Design & Implementation of Timing and synchronization for an automotive amplifier

BITS ZG629T: Dissertation

By,

Mohan Karthik

2011HZ12013

Dissertation work carried out at

Analog Devices India Pvt. Ltd., Bangalore

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN)

APRIL 2013

Design & Implementation of Timing and synchronization for an automotive amplifier

BITS ZG629T: Dissertation

By,

Mohan Karthik

2011HZ12013

Dissertation work carried out at

Analog Devices India Pvt. Ltd., Bangalore

Submitted in partial fulfillment of M.S. Software Systems degree program

Under the Supervision of
Vijaykumar N, Project Manager,
Analog Devices India Pvt. Ltd., Bangalore

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI (RAJASTHAN)
APRIL, 2013

CERTIFICATE

This is to certify that the Dissertation entitled Design & Implementation of Timing and synchronization for an automotive amplifier and submitted by Mohan Karthik having ID-No. 2011HZ12013 for the partial fulfilment of the requirements of M.S. Software Systems degree of BITS embodies the bona fide work done by him/her under my supervision.

N. Vijayakumar

Vijaykumar N, Project Manager, Analog
Devices India Pvt. Ltd., Bangalore

Place : Bangalore

Date : 29th March 2013

Abstract

Present day automobiles contain a very high quality of multimedia communications. Traditionally the Head-Unit is the source of the Audio and Video inside an automobile. An amplifier is responsible of receiving the audio from the Head-Unit, processing the audio and playing it out on the speakers. OEMs (Original Equipment Manufacturers) currently prefer the use of MOST (Media Oriented System Transport) to allow the multimedia content to stream from the Head-Unit to the amplifier. Ethernet AVB (Audio Video Bridging) is an open alternative to MOST that allows for a cheaper and a globally standardized networking option. EAVB consists of various standards that modify IEEE 802.1 such as: Timing and Synchronization (IEEE 802.1AS), Stream Reservation Protocol (IEEE 802.1Qat) and Forwarding and Queuing (IEEE 802.1Qav). In this work, the open source implementation of the IEEE 1588 (which is the parent standard of IEEE 802.1AS) is ported and implemented on to an ADI processor and demonstrate clock synchronization between two nodes connected via a 100Base-Tx Ethernet Link.

Acknowledgements

This dissertation would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, my utmost gratitude to Mr S Sivaramakrishnan and Mr Vijaykumar N, my supervisors and guides at Analog Devices India Pvt. Ltd. for this project. They were instrumental in bringing this project to life and in guiding me to solve all the problems and in completing this project.

To my colleagues Praveen Krishna, Krishna Gopalakrishnan, and others who accommodated and contributed to this project in terms of discussions and helped me to gain clarity in its execution.

To the wonderful faculty at BITs, Pilani, who have, through their content and online lectures paved the foundation for this work.

To Steven Kreuzer and George Neville-Neil for their implementation of the open source PTP engine which was used for this project demonstration.

Last but not the least, my family and friends who were always there to provide me the strength to plod on despite my constitution wanting to give up and throw the towel.

Abbreviations and Acronyms

ADI	Analog Devices Inc.
BF	BlackFin
BMCA	Best Master Clock Algorithm
CAN	Controller Area Network
CSMA/CD	Carrier Sensing Multiple Access / Collision Detection
CSN	Coordinated Shared Networks
DMA	Direct Memory Access
DSP	Digital Signal Processor
EAVB	Ethernet Audio Video Bridging
FIFO	First In First Out
FQTSS	Forwarding and queuing for Time Sensitive Streams
IP	Internet Protocol
LAN	Local Area Network
LIN	Local Interconnect Network
LVDS	Low Voltage Differential Signalling
lwIP	light weight Internet Protocol
MAC	Media Access Control
MII	Media Independent Interface
MOST	Media Oriented System Transport
OEM	Original Equipment Manufacturer
OSI	Open Systems Interconnection
PPS	Pulse per Second
PTP	Precise TimesTamping
QOS	Quality Of Service
RISC	Reduced Instruction Set Computing
RSE	Rear Seat Entertainment
SIMD	Single Instruction Multiple Data
SRP	Stream Reservation Protocol
SVN	Subversion
TAI	International Atomic Time
TCP	Transport Control Protocol
UDP	User Datagram Protocol
UTC	Co-ordinated Universal Time
VLAN	Virtual Local Area Network

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Overview of media streaming

1.1.1 Historical background

Over the past ten years, consumer demand had driven a hefty increase in audio and video features in the automobile. Once only found in luxury cars, features such as DVD playback, backup cameras, and navigation have become familiar options in many typical automobiles. Rear Seat Entertainment (RSE) units are growing in erudition with more sources and selections at your fingertips. Each of these options has added to the necessity and yearning for a common networking architecture in the automobile.

While automotive OEMs around the globe have encompassed the concept of low-bandwidth vehicle communication networking; with Controller Area Network (CAN) being espoused almost universally, the distinctive and diverse challenges in vehicle multimedia networking (e.g., cost, economies of scale, bandwidth, QOS, scalability, open vs. proprietary, and supplier choice) has left the door open for much deliberation over the best solution for multimedia from both a technical and a commercial perspective.

Traditionally, the implementation of packet-switched network has been circumvented for vehicle multimedia applications due to its nondeterministic nature. The recent work of the IEEE Audio Video Bridging (AVB) task group offers draft standards-based approach for highly consistent networked transmission for low latency applications like those found within an automobile. While these AVB protocols can be used on more than one physical layer type, this project will focus on the application of AVB over Ethernet. Wired Ethernet networks employing AVB protocols are very well suited to automotive deployment due to both simplified cabling and reliability of a hard-wired solution.

1.1.2 Requirements for A/V streaming

So, what is wrong with typical packet-switched networks? Before that can be answered, the requirements for A/V streaming must be explored:

It must be conceivable to synchronize multiple streams so that they can be rendered in time with regards to each other. At its humblest case, this might be ensuring "lip synch" so that the audio and video aspects of a movie or television show are not out of synchronization. A much more rigorous requirement comes from the need to keep multiple digital speakers in phase, this means keeping streams synchronized within approximately one microsecond.

The worst case delay for a stream in the network, including buffering delays at the source and destination, must be short and deterministic. For almost all automotive applications, this means that the network must not considerably contribute to the user-interface delay: the time from when the consumer requests an action (e.g., presses a button on a controller) until that action is readily apparent by the consumer (e.g., "play", "pause", "forward", "reverse", etc.). This is something on the order of 2ms.

Lastly, applications must be able to get a high level of assurance that the network resources required are available and will continue to be available as long as the application needs it. This is sometimes called as "reservation", and sometimes as "admission control". The intent is for an application to let know the network of the requirements for a stream ahead of time, and have the network lock down the resources required for that stream and, if they are not available, to notify the application. Typical resources needed by A/V streams are throughput and specific constraints on delay.

1.1.3 The problems with existing approaches

Virtually all existing network equipment is grounded on IT requirements: move the data through the network as fast as possible with smallest cost and marginal management. This is an admirable approach in a world with no hard bounds on delay or synchronization requirements. IT-oriented networks do not always, however, meet the requirements of the previous section:

There is no notion of "time" in an IT network – There is nothing in the network infrastructure itself that can provide assistance in synchronization or offer any kind of precision timing mechanism.

Delays can be too high – While delay through a network may, on the average, be very low, there is little exertion made to bind that delay. In an IT network, transporting data reliably is regarded as much more imperative than for the data to be delivered within a definite time.

The network itself does not avoid network congestion, so data can be lost if buffers are insufficient or link bandwidth is deficient for offered traffic – IT networks count on higher level protocols to manage congestion (e.g., TCP) by regulating transmission and retransmitting dropped packets. This is tolerable when long delays are acceptable, but will not work where low deterministic delays are the requirement.

The representative way these last two problems are handled today is with buffering, but excessive buffering can cause delays that are absolutely intolerable in an automotive environment.

1.2 The Audio and Video Bridging Standard

An "Audio Video Bridging" network is one that implements a set of protocols being developed by the IEEE 802.1 Audio/Video Bridging Task Group. There are four primary updates between the proposed Audio Video Bridging architecture and existing 802 architectures (from now on the term "AVB" will be used instead of "Audio Video Bridging"):

1. Precise synchronization,

2. Traffic shaping for media streams,

3. Admission controls (or stream reservation), and

4. Identification of non-participating devices.

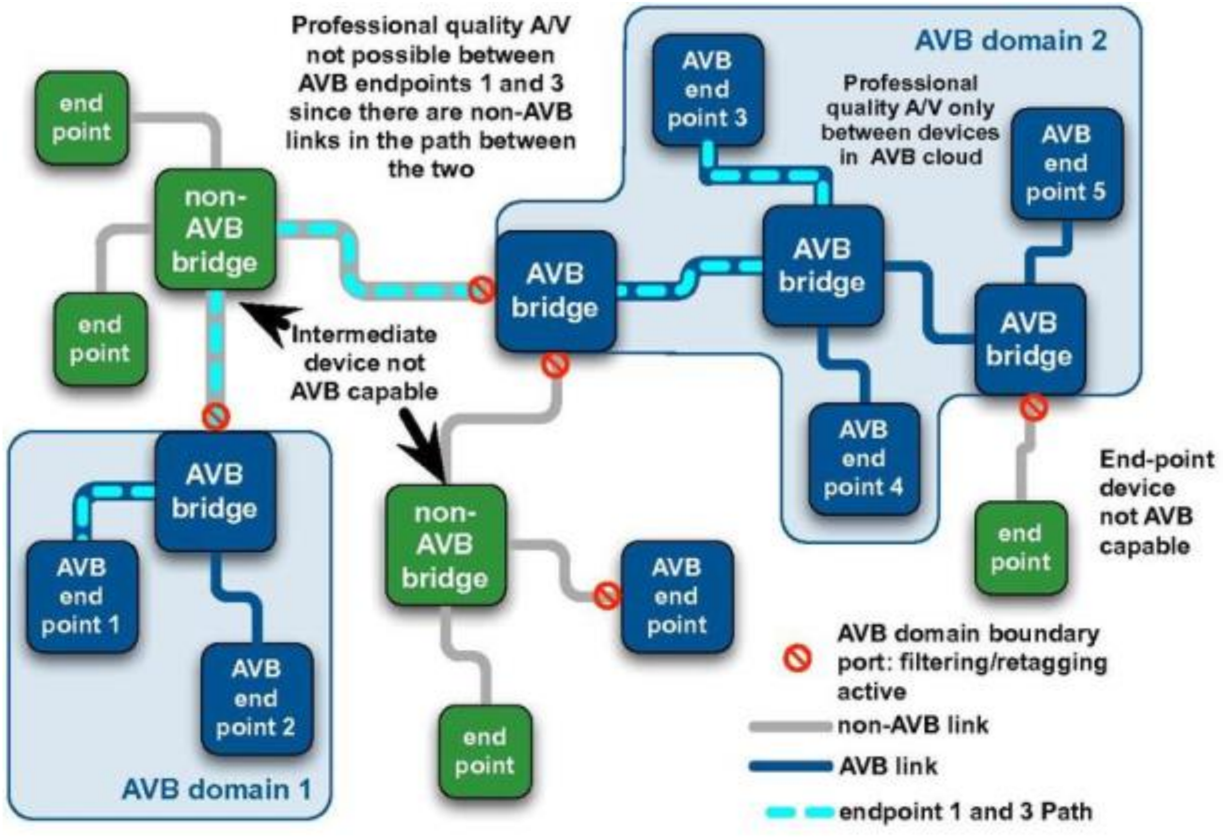

Professional quality A/V not possible between AVB endpoints 1 and 3 since there are non-AVB links in the path between the two
AVB domain 2
end point
non-AVB bridge
end point
Intermediate device not AVB capable
AVB end point 3
Professional quality A/V only between devices in AVB cloud
AVB end point 5
AVB bridge
AVB bridge
AVB bridge
AVB end point 4
End-point device not AVB capable
end point
AVB bridge
AVB end point 1
AVB end point 2
AVB domain 1
non-AVB bridge
AVB end point
end point
AVB domain boundary port: filtering/retagging active
non-AVB link
AVB link
endpoint 1 and 3 Path

Figure 1: AVB connections [1]

These updates are implemented using rather small additions to the standard layer-2 MACs[1] and bridges protocol (IEEE Std.802.1). This “minimal change” attitude allows for easy communication between non-AVB and AVB devices using standard 802 frames. Nevertheless, as shown in Figure 1, only AVB devices are capable of:

i) Reserving a portion of network resources for streaming media

ii) Sending and receiving the new time-sensitive frames.

1.2.1 Precise Time Synchronization

AVB nodes periodically synchronize their reference clocks very precisely by exchanging timing information across the links that connect them. This nano-precision synchronization has the following rationale:

[1] A “layer-2 MAC” is the part of the network interface that controls access to the network and media (e.g., wire or frequency band). “MAC” is an acronym for “Media Access Controller”.

1. To allow a common time base across the various connected nodes. This common time base now allows these nodes to communicate among themselves with respect to a time that every node in the cluster understands.
2. To allow the transmission and reception of multiple synchronized media streams. More explanation on this would follow.

The protocol used for maintaining this precise time synchronization is specified in IEEE Std.802.1AS [4] which is a very rigidly constrained subset of another IEEE standard (IEEE Std.1588 [2]). IEEE Std.1588 was drafted for industrial and control applications.

