MASTER THESIS

TITLE: GPS time synchronization through IEEE-1588

AUTHOR: Oriol Pallarés Valls

DEGREE: Master on Electronical Engineering

DIRECTOR: Doc. Jaroslav Roztocil

DEPARTMENT: Measurement

DATE: January 2011
TITLE: GPS time synchronization through IEEE-1588

SURNAME: Pallarés Valls  NAME: Oriol

DEGREE: Master on Electronical Engineering

SPECIALITY: Electronics

DIRECTOR: Doc. Jaroslav Roztocil

DEPARTMENT: Measurement

THESIS QUALIFICATION

TUTOR

PRESENTATION DATE:
Clock synchronization of seismometers is a key point to find the exact location of the earthquake. When seismometers are installed at seafloor observatories, data is collected through a marine cable and by using the Ethernet protocol. In IEEE-1588 protocol, instruments are synchronized through an Ethernet connection. They exchange timing data with a master clock through Ethernet in order to synchronize the clocks with an error below a microsecond. Therefore, instruments MUST be able to receive timing data from an external master clock.

We have developed a Stelaris Luminary LM3S9B96 to be able to implement the clock synchronization through IEEE-1588.

Furthermore this board is able to get synchronized through PTPd protocol also will supply different outputs, to implement data time stamping for the different types of GPS. The principal framing type of frame generated is TSIP (TSIP: Trimble Standard Interface Protocol) and trigger (PPS) from an external clock, due to it would be tested in a real seismometer, Taurus from Nanometrics. In a normal Taurus, GPS receiver inside Taurus provides this information. But on the seafloor it’s impossible to get signal for a GPS so on this system must be used for getting the correct time stamping for data acquirement.

It has also been improved to be able to work with other GPS receivers, just for making it interoperable with other systems such as NMEA GPS systems and IRIG-B.
keywords:

<table>
<thead>
<tr>
<th>Time Synchronization</th>
<th>GPS</th>
<th>Ethernet</th>
<th>Microcontroller</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE-1588</td>
<td>PPS</td>
<td>NMEA</td>
<td>TSIP</td>
</tr>
<tr>
<td>IRIG-J</td>
<td></td>
<td></td>
<td>Seismometer</td>
</tr>
</tbody>
</table>
SUMMARY

CHAPTER 1.- INTRODUCTION AND BACKGROUND ........................................................................................................... 6
  1.1. ORIGIN AND MOTIVATION BEHIND THE PROJECT ................................................................................................. 6
  1.2. PROJECT OBJECTIVE .................................................................................................................................................. 7
  1.3. METHODOLOGY .......................................................................................................................................................... 9

CHAPTER 2.- EMBEDDED SYSTEMS ..................................................................................................................................... 10
  2.1. STELLARIS LUMINARY DK.LM3S9B96 ....................................................................................................................... 10
  2.2. GPIO PORT EXPANSION .............................................................................................................................................. 11

CHAPTER 3.- GPS TIMING .................................................................................................................................................... 13
  3.1. GPS ................................................................................................................................................................................ 13
  3.2. USE OF GPS ON SYNCHRONIZATION ......................................................................................................................... 15
  3.3. TYPES OF GPS FRAMES ............................................................................................................................................. 17
    3.3.1. NMEA FRAMING ..................................................................................................................................................... 17
    3.3.1.1. NMEA hardware connection ............................................................................................................................. 18
    3.3.1.2. NMEA sentences ................................................................................................................................................. 19
    3.3.2. TSIP FRAMING ......................................................................................................................................................... 19
    3.3.2.1. TSIP Hardware connection .................................................................................................................................... 20
    3.3.2.2. TSIP Sentences ...................................................................................................................................................... 20
    3.3.3. IRIG FRAMING ......................................................................................................................................................... 22
    3.3.3.1. IRIG-B hardware connection .............................................................................................................................. 23
    3.3.3.2. IRIG-B Sentences ................................................................................................................................................... 23

CHAPTER 4.- TIME SYNCHRONIZATION ............................................................................................................................ 26
  4.1. 1588 PROTOCOL .......................................................................................................................................................... 26

CHAPTER 5.- SOFTWARE ................................................................................................................................................... 32
  5.1. LABVIEW ENVIRONMENT .......................................................................................................................................... 32
    5.1.1. WHAT IS LABVIEW .................................................................................................................................................. 32
    5.1.2. WHY LABVIEW ......................................................................................................................................................... 33
    5.1.3. LABVIEW FIRST TEST SYSTEM .......................................................................................................................... 33
  5.2. C PROGRAMMING ENVIRONMENT ............................................................................................................................ 37
    5.2.1. WHAT IS C ................................................................................................................................................................. 37
    5.2.2. WHY C LANGUAGE .................................................................................................................................................. 37
    5.2.3. 1588 CODE GENERATION ...................................................................................................................................... 37
      5.2.3.1. IEEE 1588 with TSIP ........................................................................................................................................... 45
      5.2.3.2. IEEE 1588 with NMEA ..................................................................................................................................... 47
      5.2.3.3. IEEE 1588 with IRIG-B .................................................................................................................................... 50

CHAPTER 6.- TEST ................................................................................................................................................................. 55
  6.1. TESTED SYSTEMS ......................................................................................................................................................... 55

CHAPTER 7.- FUTURE IMPROVEMENTS ........................................................................................................................... 62

CHAPTER 8.- CONCLUSIONS .............................................................................................................................................. 63

CHAPTER 9.- REFERENCE .................................................................................................................................................... 64
ANNEX

ANNEX A: TSIP DATA FRAME

ANNEX B: LANTIME M600/MRS/PTP DATASHEET

ANNEX C: IRIG-B DATA FRAME
Chapter 1. – Introduction and background

“In seismology, the time of the signal acquisition is highly important in order to know the magnitude and location of the earthquake. This project presents a solution to the synchronization of seismometers with GPS time stamping, in zones without signal, such us the seafloor.”

1.1. Origin and motivation behind the project:

Along these years of university studies on Telecommunication’s world and getting specialized on electronics scope, I realized that these studies, what at first seemed to be completely independents, are actually an interlocking set only by the good connection between the applications made, which are useful to us in technology.

This project represents what have been described previously, and so on it was a good choice for researching on my master thesis, and also is an application that is currently studied by the engineering scope.

The idea appeared on SARTI’s group (Vilanova I la Geltrú – Spain) where I was working on practices, where an oceanographic observatory is being developed. After having some data acquisition systems, such as CTD system, subaquatic camera, hydrophones, etc. Appeared the necessity of adding a seismometer to the observatory, this way it could be tested on a known sea zone, and so on, ones tested, it could be used in other inaccessible zones in order to detect earthquakes that can origin tsunamis, and be able to save lives with this system due to it can alert with precision the time when a tsunami can arrive to the coast.

The system that was bought on SARTI’s group was the Trillium 120P/PA Broadband seismometer together with Taurus data logger.

Fig.1.1 Trillium 120P/PA Seismometer
And the goal to achieve was modify this closed system for being able to work without a GPS antenna connected directly to it, and without losing main characteristics.

### 1.2. Project objective:

Each engineer should have the ability to design any application on his own scope of work. Consequently, my personal goal is to achieve this ability and have a small concept of developing a whole project for my future work life.

Since the term of developing and delivering this project is not so long as it should be, some guidelines and objectives has been defined, because if not, this work would be so long for finishing on the period established by the university.

As described above, the overall objective of the project carried out in the past, the present and future supposedly, is designing the needed software and hardware for synchronize a system, which is supposed to be synchronized by GPS time stamping, to be done externally sending the correct time stamping to that systems without the necessity of having GPS signal coverage.

So on the principal objectives to reach on this project are:

- Stellaris Luminary LM3S9B96 evaluation board study and test of all the parts, which are going to be used on the project.
- Make a study of the different types of GPS framing.
- Develop the program that has to run on Stelaris board.
- Test the system on a real seismometer.
- Test the different types of GPS, comparing master and slave PPS generators.
Chapter 1.- Introduction and background

GPS Time synchronization through IEEE-1588

The study of the Stelaris board is necessary for being able to use the different systems of it correctly, being capable of configuring them and programming on an optimum way for the correct behavior of the whole structure. Also, on the part of testing on a real seismometer, we must understand which parts of the seismometer have to be modified in order to achieve the same behavior than before all the modifications. And which hardware must to be designed to be able to use the system as described.

It is also necessary to develop a program able to synchronize the Stelaris board with IEEE-1588 protocol and then generate the synchronization frames to the different types of GPS to being able to get the correct time stamp on different data acquisition systems. The structure of the whole system should be the next:

![Fig.1.3 System behavior](image)

Where we have a first station with the antenna and the Master clock, and a second station where we have the Data acquisition system, which can be separated kilometers or can have hundreds of switch between the first station and the second, without losing the time precision given by the GPS antenna on the first station because of IEEE-1588 implementation till the Stelaris board, where GPS type of frames will be generated and this way we can synchronize all DAQ systems that works with external time synchronization.

One case of this type of synchronization system, and the reason of starting this project, is underwater seismometers, where you need a net of seismometers just for being able to determine where an earthquake has started and the propagation of it. So it’s needed time precision on each seismometer to know the wave propagation of the earthquake. Underwater it’s impossible to get synchronized with a GPS signal because of there is no signal, so on it is needed some system able to send time stamps to each seismometer and put them on the same time to work as is needed. This system is what is described above and what it is going to be developed on the project, for different types of GPS, due to making it interoperable with other seismometers or data acquisition systems, which can have different framing GPS synchronization.
The guideline followed for carrying through this project has been defined before starting it, just for knowing with certainty what steps must be followed for optimize the time elapsed on the development of it.

First of all is necessary to send our seismometer to Nanometrics for getting the necessary modifications for receiving the data that the internal GPS sends to the seismometer, through an external input/output. While the seismometer is receiving its modifications, Luminary board test can starts; on this step all the systems that have to be used on the project must be tested. After having the required modifications to our seismometer and the board tested, it is possible to start to operate with IEEE-1588 protocol on that board.

The PTPd protocol code, for C compiler, already is created so on I can use the libraries that had been used by other people on other projects and port to my board.

Ones reached the good behavior of the PTPd on Stelaris board, it’s necessary to make a study of the different type of GPS frames and how they have to be sent, this way, PTPd protocol program can be modified to incorporate on it the different framing to be sent, and getting the systems synchronized.

First is going to generate the framing of one type of GPS with LabView, because it is easier to program with this environment and having a first version of the program on PC and knowing if the seismometer gets synchronized, before starting the modifications on the board program.

Once Taurus seismometer is synchronized with the PC it will be possible to start the modifications of one type of GPS faming on the board program and test it again to achieve to get the seismometer synchronized, if this time synchronization is so good as expected, it would be possible to generate the other GPS framing types and test them together, to realize if the system precision change with the incorporation of the other sentences on the program.
Chapter 2. – Embedded systems

"An electronic equipment, that has been reduced both as physical level and in terms of components, in order to be adapted on a determinate environment, and for specific objectives, is called ‘Embedded system’.

2.1. Stellaris Luminary DK-LM3S9B96:

In order to realize the implementation of IEEE-1588 and generate GPS framing properly, is necessary a development platform in with a microcontroller and the necessary modules, to being able to process all the information and receive/send this information to each system that performs synchronization platform.

For mentioned operations, have been chosen the Development Kit of Luminary “LM3S9B96”, this is a feature-rich development platform for Ethernet, USB OTG/Host/Device, and CAN enabled Stellaris ARM® Cortex™-M3-based microcontrollers.

The main reason of choosing this kit is the PTPd-v1 compatibility, and the possibility of communicating PTPd against Ethernet, and generating all the necessary framing through GPIO pins.

![Fig.2.1 DK-LM3S9B96](image)

For the correct development of this application the principal parts that are going to be used on this board are the microcontroller ARM® Cortex™-M3-based, Ethernet peripheral, GPIO pins, Bright 3.5” QVGA LCD touch-screen display and USB interface.
Ethernet peripheral is going to be used for the PTP communication, GPIO pins for PPS (Pulse Per Second) generation and GPS framing, touch-screen for debugging and printing the time and GPS that is being used, and the USB interface for debugging and Launching the program on the board. On the next figure can be seen the distribution of the peripherals on the board.

![Fig.2.2 DK-LM3S9B96 Features](image)

### 2.2. GPIO Port expansion:

In order to generate three types of framing and PPS output has been necessary to expand these ports, this have been done with EPI Signal Breakout, which provides access to 31 GPIO pins additionally to the board ones.

![Fig.2.3 EPI Signal Breakout](image)

This expansion board is added on the center of the Development kit board just for getting this GPIO expansion available on the ARM3 microcontroller, this way we can use GPIO pins of the
development board for PPS generating, and the expansion for framing generation of the three kinds of GPS.

Two GPS framing are going to be generated on PC7 (Port C 7), this two frames are generated through UART output, so this way we can configure a General Purpose Output for UART and change the UART characteristics for working on one type (TSIP) or the other (NMEA). While the other GPS output frame is going to be generated on PG0, this is a binary modulated output so it isn’t possible to be generated by UART and this is the reason why is necessary another output.
Chapter 3. – GPS Timing

“In order to establish the time, with high precision and repeatability, although we are on different places on the Earth, is needed a system available all over the world, and this system is GPS (Global Positioning System) with its atomic clocks. “

3.1 GPS:

The “Global Positioning System” allows the user to determine the position on a determinate time of a specific object through a satellite communication, with a precision of centimeters for space and few nanoseconds for time. This system was developed and installed by the USA defense department as an evolution of system SPUTNIK, developed by the Soviet Union at 1957, which worked through the observation of the Doppler effect. USA army, applied this system quickly to its navigation system, and improving the atomic clocks, got synchronization, with a time reference for all of its satellites. At 1973, mixing both technologies of USA army and USA defense department, the second one providing a codification transmission method, which consist on a precision data codification using a modulation with a noise pseudo-random code PRN, appeared the named Navigation Technology Program, lately named NAVSTAR GPS. From 1978 till 1985 was developed, and were put into orbit 11 NAVSTAR satellites. Later, were added new generations of satellites, till 1993 when the constellation of satellites, achieved 24 operative satellites. This constellation ensures a minimum coverage of three satellites all over the world. From 1993 until 2006, the number of satellites has been increased unto 32, which we have nowadays. This is the number of satellites for what the GPS system was developed to work, this way is ensured the correct positioning and time reference in all situations on the Earth.

