Implementation and Performance Analysis of Precision Time Protocol on Linux based System-On-Chip Platform

Mudassar Ahmed
mudassar.ahm@hotmail.com

FACHHOCHSCHULE KIEL
University of Applied Sciences

MASTER PROJECT

Kiel University of Applied Science

MSc. Information Engineering

in Kiel

im Mai 2018

Declaration

I hereby declare and confirm that this project is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Kiel, May 9, 2018

Mudassar Ahmed
mudassar.ahm@hotmail.com

Contents

Abstract

Associated control and measurement applications require time synchronization and many embedded systems have on-board sense of time. The specific clock in a networked system needs to be checked whether the time deviation is acceptable for the particular application or there is a need for correction in time. So, parallel to control and measurement related communication there is a strong need for clock communication between system nodes.

The Precision Time Protocol (PTP) specified in IEEE 1588-2008 standard is a self-organizing and optionally hardware supported time synchronization protocol, the protocol operates on master/slave hierarchy on local area networked devices and provides the opportunity to achieve the sub-microsecond level accuracy on existing multicast supported LAN network. The PTP protocol does not require additional physical network infrastructure to establish PTP network, it uses existing LAN network infrastructure. It is possible that the existing applications running on same system node or in overall network may affect the performance of the PTP clock synchronization process.

The project aimed to implement and analyze the PTP protocol on hardware-assisted IEEE 1588 clock based system-on-chip platform (Beaglebone) by using an open-source solution. The document summarizes PTP protocol and PTP infrastructure in Linux, and it represents the comparison between software and hardware assist PTP implementation. In order to analyze the performance and behavior of established PTP network, multiple tools are used to simulate the load scenario in different dimensions (CPU, I/O and Network).

List of Figures

List of Symbols and Abbreviations

PTP: Precision Time Protocol
NTP: Network Time Protocol
GPS : Global Positioning System
SoC : System-On-Chip
BBB : BeagleBone Black
TTP : Time-Triggered Protocol
IEEE :Institute of Electrical and Electronics Engineers
UDP : User Datagram Protocol
GM : Grand Master Clock
OC : Ordinary Clock
BC : Boundary Clock
TC : Transparent Clock
BMCA : Best Master Clock Algorithm
PHC : PTP Hardware Clock
API : Application programming interface
OS : Operating System
LAN : Local Area Network
ns : Nano-Second
us : Micro-Second
ms : Mili-Second
sec : Second
POSIX : Portable Operating System Interface
Mb : Megabits
ioctl : input/output control

Chapter 1

Introduction

Continuous transformation of computer systems from large-scale isolated units to application specific distributed units is rising the challenge of time synchronization due to their associated and time-critical actions. These kinds of transformation can clearly be seen in many industries like, automation, distributed measurement systems and power distribution systems. Several solutions like GPS and NTP emerged to overcome synchronization challenges. As the large-scale distributed systems are becoming more complex and modular, where each distributed module/node is communicating via standardized communication medium, the demand for precision and accuracy of time synchronization is increasing, while existing solutions are either limited in term of synchronization accuracy or requires specialized additional hardware and operational support to ensure the common time in distributed nodes. On the other hand, installation of additions time keeping networks and devices makes the system more complex and increases the troubleshooting and maintenance problems.

IEEE 1588 Precision Time Protocol (PTP) developed to overcome these challenges, it provides the self-organizing time synchronization protocol while utilizing the existing system for timekeeping applications. The widespread adaptation of IEEE 1588 Precision Time Protocol (PTP) is not just limited to industrial automation and measurement systems, the integration of PTP is also realized and standardized in many leading technologies/sectors like Audio-Video Bridging, Smart Power Grids and Financial Systems. Due to widespread recognition of PTP, the silicon vendors are also offering on-board sense of time support with hardware-assisted timestamping capabilities for the wide variety of embedded solutions. Additionally, PTP hardware support in mainline Linux kernel is becoming a catalyst in the development and adaptation of Linux based software solution for time synchronization applications in different sectors.

Advanced features of System-on-Chip (SoC) platforms are becoming a powerful tool to implement the different industrial solution as a prototype and PTP community is providing continuous support for PTP software solutions for Linux based SoC devices, which are having Hardware timestamping capabilities but these applications are developed for Linux like operating systems. So, different Linux kernel level resource scheduling procedures can effect the time timestamping process. The source of these latencies can be other resource exhausting applications running on the same system. In some cases, these resource exhausting applications cannot be avoided as an overall system. In this project, PTP protocol is implemented on SoC platform on which the PTP hardware

capabilities are enabled by using Linux kernel PHC API and hardware timestamping socket option. An opensource PTP solution is used to establish the PTP network for further analysis.

1.1 Research Objectives and Goals

Goals

- Enabling Hardware timestamping capabilities of Linux bases SoC Platform (BeagleBone Black).
- Analyzing the behavior of open source software solution for PTP implementation, while simulating different load scenarios (Network load, Processing load) in Linux OS.

Research Objectives/Questions

- Analysis of precision uncertainty in Hardware and Software based solutions of PTP.
- What is the maximum attainable accuracy with Hardware and Software based solutions of PTP?

1.2 Approach

In order to achieve research objectives and to implement IEEE 1588 protocol, linuxPTP (Opensource PTP implementation) application is used on SoC devices (Beaglebone Black) in a Linux environment. The protocol hierarchy is established in different test case scenarios (Hardware assisted and Software based tests). The log data generated in a specific scenario is prepared and imported to MATLAB workspace, where comparison of results in different scenarios are compiled in graphical form. These results are further analyzed and evaluated to draw the final conclusion.

1.3 Outline

First, the report introduces to the time synchronization and give a overview of PTP protocol in **Chapter 2**. The **Chapter 3** focuses on the Linux support for PTP in the context of timestamping mechanism and Clock control mechanism. In **Chapter 4**, a brief introduction about tools and technologies used in implementation is presented. The results of test scenarios are presented and discussed in **Chapter 5**. The **Chapter 6** draws final conclusion on work.

Chapter 2

Literature Review

2.1 Time Synchronization

The time is a measurable period in which a particular event, process or action may occur. In the domain of distributed systems, it can be used to draw a conclusion on the rate of change, event ordering, an interval between different process and exact time of particular action.

A clock is used to measure the specific spot in the time or a whole interval of time. In an embedded system, the clock is driven by an internal oscillator, which generates the signals with a precise frequency. In some systems there is no timing sense is defined on both silicon and software level, these kinds of implicit time systems mostly use a trigger signal in the network to indicate the particular spot in time. On the other hand, time aware systems are primarily driven by an internal oscillator. In a networked system, where different nodes having different type of clocks, which are powered by non-identical oscillators. These oscillators are running at different frequencies and having different behavior in different conditions (Temperature, Voltage) [1], which results a timing error. So, there is need for time synchronization to correct the drifting clocks from the reference time. There are several methods/principles in time Synchronization. [12] [4]

External Synchronization

In External Synchronization all nodes are synchronized with external time source, that can be via Internet using NTP or using GPS.

Internal Synchronization

In Internal Synchronization nodes are synchronized with each other via establishing a master-slave hierarchy base network or with a reference clock. It is not necessary to synchronize with an external time source.

Hybrid Synchronization

This method is similar to Internal Synchronization but the reference/master clock is tuned with an external time source.

2.2 Time Synchronization Technologies

	SERCOS	TTP	GPS	NTP	IEEE-1588
Spatial Extent	Local Bus	Local Bus	Wide Area	Wide Area	A few Sub-nets
Communications	Bus	Bus or Star	Satellite	Internet	Network
Target Accuracy	Sub-Microseconds	Sub-Microseconds	Sub-Microseconds	Few Milliseconds	Sub-Microseconds
Hierarchy	Master/Slave	Peer ensemble	Client/Server	Distributed	Master/Slave
Administration	Configured	Configured	N/A	Configured	Self-Organizing
Hardware Assisted	YES	YES	YES	NO	Optional (For High accuracy)

Table 2.1: Comparison of Time Synchronization Technologies [5, 8, 17]

IEEE 1588

A protocol, which establishes master-slave hierarchy between LAN connected devices, and enables them to share timing information in order to synchronize their clocks with a reference time. It is a self-organizing protocol which uses existing infrastructure instead of dedicated bus to synchronize nodes up-to sub-microsecond level.

NTP:

The main focus of NTP protocol is on the systems, which are spread over a wide network, the target device is synchronized with a time server over the internet.

GPS:

The GPS systems communicate via satellite, which provides different services including timing. Regarding time synchronization, GPS is used for autonomous systems which are remotely located in certain area and receiving timing information via satellite communication.

TTP and SERCOS:

TTP and SERCOS are primarily used in motion control applications, where the systems are strongly integrated. Synchronization is a subset of these protocols, protocol applications are not limited to only time synchronization.

2.3 Overview of IEEE 1588 Precision Time Protocol (PTP)

IEEE-1588 Precision Time Protocol (PTP) was first introduced in 2002 [10] and then further extended in 2008 with extra features (e.g Transparent clocks) [11]. The standard defines a self-organizing or administrative actions free protocol for the synchronization

of time. The protocol is ideal for a condition where the time awarded distributed systems are networked via multicasting supported local area network. It provides flexibility in the term of requirements in order to establish a synchronization setup from hardware assisted PTP systems and PTP supported network components (Switches, Routers).

In an established PTP network, there can be protocol supported (e.g. Ordinary and Boundary clocks) and also other devices (e.g. Printers and non-PTP Bridges/Routers), which may not aware of PTP system. The hierarchy is established using PTP-devices, which are mainly real-time clocks. In the protocol hierarchy, top-level clock is called Grandmaster-Clock, which is treated as a reference time in the network and other clocks can perform a role of slave or master. In order to extend hierarchy and improve performance, a PTP-device (Boundary Clock) can also be a slave to its master and also time-source to other slave devices. The Best Master Clock Algorithm (BMCA) ensures the self-organizing property of protocol, by enabling PTP device to choose a specific role in the hierarchy, either in any case of failure of a master clock or in obtaining a master clock role on the base of clock data sets.

2.3.1 Scope of PTP Standard:

The protocol includes detailed implementation aspects of the protocol on UDP/IP based local area network along with messages exchange mechanism, standards offset calculation and management procedures are also described in detail. Along with mandatory specification, some optional features (e.g Unicast commutation, Alternate timescale. etc) are also defined for intended applications. In order to ensure backward compatibility a guideline with comparison is also described in the standard.

The standard also includes generalized software aspects, categorization of standard messages and their role in time information calculation (Two delay calculation mechanism). For PTP network management, a management procedure/sub-protocol is defined with dedicated message types to fetch information from network [3]. PTP devices role in term of time information processing is also specified.

2.3.2 Protocol Standard Messages

The whole hierarchy of PTP system is based on the PTP messages exchange, which enables every PTP device from the bottom level of the hierarchy to synchronize with their master device, this chain reaches to Grandmaster clock, which is also called reference clock. Time the message is sent or received at master/slave device is recorded, the precision of this time measurements is important in form of timestamp, it sets a fundamental level of accuracy for time synchronization. PTP messages are categorized in General Messages and Event Messages.

Event Messages

Event messages are those messages, which are critical for accuracy in both delay calculation mechanism of PTP. Disturbance during transmission of event messages can possess a huge impact on overall accuracy of the protocol. (Pdelay_Req, Pdelay_resp, Sync and Delay_Req).

General Messages

General messages are ordinary data transmitting or configuration messages, which are not timestamped but used to transport the timestamping informing and play a vital to setup protocol. (Pdelay_Resp_Follow_Up, Follow_Up, Delay_Resp, Announce, Management and Signalling messages).

2.3.3 Protocol Standard Devices

Ordinary Clock

The Ordinary device is a single port clock, which can be master or Slave clock. It communicates with other PTP devices through a single communication path. [10]

Boundary Clock

The Boundary clock node has more than one ports to communicate within the network through multiple communication paths. Boundary device is complete PTP implementation, each port of Boundary clock act as Ordinary clock. One port of the device can be a slave to a Grandmaster (GM) and other port can be a master to other PTP devices. So, in the case of failure of GM, it will act as time source in the network. [2] [6]

Transparent Clock

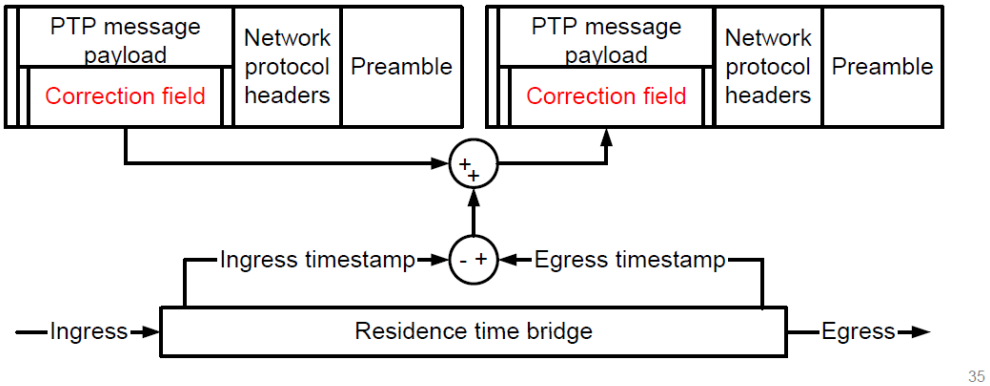

Figure 2.1: PTP Transparent Clock [9]

The Transparent clock was first defined in the second version of IEEE-1588 standard. There is no master/slave state in Transparent clock, according to the defined protocol, there is no need for synchronization in Transparent clock with Master/Grandmaster clock. It has multiple communication ports for transferring PTP messages. During forwarding messages from input port to output port, it computes delay caused by a device in transferring process of the PTP event messages. The computed delay is called residence time, this residence time then added to designated correction field part of particular timing message. Slave clock adjusts the time using this residence time in order to

overcome the fluctuation caused by the other network establishing devices (e.g Bridges). [2] [6] [11] [9]

2.3.4 Message Exchange and Delay Computation

In synchronization process, the time information is exchanged between master and slave for the calculation of offset between clocks and delay caused by the network infrastructure. For offset correction, Sync message is sent periodically from the master, which contains the exact time of master but during the process of transferring that message, it passes through the different layers of system which causes undefined time error, in order to overcome this problem the Follow_Up message is sent from master which contains the timestamping information of reading of master time in previous step (sync message).