The IEEE Std.802.1AS protocol forms a domain among all the participating devices. And this domain dissects the devices into devices that are capable of implementing and adhering to the standard and those that do not. Refer to Figure 1 for a visual representation of the domains are formed.

Each of these timing domains would have a single device named among the participating nodes as the "Grand Master Clock". The Grand master clock typically has the "best" internal clock. The definition of best in this context would be some sort of traceability to the international atomic clock (TAI) or to the coordinated universal time (UTC). All the other devices would name themselves as the "Slave" and would attempt to synchronize (periodically adjust their internal clocks) and syntonize (continuously adjust its clock ratio) with the master clock.

1.2.1.1 Systems

A PTP system is a made up of a combination of PTP and non-PTP devices in a distributed, networked environment. PTP devices can be ordinary clocks, boundary clocks, end-to-end transparent clocks, peer-to-peer transparent clocks, and management nodes. Non-PTP devices include any device that has not implemented the IEEE Std. 1588 protocol.

1.2.1.2 Architecture

There are five basic types of PTP devices, as follows:

a) Ordinary clock

b) Boundary clock

c) End-to-end transparent clock

d) Peer-to-peer transparent clock

e) Management node

All five types implement one or more aspects of the protocol. In this project only the protocol for ordinary clocks are regarded and implemented.

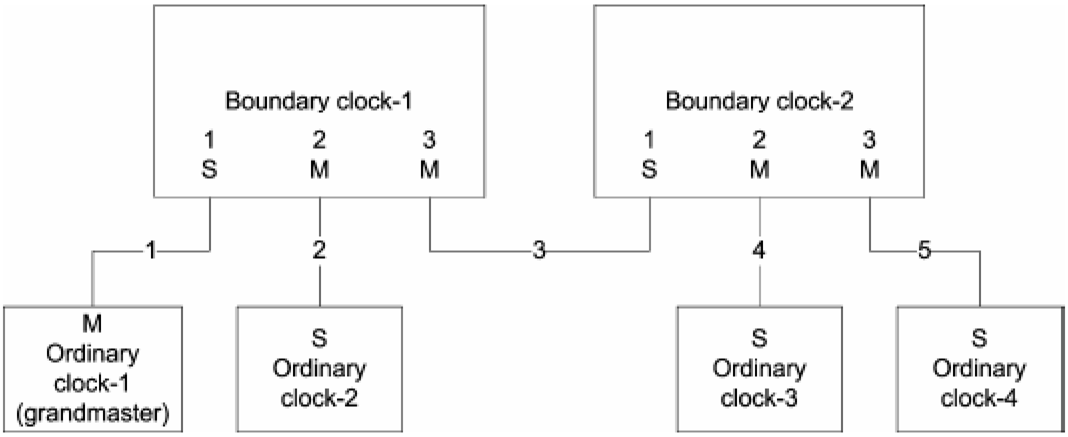


Figure 2: Simple master–slave clock hierarchy [2]

1.2.1.3 Message Classes

The protocol defines event and general PTP messages. Event messages are timed messages in that an accurate timestamp is generated at both transmission and receipt of these messages. General messages do not require accurate timestamps.

The set of event messages consists of:

a) Sync

b) Delay_Req

c) Pdelay_Req

d) Pdelay_Resp

The set of general messages consists of:

a) Announce

b) Follow_Up

c) Delay_Resp

d) Pdelay_Resp_Follow_Up

e) Management

f) Signalling

The messages are explained in detail in the IEEE Std.1588:2008 standard [2].

1.2.1.4 Clock Synchronization Model

There are two phases in the normal execution of the protocol:

1.2.1.4.1 Establishing the master-slave hierarchy

Within a domain, each port executes an independent copy of the protocol state machine. For "state decision events," each port examines the contents of the "Announce" messages received on the port. Using the best master clock algorithm, the Announce message contents and the contents of the data sets associated with the ordinary or boundary clock are analysed to determine the state of each port of the clock. The various states are defined in Table 1.

1.2.1.4.1.1 Best master clock algorithm

The best master clock algorithm compares the data describing two clocks to determine which data describes the better clock. This algorithm is used to determine which of the clocks in the PTP domain the best clock is. It is a distributed protocol, at the end of which, a single grand master clock is chosen by all the participating nodes, and a spanning tree is formed from the grand master clock to each of the participants. Note that this spanning tree is very different to the one formed by the IEEE 802.1Q.

1.2.1.4.2 Synchronizing the clocks

In a PTP system, ordinary or boundary clocks synchronize by exchanging PTP timing messages on the communication path linking the two clocks. The basic pattern of synchronization message exchange is illustrated in Figure 3.

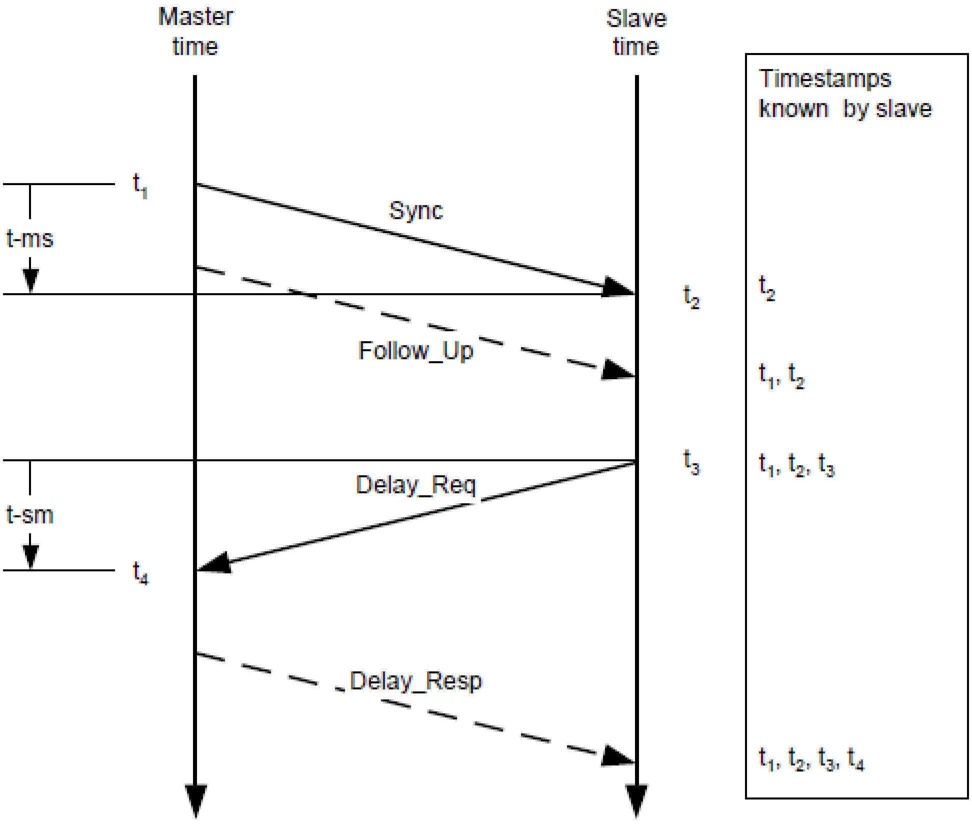


Figure 3: Basic synchronization message exchange [2]

The message exchange pattern is as follows:

a) The master initiates a Sync message to the slave and notes the time t1 at which it was sent.

b) The slave receives the Sync message and notes the time of reception t2.

c) The master conveys to the slave the timestamp t1 by:

1) Embedding the timestamp t1 in the Sync message. This requires some sort of hardware processing for highest accuracy and precision. (This is implemented in this project's target hardware)

2) Embedding the timestamp t1 in a Follow_Up message. (This is followed by the test node PC)

d) The slave sends a Delay_Req message to the master and notes the time t3 at which it was sent.

e) The master receives the Delay_Req message and notes the time of reception t4.

f) The master conveys to the slave the timestamp t4 by embedding it in a Delay_Resp message.

At the conclusion of this exchange of messages, the slave possesses all four timestamps. These timestamps will then be used by the slave to compute the offset of the slave's clock with respect to the master. It will also be used to compute the mean propagation delay between the two clocks.

1.2.1.5 Generation of message timestamps

A timestamp is generated when any PTP event message is sent or received. The timestamp has to be generated at a precise point of the message transmission / reception. The generation of the timestamps, indicated in Figure 3, is modelled in Figure 4.

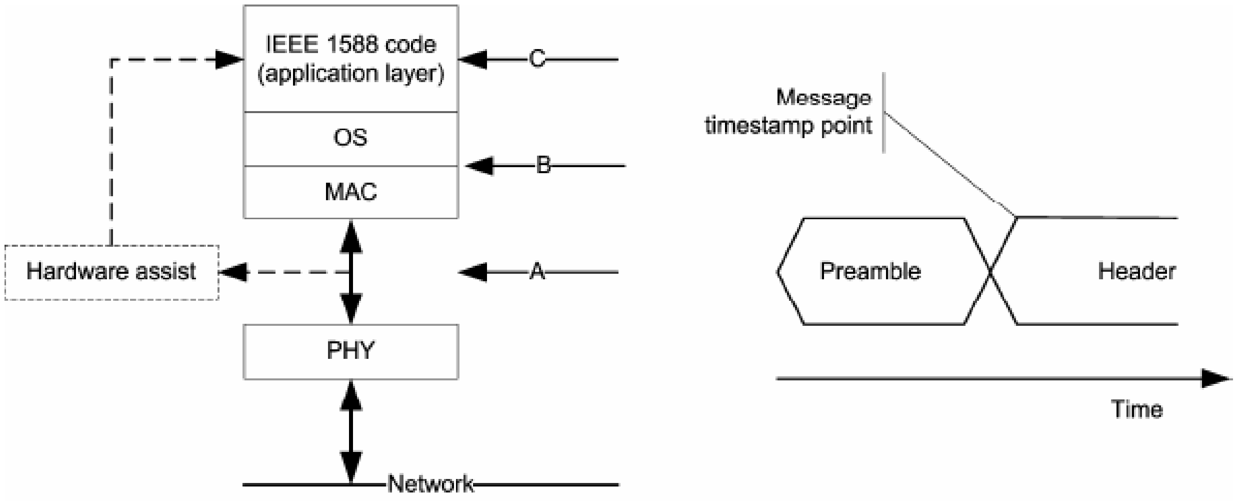


Figure 4: Timestamp generation model [2]

1.2.2 Traffic shaping for AV streams

As discussed earlier, in order to ensure professional AV services, the AVB architecture implements traffic shaping on top of the existing 802.1Q forwarding and priority mechanisms. It also identifies a particular relationship between priority tags and frame forwarding behaviour at endpoints and bridges.

Traffic shaping is the process of evenly distributing the packets making up a stream over time thus smoothing out the traffic. If traffic shaping is not done at sources and bridges, then the end points tend to produce bursts of traffic. These bursts cause the packets to bunch and clump at buffers at the subsequent bridges and listeners. This bunching causes jitter in the packet transmission, and more importantly can cause packet loss due to insufficient buffer size at the intermediate devices or even the listener end points.

Talker endpoint nodes and AVB bridges are required to very evenly transmit the media frames for a particular stream based on the AVB traffic class and the specific QoS parameters that were used when the stream was reserved / admission controlled by the network (see "Admission controls" below). The specific rules for traffic shaping are described in the IEEE 802.1Qav [5] specification, and are a spinoff of the "leaky bucket" credit-based shaping [6] where the bandwidth reserved for a stream controls the time between the packets that make up the stream.

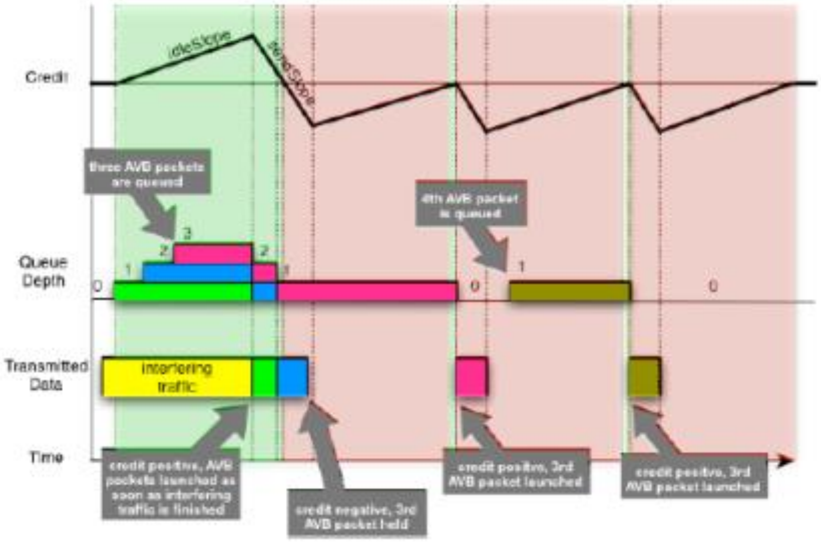

idleSlope
sendSlope
Credit
three AVB packets are queued
4th AVB packet is queued
3
2
2
1
1
1
0
0
0
Queue Depth
0
Transmitted Data
interfering traffic
Time
credit positive, AVB packets launched as soon as interfering traffic is finished
credit negative, 3rd AVB packet held
credit positive, 3rd AVB packet launched
credit positive, 3rd AVB packet launched

Figure 5: Example Qav traffic shaping [1]

1.2.3 Stream Reservation or Admission controls

The preceding mechanism ensures that the time sensitive data is delivered with a deterministically low latency and jitter. But to continue to do so over time and to fulfil its duties completely, all the participating nodes require the resource requirements of each of the streams that pass through them.