Fig.3.1 NAVSTAR satellite
This satellite network, formed different orbits at 20,200Km around the Earth, and working synchronously, covers the whole earth surface.

GPS technology has been extended not only in military field; it has been extended on civil one. It is a really useful tool, which provides to the user his position and GMT time from a GPS receiver that extracts the information that has been send from satellites. From this received signals and its delays, is extracted, by triangulation, the position where the receiver is and the GMT time with high accuracy.

There are a lot of GPS, and each one can provide different information, so for this project has been developed all types of framing of GPS in order to be able to connect all brand of GPS, and this way allow all systems to be synchronized with this application.

Time precision on GPS is a critical characteristic on GPS receivers, and this is the reason why the drift between the satellite and the GPS receiver must be treated and deleted, for guarantee that GPS receiver is going to have the same time than the satellite, and not adding any delay on the communication between them.

One of the most significant error sources in the GPS receiver’s clock. Because of the very large value of the speed of light, c, the estimated distances from the GPS receiver to satellites, the pseudo ranges, are very sensitive to errors in the GPS receiver clock; for example an error of one microsecond correspond to an error of 300 meters. This suggests that an extremely accurate and expensive clock is required for the GPS receiver to work. Because manufacturers prefer to build inexpensive GPS receivers for mass markets, the solution for this dilemma is based on the way sphere surfaces intersect in the GPS problem.

Is likely that the surfaces of the three spheres intersect, because the circle of intersection of the first two spheres is normally quite large, and thus the third sphere surface is likely to intersect this large circle. It is very unlikely that the surface of the sphere corresponding to the fourth satellite will intersect either of the two points of the intersection of the first three, because any clock error could cause it to miss intersecting a point. However, the distance from the valid estimate of GPS receiver position to the surface of the sphere corresponding to the fourth satellite can be used to compute a clock correction.

*Fig.3.2 Sphere surfaces*
Let \( r_4 = (t_r - t_4)c \) which is the distance from the valid estimate of GPS receiver position to the fourth satellite, let \( p_4 \) denote the pseudo range of the fourth satellite. Let \( da = r_4 - p_4 \), which is the distance from the computed GPS receiver position to the surface of the sphere corresponding to the fourth satellite. Thus the quotient, \( \frac{da}{c} \), provides an estimate of GPS receiver’s clock bias: \( b = \frac{t_r}{t_r} - t_r \) where \( t_r \) is the time indicated by the receiver’s on-board clock and \( t_r \) is the correct reception time. The GPS receiver clock can be advanced if \( b \) is positive or delayed if \( b \) is negative.

### 3.2. Use of GPS on synchronization:

Using GPS technology to provide a low-cost precision reference clock has application in numerous distributed computer and telecom systems. PC clocks in particular are notoriously inaccurate due to variations in crystal tolerance, calibration and temperature effects resulting in clock drift of many seconds per day. For time-critical applications such as globally-linked financial systems, mobile base-station networks, and power switching equipment, clock error can have catastrophic results.

Here is where GPS satellite synchronization provides a convenient and cost-effective solution for clock accuracy down to 15 billionths of a second.

GPS synchronization enables measurement systems to make synchronized measurements over extremely large areas. GPS provides a method of sharing timing signals without the need to run timing cables to each measurement system. Another benefit to GPS is that your data is always time stamped to a global time standard which allows you to associate your data with sets from other systems which are also synchronized to GPS with a high level of confidence.
Some applications which can take advantage of the benefits of GPS include structural monitoring, electrical power grid monitoring, and ground vibration monitoring.

**Fig.3.4 GPS synchronized measurement systems**

GPS allows a lot of flexibility in the overall system architecture. Each of the nodes does not need to be identical. For example, in many applications the sensors are not equally distributed over the test area. This means that the measurement devices also need to have a similar physical distribution. With GPS synchronization, one node may contain only 16 channels while another has 400 channels. If a specific location requires many channels, synchronization system can be synchronized together through a cabled and GPS synchronized system.

**Fig.3.5 Hybrid measurement system synchronization**
3.3. Types of GPS frames:

GPS receiver has different types of output frames, this frames contain different information each one, such as position, GMT time, altitude, velocity, direction, number of satellites and error checksum. The most common types of GPS frames are NMEA (National Marine Electronics Association), TSIP (Trimble Standard Interface Protocol) and IRIG (Inter-Range Instrumentation Group time codes). With all of these frames we can synchronize a data acquisition system, and the proposal of this project is to be able to generate this three kind of frames and being able to synchronize all the data acquisition systems that are supposed to be synchronized through a GPS with the Luminary board. Using this board we will be able to supply time stamp, because it will be synchronized through an Ethernet cable which is connected to a GPS receiver in an accessible zone for signal satellite coverage.

3.3.1. NMEA framing:

The National Marine Electronics Association (NMEA) has developed a specification that defines the interface between various pieces of marine electronic equipment. The standard permits marine electronics to send information to computers and to other marine equipment.

GPS receiver communication is defined within this specification. Most computer programs that provide real time position information understand and expect data to be in NMEA format. This data includes the complete PVT (Position Velocity Time) solution computed by GPS receiver. The idea of NMEA is to send a line of data called a sentence that is totally self-contained and independent from other sentences. There are standard sentences for each device category and there is also the ability to define proprietary sentences for use by the individual company.

All of the standard sentences have a two letter prefix that defines the device that uses that sentence type. (For GPS receivers the prefix is GP.) This followed by a three letter sequence that define the sentence contents. In addition NMEA permits hardware manufactures to define their own proprietary sentences for whatever purpose they see fit. All proprietary sentences begin with the letter P and are followed with 3 letters that identifies the manufacturer controlling that sentence. For example a Garmin sentence would start with PGRM and Magellan would begin with PMGN.

Each sentence begins with a ‘$’ and ends with a carriage return/line feed sequence and can be no longer than 80 characters of visible text (plus the line terminators). The data is contained within this single line with data items separated by commas. The data itself is just ascii text and may extend over multiple sentences in certain specialized instances but is normally fully
contained in the message. For example time might be indicated to decimal parts of a second or location may be show with 3 or even 4 digits after the decimal point. Programs that read the data should only use the commas to determine the field boundaries and not depend on column positions. There is a provision for a checksum at the end of each sentence which may or may not be checked by the unit that reads the data. The checksum field consist of a ‘*’ and two hex digits representing an 8 bit exclusive OR of all characters, but not including, the ‘$’ and ‘*’. A checksum is required on some sentences.

3.3.1.1NMEA Hardware connection:

NMEA hardware interface is designed for meet the RS-232 requirements, making this standard compatible with most computer serial ports using RS232 protocol, however strictly speaking the NMEA standard is not RS232. The standard recommends EIA-422. It’s usual to have the possibility to configure the interface speed on some models, but the NMEA standard is 4800 b/s with 8 bits of data, no parity, and one stop bit. As can be seen a baud rate of 4800 b/s, you can easily send enough data to more than fill a full second of time. For this reason some models send data every two seconds, some sentences can be sent only during a particular action of the receiver such as while following a route while other receivers may always send the sentence and just null out the values.

At 4800 b/s you can only send 480 characters in one second. Since an NMEA sentence can be as long as 82 characters you can be limited to less than 6 different sentences. The actual limit is determined by the specific sentences used, but this shows that it is easy to overrun the capabilities if you want rapid sentence response. NMEA is designed to run as process in the background spitting out sentences which are then captured as needed by the using program. Some programs cannot do this and these programs will sample the data stream then use the data for screen display, and then sample the data again.

In order to use hardware interface is needed a cable. Generally the cable is unique to the hardware model so depending on the brand of the GPS this cable will be different. Some of the latest computers no longer include a serial port but only a USB port. Most GPS receivers will work with serial to USB adapters and serial ports attached via pcmcia (pc card) adapter. For general NMEA use with a GPS receiver it will only be needed a two wire cable, one wire for data out, and the other one for ground. If it’s needed send Waypoint or some information to the receiver then will be necessary to add a wire for data in.

Fig.3.6 GPS serial communication
3.3.1.2 NMEA Sentences:

NMEA sentences are composed by a header, and then all the information separated by commas, there is a wide range of NMEA frames, each one with its own information, such as position, number of satellites, altitude, velocity, time... But for developing this project only time information is needed, so let’s focus on this type of frame.

The frame that is going to be used provides information of Date and Time. The header of this frame will have the start command common on all NMEA frames ‘$’, after that ‘GP’ characters, giving information about is a common NMEA frame, instead of a brand or personal NMEA frame. So the frame will be like this:

\[ \$GPZDA,hhmmss.ss,dd,mm,yyyy,xx,yy*CC \]
\[ \$GPZDA,201530.00,04,07,2002,00,00*60 \]

Where:

- hhmmss: HrMinSec (UTC)
- dd, mm, yyyy: Day, Month, Year
- xx: local zone hours -13..13
- yy: local zone minutes 0..59
- *CC: checksum

3.3.2. TSIP framing:

The Trimble Standard Interface Protocol (TSIP) is the native language for the Lassen LP GPS. TSIP is a binary language with a wide variety of commands and reports. TSIP reports can be output automatically, or they can be output as responses to queries. The format of the automatic reports can be easily configured. The receiver is factory configured for single precision Latitude-Longitude-Altitude. Report formats can be modified and position and velocity formats can be customized. The GPS unit sends the information it has gathered via the serial port. The output language is the format the information is received in.

This language is the most versatile, offering a large list of options that can be tacked onto the output. These options can be configured in TSIPCHAT, which will be discussed later. However, while its versatility is commendable, its format is also very hard to understand.
3.3.2.1. TSIP Hardware connection:

TSIP hardware interface, like NMEA interface, meets with RS-232 requirements. However the standard it’s not designed for working exclusively with RS-232 protocol. Each serial communication port is bi-directional, data characteristics, protocol definition, and other options are user programmable, although it’s usual to find this interface with a baud rate of 9600 bits/s, odd parity, 8 data bits and one stop bit.

The TSIP protocol is based on the transmission of packets of information between user equipment and the time source unit. Each packet includes an identification code (1 byte, representing 2 hexadecimal digits) that identifies the meaning and format of the data that follows. Each packet begins and ends with control characters.

3.3.2.2. TSIP Sentences:

TSIP packet structure is the same for both commands and reports. The packet format is:

<DLE> <id> <data string bytes> <DLE> <ETX>

Where:

- <DLE> is the byte 0x10
- <ETX> is the byte 0x03
- <id> is a packet identifier byte, which can have any value excepting <ETX> and <DLE>.

The bytes in the data string can have any value. To prevent confusion with the frame sequences <DLE> <ID> and <DLE> <ETX>, every <DLE> byte in the data string is preceded by an extra <DLE> byte (‘stuffing’). These extra <DLE> bytes must be added before sending a packet and removed after receiving the packet. Notice that a simple <DLE> <EXT> sequence does not necessarily signify the end of the packet, as these can be bytes in the middle of a data string. The end of a packet is <ETX> preceded by an odd number of <DLE> bytes.

As mentioned before, TSIP protocol has more than 20 commands, so only are going to be explained each ones which are going to be used on this project.

The only frames necessary for time synchronization are, the frame with 0x41 identifier, which is an automatic output packet with the information of GPS time, and the frame with the identifier 0x46, which gives information with the health of the receiver. Only with this two frames is possible to get the time synchronization of every TSIP input frame GPS.

Report packet 0x41 (GPS Time) this packet provides the current GPS time of week and the week number. The GPS receiver sends this packet in response to packet 0x41 and during an automatic packets update cycle. Update cycles occur approximately every 5 seconds. The data format is shown below.
Table 3.1 Report packet 0x41 Data formats

GPS week number runs from 0 to 1023 and the cycles back to week #0. Week #0 began January 6, 1980. The first cycle back to week #0 was on August 22, 1999. The extended GPS week number however, does not cycle back to 0. For example: the week # for August 22, 1999 = 1024.

The seconds count begins with “0” each Sunday morning at midnight GPS time. A negative indicated time-of-week indicates that time is not yet known; in that case, the packet is sent only on request. The following table shows the relationship between the information in packet 0x41 and the packet 0x46 status codes.

<table>
<thead>
<tr>
<th>Approximate time accuracy</th>
<th>Time source</th>
<th>Sign (TOW)</th>
<th>Packet 46 status code</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No time at all</td>
<td>-</td>
<td>0x01</td>
</tr>
<tr>
<td>unknown</td>
<td>Approximate time from real-time clock or packet 2E</td>
<td>+</td>
<td>0x01</td>
</tr>
<tr>
<td>20-50 msec + clock drift</td>
<td>Time from satellite</td>
<td>+</td>
<td>0x02 – 0x0C</td>
</tr>
<tr>
<td>Full accuracy</td>
<td>Time from GPS solution</td>
<td>+</td>
<td>0x00</td>
</tr>
</tbody>
</table>

Table 3.2 Packets 0x41 and 0x46 Status code relationships

Report packet 0x46 (Health of receiver), provides information about the satellite tracking status and the operational health of the receiver. This packet is sent every power on or software-initiated resets, in response to packet 0x26 and, every five seconds.
GPS Timing

GPS Time synchronization through IEEE-1588

<table>
<thead>
<tr>
<th>Byte</th>
<th>Bit</th>
<th>Item</th>
<th>Type</th>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>Status Code</td>
<td>UINT8</td>
<td>0x00</td>
<td>Doing position fixes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x01</td>
<td>Don’t have GPS time yet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x02</td>
<td>Need initialization</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x03</td>
<td>PDOP is too high</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x08</td>
<td>No usable satellites</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x09</td>
<td>Only 1 usable satellite</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x0A</td>
<td>Only 2 usable satellites</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x0B</td>
<td>Only 3 usable satellites</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x0C</td>
<td>The chosen satellite is unusable</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Battery Backup</td>
<td>Bit</td>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>BBRAM was not available at start-up</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>Antenna feedline fault</td>
<td>Bit</td>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Short or open detected</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Type of fault</td>
<td>Bit</td>
<td>0</td>
<td>Open detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>Short detected</td>
</tr>
</tbody>
</table>

**Table 3.3 Packet 0x46 Data Format**

### 3.3.3. IRIG framing:

The IRIG time codes were originally developed by the Inter-Range Instrumentation Group (IRIG), part of the Range Commanders Council (RCC) of the USA Army. The standard was first published in 1960 and has been revised several times by the Telecommunications and Timing Group (TTG) of the RCC. The latest version is IRIG standard 200-04, “IRIG Time Code Formats”.