For network delay computation there are two mechanisms defined, which are used with different combination of network infrastructure or PTP devices. Overview of the mechanisms are given below.

Delay Request-Response Mechanism

This mechanism yields mean path delay (Average of the time taken by data to travel between slave and master). The calculation is performed through the exchange of following messages.

- **Sync message (M):** Timestamped message sent to slave.
- **Follow_Up message (M):** Contains exact time when the Sync Message was sent.
- **Delay_Req message (S):** Timestamped message sent to master for network delay calculation.
- **Delay_Resp message (M):** Contains the exact time when the Delay_Req messaged received.

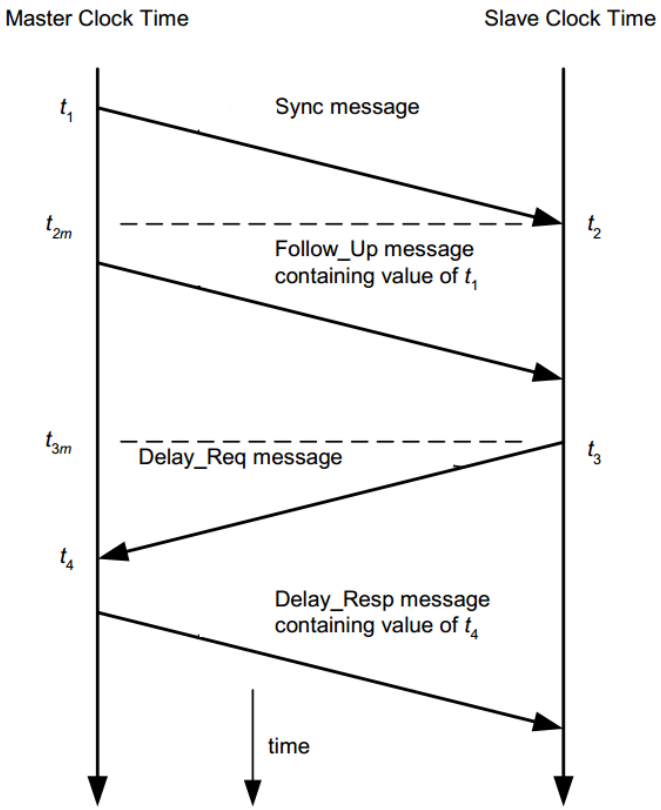

Figure 2.2: PTP Delay Request-Response Mechanism (End-to-End) [14]

As shown in Figure 2.2, Sync message is timestamped at t1 and sent to slave, practically the Sync message is sent at t2m due to unknown network delays. So, another Follow_Up message containing the timestamp value of t1 is sent to acknowledge the slave about the exact instance, when the Sync message was timestamped. Similarly, Delay_Req message is timestamped at t3 and received at the master at t4 instead of t3m due to delays, the master clock sends a response message, which contains the exact time when Delay_Req message was received.

$$MeanPathDelay = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{2.1}$$

$$OffsetFromMaster = (t_2 - t_1) - MeanPathDelay \tag{2.2}$$

Peer Delay Mechanism

In this mechanism of delay calculation, Sync and Folow_Up messages perform their roles same as the end-to-end mechanism but instead of Delay_Req message, the Pdelay_Req, and Pdelay_resp messages are exchanged. The main difference is that these messages are sent to the port which is immediately connected. The connected port sends a reply in the form of Pdely_Resp message, then node calculates the delay between two immediately

connected port to each other, due to this approach every port in the network must support PTP protocol.

In presence of transparent clock, the delay between master and slave is calculated by adding specific peer delay to the residence time of Transparent clock. So in the case of any accidental change in network hierarchy network delay fluctuation can be handled efficiently. [19] [11] [9]

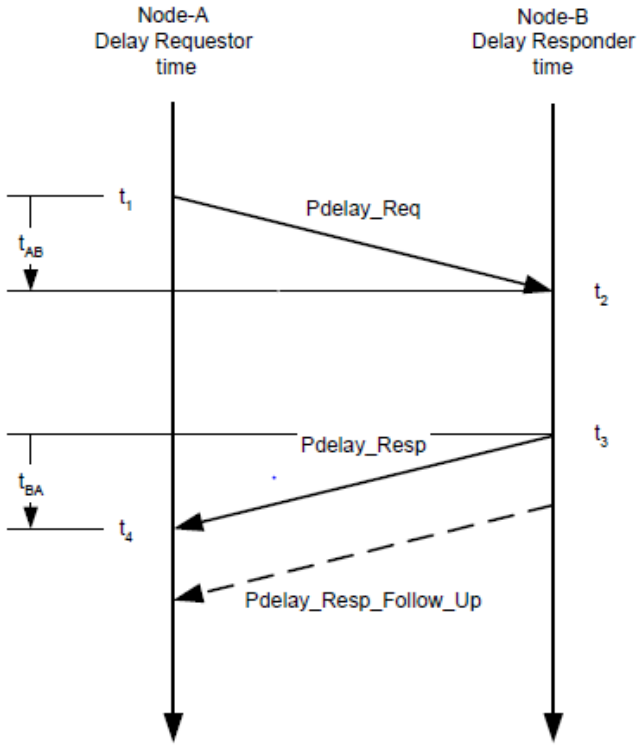

Figure 2.3: PTP Peer delay Mechanism [11]

2.3.5 Protocol Hierarchy Establishment Mechanism

The protocol hierarchy is based on the master-slave relation of clocks. As a self-organizing protocol, these states of the clock are determined by the BMCA (Best Master Clock Algorithm), every ordinary (OC) and Boundary clock (BC) is equipped with BMCA. The algorithm enables OC/BC to determine their states locally instead of negotiating within a network to decide the clock state. [3]

For every clock, there are standard clock properties (Priority1, Class, Accuracy, Variance, Priority 2 and Unique Identifier) are defined, which are used by the BMCA to compute the state of clock [11]. These properties can be used to manipulate the behavior of BMCA in order to assign the particular role to a specific clock. For example, as a Grandmaster.

In PTP-1588-v1 [10], clock properties are advertised using Sync message but in second version [11], properties are advertised using dedicated Announce message. A clock, either explicitly configured (clock properties) as the best clock or consider itself on the bases of BMCA as an eligible master clock, in both cases the clock advertises its clock properties via sending announce message in the network. In case of failure of master clock or recognition of another better clock in the network, then the master-state associated messages (Sync and Announce) stops and another clock takes the role of best master clock. [16]

Chapter 3

PTP Infrastructure in Linux

The journey of time synchronization begins with NTP (Network Time Protocol) and David Mills is known as the father of NTP. Many of Mills purposed method for timekeeping and synchronization of internal and external clocks can be seen in more advanced and accurate time synchronization protocol like PTP. [15]

The first version of PTP was standardized in 2002, three years later Kendall Correll introduced a first opensource software-only solution (*ptpd*) [13] for implementation of PTP on Linux based systems. The solution became a base for further development in this domain. Later in 2009, Patrick Ohly introduced hardware timestamping in Linux kernel [7], Ohly then altered existing version of ptpd and extend his support for hardware supported synchronization [14]. Although hardware timestamping mechanism was already introduced in Linux kernel but there was no proper solution to control the hardware clock. So, in 2010, Richard Cochran introduced PTP clock infrastructure in Linux [7]. Later in 2011, Cochran presented the LinuxPTP tool for hardware and software based time synchronization solution using hardware timestamping and PHC API of Linux kernel [15]. These opensource solutions and Linux kernel support are playing a vital role in the development and widespread use of this protocol. Many developer and researchers are using and extending these tools to realize their concepts on different platforms for intended applications.

3.1 Timestamping Mechanisms

3.1.1 Software Timestamping

Timestamps can be generated at different layers of a network, software timestamping is most widely and easily available option. In software timestamping easiest way is to copy the OS time at application level and merge it with the intended network packet. Another way is to use SO_TIMESTAMP options of Linux kernel which is the most widely exercised options in opensource implementations of PTP protocol.

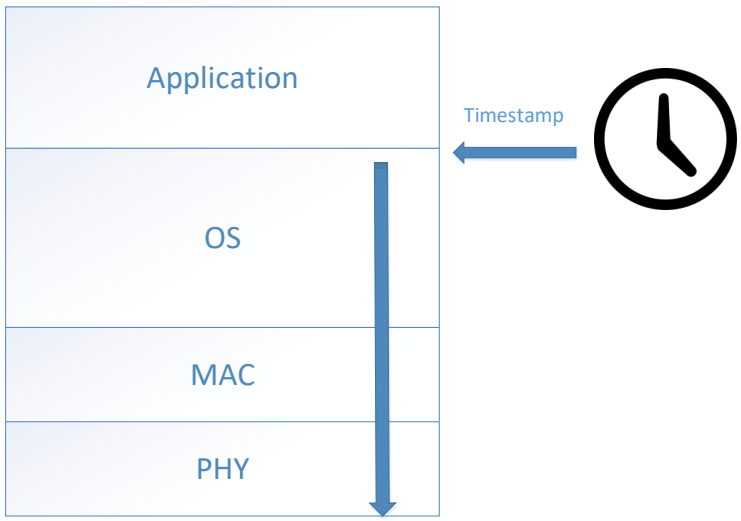

Figure 3.1: Software Timestamping

The figure above reflects software timestamping process in contrast with network layers. As the packets are timestamped at userspace using system time. Practically, the system time is stored in memory, accessing memory/system time using normal kernel routines results jitter which varies system to system and furthermore timestamped packet still need to pass through other layers of the network in order to reach physical channel of the network, which also causes some amount of error.

3.1.2 Hardware Timestamping

Hardware-based timestamping effectively reduces the jitter caused by OS level uncertainties and reduces the error resulted by network layers hardware timestamping can be either on *MAC* or *PHY* layer, it depends on the type of hardware. Linux network stack supports hardware timestamping by using SO_TIMESTAMPING feature. In order to use SO_TIMESTAMPING feature, first the device driver need to be configured using Kernel SIOCSHWTSTAMP options through *ioctl* call. After successfully configuration user can enable and disable timestamping on outgoing/incoming messages by using SO_TIMESTAMPING socket options. [7]

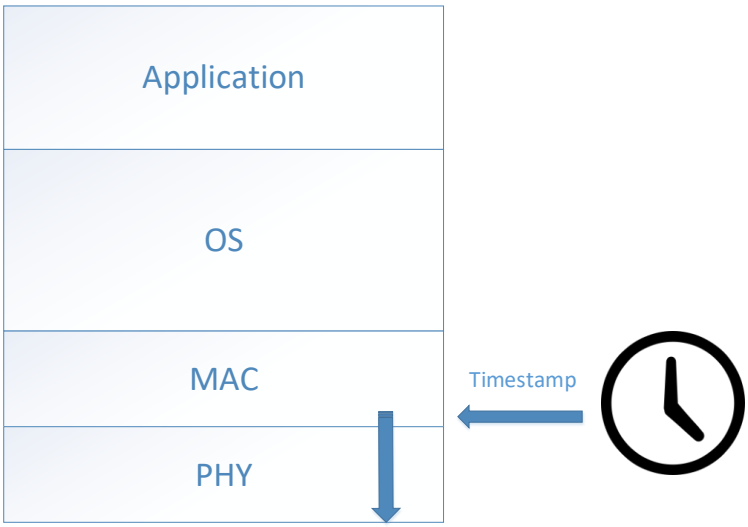

Figure 3.2: Hardware Timestamping

3.1.3 Linux kernel Support for Timestamping

There are different Linux socket interfaces for generating a timestamp for incoming/outgoing packets at different level of network stack. The following are the currently available options. [18]

SO_TIMESTAMP :

The kernel option timestamps incoming network traffic by using Linux system time. Resolution of the timestamp is in microseconds.

SO_TIMESTAMPNS :

This timestamp generation mechanism is similar to previous option but when the timestamping information is retrieved using recvmsg() function, regarding information is sent back via *timespec struct* in nano second resolution.

SO_TIMESTAMPING :

This kernel socket option supports multiple types of timestamp request including hardware timestamping. The option enables to generate the timestamp on incoming and outgoing packets.

Following are the available parameters for configuring SO_TIMESTAMPING option for timestamp generation.

- SOF_TIMESTAMPING_RX_HARDWARE
- SOF_TIMESTAMPING_RX_SOFTWARE

- SOF_TIMESTAMPING_TX_HARDWARE
- SOF_TIMESTAMPING_TX_SOFTWARE
- SOF_TIMESTAMPING_TX_SCHED
- SOF_TIMESTAMPING_TX_ACK

Following are the available parameters for configuring SO_TIMESTAMPING option for timestamp reporting.

