DEVELOPMENT OF IRIG DECODER EMBEDDED SYSTEM

A PROJECT REPORT

Submitted by

A S DHARSINI	**– 963820106004**
K R NANDU	**– 963820106008**
M P SHRI VEENA	**– 963820106011**
V VIBISHA	**– 963820106012**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

GOOD SHEPHERD COLLEGE OF ENGINEERING AND TECHNOLOGY

MARUTHAMPARAI – 629101

ANNA UNIVERSITY :: CHENNAI 600025

May 2024

ANNA UNIVERSITY : CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report "**DEVELOPMENT OF IRIG DECODER EMBEDDED SYSTEM**" is the bonafide work of "**A S DHARSINI (963820106004), K R NANDU (963820106008), M P SHRI VEENA (963820106011) and V VIBISHA (963820106012)** who carried out the project work under our supervision.

SIGNATURE	**SIGNATURE**
Mr. ANANDU. A. S. M.TECH	Mr. VIPIN. V. M.TECH
SUPERVISOR	SUPERVISOR
SCIENTIST/ENGINEER,	SCIENTIST/ENGINEER,
ATMD, CSOG, MVIT,	ATMD, CSOG, MVIT,
Vikram Sarabai Space Centre,	Vikram Sarabai Space Centre,
Thiruvananthapuram-695022.	Thiruvananthapuram-695022.

SIGNATURE	**SIGNATURE**
Dr. S. LEONARD GIBSON MOSES M.E.,PH.D	Dr. S. LEONARD GIBSON MOSES M.E.,PH.D
HEAD OF THE DEPARTMENT,	SUPERVISOR
Department of ECE,	HOD, Department of ECE,
Good Shepherd College of Engineering	Good Shepherd College of Engineering
and Technology,	and Technology,
Maruthamparai - 629101	Maruthamparai – 629101

This report is submitted for the Viva-Voce held on ……………………………

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First of all, we thank Almighty God for His blessings and also for giving us good knowledge and strength in enabling us to finish our project.

We would like to convey our sincere thanks to our Chairman Mr. G. D. MITHRA, an educationalist with PG in MATHEMATICS and a DEGREE in EDUCATION.

We wish to convey our sincere thanks to Dr. ROBIN KUMAR SAMUEL,M.E., Ph.D who provided us approval to carry out our project at VSSC, ISRO.

We are immensely grateful to our Head of the Department of Electronics and Communication Engineering Dr. S. LEONARD GIBSON MOSES M.E., Ph.D, our internal guide who provided us opportunity and guidance to carry out our project at Vikram Sarabai Space Centre, Thiruvananthapuram.

We extend our sincere thanks to Human Resource Development Division (HRDD) of Vikram Sarabai Space Centre, who accepted our request to carry out our B.E Project Work at VSSC, ISRO, Thiruvananthapuram.

We are immensely grateful and convey our sincere and graceful thanks to Mr. ANANDU. A. S. M.Tech, Scientist/Engineer, ATMD, CSOG, MVIT, Vikram Sarabai Space Centre and Mr. VIPIN. V. M.Tech, Scientist/Engineer, ATMD, CSOG, MVIT, VSSC, ISRO, our external guides who provided us complete guidance and all facilities required for the project work as a result of which we are able to complete our project successfully. We are extremely thankful to them.

We extend our sincere thanks to Mr. K. RANJITH. M.TECH, Division Head, ATMD, CSOG, MVIT, VSSC who provided us the facilities required for our project.

Last but not the least, we thank our teachers, parents and friends whose support and blessings helped us finish our project successfully.

ABSTRACT

IRIG-B is a popular and accurate form of time code used world-wide. It is a range synchronization protocol used to perfectly synchronize devices distributed over a range. IRIG Time code Generator generates IRIG time code. For long distance communication, this signal is generally modulated by a 1KHz carrier signal. Development of IRIG Decoder Embedded System provides a way of decoding the IRIG-B signal. The objective of the IRIG Decoder Embedded System is to demodulate the IRIG-B signal from the IRIG Time code generator and obtain the real time information from the demodulated signal. This project is beagle bone-based embedded project where beagle bone samples and decodes the DC level shift IRIG signal. The final output is displayed in the PC. The accuracy of this project is high and the time information is found to be precise and improved performance. This project serves as a low-cost, reliable and less complex project for the decoding of IRIG signal. The results demonstrate that the system runs steadily with high synchronization precision and better anti-interference

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AC	Alternating Current
ADC	Analog to Digital Convertor
AM	Amplitude Modulation
CDT	C/C++ Development Tooling
CMOS	Complementary Metal-Oxide-Semiconductor
CPU	Central Processing Unit
CTI	Cross Trigger Interface
DC	Direct Current
DCLS	Direct Current Level Shift
FPGA	Field Programmable Gate Array
FS	Full Speed
FTP	File Transfer Protocol
GPIO	General Purpose Input Output
GPS	Global Positioning System
HDL	Hardware Description Language
HDMI	High-Definition Multimedia Interface
HS	High Speed
IP	Internet Protocol
IRIG	Inter-Range Instrumentation Group
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LS	Low Speed
MCU	Micro Controller Unit
MMC	Multi Media Card

NTP	Network Time Protocol
Op-Amp	Operational Amplifier
PC	Personal Computer
PMIC	Power Management Integrated Circuit
PRU	Programmable Real-Time Unit
PTP	Precision Time Protocol
PWM	Pulse Width Modulation
RC	Resistor and Capacitor
RJ	Registered Jack
SCP	Secure Copy Protocol
SDRAM	Synchronous Dynamic Random-Access Memory
SFTP	SSH File Transfer Protocol
SoC	System on Chip
SPI	Serial Peripheral Interface
SSH	Secure SHell
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
`UDP	User Datagram Protocol
USB	Universal Serial Bus
UTC	Universal Time Coordinated time
VHDL	VHSIC Hardware Description Language
VHSIC	Very High-Speed Integrated Circuit
WinSCP	Windows Secure Copy

CHAPTER 1

INTRODUCTION

In many applications like aerospace, military and electrical applications, one of the main constraints is time synchronization. Time synchronization is the practice of aligning the system time across network devices to ensure consistency and accuracy. Time synchronized networks enable accurate time stamping by each of the devices on the network. Time synchronization is a necessity in many areas of computing and networking, especially in critical systems where accuracy, reliability and security are paramount and depend on maintaining time consistency between all the objects involved. It is because all machines of a complex system will be synchronized that the system will be able to coordinate actions, ensure the consistency of transactions and maintain data integrity, even though it is made up of distributed systems. IRIG-B is the most common time synchronization protocols used nowadays and it is used for various applications. This signal contains the timing information which is generated from the clocks. The timing information have to be extracted from the IRIG signal by the host device. This project is an embedded based project that was designed to make the process of the decoding of the IRIG signal easy and extract the time information in a cost-effective manner while maintaining its accuracy.

1.1 PROBLEM STATEMENT

IRIG-B is the most common time code. So, there is a need for a cost-effective system to decode this signal.

1.2 OBJECTIVES OF THE PROJECT

The objectives of the project are

- To develop a demodulation circuit for IRIG-B.
- To develop a decoder design in an embedded system.

1.3 OVERVIEW OF THE PROJECT

This project is a beagle-bone based design of the decoder of IRIG-B signal. It aims at decoding the IRIG signal at low cost and less complexity.

1.4 APPLICATIONS

The applications of this project can range from synchronizing substation automation systems to aerospace, military communications, power, industrial automation and control industries, GPS, electrical systems, power grids and marine measurement equipment.

CHAPTER 2

LITERATURE SURVEY

2.1 THE DECODED DESIGN OF IRIG-B

Authors: Aiping Wu, Qingqing Fu, Zhiqiang Gao

Year of Publishing: 2011

Description:

The time synchronization of systems has been implemented by the navigation satellite timing and ranging global position system (GPS) for many years in most countries. In these country, standard time comes from Universal Time Coordinated (UTC) time by using the GPS receiver. As the current GPS receiver outputs standard time with IRIG-B code, in addition, IRIG-B time code carries 27 control code elements itself, it can be implemented control easily. Thus, IRIG-B is widely used in military networks, power grids and other special network. Currently, the terminal decoded design of IRIG-B based on Micro Controller Unit (MCU), with a large number of peripheral circuits to achieve, so the circuit is complex, and once the design is completed, the circuit is difficult to be optimized and modified. It does not match the development trend of electronic technology. In this paper, a new decoding system about IRIG-B is provided which is based on Field Programmable Gate Array (FPGA). The system has some characteristics of small size, high integration, high stability and flexibility.

2.2 DESIGN OF FPGA-BASED IRIG-B CODE DECODER

Author: L Jia

Year of Publishing: 2017

Description:

With the increasing accuracy of measurement and control system, requirements for timing accuracy become higher and higher. As a relatively mature timing method, IRIG-B code, a kind of timing technology, has been widely used in military, aerospace, electric power system. IRIG-B code was developed by the US Inter-Range Instrumentation Group (IRIG) and has become an international standard time code. Our country has formulated the corresponding national standard and military specifications to regulate its pattern format and interface standards. Users use the decoder to obtain time information and the second pulse and the demodulation precision of decoder determines the timing accuracy of B code system. This paper first introduces the formats and interfaces of IRIG-B AC/DC code and the presently development of decoder. Besides, two kinds of AC B code demodulation method are simulated by simulation software. The paper also describes the system design of IRIG-B AC / DC code decoder based on FPGA in detail. According to the design requirements, the selection of chips relating to the filter, signal conditioning circuit, analog digital conversion circuit, control circuit and other modules and the design of the external circuit is completed. It uses VHDL to write the FPGA internal control procedures, introduce DC B code demodulation method and two kind of AC B code demodulation method: digital demodulation and direct demodulation. DC B code demodulation is realized by using the method of pulse width counting. The digital demodulation method of AC code calls the multiplier and filter of the IP core to restore the AC B code to the DC B code. AC code direct demodulation method is to write FPGA program instead of the analog circuit and the AC code is converted into pulse signal processing. Finally, the advantages and disadvantages of the two methods are analyzed. In addition, the method of second pulse delay timing is put forward to improve the accuracy of the decoder second pulse time points. According to the experiment: FPGA-based IRIG-B code

decoder is capable of doing real-time demodulation between alternating current B code and direct current code B code. According to the format display of days, hours, minutes and seconds, using Lab VIEW write PC software, it is capable of storing time information, then sending control command, transforming output mode, it has high accuracy of output decoding pulse, its design requirements achieve various function.

2.3 DESIGN OF TIME SYNCHRONIZATION SYSTEM BASED ON IRIG-B CODE DEMODULATION TECHNOLOGY

Author: C H Wang

Year of Publishing: 2018

Description:

The accuracy of time synchronization, to a large extent, determines the performance of the navigation, positioning and timing process. In addition, the high speed and large capacity information transfer and fusion between different navigation systems are very strict on the time synchronization in the integrated navigation system. Time and frequency standard is one of the most important factors affecting the accuracy of time synchronization. Phase-locked loop, as a kind of phase tracking and locking technology, is an important part of the generation and maintenance of standard time frequency. This paper is design to calibrate the local clock, and improve the local clock stability based on phase-locked loop, which is able to tracking phase. Firstly, this paper analyses the principle of the IRIG-B timing technology and method of IRIG-B demodulator, then establish the method of generating the standard second pulse by decoding the IRIG-B code. In this situation, the paper complete software design of IRIG-B code demodulator. Secondly, a mathematical model of the phase locked loop is modeled and analyzed in detail under comprehensive analysis on it. Loop filter is the most important part of the whole phase locked loop technology. This paper

focuses on the design of the phase locked loop filter, then design and discrete loop filter based on Winner linear filter theory and bilinear transformation theory. Besides direct digital frequency synthesis technology and phase-locked loop technology are adopted to achieve synchronization between local second pulse and standard second pulse. Finally, the paper completed the simulation on development software of the field programmable gate array for the designation. After that, the paper completed the design, fabrication and debugging of the circuit board. Then, this designation was downloaded to the field programmable gate array. It comes out that this design can demodulate the IRIG-B and tracking phase difference well and truly. It was proved that the design is correct, and the results are quite good through hardware verification.

CHAPTER 3

TIME SYNCHRONISATION PROTOCOLS

3.1 INTRODUCTION TO TIME SYNCHRONIZATION PROTOCOLS

Time synchronization is the process of coordinating the time of independent clocks. Time synchronization is important in many areas, especially in real time systems where accuracy, reliability and security are important. Nowadays, three synchronization protocols stand out due to their use and importance:

- NTP (Network Time Protocol)
- PTP (Precision Time Protocol)
- IRIG (Inter-Range Instrumentation Group)

3.2 NTP (NETWORK TIME PROTOCOL)

The Network Time Protocol (NTP) is a synchronization protocol based on UDP which uses port 123. In the NTP protocol, roles are asymmetrical: a computer synchronizes its clock with that of a reference server.

The NTP protocol defines the network architecture to be implemented, messages to be sent between clients and servers, as well as a whole set of algorithms to be used by clients in order for the synchronization to be successful.

The NTP network has a tree-like architecture. The reference clocks (GPS, atomic clock, and so on) are at the top of the architecture, at the 0 level, called stratum 0 in the NTP architecture. They are connected to the stratum 1 server via serial port interfaces. Then, stratum 1 is connected to stratum i+1 to broadcast the timestamp and allow synchronization to continue.

Servers in the same stratum (peers) can be connected to each other to address connection problems with the stratum above or simply to synchronize with each other.

Therefore, NTP servers take on the roles of client and server alternately, depending on the stratum they are in communication with. The standard has a limited architecture of 16 stratums. In most cases, final clients are situated between stratum 3 and 4.

NTP has been designed to provide an accuracy of less than a second.

3.3 PTP (PRECISION TIME PROTOCOL)

Precision Time Protocol (PTP) is an asymmetric synchronization protocol (master-slave) which broadcasts a timestamp across a network to ensure that equipment is synchronized.

Technically, PTP is a master-slave hierarchical protocol. PTP servers situated at the middle of the hierarchy take on the two roles alternately. The master clock at the top of the hierarchy is called grandmaster. It is traditionally connected to a reference clock (atomic clock or GPS for example) and is responsible for broadcasting a timestamping message to the entire network. Nevertheless, the PTP protocol does not force connection to a reference clock and the grandmaster can simply use its internal clock to synchronize all equipment. The grandmaster of a network is selected using an algorithm called the best master clock algorithm (BMCA).

In theory, it has a precision of nanoseconds. But in practice, it has rather of precision of microseconds, which still makes it a far more accurate protocol than the NTP.

3.4 IRIG (INTER-RANGE INSTRUMENTATION GROUP)

The Inter-Range Instrument Group time code (or IRIG for short), is a range of standard time code formats which are used to transfer timing information (hour, minute, second, day, etc.) from a GPS/ Atomic clock to connected slave devices with high accuracy. IRIG-B is a 1 kHz signal which contains 100 bits of data, each transmitted over a 10 ms time frame, taking a total time of 1 second for a complete transmission.