Most common IRIG time code is IRIG-B, which allows devices across the world to synchronize with a common time source to a resolution of one millisecond. Although the IRIG-B time code is the best known, exists other six code formats, with different data transfer rate as shown on the next table.

<table>
<thead>
<tr>
<th>Format</th>
<th>Pulse Rate</th>
<th>Index Count interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRIG-A</td>
<td>1000 PPS (Pulse Per Second)</td>
<td>1 ms</td>
</tr>
<tr>
<td>IRIG-B</td>
<td>100 PPS</td>
<td>10 ms</td>
</tr>
<tr>
<td>IRIG-D</td>
<td>1 PPM</td>
<td>1 minute</td>
</tr>
<tr>
<td>IRIG-E</td>
<td>10 PPS</td>
<td>100 ms</td>
</tr>
<tr>
<td>IRIG-G</td>
<td>10000 PPS</td>
<td>0.1 ms</td>
</tr>
<tr>
<td>IRIG-H</td>
<td>1 PPS</td>
<td>1 second</td>
</tr>
</tbody>
</table>

**Table 3.4 IRIG Time code formats**
GPS Timing

The IRIG-B time protocol is widely used by electric utilities, industrials, and others to ensure precise time synchronization of power system devices.

3.3.3.1. IRIG-B Hardware connection:

All IRIG time code formats use pulse width coding, and IRIG-b time code uses a period of 10 ms with a duty cycle of 80% for Position identifier, of 50% for a logical ‘1’ and 20% for logical ‘0’. Each frame is a 100 bits sentence, so it will elapse 1 second per frame. This communication is a TTL-level signal over coaxial cable or shielded twisted pair cable.

3.2.3.2. IRIG-B Sentences:

IRIG time code sentences can be:

- Unmodulated (DC level shift, no carrier signal)
- Modulated (amplitude-modulated, sine wave carrier).
- Modified Manchester (amplitude-modulated, square wave carrier).

Three types of coded expressions are used in the IRIG standard:

- Binary Coded Decimal time-of-year (BCDTOY) and year (BCDYEAR)
- Control Functions (CF), set of bits reserved for user applications
- Straight Binary Seconds (SBS) time-of-day (0 to 86400 seconds)

In addition to the letter used to designate one of the six code formats, signal identification numbers are used to further describe specific characteristics. Thus, the complete IRIG time code designation consists of a letter and three digits, as shown below.

![Fig.3.7. IRIG time codes](image-url)
GPS Timing

GPS Time synchronization through IEEE-1588

<table>
<thead>
<tr>
<th>1st Digit</th>
<th>Modulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unmodulated – DC Level Shift (DCLS) pulse width coded</td>
</tr>
<tr>
<td>1</td>
<td>Amplitude modulated, sine wave carrier</td>
</tr>
<tr>
<td>2</td>
<td>Manchester modulated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2nd Digit</th>
<th>Carrier Frequency / Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No carrier (DCLS)</td>
</tr>
<tr>
<td>1</td>
<td>100 Hz / 10 ms resolution</td>
</tr>
<tr>
<td>2</td>
<td>1 kHz / 1 ms resolution</td>
</tr>
<tr>
<td>3</td>
<td>10 KHz / 100 microsecond resolution</td>
</tr>
<tr>
<td>4</td>
<td>100 KHz / 10 microsecond resolution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3rd Digit</th>
<th>Coded Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$BCD_{TOY}$, CF, SBS</td>
</tr>
<tr>
<td>1</td>
<td>$BCD_{TOY}$, CF</td>
</tr>
<tr>
<td>2</td>
<td>$BCD_{TOY}$</td>
</tr>
<tr>
<td>3</td>
<td>$BCD_{TOY}$, SBS</td>
</tr>
<tr>
<td>4</td>
<td>$BCD_{TOY}$, $BCD_{Year}$, CF, SBS</td>
</tr>
<tr>
<td>5</td>
<td>$BCD_{TOY}$, $BCD_{Year}$, CF</td>
</tr>
<tr>
<td>6</td>
<td>$BCD_{TOY}$, $BCD_{Year}$</td>
</tr>
<tr>
<td>7</td>
<td>$BCD_{TOY}$, $BCD_{Year}$, SBS</td>
</tr>
</tbody>
</table>

Table 3.5 IRIG Signal identification numbers

IRIG-B is typically distributed as DC level shift, pulse width coded signal (“unmodulated IRIG-B”) or as an amplitude-modulated signal based on a sine carrier wave with a frequency of 1 KHz.

![Fig.3.8. IRIG-B time code Standard](source: IRIG Standard 200-04)
As can be seen is sent first 7 bits with seconds information, after these ones 7 more bits with the minutes, the 6 bits for hours, 10 bits for the day of the year and 9 bits for the year after each multiple of 10 year, this means that if we are on 2012, this field must be filled with ‘2’, 2010 + 2 = 2012. Every sentence is composed by 100 bits, which means that the complete sentence takes 1 second to be sent.

As mentioned before, IRIG-B uses reference markers, called position identifiers. The presence of two consecutive reference markers signifies the start of the time frame. The first reference marker alerts that the next rising edge will be PPS marker, as can be seen on the next figure, at the same time that we see different types of modulation and codification.

Fig.3.9. Coding comparisons and PPS generation
Chapter 4. – Time synchronization

“Clock synchronization is a problem from computer science and engineering which deals with the idea that internal clocks will differ after some amount of time due to clock drift, caused by clocks counting time at slightly different rates. There are several problems that occur as a repercussion of rate different and several solutions, some being more appropriate than others in certain contexts. “

4.1. 1588 Protocol:

Precise time information is especially important for distributed systems in automation technology. With the Precision Time Protocol (PTP) described in IEEE 1588, it is possible to synchronize distributed clocks with an accuracy of less than 1 microsecond via Ethernet networks for the very first time. The demands on the local clocks and the network and computing capacity are relatively low.

We use clocks to synchronize with persons or processes. The necessary accuracy of the clock depends on the application needing. Anyone wanting to catch a train has to have his eye on the clock to within a minute. In competitive sport, a hundredth of a second can be decisive and drives in a packing machine need synchronization in the microsecond range.

Many technical systems have a sense of time. An implicit system time exists when there is no actual clock and the timing behavior is determined by processes in the hardware and software. This is often sufficient in a lot of systems. An implicit time system is implemented, for example, by regular trigger events to every user which indicate the beginning of a unit of time and then trigger the appropriate actions.

The system time is explicitly available when it is represented by a clock. This is often necessary in complex systems especially. This decouples the communication from the execution. But not every clock is exact. Now and again it has to be checked whether the deviation is tolerable and whether the clock needs to be corrected. Communication between the individual clocks is necessary for this.

Two effects are in evidence when setting or synchronizing clocks: independent clocks initially run at an offset for one thing. To synchronize them, the more inaccurate clock is set to the more accurate one (offset correction). Another thing is that real clocks do not run at exactly the same speed. Therefore, the speed of the more inaccurate clock has to be regulated constantly (drift correction).
There have previously been different ways to synchronize distributed clocks through a network. The most common of these are the Network Time Protocol (NTP) and the simpler Simple Network Time Protocol (SNTP) derived from it. These methods are widely distributed in LANs (Local Area Networks) or in the Internet and allow accuracies into the millisecond range.

Another possibility is the use of radio signals from GPS satellites. However, this necessitates relatively expensive GPS receivers in every clock as well as the appropriate antennae. This theoretically gives you high-precision clocks but the high costs and effort often prevent it.

Another solution is to send a high-precision time pulse (e.g. pulse per second signal) to every user on separate lines. However, this entails an enormous additional wiring effort.

This is where the Precision Time Protocol (PTP) described in IEEE 1588 comes in. It has been developed with the following aims:

- Synchronization accuracy in the sub-microsecond range
- Minimum requirements of the processor performance and network bandwidth which enables it to be implemented on simple and low-cost devices
- Low administration effort
- Use via Ethernet networks but also via other networks
- Specification as an international standard

The idea for PTP was born at the end of the 90s in the USA at Agilent Technologies in the field of measuring technology. The process principle developed there was submitted to the IEEE as a suggestion and created the basis for the IEEE 1588 standard. At the end of 2002 PTP was passed as a standard under the name of "1588TM - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems". In addition, PTP was also adopted as an IEC standard in May 2004 and was published under the name of IEC 61588.

PTP is arousing interest in many different applications. In automation technology, PTP is in demand wherever processes need to be synchronized exactly. Here Motion Control is an important field of application in the broadest sense. Because PTP helps to synchronize drives in a robot or a printing, packing or paper processing machine for example. Interactive robots are also connected by high-precision clocks or whole machine or plant parts are linked closely by PTP so that the processes that run can be synchronized exactly. Clocks running synchronously in every component enable distributed structures to be set up and the processes to be decoupled from the communication and processing of the control commands.

For this reason, the time synchronization according to IEEE 1588 has since become a part of almost all future real-time automation protocols. CIPsync, part of the Ethernet/IP frameworks of the ODVA, relies totally on PTP for Motion Control applications. PROFINet (PNO) uses PTP as a synchronization protocol (Transport of PTP over IEC 61158 Type 10, Annex I in IEEE1588-2008) and ETHERNET Powerlink (EPSG) will also use PTP for synchronizing real-time segments in a future version.
Also, many companies are working on the evaluation and implementation of PTP outside of automation technology. For test and measurement applications the new LXI standard (LAN extension for Instrumentation) defines methods and protocols to connect devices with Ethernet and synchronize those devices using PTP version 2. Generally, wherever measured values are detected and need to be put in relation to each other, PTP is a popular solution. In energy distribution systems, parameters such as currents and voltages are measured in distributed sensors, linked centrally and evaluated. Turbine controls use the PTP protocol to set up even more efficient plants. And, for monitoring processes, de-central detected events are marked with precise time stamps and transferred to the control station for logging and analysis. In high-frequency measuring applications PTP is used for correlating de-central detected physical variables. Geo-scientists use PTP to synchronize seismic measuring instruments over great distances and to be able to localize earthquake epicenters more exactly. In telecommunications, PTP is being considered for synchronizing networks or supplying mobile radio base stations with precise time pulses. There is also interest in time synchronization in accordance with IEEE 1588 in the fields of safety technology, digital audio / video transport, automotive technology or military applications.

In 2004 the IEEE1588-2002 standard was undergoing revision to meet the interest of additional applications like Telecom, Wireless and others. The P1588 project was started in February 2005 in the IEEE Committee with the aim of extending the IEEE1588-2002 standard. The outcome of this IEEE Committee is the new IEEE1588-2008 standard which is available since March 2008 with the following new features:

- Better accuracy to achieve sub nanosecond ranges and below
- Faster synchronization (in version 1 SYNC messages occur no faster than 1 second, with version 2 SYNC messages can occur up to several 1000 messages)
- Shorter messages to reduce network bandwidth
- New messages (Announce message, PDelay_Req, PDelay_Resp, Pdelay_Resp_Follow_Up, Signaling messages and also new Management messages)
- Introduction of one-step-mode (no follow-up messages are sent/needed)
- Introduction of Transparent Clocks (End-to-End and Peer-to-Peer) to prevent error accumulation in cascaded topologies
- Introduction of profiles (to define features and settings for different applications and markets, e.g 802.1 AVB Task Group as P802.1AS or PROFINet)
- New mappings to other transport mechanisms like DeviceNet, PROFINet, ControlNet and IEEE802.3/Ethernet (direct mapping)
- Introduction of TLVs to extend the protocol with new features and options to meet the requirements of future applications
- (Optional) Unicast messaging
- (Optional) Path Trace
- (Optional) Alternate Timescales
- (Optional) Master Cluster Tables
- (Optional) Alternate Master
Chapter 4 - Time synchronization

- Security features (experimental only in IEEE1588-2008)
- Conformance specifications
- Configuration options
- Requirements for compatibility between IEEE1588-2002 (PTPv1) and IEEE1588-2008 (PTPv2)

PTP knows different types of clocks and acts as a master to slave protocol. A clock in an end device is known as an ordinary clock, a clock in a transmission component like an Ethernet Switch is a boundary clock (BC) or transparent clock (TC). A master which is controlled ideally by a radio clock or a GPS receiver synchronizes the respective slaves connected to it.

The synchronization process is divided into two phases. First the time difference between the master and the slave is corrected, this is the offset correction.

Two modes (with IEEE1588-2008) are known for the synchronization process.

**Two-step-mode:**
In two-step-mode the master sends a synchronization message – SYNC message – with an estimated value of the time cyclically to the connected slaves. Parallel to this, the time at which the message leaves the sender is measured as precisely as possible, if possible by hardware support directly on the medium. The master then sends this actual exact transmission time of the corresponding sync message to the slaves in a second message - follow-up message. These also measure the reception time of these messages as exactly as possible and can correct the correction value (offset) to the master from it. The slave clock is then corrected by this offset. If the transmission line were to have no delay, both clocks would be synchronized.

**One-step-mode:**
The master sends a synchronization message – SYNC – message with the precise value of the time cyclically to the connected slave. Other than in two-step-mode, the precise time is inserted into the SYNC message “on-the-fly” by the hardware. No FOLLOWUP – messages are needed in this mode. The calculation of the offset is the same as in two-step-mode.

![Diagram](image.png)

Fig.4.1 Boundary clock switches work as slaves in relation to the master clock and supply the other connected slaves as a master
The second phase of the synchronization, the delay measurement, determines the run time between slave and master. This is determined by so-called Delay Request and Delay Response messages in a similar way and the clocks adjusted accordingly. This can also be achieved in one-step or in two-step mode.

Boundary clocks are required wherever there is a change of the communication technology or other network elements block the propagation of the PTP messages. Furthermore it is recommended that a Boundary clock be used wherever there is a network component that inserts significant delay fluctuation. Boundary clocks have typically more than 2 ports, with one port serving as a PTP slave port to an upstream master clock, and the other ports serving as PTP clock masters to downstream PTP clocks. So with Boundary clocks you get time distribution trees.

The Boundary clocks (BC) defined in both versions of the IEEE1588 Standard respectively Draft Standard evidence two problems when used in (highly) cascaded networks. Namely, there is nonlinear decreasing synchronization accuracy and rising resynchronization time after network reconfiguration. To eliminate these effects the concept of transparent clocks (TC) has been introduced in the IEEE 1588 standard version 2. Transparent clocks were added in IEEE1588 - 2008 to correct the “residence time” of the network device like an Ethernet Switch. The residence time is accumulated in a field (correction field) of the SYNC (one-step) or FollowUP (two-step) message. Since transparent clocks are stateless they have no impact on the reconfiguration time of e.g. ring topology networks.

