UPPSALA
UNIVERSITET

IT 12 065
Examensarbete 30 hp
November 2012

Synchronization of RBS in a network using a configurable GPS Receiver Emulator

Ashar Waseem

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Synchronization of RBS in a network using a configurable GPS Receiver Emulator

Ashar Waseem

Radio Base Stations (RBS) are synchronized in a network with the help of signals received from GPS Receivers. These signals include a pulse per second signal (1PPS) along with timing and satellite positioning information sent in the form of NMEA format. The purpose of this thesis is to develop an emulator that can generate these signals and can, in turn, synchronize the RBS for testing purposes in a lab environment. Development of such a device reduces the cost of setting up a Live GPS Receiver and accompanying antennas in a lab, along with the removal of long cables that are to be connected to the RBS. Another advantage of such a device will be the ability of controlling the simulation environment conditions through the configuration of NMEA sentences parameters. Moreover, the emulator can be run to test the RBS for as long as possible with precise control on when to start or stop the test.

Handledare: Torbjörn Hansson, Kjell Fyrvall
Ämnesgranskare: Leif Gustafsson
Examinator: Philipp Rümmer
IT 12 065
Tryckt av: Reprocentralen ITC

Acknowledgement

First of all, I would like to thank my thesis supervisor, Kjell Fyrval, for all the technical assistance and guidance that he provided for the successful implementation of the project. Further, I would like to thank my section manager, Torbjörn Hansson, who arranged for all the hardware and made sure that everything was available when it was required.

I am grateful to my thesis reviewer, Leif Gustafsson, for showing the different approaches that could be taken for doing this thesis and giving the comments about the report which were really helpful in improving this thesis and bringing it into the current form. He has been a great influence during my Master studies at Uppsala University and I have learnt a lot from him.

I would also like to thank my project partner, Elham Khorami, who always worked hard and diligently for the project.

At the end, I would like to express my gratitude to my parents and my family, who always supported me in my education and everything else in life.

List of Abbreviations

RBS	Radio Base Station
BTS	Base Transceiver Station
1PPS	1 Pulse Per Second
I&V	Integration and Verification
NMEA	National Marine Electronic Association
GPS	Global Positioning System
FPGA	Field Programmable Gate Array
CPLD	Complex Programmable Logic Device
MCU	Microcontroller Unit
UART	Universally Asynchronous Receiver/Transmitter
SPI	Serial Protocol Interface
SNR	Signal to Noise Ratio
DOP	Dilution of Precision
GSM	Global System for Mobile
UTC	Coordinated Universal Time

Table of Contents

Table of Figures

1. Background

Ericsson is a worldwide leader in providing telecommunication equipment and related services to mobile and fixed networks globally, with its presence in more than 175 countries. It is estimated that 40 percent of all mobile calls are made through Ericsson's systems.

This thesis project is carried out at 'RBS I&V' Department which is responsible for the Integration and Verification of radio base stations for GSM. The main task of this department is to verify functions and features on system level and furthermore to measure characteristics.

RBS (Radio Base station) is used to provide communication between the user equipment (mobile phones, computers with wireless internet etc.) and the network. There are a large number of RBS in a network and the timing signals generated by the GPS (Global positioning system) receivers are used to synchronize these RBS in a network. However, it has been shown that to test the RBS in a lab environment, it is expensive and complicated to set up a real GPS comprising of a GPS receiver in the lab, an antenna on the roof, and long cables connecting the antenna and the receiver. This leads to a need for an accurate GPS receiver emulator.

2. Introduction

This thesis project is aimed at developing a GPS receiver emulator which can be used to test the RBS in a lab environment without the need of an actual GPS receiver. It is being carried out in order to avoid the complication of setting up a real GPS and also being able to simulate scenarios not possible in real time, when required.

The RBS in a network are synchronized using the timing signals from a GPS receiver. These signals include a 1PPS (1 pulse per second) signal and NMEA- data messages (see Section 5 for details), containing different kinds of information, transmitted to the RBS at the positive edge of each 1PPS signal. The digital unit of an RBS utilizes these signals to synchronize itself in a network. Future (and current) RBS projects require different radio technologies to be mixed for which GPS is used as a common synchronization standard. This requires an accurate reference to allow 2G, 3G and 4G radio standards to coexist.

The goal of the project is to design a system that can generate an accurate 1PPS signal, together with the NMEA sentences. These sentences are generated to replicate the data transmitted by a real GPS receiver. The NMEA data streams contain numerous parameters dependent upon the environment, position of satellites, UTC time etc. To construct a flexible system which allows a user to simulate any type of conditions, an option to configure the parameters which form the NMEA sentences is provided.

Another important aspect of this system is to have a control over when the data streams are sent to the RBS. Sometimes, it can also be needed to simulate situations that a real system cannot handle, e.g., for certain kinds of faults, or to ensure that internal timing signals behave as expected.

The block diagram of the system is given below. The blocks left to the RBS will be implemented in this project.

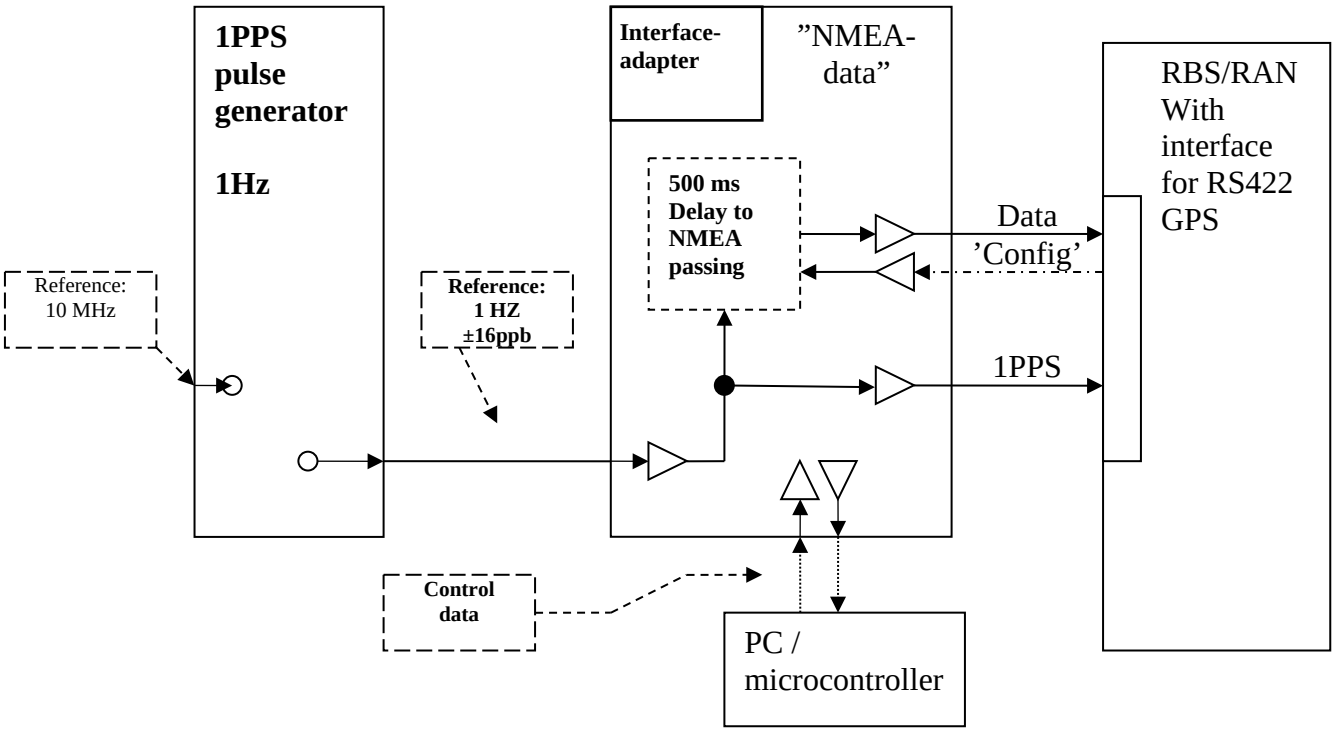

Fig. 1 Emulator Block Diagram

Some of the benefits of having such a GPS emulator are:

- No installation of antenna feeder required to get functioning GPS (NMEA data).
- The simulation can be run for any time period.
- Via control messages, the interaction between GPS and RBS can be explored by simulating fault conditions etc.

3. Synchronization using GPS

"The Global Positioning System (GPS) is a space-based satellite navigation system that provides location and time information in all weathers, anywhere on or near the Earth, where there is an unobstructed line of sight to four or more GPS satellites" [1]. A GPS receiver calculates its position by precisely timing the signals sent from the satellites. These signals include timing information at which the message was sent from the satellite and the position of the satellite at the time of transmission of the message. This information is then used to calculate the position of the receiver.

The GPS is used in various applications for different purposes. It is normally known to be used for obtaining positioning information and then using it for different purposes. It is most popularly used as a navigation device where positioning information is incorporated within maps. Apart from this popular use, it is also utilized for some other less popular but very important purposes.

The timing information along with positioning information is an essential part of the GPS information. This GPS timing information is used to synchronize RBS in a network. The GPS receiver uses the timing data received by the satellites to generate a 1PPS signal along with the periodical NMEA-0183 sentences on each pulse.

The advent of the GPS has made it much simpler to achieve time synchronization, with the help of a 1PPS signal. Even a standalone GPS receiver is enough to achieve time synchronization with a high accuracy [2]. The GPS's 1PPS signal is very stable (+/- 10 ns) as it is directly derived from atomic clocks connected to the satellites [3]. The combination of these two (1PPS and NMEA information) is then fed into the digital unit of Radio Base Stations for synchronization. The process of how this information is decoded by the Radio Base Station for synchronization is outside the scope of this paper.

There are some software based GPS simulators [4] [5] available that can generate selected NMEA sentences to simulate any condition. However, the 1PPS pulse generated through these software tools is not stable enough to meet the synchronization requirements for RBS, thus making it unsuitable. Apart from these software solutions, some hardware based GPS simulators [6] [7] [8] are also available but they only generate RF signals that are fed into a GPS receiver for the simulation purposes. As a result, these hardware simulators cannot be connected directly to the RBS. Instead, a GPS receiver is required between the simulator and the RBS.

To meet the requirements for the synchronization of the RBS that cannot be handled by the already available solutions mentioned above, a different hardware based approach is used. This method emulates a GPS receiver, instead of the RF signals transmitted from the GPS satellites and fed into a real GPS receiver for simulation.

4. 1PPS signal (1 Pulse per Second)

1PPS signal (1 pulse per second), as the name suggests, is a pulse with a frequency of 1 Hz. It is generally associated to be generated by the GPS receivers, synchronized with the UTC time [9]. For the implementation of a GPS receiver emulator, a pulse with a frequency of 1 Hz is to be generated. The requirement for the pulse, which can synchronize Radio Base Stations working on different communication standards such as WCDMA, LTE or 4G, is to have noise less than 100ns rms. There are already several products available that can generate a 1PPS signal using different approaches. However, the requirement of the 1PPS signal to be stable within 100ns rms, makes several solutions inefficient.

It is possible to generate a pulse per second using a PC internal clock. A relatively accurate method to obtain a 1PPS signal is suggested by Attila Pasztor et al. [10] by designing a software clock for PCs running UNIX. The design is based on TSC register counting CPU cycles and generating a pulse after counting cycles required for one second. An exact measurement for time for 1 CPU cycle is required. This solution provides very accurate rate stability (0.1 PPM). However, this solution does not provide a synchronized offset which makes it unsuitable for our purposes.

A low cost FPGA based 1PPS signal generator is developed by Roos et al. [11]. The system produces a local pulse per second signal, using an internal clock, which is delayed by some clock cycles of internal clock after comparing it with the1PPS signal from a GPS receiver. This is done to minimize the phase offset relative to the 1PPS output of a locked GPS receiver. This design is not suitable for our purposes as well, as it requires a GPS receiver; removal of which is the prime objective of this project.

5. NMEA-0183

The NMEA (National Marine Electronics Association) 0183 standard defines an electrical interface and a data protocol for communications between marine instrumentation [12]. The GPS receivers use this standard for communication with the digital unit of an RBS. The NMEA 0183 uses a simple ASCII, serial communication protocol to transfer these data streams.

An NMEA sentence starts with a '$' symbol, followed by the message and ends with a carriage return. A sentence is a succession of an octet (8 bits), sent in the ASCII format. The maximum length of a sentence can be up to 80 characters. There are two types of sentences that can be sent with the NMEA standard: Standard and proprietary. Standard sentences are defined by the standard association and are used as is. However, proprietary sentences are defined by some companies for their use.

In the standard sentences, the first two letters after the $ sign show the type of the device that uses that sentence (GP is used for GPS receivers). The next three characters show the type of the sentence and the rest of the sentence depends on it. A standard sentence looks like this.

$GPGGA,125301,5924.162310,N,01756.898290,E,1,9,1.29,40.497,M,23.380,M,,*65

The first 5 characters after the '$' symbol (start of sentence marker) represent the type of the sentence (transferring position information in this case) followed by the UTC, latitude, longitude and some other information. In the proprietary sentences, the first letter after the '$' sign is P which is followed by 3 letters depicting the company which uses it (ERC is used for Ericsson).

$PERC,GPppr,391997,01636,00050,08,0,0*4E

This sentence is the Ericsson's proprietary sentence, used for transmitting a 1PPS periodic report. The initial 4 characters show the company which uses this sentence, followed by the time stamp of the 1PPS signal, its standard deviation and status of GPS etc. It is not compulsory to send all the fields of each sentence, as some fields are not required by the RBS to decode information. Therefore, all the fields are classified in two categories: Mandatory and Optional.

The last field before the end of the sentence marker is for the checksum. The checksum is obtained by calculating XOR of all the bytes, preceding this field in the sentence. The checksum is a field which is optional to be sent in some sentences and mandatory in the others. It depends upon the protocol that is being used by the RBS for decoding the sentences.

