistency of code generated by checking

- Residue slew number is always < 1023 for C/A code
- Residue slew phase duration is always < 977.517 ns

8. Output: This outputs the code generated.

5.3.2.3 Exception handling

In case of any verification and validation anomaly this produces a detail report about the exception due to data errors. As code is important output, an anomaly is not desired. In this engine, engine controller notifies the user about the error and wait for user to take action.

5.3.2.4 Input Output Table

Table 5.4 provides interfaces and processes to this engine.

5.3.3 Generation of Satellite Carrier

GPS navigation information is modulated on the carrier using BPSK signaling. The data rate of GPS navigation information is 50Hz. To transport these data signals, a suitable carrier frequency is required. The choice of the carrier frequency is submitted to the following requirements [39]:

- Frequencies should be chosen below 2 GHz, as frequencies above 2 GHz would require large beam antenna for the signal reception.

- Ionospheric delays are enormous for frequency rages below 100 MHz and above 10 GHz.

- GPS signal propagates through different medium as shown in Figure 3.1. For lower frequencies, the speed of propagation of electromagnetic waves varies more in those medium.

- The PRN-codes require a high bandwidth for the code modulation on the carrier frequency. Therefore a range of high frequencies with the possibility of a high bandwidth has to be chosen.
Table 5.4: Interfaces and Processes of Satellite Generator Engine

<table>
<thead>
<tr>
<th>Access level</th>
<th>Process</th>
<th>Data/ Sub methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected</td>
<td>Initialization</td>
<td>• Reset residue slew and slew phase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Generate the base code of the satellite</td>
</tr>
<tr>
<td>Public</td>
<td>Take Passive Input</td>
<td>Generator Polynomial: Gold code (default)</td>
</tr>
<tr>
<td>Public</td>
<td>TakeActive Input</td>
<td>Satellite number</td>
</tr>
<tr>
<td>Public</td>
<td>Take Passive Parameter</td>
<td>None</td>
</tr>
<tr>
<td>Public</td>
<td>Take Active Parameter</td>
<td>None</td>
</tr>
<tr>
<td>Protected</td>
<td>Verify Inputs</td>
<td>Range of Satellite ID (Max =32)</td>
</tr>
<tr>
<td>Protected</td>
<td>Engine Controller</td>
<td>Steps in Section 5.3.2.2</td>
</tr>
<tr>
<td>Protected</td>
<td>Exception Handler</td>
<td>Steps in Section 5.3.2.3</td>
</tr>
<tr>
<td>Private</td>
<td>Method of Engine</td>
<td>Procedure as described in Section 5.3.2.1</td>
</tr>
<tr>
<td>Protected</td>
<td>Validate Output</td>
<td>Check residue slew number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check residue slew phase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Section 5.3.2.2</td>
</tr>
<tr>
<td>Public</td>
<td>Output of Engine</td>
<td>• First Completed block-A (1023 bits)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Last incomplete block-B ( &lt; 1023 bits)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Total number of completed code blocks $k_{int}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Residue slew number $S_k$ in block B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Residue slew phase $S_p$ at epoch $e_{i+1}$ in B</td>
</tr>
<tr>
<td>Public</td>
<td>Health Information</td>
<td>Code clock health flag</td>
</tr>
</tbody>
</table>


The chosen frequency should be in a range where the signal propagation is not influenced by weather phenomena like, rain, snow or clouds.

Based on these considerations, the choice of L-band proved to be advantageous. Each GPS satellite transmits two carrier signals in the microwave range, designated as L1 and L2 (frequencies located in the L-Band between 1000 and 2000 MHz) and L5 the next generation frequency for better accuracy is being planned. Civilian GPS receivers use the L1 frequency with 1575.42 MHz (wavelength 19.05 cm). The L1 frequency carries the navigation data as well as the SPS code (standard positioning code). The L2 frequency (1227.60 MHz, wavelength 24.45 cm) only carries the P code and is only used by receivers which are designed for PPS (precision positioning code). With increasing demand in accuracy, L2C (C-Civilian) has been established for civilian user. Now civilian user can have dual frequency receiver which can help in mitigating the effect of ionosphere.