The IEEE1588-2008 standard knows two types of transparent clocks, namely: End-to-End (E2E) and Peer-to-Peer (P2P).
End-to-End TCs only measure the time taken for a PTP event message (those who get time stamped) to transit the bridge and provide this information to the receiving clocks in the correction field. No propagation delay of the link connected to the port is corrected by the E2E TC. E2E TCs use the delay request / delay response mechanism for the delay measurement whereby the residence time of the delay request / delay response messages are corrected in the same way stated above.

Peer-to-Peer TCs use the peer delay mechanism for the delay measurement. In addition to providing PTP event transit time information the P2P TC also provides corrections for the propagation delay of the link connected to the port receiving the PTP event message (correction field).

The peer delay mechanism measures the port to port propagation delay time between two directly connected ports sharing the same communication technology. The peer delay mechanism is independent of the state of a port (master or slave). It operates separately in both directions of the link.
Chapter 5. – Software

“Every hardware explained previously, has to have background software controlling it efficiently. This way all the equipment connected to the platform where the program is running will be controlled and will be capable to communicate with other equipment through telecommunication techniques.”

5.1. LabVIEW environment:

5.1.1. What is LabVIEW:

LabVIEW is a graphical programing environment that uses “G Language”, in reference to Graphical – Language. Is this one of the main characteristics which makes this software interesting due to the programs aren’t a full code paper that most times are extended and hard to understand documents if aren’t programmed by yourself. With the graphical language, you have to connect different drag-and-drop blocks for make a program, like in a block diagram being.

LabVIEW born on 1986, created by National Instruments since there it has been a pioneer environment for engineers and programmers who want to create a control, measurement or test system in an efficiently way.

Each program created on LabVIEW is called “VI” (Virtual Instrument), being able to interweave different VI’s between them, or use them as libraries, on that case this VI’s are called “SubVI”.

![Fig.5.1 LabVIEW environment](image-url)
LabVIEW work environment is easy to understand and really intuitive, to start working with it, is only necessary to know some basic notions of programation. As can be seen on the figure 5.1, we have two different windows for one VI. One of these windows is frontal panel (left side with grey background) and the other one is block diagram panel (right side with white background).

Frontal panel is what is going to be seen by the user while program execution, and where the programmer design a graphical environment where the user controls the developed software.

Block diagram panel is where the software is designed and is not seen directly for the user.

### 5.1.2. Why LabVIEW:

As seen previously, LabVIEW programation environment is intuitive, light and mostly designed for develop applications with graphical environment, for working with DAQ, test or control systems.

So this way, a Virtual Instrument has been developed to work with PTP and GPS frame generation (TSIP type) on a computer, because is fast and easy with LabVIEW and will be useful to know the functionality of the system and later it will be easier to program with C language on the Luminary board.

### 5.1.3. LabVIEW first test system:

This test system is developed to get the time of the computer and with this time then synchronize a seismometer which is designed to work directly with an onboard GPS.

If we are able to synchronize the seismometer simulating the TSIP output of a GPS the we will be able to implement the other two types of frames mentioned before and program them on C language.

For this first test we have a Nanometrics seismometer with the onboard GPS deleted, and the internal connections of this onboard GPS changed to a serial input of the seismometer, in order to be accessible for the user. This way we will have inputs/outputs for communicate the PPS and the synchronization frames into and out of the seismometer.

As can be seen on the figure 5.2, on pins B, C and V we have available the GPS serial communications, and on
pin E the PPS communication.

So this first test program must be able to generate the TSIP synchronization frames due to is the format expected by the Nanometrics seismometer (TAURUS). This LabVIEW program has been designed following the Taurus onboard GPS datasheet instructions about the information shared between GPS and seismometer. The following block diagram shows the main behavior of the software:

On the next two figures we can see the implementation of this block diagram on LabVIEW, the first picture corresponds to the initialization part and the second one to the frame sending part.
Inside of each frame block there is a SubVI implementing the type of data expected for the Tarus seismometer as explained on chapter 3 on TSIP frame implementation.
Connecting the C pin of the showed connector previously to the computer, with this software running, and connecting the PPS input (Pin E) to a function generator, with a square signal with 1 Hz of frequency, the seismograph gets synchronized properly with the laptop date and all the indicators of the timing on the seismometer are lighting green which indicates that the framing and the PPS that are received is all the data that the seismometer is expecting and as conclusion, is the same data that the onboard GPS was sending to the Taurus before extracting it from the hardware.

Now is time for the implementation on C language and on the Luminary board, this way this programation will be easier to understand.
Chapter 5 - Software

GPS Time synchronization through IEEE-1588

5.2. C programming environment:

5.2.1. What is C:

Is a general-purpose computer programming language developed between 1969 and 1973 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system.

Although C was designed for implementing system software, it is also widely used for developing portable application software.

C is one of the most widely used programming languages of all time and there are very few computer architectures for which a C compiler does not exist. C has greatly influenced many other popular programming languages, most notably C++, which began as an extension to C.

5.2.2. Why C language?

Programmers around the world embrace C because it gives maximum control and efficiency to the programmer.

There are a couple of benefits you gain from working with C:

- You will be able to read and write code for a large number of platforms -- everything from microcontrollers to the most advanced scientific systems can be written in C, and many modern operating systems are written in C.
- The jump to the object oriented C++ language becomes much easier. C++ is an extension of C, and it is nearly impossible to learn C++ without learning C first.

And the first benefit is the reason on this project for working on C, because it’s possible to read and write code for every type of microcontrollers on the market.

5.2.3. 1588 code generation

IEEE 1588 as explained before synchronize the internal clock to a network master clock source, so for implementing this protocol will be necessary a TCP stack and PTP protocol software.

These two parts have been developed lots of times and there isn’t so much to improve on this part of the project because it’s easy to find it on the internet on free source, so we are going to use one of this free source code that has been ported to our board, improving some
characteristics and the adding to this code all the framing necessary to implement all the GPS frames.

For TCP stack is used lwIP (Lightweight TCP/IP), which is a small independent implementation fo the TCP/IP protocol suite.

The focus of the lwIP TSCP/IP implementation is to reduce resource usage while still having a full scale TCP. This making lwIP suitable for use in embedded systems with tens of kilobytes of free RAM and room for around 40 kilobytes of code ROM.

lwIP features:

- IP (Internet Protocol) including packet forwarding over multiple network interfaces
- ICMP (Internet Control Message Protocol) for network maintenance and debugging
- UDP (User Datagram Protocol) including experimental UDP-lite extensions
- TCP (Transmission Control Protocol) with congestion control, RTT estimation and fast recovery/fast retransmit
- Specialized raw API for enhanced performance
- Optional Berkeley-alike socket API
- DHCP (Dynamic Host Configuration Protocol)
- PPP (Point-to-Point Protocol)
- ARP (Address Resolution Protocol) for Ethernet

With this part of the software we have on the board DHCP obtaining an Ethernet address. If DHCP times out without obtaining an address, AUTOIP will be used to obtain a link-local address. The address that is selected will be shown on the QVGA display.

On the code we can see how different functions proceeding from the libraries of this code source are used for initialize the TCP/IP stack, first is initialized the DHCP for obtaining one IP direction for our physical addre

```
// Initialize the lwIP library, using DHCP.
lwIPInit(pucMACArray, 0, 0, 0, IPADDR_USE_DHCP);

// Initialize the lwIP library, using Manual.
//lwIPInit(pucMACArray, 3232235623, 4294901760, 3232235521, IPADDR_USE_STATIC);

// Setup the device locator service.
LocatorInit();
```
On this second part of code, we have the display of the IP address on the board screen. Each time that the IP changes it will be reloaded on the screen in order to be able to access to the http server.

```c
void
DisplayIPAddress(void)
{
    unsigned long ulTemp;
    char pcString[32];

    // Clear the "Waiting for IP..." string from the display.
    GrContextFontSet(&g_sContext, &g_sFontCmss24);
    GrStringDrawCentered(&g_sContext, "                          ", -1,
                         GrContextDpyWidthGet(&g_sContext) / 2, 60, true);

    // Display the new IP address.
    GrContextFontSet(&g_sContext, &g_sFontCmss12);
    usnprintf(pcString, 32, "IP %d.%d.%d.%d", g_ulLastIPAddr & 0xff,
              (g_ulLastIPAddr >> 8) & 0xff, (g_ulLastIPAddr >> 16) & 0xff,
              (g_ulLastIPAddr >> 24) & 0xff);
    GrStringDrawCentered(&g_sContext, pcString, -1,
                         GrContextDpyWidthGet(&g_sContext) / 2, 60, true);
    UARTprintf("\r%s
", pcString);

    // Display the new network mask.
    ulTemp = lwIPLocalNetMaskGet();
    usnprintf(pcString, 32, "Netmask: %d.%d.%d.%d", ulTemp & 0xff,
              (ulTemp >> 8) & 0xff, (ulTemp >> 16) & 0xff,
              (ulTemp >> 24) & 0xff);
    GrStringDrawCentered(&g_sContext, pcString, -1,
                         GrContextDpyHeightGet(&g_sContext) - 40, false);
}
```
GrContextDpyWidthGet(&g_sContext) / 2,
GrContextDpyHeightGet(&g_sContext) - 30, false);
UARTprintf("\r\n", pcString);

// Display the new gateway address.
//
ulTemp = lwIPLocalGWaddrGet();
usnprintf(pcString, 32, "Gateway: %d.%d.%d.%d",
    (ulTemp >> 8) & 0xff, (ulTemp >> 16) & 0xff,
    (ulTemp >> 24) & 0xff);
GrStringDrawCentered(&g_sContext, pcString, -1,
    GrContextDpyWidthGet(&g_sContext) / 2,
    GrContextDpyHeightGet(&g_sContext) - 20, false);
UARTprintf("\r\n", pcString);
}

With this implementation of TCP/IP stack and with all the functions of its libraries we have a
complete TCP/IP protocol implementation, and this way we will have access to the network via
an RJ 45 connector and the communication between the microcontroller and the master clock
will be possible.

For the PTP (Precision Time Protocol) implementation another open source code will be used,
this code is find on the FreeBSD foundation, this implementation is called PTPd (Precision Time
Protocol daemon) and implements the PTP as defined previously by the IEEE 1588 standard.
PTP was developed to provide very precise time coordination of LAN connected computers.
The IEEE 1588 specification dictates most of PTPd’s operation, so one should read first chapter
4 before understanding this source code. PTPd coordinates the local clock by adjusting the
effective tick rate, or slewing, the clock. This is a slow but precise process. To quickly
coordinate a local clock that is off by more than one second, PTPd will reset the clock instead
of slewing. These results in a step change in the time base, which can cause problems for
applications that, require a smooth and monotonically increasing time base.

While is fairly easy to get PTPd to provide a precisely coordinated time base, it is much more
difficult to use the coordinated time base, due to the lack of generation precisely timed events.

PTPd’s source is grouped into few components. The component delineations are based on the
functionality defined by the spec, but the delineations are not specifically defined by the spec.

The following is a block diagram of PTPd’s major components, in which occlusion indicates
interfaces between components.
Chapter 5 - Software

GPS Time synchronization through IEEE-1588

Fig.5.7. PTPd component

In general, PTPd's source is divided into platform-independent and platform-dependent code. Platform-independent code is located in the top level of the PTPd tree, and platform-dependent code is located the dep/ sub-directory.

PTPd's major source code components on this free source code are:

**Protocol Engine, protocol.c**

The main protocol state machine defined in the spec. The state machine is implemented as a forever loop with cases for each state. It is called with protocol() after start-up, and only returns on an error. Normal execution is expected to be halted externally and asynchronously. The loop sleeps in a select() call on the network sockets. The primary states, handled by doState(), are master, slave, inactive, and faulty. State transitions, handled by toState(), occur primarily due to the results of the BMC algorithm. The primary events are message receives, which are checked for in handle(), and timer expiration. The primary actions are message sends, which are done in issue(), timer resets, regular runs of the BMC algorithm, foreign master data updates, and system clock servo updates after sync message receipts.

**BMC, bmc.c**

The Best Master Clock algorithm defined by the spec. It is called with bmc(), and it returns the proper state, master or slave, based on the reported clock statistics of other PTP clocks that have sent Sync messages (foreign masters).

**Clock Servo, dep/servo.c**

The clock servo computes the offset-from-master from the master-to-slave delay and slave-to-master delays. It uses the offset-from-master to compute clock tick rate adjustments to minimize the offset-from-master. The clock servo also performs filtering to remove jitter from its input.

**Message Packer, dep/msg.c**

Gathers data into and extracts data from PTP messages, the format of which is defined by the spec.
Network Layer, dep/net.c

Initializes connections, sends, and receives data between PTP clocks. The network layer also retrieves time stamps for Event messages from the Time Stamp component.

Time Stamp

Records message send and receive times. The send and receive times are used to compute the master-to-slave and slave-to-master delay. The delays are then used to coordinate slave clocks with master clock time. Time stamps should be recorded as close to the networking hardware as possible. This minimizes jitter in the time stamps, which is the most significant detriment to PTPd’s clock coordination.

Timer, dep/timer.c

Low resolution interval timers used in the protocol engine. The timers control periodic Sync message sends by masters, Delay Request sends by slaves, periodic runs of the BMC (state change events), and Sync receive timeouts.

Start-up, dep/startup.c

Sets the program’s execution state, and retrieves run-time options from the user.

Other code

The ptpd.c file contains the execution entry point. The ptpd header files contain function prototype declarations. The data types header files contain structure and data type declarations. Finally, the constants header files contain constant declarations.

The IEEE 1588 spec does not define how the offset from master produced by a PTP slave is used to bring the slave clock into coordination with master clock time. This procedure, called clock discipline, is not trivial, and there are many possible design approaches and trade-offs.