- SOF_TIMESTAMPING_SOFTWARE
- SOF_TIMESTAMPING_SYS_HARDWARE
- SOF_TIMESTAMPING_RAW_HARDWARE

Furthermore, the detailed operational and implementation level information can be found in related kernel documentation. [18] v Additionally, regarding hardware timestamping SO_TIMESTAMPING option was first introduced by Patrick Ohly 02/2009 and then adopted in 2.6.30 version of Kernel, before that options SO_TIMESTAMP and *S*O_TIMESTAMPNS was used for timestamping of packets but these interfaces only provide the option to timestamp packets at software/application level. [7] [14]
.

3.2 PTP Clock Infrastructure and Control API

The first open source PTP solution (ptpd) was released in May 2005, which supports software only PTP implementation. Although many attempts are made to integrate PTP hardware support by using Linux *ioct* procedures and adding many pre-processor commands in the software-only solution of Kendall Correll's, which made existing code more unsuitable for support of multiple hardware [7]. After the official integration of hardware timestamping, which was proposed by Patrick Ohly, there was no PHC clock infrastructure in the Linux although Ohly proposed two concepts (Assisted system Time and Two-level PTP) for the synchronization of system clock with PHC in his publication [14] but there was no efficient mechanism was proposed to control the PHC clock[7]. In order to overcome these problems, In 2010 Richard Cochran introduce a PTP clock infrastructure, which was then integrated to Kernel version: 3.0. The solution includes the PTP hardware clock (PHC) drivers architecture and a standard mechanism in form of API to control PHC.

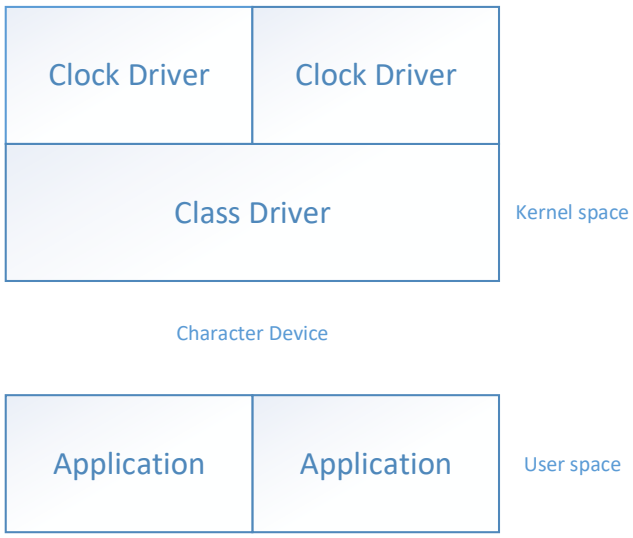

Figure 3.3: PTP Infrastructure in Linux [7]

The above figure shows the clock infrastructure, where the class drivers covers the more generalized features and specific clock need to provide the clock driver which covers the hardware aspects of specific clock. The specific clock has to register their clock driver with the class driver which will generate the Character device, the Character device will be accessible to userspace via PHC userspace API.

The main purpose of developing Linux Kernel API was to minimize the effort to develop the driver for different devices. So, Class drivers do many generalized tasks for all clock drivers like creation of character device, validation of *ioct* calls and management of the timestamped event ordering. The approach which is used to control the PHC is quite similar as NTP timer model which also reduce effort in development. [7]

ioctl	NTP / POSIX equivalent
PTP_CLOCK_ADJFREQ	adjtimex/ntp_adjtime
PTP_CLOCK_ADJTIME	(none)
PTP_CLOCK_GETTIME	clock_gettime
PTP_CLOCK_SETTIME	clock_settime
PTP_CLOCK_GETCAPS	(none)
PTP_CLOCK_GETTIMER	timer_gettime
PTP_CLOCK_SETTIMER	timer_settime
PTP_CLOCK_FEATURE_REQUEST	(none)

Figure 3.4: PHC API features [7]

The table above shows the clock operations and their comparison with NTP timer equivalent. As these ioctl names are reflecting their function like GETCAPS is used to get the PHC capabilities and ADJFREQ ioctl is used to adjust the clock frequency by ppb parameter. [7]

Chapter 4

Design and Implementation

In order to implement the PTP protocol on Linux based SoC platform, we chose LinuxPTP, a most reliable opensource PTP implementation for UNIX like operating systems. Although there is another solution (*PTPd*) available which also claims hardware timestamping and PHC control based implementation, due lack of compatibility for multiple hardware, we preferred to use linuxPTP which is compatible for chosen hardware platform (Beaglebone Black) and easily extendable to implement the additional features. For hardware platform, *Beaglebone* is chosen due to its support for hardware-based timestamping for PTP protocol and wide acceptance in the opensource community as reference implementation platform. For performance analysis of PTP protocol on selected hardware-software combination, *stess-ng* tool and *iperf* are used to put some stress on reference implementation platform in different dimensions (CPU, Network). Data collected during this process is then cleaned and imported to *MATLAB* workspace, where data is represented in bar-charts and graphs.

4.1 Tools and technologies

4.1.1 LinuxPTP

LinuxPTP was introduced by Richard Cochran[15] in 2011 after integration of PHC API in Linux mainline kernel. There was two synchronization mechanism purposed by Patrick Ohly[14] to synchronize the system and PHC clock. The LinuxPTP solution closely resembles to the second method which is "Two-Level PTP", on the other hand, the first method is considered to be inefficient for proper synchronization [15].

As shown in diagram 4.1, the packets from the master node first arrives at PTP HW Clock, where according to configuration and timestamping capabilities the packets are timestamped, the timestamped packets are accessible to PTP stack via SOTIMESTAMPING Linux socket options. In next step, the PTP stack calculates the correction and adjust the Hardware clock via Standard POSIX commands. As the PHC subsystem is dependent on the PPS subsystem for the synchronization of system time, So, PPS signal is timestamped by the system using ISR routine and then made accessible to userspace via standard PPS interface of NTP. [15]

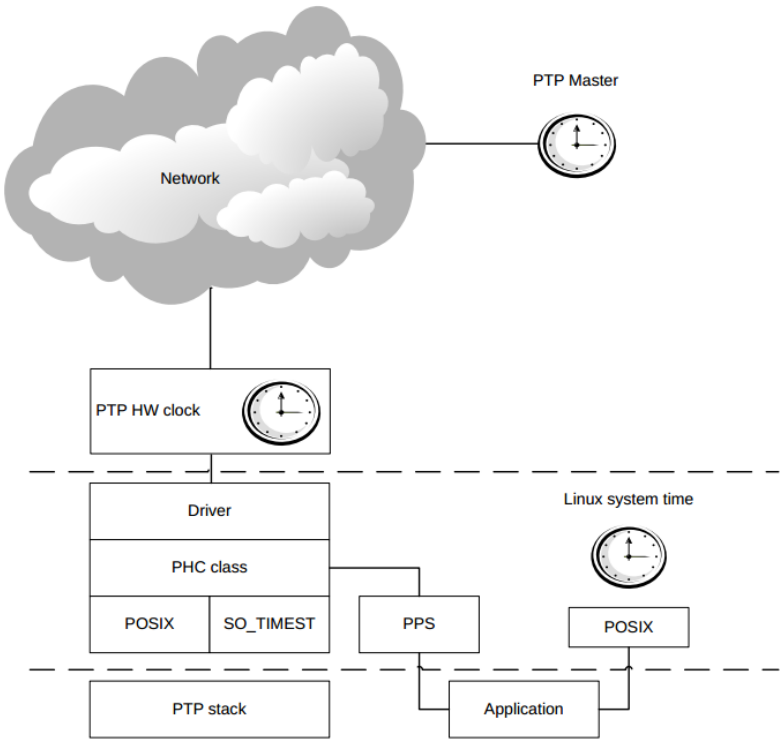

Figure 4.1: PTP Clock based Time Synchronization in Linux [15]

The LinuxPTP project yields multiple executable files in order to establish the whole two-step synchronization mechanism.

ptp4l:

The *ptp4l* tool synchronizes (by default) PHC clock with master clock in network. If the system does not have PHC then it synchronizes the system clock with a master clock using software timestamping, in this case, there is no need of phy2sys tool. For configuration, there is a default configuration file is provided, where the default behavior of the tool can be changed. For customized configuration, a new configuration file can be made in a standard way, which can be passed to the tool via proper command line argument.

pmc:

The *pmc* is the realization of PTP management client as defined in the standard. The tool is used to get the extra information from the network like identity, pathdelay, accuracy and other. The commands from pmc are transferred to all the clients but there are options to give a targeted command in the network.

phy2sys:

The *phc2sys* tool synchronizes Linux system clock to the PHC. In whole process the the *phc2sys* is synchronized with *ptp4l*, where system clock act as slave and PHC plays a role of master clock.

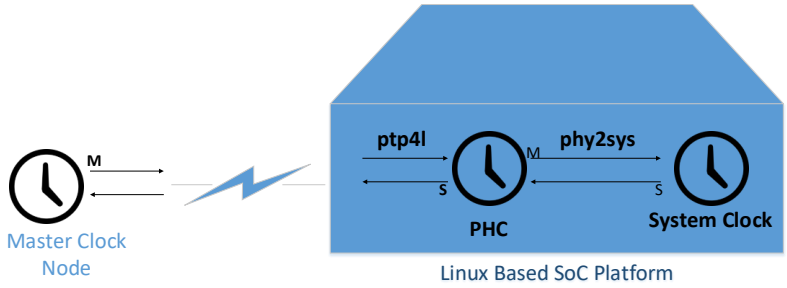

Figure 4.2: LinuxPTP based Hardware Assisted Time Synchronization

4.1.2 PTPd

PTPd is the one of first opensource implementation of IEEE-1588 protocol for UNIX like operating system. The first version of PTPd only supports software timestamping. Later, PTPd also adopted hardware timestamping and PHC API.

4.1.3 stress-ng

stress-ng is a tool to perform the different stress test on Linux system in a scalable way. Test includes I/O stress, CPU, and Network load.

4.1.4 iPerf

iPerf is a cross-platform tool, the tool is used for network performance measurements. In this project, iPref is used to create the UDP stream with specified bandwidth on established PTP network for performance analysis of PTP implementation under scalable network load.

4.1.5 Matlab

Matrix Laboratory (Matlab) is a complete set of tools. Matlab includes IDE (Integrated Development Environment), programming language and libraries. Matlab is used for creating models, develop algorithms and for analyzing data. In this project, the Matlab is used only for data preparation and representation of data.

4.1.6 Beaglebone Black

Beaglebone (BBB) is an opensource development board, equipped with TI M335X Sitara ARM Cortex-A8 process. Beside common interfaces (I2C, SPI, UART, USB), it also supports specialize industrial protocol (CAN, PROFIBUS, PORFINET, PTP-1588).

The capabilities of platform can be extended further with CAPES (capes are broads extensions with additional features). The main reason for selecting this platform is for its supports to hardware based timestamping though the MAC layer.

4.2 Design Consideration and System Hierarchy

Flexibility is one of the forefront features of the PTP protocol and it is optional to use all PTP supported network components, the network components includes routers/switch. Although the BBB device provides the hardware support but the used network router (TP-Link TL-WR940N N450) does not provide PTP support. So, the maximum accuracy and the precision is not guaranteed. On the other hand, only end-to-end delay calculation mechanism is exercised.

The following diagram shows the architecture of the test environment.

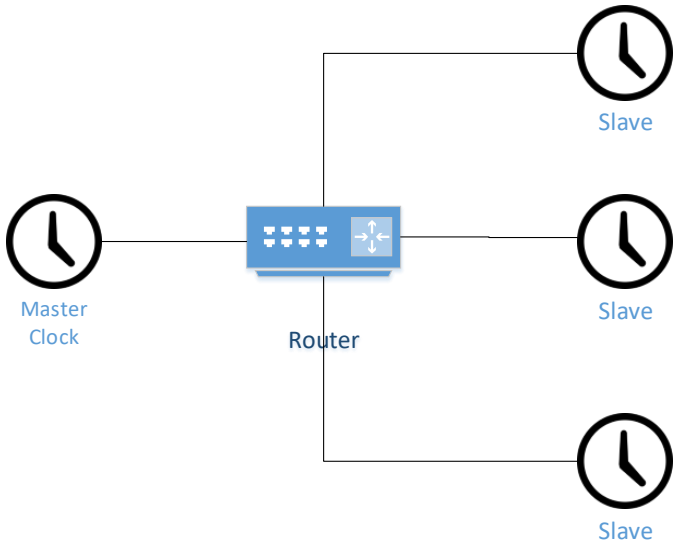

Figure 4.3: Architecture of Test Environment

Chapter 5

Test and Measurements

In order to establish PTP network for software and hardware based timestamping solutions, there are necessary tools and configuration need to be installed and configured. There are different ways to install and configure environment for PTP network.

First, it is needed to know the capabilities of the hardware, either the clock nodes support hardware-assisted time synchronization or not. Timestamping capabilities in Linux kernel version 3.4 or later can be checked through Kernel ETHTOOL_GET _TS_INFO *ioctl* calls. Ethtool is a Linux based tool give the ability to inquire the timestamping capabilities of the hardware. Following is a snapshot of the of Linux terminal output showing timestamping capabilities of selected SoC platform (Beaglebone), which is retrieved using Ethtool.