There are further two types of proprietary sentences that are required by the RBS. These are protocols from Ericsson and Furuno, each developed by the companies for their own use. Ericsson messages begin with $PERC while Furuno starts with $PFEC.

The following parts describe all the sentences that are generated by the system and a sentence that is received from RBS to be decoded, along with the description of the important parameters contained in each. The details about all the parameters that are not given in this paper can be found in [13].

5.1. GPS Fix Data ($GPGGA)

This is a standard NMEA-0183 sentence and is sent to the RBS to transport positioning information. An example of this sentence is given below:

$	GPGGA	,123456	,3444.0000,N	,13521.0000,E	,1	,04	,02.00
,000123.0	,M	,0036.0	,M	,13	,0001	<CR><LF>	

The first two fields are the start of the sentence marker and the type of the sentence identifier (GPGGA). The third field (123456) is for the UTC time. This field consists of 6 characters; 2 for hours, 2 for minutes and 2 for seconds in the same order. The fourth field (3444.0000,N) and the fifth field (13521.000,E) represent Latitude and Longitude respectively.

5.2. GNSS DOP and Active Satellites ($GPGSA)

This is another standard NMEA sentence. It sends information to the RBS about the satellites used for positioning and the accuracy achieved by them. An example sentence is as follows:

$	GPGSA	,A	,3	,01	,02.00	,03.00	,04.00	<CR><LF>

As with the other sentences, the first two fields are the start of the sentence marker and the type of the sentence respectively. The last two fields in this sentence are the same as others: the checksum and the end of the sentence markers respectively. All the fields are mandatory for this sentence, except the checksum, which is optional.

5.3. GNSS Satellites in View ($GPGSV)

This is a standard NMEA sentence as well, that transfers positioning details of the satellites, to the RBS, that are in view. The example sentence is as follows:

$	GPGSV	,2	,1	,06	,01,05,254,56	,04,11,223,44	,01,75,088,32
,01,42,234,48	<CR><LF>						

The sentence starts with the standard start of the sentence marker followed by the type of the message (GPGSV). The next field (2) represents the total number of sentences that are sent,

carrying satellite details. This number of sentences depends upon the number of visible satellites. The maximum number of visible satellites is 12 and each sentence in a sequence contains information for 4 satellites. The range of this field is between 1 and 3.

The fourth field represents the number of the current sentence that is being sent. It can take a value between 1 and 9. The fifth field is for the total number of satellites in view with an elevation angle greater than 5 degrees. The range of this field is between 01 and 32.

The sixth field (01,05,254,56) represents the satellite information for the first satellite in this sentence. In the above example, '01' is the ID of the satellite which can take value between 01 and 32. '05' is the elevation angle of the satellite in degrees with range from 05 to 90. '254' is the bearing angle which has a range between 000 and 359. The last subfield (56) is for SNR (Signal to noise ratio).

The next three fields hold the same information as in the sixth field, for second, third and fourth satellite. An example of a full sequence of sentence in a scenario when 10 satellites are visible is as follows:

```
$GPGSV,3,1,10,04,15,105,41,09,72,168,46,12,53,245,39,14,31,311,42
$GPGSV,3,2,10,15,10,188,41,17,39,060,49,22,06,281,37,24,05,004,39
$GPGSV,3,3,10,25,18,252,39,27,58,148,39
```

A checksum field is also added for Ericsson protocol for each sentence, as it is a mandatory field. However, it is omitted in Furuno Protocol.

5.4. PeriodicPPSReport ($PERC,GPppr)

A Radio Base Station uses the information in the PeriodicPPSReport sentence to determine the time stamp of the 1PPS signal, the standard deviation of the 1PPS signal and the status of the synchronization source (GPS). An example message is as follows:

$	**PERC,GPppr**	**,322768**	**,01025**	**,00034**	**,06**	**,0**	**,0**	***F4**	**<CR><LF>**

The message starts with a '$' symbol marking the start of the sentence. The second field (PERC,GPppr) points to the type of the sentence. The next field (322768) is for GPS TOW. GPS TOW stands for GPS time of week indicating the number of seconds that have elapsed since the beginning of the current week. A new week begins at 0000 on Monday. The range of this field is from 000000 to 604799.

The fourth field (01025) gives the GPS Week. The GPS week indicates the number of weeks passed since January 6th, 1980. The range of this field is between 00000 and 65535.

The next field (06) gives the number of GPS satellites used in calculating GPS Week and GPS TOW for the next 1PPS signal. This value is only valid if the GPS receiver is up and running. The range of the field is between 00 and 32.

The ninth field is for checksum. Checksum is mandatory for this message. The checksum field is followed by two characters as end of sentence markers. All the fields in this message are mandatory to be sent.

5.5. GPSS Status ($PERC,GPsts)

This sentence, part of the Ericsson protocol, reports the mode of the 1PPS signal state machine and confirmation of the 1PPS signal state machine configuration, along with the capabilities supported by the GPS receiver to the RBS. An example of this message is given below.

$	**PERC,GPsts**	**1,**	**0,**	**0,**	**1111,**	***hh**	**<CR><LF>**

The first two fields are the usual start of sentence parameter ($) and the sentence descriptor respectively. The third field tells about the 1PPS signal state machine mode. It can take the values of ‘0’, ‘1’, ‘2’ and ‘3’.

- ‘0’ represents Acquisition mode
- ‘1’ represents Survey mode
- ‘2’ represents Position-hold mode
- ‘3’ represents Acquisition-Mode (Position-Hold Completed)

This field is then followed by the usual checksum and the end of the sentence fields. All the fields for this sentence are also mandatory.

5.6. GPSS Averaged Position ($PERC,GPavp)

This sentence is also a part of the Ericsson protocol, as can be seen in the sentence descriptor. It reports the position value used in Position-Hold mode to RBS. An example sentence is as follows:

$	**PERC,Gpavp**	**,3444.0000,N**	**,13521.0000,E**	**,000123.0**	**,M**	***hh**	**<CR><LF>**

The standard first two fields (start of sentence marker and sentence type descriptor) are then followed by Latitude. This third and the fourth field contain the average latitude and longitude used in Position-Hold-Mode. The last two fields are the standard fields: checksum and end of sentence markers. All the fields, as with all the sentences for Ericsson protocol, are mandatory.

5.7. Time and Pulse Output ($PFEC, GPtps)

This sentence is part of Furuno protocol, sent to the RBS. It contains information about the availability of 1PPS signal and the GPS timestamp of the 1PPS signal which is used by the RBS. An example of this sentence is as follow:

$	**PFEC,GPtps**	**,940630123000**	**,3**	**,1**	**,1**	**,940701000000**	**,+1**	**,10**
,940626120000		**,0755**	**,390610**		***B2**	**<CR><LF>**		

The sentence begins with the standard two fields: start of sentence marker ($) and type of sentence (PFEC, GPtps). The following field contains the current date and time. This field has 12 characters, 2 characters each for year, month, day, hour, minutes and seconds respectively. The range of year is expressed between 1994 and 2040.

The seventh field (940701000000), containing 12 characters, is UTC Leap Second Adjustment Date/Time. It indicates the date and time for the next UTC Leap Second Adjustment. The format of the date/time is same as it is in the third field for the current date and time.

The next field is GPS weeks which indicates the number of weeks since January 6th, 1980. This field can have a value between 00000 and 65535. The value of this field is valid if the Time Standard ID is ‘GPS’ or ‘UTC’.

The eleventh field is GPS TOW which indicates the number of seconds elapsed since the beginning of the current week. The range of this field is between 000000 and 604799. The last two fields are the standard fields for the sentences of Ericsson protocol. They are checksum and end of sentence markers. All the fields for this sentence are mandatory.

5.8. Almanac date and satellite’s health condition ($PFEC,GPanc)

This is the second message which is part of the Furuno protocol. It sends the almanac date and satellites’ health condition to RBS. An example message is as follows:

$	**PFEC,GPanc**	**,020102030405**	**,22222200222222222222000000222221**	**<CR><LF>**

The first two fields are the start of sentence marker and sentence type descriptor. The third field is the almanac date. This field consists of 12 characters: 2 for year, 2 for month, 2 for day, 2 for hour, 2 for minutes and 2 for seconds respectively. The last field is the standard end of sentence markers. All the fields are mandatory for this sentence.

5.9. Self-test Results ($PFEC,GPtst)

This sentence, part of Furuno protocol, is sent to RBS. This is a test sentence which is transmitted to check the connection between the RBS and the GPS receiver. The sentence contains the GPS receiver's status information along with its version number. Following is an example of the sentence:

$	PFEC,GPtst	,0	,1234567123	,0	,0	<CR><LF>

5.10. Request Output ($PFEC,GPint)

This sentence is sent from RBS to set the output periods for the sentences. An example of the sentence is as follows:

$	PFEC,Gpint	,tps01	,tst00	<CR><LF>

The first two fields are the start of the sentence and the sentence type descriptors. The following fields are the Interval Definition which can vary in length. Each interval definition consists of 5 characters follows by a comma. The first three characters point to the sentence whose period is being configured and the last two numbers the period value. The above example sets the period for GPtps message to 1 second and the period for GPtst to be 0. By setting the period to 0, it is established that this sentence is transmitted only once and is not sent again unless this sentence is received again. There are a number of sentences whose periods can be configured using this sentence. However, only the ones that are implemented in this system are explained in the this paper (see section 7.2.9).

6. Hardware

To accomplish the development of a data emulator that can perform the tasks mentioned earlier, the following hardware is used:

- Altera MAX II Development Kit [14]
- Atmel STK 500 Development Board [15] with Atmel ATmega 644P [16]
- Atmel AT45DB011D Flash Memory [17]

A connection diagram of all the necessary components of the system is given in Fig. 2.

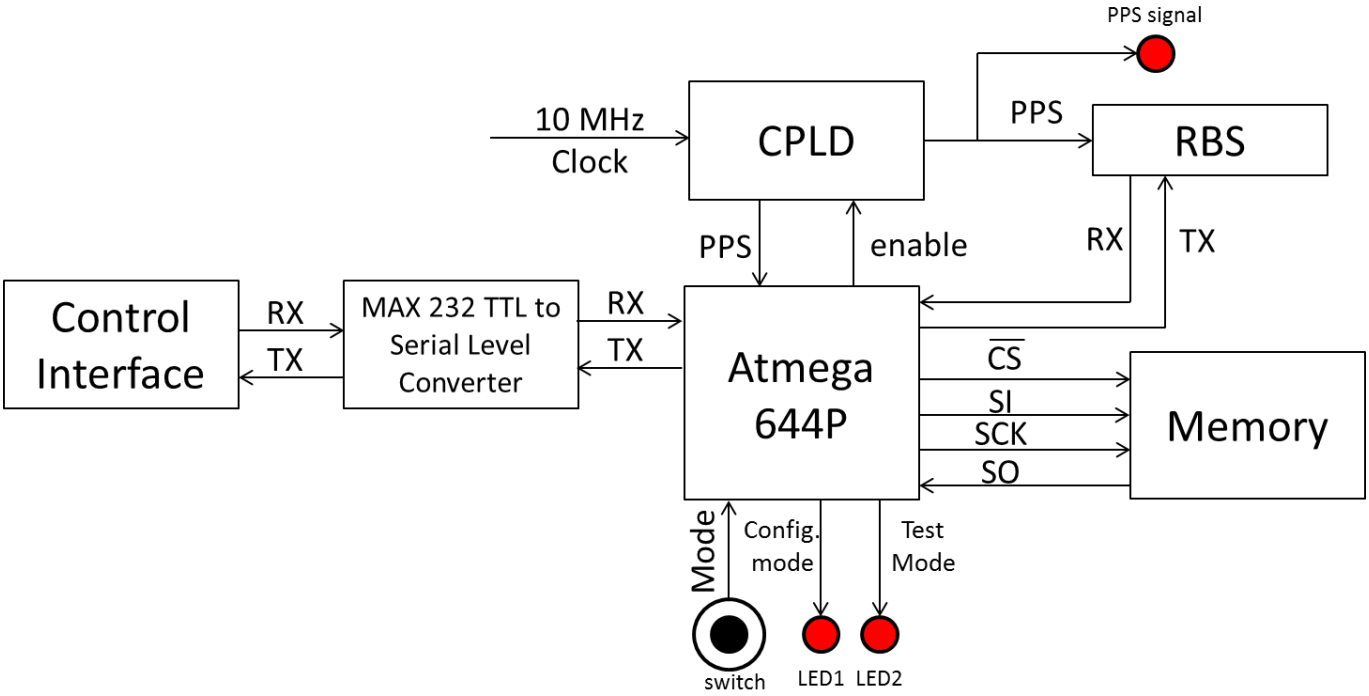

Fig. 2 Connection Diagram of the system

6.1. The Altera MAX II Development Kit

The generation of the 1PPS signal is accomplished through a CPLD. As the digital design for this purpose is small and only requires a counter that can count up to 10,000,000 coupled with a few inputs and outputs, an evaluation board with CPLD that is minimal in size is used for implementation. The smallest chip that could be found on a development board is Altera MAX II which has EPM1270F256C5 chip on it. This chip contains 1270 logic elements. A picture of the development board can be seen in Fig. 3.

Fig. 3 Altera MAX II Development Board

6.2. The Atmel STK 500 Development Board with Atmel ATmega 644P

The generator for the NMEA sentences is implemented on an Atmel Atmega644P along with an Atmel STK 500 development board. The STK500 board is compatible with several Atmel MCUs that can be plugged in for the development purposes. The requirements of the system include two serial interfaces, one for the transmission of the NMEA sentences to the RBS and the other one for the communication with the control interface on a PC.