The stream reservation protocol is described in IEEE Std.802.1Qat [3]. This protocol allows end points to identify themselves as “Talkers” (end points that source media streams into the network) and as “Listeners” (end points that sink the media streams from the network). It then allows these end points to register and advertise information about the streams that they source and sink into the network. The bridges on the path of the stream then ensure that the resources (bandwidth, latency) are guaranteed during the course of the media transmission. The bridges continue to guarantee the resources until the end points advertise their intent to stop participating in the corresponding stream.

All the streams on the AVB domain have a unique stream ID. This stream ID is then kept as a reference among all the AVB domain participants for updating and managing their individual stream forwarding, reserving and queuing tables.

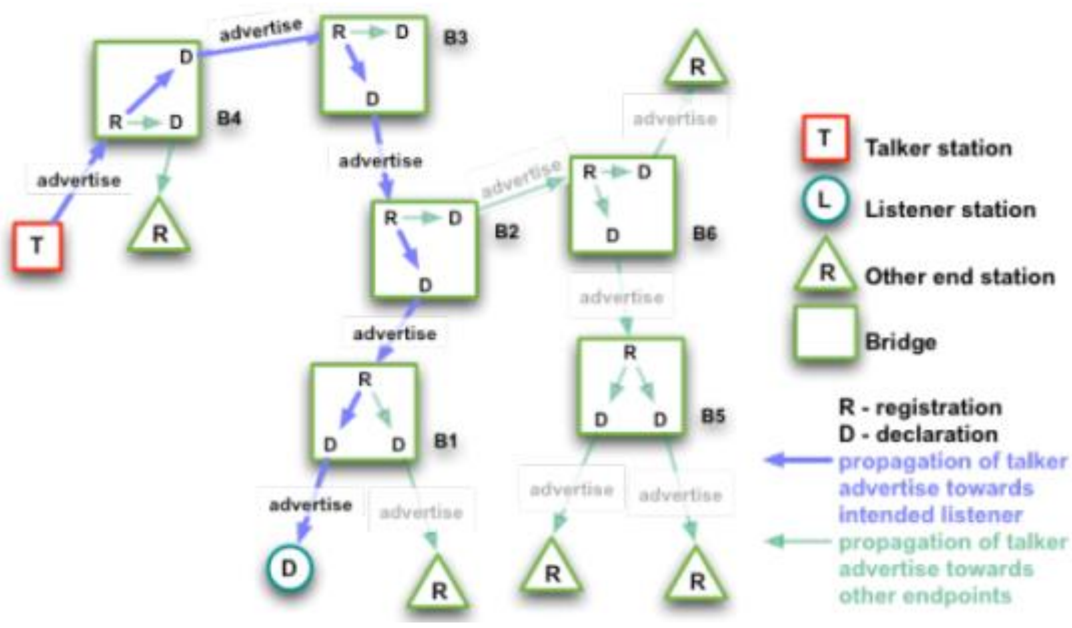


Figure 6: Successful reservation (talker advertise) [1]

1.2.4 Benefits for Automotive OEMs

The benefits of utilizing Ethernet AVB for automotive OEMs are many. This includes simpler cabling, an open and standardized technology with multiple suppliers and the precise guarantees made by the implementation with regards to latency, jitter and timing. Michael Johas Teener et al, captures the benefits of the automotive OEMs in their white paper titled "No-excuses Audio/Video Networking: the Technology behind AVnu" [1].

1.3 Analog Devices® ADSP-BF518F Blackfin™ Processor

The ADSP-BF518 (refer Figure 7) processor is a member of the Blackfin family of products, built on the Analog Devices/Intel Micro Signal Architecture (MSA). Blackfin processors combine a dual-MAC state-of-the-art signal processing engine, the advantages of a clean, orthogonal RISC-like microprocessor instruction set, and single- instruction, multiple-data (SIMD) multimedia capabilities into a single instruction-set architecture.

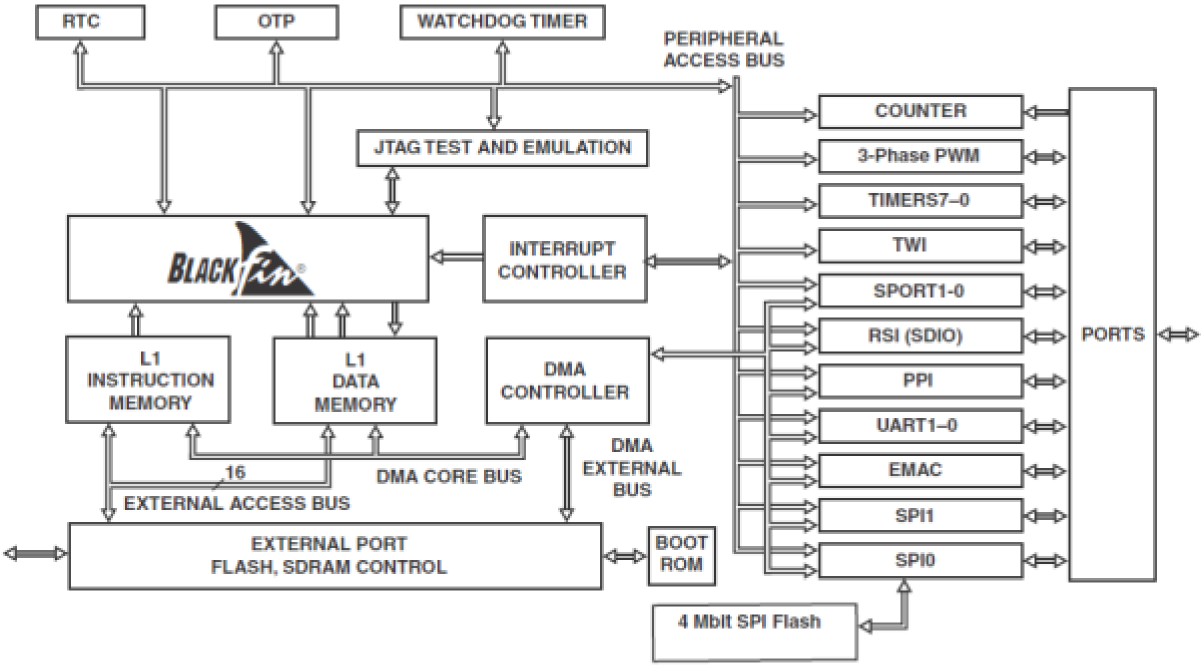


Figure 7: ADSP-BF518F Block Diagram [7]

The reason the ADSP-BF518 was chosen for this project is the availability of the hardware IEEE Std. 1588 engine which allows us to reliably generate timestamps for the incoming and outgoing messages. Chapter 22 and Chapter 23 of the ADSP-BF518 Hardware Reference describe the EMAC and the PTP engine features, capabilities and the registers [7].

1.4 Precision Time Protocol daemon

The open source Precision Time Protocol daemon is used as the base starting point in this project [9]. The open source project implements the IEEE Std.1588:2008 purely in software. This implementation has been ported onto the ADSP-BF518 platform with support for hardware time stamping.

2 Problem Statement

The following is the problem statement for this project:

Design and Implement a precise time synchronization protocol as a foundation for an automotive audio amplifier.

- Design a software architecture for implementation of time synchronization on Analog Devices® ADSP-BF518 Blackfin processor.
- Utilize the open source ptpd implementation as the base source code.
- Modify the open source implementation to utilize Blackfin's hardware time stamping features for precise accuracy.
- Demonstrate the precise time synchronization features of the ADSP-BF518 with respect to another node (PC) executing the open source implementation to prove compatibility.

The following section expands on this problem statement with the detailed requirements.

2.1 Detailed Requirements

This section discusses the requirements for this software and enumerates the same.

2.1.1 Time Sync protocol requirements

- The software shall utilize the open source ptpd as the base implementation for IEEE Std.1588:2008 protocol.
- Best Master Clock algorithm shall be implemented to facilitate dynamic master clock selection.
- The software shall implement methods to synchronize the clock of the slave and the master as defined by the protocol.

- The software shall implement methods to syntonize the clock of the slave and the master as defined by the protocol.

2.1.2 General Requirements

- The software shall be implemented on Analog Device® ADSP-BF518F Blackfin™ Processor.
- The software shall integrate the open source lwIP protocol on Analog Device® ADSP-BF518F Blackfin™ Processor.
- The software shall implement the time synchronization protocol above UDP.
- The target hardware shall be connected to the test node (PC) via a direct Ethernet link.
- The software implemented on the target hardware shall be tested and validated against an open source implementation on a PC.
- Protocol and packet validation of the implementation shall be via a packet sniffing application such as wireshark.
- Both the nodes shall support printing debug messages for easy validation. Target shall print its messages via a UART console.

2.1.3 Ethernet MAC requirements

- The software shall be capable of sending and receiving Ethernet packets.
- Auto Negotiation shall be used by the MAC to connect to the corresponding node.

2.2 Scope

The following are in the scope of this project

- Implementation of IEEE Std.1588:2008 on the ADSP-BF518F.

The following are **NOT** in the scope of this project.

- Implementation of the other Ethernet AVB standards such as IEEE 802.1AS, IEEE 802.1Qat, IEEE 802.1Qav, etc…

3 Architectural Design

This software is architected using the layered software architecture design pattern. Refer to Rubel, Barry's commentary on Layered software architecture [8] for an overview of the Layered software architecture.

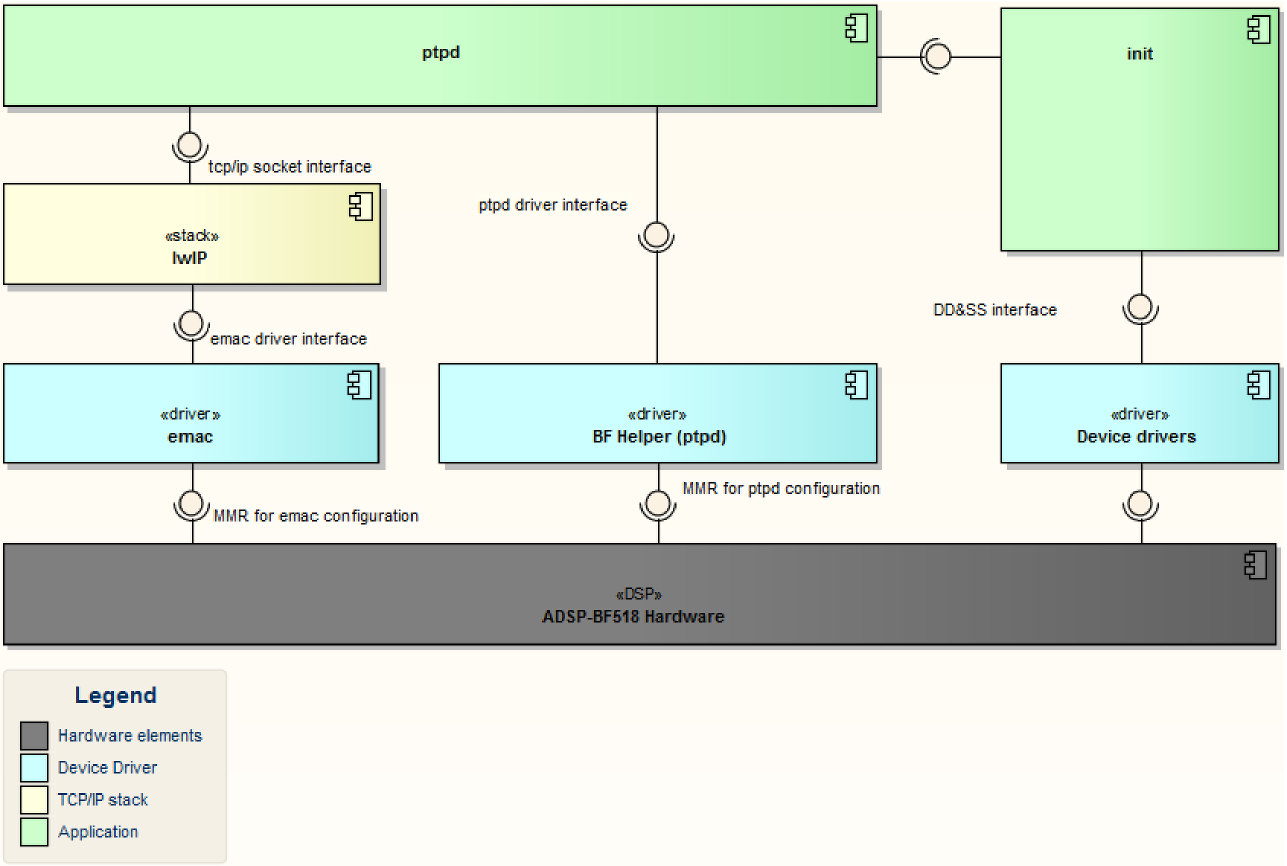


Figure 8: Software Architecture

The software is architected in a layered fashion. The hardware elements lie at the bottom. The next layer is made up of device drivers that would configure the hardware. The next layer is the LWip stack and on top of that sits the ptpd application. The following sections briefly explain each of the components.

3.1 <<driver>> emac

This component consists of the emac drivers that are directly provided by the integrated development environment (VisualDSP++) [10]. This component is invoked by the lwIP component to configure and control the Ethernet MAC peripheral.