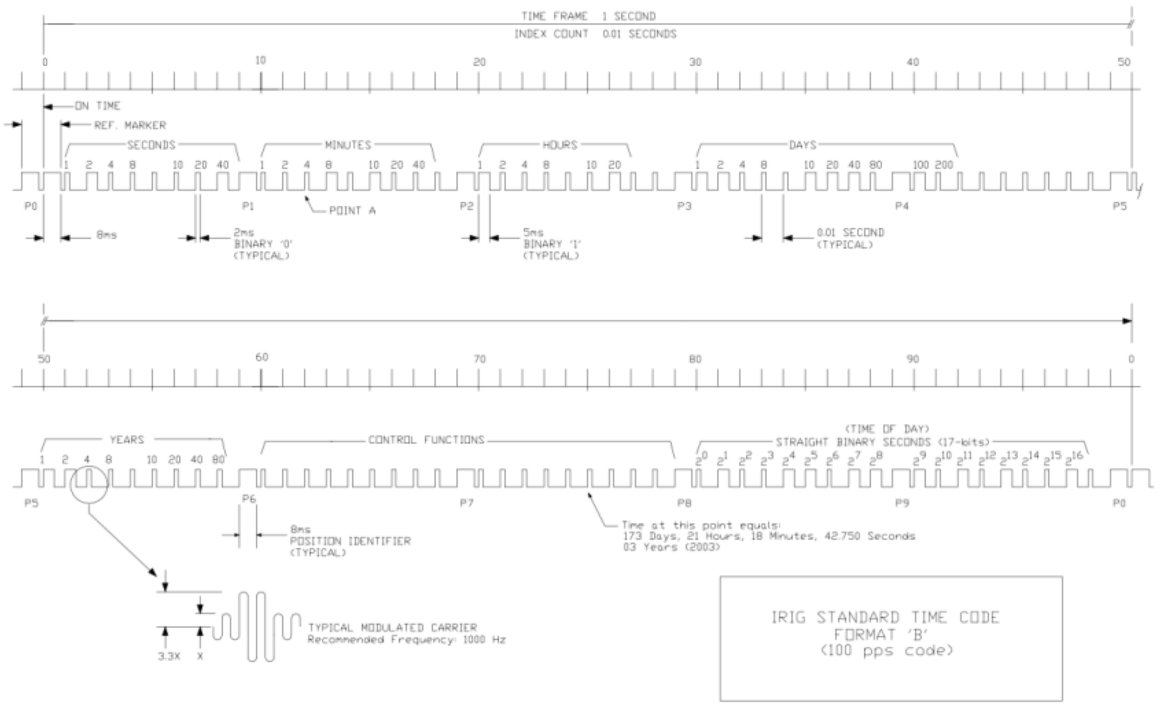

TIME FRAME 1 SECOND
INDEX COUNT 0.01 SECONDS
ON TIME
REF. MARKER
SECONDS
MINUTES
HOURS
DAYS
POINT A
8ms
2ms BINARY '0' (TYPICAL)
5ms BINARY '1' (TYPICAL)
0.01 SECOND (TYPICAL)
YEARS
CONTROL FUNCTIONS
(TIME OF DAY) STRAIGHT BINARY SECONDS (17-bits)
8ms POSITION IDENTIFIER (TYPICAL)
Time at this point equals: 173 Days, 21 Hours, 18 Minutes, 42.750 Seconds 03 Years (2003)
TYPICAL MODULATED CARRIER Recommended Frequency: 1000 Hz
3.3X
X
IRIG STANDARD TIME CODE FORMAT 'B' (100 pps code)

Figure 3.1: IRIG-B Standard Time Code

IRIG-B code is a serial time code of one frame per sec, with velocity of 100 bits/m, and the code is pulse width. It has three kinds of code elements which are L, H, P. Figure.3.2 shows their pulse width are 2ms, 5ms, 8ms respectively. The first frame of IRIG-B code is composed by position flag (p code) and reference element code (p code). The flow code contains time information about data, hour, minute, second and control information. Time information uses BCD to express. Back of time information, there have 27-bit control code and 17-bit second information, The 17-bit second information shows seconds started from 00:00:00 today. The frame speed of IRIG-B code is 1 frame per second; a frame has 10 words and each word has 10 bits.

Besides, each bit begins with high level, and has three kinds of the duration time: 2ms (such as binary “0” code and index flag), 5ms (binary “1” code) and 8ms (such as reference linear code, the first bit of first word when start in each second, position flag P0~P9, the tenth bit of each word). The first word transmits seconds information, the second word transmits minute information and third transmits hour information, the four and fifthly transmit day, and count begin. In addition, there are special flag control code of 3bits to mark previous station and branch station in the eighth and tenth word.

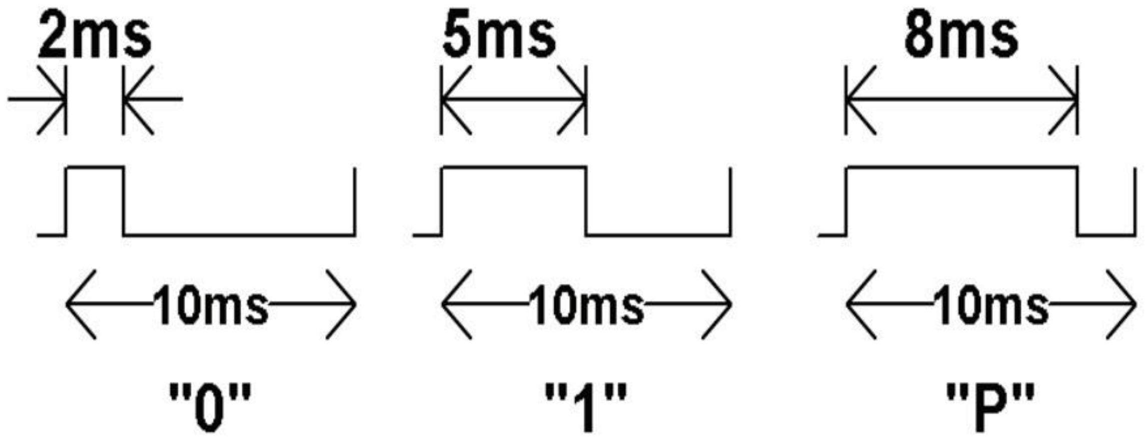


Figure 3.2: IRIG Code Element

IRIG-B has three different modulation types;

1) Direct Current Level Shift (DCLS) – typically this is a 0-5 V DC pulse width modulated signal, where the different pulse widths represent coded data. This is the most common modulation method used today due to its high accuracy (< 100 ns at the port).

2) Amplitude Modulated (AM) – modulated with a 1 kHz sine wave carrier signal with a 3:1 ratio, this signal has no DC content. This made AM popular in the past as it allowed the signal to be transmitted over long distances.

3) Modified Manchester Modulation – Modified Manchester modulation is the least common modulation type for IRIG-B. Using a 1 kHz square wave with phase modulation rather than DC level shift, this signal contains no DC bias.

Figure 3.3 shows the difference between DC level shift, amplitude modulated and Manchester Modified IRIG signal.

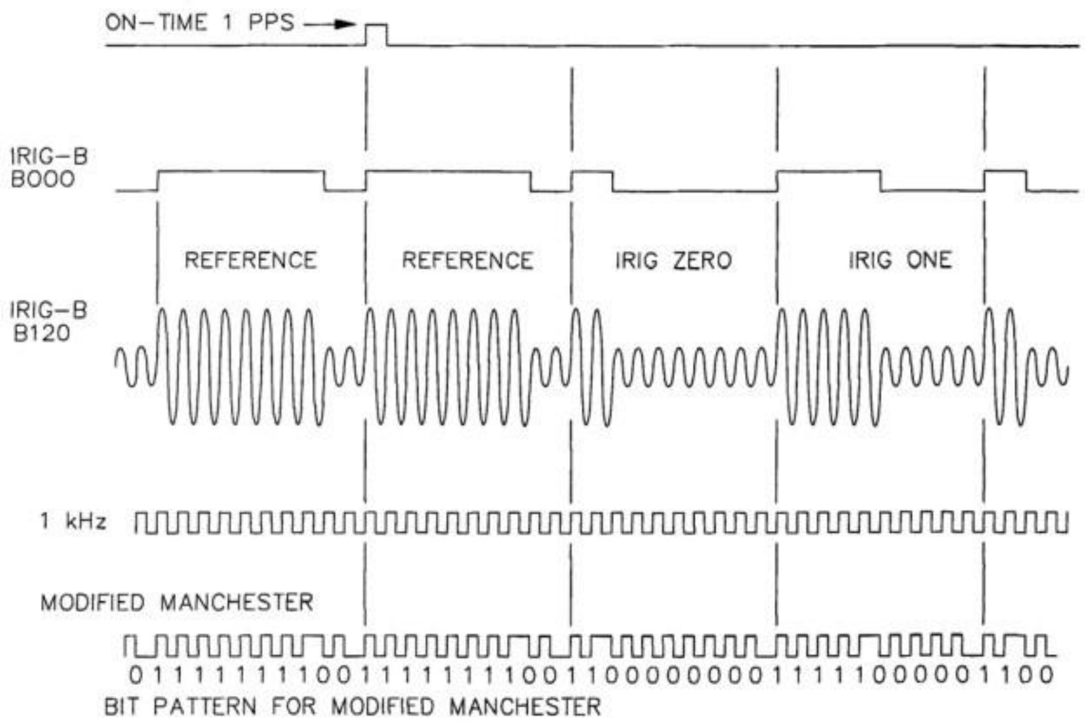


Figure 3.3: IRIG-B coding comparisons: level shift, 1 kHz amplitude modulated, and Modified Manchester

CHAPTER 4

METHODOLOGY

4.1 EXISTING SYSTEM

In the existing system, the decoding system of IRIG-B is provided which is based on Field Programmable Gate Array (FPGA)

4.1.1 DISADVANTAGES OF EXISTING SYSTEM

Since the existing system is designed with the help of FPGAs, it has the following disadvantages.

- Designing for FPGAs can be complex and time-consuming.
- FPGA designs often require expertise in hardware description languages (HDLs) such as Verilog or VHDL.
- FPGAs are expensive.
- FPGAs typically consume more power.

4.2 PROPOSED SYSTEM

In proposed system, the embedded system used is BeagleBone Black Board and demodulator circuit is designed using circuit with simple components.

4.2.1 BLOCK DIAGRAM OF THE PROPOSED SYSTEM

In proposed system, the amplitude modulated IRIG signal which is generated from the IRIG time code generator is demodulated using demodulator circuit and then it is decoded by using BeagleBone and the timing information is displayed in the PC. The block diagram of the proposed system is given in Figure 4.1

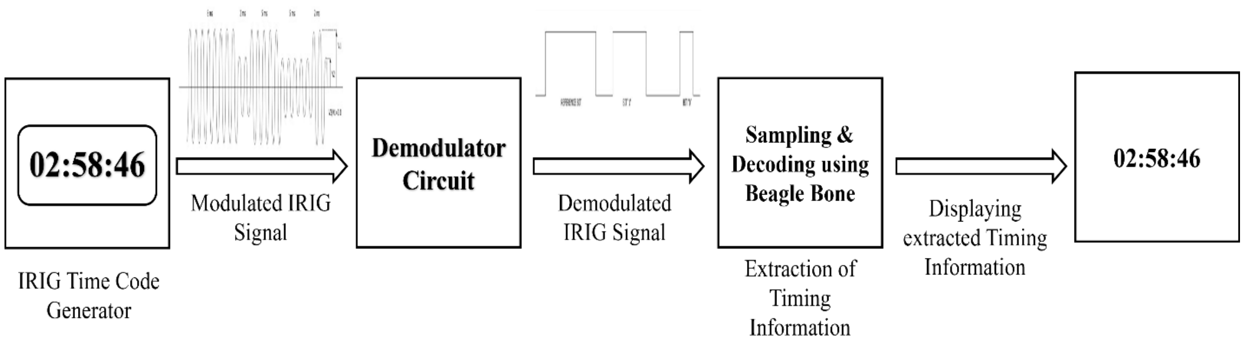

02:58:46
IRIG Time Code Generator
Modulated IRIG Signal
Demodulator Circuit
Demodulated IRIG Signal
Sampling & Decoding using Beagle Bone
Extraction of Timing Information
Displaying extracted Timing Information
02:58:46

Figure 4.1 Block Diagram of the Proposed System

The process of extraction of timing information from the IRIG-B signal can be divided into two phases. They are