The development of the system was started with an Atmel ATmega 162 [18]. However, it's 16 KB of Flash and 1 KB of SRAM proved insufficient. Therefore, it was replaced with an ATmega644P. This MCU has 64KB of Flash memory to store code and 16 KB of SRAM. Besides the memory requirements for the code, the ATmega 644P is also selected for the development to cater for the requirement of two UART interfaces. An external interrupt interface is also utilized by connecting it with the 1Pulse per Second Signal from the CPLD. SPI (Serial Protocol Interface) [19] is used to connect to an external memory. The STK-500 development board is shown in Fig. 4.

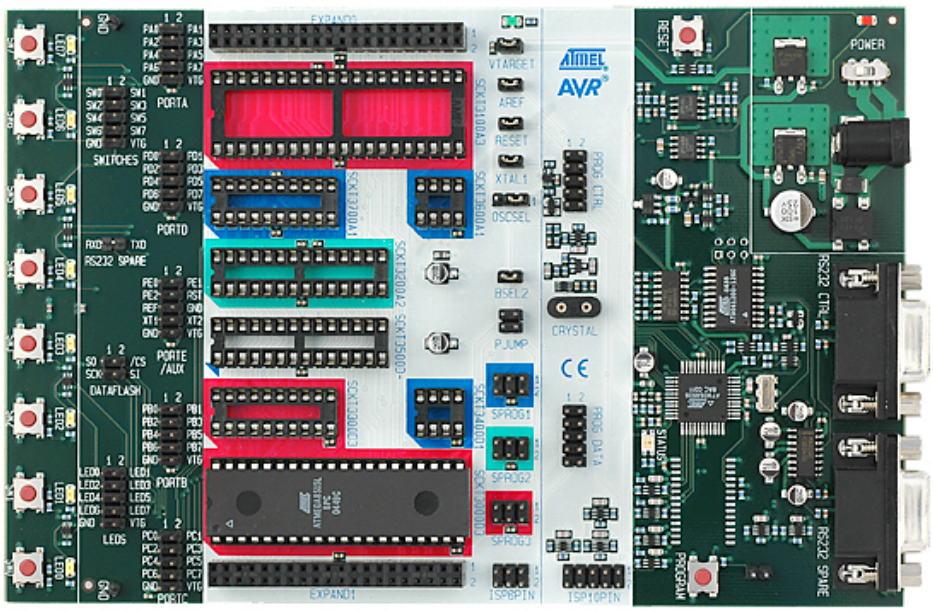

Fig. 4 STK 500 Development Board

The pin out diagram for Atmega644P is given in Fig. 5. The functionality of the pins used is as follows:

- Pin 14, 15 and pin 16, 17 are used for the two serial UART interfaces (RX and TX pins).
- Pin 3 is connected to 1PPS signal output pin of the CPLD.
- Pin 4 is connected enable pin on the CPLD.
- Pin 36 is for a switch that can select the emulator mode.
- Pin 22 and 23 are for LEDs to signal the mode the emulator is in.
- Pin 5, 6, 7 and 8 form the SPI ($\overline{SS}$, SCK, SI, SO) to connect to the external memory.

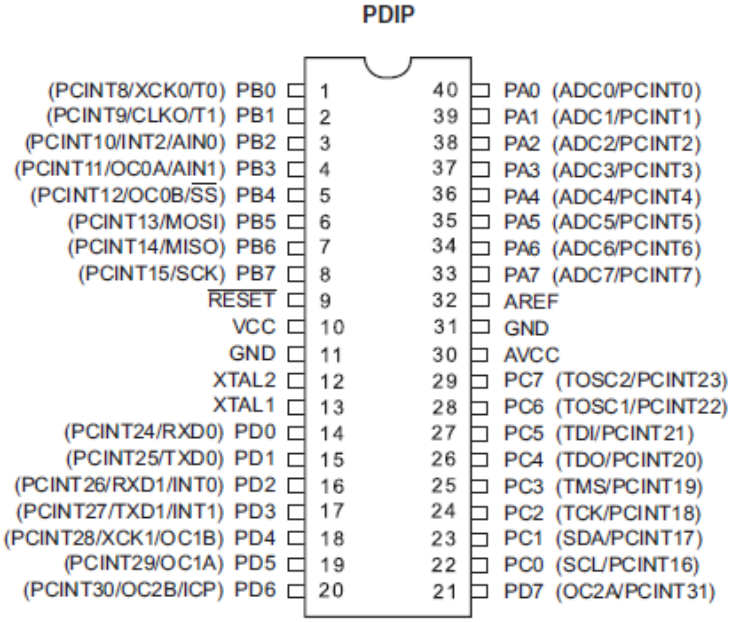

Fig. 5 Atmega 644P Pin Diagram

6.3. The Atmel AT45DB011D Flash Memory

The AT45DB011D is a 1 M-bit, 2.7V serial-interface memory, suitable for storing larger amounts of data. Its 1,081,344 bits of memory is organized in 512 pages, 264 bytes each. The memory also consists of a buffer of size 264 bytes. The memory is enabled through a chip select pin ($\overline{CS}$) and accessed via a three-wire interface consisting of the Serial Input (SI), Serial Output (SO), and the Serial Clock (SCK). These four pins combine to form a Serial Peripheral Interface (SPI). The pin diagram of the memory is given in Fig. 6.

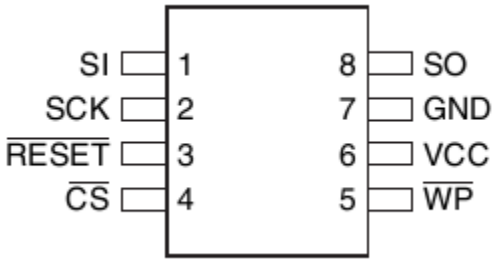

Fig. 6 Atmel AT45DB011D Pin Diagram

$\overline{CS}$ is the chip select pin which selects the memory when it is asserted and deselects it when it is deasserted. When deselected, the memory does not accept any data and goes in standby mode. A high-to-low transition on $\overline{CS}$ pin is required to start an operation and a low-to-high transition to end an operation.

There are different modes in which data can be read / written from/to the memory. However, only two modes (one for read operations and one for write operations) are used.

6.3.1. Main Memory Page Program through Buffer

For writing the data to the memory, a mode called *'Main Memory Page Program through Buffer'* is used. In this mode, the data is stored in the buffer first and then written into the addressed page. This operation is started by asserting the $\overline{CS}$ signal to select the memory, followed by clocking in the op-code (82 Hex). This is followed by three bytes for addressing. The addressing bytes consist of 6 don't care bits, 9 bit page address to select the page to write the data in, and 9 bit buffer address to select the first byte in the buffer to be written. These address bytes are followed by the data bytes which are stored in the buffer from the specified starting address. When the memory is deselected by a low-to-high transition on the $\overline{CS}$ pin, the data in the memory buffer is then transferred into the specified memory page. During this time, the status register indicates that the memory is busy.

The timing diagram to write the data is given in Fig. 7.

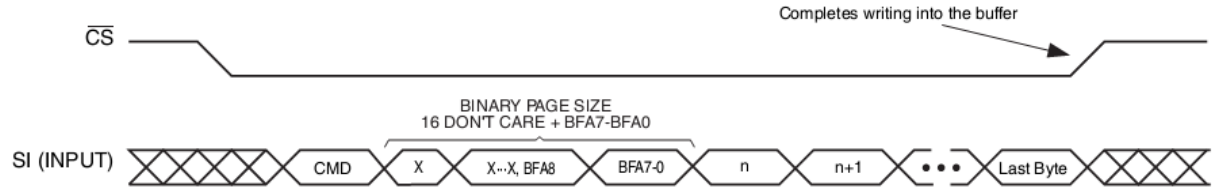

Fig. 7 Timing Diagram for a Write Operation

6.3.2. Continuous Array Read

For reading the blocks of data from the memory, the *'Continuous Array Read'* command is used. In this mode, the data can be read sequentially from the memory by providing only the starting address, followed by the clock signals based on the number of bytes to be read. No other addressing information or control signals are required to read the data. This process is also started by selecting the chip as in the other modes. The next step is to clock in the op-code (E8 Hex for this command). Then, three address bytes (for page and memory) are sent from where the reading should start in the memory, followed by four don't care bytes. These don't care bytes are required to initialize the read operation. The clock pulses that follow will output the data on SO pin, until the chip is deselected to terminate the process. The timing diagram to perform read operation using the Continuous Read Array Operation is given in Fig. 8.

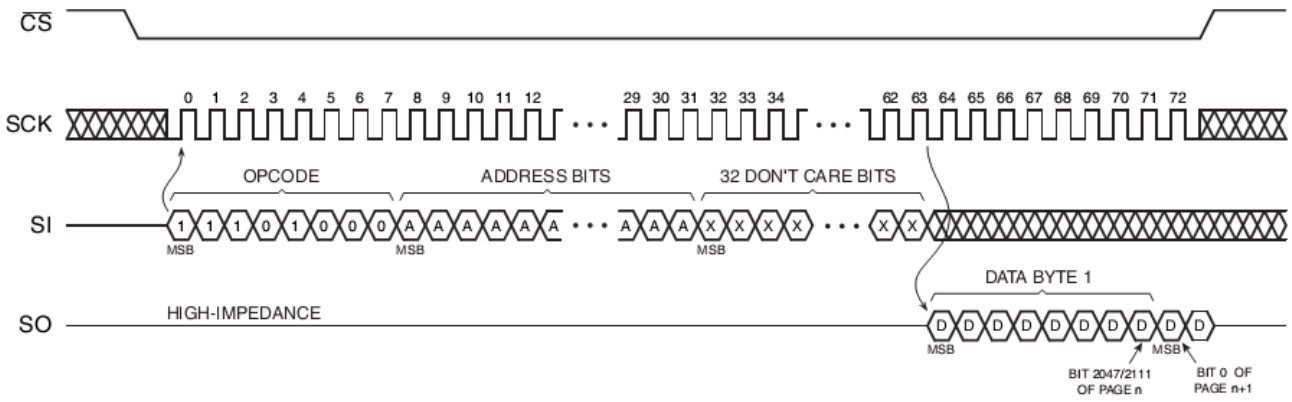

Fig. 8 Timing Diagram for a Continuous Read Array Operation

7. Implementation

The NMEA data emulator is capable of operating in several modes. These include Configuration mode, Normal mode and Test Mode. During the configuration mode, no data is transmitted to the RBS and generation of 1PPS signal is suspended. Instead, only instructions can be received from the control interface on a PC to configure the parameters that constitute different NMEA sentences. It must also be noted that these parameters can only be configured when the device is in the Configuration mode.

In the Normal mode, the emulator runs freely once it is started by pressing the start button from the control interface, generating the appropriate NMEA sentences corresponding to their respective periods, on the positive edge of each 1PPS signal. The parameters cannot be configured in this mode and the mode has to be switched to configuration mode by pressing the mode button on the device to do so.

The Test mode is implemented with the purpose of testing the system to see if it generates the correct messages corresponding to their time periods, in a lesser time. In this mode, the NMEA sentences are not generated depending on the 1PPS signal. Instead, they are generated with an interval of 100ms between two sentences. This is done in order to be able to see the results of the sentences more quickly and in shorter time. This mode can be chosen by pressing the mode button twice when the device is in normal mode and by pressing the button once when in the Configuration Mode.

Two LEDs (LED1 and LED2) on the device, as well as status of the emulator on the control interface, show the mode that it is currently in. If both the LEDs are off, then the device is in the Normal Mode. In the Configuration Mode, LED1 turns on while LED2 turns on in Test mode.

Another mode implemented in this system is called the Programming mode. This mode is responsible for storing the data in an external 1 Mbit flash memory, Atmel AT45DB011D, connected to the MCU. This data contains the positioning information of 16 satellites sampled over a day. This is an independent process and can only be performed when the emulator is not running. During this mode, all other functionalities of the device are disabled and are enabled only when the programming process finishes. The details about the functionality and methodology of this mode are described in Section 7.3.1.

The NMEA-data emulator design is divided into three components: a 1PPS signal Generator, an NMEA Generator, a Control Interface and an External Memory Programmer.

i) The 1PPS signal Generator is implemented on a CPLD (Altera MAX II) and it is responsible for 1PPS signal generation.

ii) The NMEA Generator is responsible for the generation of NMEA sentences transmitted to the RBS along with the decoding of sentences received from the RBS and is implemented on the MCU (Atmel ATmega 644P).

iii) The Control Interface is implemented in Microsoft Visual C# Express and is responsible for configuring the different parameters that make up NMEA sentences.

iv) An external memory is interfaced to the MCU through SPI. Satellites' positioning information, for the period of a whole day, is stored in the external memory and is retrieved later every minute by NMEA generator for the generation of NMEA GPGSV sentence.

The details of the control interface are not given in this paper as it is done by a project partner. The block diagram of the full design is given in Fig. 9.