3.2 <<driver>> BF Helper

This component provides helper functions for the higher layers in configuring and invoking the various features of the PTP_TSYNC module as described in Chapter 23 of the ADSP-BF518 Hardware Reference [7]. This component is directly invoked by the ptpd component to implement hardware specific functionalities.

3.3 <<driver>> Device drivers

This component consists of the all the device drivers (such as clock control, power management, Direct Memory Access, Timers, Serial Ports, etc..) that are directly provided by the integrated development environment (VisualDSP++) [10]. This component is invoked by the init component, to initialize the hardware and bring it up.

3.4 <<stack>> lwIP

This is the light weight TCP/IP stack provided by the IDE. It consists of the following features that would be used in this project

- IPv4
- IGMP
- UDP
- DHCP / Static IP management

The application note by Kaushal Sangh [11] describes the use of the lwIP stack provided by Analog Devices®. This component is invoked by the init function and this controls the sending and receiving of Ethernet data. It is also periodically invoked by the ptpd component to send and receive ptp packets.

3.5 <<application>> ptpd

This is the ptpd application / protocol implementation. It sends the protocol information over UDP via the lwIP stack. It is configured and started up by the init component. This component is a derivative of the open source Precision Time Stamping daemon (ptpd) [9].

3.6 <<application>> init

This component is responsible for initializing the ADSP-BF518 hardware and starting the lwIP and ptpd components.

3.7 Behavioural Model

The following diagram explains the top level sequence that is executed when the software begins. It also summarizes the flow of the software.

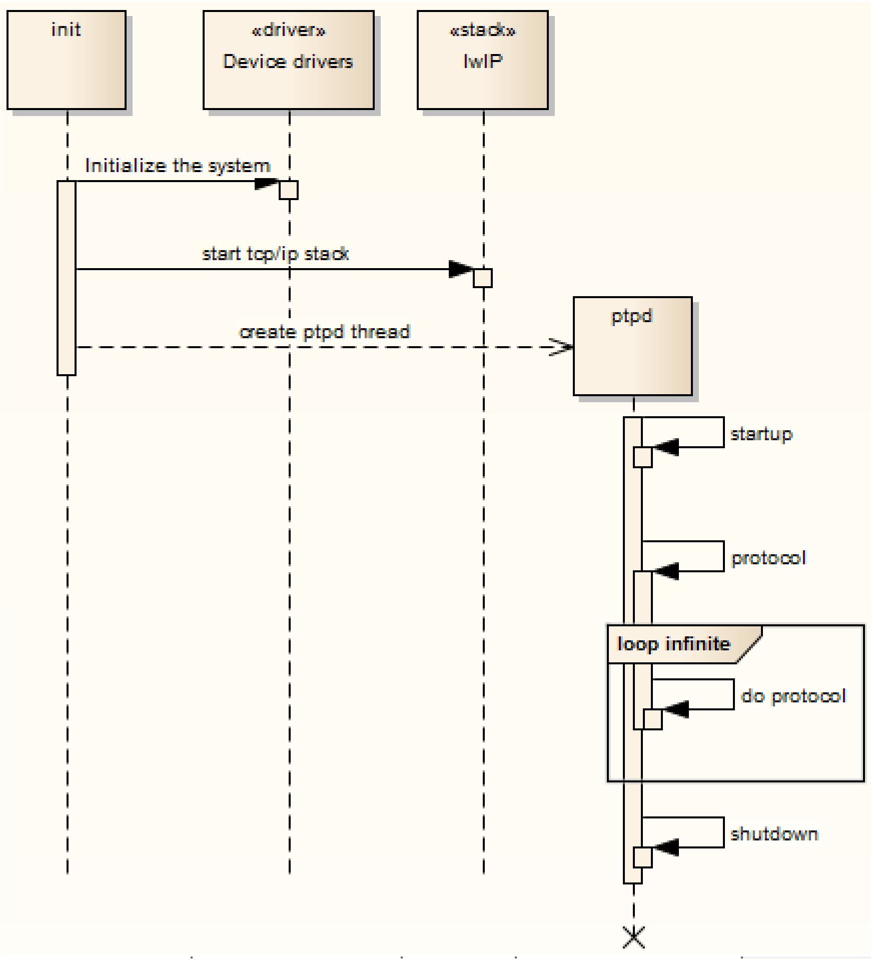


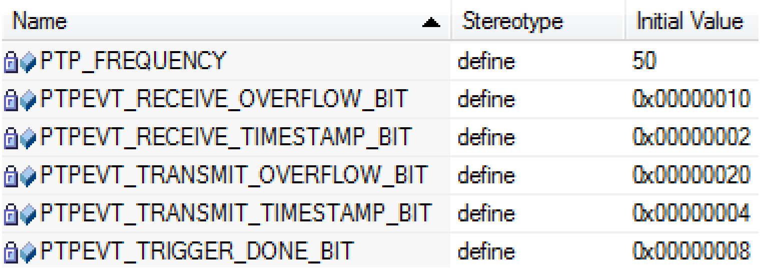
Figure 9: High level software sequence

4 Mechanistic Design

This section covers the detailed design for some of the components, including the interfaces, structural and behavioural elements. The components described in detail are the BF Helper and the ptpd.

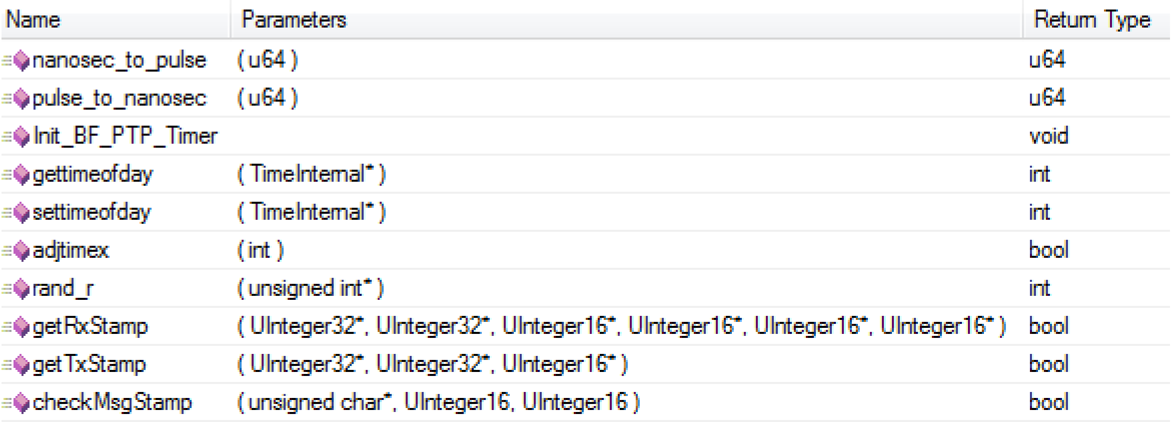
4.1 <<driver>> BF Helper

This section covers mechanistic design for the BF helper module. This component contains a set of operations that will configure the PTP hardware and provide utility functions to the upper layer in utilizing the features of the PTP_TSYNC module.



Name	Stereotype	Initial Value
PTP_FREQUENCY	define	50
PTPEVT_RECEIVE_OVERFLOW_BIT	define	0x00000010
PTPEVT_RECEIVE_TIMESTAMP_BIT	define	0x00000002
PTPEVT_TRANSMIT_OVERFLOW_BIT	define	0x00000020
PTPEVT_TRANSMIT_TIMESTAMP_BIT	define	0x00000004
PTPEVT_TRIGGER_DONE_BIT	define	0x00000008

Figure 10: BF Helper list of attributes



Name	Parameters	Return Type
nanosec_to_pulse	(u64)	u64
pulse_to_nanosec	(u64)	u64
Init_BF_PTP_Timer		void
gettimeofday	(TimeInternal*)	int
settimeofday	(TimeInternal*)	int
adjtimex	(int)	bool
rand_r	(unsigned int*)	int
getRxStamp	(UInteger32*, UInteger32*, UInteger16*, UInteger16*, UInteger16*, UInteger16*)	bool
getTxStamp	(UInteger32*, UInteger32*, UInteger16*)	bool
checkMsgStamp	(unsigned char*, UInteger16, UInteger16)	bool

Figure 11: BF Helper list of operations

4.1.1 Attributes

4.1.1.1 <<define>>PTP_FREQUENCY

This is the PTP input clock frequency.

4.1.1.2 <<defines>>PTPEVT_*

These definitions define the bit map of the PTP registers for easy access.

4.1.2 Operations

4.1.3 nanosec_to_pulse()

```
Input: u64 nanosec
Return: u64 (pulse value)
```

Converts a given input nanosecond value into a pulse with reference to the PTP_FREQUENCY.

4.1.4 pulse_to_nanosec ()

```
Input: u64 pulse
Return: u64 (nanosecond value)
```

Given a pulse with reference to the PTP_FREQUENCY, this API converts it into a nanosecond value.

4.1.5 Init_BF_PTP_Timer ()

```
Input: none
Return: none
```

Enables the PTP_TSYNC module and uses SCLK as input clock source by default. Sets up addend register value to an initial default.

4.1.6 Gettimeofday ()

```
Output: TimeInternal *tv
Return: int (success / failure)
```

Gets the current PTP clock time and returns it in tv.

4.1.7 settimeofday ()

```
Input: TimeInternal *tv
Return: int (success / failure)
```

Sets the PTP clock time to the value in tv.

4.1.8 adjtimex ()

```
Input: int adj
Return: bool (success / failure)
```

Adjusts the addend register found in the PTP_TSYNC module of the ADSP-BF518. This is described in Chapter 23 of the ADSP-BF518 Hardware Reference [7]. This register allows the hardware to finely tune its PTP clock by providing a frequency ratio offset. This is the clock syntonization scheme in ADSP-BF518.

4.1.9 rand_r()

```
Input: unsigned int *seedp
Return: int (random runmber)
```

A random number generation function.

4.1.10 getRxStamp()

```
Output: UInteger32 *retNumberOfSeconds
Output: UInteger32 *retNumberOfNanoSeconds
Output: UInteger16 *overflowCount
Output: UInteger16 *sequenceId
Output: UInteger16 *messageType
Output: UInteger16 *hashValue
Return: bool (success/failure)
```

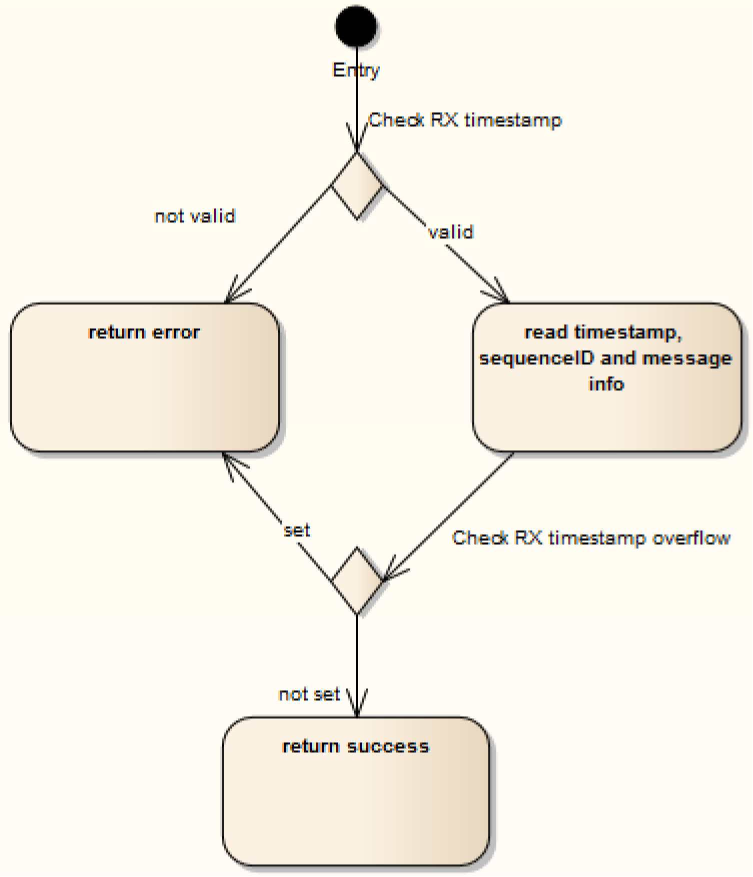


Figure 12: getRxStamp activity

This API gets the receive timestamp if it is valid. It also checks if the RX timestamp has overflowed (i.e. if more than one PTP RX event has occurred and the software has missed the previous time stamp).