In the simulator the carrier is generated by satellite carrier generator (CarrGen) engine. This uses standard sinusoidal signal to generate both in-phase (I) and quadrature phase (Q) signal.

5.3.3.1 Mathematical Model

Sinusoidal waves are inherently periodic in nature with period $2\pi$. In this simulator represents sinusoidal wave with six sample points as shown in Figure 5.8. Again using periodic nature of this wave and the literature discussed in Section 5.3.2.1 we can minimize the data generation for high carrier frequency samples. From Equation 5.7 its obvious that received carrier frequency $f_{rx-carr}(t)$ is much different and time varying.

Using the periodic nature of sampled carrier wave, at any measurement epoch at receiver, there are $C_{int} = \lfloor f_{rx-carr}/L1Frequency \rfloor$ completed carrier cycles and $C_{frac} = f_{rx-carr}(t) - C_{int}$ fractional cycles of carrier. The fractional cycles at the measurement epoch will have Z number of completed phase point (out of sampled carrier points) called completes phase and
\( Z_p \) amount of fractional phase (between the sample points) as shown in Figure 5.8.

From Figure 5.8, the sufficient amount of information from which all sample points of carrier wave can be generated at a measurement epoch are:

- First Complete cycle-A
- Last incomplete cycle-B
- Total number of complete cycle = \( C_{int} \)
- Residue carrier phase index $Z_k$ of the last completed phase point in B
- Residue fractional carrier phase $Z_{pk}$ at epoch $e_{t+1}$ in cycle B

CodeGen engine take the satellite number and generate this information every second.

### 5.3.3.2 Engine controller

The steps required by CarrGen engine controller are:

1. Take passive input: Generate the sampled sine and cosine wave (six points sampling)
2. Take active input: Carrier frequency, either GPS L-band frequencies or receiver IF frequency
3. Take passive control: None
4. Take active control: None
5. Verify input: As a quality process, doppler and input frequency are verified.
6. Method of Engine: This hosts the routines to generate the sufficient information of the carrier as described in Section 5.3.3.1
7. Validate Output: This validates the data consistency of carrier generated by checking
   - Residue carrier phase index $Z_k < 6$
   - Residue fractional carrier phase < 1
8. Output: This outputs the sampled carrier.

### 5.3.3.3 Exception handling

In case of any verification and validation anomaly, this throws out the detail report of failure. As carrier wave is important output, an anomaly is not desired. In this engine, engine controller notifies the user about the error and wait for user to take action.
5.3.3.4 Input Output Table

Table 5.5 provides interfaces and processes to this engine.

Table 5.5: Interfaces and Processes of Carrier Generator Engine

<table>
<thead>
<tr>
<th>Access level</th>
<th>Process</th>
<th>Data/ Sub methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protected</td>
<td>Initialization</td>
<td>• Reset residue carrier phase and fractional phase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Generate the base sampled sinusoid</td>
</tr>
<tr>
<td>Public</td>
<td>Take Passive Input</td>
<td>No of samples in 1 cycle of carrier</td>
</tr>
<tr>
<td>Public</td>
<td>Take Active Input</td>
<td>Carrier frequency</td>
</tr>
<tr>
<td>Public</td>
<td>Take Passive Parameter</td>
<td>None</td>
</tr>
<tr>
<td>Public</td>
<td>Take Active Parameter</td>
<td>None</td>
</tr>
<tr>
<td>Protected</td>
<td>Verify Inputs</td>
<td>Doppler and carrier frequency</td>
</tr>
<tr>
<td>Protected</td>
<td>Engine Controller</td>
<td>Steps in Section 5.3.3.2</td>
</tr>
<tr>
<td>Protected</td>
<td>Exception Handler</td>
<td>Steps in Section 5.3.3.3</td>
</tr>
<tr>
<td>Private</td>
<td>Method of Engine</td>
<td>Procedure as described in Section 5.3.3.1</td>
</tr>
<tr>
<td>Protected</td>
<td>Validate Output</td>
<td>Check residue carrier cycle phase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check residue carrier cycle fractional phase</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See Section 5.3.3.2</td>
</tr>
<tr>
<td>Public</td>
<td>Output of Engine</td>
<td>• First Complete cycle-A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Last incomplete cycle-B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Total number of complete cycle = $C_{int}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Residue carrier phase index $Z_k$ of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>last completed phase point in B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Residue fractional carrier phase $Z_{pk}$ at</td>
</tr>
<tr>
<td></td>
<td></td>
<td>epoch $t_{e+1}$ in cycle B</td>
</tr>
<tr>
<td>Public</td>
<td>Health Information</td>
<td>Carrier clock health flag</td>
</tr>
</tbody>
</table>