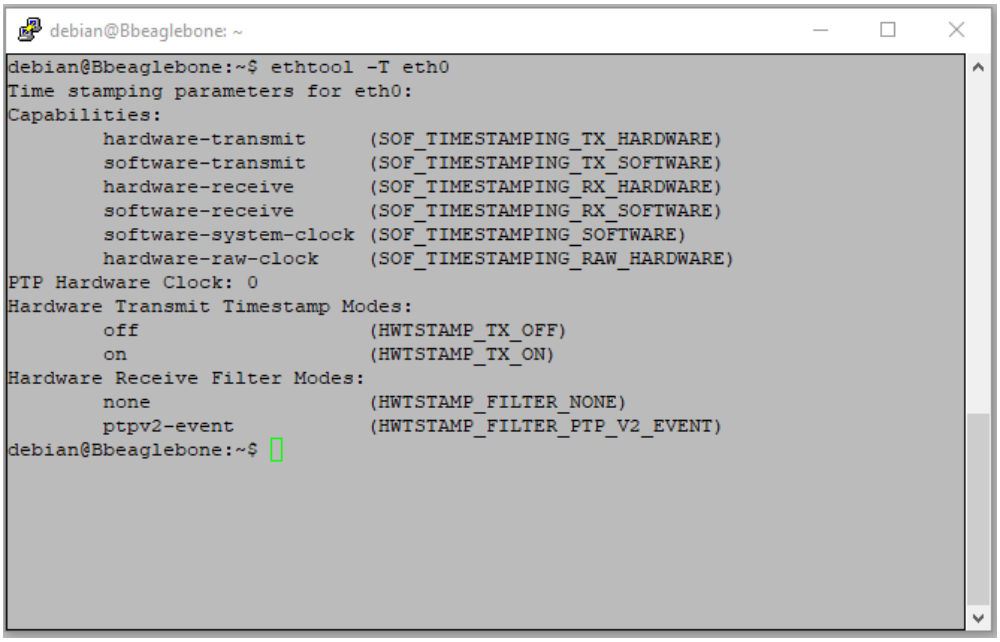

Figure 5.1: Timestamping Capabilities of Beaglebone

The above figure reflects the hardware timestamping support of Beaglebone. In order

to utilize this hardware assistance following Kernel option should be enabled.

```
1 CONFIG_PPS
2 PTP_1588_CLOCK
```

There are two ways to get a Linux kernel with enabled above options. First, compile a customized kernel with desired kernel options. Second, obtain a pre-compiled kernel for selected hardware with required configuration. Although, obtaining a pre-compiled kernel is the easiest way to establish a hardware-assisted PTP setup but most of the time pre-compiled kernel is shipped with many generalized modules and desktop environments, which may not necessary for intended applications and sometimes these additional unnecessary components of the kernel and root file systems impose considerable amount of extra stress on the system, which further leads to uncertainty in of whole synchronization process. On the other hand, compiling the customized kernel using conventional way is not always a favourite option. So, instead we can use the Yocto project, for building a customized Linux system image including bootloaders and root file system. Yocto project is an opensource solution for building customized kernel for embedded systems. Yocto project uses Beaglebone as a reference platform. In **Appendix B**, the procedure to build the Linux kernel and porting pre-compiled kernel in Beaglebone is briefly described.

5.1 Test Case Scenarios

After installing and configuring necessary tools and software, which includes enabling the Linux kernel for hardware-based PTP support and installing LinuxPTP, stress-ng and iPerf tools. The SoC-based clock nodes are ready to establish PTP network for test and measurement purpose.

Regarding test case scenarios, first, pure software-based network is established and results are collected through standard logging. In next section the results from pure hardware based solution are collected, after that, a comparison between the software and hardware-based solution is drawn in the context of offset from master clock and precision of synchronization process. In following sections, there are some CPU, I/O and network-based stress tests are conducted on hardware-assisted time synchronization implementation.

In order to present the results in a coherent form, data fitting process with default Smoothing Splines model from Matlab tool is used. In some graphs, we found possible and necessary to present the actual data points parallelly. So, where ever the Data is shown in the description window of the graph, it shows the actual data of respective slave.

5.1.1 Software Timestamping

In software timestamping based test scenario, there is no special need for kernel configuration. In software-based scenario, network hierarchy is established using one Beaglebone device acting as a master clock and three Beaglebone devices acting as slave clocks in LAN based network. The network is established using a low-end home router, which is unaware of PTP hierarchy. Due to unsupported network components, end-to-end delay computation mechanism of the PTP protocol is exercised.

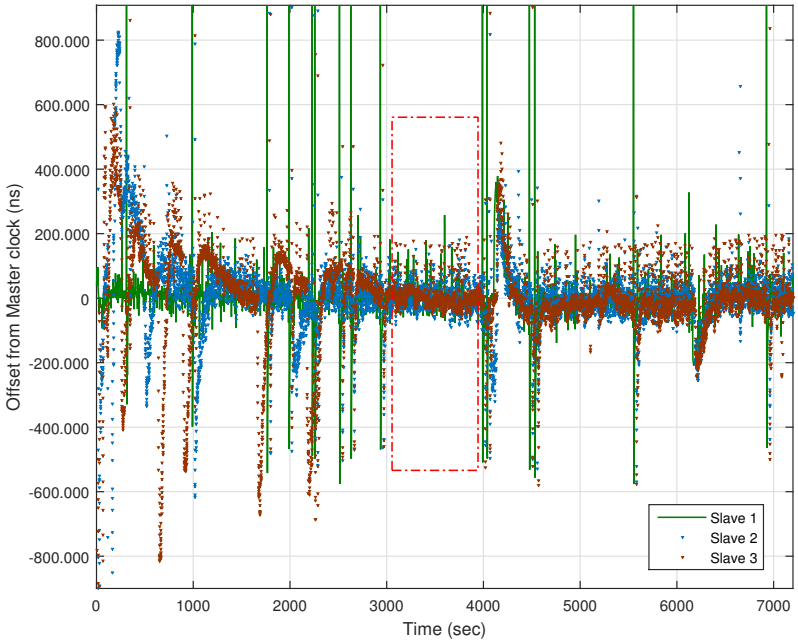

Figure 5.2: Software Timestamping based Time Synchronization I

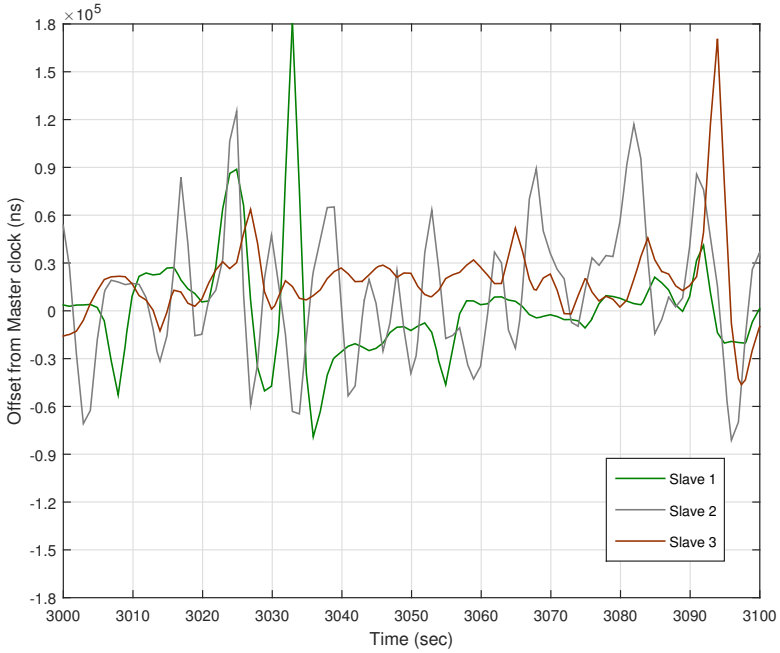

Figure 5.3: Software Timestamping based Time Synchronization II

The figure 5.2 shows the offsets of slave clocks over the span of approximately two hours, where the drift of slave clocks are up to 2ms and it can also be seen that clocks are showing uncertain behavior. Overall the offsets are between ± 0.5ms but due to unknown error, there are sudden changes in offset in all clocks, which may be the result of a change in speed of the clock. If we take a closer look at the relatively stable part of test duration, which is highlighted by a rectangular shape in figure 5.2.

The figure 5.3 is drawn on the basis of data taken from stable synchronization span of the figure 5.2. In this graph, only first 100 samples of data are considered, in order to visualize the results more clearly. During whole span (approx 15 min) of synchronization, the offset remained under 300 microseconds.

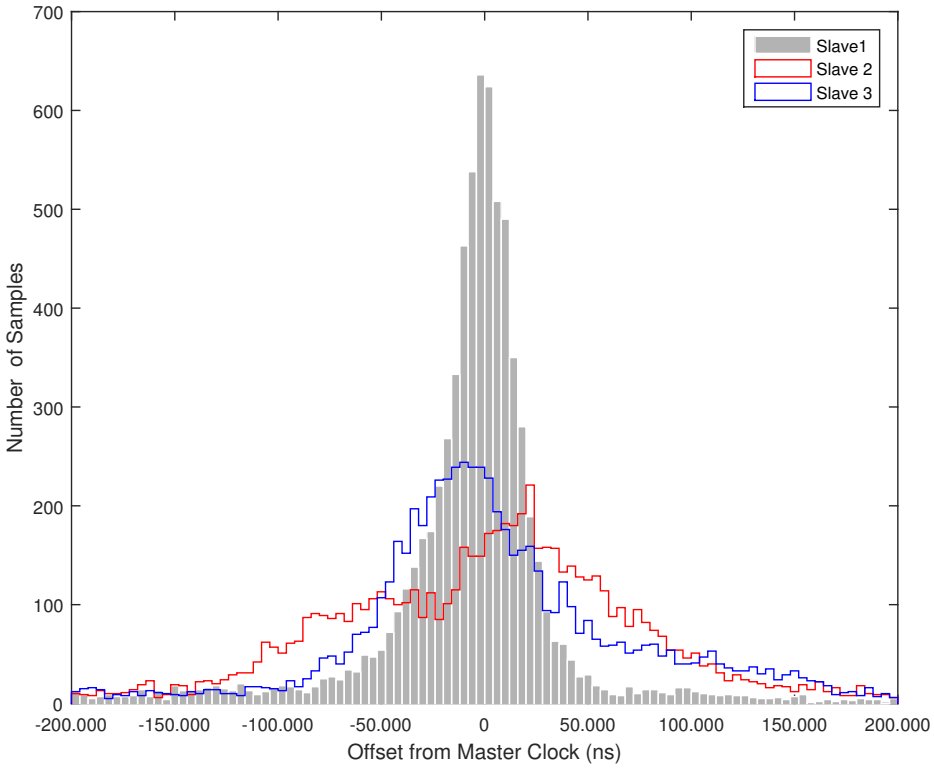

Figure 5.4: Precision of Software based PTP Implementations

The figure above shows the precision of clocks, where the number of samples by specific clock are plotted between the range of -200us to 200us. It can be seen that slave 1 have relatively better precision as compare to other slaves clocks, where the offset fluctuation can clearly be seen.

5.1.2 Hardware Timestamping

The graph below shows the synchronization of slave clocks over the period of twelve hours. The synchronization period is divided into two parts. In the first part, the syn-

chronization network consists of three slaves and one master clock, performing their roles for approximately 1.3 hours and in the second part of synchronization period, two slaves are removed from network in order to analyze the behavior of synchronization process with fewer slaves communicating with the master clock. It can be seen that after removing slaves form network, results form remaining slaves are relatively stable and also remained stable for long period of time.

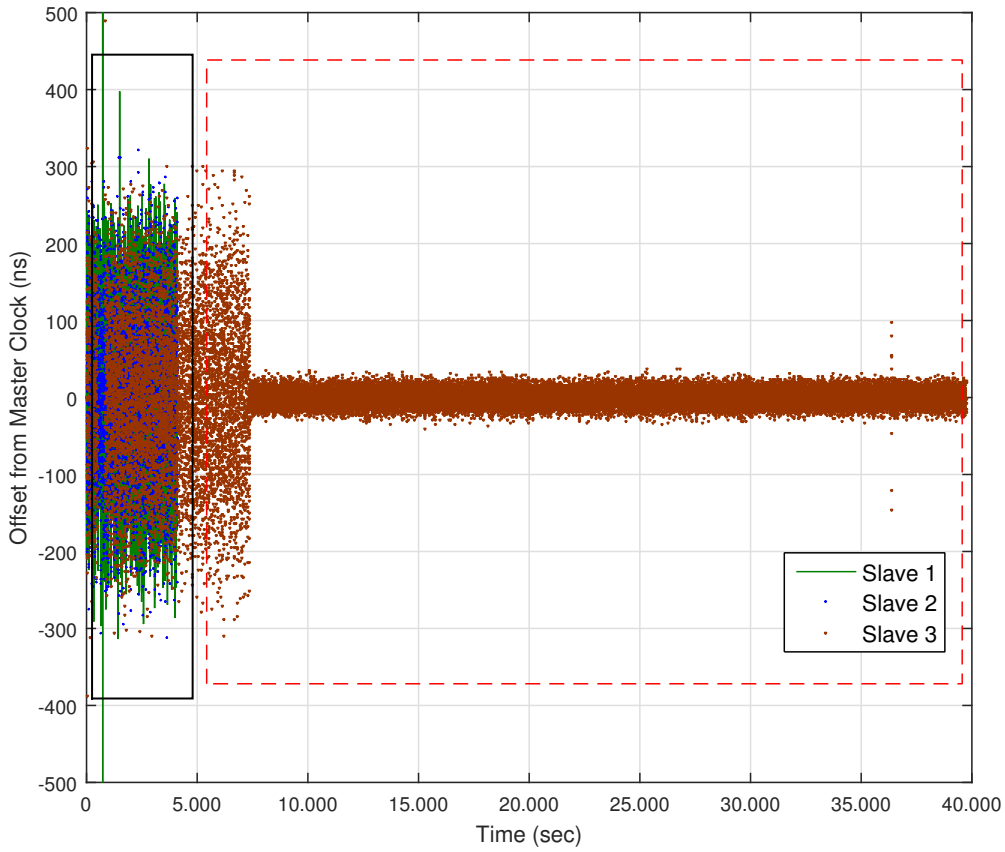

Figure 5.5: Hardware Timestamping based Time Synchronization

The graph 5.6 shows the magnified version of first half of the previous figure where three slaves are active in PTP network and continuously synchronizing their time with master clock by every second and the accuracy of slave clocks is between ± 250ns.