4.1.11 getTxStamp ()

```
Output: UInteger32 *retNumberOfSeconds
Output: UInteger32 *retNumberOfNanoSeconds
Output: UInteger16 *overflowCount
Return: bool (success / failure)
```

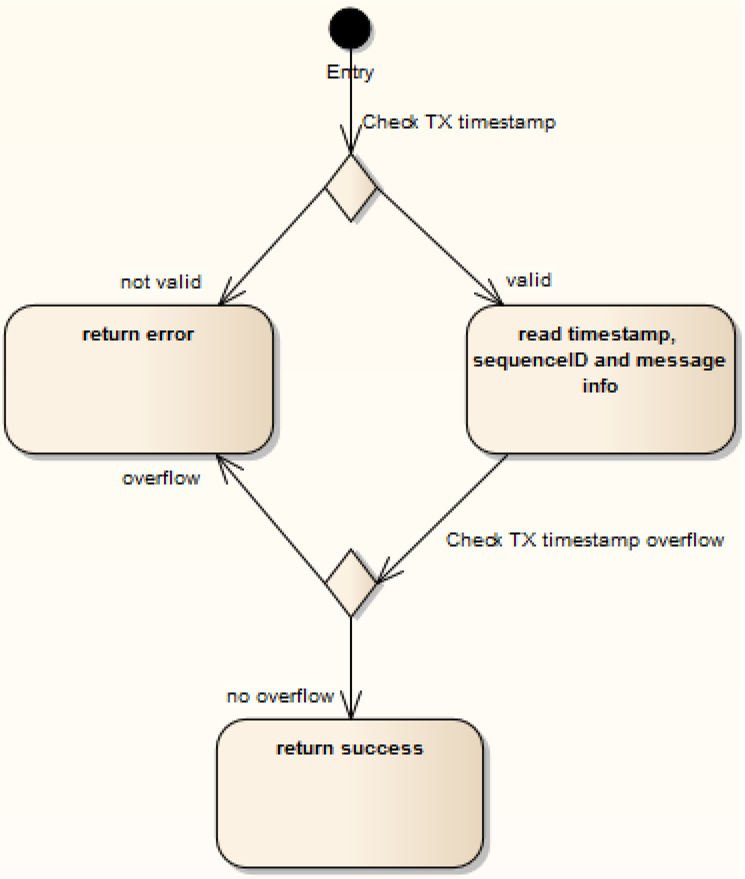


Figure 13: getTxStamp activity

This API gets the transmit timestamp if it is valid. It also checks if the TX timestamp has overflowed (i.e. if more than one PTP TX event has occurred and the software has missed the previous time stamp).

4.1.12 checkMsgStamp ()

```
Input: unsigned char* buf
Input: UInteger16 sequenceId
Input: UInteger16 messageType
Return: bool (Success / failure)
```

This API checks if the received message is the correct one and as expected.

4.2 <<application>> Ptpd

This section describes the ptpd protocol implementation in detail. The ptpd component has the following sub classes

- protocol (Implements the ptpd protocol)
- bmca (Implements the Best Master Clock Algorithm)
- msg (Handles message packing and unpacking)
- net (Handles all the net services)
- timer (Handlers the timer services)
- sys (Handles initialization and other utility functionalities)
- servo (Handles syntonization of clock)

The following diagram represents the class diagram of the ptpd component.

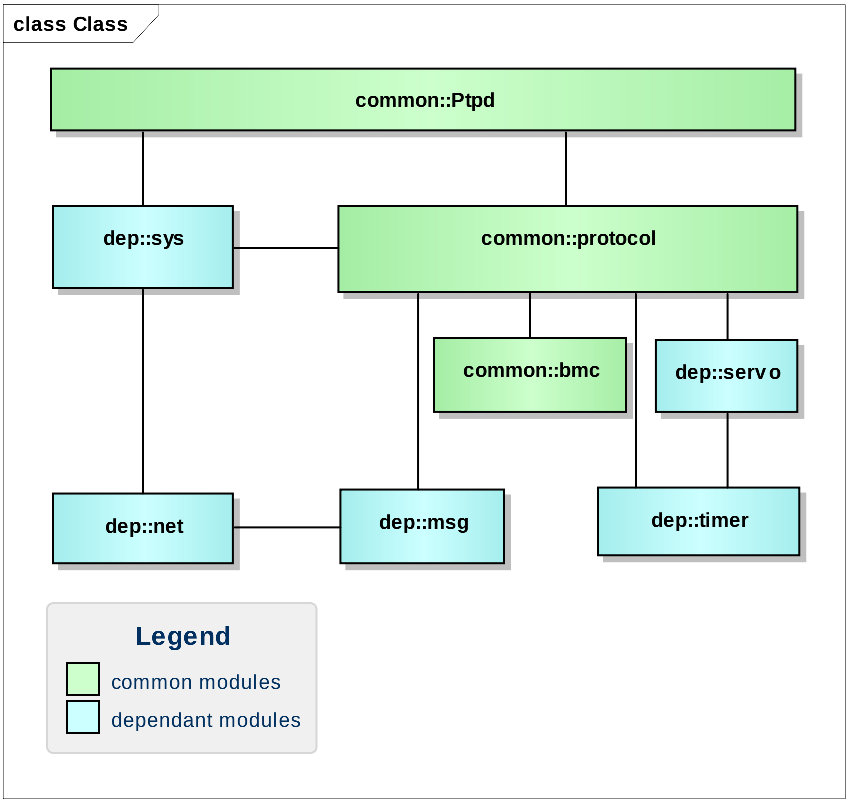


Figure 14: ptpd class diagram

4.2.1 Behavioural model

4.2.1.1 Initialization

The ptpd component is created by the init function. On creation, it initializes its parameters. This includes the clock, offset, variance, etc… It then starts the ptpd.

The ptpd has a state machine as described in Figure 16. As seen, the ptpd always begins in state PTP_INITIALIZING. From this state, the ptpd brings the net up using the net init call to the lwIP stack. It then initializes its data. It then configures the timer with the appropriate parameters. This call actually is translated by the BF helper as described in section 4.1 . The ptpd then initializes the clock, which is also translated by the BF helper. Thus having initialized itself, the ptpd transitions to the PTP_LISTENING state where the state machine takes over. The following is the activity model for the ptpd initialization.

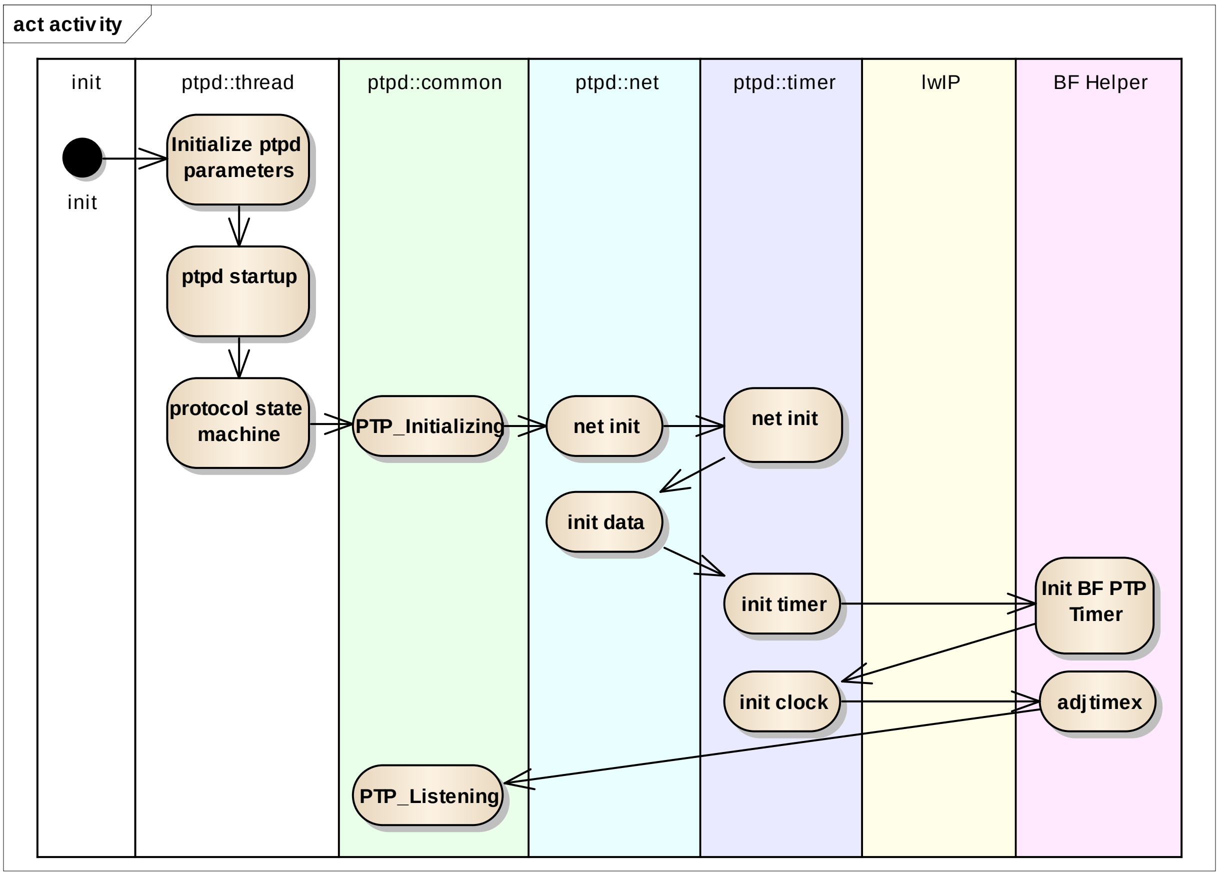


Figure 15: ptpd initialization

4.2.1.2 State machine

The PTP states are described in Table 1.

State	Description
INITIALIZING	While a port is in the INITIALIZING state, the port initializes its data sets, hardware, and communication facilities. No port of the clock shall place any PTP messages on its communication path. If one port of a boundary clock is in the INITIALIZING state, then all ports shall be in the INITIALIZING state.
FAULTY	The fault state of the protocol. A port in this state shall not place any PTP messages except for management messages that are a required response to another management message on its communication path. In a boundary clock, no activity on a faulty port shall affect the other ports of the device. If fault activity on a port in this state cannot be confined to the faulty port, then all ports shall be in the FAULTY state.
DISABLED	The port shall not place any messages on its communication path. In a boundary clock, no activity at the port shall be allowed to affect the activity at any other port of the boundary clock. A port in this state shall discard all PTP received messages except for management messages.
LISTENING	The port is waiting for the announceReceiptTimeout to expire or to receive an Announce message from a master. The purpose of this state is to allow orderly addition of clocks to a domain. A port in this state shall not place any PTP messages on its communication path except for Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up, or signalling messages, or management messages that are a required response to another management message.
MASTER	The port is behaving as a master port.
PASSIVE	The port shall not place any messages on its communication path except for Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up, or signalling messages, or management messages that are a required response to another management message.
UNCALIBRATED	One or more master ports have been detected in the domain. The appropriate master port has been selected, and the local port is preparing to synchronize to the selected master port. This is a transient state to allow initialization of synchronization servos, updating of data sets when a new master port has been selected, and other implementation-specific activity.
SLAVE	The port is synchronizing to the selected master port.

Table 1: PTP states [2]

After initialization, the ptpd component follows the following state machine. It uses two APIs, doState and toState to manage the state machine.

The doState API is continuously invoked and it automatically handles any new events such as timer expirations, new messages, etc…

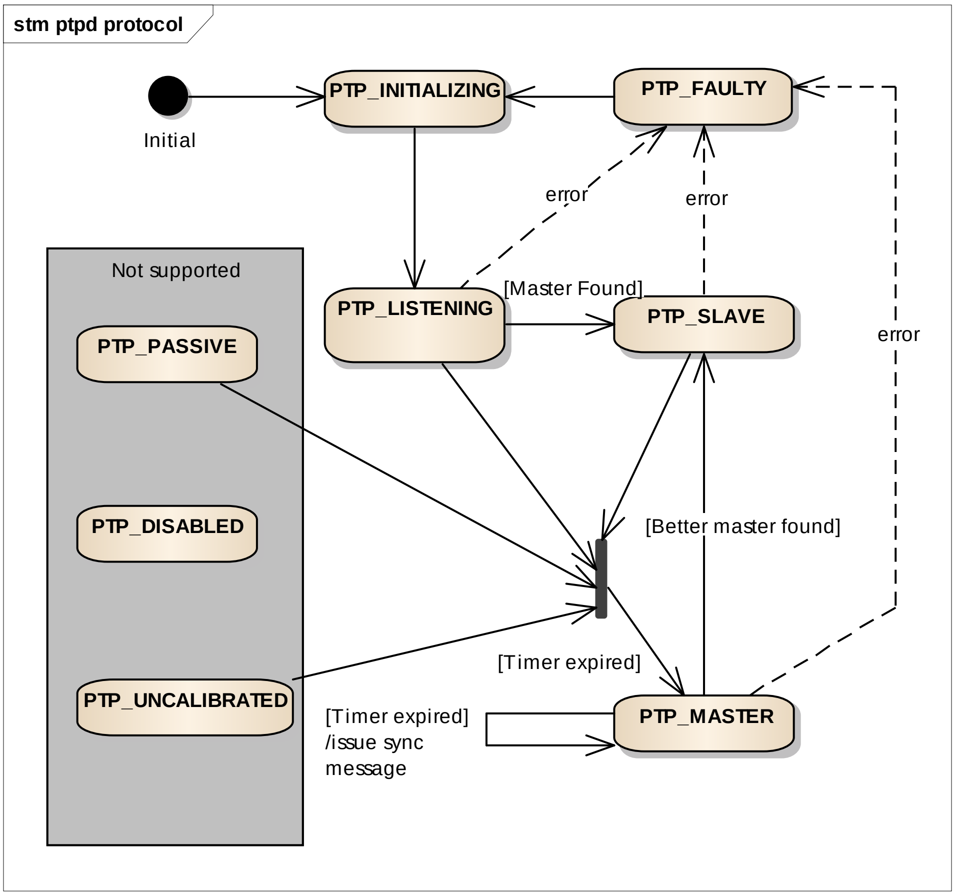


Figure 16: ptpd state machine

4.2.1.3 Event handling

Each of these states continuously invokes a "handle" API that handles all the incoming events.