In this chapter we modeled satellite section of a GPS satellite. We architected each subsection of satellite using engine concept based on OOSDM that we built in Chapter 4. The simulator designed in this work involves many other small modules. We only highlighted the
Figure 5.9: GPS Engine Blocks of Simulator

main functional modules of satellite section in this chapter.

We used GPSTk library to model the channel section. Again each sub block in channel section as shown in Figure 5.9 is modeled using engine concept. We only modeled receiver clock in receiver section of simulator. In the next chapter, we will hypothetically show how our simulator based on OOSDM architecture can be useful in generating GPS measurements for use in attitude determination.
Chapter 6

Case Study: Attitude Determination

GPS signals are used widely for attitude determination of an object. Attitude of a free body is described by its orientation along its center of mass. For a vehicle above the earth surface, GPS/GNSS satellites are so distant that if we keep two GPS antenna in a close proximity then an incident wavefront of electromagnetic signal from GPS satellites can be considered as planer. This is the core assumption that makes GPS signals suitable for determining the attitude of a body above the earth surface or on water.

6.1 Definitions

Motion of a body suspended in space is specified by its position, velocity, attitude, and attitude motion. Knowledge of these quantities can enable a precise control of the body. This section defines each of these parameters and builds upon terminology that would be useful in applying GPS for determination of all four quantities[27, 3].

- Position and Velocity: Translation motion of center of mass of the body in space describes its position and velocity. Position and velocity is defined w.r.t a coordinate frame. For example, a GPS receiver gives position and velocity of a body on which its mounted in earth center earth fixed frame (ECEF).
• Attitude and Attitude Motion: These two quantities describe the rotational motion of a freely suspended body about its center of mass. In a three dimensional right handed axes coordinate frame with its origin at the center of the mass, a body can have three degree of freedom with respect to the body frame. These are defined to be:
  
  – Roll: rotation along the x-axis
  – Pitch: rotation along the y-axis
  – Yaw: rotation along the z-axis

• Attitude determination: It is a process of determining these three attitude angles (roll, heading, and pitch) for a freely suspended body relative to an inertial reference or some other object of interest. For example, a spacecraft the attitude is determined w.r.t to its earth’s observatory.

• Attitude prediction: This is a process of forecasting the future orientation of a body by using the dynamic model to extrapolate the determined attitude history.

• Attitude control: It is a process of controlling and assigning the attitude of a body. This may involve rotating a body along its axes or in any other direction along its center of mass.

To understand the process of attitude determination of a freely suspended body requires correctly defining the body frame and local reference frame. Next section briefly describes some of the common reference frames which are used in the literature.

6.2 Coordinate Frames, Kinematics and Earth

Coordinate frames defines position and orientation of an object. Its sometimes also referred as reference frame. For example, a GPS satellite position is given with reference to Earth Centered Earth Fixed (ECEF) frame. For an object suspended in free space we deal with two
frame problem where object center of mass is w.r.t to one frame and then the object orientation is w.r.t its body frame or some other frame. In two frame problem, position of a point in any of one frame can be defined in relative to other frame.