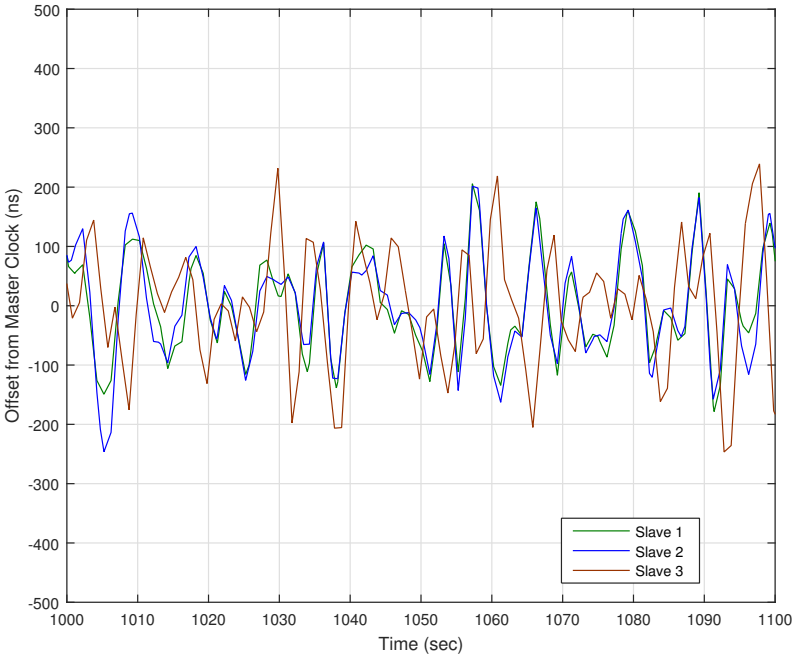

Figure 5.6: Hardware Timestamping based Time Synchronization (Multiple Slaves)

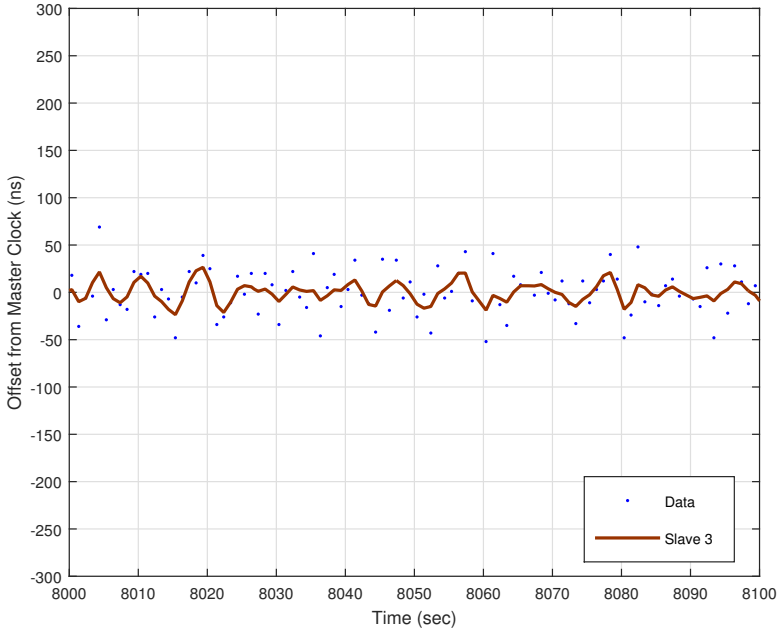

Figure 5.7: Hardware Timestamping based Time Synchronization (Single Slave)

The graph 5.7 reflects second half of the figure 5.5, where only one slave is correcting their offset with one master clock and other two slaves are removed from the network. It can be clearly seen that the results are remarkable with fewer slave devices in a network. The offset is fluctuating between the -50 to 50 ns, which is the highest accuracy achieved during all kind of test scenarios.

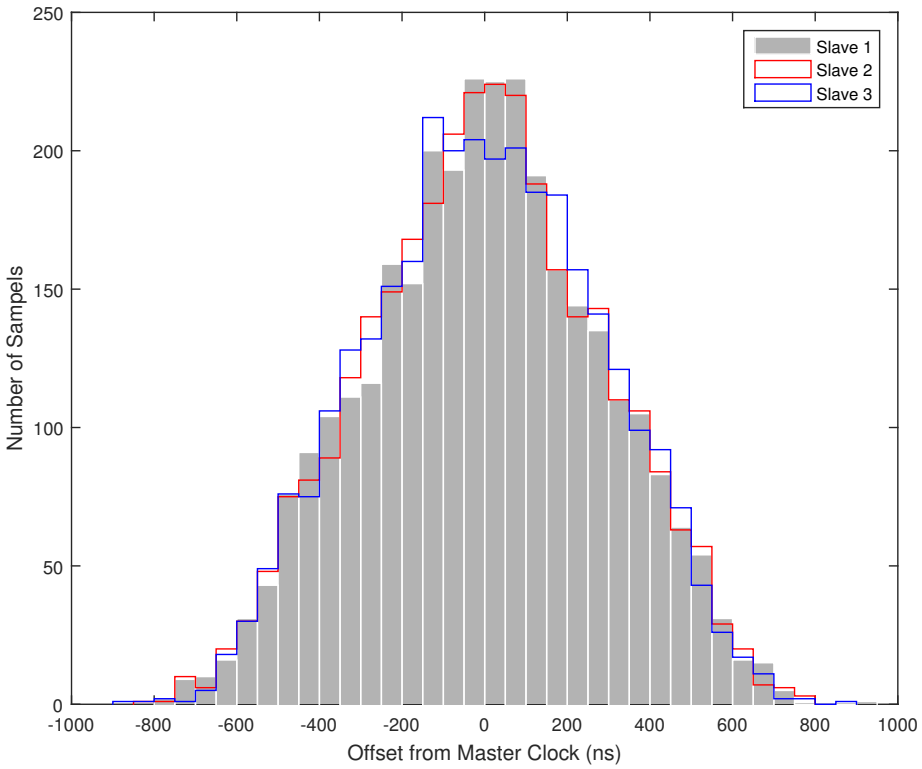

Figure 5.8: Precision of Hardware based PTP Implementation

The stability of clock can be noticed from above bar chart, in most of the results the offset is between ± 400ns, with four PTP devices connected via LAN network. On the other hand, almost all slave clock have similar precision unlike software timestamping base PTP network, where all slave clocks have different behavior in whole test case duration with similar configuration and environment.

5.1.3 Comparison of Software and Hardware based Synchronization

The previous sections shows that the hardware and software based synchronization not only differ in offset from the master but also in precision. It can be seen in following graph, after plotting software and hardware-based measurements on the same graph it is hard to distinguish that whether hardware timestamping is equal to zero or not.

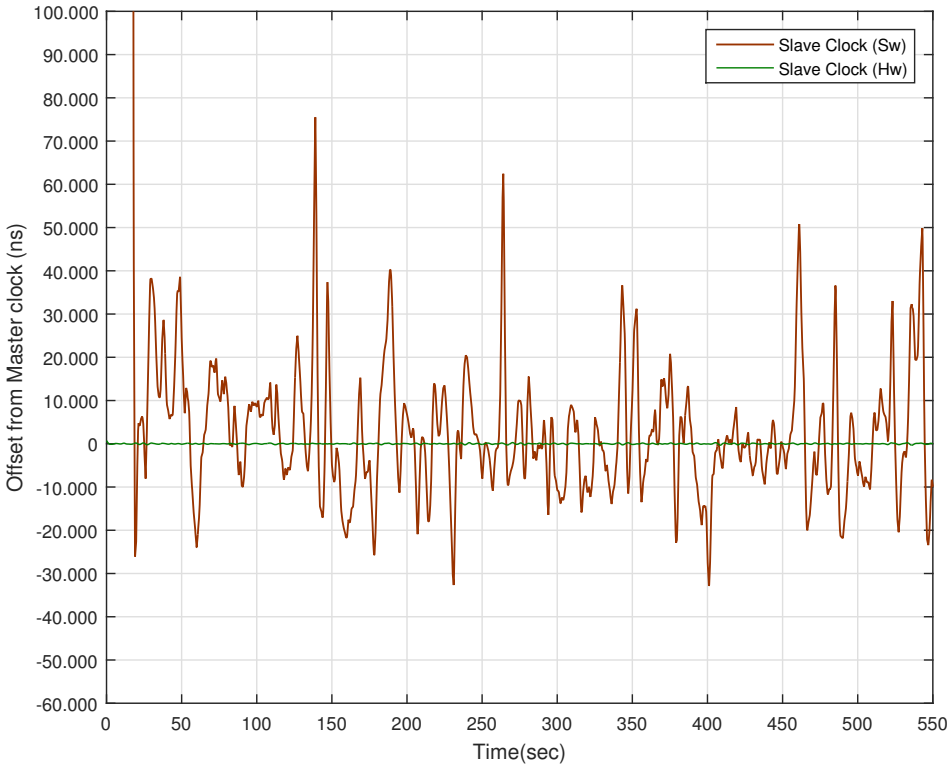

Figure 5.9: Comparison of Software and Hardware based Synchronization

5.1.4 Hardware Assisted Time Synchronization under CPU Load

It was expected that the hardware-assisted time synchronization will not have the prominent effect of CPU stress on the accuracy because the PTP messages are time-stamped at MAC/PHY layer. The graph 5.10 compares the three slave with simulated CPU load by using the stress-ng tool. The slave 1 shows the normal synchronization period, slave 2 device is stressed using stress-ng with 50% utilization parallel to synchronization process and similarly, slave 3 is stressed with 100% CPU utilization. In this test case, no considerable change in the behavior of devices are noticed, all the slave offsets are fluctuating between -300 to 300 nanoseconds.

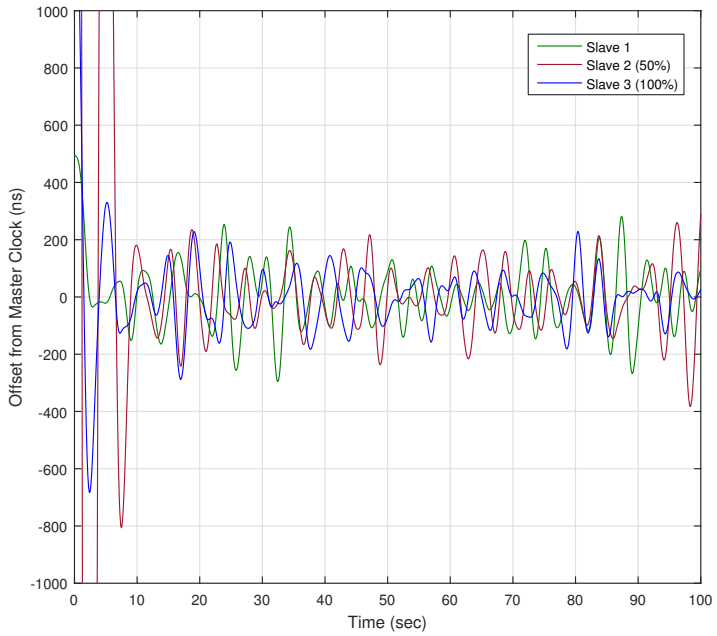

Figure 5.10: Hardware Assisted Time Synchronization under CPU Load

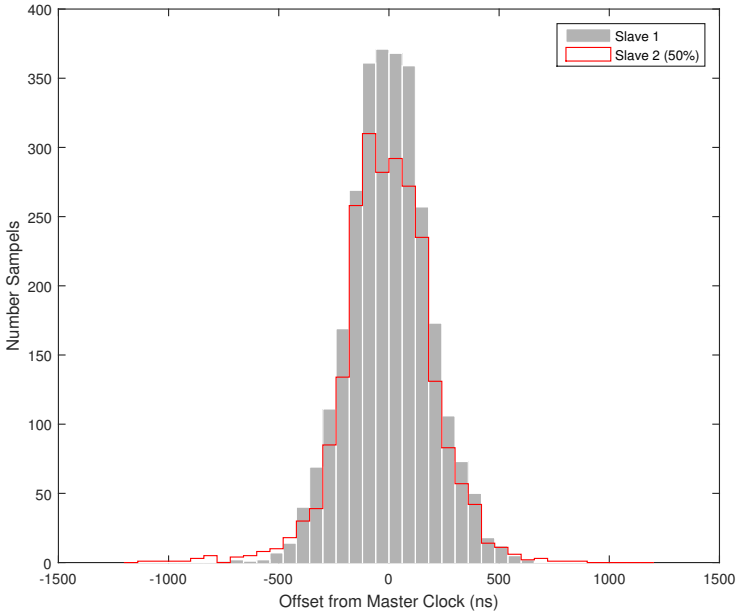

Figure 5.11: Precision of Hardware Assisted Time Synchronization under CPU Load

We have tried to find out possible effects on the precision but there is no consid-

erable change found in the precision of the synchronization. The figure 5.11 shows the comparison of two slaves, slave 1 is not experiencing any simulated load and slave 2 is stressed via 50% of CPU utilization. There is little change visible in the precision of clocks but this observation goes wrongs, when we compare it with 100% load, because with 100% load scenario there is no change found. So, it is concluded that there is no change observed in the precision of clock in current type of network hierarchy and environment. It is expected, there might be considerable effects in more stable PTP network hierarchy.