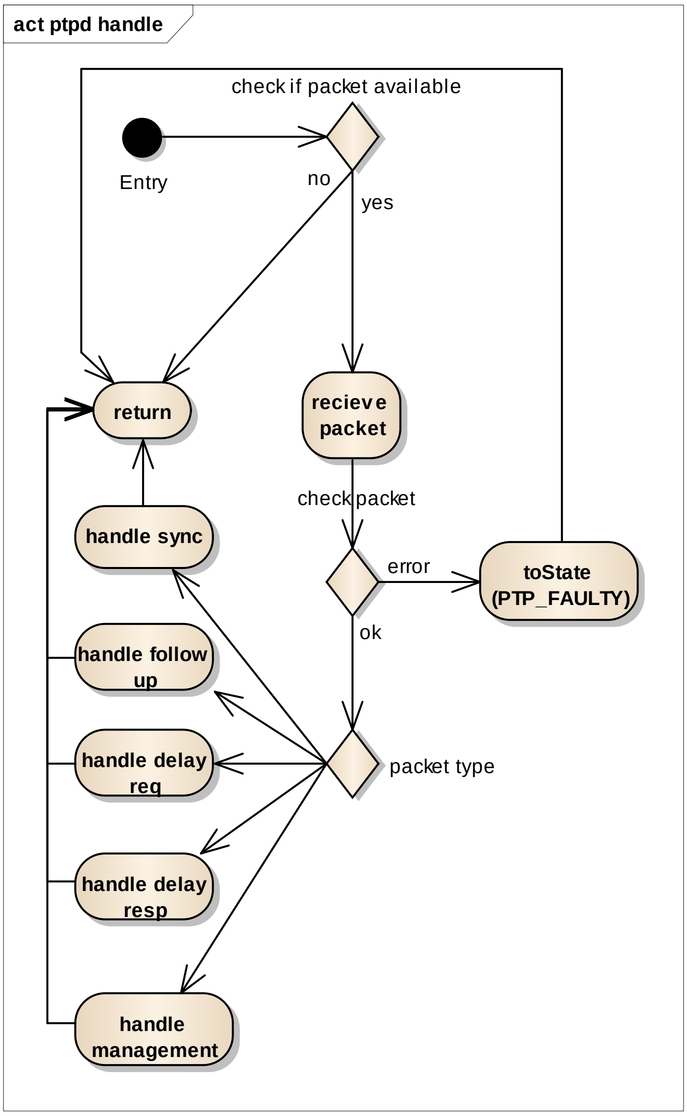
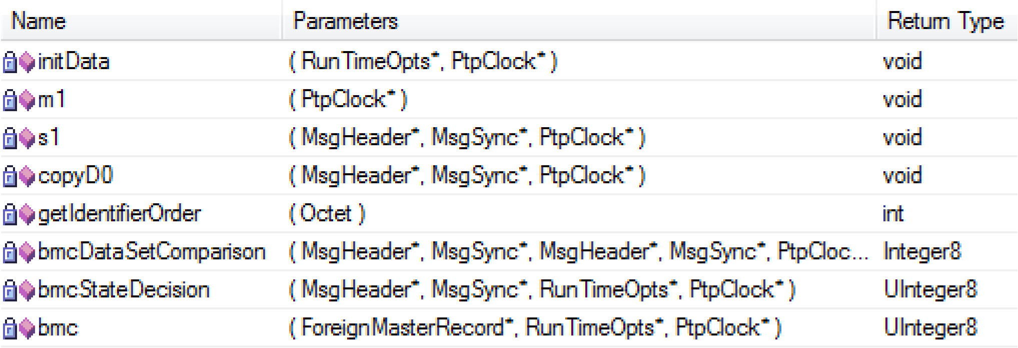


Figure 17: ptpd event handling

4.2.2 bmca (Best Master Clock Algorithm)

The following are the set of operations under the Best master clock algorithm. This class implements the algorithm described in section 1.2.1.4.1.1 . These operations are unchanged from the open source implementation expect for implementation specific changes such as IDE. The implementation can be referred at [9].



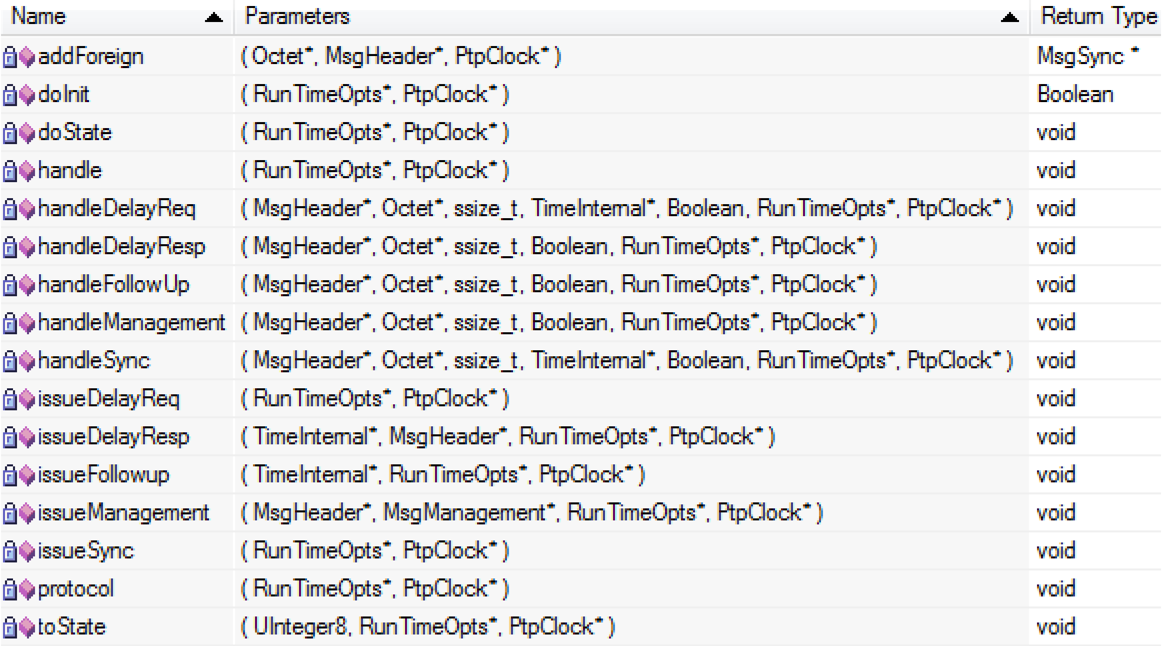
Name	Parameters	Return Type
initData	(RunTimeOpts*, PtpClock*)	void
m1	(PtpClock*)	void
s1	(MsgHeader*, MsgSync*, PtpClock*)	void
copyD0	(MsgHeader*, MsgSync*, PtpClock*)	void
getIdentifierOrder	(Octet)	int
bmcDataSetComparison	(MsgHeader*, MsgSync*, MsgHeader*, MsgSync*, PtpCloc...	Integer8
bmcStateDecision	(MsgHeader*, MsgSync*, RunTimeOpts*, PtpClock*)	UInteger8
bmc	(ForeignMasterRecord*, RunTimeOpts*, PtpClock*)	UInteger8

Figure 18: ptpd::bmca operations

4.2.3 protocol (Protocol implementation)

The following are the set of operations supported under protocol. This class implements the behaviour described in section 4.2.1 . Some of the operations in this class were modified in this implementation from the open source ptpd implementation. The operations that were modified are described in detail below. The other operations can be referenced from ptpd [9].

4.2.3.1 Operations



Name	Parameters	Return Type
addForeign	(Octet*, MsgHeader*, PtpClock*)	MsgSync *
doInit	(RunTimeOpts*, PtpClock*)	Boolean
doState	(RunTimeOpts*, PtpClock*)	void
handle	(RunTimeOpts*, PtpClock*)	void
handleDelayReq	(MsgHeader*, Octet*, ssize_t, TimeInternal*, Boolean, RunTimeOpts*, PtpClock*)	void
handleDelayResp	(MsgHeader*, Octet*, ssize_t, Boolean, RunTimeOpts*, PtpClock*)	void
handleFollowUp	(MsgHeader*, Octet*, ssize_t, Boolean, RunTimeOpts*, PtpClock*)	void
handleManagement	(MsgHeader*, Octet*, ssize_t, Boolean, RunTimeOpts*, PtpClock*)	void
handleSync	(MsgHeader*, Octet*, ssize_t, TimeInternal*, Boolean, RunTimeOpts*, PtpClock*)	void
issueDelayReq	(RunTimeOpts*, PtpClock*)	void
issueDelayResp	(TimeInternal*, MsgHeader*, RunTimeOpts*, PtpClock*)	void
issueFollowup	(TimeInternal*, RunTimeOpts*, PtpClock*)	void
issueManagement	(MsgHeader*, MsgManagement*, RunTimeOpts*, PtpClock*)	void
issueSync	(RunTimeOpts*, PtpClock*)	void
protocol	(RunTimeOpts*, PtpClock*)	void
toState	(UInteger8, RunTimeOpts*, PtpClock*)	void

Figure 19: ptpd::protocol operations

4.2.3.1.1 protocol ()

`Input: RunTimeOpts *rtOpts`

```
Input: PtpClock *ptpClock
Return: None
```

This operation is the fundamental interface operation for the entire ptpd module. The implementation of this operation has one fundamental difference from the open source implementation. A VDK_Sleep () call was added to this function that allows it to sleep during the execution of the protocol thus preserving the processor's resources.

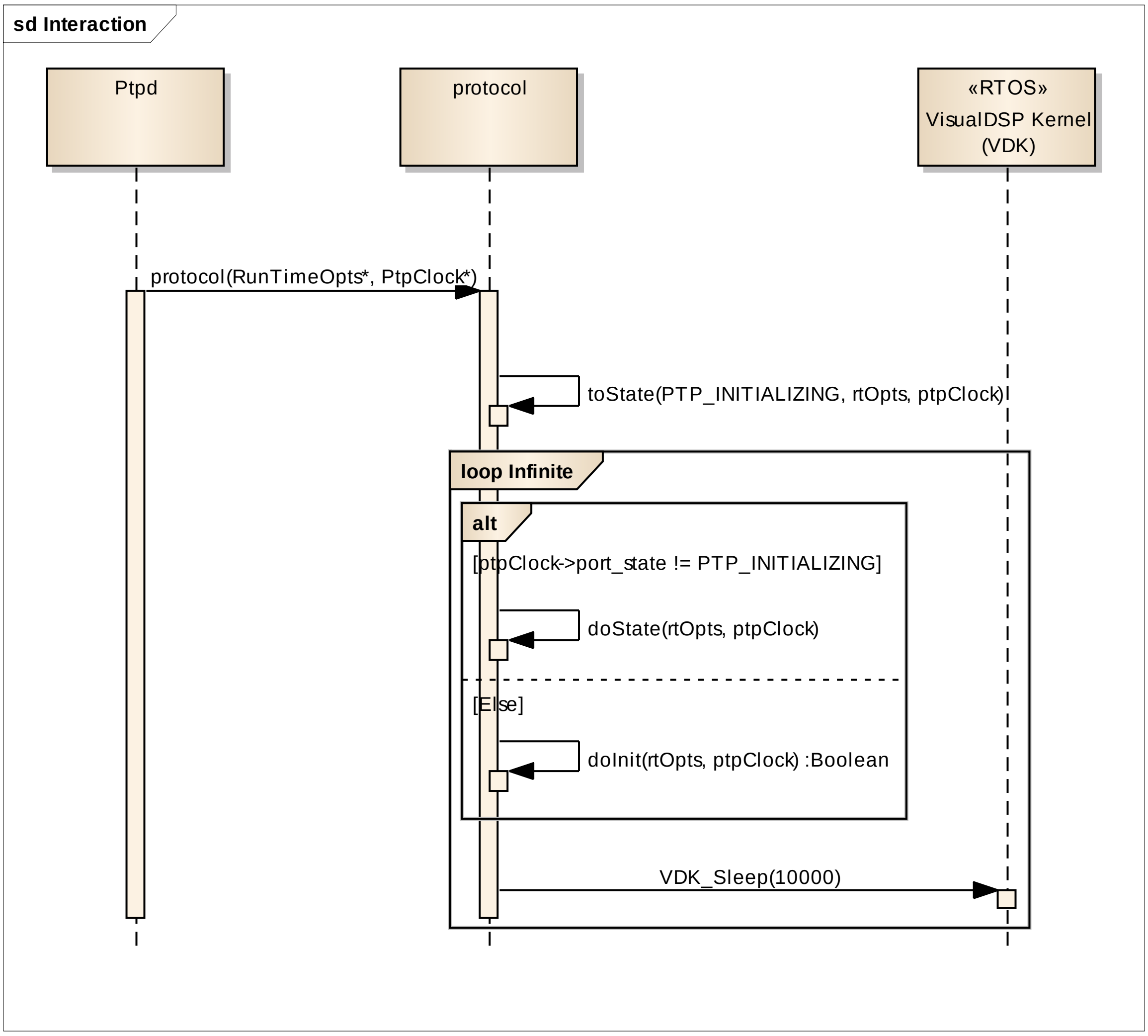

sd Interaction
Ptpd
protocol
«RTOS»
VisualDSP Kernel
(VDK)
protocol(RunTimeOpts*, PtpClock*)
toState(PTP_INITIALIZING, rtOpts, ptpClock)
loop Infinite
alt
[ptpClock->port_state != PTP_INITIALIZING]
doState(rtOpts, ptpClock)
[Else]
doInit(rtOpts, ptpClock) :Boolean
VDK_Sleep(10000)

Figure 20: ptpd::protocol::protocol() sequence

4.2.3.1.2 handleSync ()

```
Input: MsgHeader *header
Input: Octet *msgIbuf
Input: ssize_t length
Input: TimeInternal *time
Input: Boolean isFromSelf
Input: RunTimeOpts *rtOpts
Input: PtpClock *ptpClock
Return: None
```

This operation handles the Sync messages. The modification in this operation is in handling sync message as a master. The operation attempts to get the Transmit timestamp of the previous Sync message and updates its internal clock based on this time stamp.