Attitude determination navigation problems thus involve at least two co-ordinate frames, an object frame and a reference plane. Object frame describes the object whose position is desired and reference frame describes a known object such as earth. These two frame may have relative position, velocity or acceleration. Some of the common frames are

- ECI : Earth Centered Inertial frame
- ECEF: Earth Centered Earth Fixed frame
- Local Navigation frame
- Body Frame

6.3 Attitude Determination Process

GPS satellites are so distant relative to the antenna separation that arriving wavefront can be considered as planer. A signal travelling at a speed of light arrives first on the nearer antenna. By measuring the difference in carrier phase between the antennas, a receiver can determine the relative range between the pair of antenna w.r.t a known coordinate frame. So in attitude determination we are typically concerned with describing an object in two separate frame: local horizontal (inertial) coordinate frame and vehicle body coordinate frame as shown in Fig. 6.1.

Given a vector \( \mathbf{r} = \begin{bmatrix} x & y & z \end{bmatrix} \) expressed in local horizontal coordinate frame can also be expressed in local body frame through a coordinate transformation:

\[
\mathbf{r}' = A\mathbf{r} \tag{6.1}
\]

Where matrix \( A \) is known as attitude matrix or direction cosine matrix. Normally attitude
matrix is an assembly of dot products of orthogonal coordinate frame unit vectors, hence $A$ is given by:

$$
A = \begin{bmatrix}
x'x & x'y & x'z \\
y'x & y'y & y'z \\
z'z & z'y & z'z 
\end{bmatrix} 
$$

If we place orthonormal constraint on the transformational frames i.e $(A^TA = I)$ we can reduce degree of freedom from nine to three. Most commonly accepted convention for defining the coordinate frames and rotation angles is shown in Figure 6.2. Rotations are defined in a specific Euler sequence about the coordinate axes. To construct a local horizontal frame in right handed set of axes we can define $x$ axis pointing due north, the $y$ axis pointing due east, and the $z$ axis points directly downward along the local vertical. Body reference plane is unique to the body so the $x'$ axis (roll) points out in east with some rotation as shown in Figure 6.2. Similarly $y'$ (pitch) and $z'$ (heading) axes are shown to make the frame a right handed coordinate axis set. When the heading, pitch and roll angles are all zero, the body frame is aligned to local horizontal frame and attitude matrix $A$ is a zero matrix.
These three angles (heading, pitch, and roll) specify the vehicle attitude. When there is some rotation of body along its center of mass, then that rotation can be described by these three angles. Starting from the reference attitude (where the body and local horizontal frames are aligned), the body frame is rotated (always in positive, right-handed sense) about the local vertical down axis by the heading angle $\psi$. Then, the body frame is rotated about the new pitch axis by the pitch angle $\theta$. Finally, the body frame is rotated about the roll $x'$ axis by angle $\phi$. These transformations and transrotations results in the attitude matrix $A$ which is given by [3]:

$$A = \begin{bmatrix}
cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\
-\cos \phi \sin \psi + \sin \phi \sin \theta \cos \psi & \cos \phi \cos \psi + \sin \phi \sin \theta \sin \psi & \sin \phi \cos \theta \\
\sin \phi \sin \psi + \cos \phi \sin \theta \cos \psi & -\sin \phi \cos \psi + \cos \phi \sin \theta \sin \psi & \cos \phi \cos \theta 
\end{bmatrix}$$  (6.3)
6.4 Attitude Determination Solution Processing

This section discusses how a differential phase measurement measured from two different antenna can be converted into attitude solution. One such assembly is described in Figure 6.3, where the carrier phase difference is measured by taking the difference between the GPS wavefront phase for each antenna. Due to the separation between the antennas the wavefront from the GPS satellites falls on them at different times. This delay can be converted into carrier cycles and phase. Since electromagnetic waves are normally periodic sine or cosine wave we can map the delay in the integer number of wavelengths and some fractional wavelength (phase) as shown in Figure 6.3.