- Demodulator Circuit Design
- Beagle Bone Programming

4.3 DEMODULATOR CIRCUIT DESIGN

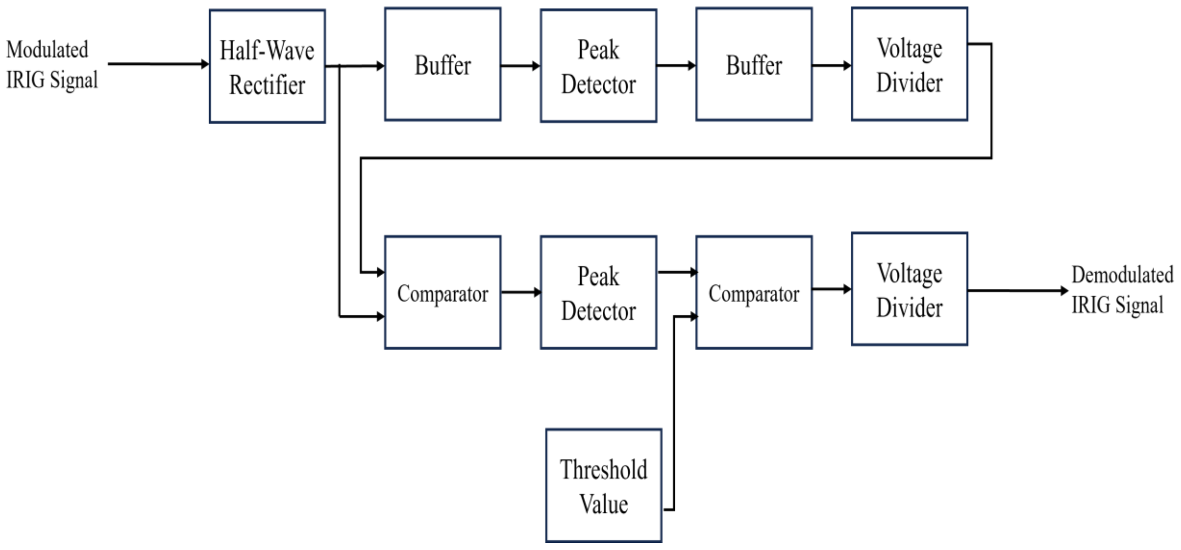


Figure 4.2 Block Diagram of the Demodulator Design

4.4 FLOW CHART FOR BEAGLE BONE PROGRAM

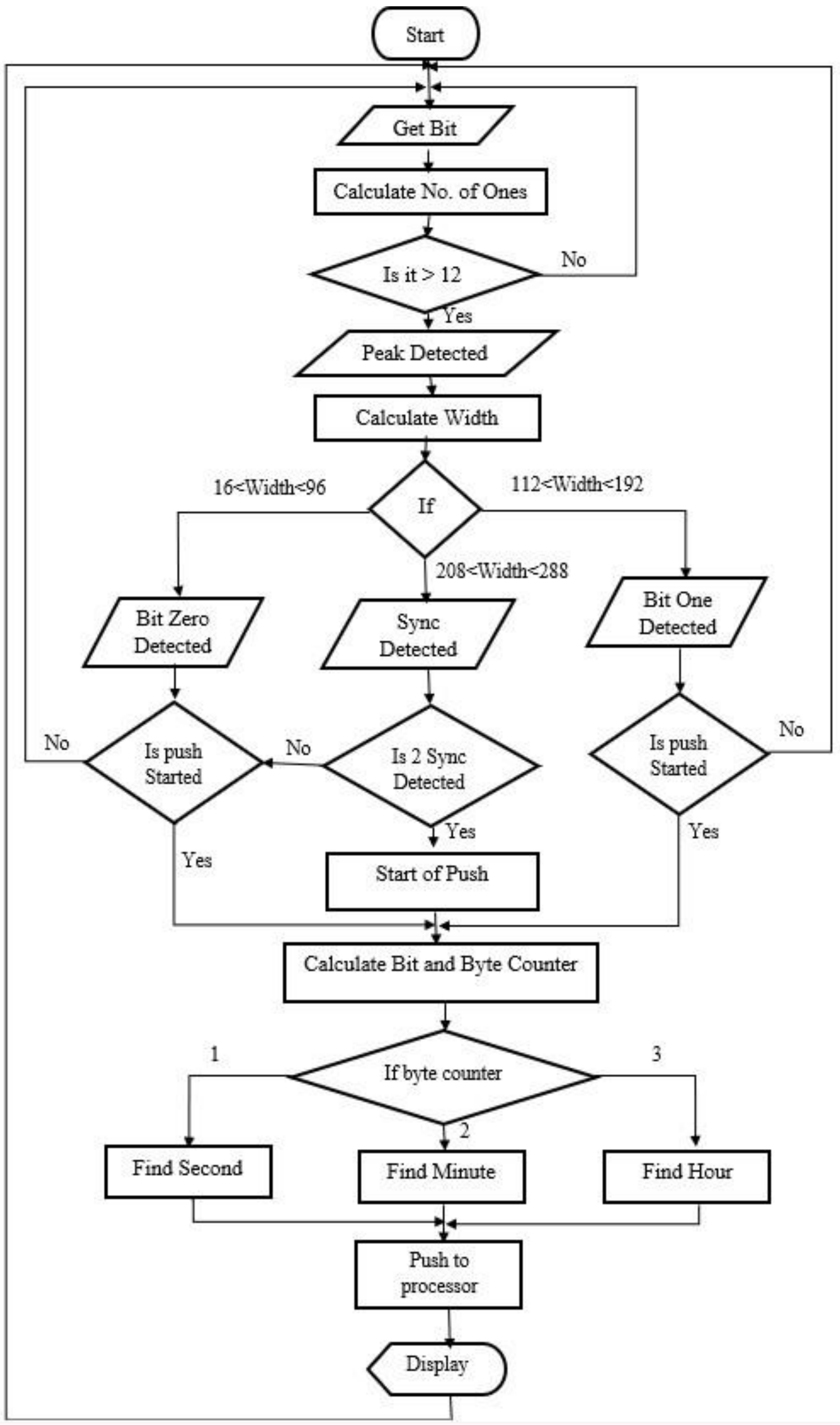


Figure 4.3 Flow chart of the program

4.5. DATAFLOW DIAGRAM

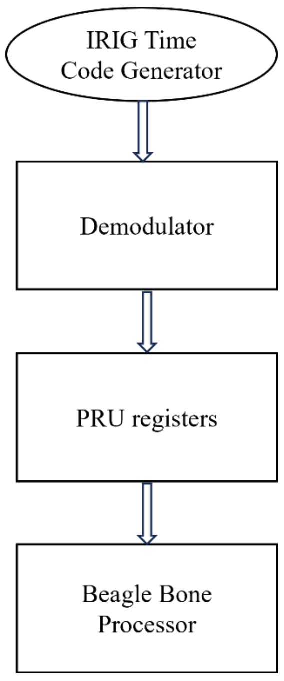


Figure 4.4 Dataflow Diagram

CHAPTER 5

HARDWARE AND SOFTWARE REQUIREMENTS

5.1 HARDWARE REQUIREMENTS

Hardware requirements refer to the necessary physical components or equipment needed to run a particular software application, operate a device, or execute a task effectively. These requirements are the components needed for electronic projects. The hardware requirements needed for our project are listed below.

- IRIG Time Code Generator
- Components for demodulator circuit
- BeagleBone Black
- PC
- USB Cable

5.1.1 IRIG TIME CODE GENERATOR

An IRIG time code generator usually consists of a hardware module or integrated circuit capable of generating the IRIG time code signal according to the specified format and timing requirements. It may include components such as a real-time clock (RTC), a microcontroller or FPGA for signal processing and encoding, and output interfaces such as serial ports or specialized connectors for transmitting the generated time code signal to other devices.

5.1.2 COMPONENTS FOR DEMODULATOR CIRCUIT

The components required for demodulator circuit are simple components like resistors, capacitors, diodes, op-amps, wires and voltage supply and additionally we may require oscilloscope and multimeter for testing.

5.1.3 BEAGLEBONE BLACK

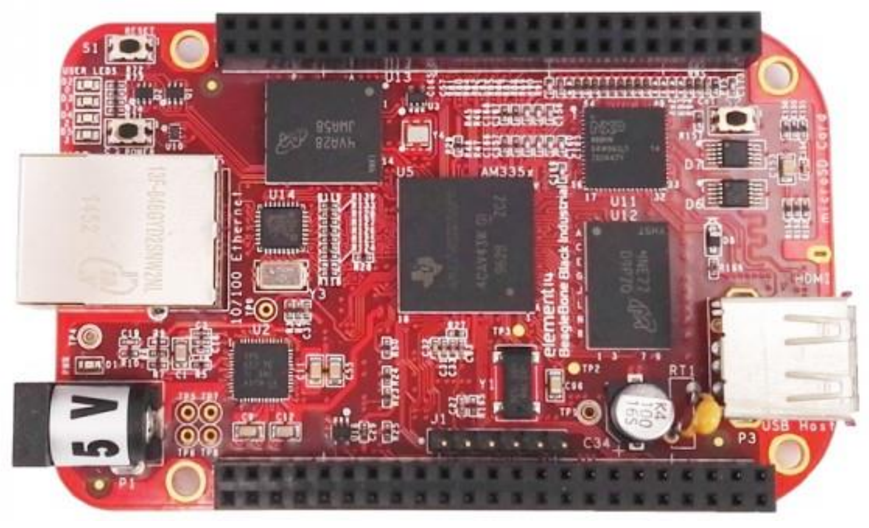


Figure 5.1 BeagleBone Black

BeagleBone black is a low-cost, community-supported development platform meant for industrial applications in harsh temperature environments. It boots Linux in under 10 seconds and get started on development in less than 5 minutes with just a single USB cable. The default operating system provided by BeagleBoard.org, the organization behind BeagleBone, is usually Debian-based, often specifically a version of Debian called "BeagleBoard Debian" or "Bonescript". A brief description of the Beagle Bone features is listed below.

- Processor: TI Sitara AM3358BZCZA100, 1GHz, 2000 MIPS 1 GHz ARM®Cortex™-A8
- SGX530 Graphics Engine
- Programmable Real-Time Unit Subsystem
- Operating temperature can span from -40C to +85C
- Memory SDRAM: 512MB DDR3L 800MHZ
- Onboard Flash: 4GB, 8bit Embedded MMC (eMMC)
- SD/MMC Connector for microSD

- Power management: TPS65217C PMIC is used along with a separate LDO to provide power to the system
- Debug Support: Optional Onboard 20-pin CTI JTAG, Serial Header
- Power Source mini-USB USB or DC Jack
- 5VDC External Via Expansion Header
- Connectivity High speed USB 2.0 Client port: Access to USB0, Client mode via mini-USB
    - High Speed USB 2.0 Host port: Access to USB1, Type A Socket, 500mA LS/FS/HS
    - Serial Port: UART0 access via 6 pin 3.3V TTL Header. Header is populated
    - 10/100M Ethernet (RJ45)
- User Input / Output Reset Button
    - Boot Button
    - Power Button
    - LED power indicator
    - 4 user configurable LEDs
- Video/Audio Interfaces HDMI D type interface
    - LCD interface
    - Stereo audio over HDMI interface
- Expansion Interfaces LCD, UART, eMMC
    - ADC, I2C, SPI, PWM

The Programmable Real-Time Units (PRUs) in BeagleBone are small, low-power microcontrollers embedded within the system-on-chip (SoC) of the BeagleBone series of single-board computers. These PRUs offer real-time processing capabilities, which can be utilized to offload real-time tasks from the main CPU, enhancing the system's performance in real-time applications. Since our project is a real time application, beagle bone is more suitable than many other embedded systems because of the presence of PRUs.

5.1.4 PC

A PC is required for the beagle bone project with required system requirements for the project for writing programs and displaying the output.

5.1.5 USB CABLE

The most common way to connect a BeagleBone to a computer is via a USB cable. BeagleBone boards typically feature a mini or micro-USB port that allows them to be connected to a computer. This connection can be used for power, programming, and data transfer**.**

5.2 SOFTWARE REQUIREMENTS

Software requirements refer to the software needed to be installed for the specific task in a project design to serve as a foundation for the design, implementation, and testing of the project. The software requirements needed for our project are listed below.

- PuTTY
- WinSCP

5.2.1 PUTTY

PuTTY is a free and open-source terminal emulator, serial console, and network file transfer application. It is commonly used on Windows computers to establish SSH (Secure Shell), Telnet, and serial connections to remote devices, such as servers, routers, switches, and embedded systems like the BeagleBone.

5.2.2 WINSCP

WinSCP (Windows Secure Copy) is a popular open-source SFTP (SSH File Transfer Protocol), FTP (File Transfer Protocol), and SCP (Secure Copy Protocol) client for Windows. It allows users to securely transfer files between a local computer and a remote server or device over SSH (Secure Shell) or FTP connections. WinSCP can integrate with PuTTY.

5.3 SYSTEM REQUIREMENTS

System requirements specify the platform, operating system, databases, libraries, and other dependencies needed to deploy and run the project successfully.

The operating system of the PC can be any Linux-based operating system or any version of Windows higher than Windows 7.

CHAPTER 6

IRIG DEMODULATOR CIRCUITS

6.1 INTRODUCTION

The need for the design of the demodulator circuit is to demodulate the amplitude modulated IRIG-B signal into the DC level shift pulse width modulated IRIG signal. The design of the demodulator circuit is quite challenging and came out with two demodulator circuit out of which, one is selected because of its advantages.

6.2 CHALLENGES IN DESIGNING DEMODULATOR CIRCUIT

- The demodulated output must be accurate with no delay.
- There must be no internal noise and the distortions must be low.

6.3 CIRCUIT NUMBER 1

6.3.1 REQUIREMENTS

Sl. No	**Components**	**Quantity**
1.	1N4007 Diode	1
2.	Resistors (330 Ω,750 Ω,100 Ω)	330 Ω – 2 750 Ω – 1 100 Ω - 1
3.	Capacitors (1µF, 100µF)	1µF – 1 100µF - 1
4.	Op-Amp IC741	1
5.	Power Supply	1
6.	Oscilloscope	1
7.	Wires	few

Table 6.1 Requirements of the circuit 1

6.3.2 CIRCUIT DIAGRAM OF CIRCUIT 1

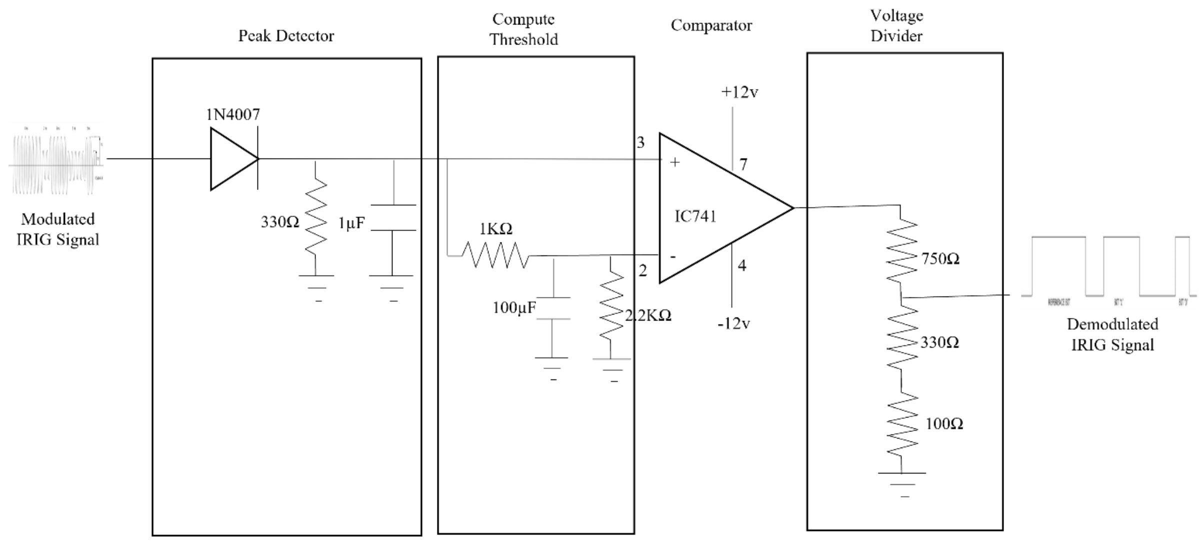


Figure 6.1 Circuit diagram of circuit 1

6.3.3 CIRCUIT OPERATION OF CIRCUIT 1

This circuit consists of four phases. They are

1) Peak Detector
2) Threshold Computer
3) Comparator
4) Voltage Divider

1) Peak Detector

The diode**,** capacitor and resistor form the peak detector. Since the peak detector contains capacitor, it charges to the maximum voltage level of the input and discharges later. Here, there is resistor to provide the discharge path. Hence peaks are visible.

The output of the peak detector is given in Figure 11.1.

From the figure, it is to be noted that the threshold value is 160 mV. Hence, we have to generate this threshold value to compare it with the output of peak detector.

2) Threshold Computer

This stage is used to generate the threshold value of 160 mV. The RC circuit re-uses the two resistors and adds a capacitor to perform threshold computing.

The output of the threshold computer is given in Figure 11.2. The output will be a DC voltage of 160 mV.

3) Comparator

A comparator is an electronic circuit or device used to compare the magnitudes of two voltages or currents and determine which one is larger. It essentially produces an output based on whether one input is higher or lower than the other. Here op-amp serves as comparator and finds whether the peak detector voltage which is supplied at the non-inverting terminal is greater than the threshold voltage i.e., 160mV which is supplied at the inverting terminal. If it is greater than 160 mV, it returns 1, else it returns 0.

The output of the comparator is given in Figure 11.3.

From the figure, it clearly shows that we obtained the required signal of 10.6V. But beagle bone is designed to operate at the maximum voltage of 5V. The voltage of the output signal must be reduced.

4) Voltage Divider

Voltage divider circuit is used to reduce the voltage of the comparator output. It is also to be noted that the voltage must be greater than 3.3V because in BeagleBone, input voltage must exceed the CMOS level of 3.3V to guarantee reliable digital signal processing and prevent potential malfunctions.

Here the output of the Voltage Divider is computed by,

$$V_o = V_i \frac{R2}{R1+R2}$$

$$V_o = 10.6 \ \frac{330+100}{750+330+100}$$

$$V_o = 3.86 \text{ V}$$

Practically, the voltage obtained is 3.600 V. The output of the Voltage Divider is given in Figure 11.4.

6.3.4 DISADVANTAGES OF CIRCUIT 1

- Starting and ending points of the peak are not accurate.
- At some peaks, there arises some grooves which may result in incorrect computation of the timing information. The grooves can be seen in Figure 11.5.