5.1.5 Hardware Assisted Time Synchronization under I/O Load

In order to perform I/O based test case scenario, I/O load type from stress-ng is used, which issues many tiny synchronous I/O reads and writes on a temporary file by utilizing Linux POSIX aio interface. In this case, there is no considerable change is found in accuracy and precision. The figure 5.12 shows the result, where both slaves (with and without load), are having the same accuracy.

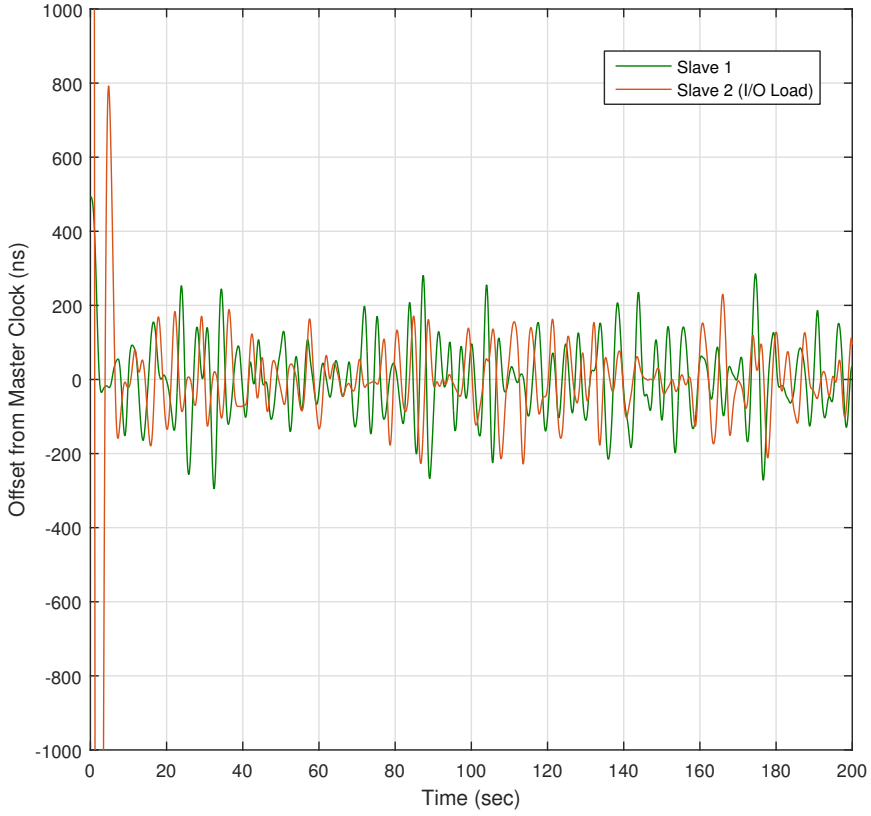

Figure 5.12: Hardware Assisted Time Synchronization under I/O Load

5.1.6 Hardware Assisted Time Synchronization under Network Load

The effects of network-based load highly depend on the type of switches and routers are used in network. In this project, the network components used to connect the slaves and master clock is not PTP supported. So, we need to considered that the results shown here can be different if more advanced networking devices will be used.

In order to simulate the network load, an extra Linux based device is connected to the same network and iPerf tool is used to create a multicast stream with specified bandwidth.

Additionally, in graph 5.13 and 5.16, the absolute value of data is considered in order to avoid the entanglement.

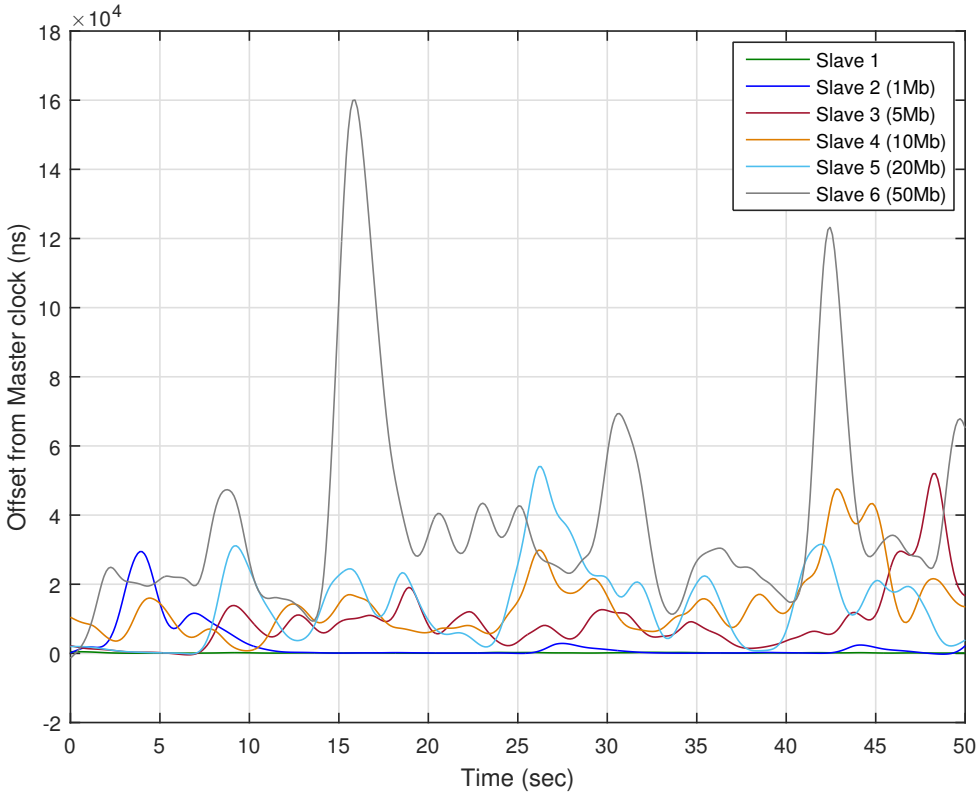

Figure 5.13: Hardware Assisted Time Synchronization under Network Load

The figure 5.13 shows, slaves and their respective master clock expose to network traffic between the range of 0Mb to 50Mb. It can be seen that, as the amount of the network traffic grows the offset from the master clock also rises. let assume that the 0-5Mb network load comes under the category of Low network load and 10-50Mb is the High load. First, we inspect the accuracy and precision in low network traffic and then High network traffic.

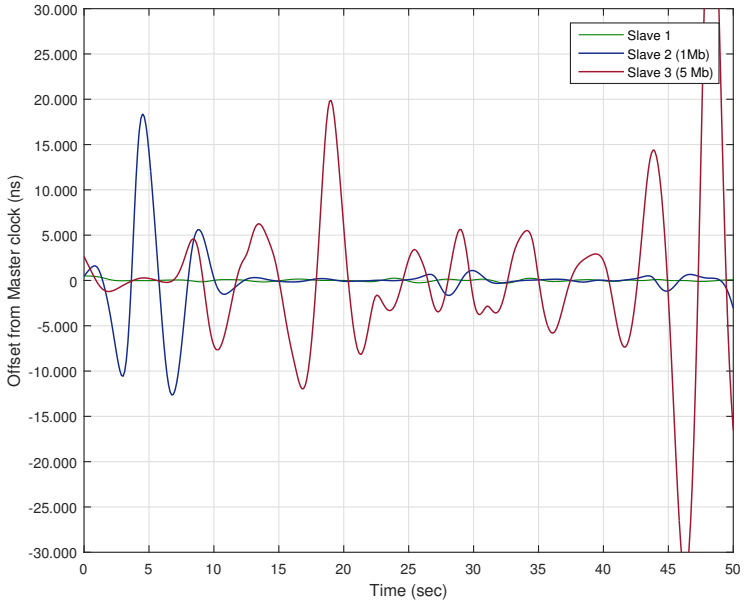

Figure 5.14: Hardware Assisted Time Synchronization under Network Load (Low)

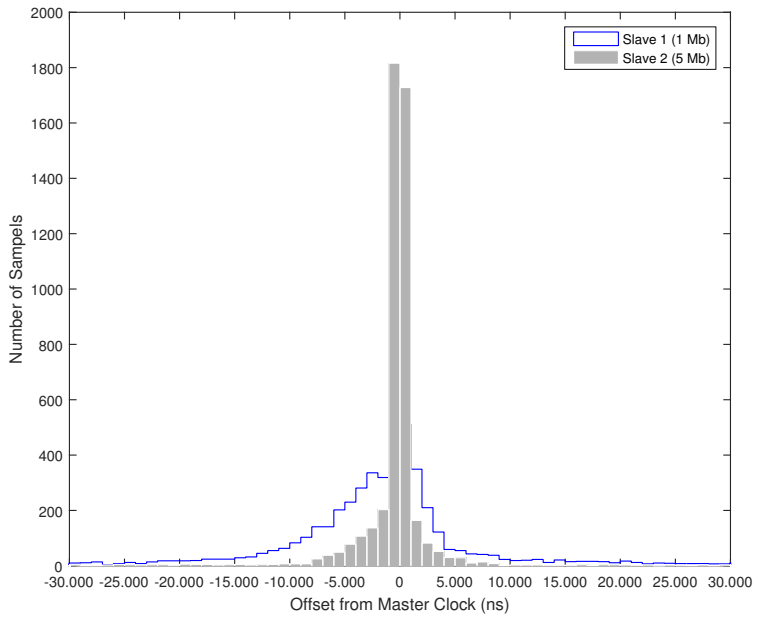

Figure 5.15: Precision of Hardware Assisted Time Synchronization under Network Load (Low)

The figure 5.14 shows the offset calculations of slaves under low network load. During

synchronization period of slave 1, there was no extra network traffic and the accuracy was around ± 200 ns. On the other hand, while applying simulated network traffic of 1Mb and 5Mb on slave 2 and slave 3 the offset was remarkably advanced up-to ± 5us and ± 15us respectively.

The precision of both slave clocks is also dissimilar, as it can be seen in figure 5.15 where slave 1 exposed to 1Mb of network traffics, most to the time offset was between the normal range as compare to slave with network load but there are some measurements where offset went up to ± 5000 ns. On the other hand, the slave with 5 Mb network traffic is having very low precision and very uncertain behavior, where the measurement samples are spread over the range of - 30000 to 30000 ns.

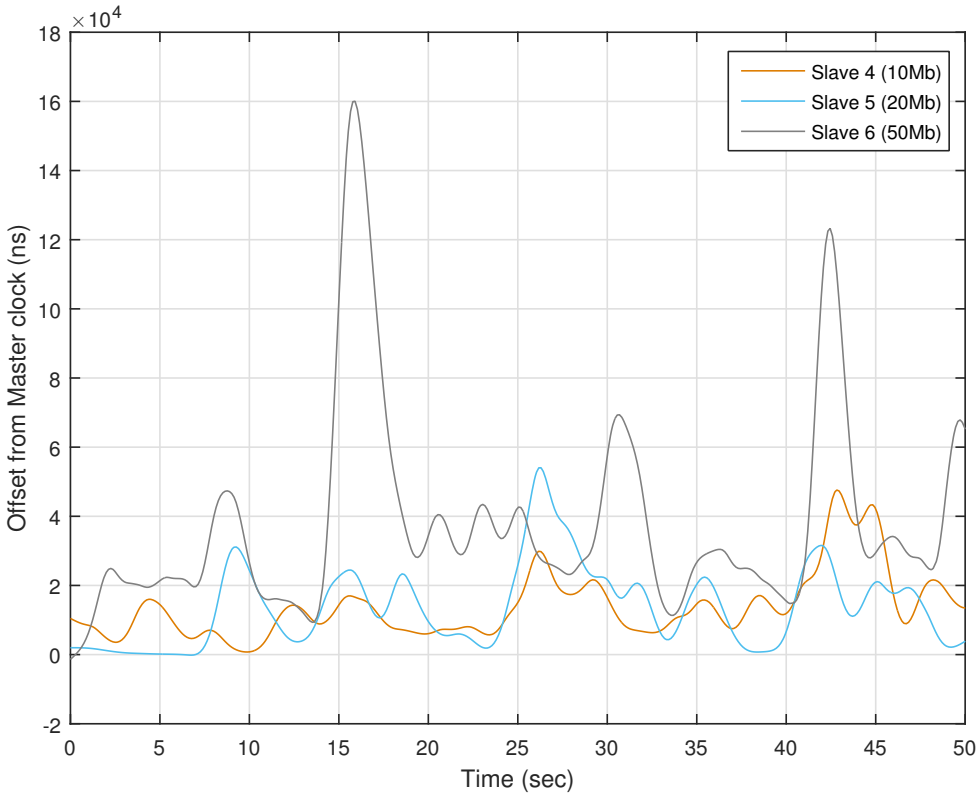

Figure 5.16: Hardware Assisted Time Synchronization under Network Load (High)

In graph 5.16, three slaves are experiencing high network traffic, where the accuracy is reached similar to software timestamping based solution. The slave 4-5 which are having 10Mb and 20Mb of network traffic load, the offsets of these clocks reached up to ± 40000ns. On the other hand slave with 50Mb of network traffic have worst accuracy.