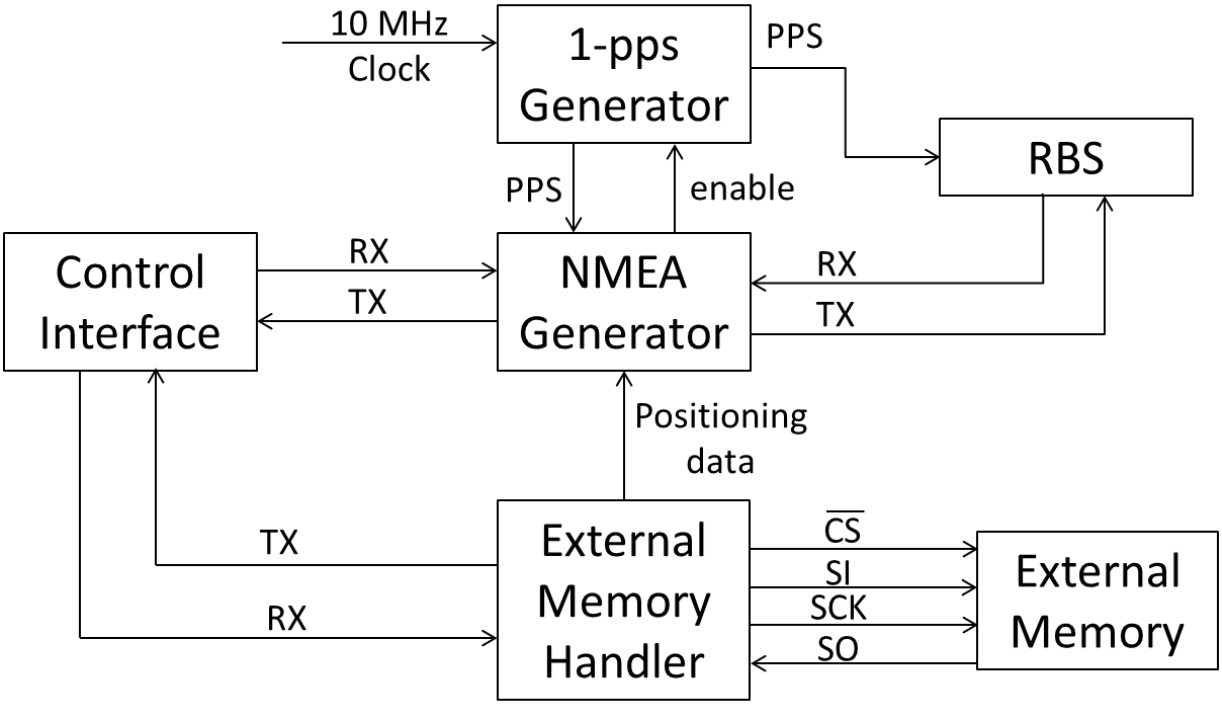

Fig. 9 Block Diagram of the whole system

7.1. 1PPS signal generator

The 1PPS signal Generator is implemented on Altera MAX II development board using Altera Quartus II and VHDL for writing code. The 1PPS signal generator is responsible for the generation of 1 pulse per second (1 Hz) which is fed to the MCU to act as a trigger for the NMEA generator to transmit data. The requirement on the stability of the 1PPS signal is to be not more

than 100 ns rms and frequency stability to be +/- 16 ppb. To achieve such a high level of stability, the idea of the design is to divide a 10 MHz clock to 1PPS signal with a clock divider.

It could be possible to generate a pulse by using the timer interrupt of the MCU. However, generating an exact 1 second interrupt using a 10 MHz clock source and a 16 bit timer is not possible. The formula to calculate the time for a timer interrupt is as follows:

$$F_{out} = \frac{F_{clk}/_{Prescalar\ factor}}{Counter\ value}$$

Where F_{clk} is the frequency of the master clock (10 MHz in this scenario), Pre-scalar factor is the factor by which the clock is divided for the counter and Counter value is the value which the counter should count up to in order to get the required frequency F_{out} (1 Hz).

By plugging in different values of the pre-scalar in the above formula, it can be seen that no integer value can be obtained to get an interrupt exactly every 1s. This makes it impossible to generate an exact 1PPS signal using the MCU built-in timers.

A solution to this problem has been provided in [20] by using Bresenham's algorithm [21]. Using this algorithm, two imperfect periods can be transformed to produce an average signal that can match any period. However, the 1PPS that can be generated by this technique gives an 'averaged' 1Hz pulse. This means that each generated pulse does not have the frequency of exact 1Hz. However, over a period of time, the average would be 1Hz. This makes it inefficient for the synchronization of the RBS.

A safer approach is to design a clock divider on the MAX II CPLD which can count the master clock pulses and generate an accurate 1PPS signal. The accuracy and jitter of the resulting 1PPS signal depends significantly upon the clock source. However, it can be improved using a stable Rubidium clock source [22].

An external clock signal of 10 MHz, from a Rubidium clock, is fed into the design. The design implemented in this system consists of a counter which counts up to 10,000,000 at every positive edge of the clock, generates a pulse and starts over. The counter also depends upon another input signal 'enable', from the NMEA generator, which controls when to start counting and generate pulses. This is done in order to provide control, so that the 1PPS signal is not generated when the emulator is not in run mode and is only generated when required by the NMEA generator.

7.2. NMEA Generator

The NMEA generator is the backbone of the system upon which the core functionality of the system depends. This module is responsible for the following:

- Generation of NMEA sentences (Ericsson or Furuno protocol) at pre-defined intervals and its transmission to RBS
- Decoding of NMEA sentences received from RBS
- Decoding of the messages received from the Control Interface to configure parameters for NMEA sentences to simulate any type of conditions
- Controlling the start and end of emulation

These features are achieved by designing and implementing the following architecture, shown in Fig. 10.

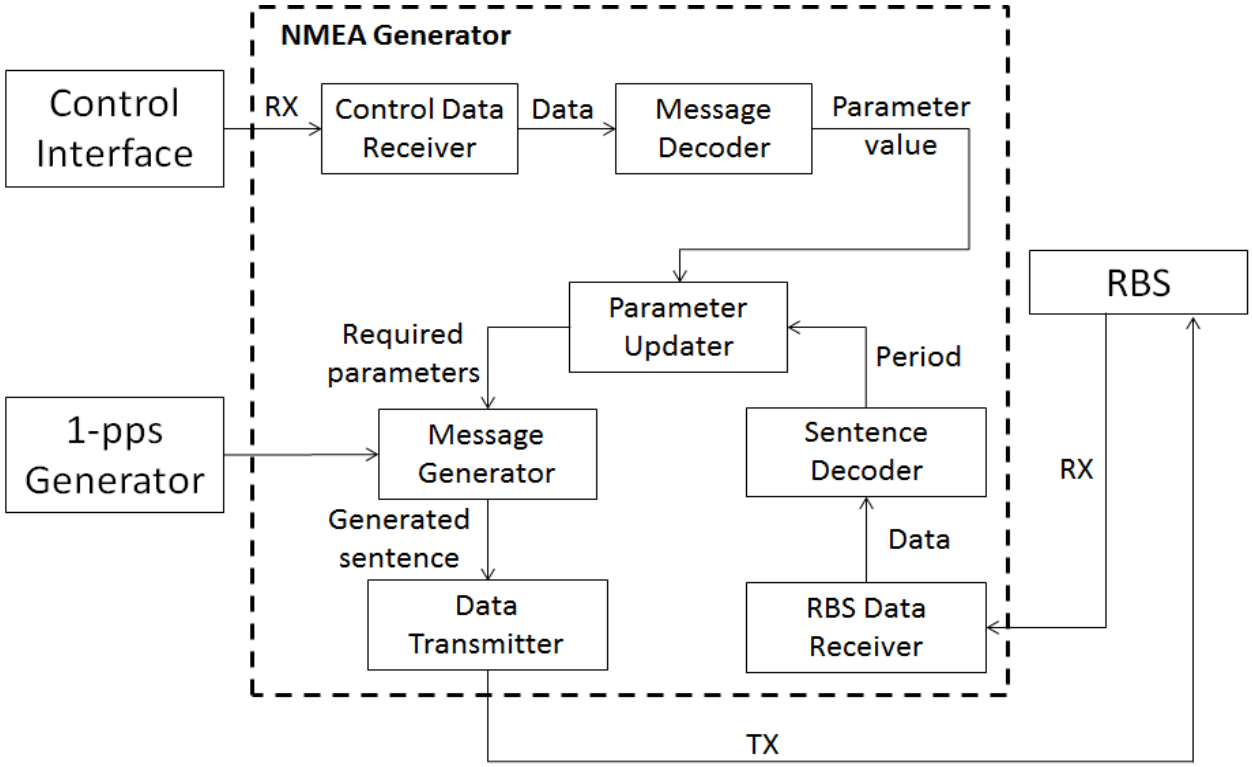

Fig. 10 NMEA Generator Block Diagram

In the above figure, it can be seen that there are numerous blocks that have their responsibilities. Each block is detailed with its functionality in the coming sections.

7.2.1. Control Data Receiver

The control interface sends data through one of the serial ports and is handled in the MCU's UART. There are two UART interfaces on ATmega644P. One is utilized to communicate with the Control Interface and the other for the communication with the RBS.

The serial communication is done at 9600 bps with 1 stop bit and no parity bit. UART on the MCU is configured such that each time a byte is received by the UART, an interrupt occurs. On the generation of every interrupt, the data is fetched from the UART register and is stored in a buffer in the code, which is then decoded by the Control Message Decoder (Section 7.2.2). The C code for the configuration of MCU UART is given in Appendix A.

7.2.2. Control Message Decoder

The Message Decoder relies on the characters received in the Control Data Receiver, which are decoded in this block. An instruction format has been formulated to facilitate the decoding of the incoming messages. The instruction format with all the instructions is given in Section 7.2.3. Each instruction starts with a '#' mark, followed by the instruction number and the message contents. The decoder looks for the '#' mark and then converts the next two received ASCII characters into hex format to obtain an instruction number. When this instruction number becomes available, the following characters (based on the instruction number) are then written into its respective parameter storage.

The instruction format has no end of instruction sign. This puts an extra responsibility on the control interface to send the messages using the correct format and with the exact message length. The control interface takes care of this limitation and checks the correctness of the messages before transmission.

7.2.3. Receiving Instruction Format

The instructions that are required for the current system are 23 in total. However, they can be increased if needed so. They not only configure the parameters for all the messages but also select the protocol whose messages are to be sent to the RBS. The description of each instruction along with the format is given below:

1. For configuring the number of visible satellites for GPGSV

#	01	No. Of Satellites (2 chars)

2. For configuring satellite information for each of visible satellites for GPGSV

#	02	Sat. No (2 chars)	ID (2 chars)	Elevation Angle (2 chars)	Bearing Angle (3 chars)	SNR (2 chars)

3. For configuring Health Conditions for 32 satellites for GPanc

#	03	Health (32 chars)

4. For configuring latitude for GPGGA

#	04	Degrees (2 chars) 00-90	Minutes (2 chars) 00-59	Separator '.'	Fractional Minutes 0000-9999	Separator ','	Direction N or S

5. For configuring Longitude for GPGGA

#	05	Degrees (3 chars) 000-180	Minutes (2 chars) 00-59	Separator '.'	Fractional Minutes 0000-9999	Separator ','	Direction N or S

6. For configuring GPS Status for GPGGA

#	06	Status (1 char) 0,1,2

7. For configuring Number of Tracking Satellites used for positioning

#	07	No of satellites (2 char) 00-32

8. For configuring DOP for GPGGA

#	08	DOP (5 chars)

9. For configuring Antenna Altitude for GPGGA

#	09	Altitude (8 chars) -00999.9-017999.9

10. For configuring Geoid Altitude for GPGGA

#	10	Altitude (6 chars) -999.9-9999.9

11. For configuring Operational Mode for GPGSA

#	11	Mode 'M' or 'A'

12. For configuring Positioning Status for GPGSA

#	12	Status (1 char) 1,2,3

13. For configuring the number of GPS weeks based on the current UTC time

#	13	GPS Weeks (4 char) 0000-3182

14. For configuring PDOP for GPGSA

#	14	PDOP (5 char) XX.XX

15. For configuring HDOP for GPGSA

#	15	HDOP (5 char) XX.XX

16. For configuring VDOP for GPGSA

#	16	VDOP (5 char) XX.XX

17. For Protocol Selection (Ericsson or Furuno)

#	17	Mode (1 char) 0 for Furuno, 1 for Ericsson

18. For configuring number of GPS seconds passed in the current week

#	18	GPS Seconds (6 Chars) 000000-604800

19. For configuring GPSS Status for GPppr

#	19	Status (1 char) 0 = GPS locked 1 = Free running

20. For configuring State Mode for GPsts

#	20	State 0 = Acquisition Mode 1 = Survey Mode 2 = Position Hold Mode

21. For configuring Position Hold Mode for GPsts

#	21	Pmode 0 = P hold mode not disabled 1 = P hold mode disabled

22. For configuring UTC time

#	22	Time (12 chars) YYMMDDhhmmss

23. For configuring GPGSV mode

#	23	Mode (1 char) 1 for Dynamic Mode 0 for Static Mode

7.2.4. Parameter Updater

On the successful decoding of a message, the message characters contained in it are written into their respective parameter storage spaces. These stored parameters are used by the sentence generator to generate NMEA sentences.

7.2.5. Sentence Generator

There is a total of 8 different sentences, of two different protocols, that are required to be generated in the emulator. Some of the sentences are generated for both the Ericsson and Furuno protocol. Each sentence is characterized by its specific time period after which it should be generated. This block generates the sentences that are to be sent on each 1PPS signal, depending on its turn.

A variable is assigned for each sentence to keep track of the time for which it is not sent. This variable is incremented every second and when the turn comes of a specific sentence to be generated, its time tracking variable is reset to 0.

A buffer, 300 bytes in size, is used to save the generated sentences that are to be transmitted on each pulse. This results in a limit to the number of characters that can be sent in one pulse. Although the maximum number of characters that can be sent in one pulse are 480 when communicating at 9600 bps, a buffer of 300 bytes is allocated to send these characters due to limited memory on the MCU. This does not affect the system severely because the size of the message for one pulse exceeds 300 bytes rarely. All the sentences have large periods and the instances on which the message size can become over 300 bytes are seldom.

However, this rare situation is also taken into account in the design of the system and the problem is rectified by keeping a message queue. In scenarios, when the messages to be sent sum over 300 bytes, the IDs of the messages that cannot be sent during the current pulse are stored in the queue and are sent on the next pulse. This makes sure that none of the sentences miss the transmission.

A length parameter is also updated during each cycle which informs the Data Transmitter (see Section 7.2.7), with the number of characters that are to be transmitted for the current pulse.

The procedure for the generation of each sentence is given in Section 7.2.6.