Assuming the integer number of the carrier cycles (wavelengths) are known, then the differential range measurement between two antenna as shown in Figure 6.3 is given by:

\[ \Delta r = \Delta \varphi + k \]  

(6.4)
where \( k \) is number of integer (complete) cycles and \( \Delta \varphi \) is the differential phase. An optimal attitude solution for a given set of range measurement \( \Delta r_{ij} \) taken at a single epoch for baseline \( i \) and satellite \( j \) is obtained by minimizing the cost function [3]:

\[
J(A) = \sum_{i=1}^{m} \sum_{j=1}^{n} (\Delta r_{ij} - b_i^T A \hat{s}_j)^2
\]  \hspace{1cm} (6.5)

for \( m \) baseline and \( n \) satellites, where \( b \) (3 \times 1) is the baseline vector defined in the body frame, \( \hat{s} \) (3 \times 1) is the line of sight vector to the GPS satellite given in the local horizontal frame, and \( A \) (3 \times 3), the variable used in minimization of cost function is the right handed, orthonormal attitude transformation matrix from local horizontal frame to the body frame.

Given a trial attitude matrix \( A_0 \), a better estimate may be obtained by linearizing this cost function about the trial solution for the correction matrix \( \delta A \). This iterative process continues and after \( p^{th} \) iteration we get new and better trial matrix for iteration \( p + 1 \), so that \( A_{p+1} = \delta A_p A_p \). A simple correction matrix can be constructed of small angle rotations [3], so that \( \delta A(\delta \theta) \approx I + \Theta^\times \), where \( I \) (3 \times 3) is the identity matrix, and \( \delta \theta \) (3 \times 1) is a vector of small-angle rotations about the three body frame axes:

\[
\delta \theta = \begin{bmatrix} \delta \theta_x' \\ \delta \theta_y' \\ \delta \theta_z' \end{bmatrix}, \quad \Theta^\times = \begin{bmatrix} 0 & \delta \theta_x' & \delta \theta_y' \\ -\delta \theta_x' & 0 & -\delta \theta_z' \\ \delta \theta_y' & \delta \theta_z' & 0 \end{bmatrix}.
\]  \hspace{1cm} (6.6)

The \( \Theta^\times (3 \times 3) \) in the above equation is the skew-symmetric matrix associated with vector \( \delta \theta \) so that \( \Theta^\times b = \delta \theta \times b \). Using above relations, the Equation 6.5. becomes:

\[
J(\delta \theta)|_{A_0} \approx \sum_{i=1}^{m} \sum_{j=1}^{n} (\Delta r_{ij} - b_i^T \Theta^\times A_0 \hat{s}_j)^2.
\]  \hspace{1cm} (6.7)

Using the properties of skew-symmetric matrix \( \Theta^\times \) and relation between \( \Theta^\times \) and \( \delta \theta \), the
above equation can be written as:

\[ J(\delta \theta) |_{A_0} \approx \sum_{i=1}^{m} \sum_{j=1}^{n} (\Delta r_{ij} - [(A_0 \hat{s}_j) \times b] \cdot \delta \theta)^2, \quad (6.8) \]

or in a matrix format the linearized cost function can be written as follows:

\[ J(\delta \theta) |_{A_0} \approx \|H\delta \theta - \delta r\|_2^2. \quad (6.9) \]

In the equation 6.9 \( \delta r \) is the vector formed by stacking all measurements, and \( H \) is the observation matrix formed by stacking measurement geometry for each separate measurement. Observation matrix \( H \) for the cost function can be written as:

\[ H = \begin{bmatrix}
\vdots \\
\hat{s}_j^T A_0^T B_i^x \\
\vdots 
\end{bmatrix} \quad (6.10) \]

From the Equation 6.8 we find that the angle sensitivity to a measurement from a given baseline and GPS satellite is simply the cross-product of the line-of-sight vector with the baseline vector. Thus, the estimate for \( A \) is then refined iteratively until the process converges to the numerical precision of the converging computer.