The following shows the message send and receive paths in a typical system running PTPd, along with the associated time stamps that form the basis of the master-to-slave and slave-to-master delay measurements.
Fig.5.8. send and receive message paths

The following is a system diagram of PTPd's clock servo. The FIR filtering of the offset from master input is a simple, two-sample average. The IIR filtering of the input one-way delay is described below. The PI controller that mediates the tick rate adjustment output has the difference equation: \( y[n] = e[n]/Ap + a[n], \ a[n] = e[n]/Ai + a[n-1]. \)

Fig.5.9. Clock servo

This is the behavior of this code, but for the properly functionality some parts of the code have to be changed. On the main code there are some options of configuration of the PTPd code for improving its timing characteristics, but there is one thing that must be changed. The loopback mode is not working properly and when a message is sent, as explained on chapter 4, this time of the message sending must be saved, in order to delete the delay between the message sent and the received, and this way be able to correct the offset from master. In order to correct this error in loopback mode, has been added some code in protocol.c library to save the time when the handle sync message is sent. The reason to add the saving of this time stamp on protocol.c instead of another layer, like could be on TCP/IP layer, is because this way we consider all the way between the PTP clock servo and the master clock, on the drift calculus.
Now that we have precise time reference in our board system clock we need to generate properly the GPS timing frames to synchronize the different systems that work with this type of data.

Some of GPS synchronization systems need a PPS (Pulse Per Second) signal in order to synchronize some system. In this PTPd code is a generation of PPS implemented and we only assigned the pin B0 in order to output this PPS and be used later on these GPS synchronization implementations. This PPS generation uses the PTPd code to know every time that the second changes and the generates the rising edge with a high priority interruption. For implementing the falling edge, there is the nanosecond counter, that makes the PPS signal fall to ‘0’ when this counter achieve half of a second, so that means that we will have a really high precision on these PPS generation as we will see on the next chapter.

```c
// Update internal time and set PPS output, if needed.
//
g_ulSystemTimeNanoSeconds += SYSTICKNS;
if(g_ulSystemTimeNanoSeconds >= 1000000000)
{
    ROM_GPIOPinWrite(PPS_GPIO_BASE, PPS_GPIO_PIN, PPS_GPIO_PIN);
    g_ulSystemTimeNanoSeconds -= 1000000000;
    g_ulSystemTimeSeconds += 1;
    HWREGBITW(&g_ulFlags, FLAG_PPSOUT) = 1;
}
```

This code generates the rising edge, and the following the falling edge:

```c
// Clear PPS output when needed and display time of day.
//
if(HWREGBITW(&g_ulFlags, FLAG_PPSOFF))
{
// Negate the PPS output.
//
    ROM_GPIOPinWrite(PPS_GPIO_BASE, PPS_GPIO_PIN, 0);
//
// Indicate that the PPS output has been negated.
//
    HWREGBITW(&g_ulFlags, FLAG_PPSOFF) = 0;
    counter++;
With PTPd code we achieve time synchronization to our board, that will be situated close to the system to synchronize, so now only is needed the synchronization code. First of all is generated TSIP code, because it was implemented with LabVIEW first and was easier to start working with this protocol. This protocol as explained on chapter 3 is a serial communication data output, so is needed assign to one GPIO pin the serial communication corresponding to this protocol.

TSIP has a baud rate of 9600 bits/s, 8 data bits, odd parity, 1 stop bit and no control flux. On the next code is seen the implementation of this configuration on C code and the initialization of GPIO pin:

```c
// //initialize the UART for Taurus Frame output. //
GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_6 | GPIO_PIN_7);
GPIOPinConfigure(GPIO_PC6_U1RX);
GPIOPinConfigure(GPIO_PC7_U1TX);
ROM_GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_6 | GPIO_PIN_7);
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
```

Here has been configured pin C6 for serial receiving and C7 for transmission.

```c
UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,
(UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
UART_CONFIG_PAR_ODD));
```

And this is for serial communication configuration.

This is the serial port that is used to TSIP communication; now let’s see how is done the frame implementation.

Previously, on chapter 3, we read about the frame, and there are two types of frames needed for this implementation, the frame with the identifier 41 and 46. Identifier 41 gives information about time reference, and identifier 46 frame gives information about state of health of the GPS. This second type of frame is not necessary to synchronization functionality, so this is the reason why it will be fulfilled completely by ‘0’ in order to inform that the GPS receiver is working properly. On the other frame, with 41 identifier, we have to fulfill the first byte with the starter header 0x10, after that is necessary the identifier 0x41. The next 4 bytes give information of seconds of the week in single format, so for this bytes will be necessary to calculate the seconds since last Sunday at 00:00 and convert them to single format to be sent.

For calculating it what is going to be done is generate a software counter, because for this protocol the frame doesn’t have to be synchronized with anything because PPS give sync. This software counter is going to increase each time that PPS generate a pulse, so is going to
increase every second, and we are going to reset to ‘0’ every Sunday at 00:00 and we do it with
the time received from protocol PTPd that informs us of the day of the week.

\[
\text{weeksec} = \frac{\text{g\_ulSystemTimeNanoSeconds}}{10^9} + \text{Taurussec};
\]
\[
\text{singleweeksec} = (\text{float}) \text{weeksec};
\]
\[
\text{intsingleweeksec} = *(\text{int} *) \& \text{singleweeksec};
\]
\[
c3 = (\text{intsingleweeksec} \& 0xFF);
\]
\[
c2 = (\text{intsingleweeksec} >> 8) \& 0xFF;
\]
\[
c1 = (\text{intsingleweeksec} >> 16) \& 0xFF;
\]
\[
oc0 = (\text{intsingleweeksec} >> 24) \& 0xFF;
\]

if (sLocalTime.ucSec != sLastTime.ucSec)
{
    usnprintf(pcStringBuffer, 32, " %02d ", sLocalTime.ucSec);
    GrStringDrawCentered(g_sContext, pcStringBuffer, -1, 160 + (2 * FIELD_WIDTH), TIME_POS_Y + 25, true);
    Taurussec ++;
    if ((sLocalTime.ucWday == 0) && (sLocalTime.ucHour == 0) && (sLocalTime.ucMin == 0) && (sLocalTime.ucSec == 0))
    {
        Taurussec = -1;
    }
}

This is the calculus for the second of the week and the conversion to single format with the
conversion to single format ready to be sent, and the second of the week increasing code
(Taurussec), which is done every second change. Cx is the name given to the different positions
on the frame, each Cx is one byte on the frame and this way then we concatenate the full
frame and is send with the serial port configured previously.

The following two bytes corresponds to the weeks since 6 of January of 1980 in INT16 format.
For obtaining this number we proceed to get the time of the board in seconds
(g\_ulSystemTimeSeconds variable) this value is the seconds since 1 January of 1980 till today.
So is needed to adapt all data to TSIP date count start; for doing it and for take into account
leap-years, the difference between two dates has been realized on excel which calculate with
precision this subtraction.

\[
\text{ExtGPSWeekNumBase} = (\text{g\_ulSystemTimeSeconds} - (10*365*24*3600) - (7*24*3600)) / 3600; // Adapt all data to 6 Jan 1980 to working with Taurus time base
\]
\[
\text{ExtGPSWeekNum} = (\text{ExtGPSWeekNumBase} / 7);
\]

With this conversion we get the number of weeks since 6 of January of 1980 and now is only
necessary to convert into INT16 format and put it in two bytes for concatenating later the frame:

\[
c5 = \text{ExtGPSWeekNum} \& 0x00FF;
\]
\[
c4 = (\text{ExtGPSWeekNum} >> 8) \& 0xFF;
\]

The last 4 bytes before the starting byte again and the closing byte, 0x03, to indicate that the
frame has ended, correspond to GPS UTC offset in single format again, so for example in Spain
it is +1 hour so 4 bytes in single format indicating +1 are: 3F800000. So now is only needed to concatenate the frame and to send it every 5 seconds refreshing the data every system loop.

```
/*
 // Periodic send to Taurus (every 5 seconds) Loop Forever
 //
 if (GPS_TSIP==1){
   if (counter==1)
     {
       count4++;
     }
   if (count4==4)
     {
       lIdx = 0;
       while(lIdx<14)
       {
         UARTCharPut(UART1_BASE,pcChars3[lIdx]);
         lIdx++;
       }
       lIdx = 0;
       while(lIdx<6)
       {
         UARTCharPut(UART1_BASE,pcChars4[lIdx]);
         lIdx++;
       }
       count4=0;
     }
   }
}
```

This is how first we send `pcChars3` corresponding to 0x41 identification frame and after that 0x46 frame, `pcChars4`.

### 5.2.3.2. IEEE 1588 with NMEA

This type of synchronization framing, like previous ones is asynchronous framing, because the synchronization is realized with PPS generated with 1588 protocol. This type of framing is output again with a serial communication, so we need again to configure a GPIO pin for this function.

We use again the same pins than before for making Rx/Tx but this time is necessary to modify the configuration for the NMEA specification compliment.

So now the NMEA protocol configuration match a baud rate of 4800, 8 bits of data, one bit of parity and 1 stop bit.

Following the previous explanation, we proceed to realize the same process for configuring this GPIO pin, but this time only in case that NMEA GPS type of synchronization is selected.
Chapter 5 - Software

GPS Time synchronization through IEEE-1588

// Serial port configuration for GPS frame sending
if (GPS_NMEA==1)
{
    UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 4800,
        (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
}

NMEA frame type is an ASCII frame, what means that this frame is composed completely by printable symbols. NMEA has a wide range of frames, each one with its own information, but for synchronization system the most common frame is identified by the header $GPZDA, that provides us only timing information.

$GPZDA,hhmmss.ss,dd,mm,yyyy,xx,yy*CC

$GPZDA,201530.00,04,07,2002,00,00*60

Where:

hhmmss  HrMinSec(UTC)
dd,mm,yyy  Day,Month,Year
xx  local zone hours -13..13
yy  local zone minutes 0..59
*CC  checksum

We proceed like before for the frame generation, first get all the data needed and after that it is concatenated and sent every second.

First data needed is hours, minutes and seconds, this is provided by 1588 protocol code implementation, like day, month and year so only is necessary to copy this values to our variables to be sent:

if (GPS_NMEA==1)
{
    lIdx = 0;
    //transform all data into char for being sent
    sprintf(bufferNMEA,"%02d%02d%02d%02d%02d%04d%2d",
        sLocalTime.ucHour, sLocalTime.ucMin, sLocalTime.ucSec,
        sLocalTime.ucMday, (sLocalTime.ucMon+1), sLocalTime.usYear,
        g_ulSystemTimeNanoSeconds);
    pcCharsNMEA[7]=bufferNMEA[0];
    pcCharsNMEA[8]=bufferNMEA[1];
    pcCharsNMEA[9]=bufferNMEA[2];
    pcCharsNMEA[10]=bufferNMEA[3];
    pcCharsNMEA[12]=bufferNMEA[5];
    pcCharsNMEA[14]=bufferNMEA[14];
    pcCharsNMEA[15]=bufferNMEA[15];
    pcCharsNMEA[17]=bufferNMEA[6];
    pcCharsNMEA[18]=bufferNMEA[7];
    pcCharsNMEA[20]=bufferNMEA[8];
    pcCharsNMEA[21]=bufferNMEA[9];
    pcCharsNMEA[23]=bufferNMEA[10];
    pcCharsNMEA[25]=bufferNMEA[12];
    pcCharsNMEA[26]=bufferNMEA[13];
Local zone hours and local zone minutes, is a constant value, in case of Europe is +1 hour so we force this value to +1.

Finally is needed the checksum calculation, done by the next algorithm which implements an XOR operation which provide the information of the parity of the frame:

```c
// CheckSum Calculation:
for (a=0; a<32; a++) {
    Buffcs[a]= pcCharsNMEA[a+1];
}
Buffcs[32]=0x00;

XOR=nmea_generateChecksum(Buffcs);
sprintf(bufferNMEA2,"%x", XOR);
pcCharsNMEA[34]= bufferNMEA2[0];
pcCharsNMEA[35]=bufferNMEA2[1];
```

Where `nmea_generateChecksum` is the function that returns the frame checksum to be sent:

```c
// Calculate NMEA Checksum
unsigned int nmea_generateChecksum(char *strPtr) {
    int p;
    char c;
    unsigned int chksum;

    c = strPtr[0]; // get first chr
    chksum = c;
    p = 1;
    while ( c != 0x00 )
    {
        c = strPtr[p]; // get next chr
        if ( c != 0x00 ) { chksum = chksum ^ c; }
        p++;
    }

    return chksum;
}
```

Ones all the frame is concatenated is only necessary to send it:

```c
while(lIdx<38)
{
    UARTCharPut(UART1_BASE,pcCharsNMEA[lIdx]);
    lIdx++;
}

lIdx = 0;
```
This is probably one of the most used protocols in synchronization systems. IRIG-B is the most common and at the same time the most precise protocol, due to it is a binary communication system and the output frame has the information of sync, it’s not necessary to add a PPS signal to the frame in order to synchronize like has been done on the previous protocols.

As seen on chapter 3 is needed information about current date in seconds, minutes, hours, day of the year and year. And we know that like in previous protocols we have this information available on PTPd code, the only problem is that all this information is needed in BCD codification, so the procedure is to copy all PTPd code information needed in new variables that are treated in order to change its information to BCD codification.

```c
// Case of IRIG-B GPS data type
if (GPS_IRIG==1){

    //generate framing
    if((sLocalTime.usYear % 4)==0){
        Month=Monthb;
    } else{
        Month=Monthnb;
    }
    Yday= (sLocalTime.ucMday) + Month[(sLocalTime.ucMon)];

    year2000=sLocalTime.usYear-2000;
    sd2bcd=bcd(sLocalTime.ucSec);
    md2bcd=bcd(sLocalTime.ucMin);
    hd2bcd=bcd(sLocalTime.ucHour);
    dd2bcd=lbcd(Yday);
    yd2bcd=0;
}
```

Bcd is a function that transforms decimal expresions on BCD format:

```c
char bcd(long decimal)
{
    int i;
    long result = 0;

    for(i = 0; decimal; ++i) {
        result += (decimal % 10) * (int) pow(16,i);
        decimal /= 10;
    }

    return(result);
}
```
Now we have the main information to be send, but as explained, this information must be send in a binary communication where a logical ‘1’ is a PWM with duty cycle of 50%, a logical ‘0’ is a duty cycle 10ms signal of 20% and the positioner bit is a 80% duty cycle signal. For doing this and matching the protocol specifications has been generated a PWM module with a 10ms signal and every time that PWM arrive to 10ms generates an interruption and rises to ‘1’ in this interruption is evaluated the next incoming information bit, and depending on its value the PWM goes to ‘0’ at 20%, 50% or 80%. Some positioners markers are implemented as a template and all time are situated on the same position.