7.2.6. Procedure for Sentence Generation

Each of the 8 sentences to be generated for the purposes of this data emulator has different parameters, each with different requirements. Each sentence has a default period which can be changed by sending the GPint sentence along with the appropriate parameters. The description of the parameters for each sentence is given in Section 5. The generation details about the 6 messages that are implemented by me are given below:

7.2.6.1. GPtps Generator

The GPtps (time and pulse output) contains information about the availability of the 1PPS signal and its associated time stamp. This sentence is generated every second and is part

of the Furuno protocol. The length of this sentence is fixed (76 bytes) and is added to the total length variable of the message buffer. As this sentence is to be sent every second, it is written to the buffer first every time and then the rest of the sentences for that time instant are written into the buffer.

One of the most important parameter for this message is the UTC time. A function is implemented to calculate the current UTC time. On every 1PPS signal, a variable calculating the seconds is incremented by 1 which in turn increments the minute variable every 60 seconds. The hours, days, months and years are also updated in the same way.

Some of the other important parameters include a count of the GPS weeks and a count of the seconds in the current GPS week. The count of the GPS weeks is kept updated by counting the number of seconds in a week till it reaches 604799, incrementing the GPS weeks by 1 and starting over the count from 0. However, the number of seconds in a week (604800) can not be counted in a 16 bit variable. As a result, two variables are incorporated to do the counting.

The first variable counts up to 6048, increments the second variable and resets itself to 0. When the second variable reaches 100, it resets itself and increments the GPS week by 1. These two 16 bit variables are used instead of one 32 bit variable because of the inability of sprintf command [23], used to form a string, to handle 32 bit integers.

7.2.6.2. GPGSV Generator

The GPGSV (GNSS Satellites in view) is used to transport the satellite details. It is part of the Furuno protocol and is sent every 59 seconds. The generation of this sentence is dependent upon the mode of the system which affects the way this sentence is generated. This GPGSV mode is selectable from the control interface and is configurable by sending the instruction number 23, given in Section 7.2.3.

The system is designed in such a way that the positioning information of the GPS satellites can be generated in two ways. One way is to manually configure the satellite information (Bearing Angle, Elevation Angle and SNR, see Section 5.3 for details) through the control interface. The positioning information can be stored for a maximum of 12 satellites. If this method is used, the positioning information stays static for the period of one full simulation.

The other method is by storing the satellite positioning information of 16 satellites for a full day, in an external memory attached to the ATmega644P. This information can be obtained by sampling the positioning information from a live GPS every minute for the duration of a day and saving it in a text file. The positioning information for the satellites

is retrieved from the memory every minute and is used as the parameters for the generation of GPGSV sentence. The details about storing and retrieving information from the external memory are given in Section 6.3.

Even after receiving the parameters, the generation of this sentence is still a bit tricky as the content of this sentence is dynamic, depending solely upon the number of visible satellites. GPGSV can generate up to three messages in one go, each message containing information for four satellites. The number of visible satellites to the GPS receiver decides the number of messages generated for each pulse. Since the number of messages and information contained in it depends upon the visible satellites, which can change over time, the length of the message is dynamic. It is calculated each time the message is generated and is added to the total message length. The C code for this function can be seen in Appendix C.

7.2.6.3. GPanc Generator

The GPanc (Almanac Date and Satellite's Health) sends the almanac date and the satellites' health conditions to the RBS every 49 seconds and is part of the Furuno protocol. This message is generated by appending the time parameter along with the satellite health. This satellite health parameter is configurable through the configuration interface. The total length of the message is fixed (59 bytes) and is added to the length variable when generated. The details about this sentence can be found in Section 5.8

7.2.6.4. GPGGA Generator

The GPGGA (GPS Fixed Data) is a standard NMEA-0183 sentence and is sent under both Furuno and Ericsson protocol, with a periodicity of 60 seconds. The sentence is responsible for sending the positioning information. This sentence contains several parameters, details of which are given in Section 5.1. All these parameters are appended together using the sprintf statement, as for the other sentences. All of the parameters in this sentence are configurable through the configuration interface. The length of the message is fixed (73 bytes).

7.2.6.5. GPGSA Generator

The GPGSA (The GNSS DOP and Active Satellites) is also a standard NMEA-0183 sentence and is transmitted for both Furuno and Ericsson protocol. The period of the sentence is 53 seconds. This sentence is also responsible for sending positioning information and contains fixed number of bytes (33). For the generation of this sentence, all the parameters are appended together through sprintf statement as earlier and are also configurable through the computer. The details about the parameters can be found in Section 5.2

7.2.6.6. GPtst Generator

The GPtst is a sentence used to self-test the RBS to check if it is alive. This is part of the Ericsson protocol. As it is only a test message, it is only generated when it is requested by the RBS through GPint sentence. Theoretically, it has a period 0, which means that it is only generated once when requested. The length of the message is fixed i.e. 30 bytes. All the required parameters to form this sentence are appended together with some dummy values through sprintf statement. The dummy values are used to constitute the sentence because it is just a test sentence and does not affect the behavior of the system in any way. For details about this sentence, see Section 5.9

7.2.7. Data Transmitter

Once the message is generated that is to be sent for the second pulse, it is then transmitted to the digital unit of the RBS. The data transmitter utilizes the UART interface of the MCU and sends each character using the transmitter interrupt. The transmit interrupt occurs each time the transmission is completed. The UART interface is configured to operate at 9600 bps, 1 stop bit and no parity.

For the transmission purposes, a pointer is used that points to a byte in the buffer that is to be sent at that particular time instant. This buffer is filled during sentence generation process and a length variable for this buffer is also updated. In this way, the transmitter is aware of the length of the buffer and the number of bytes to be sent during a transmission cycle.

Each time, before a new message transmission is initiated, this pointer is reset to 0 (start of the buffer). When the transmission of a byte is completed, an interrupt is raised on which the pointer of the buffer increments by 1 and sends the next character from the buffer. This procedure repeats till the pointer to the buffer becomes equal to the length of the message, which is updated each time when the sentence generation for the current pulse is completed. That point also signals the completion of the transmission of data to the RBS in the buffer.

7.2.8. RBS Data Receiver

This block receives data from the RBS which is stored to be decoded by the Sentence Decoder. This block functions in the same way as the Control Data Receiver, with the second UART being configured in the same way as the first one. The communication is done at 9600 bps along with no parity bit and 1 stop bit. The UART generates an interrupt each time it receives a byte and stores it into a buffer.

7.2.9. Sentence Decoder

The data that is stored in the buffer by RBS Data Receiver is decoded by the Sentence Decoder. Currently, there is only one sentence from the RBS that is being decoded in this system, the details of which are given below.

PFEC, GPint (see Section 5.10 for details) is responsible for setting up the periods of each sentence. The sentences whose generation periods can be configured in this system are given in Table 1 below.

Parameter	Range	Configured Sentence
tps<nn>	tps00-tps60	GPtps
anc<nn>	anc00-anc60	GPanc
GGA<nn>	GGA00-GGA60	GPGGA
GSA<nn>	GSA00-GSA60	GPGSA
GSV<nn>	GSV00-GSV60	GPGSV

Table 1 GPint sentence parameters

The decoder looks for a‘$’ symbol to mark the start of the decoding process and a carriage return (‘\r’) symbol to signal its end. Once the decoding starts, the decoder compares the next 10 characters with “PFEC,GPint”. If this comparison comes out to be true, the decoder moves on to the next step.

The next step is to decode the sentences whose periods are to be modified. In this step, the decoder looks for the next three characters. If these characters match with any of the parameters in the above table, the following two characters are stored as that sentence’s new period. The two characters for the period are, however, converted from ASCII to decimal and then saved into the corresponding variable. This process is repeated until the carriage return is detected. Upon its detection, the decoding process is terminated.

7.3. External Memory Handler

An external memory, Atmel AT45DB011D, is interfaced to the MCU through SPI (Serial Peripheral Interface bus). This external memory keeps the positioning information for satellites. This information comprises of number of visible satellites, number of satellites tracking the position followed by Bearing Angle, Elevation Angle and SNR of 16 satellites. This information can be generated by sampling satellite information from Live GPS and then storing it into a text file. Such files can be generated by using the software.

The idea behind storing and then reading this positioning information from the memory every minute is to be able to simulate the position of the satellites as close to reality as possible. With the implementation of this feature, satellites' positions of any day can be simulated, provided that this data was sampled earlier.

7.3.1. Programming Mode

The device has to be switched to the programming mode before the external memory, attached to the MCU, can be programmed. This device goes into this mode itself when user selects the appropriate text file and starts the programming from the interface. All the other device processes are halted, while this operation is performed.

When the programming process is started from the interface, it sends an '&' character to the MCU through the UART to signal the start of the programming mode. On receiving the '&' character, the MCU sends an acknowledgement that the device is in the programming mode by sending back another '&' character to the control interface.

The control interface waits to receive the '&' character from the MCU to make sure that the state of the MCU was successfully changed to the programming mode. As it receives the acknowledgement, the interface gets ready to send the data to the MCU that is to be programmed in the memory.

For programming the memory, the control interface merges three rows from the text file together to form a packet of data to be sent to the MCU. As the MCU completes the receiving of a packet of data containing positioning information (246 bytes), it starts programming it in the external memory through the SPI. When the programming of the received data in the memory is successfully completed, the MCU signals the completion of data to the control interface by sending a '%' character.

The control interface waits for the completion of programming of row signal, i.e. the '%'character, from the MCU before sending the next packet of data. This process is repeated until all the rows are successfully programmed. When the programming of all the rows is successfully completed, the MCU transmits a '!' character to mark the end of the programming to the interface and switches itself back to the normal mode. This whole programming cycle can also be seen in Fig. 11 Programming Mode Activity Diagram.

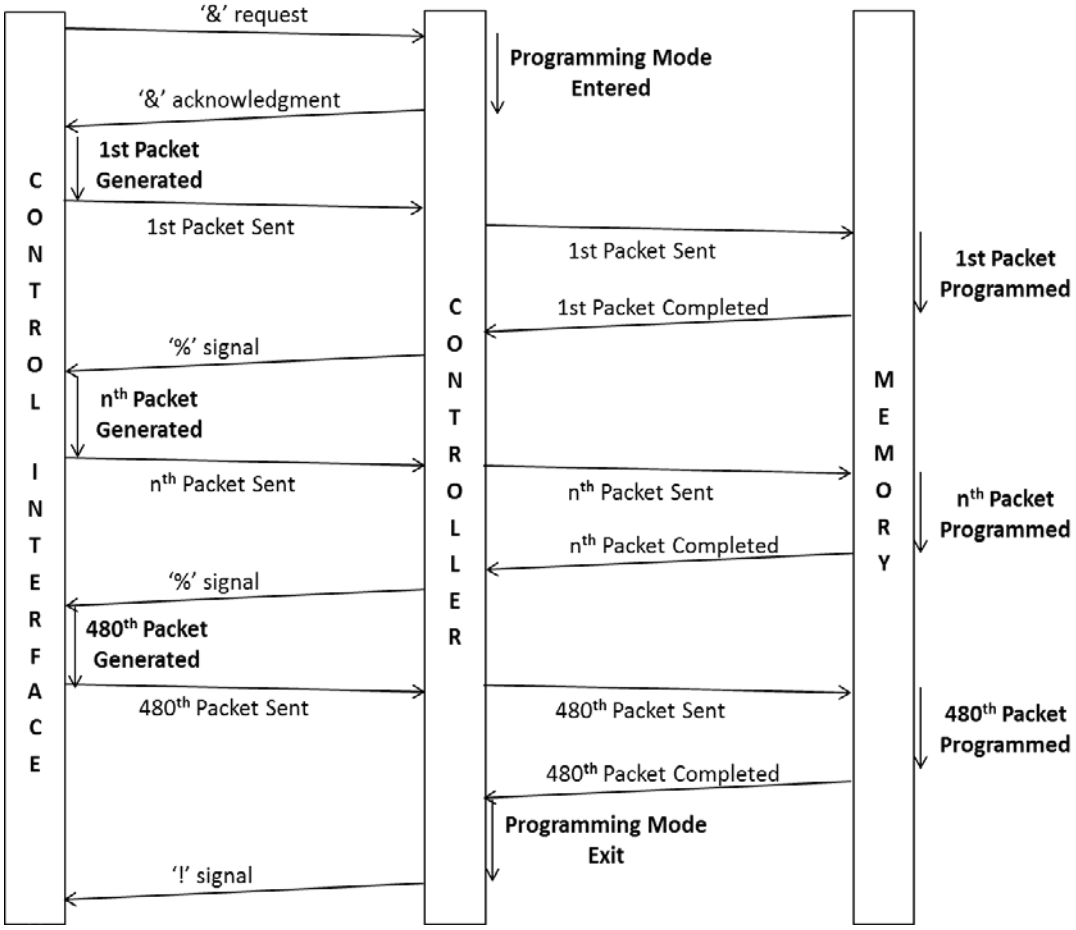

Fig. 11 Programming Mode Activity Diagram

The architectural diagram for the External Memory Handler, that programs and reads from the memory, is given in Fig. 12. The details about the functionality of each block are also given in the coming sections.

7.3.2. Instruction Decoder

For the MCU to understand that the data being received from the control interface is the satellite positioning information that is to be stored in the memory, an instruction based on the similar format, explained in Section 7.2.3, is reserved. The format of this instruction is as follows:

#	24	Packet No (3 chars)	Satellite Information for 16 satellites from three rows containing IDs, Elevation Angle, Bearing Angle, SNR (246 bytes)

The packet number informs the MCU about the number of the packet that is being sent, which in turn helps in finding the page number where the current packet should be programmed.