6.4 CIRCUIT NUMBER 2

6.4.1 REQUIREMENTS

Sl. No	Components	Quantity
1.	1N4007 Diode	4
2.	Resistors (330 Ω,1KΩ,10KΩ)	330 Ω – 6 1KΩ – 2 10KΩ - 1
3.	Capacitors (1μF, 10μF)	1μF – 1 10μF - 1
4.	Op-Amp IC741	4
5.	Power Supply	1
6.	Oscilloscope	1
7.	Wires	few

Table 6.2 Requirements of the circuit 2

6.4.2 CIRCUIT DIAGRAM OF CIRCUIT 2

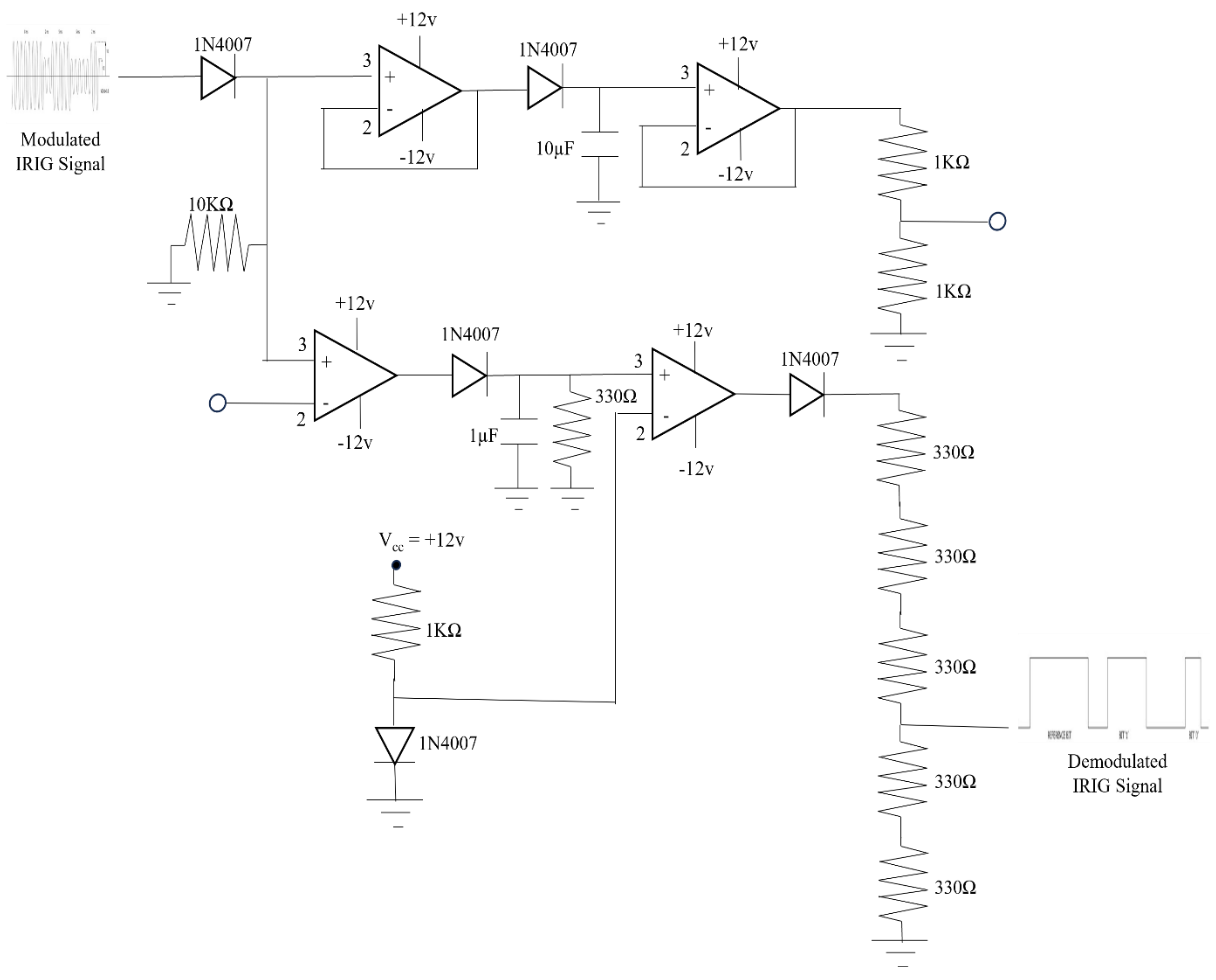


Figure 6.2 Circuit diagram of circuit 2

6.4.3 CIRCUIT OPERATION OF CIRCUIT 2

The circuit operation of the second circuit can be divided into 8 stages. They are

1) Half Wave Rectifier Stage

2) Buffer Stage

3) Threshold Computer Stage

i) Peak Detector

ii) Buffer Stage

iii) Voltage Divider Stage

4) Comparator

5) Peak Detector

6) Threshold Value

7) Comparator

8) Voltage Divider

1) Half Wave Rectifier Stage

A half-wave rectifier is a type of rectifier circuit used to convert alternating current (AC) into direct current (DC). It does so by allowing only one half of the AC input waveform to pass through, resulting in a pulsating DC output. Here diode acts as a half wave rectifier.

The output of the Half Wave Rectifier at point 1 is given in Figure 11.6. From the figure, it is to be noted that the threshold value is 600 mV. Hence, we have to generate the threshold value to compare it with the output of the Half Wave Rectifier.

2) Buffer Stage

A buffer is a small amplifier circuit that converts the high impedance input to low impedance output and isolate one circuit from another while maintaining the signal integrity. Here the impedance of the output of the half wave rectifier is high. Hence to convert it to low impedance, an op-amp is used with negative feedback for buffer before generating threshold value. The output of the buffer will be same as half wave rectifier output.

The output of the Buffer at point 2 is given in Figure 11.7.

3) Threshold Computer Stage

This stage is used to generate the threshold value of 600 mV. It consists of three stages. They are Peak Detector, Buffer and Voltage divider.

3) i) Peak Detector

The diode and the capacitor form the peak detector. Since the peak detector contains capacitor, it charges to the maximum voltage level of the input and discharges later. But here, there is no resistor to provide the discharge path. Hence the output of the peak detector will be a DC voltage of the threshold value which is the average of the two signal levels, i.e., $\frac{V0+V1}{2}$, where V_0 and V_1 are the maximum and minimum voltage levels of the buffer output.

Here the output of the peak detector is computed by,

$$V = \frac{V0+V1}{2}$$

$$V = \frac{2.24+0.160}{2}$$

$$V = 1.2\ V$$

The output of the Peak Detector at point 3 is given in Figure 11.8.

3) ii) Buffer Stage

Here also, the impedance of the output of the peak detector is high. Hence to convert it to low impedance, an op-amp is used with negative feedback for buffer. The output of the buffer will be same as peak detector output.

The output of the Buffer at point 4 is given in Figure 11.9.

3) **iii) Voltage Divider Stage**

Since the output of the buffer is 1.2V, when it is divided into half, we get the required threshold value of 600 mV. So, two 1KΩ resistors are connected in parallel and voltage is divided into half.

Here the output of the Voltage Divider is computed by,

$$V_o = V_i \frac{R2}{R1+R2}$$

$$V_o = 1.2 \ \frac{1K}{1K+1K}$$

$$V_o = 600 \text{ mV}$$

The output of the Voltage Divider at point 5 is given in Figure 11.10.

4) Comparator

Here op-amp serves as comparator. The output of the half wave rectifier is fed to non-inverting terminal and the threshold voltage is fed to the inverting terminal of the op-amp. The comparator finds whether the half wave rectifier voltage is greater than 600mV. If it is greater than 600 mV, it returns 1, else it returns 0.

The output of the comparator at point 6 is given in Figure 11.11.

5) Peak Detector

The diode**,** capacitor and resistor form the peak detector. Since the peak detector contains capacitor, it charges to the maximum voltage level of the input and discharges later. Here, there is resistor to provide the discharge path. Hence peaks are visible.

The output of the peak detector at point 7 is given in Figure 11.12.

From the figure, it is to be noted that the threshold value is 700 mV. Hence, the threshold value should be generated to compare with the output of peak detector.

6) Threshold Value

The voltage drop across a diode, often referred to as the forward voltage drop, typically ranges from about 0.6 to 0.7 volts for silicon diodes under normal operating conditions. This voltage drop occurs when the diode is forward-biased, meaning the voltage at the anode (positive terminal) is higher than the voltage at the cathode (negative terminal). This idea is utilized to generate the threshold value of 700 mV. A diode is forward biased with supply voltage of 12 V and thus 0.7 V is generated and it is a DC voltage.

The output of the diode at point 8 is given in Figure 11.13.

7) Comparator

Here also op-amp serves as comparator. The output of the peak detector is fed to the non-inverting terminal and the threshold value is fed to the inverting terminal of the op-amp. The comparator finds whether the peak detector voltage is greater than 700 mV. If it is greater than 700 mV, it returns 1, else it returns 0.

The output of the comparator at point 9 is given in Figure 11.14.

From the figure, it clearly shows that we obtained the required signal of 11.2 V. But BeagleBone is designed to operate at the maximum voltage of 5V, The voltage of the output signal must be reduced.

8) Voltage Divider

Voltage divider circuit is used to reduce the voltage of the comparator output. Since in beagle bone, input voltage must exceed the CMOS level of 3.3, the output of the voltage divider must be greater than 3.3V but less than 5V.

Here the output of the Voltage Divider is computed by,

$$V_o = V_i \frac{R2}{R1+R2}$$

$$V_o = 11.2 \frac{330X2}{330X5}$$

$$V_o = 4.48\ V$$

Practically, the voltage obtained is 3.88 V. The output of the Voltage Divider at point 10 is given in Figure 11.15.

6.4.4 ADVANTAGES

- The starting and ending point of the peak are accurate.
- There are no grooves in the output.
- Clear and precise output than circuit 1.

Since the circuit 2 has more advantages than circuit 1, the second circuit is chosen as the demodulator circuit for our project.

CHAPTER 7

BEAGLEBONE PROGRAMMING PROCEDURE

7.1 STEPS IN BEAGLE BONE PROGRAMMING

STEP 1 : ESTABLISHING CONNECTIONS TO BEAGLE BONE

PuTTY allows you to establish SSH (Secure Shell) connections to the BeagleBone. The following commands are executed in the command prompt to establish connection between the computer and the Beagle Bone.

- ping 192.168.7.2
- ssh debian@192.168.7.2
- Type the password.

Connection is now established. The Debian based Linux distribution of the Beagle Bone can now be accessed.

STEP 2 : LOGIN TO WINSCP

WinSCP includes a built-in text editor which allows us to write and edit files like assembly, C codes, script files, etc. Hence login into it using Beagle Bone IP address.

STEP 3 : SET DATE AND TIME

Setting the date and time correctly on a BeagleBone is important for timestamping, authentication and security, synchronization, scheduled tasks and troubleshooting: Setting the date and time correctly ensures the proper functioning of various system processes, security measures, and network interactions on the BeagleBone device. Setting the date and time avoids clock skew error.

So, date and time is set by using the following command

- sudo date -s 2024-01-01T10:00:00

Replace it with present time.

STEP 4 : PIN CONFIGURATION

To enable I/O functionality of the GPIO pins in the beagle bone, the input and output pins must be configured. The script for pin configuration is written in a script file named “configpin.sh” where the input pin is 28th pin of P9 header (pr1_pru0_pru_r31_3) and the output pin is 27th pin of P9 header (pr1_pru0_pru_r30_5).

Execute the following commands to configure I/O pins.

- sudo ./config.sh
- sudo ./pin_setup.sh

STEP 5 : ASSEMBLY CODE

To extract the timing information from the demodulated IRIG signal, a program is written in assembly language, which is straightforward when designing timing-critical programs. The assembly code has the extension .asm. This assembly code uses registers which store information according to time.

STEP 6 : C CODE

When the program is written in assembly language, it still requires a C container program. When the assembly code generates interrupt, the timing information stored in the processor is displayed on the screen.

STEP 7 : EXECUTION

The following commands are executed for the execution of the program.

- make clean
- make
- make run

1) MAKE CLEAN:

Sometimes during the compilation process, temporary files such as Object files (.o files), executable files (e.g., binaries) or other artifacts are generated. “make clean” command removes these files to leave the project directory in a clean state.

2) MAKE:

The primary purpose of the make command is to automate the process of compiling source code into executable programs or libraries. It reads a file called Makefile, which contains instructions on how to compile the project, and then executes the necessary compiler commands to build the software.

3) MAKE RUN:

The primary purpose of “make run” command is to execute the compiled program or script after it has been built. It typically involves running a command to start the program directly from the command line. It generates binary file, object file and map file inside a folder called ‘build’. After the generation of temporary files, the output is displayed on the screen.

7.2 DETAILED DESCRIPTION OF THE ASSEMBLY CODE

The detailed procedure of the assembly language program is mentioned below.

STEP 1: FIX SAMPLING RATE

In beagle bone, each instruction is executed in 5ns. Hence when number of instructions changes continuously, the sampling rate also changes. Therefore, the first step is to fix the sampling rate by fixing the number of instructions.

According to sampling theorem, the sampling frequency must be strictly greater than or equal to two times the maximum frequency of the signal. Here, the frequency of the carrier signal used is 1KHz. Hence the sampling frequency must be at least 2KHz. Here the sampling frequency used is 32KHz for greater accuracy.

If the sampling frequency is 32KHz, the number of instructions must be

$$\text{Number of instructions} = \frac{1}{32\text{KHz}X5\text{ns}}$$

Number of instructions = 6250 instructions

STEP 2: ASSIGN NAMES TO REGISTERS

For easy access, names are assigned to registers. Here, 17 registers are used. Also, main function is declared here. Other functions are written below it.

STEP 3: FUNCTIONS:

1) GETBIT

The DATA register is left shifted to get the next bit. Since pr1_pru0_pru_r31_3 is configured as input, the 10th bit of the r31 register will be set when the voltage is high. So, when it is set, set the last bit of DATA register. It has a total of 4 instructions.

2) COUNTSTART

This function is used to count the number of ones in the last 16 bits of the DATA register. The value of number of ones is stored in COUNT register. It has a total of 67 instructions.

3) CHECKONESFORWIDTH

This function calculates the width of the peak. When the number of ones is greater than 12, width is calculated by incrementing DATAWIDTH register. When the DATAWIDTH is not equal to zero, ISPEAK flag is set which denotes a peak is detected. It has a total of 6 instructions.

4) WHICHPEAK

To find which peak is detected, first width of sync, bit one and bit zero must be calculated. The width obtained from calculation may not be equal to the actual width of the demodulated signal if there are errors. So, window is set instead of single value.

Width of sync = 32KHz x 8ms

= 256

Hence, window of sync is set to (208,288).

Width of bit one = 32KHz x 5ms

= 160

Hence, window of bit one is set to (112,192).

Width of bit zero = 32KHz x 2ms

= 64

Hence, window of bit zero is set to (16,96).

Move the CURRENTDATA to PREVDATA and CURRENTDATA is made to zero. The aim of this function is to detect the CURRENTDATA. The ISPEAK flag is made to zero. If the DATAWIDTH is greater than 208, check if it is less than 288. If it is so, sync is detected and CURRENTDATA is made to 1, else no bit is detected. Else if the DATAWIDTH is greater than 112, check if it is less than 192. If it is so, bit one is detected and CURRENTDATA is made to 2, else no bit is detected. Else if the DATAWIDTH is greater than 16, check if it is less than 96. If it is so, bit zero is detected and CURRENTDATA is made to 2, else no bit is detected. It has a total of 11 instructions.

5) MAINTAINSAMPLINGRATE

This function is to maintain the number of instructions in the program according to sampling rate. There is a total of 102 instructions. Hence remaining 6148 instructions are necessary and are provided in this function.