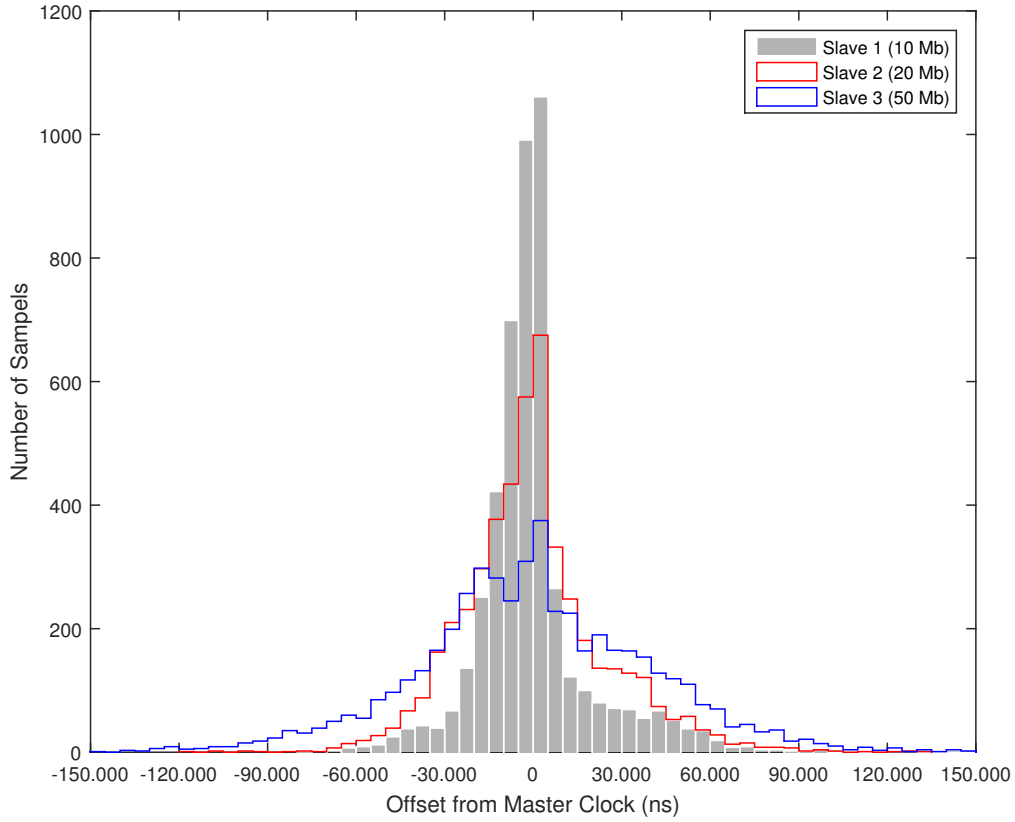

Figure 5.17: Precision of Hardware Assisted Time Synchronization under Network Load (High)

The graph above confirms that, as the bandwidth consumption's of UDP stream increases, the accuracy and precision of the clocks decreases.

Chapter 6

Conclusion

In software-based PTP synchronization network, the average offset of slave clocks remained between ± 0.5ms, but there are some timespans where the offset of clocks reached to 2ms with multi-slave synchronization model. Beside offset, the precision of slave clocks was also dissimilar with same configuration and communication medium.

In hardware timestamping based PTP implementations, the long-term results were very promising, where the accuracy of clock reached to approx 50ns. Overall in short-term measurements and random test cases, the accuracy of 200ns was frequently achieved. It is also observed that in a multi-slave synchronization scenario, the slaves showed similar behavior in term of accuracy and precision.

The tests with CPU and I/O based load do not show any prominent change in accuracy of the clock on hardware-based solution. There are also chances that the changes remained undetected due to smaller in scale. On the other hand, the test case scenarios with alien network traffic showed striking results, where the accuracy went to the lowest standard as compare other test case scenario including software timestamping based solutions.

Overall, the difference between the hardware and software based timestamping was significant. In load case scenarios, apparently network traffic based tests showed some considerable effect. Expected effects from the CPU and I/O based tests, either did not appear or we remained unable to detect them.

In order to regenerate results for further analysis or to setup a PTP hardware assisted environment, the data form all tests case scenarios with a brief description is made available, some of the tests are almost similar with little configuration change. A brief guide to setup the PTP network is also made available in appendix section.

6.1 Improvements and Future Work

The tools used to simulate particular kind of load on PTP environment does not reflect any real-world patterns of the load and it is possible with applying real-world resource exhausting application may show the different results. For example, there can be different types of network traffics that might affect the PTP setup in a totally different manner.

As discussed in earlier sections, the network component used in the PTP hierarchy

establishment are unaware of PTP network, and there can be undetected effects on PTP network. Further study can be done in this area by comparing the results with PTP supported network component. The results of simulated load based test case scenarios are only from hardware assisted PTP solution, there might be different results in software-based solution.

Appendix A

Appendix: Technical Details

Beaglebone Black

Following two figures shows the specifications of Beagleboard family device which is used as test-bed in the project.

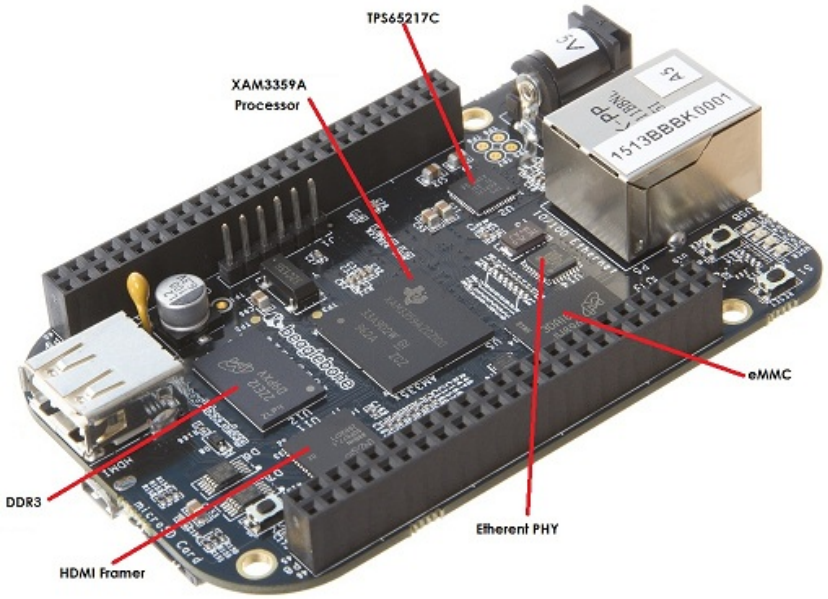

Figure A.1: Beaglebone Black Key Components

	Feature	
Processor	Sitara AM3358BZCZ100 1GHz, 2000 MIPS	
Graphics Engine	SGX530 3D, 20M Polygons/S	
SDRAM Memory	512MB DDR3L 800MHZ	
Onboard Flash	4GB, 8bit Embedded MMC	
PMIC	TPS65217C PMIC regulator and one additional LDO.	
Debug Support	Optional Onboard 20-pin CTI JTAG, Serial Header	
Power Source	miniUSB USB or DC Jack	5VDC External Via Expansion Header
PCB	3.4" x 2.1"	6 layers
Indicators	1-Power, 2-Ethernet, 4-User Controllable LEDs	
HS USB 2.0 Client Port	Access to USB0, Client mode via miniUSB	
HS 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	
Ethernet	10/100, RJ45	
SD/MMC Connector	microSD , 3.3V	
User Input	Reset Button Boot Button Power Button	
Video Out	16b HDMI, 1280x1024 (MAX) 1024x768,1280x720,1440x900 ,1920x1080@24Hz w/EDID Support	
Audio	Via HDMI Interface, Stereo	
Expansion Connectors	Power 5V, 3.3V , VDD_ADC(1.8V) 3.3V I/O on all signals McASP0, SPI1, I2C, GPIO(69 max), LCD, GPMC, MMC1, MMC2, 7 AIN(1.8V MAX), 4 Timers, 4 Serial Ports, CAN0, EHRPWM(0,2),XDMA Interrupt, Power button, Expansion Board ID (Up to 4 can be stacked)	
Weight	1.4 oz (39.68 grams)	
Power	Refer to Section 6.1.7	

Figure A.2: Beaglebone Black Features

Source of figures (A1.1 and A1.2) : https://elinux.org/Beagleboard:BeagleBoneBlack

Appendix B

Appendix: Installation Guide

In order to establish test environment, there is need for a Linux kernel image with required configuration for enabling hardware timestamping. Following are the Linux kernel options need to be enabled for the support of hardware timestamping.

```
1 CONFIG_PPS
2 CONFIG_NETWORK_PHY_TIMESTAMPING  (Only required for Timestamping in PHY  )
3 PTP_1588_CLOCK
```

There are two prominent ways to get the Linux kernel with these options enabled. First, download pre-compiled kernel, rootfs (Root file system) and bootloaders from a suitable resource and write these components on a SDcard, then boot device form memory card. Second, cross-compile a customized kernel and bootloader on a host machine, create a root file system with required modules and then finally write these components on the memory card in a standard manner.

Pre-Compiled kernel

The beagleboard community provides extensive support in development from beginner level to advanced. There are different type of Beaglebone firmware image packages available on their website (https://beagleboard.org/latest-images) with a detailed guide to port images in Beaglebone black. In most of the images, the configurations required for PTP support are already enabled.

Compiling Linux Kernel

There are different ways to compile a kernel. Instead of compiling boot-loader and kernel separately and then generating root file system in a conventional way, we recommend to use the Yocto Project. The project provides an open-source platform to build a customized kernel, bootloaders and root file system for multiple hardware architectures. The further details can be found under official documentations [20]. Following are the abstract level steps to build kernel image using Yocto Project.

Cloning Poky, Openembedded and other layers repositories.

```
1 git clone git://git.yoctoproject.org/poky.git poky-rocko
2 git clone git://git.openembedded.org/meta-openembedded
3 git clone git://github.com/jumpnow/meta-bbb
```

Initializing Build Environment using provided script

```
1   mudassar@HP-250-G4-Notebook-PC:~$ source poky-rocko/oe-init-build-env bbb/
2 You had no conf/local.conf file. This configuration file has therefore been
3 created for you with some default values. You may wish to edit it to, for
4 example, select a different MACHINE (target hardware). See conf/local.conf
5 for more information as common configuration options are commented.
6
7 You had no conf/bblayers.conf file. This configuration file has therefore been
8 created for you with some default values. To add additional metadata layers
9 into your configuration please add entries to conf/bblayers.conf.
10
11 The Yocto Project has extensive documentation about OE including a reference
12 manual which can be found at:
13     http://yoctoproject.org/documentation
14
15 For more information about OpenEmbedded see their website:
16     http://www.openembedded.org/
17
18
19 ### Shell environment set up for builds. ###
20
21 You can now run 'bitbake <target>'
22
23 Common targets are:
24     core-image-minimal
25     core-image-sato
26     meta-toolchain
27     meta-ide-support
28
29 You can also run generated qemu images with a command like 'runqemu qemux86'
30
31
```

Configuration Files

There will be two main files, containing layer and machine configuring **bblayers.conf** **local.conf**. In bblayers.conf file looks like follows, where paths of inter-depended layers are provided, these dependencies changes according to requirements. In local.conf file the architecture and build process level configurations are defined.