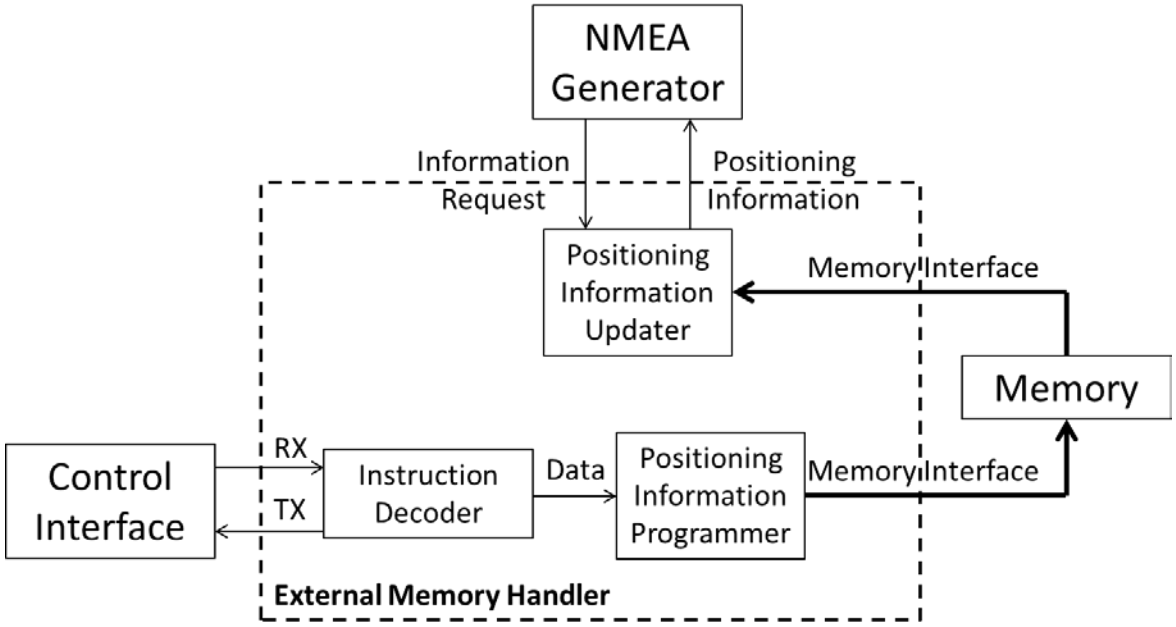

Fig. 12 External Memory Handler Block Diagram

7.3.3. Positioning Information Programmer

The main purpose of this block is to receive the data from the control interface, store it into a buffer, and to keep track of the number of characters received in the cycle of programming a packet of data. It also keeps track of the number of packets that have been programmed.

The positioning information of satellites for one day is organized in 1440 rows. Each row represents the position of satellites for one minute. This is done to get an almost exact simulation of the positions of the satellites for the course of a whole day.

Since the positioning data for one minute, i.e. one row, consists of 82 bytes (in compressed form) and a page in the memory is 264 bytes in size, three rows of compressed data is appended together to form one packet (246 bytes), in order to program a whole page in one iteration. This avoids the overhead of the op-code and address to program the data in the memory which is required for each program operation. With this strategy, a total number of 480 blocks of data are programmed in the memory, in a fastest way possible.

When a whole packet of data (246 bytes) is received from the interface, the buffer is ready to be programmed into the memory and a ready flag is set in the MCU to select the memory. However, before beginning the programming of the memory by going through the steps detailed in Section 6.3.1, the memory status is checked to make sure it is not busy doing any operations. If the memory happens to be busy, the MCU waits till the memory finishes its current operations.

The address of the page in which the current packet is programmed depends upon the current number of the packet being programmed. The address of the memory buffer is always 0 since the data is written from the start of the buffer for programming. The details about the addressing bits can be found in Section 6.3. After the addressing bits, all the bytes saved in the MCU buffer are programmed sequentially in the memory buffer through the SPI. When the whole packet of data is successfully programmed into the memory, a '%' character is sent to the control interface to notify the completion of the programming of the packet. This also acts as a signal to the control interface that the MCU is ready to receive the next packet of data and program the memory.

7.3.4. Positioning Information Updater

This block is responsible to read the satellite positioning information from the external memory. It has to be made sure that the correct chunk of data is read from the memory, based on the time of the day (number of minutes passed in a day). There are 1440 rows of satellite information on the memory, each row for a minute (one day consists of 1440 minutes).

The data reader has to update the satellite information every minute by reading the information from the memory for the corresponding minute. Since, each page consists of 3 rows of data; calculation of the exact starting address in a page, from where the page should be read, is required. The location of the data in the memory is based upon two parameters: page number and starting address. While designing the system, care was taken to store the information in a way so that it could also be retrieved easily. The decided optimal solution was to store the three rows of data in one memory page. This resulted in each page consisting of three rows and each row consists of 82 bytes of data. Therefore, the page number, where the required data is to be retrieved from, can easily be calculated by dividing the number of minutes by rows of data stored in one page i.e. 3. The address from where the reading of the data should start is calculated by getting the remainder after dividing the number of minutes passed by 3 and then multiplying it with 82. Both the formulae are listed as follows:

$$Page\ number = \frac{number\ of\ minutes\ passed}{3}$$

$$Address\ offset = (number\ of\ minutes\ passed\ \%\ 3) * 82$$

% signifies remainder when divided by 3

These two formulae make sure that the correct chunk of data is calculated from the memory. Once the correct addressing information is calculated, the read operation from the memory is performed by going through the information detailed in Section 6.3.2. The fetched data is then updated in the satellite information structure which is then utilized by the Sentence Generator in NMEA Generator (See Section 7.2.5).

8. Results

The whole system is extensively tested so that it responds appropriately under different operating conditions. Not only was the whole system, but the individual components were tested separately as well. The results of each component will be detailed in this section.

8.1. 1PPS signal Generator Result

The results obtained from the implementation came out to be quite good. These measurements were made using time interval analyzer (Hewlett Packard, E1725B). The phase noise was around 360ps rms. The phase deviation (Fig. 13) and frequency deviation (Fig. 14) charts are given.

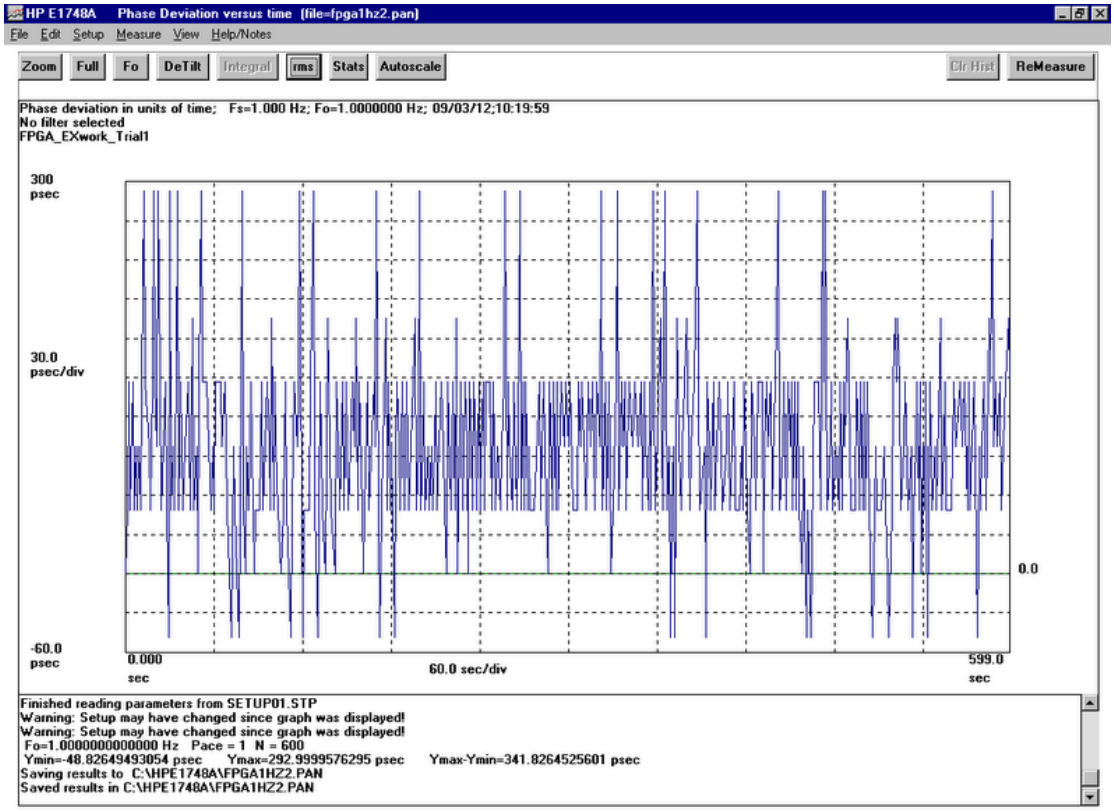

Fig. 13 Phase Deviation Chart

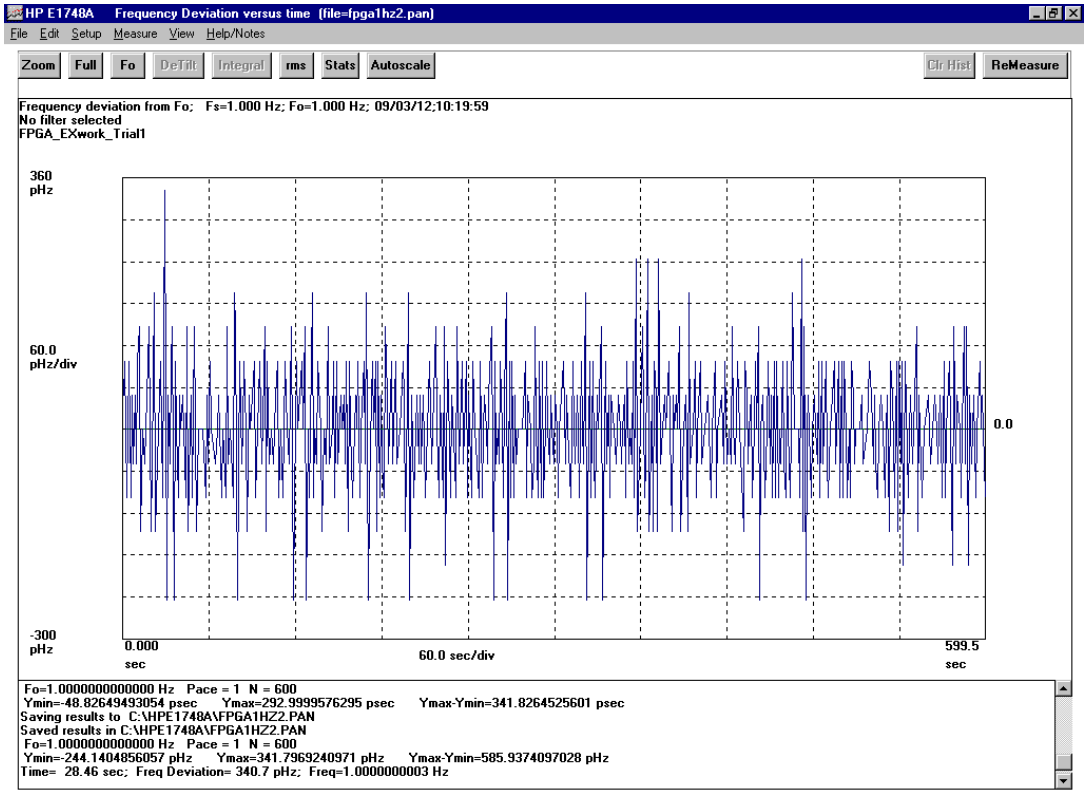

Fig. 14 Frequency Deviation Chart

8.2. NMEA Generator Results

The NMEA Generator is implemented on an ATmega 644P MCU with Atmel STK 500 development kit. The design was thoroughly tested and different scenarios were considered. The emulator was configured with different parameters and the generated sentences were sent serially through UART. The sentences were received and displayed on PC using an open source tool SSCOM, for debugging and testing. The following figures show different situations. Fig. 15 - Fig. 18 show the sentence generation under Furuno protocol while Fig. 19 – Fig. 21 show sentence generation for Ericsson protocol.