### 6.5 Performance

This section examines how GPS errors described in Chapter 3, contributes to the overall performance of attitude determination and quantifies the most significant sources of error in attitude determination. The major task in the attitude determination problem is finding the correct number of integer cycle in differential range measurement, which was assumed to be
known in previous section. Number of method exists for integer cycle ambiguity problem for planer and non-coplanar antenna arrangement [28]. Examining the cost function from Equation 6.9 we observe that it depends on the range difference and baseline length. Since for short base line the common errors for two antenna can be removed some of the errors that contribute more are:

- Multipath
- Structural Distortion
- Troposphere
- Signal-to-Noise Ratio (SNR)
- Receiver-Specific Errors

Table 6.1 specifies $1\sigma$ errors due to the major contributing sources [3]. From the table we see that the major source of error is the multipath error which can be handled to an extent by using narrow correlator [9]. Recent advancements and modernization of GPS helps in tackling and circumventing this error by proper code design, for example new L5 signals have a longer code length than existing L1 signals. L1 chip length is 300 meters hence can resolve signal path which are delayed by 300 meters or more, whereas the new L5 signals have chipwidth of 30 meters hence it can resolve the reflection delayed by 30 meters or more [29]. Taking in account the contribution of major error sources as listed in Table 6.1 the root of sum of squares (rss)
error leads to a rule of thumb for attitude determination angular accuracy (in radians) for a baseline length of $L$ (in cm) and is given by [3]:

$$\sigma_\theta (\text{in radians}) \approx \frac{0.5 \text{ cm}}{L \ (\text{in cm})} \quad (6.11)$$

**Short Baseline ( $L < 100 cm$ )**

There is an increasing interest to use GPS for attitude determination of small satellites, where baseline is limited due to its small size. Longer baseline helps in better attitude angular accuracy but for shorter baseline following errors gets eliminated:

- Troposphere error
- Orbital Clock
- Receiver Clock

Multipath error and receiver noise error gets amplified in differencing process. If we use the code phase measurements, the $1\sigma$ errors for receiver noise for L1 frequency receiver can go up-to 3~4 meters and multipath error can go up-to 10~15 meters [27]. As described in Equation 6.11, this accuracy is very poor hence, the use of code phase receiver is not suited for attitude determination. Alternatively if we use carrier phase measurement, the range error is less as compared to code-phase, this listed in Table 6.1. For a baseline length of 30 cm we get an angular accuracy of $3^\circ$, which also is not an acceptable solution as the desired accuracy required should be less than $1^\circ$. By taking in account the dimension of small satellites, the local multipath is less as compared to bigger satellites or some near earth objects. The $1\sigma$ range error due to multipath for the small satellite becomes less than a millimeter. This gives us an attitude angular accuracy of $0.6^\circ$ which is an acceptable solution.
6.6 Using GPS-Gyan for Simulating Performance

Performance evaluation and antenna configuration for different baseline is one of the challenging task that a system engineer faces. To do a performance evaluation for multiple antenna based attitude determination system for a far earth body simulators are used. Multiple antenna requires multiple signal feeds from simulators which are limited in a standard hardware simulator. The proposed simulator (GPS-Gyan) can be extended to get measurement for different antenna placements. A feasible solution and architecture to get different measurements for different antennas using GPS-Gyan is shown in Figure 6.4. Object oriented nature of GPS-Gyan simulator allows multiple instantiation of each module for different measurement generation at same epoch, additionally, each error can be independently modeled as shown in the Figure 6.4. In current version of simulator user have run the code multiple times to get results. A complete feature to support the multiple antenna will be integrated in next version of this open source software.

This chapter highlighted attitude determination which is one of the widely used application of GPS and envisaged how GPS-Gyan simulator can be useful in studying and doing performance evaluation for accuracy in GPS/GNSS based attitude determination algorithms.
Chapter 7

Limitations and Conclusion

The work presented in this thesis is to demonstrate how a well-designed architecture can be useful to model complex system like GPS simulator. We divided work in two parts:

- Design of a robust software architecture
- Use of designed architecture to architect a GPS signal simulator

With the given time frame, the main focus was to develop the simulator software architecture. Also a full working model of the satellite section and channel has been demonstrated. Present state of the software simulator poses some limitation, which becomes the motivation for future work.

7.1 Limitations

Some of the current limitations of the designed simulator are:

- Lack of using Ephemeris data: Ephemeris data is the more precise data sent by GPS satellites but valid for only 6 hours. This simulator uses almanac data to model the satellite section and navigation information. Almanac data is valid for months but is less precise as compared to ephemeris data. Almanac data processing is done using
GPSTk libraries. This library also supports ephemeris data processing which can be easy incorporated in GPS-Gyan simulator.