```c
// Configure the PWM0 to count up/down without synchronization.
// Note: Enabling the dead-band generator automatically couples the 2
// outputs from the PWM block so we don't use the PWM synchronization.

PWMGenConfigure(PWM_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN |
                 PWM_GEN_MODE_NO_SYNC);

PWMGenPeriodSet(PWM_BASE, PWM_GEN_0, 31250);

// Set PWM0 at 20%
PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, 
                 PWMGenPeriodGet(PWM_BASE, PWM_OUT_0) / 5);

// Enable the PWM0 Bit 0 (PD0) output signals.

PWMOutputState(PWM_BASE, PWM_OUT_0_BIT, true);

// Enables the counter for a PWM generator block.

PWMGenEnable(PWM_BASE, PWM_GEN_0);

if(GPS_IRIG==1){

    PWMGenIntClear(PWM_BASE, PWM_GEN_0, PWM_INT_CNT_ZERO);
    PWMGenIntRegister(PWM_BASE, PWM_GEN_0,*PWM0IntHandler);
    PWMGenIntTrigEnable(PWM_BASE, PWM_GEN_0, PWM_INT_CNT_ZERO);
    IntEnable(INT_PWM0);
    UARTprintf("Activate PWM INT\n");
    PWMinEnable(PWM_BASE, PWM_INT_GEN_0);
    UARTprintf("Activate PWM INT\n");
    //
    // Enables the counter for a PWM generator block.
    //
    PWMinEnable(PWM_BASE, PWM_GEN_0);

}
```

This is the PWM enabling and the interrupt function start. Now each time that appears this interrupt is generated the full frame with the previous information of current date.
void PWM0IntHandler(void)
{
    int b=0;
    char srr;
    short lsrr;
    //UARTPRINTF("Entered to PWM INT %d\n",count);
    //Clear the interrupt
    if(count==100)
    {
        count=0;
    }
    /*
    if(count==0)
    {
        // Set PWM0 at 80% to generate 'P'
        PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, 6249);
    }*/
    if((count==0)||(count==1)||(count==10)||(count==20)||(count==30)||(count==40)
        ||(count==50)||(count==60)||(count==70)
        ||(count==80)||(count==90))
    {
        // Set PWM0 at 80% to generate 'P'
        PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, 24997);
    }
    else if
    {
        // Set PWM0 at 20%
        PWMPulseWidthSet(PWM_BASE, PWM_OUT_0,
        PWMGenPeriodGet(PWM_BASE, PWM_OUT_0) / 5);
    }
    else
    {
        //Read IRIG frame, if '1' is incoming generate a 50% PWM pulse
        //if '0' is incoming, generate a 20% PWM pulse
        if ((count>1)&&(count<10))
        {
            if(count==2)
            {
                t=0;
            }
            srr=sd2bcd;
            srr >>= t;
            b=srr & 1;
            t++;
        }
        if ((count>10)&&(count<20))
        {
            if(count==11)
            {
                t=0;
            }
        }
Chapter 5: Software

GPS Time synchronization through IEEE-1588

```c
if ((count>20) && (count<30))
{
    if (count==21)
    {
        t=0;
    }
    srr=hd2bcd;
    srr >>= t;
    b=srr & 1;
    t++;
}

if ((count>30) && (count<43))
{
    if (count==31)
    {
        t=0;
    }
    lsrr=dd2bcd;
    lsrr >>= t;
    b=lsrr & 1;
    t++;
}

if ((count>50) && (count<60))
{
    if (count==51)
    {
        t=0;
    }
    srr=yd2bcd;
    srr >>= t;
    b=srr & 1;
    t++;
}

if (b==0)
{
    // Set PWM0 at 20%
    PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, PWMGenPeriodGet(PWM_BASE, PWM_OUT_0) / 5);
}
if (b==1)
{
    // Set PWM0 at 50%
    PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, PWMGenPeriodGet(PWM_BASE, PWM_OUT_0) / 2);
}
```
This code is IRIG-B frame generation.
Chapter 6. – Test

“In order to know if all the software developed and adapted to the Stellaris board works properly as designed, some tests must be done in order to observe the behavior of all the systems developed and the know if together have the same behavior.”

6.1. Tested systems:

This system requires time synchronization, so all tests done on this chapter will be timing tests. This means that we are going to compare a source time and destination time all time and determine which is the difference between them for knowing how good our system is.

The first test to be done is the PTPd test. This way we can make our system being so precise as possible and then the following parts of the project will be time precision improved.

We must start demonstrating that the PPS generated is completely stable and always have a 1 second period with a precision of nanoseconds. If not we never will be able to synchronize completely with the master clock, because the phase will be all time changing.

![Fig.6.1 PPS period](image)

As seen on this figure, we can assume that our PPS is generated with a precision of 1µs of period, tested with a counter indicating the precision on 1 second, but for achieving this precision has been necessary to change the adjustment of the PTP code for PPS generation due to if we use the code which can be found on free code it’s impossible to achieve big precision. For doing it has been added a fix number of counts each time that the adjustment is done, this way the precision is much better and our system will improve its specifications.
"Adj" is the variable modified each cycle for adjusting the PPS generation for being synchronized with the master, so is this variable where we add a specific value.

```
// Check for max/min value of adjustment.
if(adj > ADJ_MAX)
{
    adj = ADJ_MAX;
} else if(adj < -ADJ_MAX)
{
    adj = -ADJ_MAX;
}
```

This adjustment value has been obtained connecting the debugging of the board and observing the offset between master and slave, which always achieve the same value and it was impossible to arrive to the order of microsecond on this drift. So has been added this fix value decreasing the drift between master and slave; The system starts with a drift of 50860ns:

```none
(ptpd debug) msgUnpackFollowUp: preciseOriginTimestamp.seconds 1326361307
(ptpd debug) msgUnpackFollowUp: preciseOriginTimestamp.nanoseconds 117173522
(ptpd debug) one-way delay: 0s -6277ns
(ptpd debug) offset from master: 0s -50860ns
(ptpd debug) observed drift: 49938
Thu Jan 12, 2012 09:41:48 (GMT)
```

And after the adjustment and with some stabilizing time we get a drift of 1250 ns