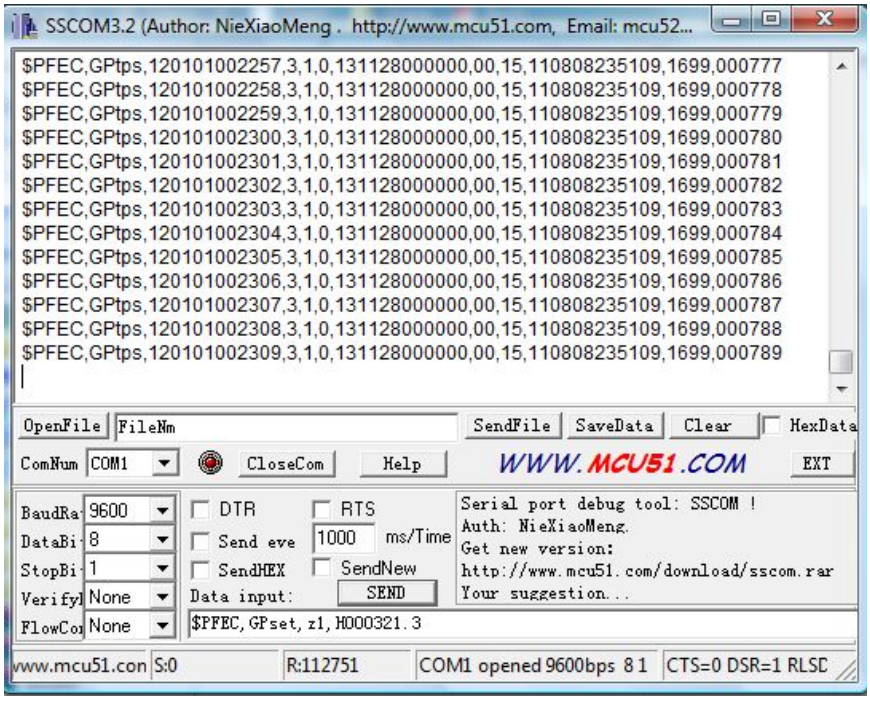

Fig. 15 GPtps Generation per second

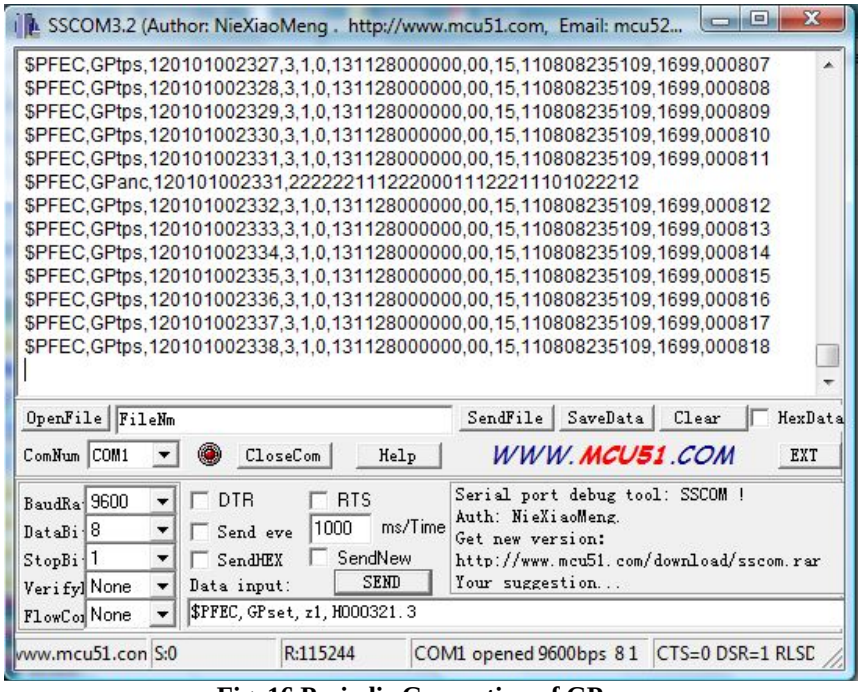

Fig. 16 Periodic Generation of GPanc

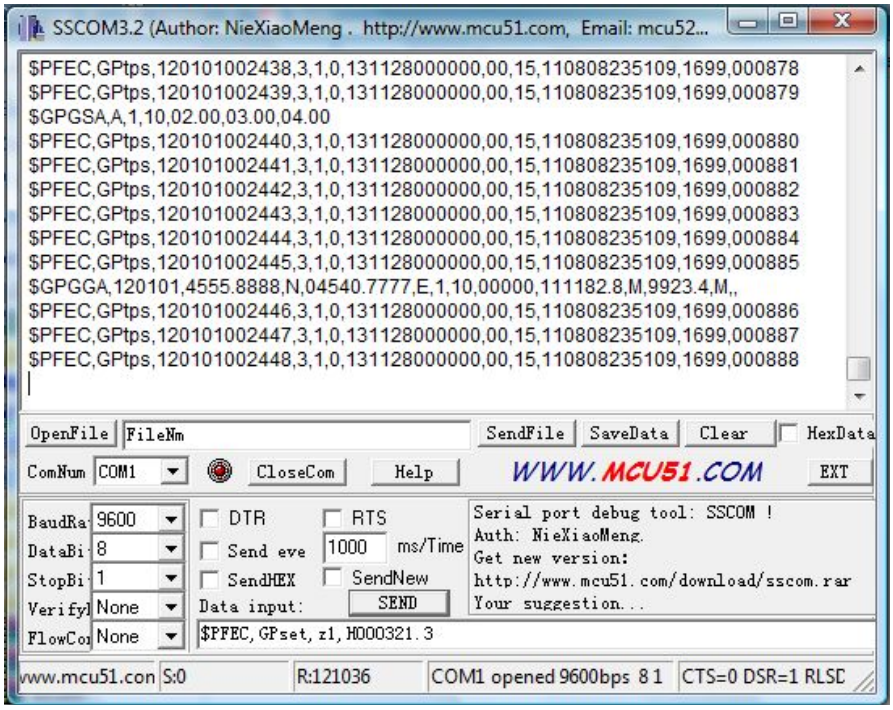

Fig. 17 Periodic Generation of GPGSA and GPGGA

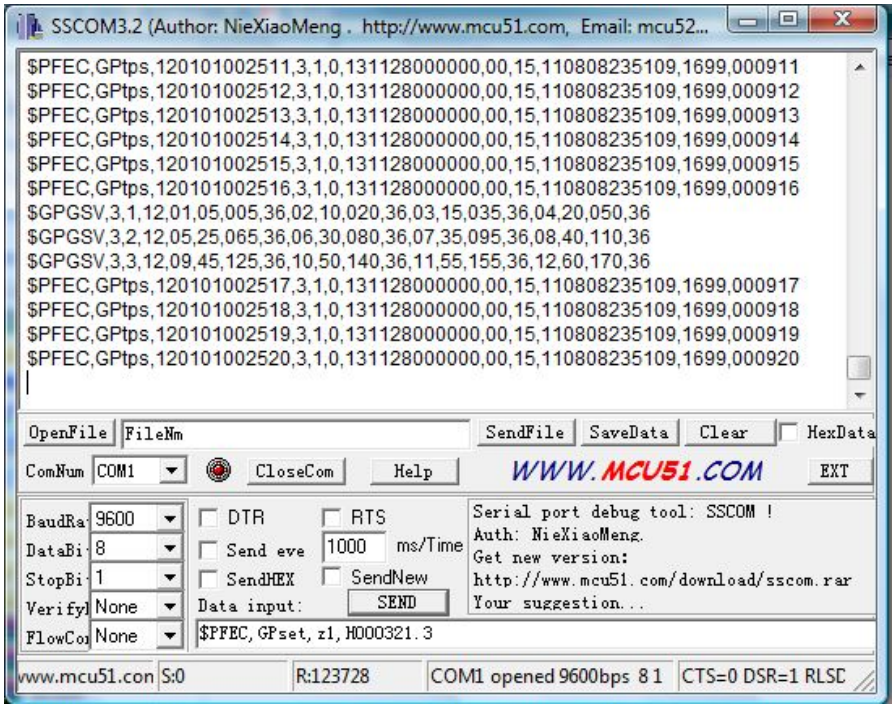

Fig. 18 Periodic Generation of GPGSV

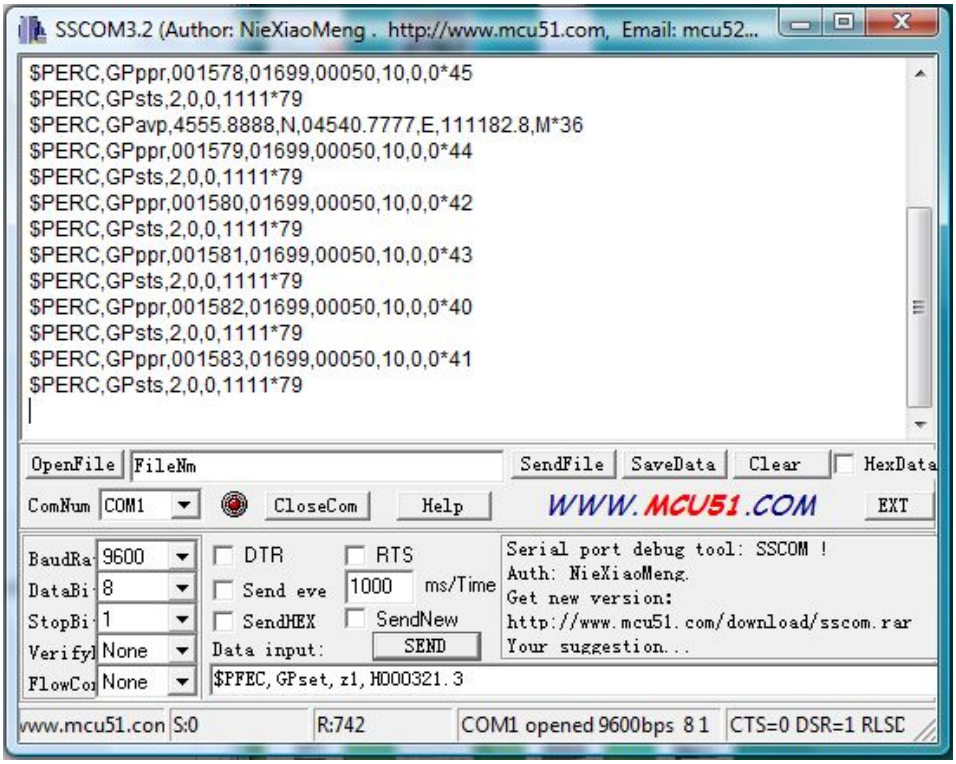

Fig. 19 Generation of GPppr and GPsts per second

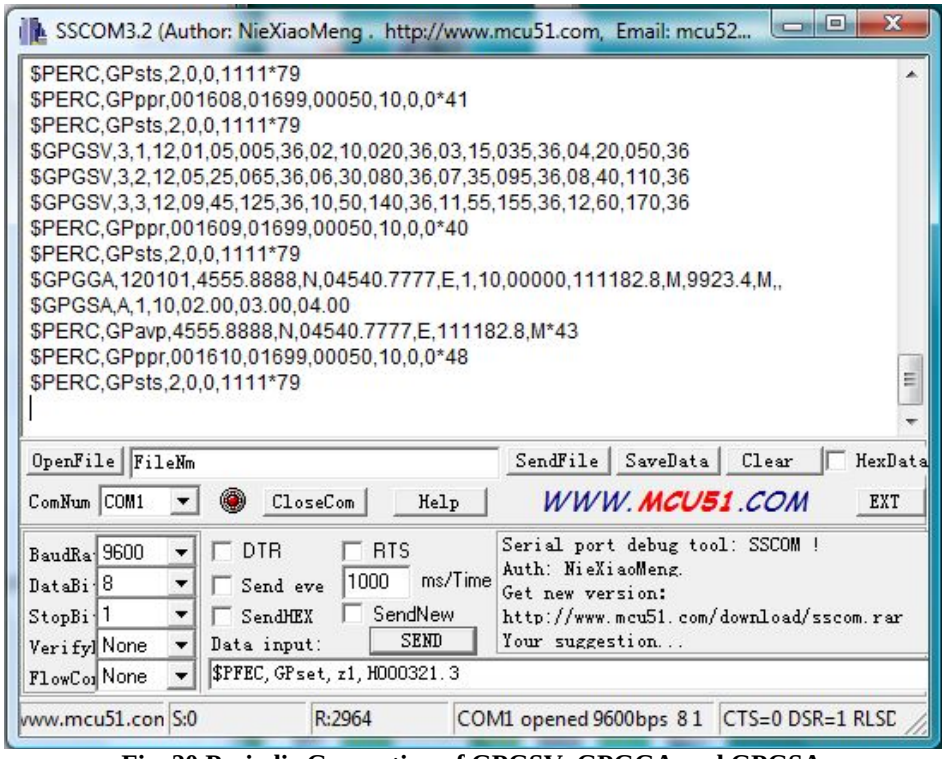

Fig. 20 Periodic Generation of GPGSV, GPGGA and GPGSA

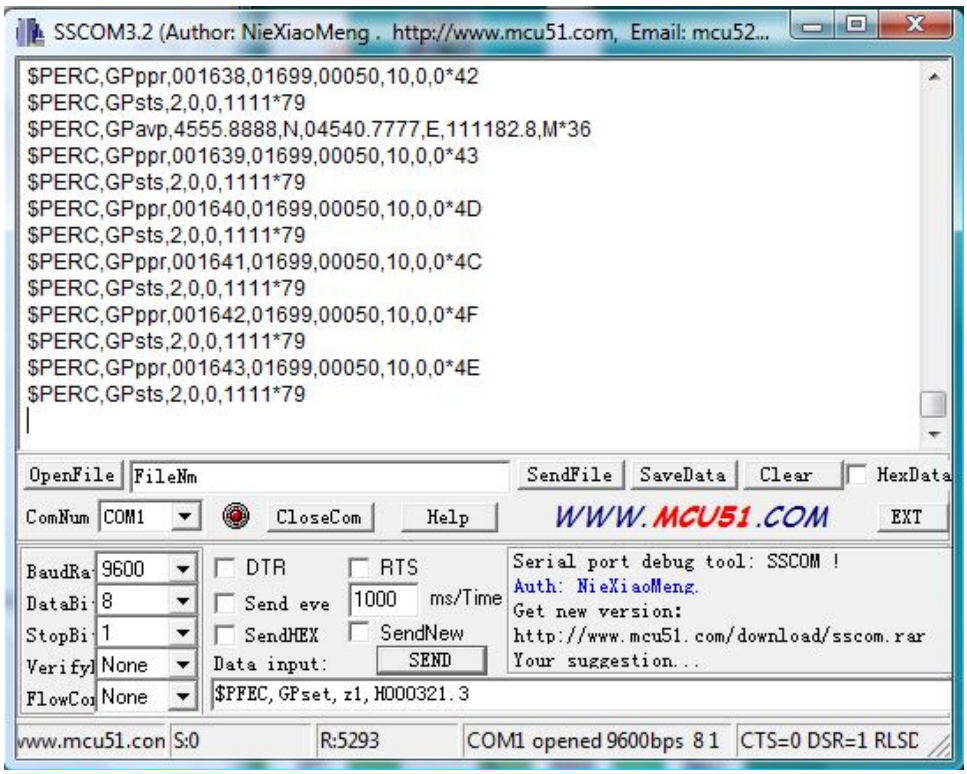

Fig. 21 Periodic Generation of GPavp sentence

8.3. NMEA Data Emulator's Results

The results for the whole system tested together are given in Fig. 22. The figure was captured after the system was run successfully for 90 hours. The tool shown in the figure is an Ericsson's internal tool developed in VEE Pro to monitor the data coming from the GPS receivers. The NMEA data simulator was connected to this tool along with the RBS, to monitor the Data Emulator's output.