6) CHECKTWOSYNC

A frame starts when two syncs are detected. Hence, this function detects two syncs by checking whether the PREVDATA and CURRENTDATA are syncs. It has a total of 6 instructions.

7) CALCULATEBITANDBYTECOUNTER

This function increments the BITCOUNTER and BYTECOUNTER according to the IRIG signal. When the BITCOUNTER is 10, the BYTECOUNTER is copied to the processor and then interrupt is generated to display the timing information on the screen. It has a total of 12 instructions.

8) FORBYTEONE

If the BYTECOUNTER is 1, then seconds are calculated. When the CURRENTDATA is 2, the 8th bit of the SECOND register is set and then right-shifted by one unit. Then the contents of the SECOND register are copied to processor. It has a total of 9 instructions.

9) FORBYTETWO

If the BYTECOUNTER is 2, then minutes are calculated. When the CURRENTDATA is 2, the 8th bit of the MINUTE register is set and then right-shifted by one unit. Then the contents of the MINUTE register are copied to processor. It has a total of 9 instructions.

10) FORBYTETHREE

If the BYTECOUNTER is 3, then hours are calculated. When the CURRENTDATA is 2, the 8th bit of the HOUR register is set and then right-shifted by one unit. Then the contents of the HOUR register is copied to processor. It has a total of 9 instructions.

STEP 4: MAIN

The global main which is declared after the register assignment calls the main function. The main function initializes the variables and executes the other functions in the order specified.

CHAPTER 8

RESULTS AND FINDINGS

8.1 RESULTS

When the commands for execution is executed, the assembly code which is converted to object file is executed. When the assembly code generates the interrupt, the hour, minute and second information which is stored in the processor at that point is displayed in the screen as “CDT PRU time”. Thus, the timing information is successfully extracted and displayed.

The final output of display is shown in Figure 11.16.

8.2 FINDINGS

The findings of this project are given briefly below.

- The accuracy of the time is high as expected.
- Steady and precise.
- Clear display and adapt to change in time immediately.

8.2.1 FINDINGS OF ACCURACY

The time delay of the start and end of the demodulated IRIG signal from the modulated IRIG signal is observed. The time delay is found to be 0.2 ms. The length of the IRIG code element is 10 ms. Hence accuracy is calculated as follows.

$$\text{Accuracy} = \frac{10\text{ms} - 0.2\text{ms}}{10\text{ms}} \text{ X } 100\%$$

$$= 98\%$$

Thus, accuracy is found to be 98% as expected to be high.

CHAPTER 9

CONCLUSION

9.1 CONCLUSION

The realization of decoding IRIG-B using BeagleBone is successfully demonstrated in our project. The results of experiment shows that the system is steady and precision is very high. It outputs the precise synchronization pulse PPS signal and time signals. It is also convenient to monitor and manage the remote facilitate. The system has characteristics of small size, low power, high integration, low cost and easy to upgrade.

9.2 APPLICATIONS

1) AEROSPACE

This project provides interoperability in aeronautical and aerospace telemetry applications.

2) MILITARY COMMUNICATIONS AND RESCUE OPERATIONS

Defense and emergency services rely on various synchronization solutions for their communication networks and specific applications.

IRIG-B interface is more specific to air navigation requirements. It is used by Air forces for various strategic applications or air navigation and missile control.

2156 range of time servers are qualified by several air forces of several countries for its broad choice of interfaces including IRIG-B that are fully compliant with military navigation systems.

CXR synchronization systems embeds ultra-low phase noise oscillators that bring extremely high accuracy to applications and satellite communications.

In these areas, this project can be effective because of the use of IRIG.

3) SUBSTATIONS

Traditional substation design uses a separate timing bus, with a time-synchronization signal, such as IRIG, to synchronize IED clocks in the substation. Hence this project can be deployed in substations for time synchronization.

4) AUTOMATION AND CONTROL INDUSTRIES

The IRIG is an industrial protocol that is used for automation applications pertaining to control, safety, synchronization, and motion. In such industries, this project is very useful because of its low cost and high accuracy.

5) GPS

Global Positioning System (GPS) makes use of high-accuracy synchronized time code signals. This needs extremely high accurate time synchronization for its usage and hence this project can be used in GPS to be designed for simple and reliable installation in the harsh environment of electric power facilities.

6) POWER AND ELECTRICAL SYSTEMS

This project can be employed to facilitate the synchronization of devices like breakers, relays, switches and other components that play roles in power generation and distribution.

8) MARINE MEASUREMENT EQUIPMENT

In marine navigations which uses low frequency systems that mostly operates at 1000Hz and for other measurements it uses devices that uses IRIG for its time synchronization. In such systems, this project can be deployed for better clarity.

9.3 SCOPE OF THE PROJECT

This project covers the decoding of the IRIG signal in an effective manner. Hence this can be implemented in the places where IRIG signal are being utilized and the decoding is necessary.

CHAPTER 10

FUTURE ENHANCEMENT

9.1 FUTURE WORK

This project is found to be tested and examined successfully and found to be effective. Future change can be brought to the processing by simple modification in the code. This project implements the decoding of hours, minutes and seconds information only. As a future implementation, days can be decoded. A design to decode count and hold status can also be added in future.