4.2.3.1.3 handleDelayReq ()

```
Input: MsgHeader *header
Input: Octet *msgIbuf
Input: ssize_t length
Input: TimeInternal *time
Input: Boolean isFromSelf
Input: RunTimeOpts *rtOpts
Input: PtpClock *ptpClock
Return: None
```

This operation handles the Delay Request messages. The modification in this operation is in handling Delay Request message as a slave. The operation attempts to get the Transmit timestamp of the previous Delay Request message and updates its internal clock based on this time stamp.

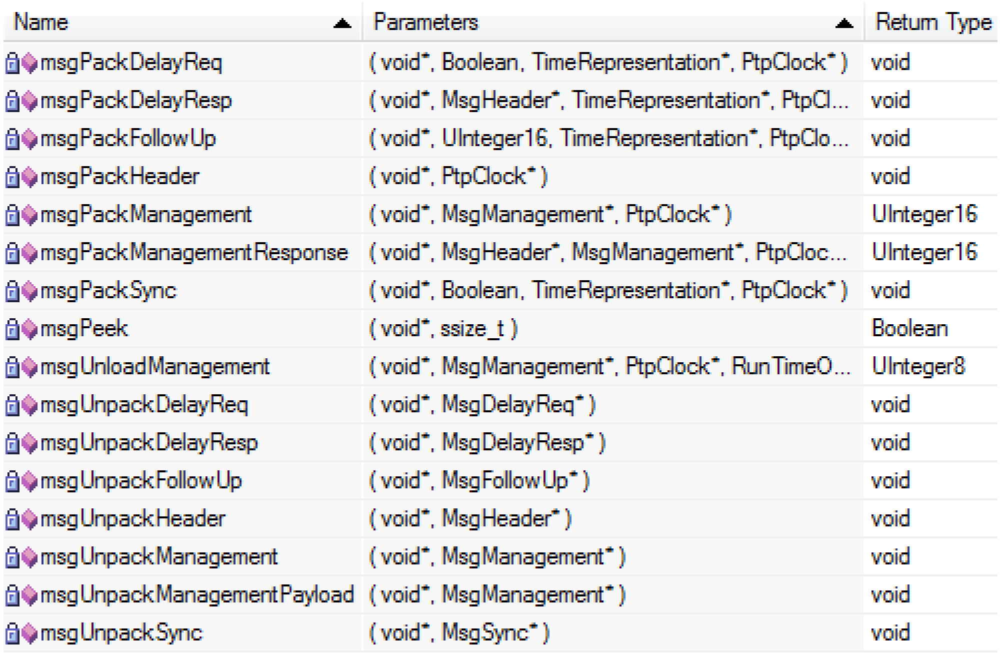
4.2.3.1.4 HandleDelayResp ()

```
Input: MsgHeader *header
Input: Octet *msgIbuf
Input: ssize_t length
Input: TimeInternal *time
Input: Boolean isFromSelf
Input: RunTimeOpts *rtOpts
Input: PtpClock *ptpClock
Return: None
```

This operation handles the Delay Response messages. The modification in this operation is in handling Delay Response message as a slave. The operation attempts to get the Transmit timestamp of the previous Delay Response message and updates its internal clock based on this time stamp.

4.2.4 msg (Message handling)

The following operations handle all message processing such as packing and unpacking the PTP messages. These operations are unchanged from the open source implementation expect for implementation specific changes such as IDE. The implementation can be referred at [9].



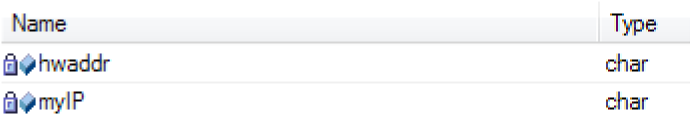
Name	Parameters	Return Type
msgPackDelayReq	(void*, Boolean, TimeRepresentation*, PtpClock*)	void
msgPackDelayResp	(void*, MsgHeader*, TimeRepresentation*, PtpCl...	void
msgPackFollowUp	(void*, UInteger16, TimeRepresentation*, PtpClo...	void
msgPackHeader	(void*, PtpClock*)	void
msgPackManagement	(void*, MsgManagement*, PtpClock*)	UInteger16
msgPackManagementResponse	(void*, MsgHeader*, MsgManagement*, PtpCloc...	UInteger16
msgPackSync	(void*, Boolean, TimeRepresentation*, PtpClock*)	void
msgPeek	(void*, ssize_t)	Boolean
msgUnloadManagement	(void*, MsgManagement*, PtpClock*, RunTimeO...	UInteger8
msgUnpackDelayReq	(void*, MsgDelayReq*)	void
msgUnpackDelayResp	(void*, MsgDelayResp*)	void
msgUnpackFollowUp	(void*, MsgFollowUp*)	void
msgUnpackHeader	(void*, MsgHeader*)	void
msgUnpackManagement	(void*, MsgManagement*)	void
msgUnpackManagementPayload	(void*, MsgManagement*)	void
msgUnpackSync	(void*, MsgSync*)	void

Figure 21: ptpd::msg handling operations

4.2.5 net (Net services)

The following operations handle all the net services such as initializing, sending, receiving and shutting down the net interface. These operations are internally translated into either lwIP or BF helper calls. Almost all of these operations were modified in this project and the subsequent sections explain their design.

4.2.5.1 Attributes



Name	Type
hwaddr	char
myIP	char

Figure 22: ptpd::net attributes

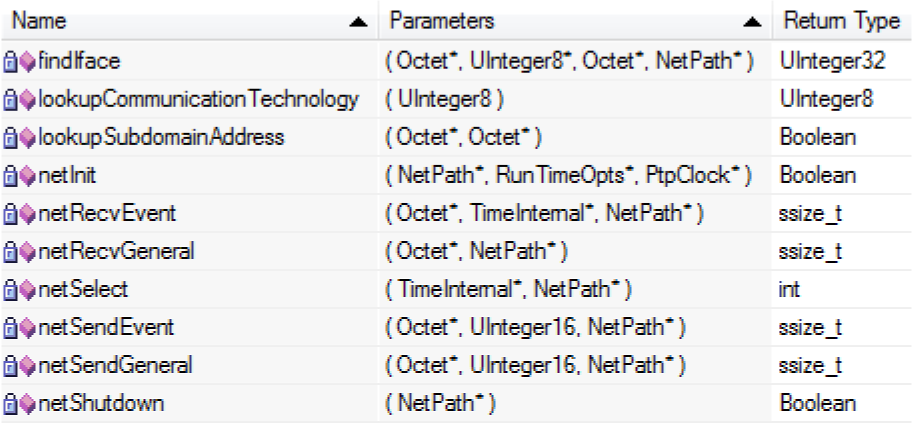
4.2.5.1.1 Hwaddr

This attribute stores the MAC address of the device.

4.2.5.1.2 myIP

This attribute stores the IP address of the device.

4.2.5.2 Operations



Name	Parameters	Return Type
findIface	(Octet*, UInteger8*, Octet*, NetPath*)	UInteger32
lookupCommunicationTechnology	(UInteger8)	UInteger8
lookupSubdomainAddress	(Octet*, Octet*)	Boolean
netInit	(NetPath*, RunTimeOpts*, PtpClock*)	Boolean
netRecvEvent	(Octet*, TimeInternal*, NetPath*)	ssize_t
netRecvGeneral	(Octet*, NetPath*)	ssize_t
netSelect	(TimeInternal*, NetPath*)	int
netSendEvent	(Octet*, UInteger16, NetPath*)	ssize_t
netSendGeneral	(Octet*, UInteger16, NetPath*)	ssize_t
netShutdown	(NetPath*)	Boolean

Figure 23: ptpd::net operations

4.2.5.2.1 lookupCommunicationTechnology ()

```
Input: UInteger8 communicationTechnology
Return: UInteger8 (Enumeration value of the technology)
```

This operation should return the underlying physical technology used by the low level layers. In the case of ADSP-BF518, the function statically returns PTP_DEFAULT.

4.2.5.2.2 findIface ()

```
Output: Octet *ifaceName
Output: UInteger8 *communicationTechnology
Output: Octet *uuid
Output: NetPath *netPath
Return:  UInteger32 (Enumeration value of the interface)
```

This operation dynamically attempts to find the interface used by the low level layers. In the case of ADSP-BF518, the following is statically returned.

```
    ifaceName: Micrel KSZ8893M (The PHY used in the ADSP-BF518 board)
    communicationTechnology: PTP_ETHER
    uuid: The unicast MAC address
```

4.2.5.2.3 netInit ()

```
Input: NetPath *netPath
Input: RunTimeOpts *rtOpts
Input: PtpClock *ptpClock
Return: Boolean (Success / Failure)
```

This operation is responsible for initializing the net interface, open sockets and bind them. The following is the flow of this operation.

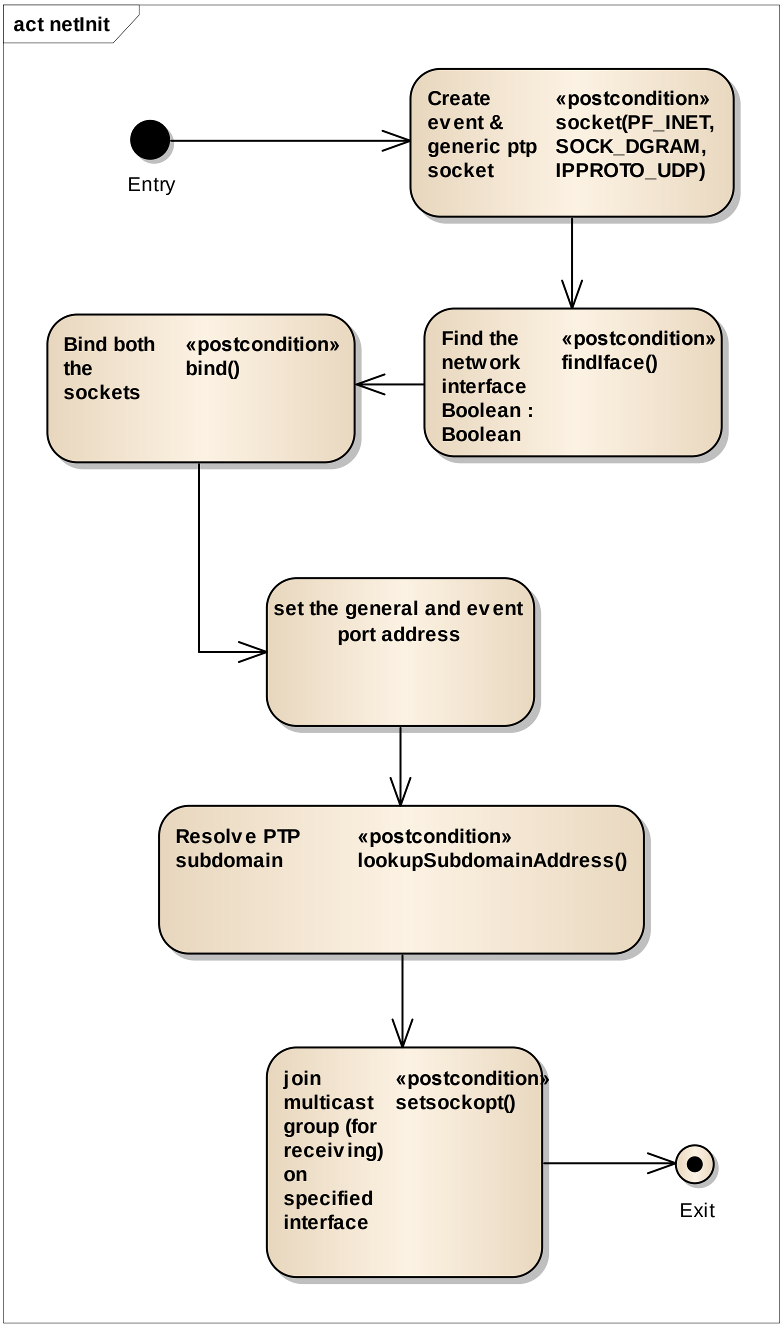

act netInit
Entry
Create event & generic ptp socket
«postcondition» socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)
Find the network interface Boolean : Boolean
«postcondition» findIface()
Bind both the sockets
«postcondition» bind()
set the general and event port address
Resolve PTP subdomain
«postcondition» lookupSubdomainAddress()
join multicast group (for receiving) on specified interface
«postcondition» setsockopt()
Exit

Figure 24: ptpd::net::netinit() flow

4.2.5.2.4 netRecvEvent ()

```
Input: Octet *buf
Input: TimeInternal *time
Input: NetPath *netPath
Return: ssize_t (Size of the received packet)
```

This operation is responsible for receiving “event” messages for the PTP component. The modification in this operation is the reading of the received hardware time stamp that will allow the PTP component to reliably implement the PTP protocol.