The main tab at the top displays the NMEA sentences being received from the emulator. The errors tab on the right displays any errors that may occur during the run time. The polar plot on the bottom right, under the tab named Polaris, is the polar plot for the position of satellites for the duration of 24 hours. The black lines on the bottom right show the signal to noise ratio of the satellites.

The plot under the Curves tab shows the status of the 1PPS pulse, the available satellites and the tracking satellites for positioning. The red lines show the generation of 1PPS pulse, the blue lines show the available satellites varying from 8 to 12 and the green lines show the tracking satellites, varying from 5 to 8.

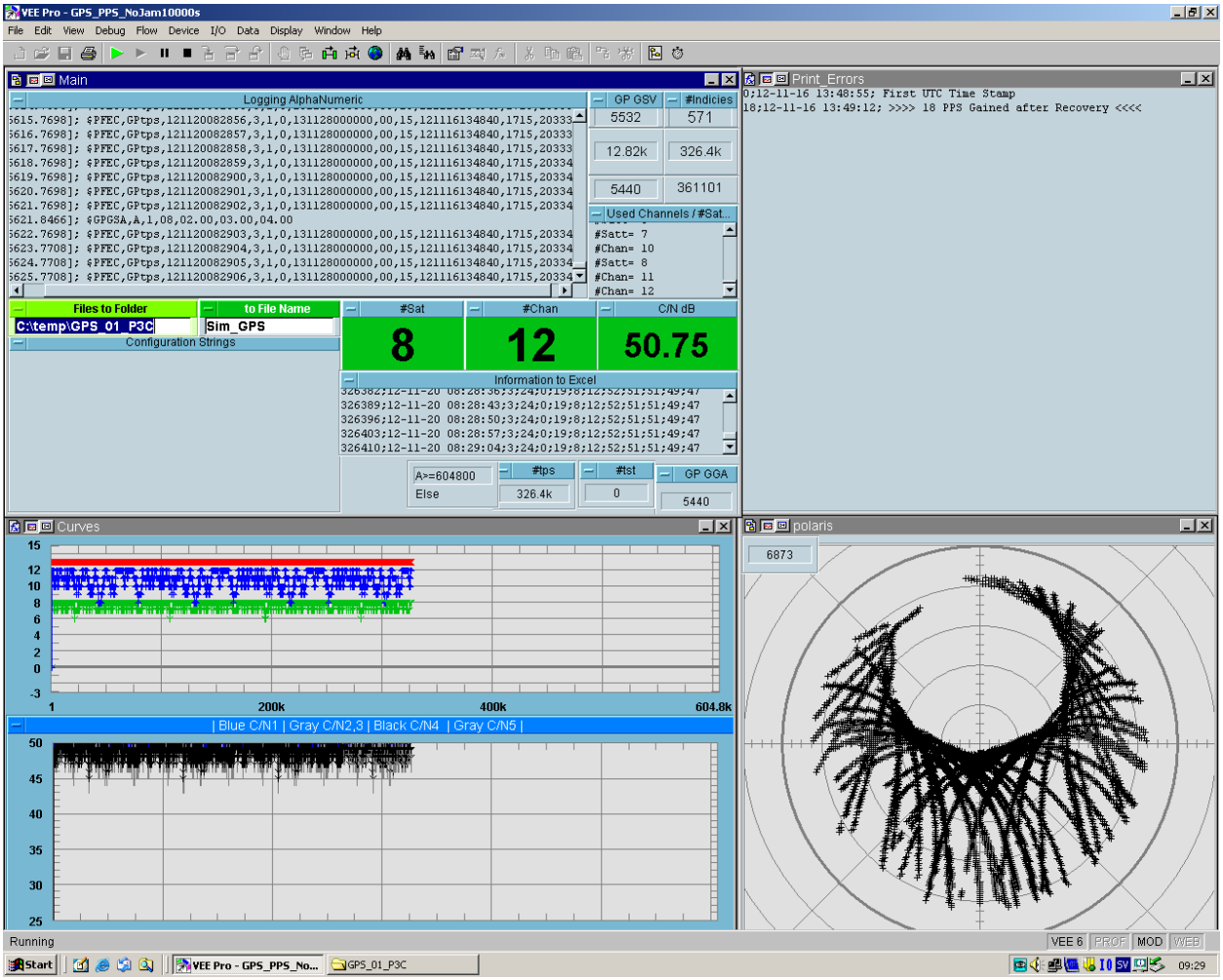

Fig. 22 NMEA Data Emulator Results

9. Conclusion & Future Work

The goals set at the start of the thesis project were achieved successfully. The developed prototype, based on the development kits, was tested with the RBS in the lab environment for a period of one week, with both Ericsson and Furuno protocols.

The results achieved are quite good and the RBS can be synchronized successfully using the hardware. Also, the emulator can be configured with the desired parameters to simulate any conditions, using the control interface on the PC. The interfaced external memory can also be programmed successfully and the satellite information can be retrieved from the memory for a whole day to give a full view of satellite positions for a whole day. The 1PPS signal generated by the emulator was also tested and it showed very good results with the stability of 360ps rms.

There is still some room to further develop and improve this design. Although all the important sentences necessary for the synchronization of the RBS are generated in this design, this work can further be enhanced by implementing other NMEA sentences that are generated by a GPS receiver.

For the generation of 1PPS, the CPLD requires a clock source which is a square wave. However, the clock signal available from the equipment in the Ericsson VERA lab is a sinusoidal. Another improvement for the device will be to implement a Sin to Square Wave converter which can be fed to the CPLD as a source.

This system was developed as a prototype on the development boards. Further steps can also include the development of a single board on which all the three main hardware components (MCU, CPLD and Memory) can be connected.

10. Bibliography

[1] "Global Positioning System," [Online]. Available: http://en.wikipedia.org/wiki/Global_Positioning_System. [Accessed 23 September 2012].

[2] P. MumFord, "Relative timing characteristics of the one pulse per second (1PPS) output pulse of three GPS receivers," in *SatNav 2003*, Melbourne, 2003.

[3] "Common View GPS Time Transfer," NIST, [Online]. Available: http://tf.nist.gov/time/commonviewgps.htm. [Accessed 12 October 2012].

[4] avangardo.com, "GPS Generator PRO," [Online]. Available: http://avangardo.com/files/gpsgen/gpsgen_manual.pdf. [Accessed September 2012].

[5] "Virtual GPS - GPS Simulator," Zyl Soft, [Online]. Available: http://www.zylsoft.com/vgps.htm. [Accessed September 2012].

[6] Spirent, "STR4500 Multi-Channel GPS/SBAS Simulator," Spirent, [Online]. Available: http://www.spirentfederal.com/GPS/Products/STR4500/Overview/. [Accessed September 2012].

[7] Spectracomp, "What is a GPS simulator," [Online]. Available: http://www.spectracomcorp.com/ProductsServices/TestandMeasurement/GPSSimulators/WhatisaGPSSimulator/tabid/1501/Default.aspx. [Accessed September 2012].

[8] "LabSat," [Online]. Available: http://www.labsat.co.uk/index.php/products/labsat. [Accessed September 2012].

[9] H. Berns and . R. J. Wilkes, "GPS Time Synchronization System for K2K," in *11th IEEE NPSS Real Time Conference (RT99)*, Santa Fe, New Mexico, 1999.

[10] A. Pasztor and D. Veitch, "PC Based Precision Timing Without GPS," in *Proceedings of the International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS 2002*, Marina Del Rey, California, 2002.

[11] K. Roos, F. D. L. Garza, J. Frenzel and D. Edwards, "Low-cost Pulse-per-Second (PPS) Signal Generator for Autonomous Underwater Vehicles," in *Proceedings of the 17th International Symposium on Unmanned Untethered Submersible Technology (UUST'11)*, Portsmouth, New Hampshire, 2011.

[12] Tronico, "The NMEA 0183 Protocol," [Online]. Available: http://www.tronico.fi/OH6NT/docs/NMEA0183.pdf. [Accessed August 2012].

[13] Ericsson, "IWD L2 L3 GPS Digital Interface".

[14] Altera, "MAX II Development Board Data Sheet," [Online]. Available: http://www.altera.com/literature/ds/ds_maxII_develop_board.pdf. [Accessed July 2012].

[15] Atmel, "STK500 User Guide," [Online]. Available: http://www.atmel.com/Images/doc1925.pdf. [Accessed July 2012].

[16] Atmel, "ATmega164P/324P/644P Complete," [Online]. Available: http://www.atmel.com/Images/doc8011.pdf. [Accessed September 2012].

[17] Atmel, "AT45DB011D Complete," [Online]. Available: http://www.atmel.com/Images/doc3639.pdf. [Accessed October 2012].

[18] Atmel, "ATmega162 (V) Complete," July 2009. [Online]. Available: http://www.atmel.com/Images/doc2513.pdf. [Accessed 10 July 2012].

[19] "Serial Peripheral Interface Bus," Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus. [Accessed October 2012].

[20] R. Black, "Zero-error 1 second Timer," [Online]. Available: http://www.romanblack.com/one_sec.htm. [Accessed November 2012].

[21] Wikipedia, "Bresenham's line algorithm," [Online]. Available: http://en.wikipedia.org/wiki/Bresenham's_line_algorithm. [Accessed November 2012].

[22] Wikipedia, "Rubidium Standard," [Online]. Available: http://en.wikipedia.org/wiki/Rubidium_standard. [Accessed October 2012].

[23] [Online]. Available: http://www.cplusplus.com/reference/clibrary/cstdio/sprintf/. [Accessed November 2012].

[24] "NMEA 0183 Standard for Interfacing Marine Electronic Devices Version 3.01," NMEA, 2002.

[25] "GPS - NMEA sentence information," [Online]. Available: http://aprs.gids.nl/nmea/. [Accessed October 2012].

Appendix A

UART Configuration Code

```
void uartInit(void)
{
        // Calculating Baud Rate i.e. 9600 bps
        uint16_t bauddiv = ((F_CPU+(baudrate*8L))/(baudrate*16L)-1);
        // Setting the baud rate
        UBRR0L = bauddiv;
        UBRR0H = bauddiv>>8;
        // Setting 1 stop bit and 0 parity bits
        UCSR0C = (3<<UCSZ00);
        // UART transmitter, receiver and Receive interrupt enabled
        UCSR0B = (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0);
}
```

Appendix B

C code for GPtps Generator

```
void GPtps_Gen(uint8_t gen){
        // Calculate current UTC
        utc_conv();
        // Update Message Length
        msgLength +=  76;
        // Merge parameters to form a sentence
        if (gen == 1)             // if it's the first sentence of the message to be sent
                sprintf(string, "$PFEC,GPtps,%s,3,1,%d,%s,00,15,%s,%04d,%04d%02d\r\n", utc,
                GPtpsmode, leap, almanac, GPSWeek, GPSSecond1, GPSSecond);
        else
                sprintf(string, "%s$PFEC,GPtps,%s,3,1,%d,%s,00,15,%s,%04d,%04d%02d\r\n",
                string, utc, GPtpsmode, leap, almanac, GPSWeek, GPSSecond1, GPSSecond);
}
```

Appendix C

C code for GPGSV Generator

```
void GPGSV_Gen(uint8_t gen){
```

```
        // GPGSV msg Generator based on the total number of visible satellites
        if (GPGSVmode == 1){                    // Satellite Data from memory
                // Merging the parameters based on the number of position tracking satellites
                for (int i = 1; i <= ((TotalSat%4) == 0 ? (TotalSat/4) : ((TotalSat/4) + 1)); i++){
                        if (i == 1 && gen == 1)
                                sprintf(string, "$GPGSV,%d,%d,%02d", ((TotalSat%4) == 0 ?
                                (TotalSat/4) : ((TotalSat/4) + 1)), i, TotalSat);
                        else
                                sprintf(string, "%s$GPGSV,%d,%d,%02d", string, ((TotalSat%4)
                                == 0 ? (TotalSat/4) : ((TotalSat/4) + 1)), i, TotalSat);
                        // Updating the message length
                        msgLength += 13;

                        for (int j = 0; j <= (((TotalSat - ((i-1) * 4)) < 4) ? (TotalSat - ((i-1) * 4)) :
                        4)-1; j++){
                        sprintf(string, "%s,%02d,%02d,%03d,%02d", string, satellite_memory[(i-
                        1)*4 + j].id, satellite_memory[(i-1)*4 + j].EleAngle, satellite_memory[(i-
                        1)*4 + j].BearAngle, satellite_memory[(i-1)*4 + j].SNR);
                                msgLength += 13;
                        }
                        sprintf(string,"%s\r\n", string);
                                msgLength += 2;
                }
        }
        else
        {       // Static Satellite Data
                // Merging the parameters based on the number of position tracking satellites
                for (int i = 1; i <= ((TotalSat%4) == 0 ? (TotalSat/4) : ((TotalSat/4) + 1)); i++){
                        if (i == 1 && gen == 1)
                                sprintf(string, "$GPGSV,%d,%d,%02d", ((TotalSat%4) == 0 ? (
                                TotalSat/4) : ((TotalSat/4) + 1)), i, TotalSat);
                        else
                                sprintf(string, "%s$GPGSV,%d,%d,%02d", string, ((TotalSat%4)
                                == 0 ? (TotalSat/4) : ((TotalSat/4) + 1)), i, TotalSat);
                        msgLength += 13;
                        for (int j = 0; j <= (((TotalSat - ((i-1) * 4)) < 4) ? (TotalSat - ((i-1) * 4)) :
                        4)-1; j++){
                                sprintf(string, "%s,%02d,%02d,%03d,%02d", string, satellite[(i-
                                1)*4 + j].id, satellite[(i-1)*4 + j].EleAngle, satellite[(i-1)*4 +
                                j].BearAngle, satellite[(i-1)*4 + j].SNR);
                                msgLength += 13;
                        }
                        sprintf(string,"%s\r\n", string);
                                msgLength += 2;
                }
        }
}
```