APPENDICES

APPENDIX 1

SCREENSHOTS

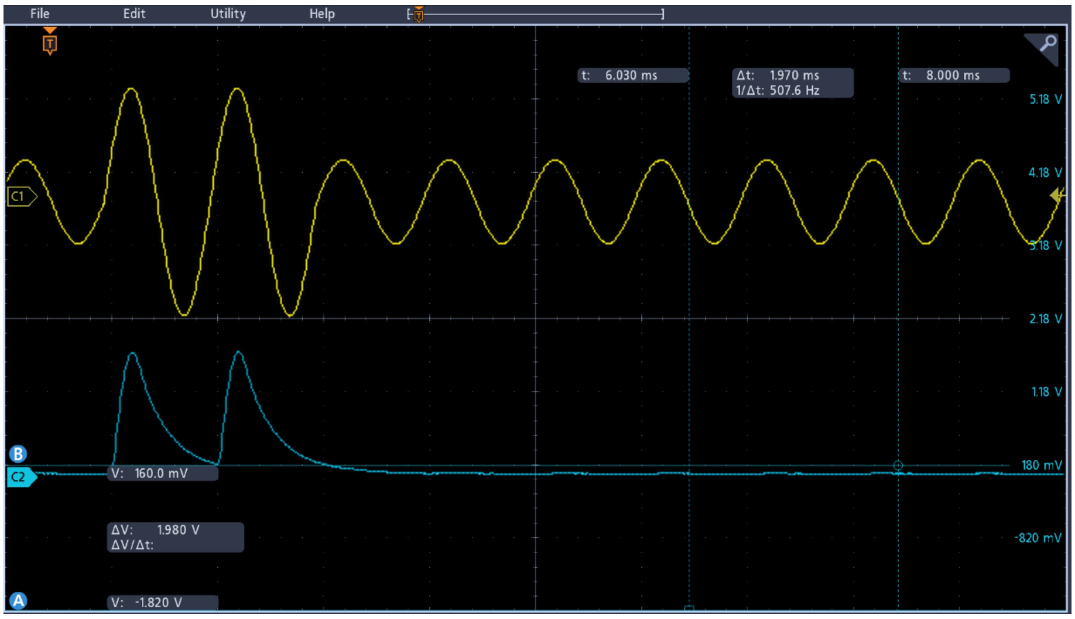


Figure 11.1 Output of Peak Detector of Circuit 1

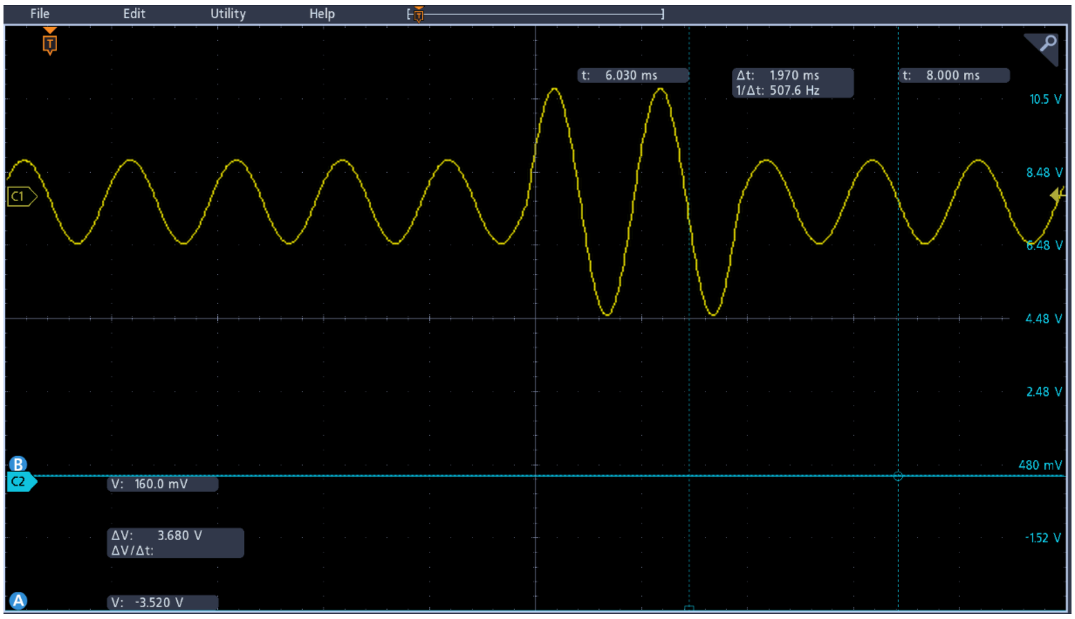


Figure 11.2 Output of Threshold Computer of Circuit 1

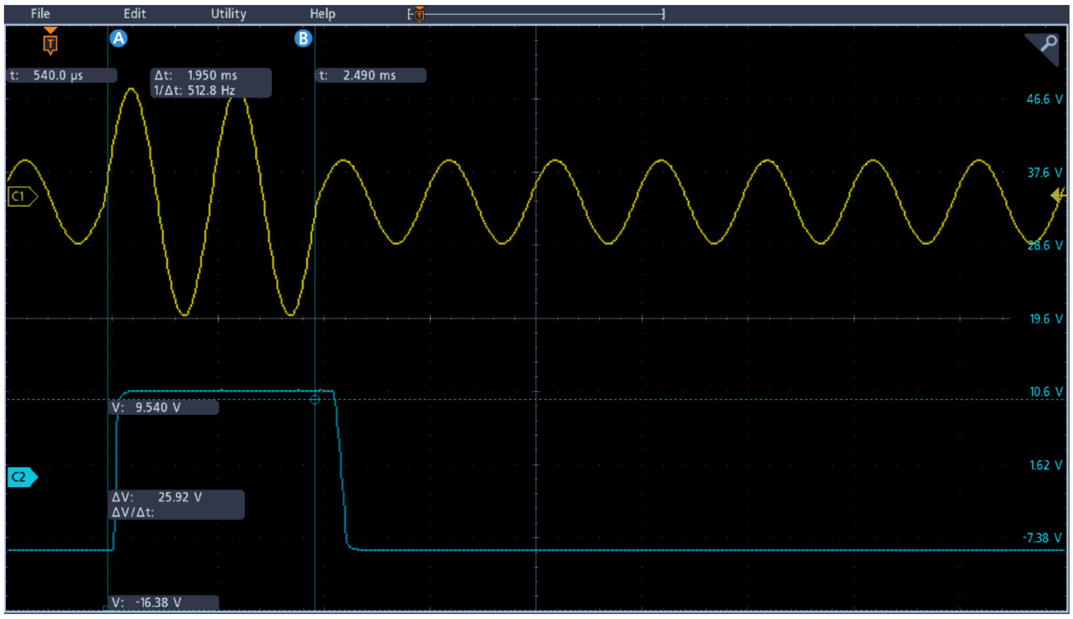


Figure 11.3 Output of Comparator of Circuit 1

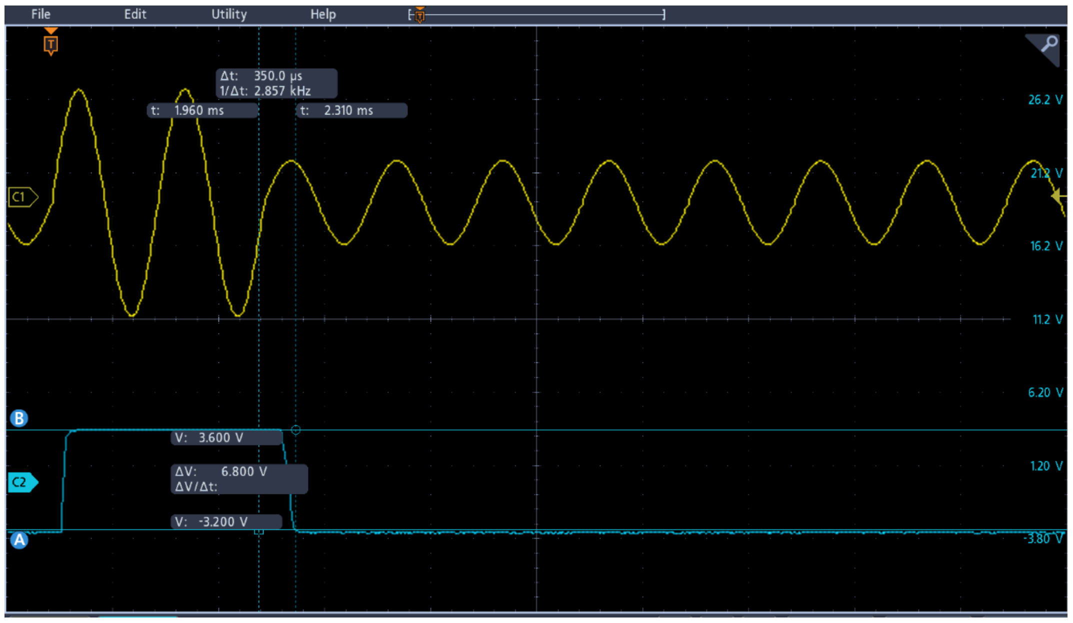


Figure 11.4 Output of Voltage Divider of Circuit 1

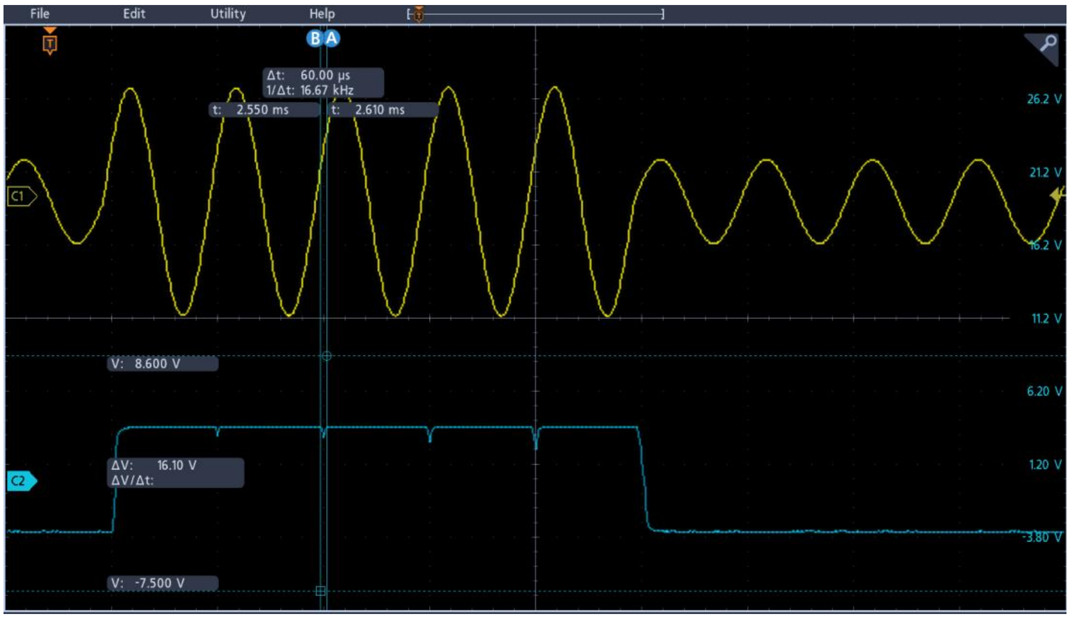


Figure 11.5 Grooves in output signal of Circuit 1

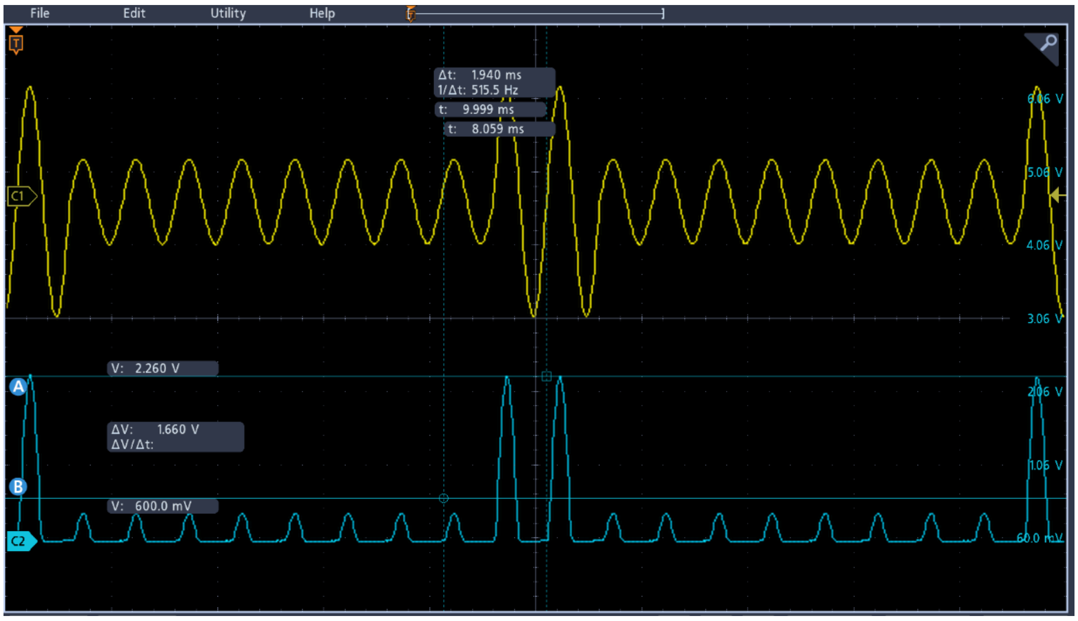


Figure 11.6 Output of Half Wave Rectifier at point 1 of Circuit 2

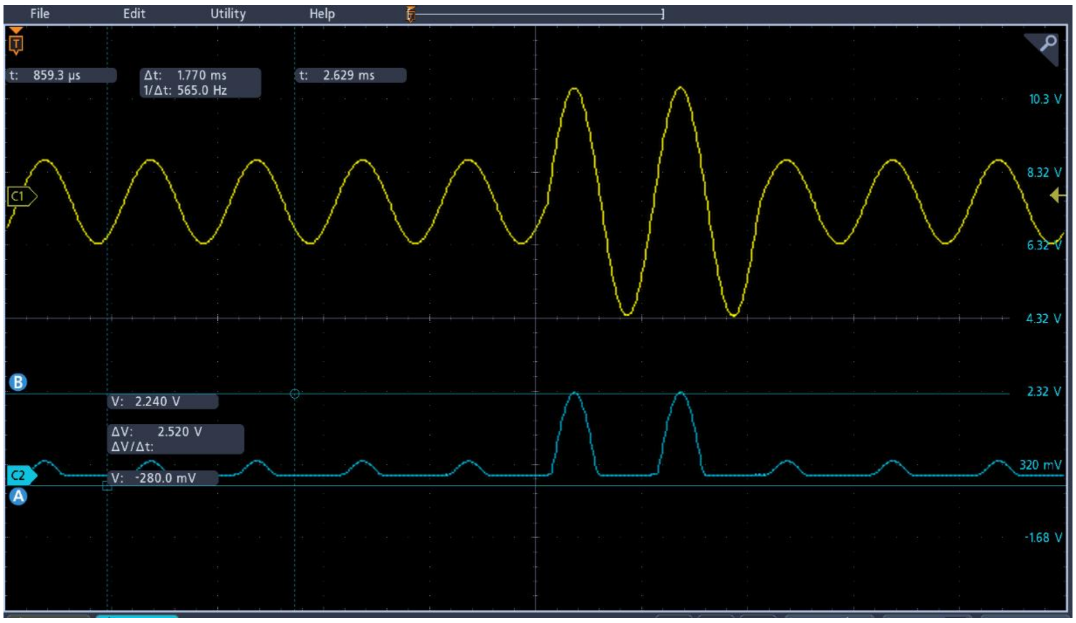


Figure 11.7 Output of Buffer at point 2 of Circuit 2

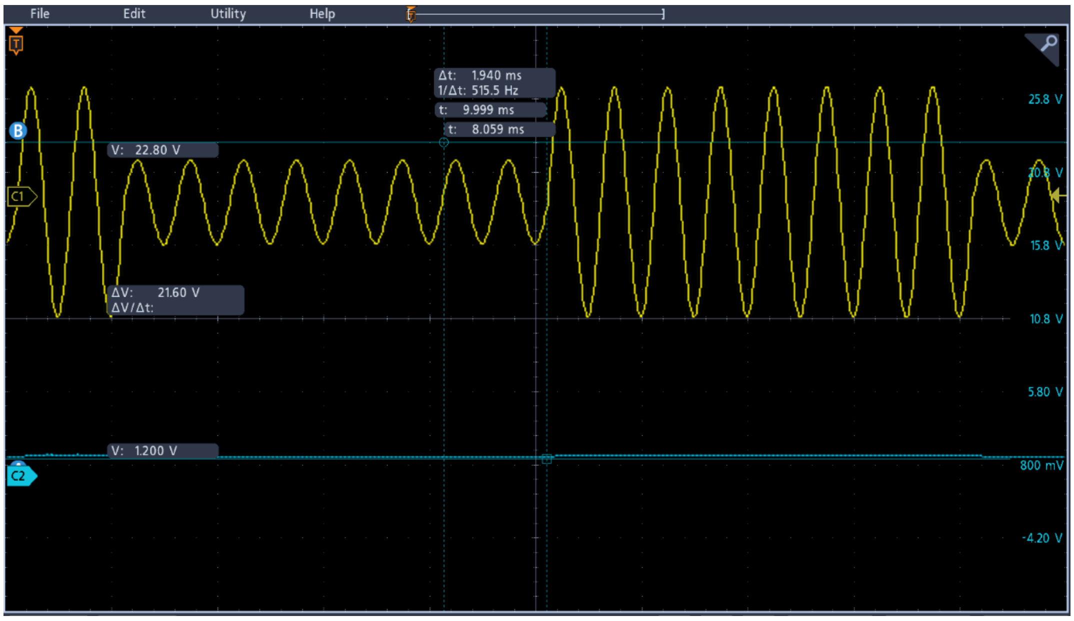


Figure 11.8 Output of Peak Detector at point 3 of Circuit 2

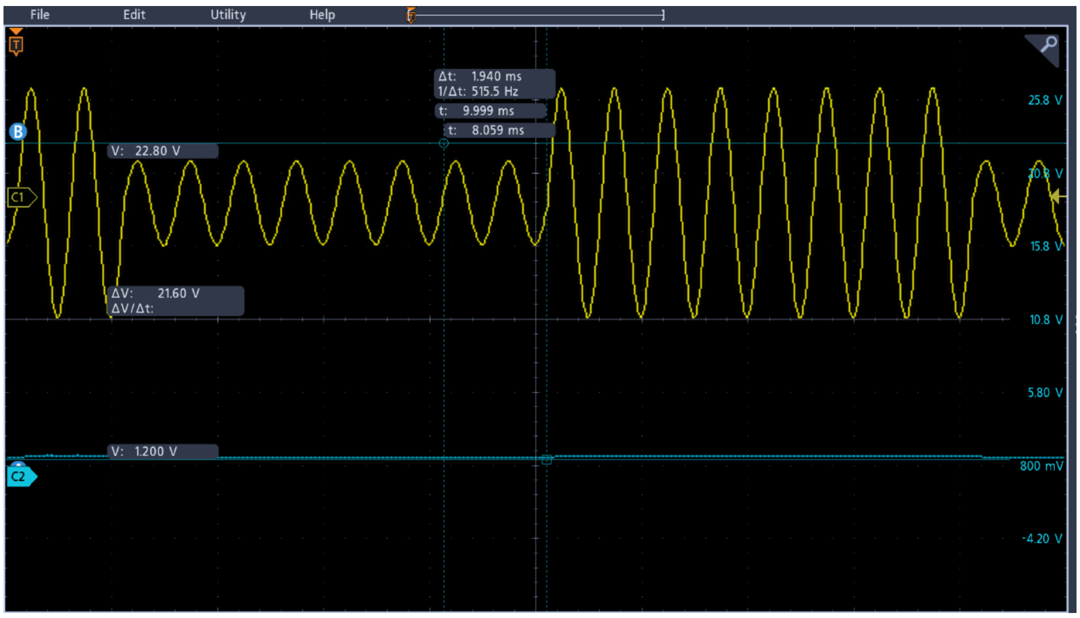


Figure 11.9 Output of Buffer at point 4 of Circuit 2

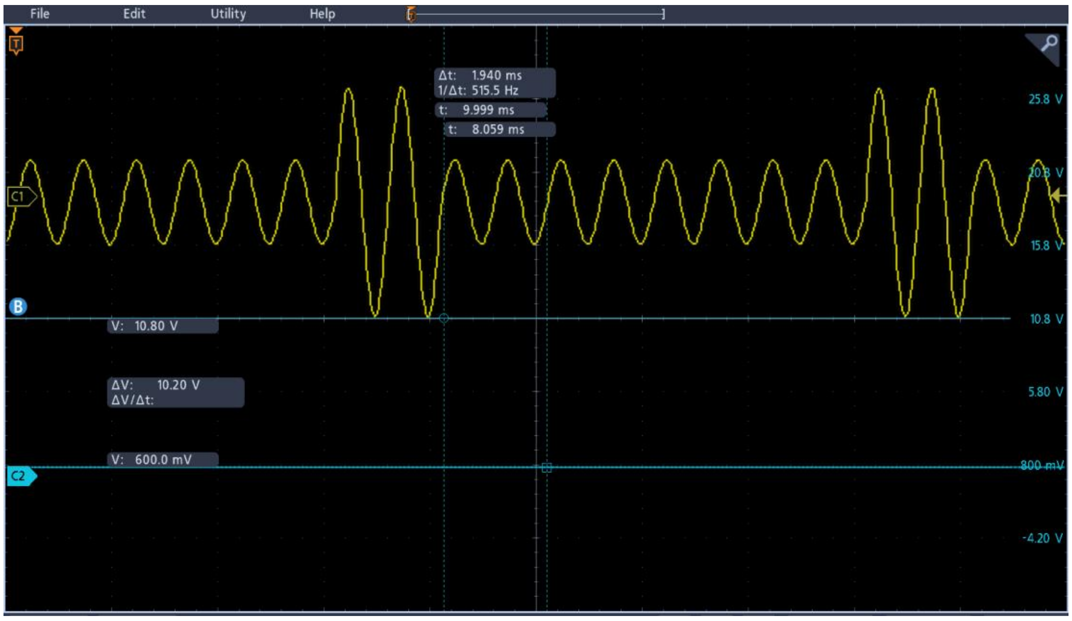


Figure 11.10 Output of Voltage Divider at point 5 of Circuit 2

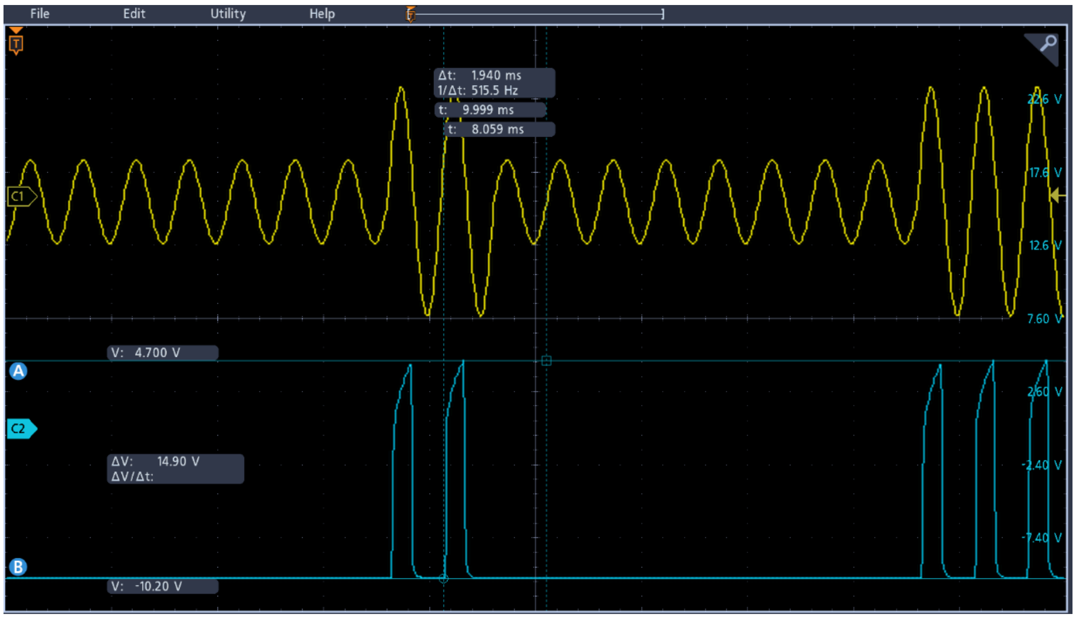


Figure 11.11 Output of Comparator at point 6 of Circuit 2

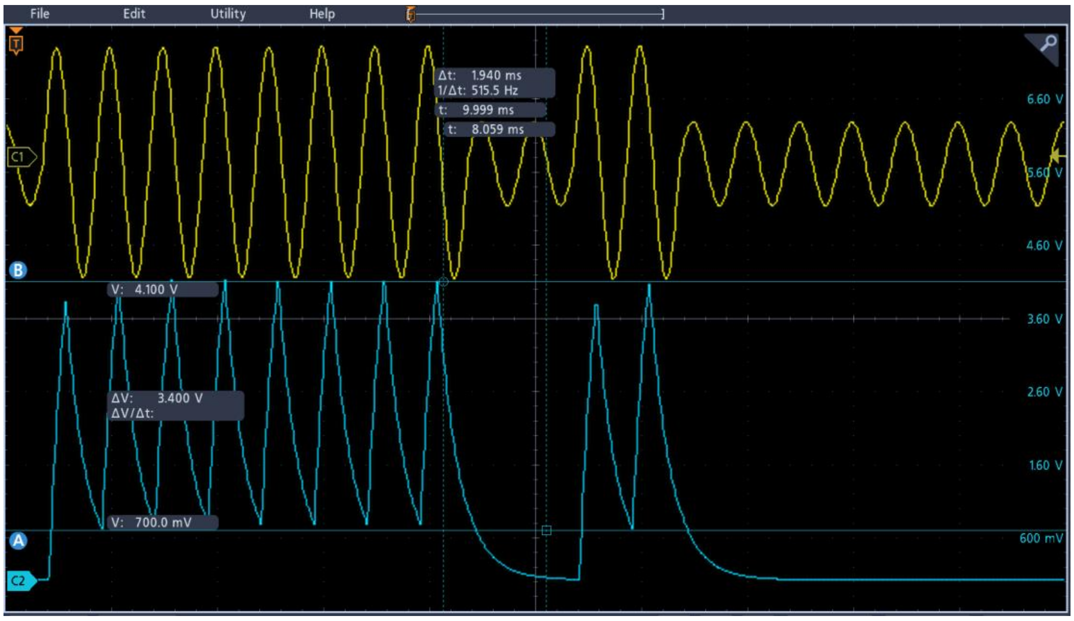


Figure 11.12 Output of Peak Detector at point 7 of Circuit 2

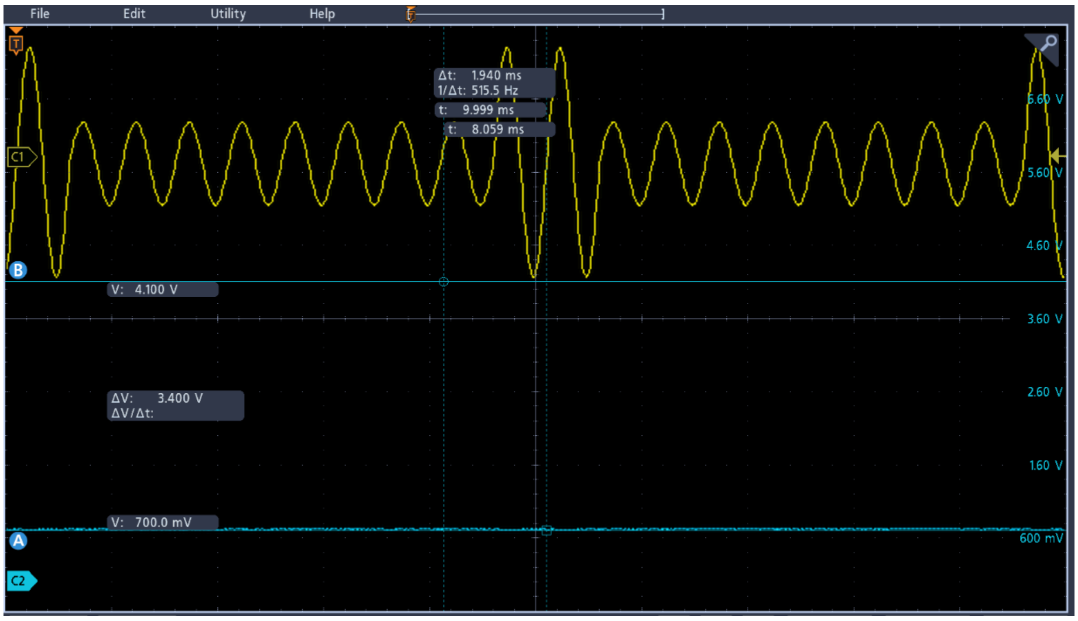


Figure 11.13 Output of the diode at point 8 of Circuit 2

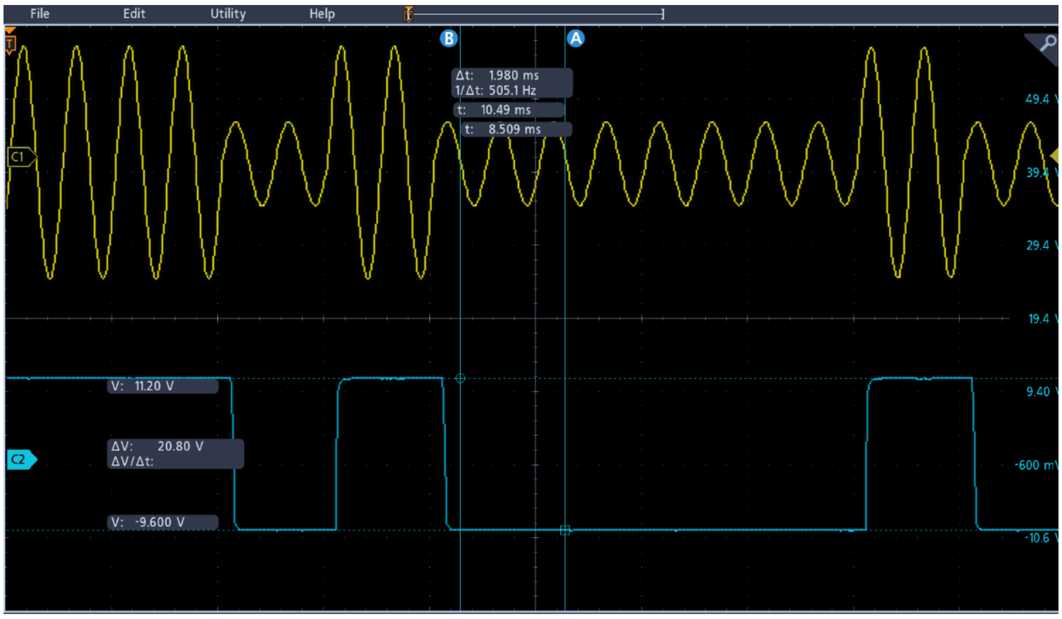


Figure 11.14 Output of Comparator at point 9 of Circuit 2

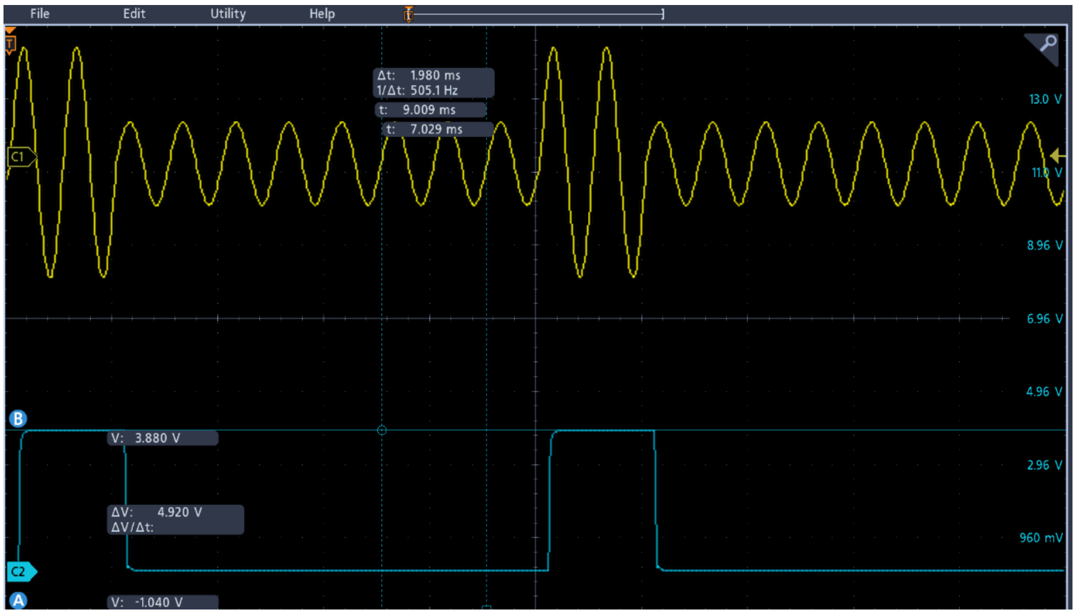
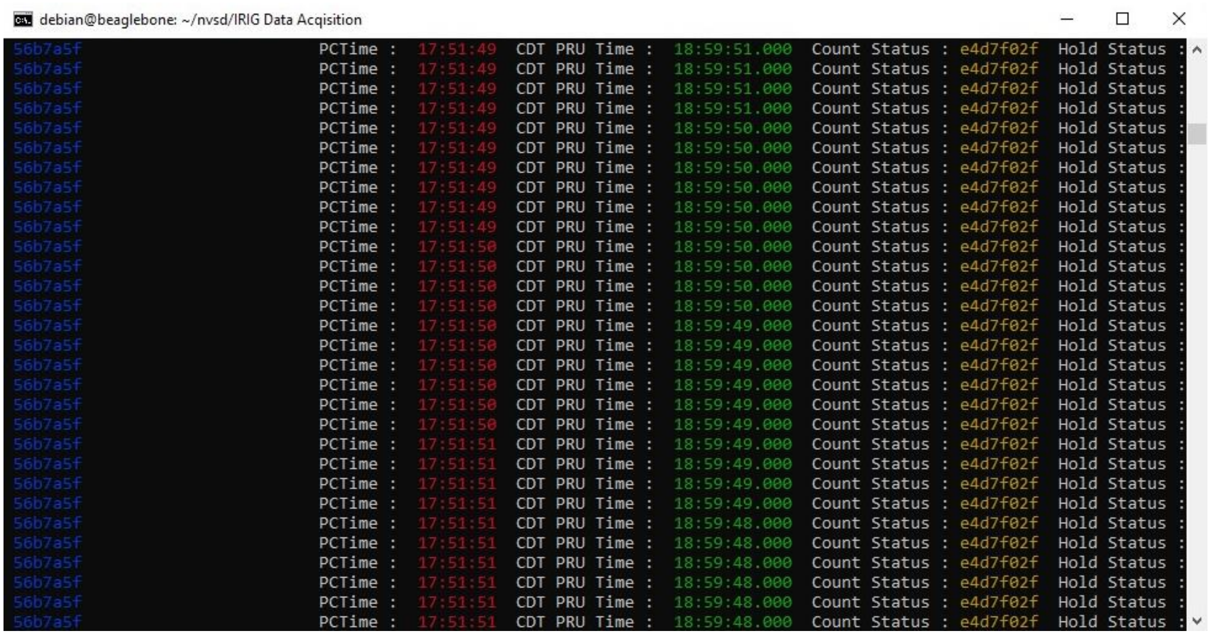


Figure 11.15 Output of voltage divider at point 10 of Circuit 2 (Final Required output of Demodulator circuit)



debian@beaglebone: ~/nvsd/IRIG Data Acqisition

```
56b7a5f                PCTime :  17:51:49  CDT PRU Time :  18:59:51.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:49  CDT PRU Time :  18:59:51.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:49  CDT PRU Time :  18:59:51.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:49  CDT PRU Time :  18:59:51.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:49  CDT PRU Time :  18:59:50.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:49  CDT PRU Time :  18:59:50.