- Troposphere modeling: Troposphere modeling has not been done for this simulator based on the requirements. GPSTk again supports troposphere modeling, which can be added in the channel section of simulator.

- Receiver Section: This module is intended for modeling the delay caused by receiver RF section as shown in Figure 5.9. In the Chapter 3 the importance of line bias has been highlighted. This can be included in the design with proper knowledge receiver RF section. This was out of scope of this work.

- Raw signal out: This simulator produces the sufficient statistics to generate the sampled intermediate frequency (IF). Another program (post processing) is required to generate the sampled IF bits. This is done offline due to limitation posed by the speed of workstation to generate and store huge amount of generated.

- GUI: An easy to use user interface is desired by the community. GPS-Gyan currently supports only command line execution. Our simulator architecture supports GUI, an attempt is being made to release the simulator with GUI and some of it is presented in future work chapter of this thesis.

- Benchmarking: Quality of this simulator has not been tested. Data consistency has been verified but data accuracy needs to be benchmarked. This is a long term process and was out of scope of this work.

### 7.2 Conclusion

The work presented in this thesis focuses on the need of a low cost GPS simulator. By the analysis of related work, we propose a software based GPS signals and measurements
simulator that has a flexible and expandable architecture to support the rapid growth in GNSS signals and systems. The intellectual merit of this work is the design of object oriented system design modeling architecture (OOSDM). In this simulator we used the processes and guidelines designed in OOSDM architecture to model the satellite and channel section. Use of OOSDM guidelines ensures the data reliability and quality. By increasing the reliability of data and flexibility in the software architecture, our simulator removes major bottlenecks of existing GPS software signal simulators which are not flexible to support future signal. With a generic software architecture and open source nature of the simulator, GPS-Gyan will encourage the GPS research community participation in the future design and development of this simulator and improve upon the current limitations.
Chapter 8

Future Work

In GPS-Gyan simulator we currently have modeled GPS satellite section and the channel, receiver section modeling and real time signal generation is some of the future work that can be added to the simulator. Chapter 7 highlights various limitations of the current simulator and discusses how they can be removed. This chapter list out some features which can be added to the current simulator. Some of the envisaged future steps to this work are:

- Easy to use Graphical user interface (GUI)
- Creation of software development kit (SDK)
- Feasibility to generate real analog signal.

These future works, if done, would help community to readily use the simulator for their application. This chapter will explain the importance of each of these mentioned future works and propose an approach to move ahead.

8.1 Easy to use Graphical User Interface

GUI of software is the marketing part of the software. User of a software desires an easy to use GUI. With growth of object oriented languages, there are many easy to use open-source
GUI frameworks available to the user to program. Qt by Nokia is an open-source framework which fits the philosophy of OOSDM described in this work [45]. Qt is based on signal and slots, which is simple terms resembles input and output of an engine of OOSDM. Advantages of Qt are:

- Object oriented
- Signal and slot mechanism
- Open-source for non-commercial use
- Properties of each object is independent

The signals and slots makes the different Qt components as reusable. They provide a mechanism through which it is possible to expose interfaces that can be freely interconnected. For example a menu item, push button, toolbar button and any other item can expose signal corresponding to "activated", "clicked" or any other appropriate event. By connecting such a signal to a slot of any other item, the event automatically calls the slots. A signal is just like the output an engine generates with help of engine controller. This signal can be directly used by user end application or fed to another engines. Output of an engine can be connected to many engines as shown in Figure 8.1.