```none
Thu Jan 12, 2012 09:53:23 (GMT)
```

```
(ptpd debug) msgUnpackFollowUp: preciseOriginTimestamp.seconds 1326362002
(ptpd debug) msgUnpackFollowUp: preciseOriginTimestamp.nanoseconds 913605538
```
GPS Time synchronization through IEEE-1588

After that we will proceed to compare the PPS emitted by the source (master clock) and the PPS generated by the destination 1588 protocol implementation (slave), if the protocol is working properly we will see on the screen that the time is set properly and that the PPS is completely synchronous.

This shows that the PPS generated by our board (below pulse) is completely synchronized with the PPS generated by the master (above pulse). This is seen in a 500ms timing resolution, but if it’s observed in a 500 μs timing resolution screen it continues being synchronized.

**Fig.6.2 Phase sync**

Here we can see both PPS, master and slave, overlapping in a 500 μs time resolution capture.

**Fig.6.3 High resolution phase synchronization**

So, seeing this PPS synchronization and realizing that the board is properly synchronized with the master clock time, we can assume that the PTPd code is working as expected with an error of 1 μs.
This accuracy of 1 \( \mu s \) is the precision that we can assume on NMEA framing and TSIP, because these two protocols of systems synchronization are based on asynchronous framing and synchronous PPS for triggering the entrance of the time stamp. So for these two systems it’s only necessary to test the correct framing, and being sure that is on correct format for the systems which use its types of sync systems.

First we can talk about TSIP protocol, for testing this protocol has been used a seismometer which uses this type of framing for synchronize itself.

![Fig.6.4 Stellaris board connected to the seismometer (white box); blue wire for PPS and orange one for TSIP frames TX.](image1)

![Fig.6.5 Seismograph (Taurus) synchronization](image2)

As seen on the previous image the time indicator shows that the received framing is OK and the time is properly received and used for the synchronization.
Now it’s time for NMEA frame test. This time we are talking of an asynchronous frame sending again and a PPS for the time synchronization, so the precision achieved will be of $1\mu$s again if the frames send are correct and accepted by the receiver.

This time the receiver is the own laptop, with a time synchronization program, “GPS Time”, as can be seen on the next figure, this software gets the time from the NMEA command used for time, $GPZDA$, and shows you the frame received, the communication configuration and if the checksum is correct, it synchronize the laptop with the received time. We can observe that there is no error in any packet received.

![Fig.6.6. NMEA synchronization software](image)

The connection between laptop and PC has been done with a converter wire, from 3.3TTL levels to RS-232 (USB) communication, this way in a USB is received the serial communication with the NMEA information.

The next protocol to test is IRIG-B, this protocol is a synchronous communication protocol, so this means that is not going to be used any PPS as triggering. The main signal is synchronized every pulse that is send. It’s done with the two first identifiers of each frame that indicates the synchronization with PPS, and each bit has bit duration of 10ms.
With the counter we can analyze the period of our signal, this period is properly generated, with the maximum precision that our PWM allows. Maybe this precision is not sufficient for achieving exactly 1 second after 100 bits per frame, but the synchronization on IRIG-B is not the bit time, it is the start frame. So it’s not completely determinant this time as could be the start frame time, for this we can see that the frame starts just when the rising edge on PPS occurs.

The PPS is completely synchronous with the start of the frame (two identifiers). For ensure the correct behavior the system is connected to a timeserver, which can be synchronized by an external system with IRIG-B002 framing. This system has a very high precision clock and our system must work with precision to be able to synchronize the timeserver Meinberg Lantime M600. Once we connect both systems, can be seen on timeserver screen that the time base is given by the external connection and that the error between the main clock of the timeserver and the IRIG-B frames is 23.15ns.
This indicates that the master clock will be the time reference, and as can be seen now the master clock is our Stellaris board, and three LEDs indicates that there is time reference and time service given by our developed system, also there is no alarm.

*Fig.6.9. Meinberg state indicators*

Now we have tested all our systems and we know the precision that each type of framing is going to give us, we can ensure that the best synchronization system is IRIG-B framing, and this is the reason why this type of framing systems are the most common on sync systems, due to its precision.
Chapter 7 – Future improvements

This system as mentioned before doesn’t have a precise time stamp, and this is one field to improve, for doing this the best solution, is to get a board able to work with hardware time stamping and start using with this board protocol IEEE-1588 v2.

With this implementation is possible to achieve an average clock offset of 2.18 ns, standard deviation of 13.88 ns and Peak-to-Peak range -62ns to +62ns.

<table>
<thead>
<tr>
<th>Average clock offset</th>
<th>-2.18 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard deviation</td>
<td>13.88ns</td>
</tr>
<tr>
<td>Peak-to-Peak range</td>
<td>-62 ns to 62ns</td>
</tr>
</tbody>
</table>

This will improve the main characteristics of our PPS generation that now is 1µs precise and this will make every frame generation much better.

These time features can be reached due to the ability to adjust the hardware clock very precisely (as precise as one part per billion). Software algorithm for fine (as small as one part per billion) adjustment of the hardware clock. Ability to set Sync/Follow-up transmit time values of less than one second. This allows for multiple Sync/Follow-ups per second, which provides enhanced accuracy and tracking of the "slave" system to the "grandmaster" system.

On a Freescale MPC8313E-RDB board running eight Sync/Follow-up Messages per second, it will support a plus or minus 50 nanosecond offset from master more than 99.9% of the time. At 128 Sync/Follow-ups per seconds, the system will synchronize within 16 nanoseconds of the "grandmaster" system.

This Freescale board supports hardware time stamping and has a prize of 300$, so with this changes, and developing again the frame generation would be possible to improve the system main time characteristics.

Fig.7.1 MPC8313E Freescale test board
Chapter 8. – Conclusions

Time synchronization is really needed on data acquisition systems for subsequent analysis of this data. This system allows the user to work with a wide range of DAQ’s synchronized by a GPS receiver, although this GPS doesn’t have enough signal for synchronizing it with precision.

So we can connect the designed system to synchronize any other machine, only with a master clock connection via Ethernet, and with an accuracy of 1µs, which for most kind of industrial systems is enough, and for the proposal of this project, that is synchronizing an underwater seismometer, is much more than enough.

For doing this has been developed:

- Synchronization software
- Frame generation software
- Debug / information screen on Stellaris board
- Conversion system between 3.3 TTL systems and RS-232 systems

In addition to an accurate test for each system to characterize its performance.

As final report can be said that all over this project, a big knowledge on synchronization systems has been achieved and the most important thing to take into account when designing a precise time system, is to try to avoid as much as possible the software part, and try to get all the time stamping via hardware, and use Direct Memory Access to work with generation of framing, this way all the delays added on this project due to excess of software, can be deleted and then have a better behavior on time synchronization.
Chapter 9. – Reference


[3] IRIG features

[4] What is IRIG time code
http://www.control.com/thread/123011504


[6] NMEA data
http://www.gpsinformation.org/dale/nmea.htm#ZDA

[7] IRIG programming handbook


[10] Texas instruments forum
http://e2e.ti.com/members/1709837/default.aspx

http://ptpd.sourceforge.net/

[12] PTPv2 information
http://code.google.com/p/ptpv2d/wiki/Introduction

[13] Freescale test board PTPd v2
http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MPC8313E-RDB

[14] Nanometrics
www.nanometrics.ca
ANNEX A

GPS Time synchronization through IEEE-1588

The Lassen iQ GPS Receiver uses a proprietary communicates via serial communication. The serial stream of data from the receiver is broken into bytes that form packets of varying length, which can be decoded into a command with data. The receiver uses a proprietary packet structure, which was developed by the manufacturer, called Trimble Standard Interface Protocol (TSIP). According to the datasheet, the packet structure is as follows:

“TSIP packet structure is the same for both commands and reports. The packet format is:

<DLE> <id> <data string bytes> <DLE> <ETX>

Where:
- <DLE> is the byte 0x10
- <ETX> is the byte 0x03
- <id> is a packet identifier byte, which can have any value excepting <ETX> and <DLE>.

The bytes in the data string can have any value. To prevent confusion with the frame sequences <DLE> <ID> and <DLE> <ETX>, every <DLE> byte in the data string is preceded by an extra <DLE> byte (‘stuffing’). These extra <DLE> bytes must be added (‘stuffed’) before sending a packet and removed after receiving the packet. Notice that a simple <DLE> <ETX> sequence does not necessarily signify the end of the packet, as these can be bytes in the middle of a data string. The end of a packet is <ETX> preceded by an odd number of <DLE> bytes.

Multiple-byte numbers (integer, float, and double) follow the ANSI/IEEE Std. 754 IEEE Standard for binary Floating-Point Arithmetic. They are sent most-significant byte first. This may involve switching the order of the bytes as they are normally stored in Intel based machines.

Specifically:
- UINT8 = Byte: An 8 bit unsigned integer.
- UINT16 = Word: A 16 bit unsigned integer.
- INT16 = Integer: A 16 bit integer.
- INT32 = Long: A 32 bit integer.
- UINT32 = ULong: A 32 bit unsigned integer.
- Single - Float, or 4 byte REAL has a precision of 24 significant bits, roughly 6.5 digits.
- Double - 8 byte REAL has a precision of 52 significant bits. It is a little better than 15 digits” (Lassen 87).

The following pages reference packets which were recorded during a capture of all output from the GPS receiver in the early morning of July 12, 2005. Due to the amount of data recorded, only the excerpts are presented here. Please contact the author for a digital copy of the complete data capture.
From the data stream, we will examine the following packet: 104148 44231D05334150001003

**-GPS Time of Week** – Bytes 0 – 3 – This is given as the number of seconds (SINGLE) from 12:00:00 am on Sunday. If this were to be negative, it would denote the time is not yet known.

From the packet, bits 0:3, 0x4844231D, is 0100 1000 0100 0100 0010 0011 0001 1101 in binary. This equals 2^(1001 0000 – 0111 1111) * 1.100 0100 0010 0011 0001 1101, which rounded is 1100 0100 0010 0011 00 or 200,844.

**-Extended GPS Week Number** – Bytes 4 – 5 – This is the number of weeks (INT16) since January 6, 1980.

From the packet, bits 4:5, 0x0533, equals 1,331 decimal. Given there are 1331 weeks and two days between the recording date and January, 6 1980, this result is as expected.

**-GPS UTC Offset** – Bytes 6 – 9 – This is the offset in seconds (SINGLE) to calculate From the packet, bits 6:9, 0x41500000, is
ANNEX A

GPS Time synchronization through IEEE-1588

10 4A 3F 04 5F 0E BF B8 09 A9 42 58 69
D0 C7 EF BF 4A 48 44 21 C0 10 03

The following map was generated from entering the GPS coordinates derived from the data stream, converted to degrees, at Jeff Boulter’s website. As you can see this exactly pinpoints my apartment in Windmeadows.

-Latitude – Bytes 0:3 – This is the latitude radians (SINGLE) of the current GPS coordinate, with positive for north latitude and negative for south latitude.

Hex: 3F 04 5F 0E Binary (float): 0011 1111 0000 0100 0101 1111 0000 1110
Sign = 0, Exp = 0111 1110, Mantissa = 1.000 0100 0101 1111 0000 1110 = (Sign ? (-1) : (1)) * 2^(Exp - 127) * Mantissa = Mantissa with binary point shifted left 1 place = 0.1000 0100 0101 1111 0000 1110 = ~0.517075 radians north latitude.

-Longitude – Bytes 4:7 – This is the longitude radians (SINGLE) of the current GPS coordinate, with positive for east longitude and negative for west longitude.

Hex: BF B8 09 A9 Binary (float): 1011 1111 1011 1000 0000 1001 1010 1001
Sign = 1, Exp = 0111 1111, Mantissa = 1.011 1000 0000 1001 1010 1001 = (Sign ? (-1) : (1)) * 2^(Exp - 127) * Mantissa = Sign bit denotes negative = Mantissa with binary point shifted 0 places = 1.011 1000 0000 1001 1010 1001 = ~1.437795 radians west longitude.
ANNEX A

GPS Time synchronization through IEEE-1588

10 56 00 00 00 00 00 00 00 00 00 00 00 00 43 2D 91 53 48 44 23 00 10 03

-East velocity is bytes 0:3 and of type SINGLE. It is in meters / second and positive for east and negative for west.

Since the receiver could not be in motion at the time of testing this value is zero.

-North velocity is bytes 4:7 and of type SINGLE. It is in meters / second and positive for north and negative for south.

Since the receiver could not be in motion at the time of testing this value is zero.

-Up velocity is bytes 8:11 and of type SINGLE. It is in meters / second and positive for up and negative for down.

Since the receiver could not be in motion at the time of testing this value is zero.

-Clock Bias Rate is bytes 12:15 and of type SINGLE. It is in meters per second.

Hex: 43 2D 91 53 Binary (float): 0100 0011 0010 1101 1001 0001 0101 0011
   Sign = 0, Exp = 1000 0110, Mantissa = 1.010 1101 1001 0001 0101 0011 = (Sign ? (-1) : (1)) * 2^(Exp - 127) * Mantissa = Mantissa with binary point shifted right 7 places = 1010 1101.1001 0001 0101 0011 = ~173 decimal seconds difference.

-Time of Fix is bytes 16:19 and of type SINGLE. It is in GPS seconds and is the time of the fix that produced the data packed occurred.

Hex : 48 44 23 00 Binary (float): 0100 1000 0100 0100 0010 0011 0000 0000
   Sign = 0, Exp = 1001 0000, Mantissa = 1.1100 0100 0010 0011 0000 0000 = (Sign ? (-1) : (1)) * 2^(Exp - 127) * Mantissa = Mantissa with binary point shifted right 17 places = 1100 0100 0010 0011 00.00 0000 = 200,844 decimal seconds from midnight Sunday of the week, or about 7:45 am.
Once I had written a program to output the GPS coordinates to the LCD of Mr. 2-Bots, I performed an experiment to ascertain how much change in the coordinate corresponded to a given distance of movement.

**Lat Lon** Left 12’ 3F04 5F35 BFB8 0992 Center 3F04 5F3E BFB8 09A0 Right 12’ 3F04 5F3C BFB8 09A4

Latitude: 0 0111 1110 Sign = 0 Exponent = 126 Longitude: 1 0111 1111 Sign = 1 Exponent = 127

= (Sign ? (-1) : (1)) * 2^(Exp - 127) * Mantissa

Center – Left: sqrt((9>>1)^2 + 14^2) = ~15 Center – Right: sqrt((2>>1)^2 + 4^2) = ~5
Works Cited
  <http://boulter.com/gps/>
Description

Being a very stable IEEE 1588-2008 Grandmaster clock, the LANTIME M600/MRS/PTP not only provides a highly accurate source of synchronization for PTP clients ("slaves" like the PTP270PEX), it additionally introduces the absolute time ("current time of day") to your PTP networks.

The PTP V2 (Multicast/Unicast) implementation is fully compliant to the IEEE 1588-2008 standard and provides PTP management messages as well. In master mode, the PTP unit is able to synchronize two-step clocks, whereas in Slave mode, it can be synchronized by either a one-step clock or a two-step clock.

As an ultra stable NTP and PTP time provider, the LANTIME M600/MRS/PTP not only represents a highly accurate source of synchronization for all network devices supporting the Precision Time Protocol (PTP), the Network Time Protocol (NTP) and the Simple Network Time Protocol (SNTP), it also offers a number of legacy time and frequency outputs for keeping non-networked devices in sync.

The Meinberg MRS technology (Multi Reference Sources) enables you to utilize one or more time and frequency references in prioritized order defined by your individual requirements. The Meinberg Intelligent Reference Switching Algorithm (IRSA) ensures that switching from a highly accurate reference source (e.g. GPS) to a less accurate one (e.g. IRIG or NTP) is delayed as long as the internal ultra stable oscillator is capable of maintaining an accuracy level that is better than the one of the next available reference source in the priority list.

Redundancy of input references:
The MRS technology offers a flexible solution to the changing availability of different synchronization sources for highly critical operating systems. The ability to use multiple independent sync references allows you to fulfill redundancy requirements of your network synchronization solution.

Lab environments
Monitoring and measurement of synchronization sources such as determining and logging the accuracy of an IRIG generator or a PPS source is easily done with the MRS. Furthermore, the LANTIME M600/MRS/PTP is a perfect solution to test the PTP synchronization quality within existing network environments by automatically comparing the PTP input source to another high accurate reference source like GPS or 1PPS. All reference inputs can be measured against each other, and the high quality VF-Display provides a graphical representation of the measurements between different reference sources.

PTPv2 Translational
The MRS system allows you to translate PTP into a variety of output signals like 1PPS, 10MHz or IRIG. This feature makes it easy to transfer legacy timing signals over a IP based infrastructure while maintaining very high accuracy.

The GNU/Linux operating system of the LANTIME's SBC (Single Board Computer) has been optimized to ensure a high level of security and reliability.

The configuration of the system can be done by using a standard web browser to access the extensive but straightforward HTML interface. Alternatively a text based and menu driven setup utility can be started from the shell.
prompt after logging into the unit via Telnet or SSH.

The security-related features of LANTIME time servers satisfy highest demands. The time synchronization data can be reliably signed and secured by symmetric keys (MD5) and the NTP autokey procedures. This protects the clients against manipulated time and man-in-the-middle attacks and allows them to verify that the NTP packets they received were sent by the LANTIME. Additionally the whole LANTIME configuration can be done by using encrypted channels (e.g. SSH, HTTPS or SNMPv3). Every unused/unneeded protocol can be disabled in order to reduce possible points of attack.

In order to support network management systems the LANTIME time servers offer an extensive SNMP interface, which can be accessed by SNMP V1, V2c and V3. It allows the monitoring of all relevant system parameters (including operating system parameters, network interface statistics, detailed GPS and NTP status information as well as the complete system configuration) and can be used to alter the LANTIME configuration via SNMP set commands, too.

LANTIME time servers are designed to be deployed in IPv6 networks, the NTP time synchronization as well as the configuration interfaces (Web-based, SSH and SNMP) comes with IPv6 support. You can assign several IPv6 addresses and the system supports automatic configuration by IPv6 autoconf.

Because of its modular system architecture it is possible to equip a LANTIME time server with a variety of different reference time sources. Optionally several additional frequency-, serial string- and pulse outputs are available and by combining two (even different) time sources and redundant power supplies, high-availability systems are no problem.

The LANTIME M900/MRS/PTP is equipped with high precision oscillator "OCXO HQ" (look at oscillator options for details) as standard. The oscillator determines the holdover characteristics (e.g. when the GPS signal is disturbed or jammed). The oscillator option "OCXO DHQ" is available to fulfill higher requirements.

Please note that the GPS antenna is not included in the standard scope of delivery and has to be ordered separately, if you are planning to use GPS as a synchronization source.
## Characteristics

**Type of receiver**: 6 channel GPS C/A-code receiver  
**Display**: Vacuum fluorescent graphic display (VFD), 256 x 64 dots  
**Control elements**: Eight push buttons to set up basic network parameters and to change receiver settings  
**Status info**: Four bicolor LEDs showing status of:  
- reference time  
- time service  
- network  
- alarm  
**Input signal**:  
- 1x GPS Antenna Input  
- 1x PPS in  
- 1x 10MHz in  
- 1x IRIG DCLS in  
- 1x IRIG AM in  
- 1x PTPv2 in (if configured as PTP slave)  

### Pulse Synchronization Input Signals  
**1 Pulse Per Second, TTL (BNC)**  
**Frequency inputs**:  
- 10 MHz sine (1.5Vpp) or  
- 10 MHz TTL  
**IRIG Time Code Input**: IRIG-B123, B122, B003, B002, B008, B007, B126, B127, IEEE 1344 and AFNOR NFS 87-500  
**Frequency outputs**:  
- 10 MHz via female BNC connector, TTL into 50 Ohm  
- Synthesizer 1/8 Hz up to 10 MHz via female BNC connector, TTL into 50 Ohm  
Accuracy depends on oscillator (standard: OCXO HQ), look at Oscillator options  
**Pulse outputs**: Pulse per second (PPS) and pulse per minute (PPM) via female BNC connectors, TTL into 50 Ohm, pulse width: 200msec, active high  
**Accuracy of pulse outputs**: \( \pm 100 \text{ns (OCXO HQ, OCXO DHQ)} \)  
**Interface**: Two independent serial RS232-interfaces, menu configurable  
**Data format of interfaces**: Baud rates: 300, 600, 1200, 2400, 4800, 9600, 19200 Baud  
Data formats: 7N2, 7E1, 7E2, 7O1, 8E1, 8N1, 8O1  
Time strings: r:MyMeinberg Standards-Telegram, SAT, Uni Erlangen (NTP), SPA, RACAL, Syplex, NMEA0183 (RMC, GGA, ZDA), Meinberg GPS, COMPUTIME, ION oder [4]  
Capture-Telegramm  
**Unmodulated time code output**: DCLS, TTL into 50 Ohm via female BNC connector, active high  
**Modulated time code output**: IRIG AM sine wave signal via female BNC connector: 3Vpp (MARK), 1Vpp (SPACE) into 50 Ohm
### ANNEX B

**GPS Time synchronization through IEEE-1588**

| Generated time codes | IRIG B002: 100pps, DCLS signal, no carrier, BCD time of year  
IRIG B122: 100pps, AM sine wave signal, 1 kHz carrier, BCD time of year  
IRIG B003: 100pps, DCLS signal, no carrier, BCD time of year, SBS time of day  
IRIG B123: 100pps, AM sine wave signal, 1 kHz carrier, BCD time of year, SBS time of day  
IEEE1344: Code according to IEEE1344-1995, 100pps, AM sine wave signal, 1 kHz carrier, BCD time of year, SBS time of day, IEEE1344 expansion for date, time zone, daylight saving and leap second in Control Functions Segment  
AFNOR: Code according to NFS-87590, 100pps, AM sine wave signal, 1 kHz carrier, BCD time of year, complete date, SBS time of day |
| Alarm output | Synchronous state of the module, relay output (changeover contact) |
| Network Interface | 1 x 10/100 MBit with RJ45, IEEE 1588  
5 x 10/100 MBit with RJ45 |
| Power supply | Standard: 100-240 VAC  
available DC variants: 100-240 VDC, 12 VDC, 24 VDC and 48 VDC |
| Power consumption | 30 W |
| Universal Serial Bus (USB) Ports | 1x USB Port in front panel:  
- install firmware upgrades  
- backup and restore configuration files  
- copy security keys  
- lock/unlock front keys |
| Supported Time String Formats | Meinberg Standard Timestring, Uni Eifangcn Timestring, SYSPLEX Timer, NMEA, Computime, ABB-SPA, SAT, Arbiter |
| Single-Board-Computer | i386 compatible 500MHz CPU, 128 MB RAM |
| Operating System of the SBC | Linux with nano kernel (incl. PPSkit) |
| Network protocols OSI Layer 4 (transport layer) | TCP, UDP |
| Network protocols OSI Layer 7 (application layer) | TELNET, FTP, SSH (incl. SFTP, SCP), HTTP, HTTPS, SYSLOG, SNMP |
| Internet Protocol (IP) | IP v4, IP v6 |
| Network Autoconfiguration Support | IPv4: Dynamic Host Configuration Protocol - DHCP (RFC 2131)  
IPv6: Autoconfiguration Networking - AUTOCONF |
| Network Time Protocol (NTP) | NTP v2 (RFC 1119), NTP v3 (RFC 1305), NTP v4 (no RFC)  
SNTP v3 (RFC 1769), SNTP v4 (RFC 2030)  
MD5 Authentication and Autokey Key Management |
ANNEX B

GPS Time synchronization through IEEE-1588

<table>
<thead>
<tr>
<th>Precision Time Protocol (IEEE 1588)</th>
<th>PTP/IEEE 1588-2008 including</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* Network Protocols:</td>
</tr>
<tr>
<td></td>
<td>- UDP/Pv4 (Layer 3) (Multicast/Unicast)</td>
</tr>
<tr>
<td></td>
<td>- IEEE 802.3 (Layer 2) (Multicast)</td>
</tr>
<tr>
<td></td>
<td>* Delay Mechanisms:</td>
</tr>
<tr>
<td></td>
<td>- End-to-End (Multicast/Unicast)</td>
</tr>
<tr>
<td></td>
<td>- Peer-to-Peer (Multicast)</td>
</tr>
<tr>
<td></td>
<td>* PTP Management Messages for monitoring and configuration</td>
</tr>
</tbody>
</table>

| Time Protocol (TIME)               | Time Protocol (RFC 868)          |
| Daytime Protocol (DAYTIME)         | Daytime Protocol (RFC 867)       |
| IEC 61850                          | Synchronization of IEC 61850 compliant devices by using SNTP |
| Hypertext Transfer Protocol (HTTP) | HTTP/HTTPS (RFC 2068)            |
| Secure Shell (SSH)                 | SSH v1.3, SSH v1.5, SSH v2 (OpenSSH) |
| Telnet                             | Telnet (RFC 854-RFC 851)         |
| Simple Network Management Protocol (SNMP) | SNMPv1 (RFC 1157), SNMPv2c (RFC 1901-1909), SNMP v3 (RFC 3411-3419) |
| Ambient temperature                | 0 ... 50°C / 32 ... 122°F        |
| Humidity                           | Max. 85%                         |
| Scope of supply                    | Time Server, power cable and a USB storage device with Quick-Start Guide and a detailed reference manual as PDF file in the "Manual" folder. |
| Technical Support                  | Meinberg offers free lifetime technical support via telephone or e-mail. |
| Warranty                           | Three-Year Warranty              |
| Firmware Updates                   | Firmware is field-upgradeable, updates can be installed directly at the unit or via a remote network connection. Software updates are provided free of charge, for the lifetime of your Meinberg product. |
| RoHS-Status of the product         | This product is fully RoHS compliant |
| WEEE status of the product         | This product is handled as a B2B category product. In order to secure a WEEE compliant waste disposal it has to be returned to the manufacturer. Any transportation expenses for returning this product (at its end of life) have to be incurred by the end user, whereas Meinberg will bear the costs for the waste disposal itself. |
GPS Time synchronization through IEEE-1588

Options and Accessories
Optional add-ons and accessories: [Product Options]

Additional Information
Additional information about the Meinberg LANTIME family of NTP time servers and other LANTIME models can be found on the [LANTIME NTP Time Server Family Page]

Product Data Sheet
PDF Infosheet [LANTIME M600/MRS/PTPv2]

Manual
The English manual is available as a PDF file: [Download PDF]

Links:
Overview of IRIG-B Time Code Standard

Introduction
The IRIG time codes were originally developed by the Inter-Range Instrumentation Group (IRIG), part of the Range Commanders Council (RCC) of the US Army. The standard was first published in 1950 and has been revised several times by the Telecommunications and Timing Group (TTG) of the RCC. The latest version is IRIG standard 200-04, “IRIG Time Code Formats,” updated in September, 2004.

Available Formats
Although the ‘IRIG-B’ time code is best known, the standard actually defines a family of rate-scaled serial time codes. The six code formats use different pulse rates, or bit rates, as shown in the table below.

<table>
<thead>
<tr>
<th>Format</th>
<th>Pulse Rate (or Bit Rate)</th>
<th>Index Count Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRIG-A</td>
<td>1000 PPS (pulse per second)</td>
<td>1 ms</td>
</tr>
<tr>
<td>IRIG-B</td>
<td>100 PPS</td>
<td>10 ms</td>
</tr>
<tr>
<td>IRIG-D</td>
<td>1 PPM</td>
<td>1 minute</td>
</tr>
<tr>
<td>IRIG-E</td>
<td>10 PPS</td>
<td>100 ms</td>
</tr>
<tr>
<td>IRIG-G</td>
<td>10000 PPS</td>
<td>0.1 ms</td>
</tr>
<tr>
<td>IRIG-H</td>
<td>1 PPS</td>
<td>1 second</td>
</tr>
</tbody>
</table>

Time Code Attributes
All IRIG time code formats use pulse-width coding. A ‘binary 1’ pulse has a duration of 50% of the index count interval, and a ‘binary 0’ pulse has a duration of 20% of the index count interval. In addition “Position Identifiers” have a duration of 50% and are used as reference markers.

IRIG time code signals may be:
- Unmodulated (DC level shift, no carrier signal)
- Modulated (amplitude-modulated, sine wave carrier)
- Modified Manchester (amplitude-modulated, square wave carrier).

Three types of coded expressions are used in the IRIG standard:
- Binary Coded Decimal time-of-year (BCD1yr) and year (BCD1yw)
- Control Functions (CF), set of bits reserved for user applications
- Straight Binary Seconds (SBS) time-of-day (0 to 86400 seconds)
GPS Time synchronization through IEEE-1588

IRIG STANDARD 200-04 (Continued)

IRIG Time Code Designations

In addition to the letter used to designate one of the six IRIG code formats, signal identification numbers are used to further describe specific characteristics. Thus, the complete IRIG time code designation consists of a letter and three digits, as shown below.