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:49  CDT PRU Time :  18:59:50.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:49  CDT PRU Time :  18:59:50.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:49  CDT PRU Time :  18:59:50.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:49  CDT PRU Time :  18:59:50.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:50  CDT PRU Time :  18:59:50.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:50  CDT PRU Time :  18:59:50.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:50  CDT PRU Time :  18:59:50.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:50  CDT PRU Time :  18:59:50.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:50  CDT PRU Time :  18:59:49.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:50  CDT PRU Time :  18:59:49.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:50  CDT PRU Time :  18:59:49.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:50  CDT PRU Time :  18:59:49.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:50  CDT PRU Time :  18:59:49.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:50  CDT PRU Time :  18:59:49.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:51  CDT PRU Time :  18:59:49.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:51  CDT PRU Time :  18:59:49.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:51  CDT PRU Time :  18:59:49.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:51  CDT PRU Time :  18:59:49.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:51  CDT PRU Time :  18:59:48.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:51  CDT PRU Time :  18:59:48.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:51  CDT PRU Time :  18:59:48.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:51  CDT PRU Time :  18:59:48.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:51  CDT PRU Time :  18:59:48.000  Count Status : e4d7f02f  Hold Status :
56b7a5f                PCTime :  17:51:51  CDT PRU Time :  18:59:48.000  Count Status : e4d7f02f  Hold Status :
```

Figure 11.16 Final Output on the screen of the computer displaying Timing information