Initializing Build Process

The build process will take couple of hours to complete in the first time and it also depends on the specification of host system and speed of internet connection. Following console output reflects the environment during the build process.

```
 1  mudassar@HP-250-G4-Notebook-PC:~/bbb$ MACHINE=beaglebone bitbake core-image-full-
       cmdline
 2 Parsing recipes: 100% |#####################################| Time: 0:01:37
 3 Parsing of 820 .bb files complete (0 cached, 820 parsed). 1279 targets, 60 skipped,
       0 masked, 0 errors.
 4 NOTE: Resolving any missing task queue dependencies
 5
 6 Build Configuration:
 7 BB_VERSION           = "1.36.0"
 8 BUILD_SYS            = "x86_64-linux"
 9 NATIVELSBSTRING      = "universal"
10 TARGET_SYS           = "arm-poky-linux-gnueabi"
11 MACHINE              = "beaglebone"
12 DISTRO               = "poky"
13 DISTRO_VERSION       = "2.4.2"
14 TUNE_FEATURES        = "arm armv7a vfp neon callconvention-hard cortexa8"
15 TARGET_FPU           = "hard"
16 meta
17 meta-poky
18 meta-yocto-bsp       = "rocko:1648bcafa3d0e46acee61a34d0a07fabb85b1f8d"
19
20 Initialising tasks: 100% |#################################### Time: 0:00:11
21 NOTE: Executing SetScene Tasks
```

After completing of the build process, the build directory will look like following, where all the compiled and downloaded modules are saved. So, the next build will no take much time.

```
 1
 2  mudassar@HP-250-G4-Notebook-PC:~/bbb$ tree -L 1 build/
 3 build/
 4          bitbake-cookerdaemon.log
 5          cache
 6          conf
 7          downloads
 8          sstate-cache
 9          tmp
10
11
```

Build Image Directory

Following sanpshorts shows the view of Build Image directory, where couple of files are generated. The most important are

MLO: Second stage bootloader (SPL)

u-boot.img: Third stage bootloader

console-image-beaglebone.rootfs.tar.xz: Root file system which also includ the kernel (zImage)

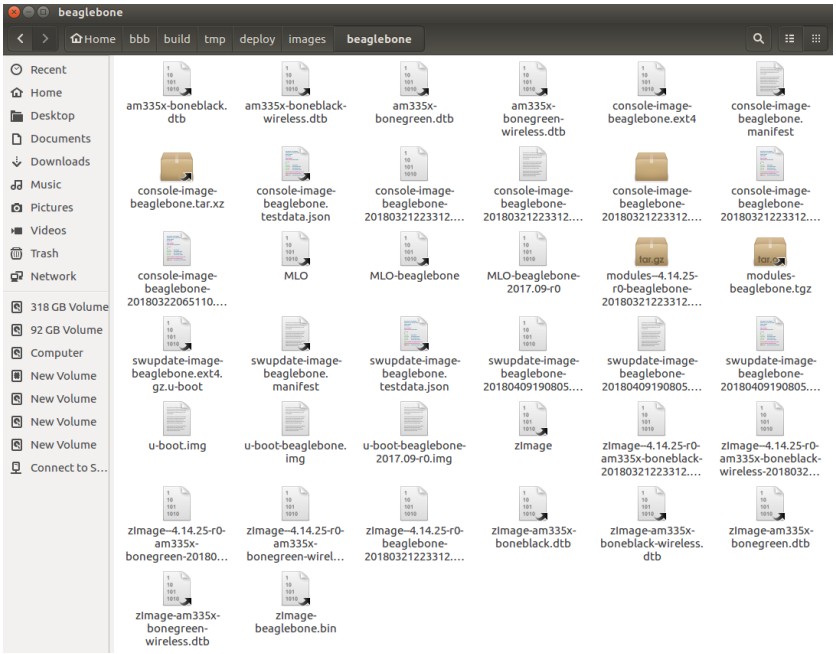

Figure B.1: Output of Yocto build

SDcard based Linux Image for Beaglebone Black

There is a standard partition pattern is defined to entertain the multistage boot process of Beaglebone. There are two partitions are required first, boot partition (FAT file system), where both bootloaders (MLO and u-boot) are placed and the second root partition for root file system and kernel.

Creating Partitions

There are many scripts available to make partitions for BBB device. By using fdisk utility one can utilize the extra space of memory card and make more partitions other than necessary. Following snapshot show the process to make required partitions.

```
1 mudassar@HP-250-G4-Notebook-PC:~$ sudo fdisk /dev/sdd
2 [sudo] password for mudassar:
3
4 Welcome to fdisk (util-linux 2.27.1).
5 Changes will remain in memory only, until you decide to write them.
6 Be careful before using the write command.
7
8
```

```
Command (m for help): o
Created a new DOS disklabel with disk identifier 0xb017558e.

Command (m for help): p
Disk /dev/sdd: 14,8 GiB, 15833497600 bytes, 30924800 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xb017558e

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-30924799, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-30924799, default 30924799): +72261K

Created a new partition 1 of type 'Linux' and of size 71 MiB.

Command (m for help): t
Selected partition 1
Partition type (type L to list all types): c
Changed type of partition 'Linux' to 'W95 FAT32 (LBA)'.

Command (m for help): a
Selected partition 1
The bootable flag on partition 1 is enabled now.

Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (2-4, default 2): 2
First sector (147456-30924799, default 147456):
Last sector, +sectors or +size{K,M,G,T,P} (147456-30924799, default 30924799):

Created a new partition 2 of type 'Linux' and of size 14,7 GiB.

Command (m for help): p
Disk /dev/sdd: 14,8 GiB, 15833497600 bytes, 30924800 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xb017558e

Device     Boot  Start      End  Sectors  Size Id Type
/dev/sdd1  *      2048   147455   145408   71M  c W95 FAT32 (LBA)
/dev/sdd2       147456 30924799 30777344 14,7G 83 Linux
```

Formatting Partitions

```
1 mkfs.vfat  -F 16 -n "boot" /dev/sdd1
2 mke2fs -j -L "root" /dev/sdd2
```

Mounting Partitions

```
1 mkdir /mnt/sdd1
2 mkdir /mnt/sdd2
3 mount /dev/sdd1 /mnt/sdd1
4 mount /dev/sdd2 /mnt/sd2
```

Copying Bootloaders in First Boot Partition

```
1 cp MLO-beaglebone /mnt/sdd1/MLO
2 cp u-boot-beaglebone.img /mnt/sdd1/u-boot.img
```

Extracting Root File System in Second Root Partition

```
1 tar x -C /mnt/sdd2 -f console-image-beaglebone.tar.gz
```

Unmounting Partitions

```
1 umount /mnt/sdd1
2 umount /mnt/sdd2
```

Appendix C

Appendix: Additional Tests and Measurements

There were multiple test conducted during this study but not all of them are discussed and presented in the report.However, the data from all tests are made available after preparing in form of MATLAB workspace file. One can regenerate the graphs and also further investigate other tests which are not the part of this report. The table C1, C2 and C3 shows test and the abstract level configuration during test.

Test Name	ID	Demon Config			Timestamping mode	Remarks
		Message Frequency	Filter 1	Filter 2		
3*Pure Software	pure_Sw_offset_100	1			Sofware	
	pure_Sw_offset_101	1			Software	
	pure_Sw_offset_106	1			Software	
3*Pure Harware	pure_Hw_offset_100	1			HW	
	pure_Hw_offset_101	1			HW	
	pure_Hw_offset_106	1			HW	

Table C.1: Multi-Slave Test

Test ID	PTP Demon Config			Stress Test Config		Load distribution	Remarks
	Message Frequency	Filter 1	Filter 2	Load type	Intensity		
HwT1.1	1						Slave: HW , Master: SW
HwT1.2	1						Slave: SW , Master: HW
HwT1.3	1						Slave: HW , Master: HW
HwT1.4	8						
HwT1.5	8						log data: phc2sys + ptp4l
HwT1.6	8						log data: phc2sys
HwT1.7	8						log data: phc2sys + ptp4l
HwT1.8	8			CPU	50	Slave Only	
HwT1.9	8			CPU	50	Both	
HwT1.10	8			CPU	100	Slave Only	
HwT1.11	8			CPU	100	Both	
HwT1.12	8			I/O	–	Slave Only	
HwT1.13	8			I/O	–	Both	
HwT1.14	8			CPU	100	Slave Only	log data: phc2sys + ptp4l
HwT1.15	8			CPU	100	Both	log data: phc2sys + ptp4l
HwT1.16	8			I/O	–	Slave Only	log data: phc2sys + ptp4l
HwT1.17	8			I/O	–	Both	log data: phc2sys + ptp4l
HwT1.18	8			Network	1Mb	Source: Alien	
HwT1.19	8			Network	5Mb	Source: Alien	
HwT1.20	8			Network	10Mb	Source: Alien	
HwT1.21	8			Network	20Mb	Source: Alien	
HwT1.22	8			Network	50Mb	Source: Alien	

Table C.2: Harware Timestamping Test

Test ID	PTP Demon Config			Stress Test Config		Load distribution	Remarks
	Message Frequency	Filter 1	Filter 2	Load type	Intensity		
SwT1.1	1	raw					
SwT1.2	1	raw	moving$_{m}edium$				
SwT1.3	1	filter					
SwT1.4	8	filter					
SwT1.5	8	raw					
SwT1.6	8	filter$_{w}eight$					
SwT1.7	8	filter$_{w}eight$					
SwT1.8	8	filter$_{w}eight$	moving$_{a}verage$				
SwT1.9	8	filter		CPU	50		
SwT1.10	8	filter		CPU	100		
SwT1.11	8	filter		I/O	Auto		
SwT1.12	8	filter		I/O	Manual-aio-4094		Restlts:fail
SwT1.13	8	filter		I/O	Manual-aio-10		
SwT1.14	8	filter					Restart:stable:pure
SwT1.15	8	filter		I/O	Manual aio>20>5>10		1 break;pathdely bug
SwT1.16	8	filter		I/O	Auto 10		
SwT1.17	8	filter		CPU	50	Slave Only	
SwT1.18	8	filter		CPU	100	Slave Only	Can not able to set frequency
SwT1.19	8	filter		CPU	Manual-cpu	Slave Only	Can not able to set frequesncy
SwT1.20	8	filter		CPU	Manual-cpu-1	Slave Only	
SwT1.21	8	filter		CPU	Manual-cpu-1	Slave Only	
SwT1.22	8	filter		I/O	I/O-8	Slave Only	Error:pcsleep

Table C.3: Software Timestamping Test

References

Literature

[1] D. W. Allan et al. "Precision oscillators: Dependence of frequency on temperature, humidity and pressure". *IEEE Frequency Control Symposium* (1992) (cit. on p. 3).

[2] *AN-1838 IEEE 1588 Boundary Clock and Transparent Clock Implementation Using the DP83640*. Tech. rep. SNLA104A. Edwards, CA: Texas Instruments Incorporated, Apr. 2013. URL: http://www.ti.com/lit/an/snla104a/snla104a.pdf (cit. on pp. 6, 7).

[3] Markus Seehofer. Andreas Dreher Dirk Mohl. *Precision Clock Synchronization – IEEE 1588 (White Paper Rev 1.2)*. Version 1.2. URL: http://www.hirschmann.com (cit. on pp. 5, 9).

[4] Markus Seehofer. Andreas Dreher Dirk Mohl. *Precision Clock Synchronization – IEEE 1588 (White Paper Rev 2.0)*. Version 2.0. URL: http://www.hirschmann.com (cit. on p. 3).

[5] Suchit Lapcha Changrong Li and Mingkai Hu. *IEEE 1588 for Telecom and Networking Applications*. Tech. rep. FTF-NET-F0102. NXP Freescale Technology Forum(FTF), June 2012. URL: https://www.nxp.com/files-static/training_pdf/FTF/2012/americas/WBNR_FTF12_NET_F0102.pdf (cit. on p. 4).

[6] *Cisco Nexus 3000 Series NX-OS System Management Configuration Guide, Release 5.0(3)U3(1)*. Tech. rep. OL-26558-01. Cisco Systems Networking hardware company. URL: https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus3000/sw/system_mgmt/503_U3_1/b_Cisco_n3k_System_Mgmt_Config_503_U3_1/b_Cisco_n3k_System_Mgmt_Config_503_U3_1_chapter_0101.pdf (cit. on pp. 6, 7).

[7] Richard Cochran and Cristian Marinescu. "Design and Implementation of a PTP Clock Infrastructure for the Linux Kernel". *International IEEE Symposium on, Sep. 27–Oct* 1 (2010), pp. 116–121 (cit. on pp. 11, 12, 14, 15).

[8] John Eidson. *Tutorial on IEEE 1588*. Tech. rep. Agilent Technologies, Inc, Oct. 2005. URL: https://www.nist.gov/sites/default/files/documents/el/isd/ieee/tutorial-basic.pdf (cit. on p. 4).

[9] Geoffrey M. Garner. *IEEE 1588 Version 2*. Tech. rep. 2008 International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication, Sept. 2008. URL: http://passthrough.fw-notify.net/download/570185/http://www.ieee802.org/1/files/public/docs2008/as-garner-1588v2-summary-0908.pdf (cit. on pp. 6, 7, 9).

[10] *IEEE Std 1588 -2002*. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. URL: https://standards.ieee.org/findstds/standard/1588-2002.html (cit. on pp. 4, 6, 10).

[11] *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. July 24, 2008. URL: https://standards.ieee.org/findstds/standard/1588-2008.html (cit. on pp. 4, 7, 9, 10).

[12] Mike Gilson Jean-Loup Ferrant. *Synchronous Ethernet and IEEE 1588 in Telecoms. Next Generation Synchronization Networks*. 1st ed. London: ISTE Ltd, 2013 (cit. on p. 3).

[13] Nick Barendt Kendall Correll and Michael Branicky. "Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol". *IEEE 1588 Conference, Zurich, October 2005* 1 (2005) (cit. on p. 11).

[14] Kevin B. Stanton Patrick Ohly David N. Lombard. "Hardware Assisted Precision Time Protocol.Design and case study." *in Proceedings of LCI International Conference on High-Performance Clustered Computing.2008* 1 (2008) (cit. on pp. 8, 11, 14, 16).

[15] Cristian Marinescu Richard Cochran and Christian Riesch. "Synchronizing the Linux System Time to a PTP Hardware Clock." *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on 12-16 Sept. 2011* 1 (2011) (cit. on pp. 11, 16, 17).

[16] Steve T. Watt. "Understanding and Applying Precision Time Protocol". *Power and Energy Automation Conference, March 2014* (2014), pp. 2–3 (cit. on p. 10).

Online sources

[17] *IEE 1588*. URL: https://www.nist.gov/el/intelligent-systems-division-73500/introduction-ieee-1588 (cit. on p. 4).

[18] *Linux Kernel Documentation: Timestamping Control Interfaces*. URL: https://www.kernel.org/doc/Documentation/networking/timestamping.txt (cit. on pp. 13, 14).

[19] *Time-Stamping and Time Synchronization*. URL: https://docs.napatech.com/reader/897VV1LF3lshs6Q23PkgzQ/5TKHcCpFK5gUCnKBkD7shQ (cit. on p. 9).

[20] *Yocto Project Documentaion*. URL: https://https://www.yoctoproject.org/docs/ (cit. on p. 38).