```
+-----+-----+-----+-----+-----+-----+
| B   | 0   | 0   | 6   |     |     |
+-----+-----+-----+-----+-----+-----+

IRIG time codes – naming convention
```

**IRIG Signal Identification Numbers (3 Digits)**

<table>
<thead>
<tr>
<th>1st Digit</th>
<th>Modulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Unmodulated – DC Level Shift (DCLS), pulse-width coded</td>
</tr>
<tr>
<td>1</td>
<td>Amplitude modulated, sine wave carrier</td>
</tr>
<tr>
<td>2</td>
<td>Manchester modulated</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2nd Digit</th>
<th>Carrier Frequency / Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No carrier (DCLS)</td>
</tr>
<tr>
<td>1</td>
<td>100 Hz / 10 ms resolution</td>
</tr>
<tr>
<td>2</td>
<td>1 kHz / 1 ms resolution</td>
</tr>
<tr>
<td>3</td>
<td>10 kHz / 100 microsecond resolution</td>
</tr>
<tr>
<td>4</td>
<td>100 kHz / 10 microsecond resolution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3rd Digit</th>
<th>Coded Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BCDHV, CF, SBS</td>
</tr>
<tr>
<td>1</td>
<td>BCDHV, CF</td>
</tr>
<tr>
<td>2</td>
<td>BCDHV</td>
</tr>
<tr>
<td>3</td>
<td>BCDMV, SBS</td>
</tr>
<tr>
<td>4</td>
<td>BCDMV BCDMV, CF, SBS</td>
</tr>
<tr>
<td>5</td>
<td>BCDMV BCDMV, CF</td>
</tr>
<tr>
<td>6</td>
<td>BCDMV BCDMV</td>
</tr>
<tr>
<td>7</td>
<td>BCDMV BCDMV, SBS</td>
</tr>
</tbody>
</table>
IRIG-B PROTOCOL DESCRIPTION

IRIG-B Overview
IRIG time code B (IRIG-B) is widely used in the electrical power industry. IRIG-B has a pulse rate of 100 pulses-per-second with an index count of 10 milliseconds over its one-second time frame. It contains time-of-year and year information in a BCD format, and (optionally) seconds-of-day in SBS.

IRIG-B Signals
IRIG-B is typically distributed as a DC level shift, pulse-width coded signal ("unmodulated IRIG-B") or as an amplitude-modulated signal based on a sine wave carrier with a frequency of 1kHz ("modulated IRIG-B"). Modified Manchester modulation is also specified in the standard but is less common. A comparison of IRIG-B coding methods is given in the figure below.

IRIG-B Reference Markers
IRIG-B uses reference markers called "Position Identifiers." The presence of two consecutive reference markers signifies the start of the time frame. The first reference marker alerts that the next rising edge will be the PPS marker. ("On-Time 1 PPS" shown above.)
INFORMATION TECHNOLOGY—Overview of IRIG-B Time Code Standard

IRIG-B Encoding
IRIG-B consists of 100 bits produced every second, 74 bits of which contain various time, date, time changes and time quality information of the time signal. Consisting of logic ones, zeros and position identifier bits, the time code provides a reliable method of transmitting time to synchronize power equipment devices. There are three functional groups of bits in the IRIG-B time code: Binary Coded Decimal (BCD), Control Functions (CF) and Straight Binary Seconds (SBS).

The BCD group contains time information including seconds, minutes, hours and days, recycling yearly. The BCD time-of-year code (BCDomy) reads zero (0) hours, minutes, seconds, and fraction of seconds at 2400 each day and reads day 001 at 2400 of day 365, or day 366 in a leap year. The BCD year code (BCDoy) counts year and cycles to the next year on January 1st of each year and will count to year 2099.

The (optional) SBS time-of-day code consists of the total elapsed seconds, recycling daily. SBS reads zero (0) seconds at 2400 each day excluding leap second days when a second may be added or subtracted.

The CF group contains year, time quality, leap year, pending leap seconds and parity. Other CF bits are reserved for user-defined purposes, depending on application.

Lastly, position identifiers separate the various components of the IRIG-B time code.

IEEE-1344 Extensions
Year information was not specified in the IRIG standard prior to its 2004 revision. Before 2004, the IEEE adopted a standard (IEEE-1344) which included year data as part of the IRIG-B signal. This variation came to be known as “IEEE-1344 extensions.”

IEEE-1344 extensions use extra bits of the Control Functions (CF) portion of the IRIG-B time code. Within this portion of the time code, bits are designated for additional features, including:
- Calendar Year (now called BCDmyw)
- Leap seconds, and leap seconds pending
- Daylight Saving Time (DST), and DST pending
- Local time offset
- Time quality
- Parity
- Position identifiers

To be able to use these extra bits of information, power system devices and other equipment receiving the time code must be able to decode them. Refer to individual product manuals to determine whether IEEE-1344 extensions are supported.

Since year information is now considered part of BCD (denoted as BCDomyw), what was formerly considered B002 and B122 (with IEEE Extensions ON) would now be denoted as B006 and B126.
GPS Time synchronization through IEEE-1588

ANNEX C

TECH NOTE—Overview of IRIG-B Time Code Standard

IRIG-B BCD time-of-year (in days, hours, minutes, seconds) and year and straight binary seconds-of-day and control bits

WIRING

Unmodulated or Demodulated?

An IRIG-B time signal can be modulated (over a carrier signal) or unmodulated (no carrier signal).

In some manufacturers’ literature the term “demodulated” is used to describe an IRIG-B signal with no carrier signal. However, the term “demodulated” does not appear in the IRIG standard. In most cases, it may be assumed that this term is synonymous with unmodulated.

IRIG-B Implementation

The IRIG 200-04 standard does not define specific signal levels for IRIG-B.

Typical techniques for transmission of unmodulated IRIG-B include:

- TTL-level signal over coaxial cable or shielded twisted-pair cable
- Multi-point distribution using 24 Vdc for signal and control power
- RS-422 differential signal over shielded twisted-pair cable
- RS-232 signal over shielded cable (short distances only)
- Optical fiber

Typical techniques for transmission of modulated IRIG-B include:

- Coaxial cable, terminated in 50 ohms or higher.
- Shielded twisted-pair cable

© 2011 Cyber Sciences, Inc. All rights reserved.

www.cyber-sciences.com
ANNEX C

GPS Time synchronization through IEEE-1588

APPLICATION OF IRIG-B IN CSI PRODUCTS

STR-100 Satellite Time Reference

STR-100/IRIG-B

Like the base model, the STR-100/IRIG-B Satellite Time Reference accepts a GPS smart antenna input, but its output is an unmodulated IRIG-B signal, type B002 (unmodulated signal, DC level shift pulse, BCDrxc) at 5 Vdc nominal.

STR-IDM

The STR-IDM Satellite Time Reference IRIG-B Distribution Module is a companion product to the STR-100/IRIG-B and enables the distribution of an unmodulated IRIG-B signal over long distances and to multiple devices. It also is used to distribute 24 Vdc control power.

The IRIG-B signal levels between the STR-100/IRIG-B and the IDM (as well as to other IDMs) are 24 Vdc nominal, along with 24 Vdc control power. Each STR-IDM provides 8 additional IRIG-B outputs, type B002 (unmodulated signal, DC level shift pulse, BCDrxc) at 5 Vdc nominal.

SER-3200

The CyTime® Sequence of Events Recorder, SER-3200, accepts an unmodulated IRIG-B signal to provide its precision time reference, and supports IEEE-1344 extensions, type B006 (unmodulated signal, DC level shift pulse, BCDrxc and BCDrxc).

REFERENCES

For More Information (CSI)

STR Instruction Bulletin (IB-STR-01)
STR/IRIG-B Addendum (IB-STR-02)
STR-IDM Instruction Bulletin (IB-IDM-01)
SER-3200 Instruction Bulletin (IB-SER-02)
SER-3200 Reference Guide (IB-SER-02)
Tech Note: SER System Architecture (TN-101)

Doc. no: TN-102R1
Jan-2011

Cyber Sciences, Inc. USA

6 | www.cyber-sciences.com

© 2011 Cyber Sciences, Inc. All rights reserved.