APPENDIX 2

ASSEMBLY CODE

```
;*Set the date and time to avoid clock skew error: sudo date -s 2023-11-
08T10:23:00
;*Open IRIG DATA ACQUISITION folder
;*cd nvsd
;*sudo ./configpin.sh
;*sudo ./pin_setup.sh
;*cd IRIG DATA ACQUISITION
;*COMMANDS FOR EXECUTION
;*MAKE CLEAN..........MAKE..........MAKE RUN

        .include "tm-pru.inc"
        .asg 32, PRU0_R31_VEC_VALID
        .asg 3, PRU_EVTOUT_0
        .asg r23,DATA
        .asg r22,TEMP
        .asg r2,HOUR
        .asg r3,MINUTE
        .asg r4,SECOND
        .asg r6,COUNT
        .asg r7,ITERATE
        .asg r8,ISPEAK
        .asg r9,DATAWIDTH
        .asg r10,PREVDATA
        .asg r11,CURRENTDATA
        .asg r12,SAMPLENO
        .asg r13,SYNCUPPERLIMIT
        .asg r14,ISTWOSYNC
        .asg r15,BYTECOUNTER
        .asg r16,BITCOUNTER
        .global main

GETBIT .macro
        lsl DATA,DATA,1
        qbbs loop1,r31,10
        jmp loop2
loop1:
        set DATA,DATA,0
loop2:
        nop
        .endm                   ;4 instructions

COUNTSTART .macro
        ldi32 COUNT,0
        ldi32 ITERATE,0
        mov TEMP,DATA                   ;*3 instruction
CHECKONE:
        qbbs COUNTONE,TEMP,ITERATE
        jmp INCREMENTITERATE
COUNTONE:
        add COUNT,COUNT,1
INCREMENTITERATE:
        add ITERATE,ITERATE,1
        qbne CHECKONE,ITERATE,16    ;*( 16 x 4 = 64 instructions )
        .endm
```

```
CHECKONESFORWIDTH .macro
        ldi32 ISPEAK,0
        qble CALCULATEWIDTH,COUNT,12
        jmp SKIPCALCULATEWIDTH
CALCULATEWIDTH:
        add DATAWIDTH,DATAWIDTH,1
        nop
        jmp ENDCALCULATEWIDTH
SKIPCALCULATEWIDTH:
        qbne SETISPEAK,DATAWIDTH,0
        jmp ENDCALCULATEWIDTH
SETISPEAK:
        set ISPEAK,ISPEAK,0
ENDCALCULATEWIDTH:
        nop
        .endm                   ;*6 instructions

WHICHPEAK .macro
        mov PREVDATA,CURRENTDATA
        ldi32 CURRENTDATA,0
        ldi32 ISPEAK,0
        qble CHECKSYNC,DATAWIDTH,208
        qble CHECKBITONE,DATAWIDTH,112
        qble CHECKBITZERO,DATAWIDTH,16
        jmp NOBITDETECTED
CHECKSYNC:
        qbge SYNCDETECTED,DATAWIDTH,SYNCUPPERLIMIT
        nop
        nop
        nop
        jmp NOBITDETECTED
SYNCDETECTED:
        ldi32 CURRENTDATA,1
        nop
        nop
        nop
        jmp ENDWHICHPEAK
CHECKBITONE:
        qbge BITONEDETECTED,DATAWIDTH,192
        nop
        nop
        jmp NOBITDETECTED
BITONEDETECTED:
        ldi32 CURRENTDATA,2
        nop
        nop
        jmp ENDWHICHPEAK
CHECKBITZERO:
        qbge BITZERODETECTED,DATAWIDTH,96
        nop
        jmp NOBITDETECTED
BITZERODETECTED:
        ldi32 CURRENTDATA,3
        nop
        jmp ENDWHICHPEAK
NOBITDETECTED:
        ldi32 CURRENTDATA,4
```

```
ENDWHICHPEAK:
        ldi32 DATAWIDTH,0
        .endm                   ;*11 instructions

MAINTAINSAMPLINGRATE .macro
MAINTAIN:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop                       ;*23
        sub SAMPLENO,SAMPLENO,1
        qbne MAINTAIN,SAMPLENO,0     ;*6148 instructions
        ldi32 SAMPLENO,245
        .endm

CHECKTWOSYNC .macro
        qbeq ISTWOSYNCNOP,ISTWOSYNC,1
        qbne CURRENTDATANOT1,CURRENTDATA,1
        qbne PREVDATANOT1,PREVDATA,1
        ldi32 ISTWOSYNC,1
        jmp ENDCHECKTWOSYNC
ISTWOSYNCNOP:
        nop
CURRENTDATANOT1:
        nop
        nop
        jmp ENDCHECKTWOSYNC
PREVDATANOT1:
        nop
        nop
ENDCHECKTWOSYNC:
        nop
        .endm                   ;*6 instructions

CALCULATEBYTEANDBITCOUNTER .macro
        qbeq STARTCOUNT,BYTECOUNTER,11
        qbeq STARTCOUNT,BYTECOUNTER,0
        add BITCOUNTER,BITCOUNTER,1
        qbeq RESETBITCOUNTER,BITCOUNTER,10
        qbeq DISCARDALL,CURRENTDATA,1
        jmp ENDNORMALCOUNT
STARTCOUNT:
        ldi32 BYTECOUNTER,1
        ldi32 BITCOUNTER,1
        jmp ENDSTARTCOUNTER
RESETBITCOUNTER:
        qbne DISCARDALL,CURRENTDATA,1
        SBBO &BYTECOUNTER,r29,0,4
        ldi R31.b0, PRU0_R31_VEC_VALID | PRU_EVTOUT_0
        ldi32 BITCOUNTER,0
        add BYTECOUNTER,BYTECOUNTER,1
        jmp ENDRESETBITCOUNTER
DISCARDALL:
        ldi32 ISTWOSYNC,0
        ldi32 BYTECOUNTER,0
        ldi32 BITCOUNTER,0
        ldi32 SECOND,0
        ldi32 MINUTE,0
        ldi32 HOUR,0
        jmp ENDCALCULATEBYTEANDBITCOUNTER
ENDSTARTCOUNTER:
        nop
ENDNORMALCOUNT:
        nop
ENDRESETBITCOUNTER:
        nop
        nop
        nop
        nop
ENDCALCULATEBYTEANDBITCOUNTER:
        nop                         ;*12 instructions
        .endm

FORBYTEONE .macro
        qbeq BYTE1SKIPBIT0,BITCOUNTER,0
        qbeq BYTE1SKIPBIT1,BITCOUNTER,1
        qbeq BYTE1SKIPBIT6,BITCOUNTER,6
        lsr SECOND,SECOND,1
        qbeq SETSECONDBIT,CURRENTDATA,2
        jmp MAINTAINBYTE1
SETSECONDBIT:
        set SECOND,SECOND,7
        jmp ENDPUSHBYTEONE
BYTE1SKIPBIT0:
        nop
BYTE1SKIPBIT1:
        nop
BYTE1SKIPBIT6:
        nop
        nop
        nop
MAINTAINBYTE1:
        nop
ENDPUSHBYTEONE:
        qbne ENDFORBYTEONE,BITCOUNTER,9
        lsr SECOND,SECOND,1
        SBBO   &SECOND, r29, 12, 4
        nop
ENDFORBYTEONE:
        nop
        .endm                   ;*9 instructions

FORBYTETWO .macro
        qbeq BYTE2SKIPBIT0,BITCOUNTER,0
        qbeq BYTE2SKIPBIT5,BITCOUNTER,5
        lsr MINUTE,MINUTE,1
        qbeq SETMINUTEBIT,CURRENTDATA,2
        jmp MAINTAINBYTE2
SETMINUTEBIT:
        set MINUTE,MINUTE,7
        jmp ENDPUSHBYTETWO
BYTE2SKIPBIT0:
        nop
BYTE2SKIPBIT5:
        nop
        nop
        nop
MAINTAINBYTE2:
        nop
ENDPUSHBYTETWO:
        qbne ENDFORBYTETWO,BITCOUNTER,9
        SBBO   &MINUTE, r29, 8, 4
ENDFORBYTETWO:
        nop
        .endm                   ;*9 instructions
```

```
FORBYTETHREE .macro
        qbeq BYTE3SKIPBIT0,BITCOUNTER,0
        qbeq BYTE3SKIPBIT5,BITCOUNTER,5
        lsr HOUR,HOUR,1
        qbeq SETHOURBIT,CURRENTDATA,2
        jmp MAINTAINBYTE3
SETHOURBIT:
        set HOUR,HOUR,7
        jmp ENDPUSHBYTETHREE
BYTE3SKIPBIT0:
        nop
BYTE3SKIPBIT5:
        nop
        nop
        nop
MAINTAINBYTE3:
        nop
ENDPUSHBYTETHREE:
        qbne ENDFORBYTETHREE,BITCOUNTER,7
        SBBO   &HOUR, r29, 4, 4
ENDFORBYTETHREE:
        nop
        .endm                          ;*9 instructions



main:
        ldi32 r29, 0x00000000
        ldi32 DATA,0
        ldi32 TEMP,0
        ldi32 ITERATE,0
        ldi32 COUNT,0
        ldi32 ISPEAK,0
        ldi32 DATAWIDTH,0
        ldi32 PREVDATA,0
        ldi32 CURRENTDATA,0
        ldi32 SAMPLENO,245
        ldi32 SYNCUPPERLIMIT,288
        ldi32 ISTWOSYNC,0
        ldi32 BYTECOUNTER,0
        ldi32 BITCOUNTER,0
        ldi32 SECOND,0
        ldi32 MINUTE,0
        ldi32 HOUR,0

START:
        GETBIT
        COUNTSTART
        CHECKONESFORWIDTH               ;*80 instructions
        MAINTAINSAMPLINGRATE
        qbne SKIPALLCHECK,ISPEAK,1      ;*49 instructions
        WHICHPEAK                       ;*12 instructions
        CHECKTWOSYNC                    ;*7 instructions
        qbne SKIP,ISTWOSYNC,1           ;*46 instructions
        CALCULATEBYTEANDBITCOUNTER      ;*13 instructions
        qbeq CALLFORBYTE1,BYTECOUNTER,1 ;*10 instructions
;*49 instructions
```

```
	qbeq CALLFORBYTE2,BYTECOUNTER,2	;*10 instructions
;*49 instructions
	qbeq CALLFORBYTE3,BYTECOUNTER,3	;*10 instructions
;*49 instructions
	jmp MAINTAINSKIP

CALLFORBYTE1:
	FORBYTEONE
	nop
	nop
	jmp NOPUSH			;*49 instructions
CALLFORBYTE2:
	FORBYTETWO
	nop
	jmp NOPUSH			;*49 instructions
CALLFORBYTE3:
	FORBYTETHREE
	jmp NOPUSH			;*49 instructions
SKIPALLCHECK:
	nop
	nop
	nop
	nop
	nop
	nop
	nop
	nop
	nop
	nop
```

```
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop         ;*20
SKIP:
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
MAINTAINSKIP:
        nop
        nop
        nop
        nop
        nop
        nop
        nop         ;*27
NOPUSH:
        nop
        jmp START
```

APPENDIX 3

C CODE

```
// Loads  PRU0 and PRU1 memory. Initialises buffer and displays (system-
wide) DDR memory written by PRU0
// TODO: load data.bin also.
//
// Pass in the filename of the .bin file on the command line, eg:
// $ ./tm-arm pru0-text.bin  pru1-text.bin
//
// Compile with:
// gcc -std=gnu99 -o tm-arm tm-arm.c -lprussdrv
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <inttypes.h>
#include <prussdrv.h>
#include <pruss_intc_mapping.h>
#include <time.h>
#include <pthread.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <arpa/inet.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>
```

```
#include <time.h>

#define         false   0
#define         true    !false
#define         FRAMELEN        128
#define PREAMBLE        0xF9A42BB1
#define SAMPLINGFREQ        7                   //If changed, code masking
patterns also need to be changed
#define NOISEMARGIN2
#define OUTBUFFNUMFRAME 50
#define TCPPACKLENGTH     (OUTBUFFNUMFRAME*FRAMELEN)
#define PORT1 8070
#define PORT2 8074
#define PORT3 8078
#define MAXTCPRETRYCNT 3
#define SA struct sockaddr

 static void *pru0DataMemory;
 static unsigned int *pru0DataMemory_int;

volatile void *shared_ddr = NULL;
volatile void *pingStartAddress = NULL;
volatile void *pongStartAddress = NULL;
unsigned int *localMemory;
unsigned char lookupArray[0xffff] = {0};
union StreamUnion {
        unsigned long long int bitStream;
        unsigned int bitStreamInt[2];
} currStreamA, currStreamB;
```

```
const char* get_process_name_by_pid(const int pid) {
        char *name = (char*)calloc(1024,sizeof(char));
        char *name1 = (char*)calloc(1024,sizeof(char));
        if (name) {
                sprintf(name,"/proc/%d/cmdline",pid);
                FILE *f = NULL;
                f = fopen(name,"r");
                if (f != NULL) {
                        fscanf (f,"%s\n",name1);
                        fclose(f);
                }
        }
        return name1;
}
void CheckMultipleInstances() {
        FILE *fp = NULL;
        int prevPid = 0, currPid = 0;
        fp = fopen("pid.tm", "r");
        if (fp != NULL) {
                fscanf(fp,"%d", &prevPid);
                currPid = getpid();
                if
(strcmp(get_process_name_by_pid(prevPid),get_process_name_by_pid(currPid
)) == 0) {
                        printf("Another instance is running\n");
                        fclose (fp);
                        exit(0);
                }
                else {
```

```
                fclose (fp);
                fp = NULL;
                fp = fopen("pid.tm", "w");
                if (fp != NULL) {
                        fprintf(fp,"%d", currPid);
                        fclose(fp);
                }
            }
        }
        else {
            fp = fopen("pid.tm", "w");
            if (fp != NULL) {
                currPid = getpid();
                fprintf(fp,"%d", currPid);
                fclose(fp);
            }
        }
}
int main(int argc, char **argv)
{

        CheckMultipleInstances();
        if (argc != 3)
        {
            printf("Usage: %s pru0_code.bin pru1_code\n", argv[0]);
            return 1;
        }
        sleep(1);
        prussdrv_init();
        if (prussdrv_open(PRU_EVTOUT_0) == -1)
        {
                printf("prussdrv_open() event 0 failed\n");
                return 1;
        }
        tpruss_intc_initdata pruss_intc_initdata = PRUSS_INTC_INITDATA;
   // Map PRU's INTC
        prussdrv_pruintc_init(&pruss_intc_initdata);
   // Copy data to PRU memory - different way
        prussdrv_map_prumem(PRUSS0_PRU1_DATARAM,
&pru0DataMemory);
        pru0DataMemory_int = (unsigned int *) pru0DataMemory;
        prussdrv_exec_program(1, argv[2]);
        sleep(.5);
        prussdrv_exec_program(0, argv[1]);
        sleep (1.);
        time_t rawtime;
        struct tm *timeinfo;
        system("clear");
        int current_event_no=0;
        int prev_event_no=0;
        int miss_interrupt=-1;
        unsigned int millisec=0;
        printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
        do
        {
                prev_event_no=current_event_no;
                current_event_no=prussdrv_pru_wait_event (PRU_EVTOUT_0);
                /*if((current_event_no-prev_event_no)!=1)
                {miss_interrupt++;
                printf("Current event no:%d\nPrevious event
no:%d",current_event_no,prev_event_no);
                }*/
            unsigned int ByteCounter = *(pru0DataMemory_int);
            unsigned int Hour = *(pru0DataMemory_int+1);
            unsigned int Minute = *(pru0DataMemory_int+2);
            unsigned int Second = *(pru0DataMemory_int+3);
            unsigned int Count = *(pru0DataMemory_int+4);
            unsigned int Hold = *(pru0DataMemory_int+5);
            time(&rawtime);
            timeinfo= localtime(&rawtime);
            if(Hold!=1)
                if(Count==0x09)
                    millisec=1000-(ByteCounter*100);
                else if(Count==0x50)
                    millisec=(ByteCounter*100)-100;
        printf("\r\t\t\t\tPCTime : \x1b[31m %02d:%02d:%02d \x1b[0m CDT
PRU Time : \x1b[32m %02x:%02x:%02x.%03d \x1b[0m Count Status
:\x1b[33m %02x \x1b[0m Hold Status :\x1b[34m %02x \x1b[0m",
        timeinfo->tm_hour,timeinfo->tm_min,timeinfo-
>tm_sec,Hour,Minute,Second,millisec,Count,Hold);
        //printf("\nNo of interrupts missed:%d",miss_interrupt);
        //printf("\nCount Status:%x Hold Status:%x",);
       fflush(stdout);
            prussdrv_pru_clear_event (PRU_EVTOUT_0,
PRU0_ARM_INTERRUPT);
        }
        while (1);
        prussdrv_pru_disable(0);
        prussdrv_pru_disable(1);
        prussdrv_exit();

        printf ("Program exited\n");
        return 0;
}
```

REFERENCES

[1] Aiping Wu, Qingqing Fu, Zhiqiang Gao (2011).The Decoded Design of Irig-B

[2] Air force navigation synchronization. Retrieved from https://www.cxr.com/en/applications/irig-b-synchronization-42.html

[3] Arbiter Systems. IRIG-B Specifications

[4] BeagleBone Black Industrial 4G. Retrieved from https://www.prayogindia.in

[5] C H Wang (2018). Design of Time Synchronization System Based on Irig-B Code Demodulation Technology

[6] Comparing synchronization protocols: IRIG vs. NTP vs. PTP. Retrieved from https://www.bodet-time.com

[7] Derek Molloy. Exploring BeagleBone. Second Edition. Tools and Techniques for Building with Embedded Linux

[8] L Jia (2017). Design of FPGA-Based Irig-B Code Decoder

[9] Quentin Griffiths. Tekron International Limited 2017. Almost Everything You Need To Know About IRIG-B Time Sync

[10] Timing Committee Telecommunications And Timing Group Range Commanders Council (1998). IRIG STANDARD 200-98

[11] Vincent Liu, Aaron Parks, Vamsi Talla, Shyamnath Gollakota, David Wetherall & Joshua R. Smith Ambient Backscatter (2013) Wireless Communication Out of Thin Air.

[12] Welcome to IRIG 106. Retrieved from https://www.irig106.org/

[13] What is IRIG-B Time Code? Retrieved from https://www.masterclock.com/understanding-irig-b-time-code.html

[14] What is BeagleBone Black. Retrieved from https://www.beagleboard.org