Signal mechanism ensure that signal reaches all the engine to which it’s connected to. From outside world (another engine or an user), an engine receives the signals on its slot. Slot is same as the input port of an engine. The key advantage of the signals and slots is that the caller does not have to know anything about the receiver and vice versa. This makes it possible to integrate many components easily without the component’s designer having actually thought about the used configuration. This is truly loose coupling. Thus OOSDM can be easily adapted to Qt and a good GUI based GPS simulator can be designed. A sample GUI, which is under development is shown in Figure 8.2. In the front window of GUI, “Simulator Session” navigation panel show all engines in tree format. Each branches of a tree represents an engine based on OOSDM. For
Figure 8.1: Signal and Slot Analogy in an OOSDM Engine
example in Figure 8.2, “Channel Model” tree in GUI contains “Space Channel”, “Ionosphere”, “Troposphere” and others as the branches of “Channel Model” and each branch is modeled as an independent and separate engine.

### 8.2 Creation of Software Development Kit (SDK)

One of the objectives of the simulator is its expandability. With SDK, it will help community to modify the code and add plug-ins. Current software architecture can be extended to support SDK. The main step needed to perform this is the library creation and documentation of that library. This is a time consuming task and can be done gradually using open-source base.
8.3 Extend to Generate Real Analog Signal

Real analog signal generation is the need of GPS measurements users’ community. It would be more useful if this software simulator can be extended to hardware simulator to generate real analog signal at GPS transmit frequencies.

Approach

Careful analysis of a hardware simulator is done in Figure 2.1. A hardware simulator requires a real fast processor and an RF generator to generate measurements of GPS signals. Thus, the main cost of existing 12 channel GPS signal hardware simulators lies in using dedicated processors and several RF generators to generate the signals.

Today’s computers rely on powerful graphics processing units (GPUs) to create the spectacular graphics in video games. In fact, these GPUs are now more powerful than the traditional central processing units (CPUs) - or brains of the computer. As a result, computer developers are trying to tap into the power of these GPUs. Taking advantage of a GPU’s processing ability is a big deal, because of the amount of computing power a GPU contains. The CPU from an average computer has about 10 gigaflops of computing power - or 10 billion operations per second. That sounds like a lot until you consider that the GPU from an average modern computer has 1 teraflop of computing power - which is 1 trillion operations per second [43].

But using a GPU for general computing functions isn’t easy. The actual architecture of the GPU itself is designed to process graphics, not other applications. Most used GPU’s are from Nvidia and ATI [25]. The CUDA environment is NVIDIA’s parallel computing architecture that enables dramatic increases in computing performance by harnessing the power of the GPU (graphics processing unit). Software developers, scientists and researchers are finding broad-ranging uses for CUDA, including image and video processing, computational biology and chemistry, fluid dynamics simulation, Computed tomography (CT) image reconstruction, seismic analysis, ray tracing, and much more. CUDA helps boost performance for signal pro-
processing algorithms, e.g. FFT, matrix multiplication (convolution) and other matrix operations [42]. The CUDA environment has been successfully used to implement multichannel GPS receivers by students at National Institute of Information and Communications Technology, Japan [52]. A proposed architecture using CUDA is demonstrated in Fig 8.3.

The work of generating an analog signal with help of this simulator will only require the additional cost of RF vector or signal generator. National instruments “RF and Microwave Signal Generators with Modulation Capability” can generate the modeled signal with BPSK modulation and comes under $3500. The complete cost of hardware will be less than $5000 [44]. This cost is affordable by researchers and scientist to use hardware simulator for more accurate measurements.

GPS-Gyan simulator has been designed in parallel engine and thus supports parallelism. Additionally the communication system modeling mainly involves convolution and Fourier tran-
forms (FFT), hence is suited for GPU. Taking advantage of both the architecture of simulator using OOSDM and the mathematical model to support GPU, a real time simulator can be designed as shown in Figure 8.3.

In this Chapter we proposed some of the envisaged future work and extensions that can be applied to our designed simulator. Any extension to this work will be uploaded to the open-source repository of GPS-Gyan at sourceforge (http://sourceforge.net/projects/gpsgyan/).
Bibliography


107


[24] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides "Design Patterns", Addison-Wesley, 1995


[27] Adams, J.C., "Robust GPS Attitude Determination For Spacecraft", PhD Dissertation, Standfort University, December 1999


[44] RF and Microwave Signal generators, 

[45] Qt., http://qt.nokia.com/