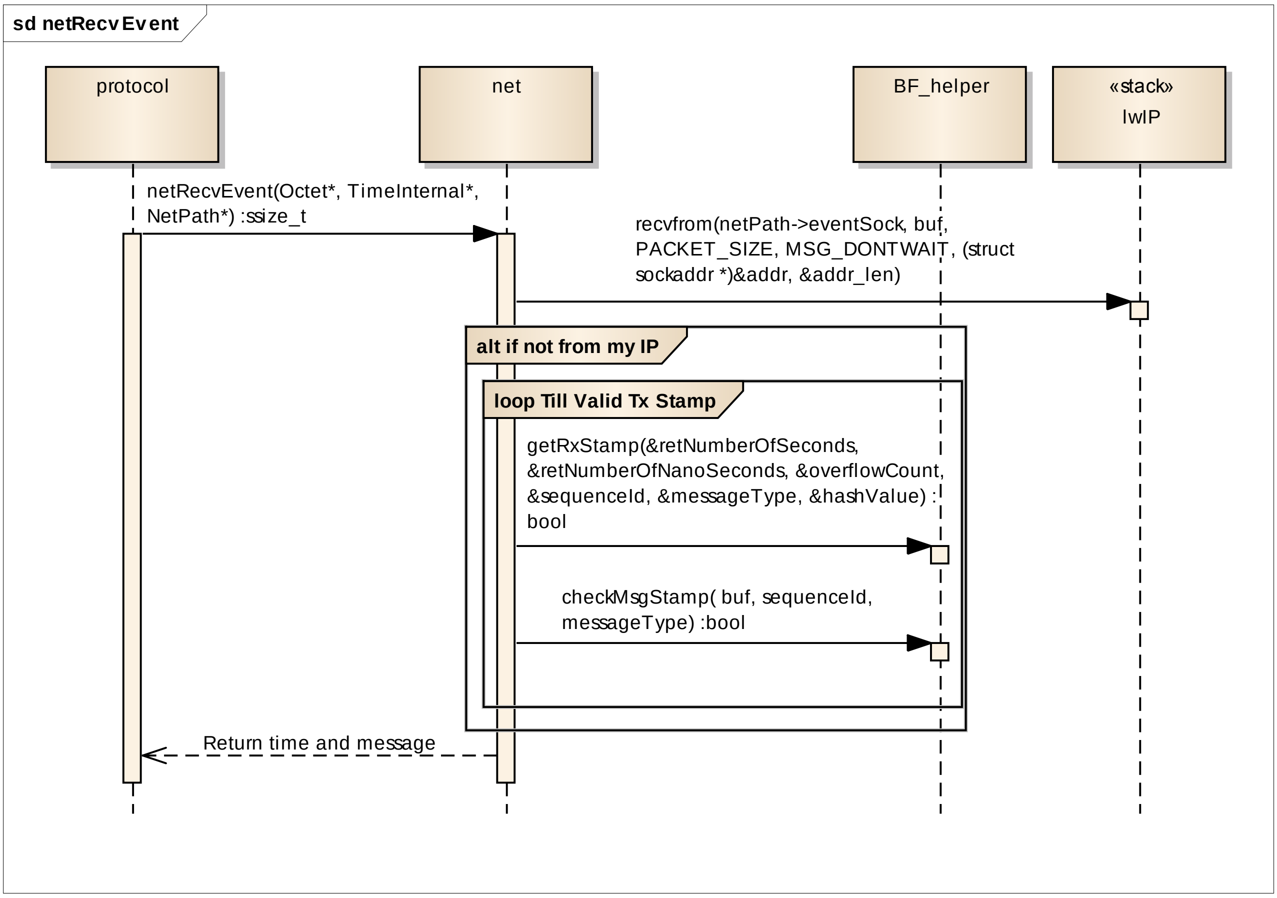
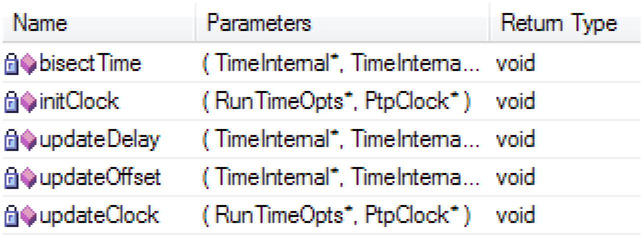

sd netRecvEvent
protocol
net
BF_helper
«stack»
lwIP
netRecvEvent(Octet*, TimeInternal*, NetPath*) :ssize_t
recvfrom(netPath->eventSock, buf, PACKET_SIZE, MSG_DONTWAIT, (struct sockaddr *)&addr, &addr_len)
alt if not from my IP
loop Till Valid Tx Stamp
getRxStamp(&retNumberOfSeconds, &retNumberOfNanoSeconds, &overflowCount, &sequenceId, &messageType, &hashValue) : bool
checkMsgStamp(buf, sequenceId, messageType) :bool
Return time and message

Figure 25: net::ptpd::netRecvEvent () Sequence

4.2.6 servo (Clock syntonization functions)

This class implements all the operations required for the device to syntonize and manage its clock. These functions are not modified from the original implementation and they can be referred to at the ptpd code base [9].



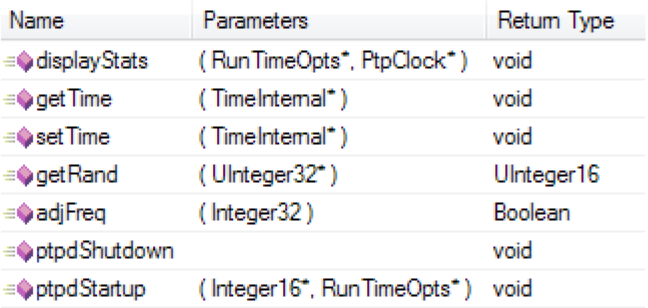
Name	Parameters	Return Type
bisectTime	(TimeInternal*, TimeInterna...	void
initClock	(RunTimeOpts*, PtpClock*)	void
updateDelay	(TimeInternal*, TimeInterna...	void
updateOffset	(TimeInternal*, TimeInterna...	void
updateClock	(RunTimeOpts*, PtpClock*)	void

Figure 26: ptpd::servo operations

4.2.7 sys (utility functions)

This class implements all the init, shutdown and utility functionalities of the PTP component. These functions are internally translated into BF helper calls.

4.2.7.1 Operations



Name	Parameters	Return Type
displayStats	(RunTimeOpts*, PtpClock*)	void
getTime	(TimeInternal*)	void
setTime	(TimeInternal*)	void
getRand	(UInteger32*)	UInteger16
adjFreq	(Integer32)	Boolean
ptpdShutdown		void
ptpdStartup	(Integer16*, RunTimeOpts*)	void

Figure 27: ptpd::sys operations

The following sequence explains the implementation of all the “sys” class functions.

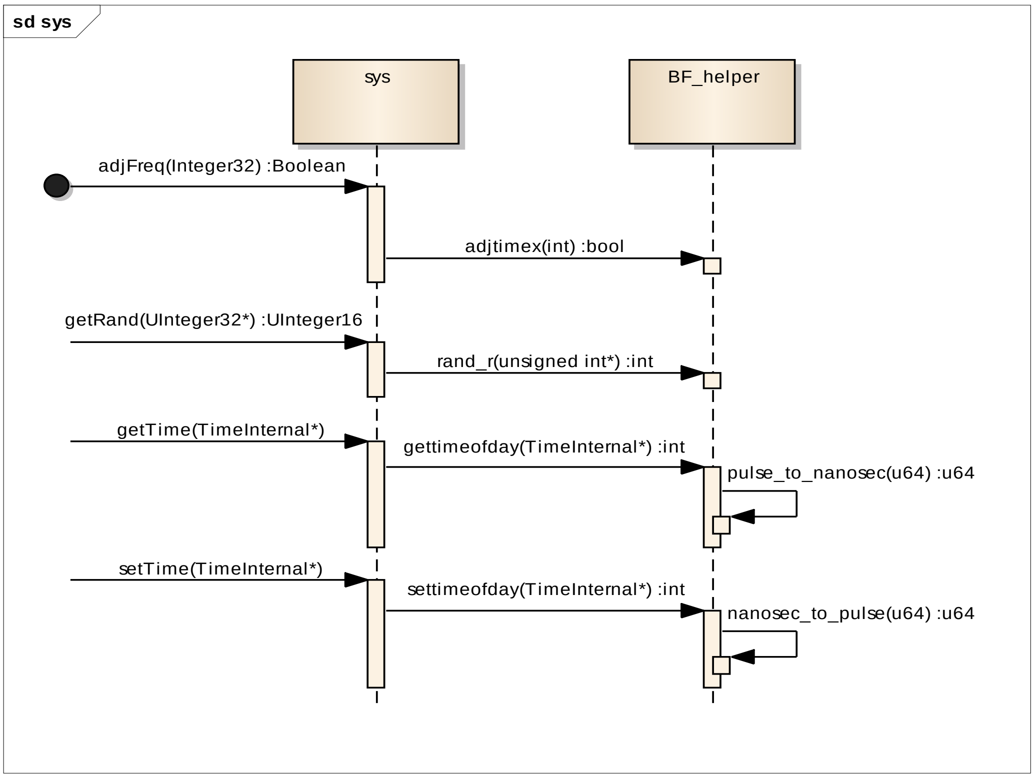

sd sys
sys
BF_helper
adjFreq(Integer32) :Boolean
adjtimex(int) :bool
getRand(UInteger32*) :UInteger16
rand_r(unsigned int*) :int
getTime(TimeInternal*)
gettimeofday(TimeInternal*) :int
pulse_to_nanosec(u64) :u64
setTime(TimeInternal*)
settimeofday(TimeInternal*) :int
nanosec_to_pulse(u64) :u64

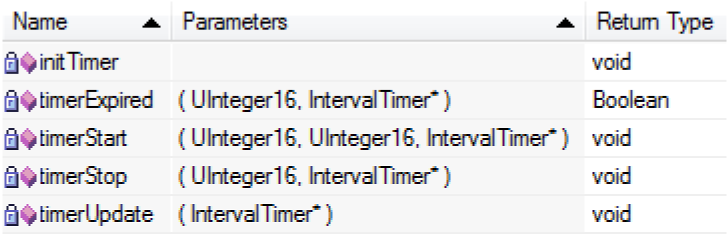
Figure 28: ptpd::sys sequence

4.2.8 timer (Timer functions)

These operations implement all the timer related functions needed by the PTP protocol. They are internally translated into BF helper calls. Many of these functions were modified and the sections below describe the design in detail.

4.2.8.1 Operations

The following are the operations within the timer class.



Name ▲	Parameters ▲	Return Type
initTimer		void
timerExpired	(UInteger16, IntervalTimer*)	Boolean
timerStart	(UInteger16, UInteger16, IntervalTimer*)	void
timerStop	(UInteger16, IntervalTimer*)	void
timerUpdate	(IntervalTimer*)	void

Figure 29: ptpd::timer operations

4.2.8.1.1 initTimer ()

```
Input: None
Return: None
```

This operation is responsible for initializing the timer. In ADSP-BF518, this operation simply translates the call to the BF Helper function Init_BF_PTP_Timer ().

5 Testing

5.1 Test Setup

The following is the test setup that was used.

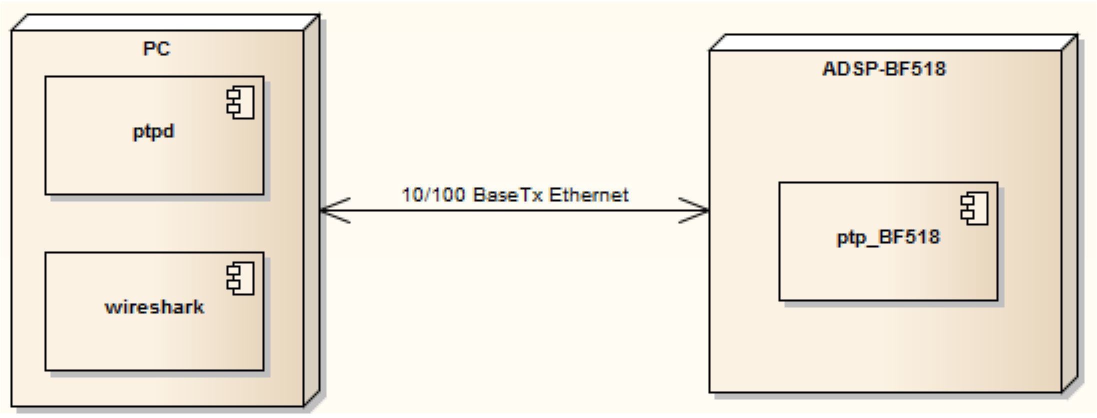


Figure 30: Test Setup

The above test setup is used for testing the ptp_BF518 software. An open source implementation of ptp is executed from a windows PC. The PC also runs wireshark to capture the network packets as they sent / received.

5.2 Test Sequence

The following sequence is followed while executing the test setup

1) Power on the PC

2) Connect the PC to the ADSP_BF518 via a direct 10/100BaseTx Ethernet cable

3) Configure the PC to static IP (169.254.0.1)

4) Start the wireshark application on the PC and set it to capture mode.

5) Start the UART terminal emulator on the PC to capture the BF518 logs

6) Start the ptpd application on the PC.

7) Configure BF518 to static IP (169.254.0.2)

8) Start the BF518 application.

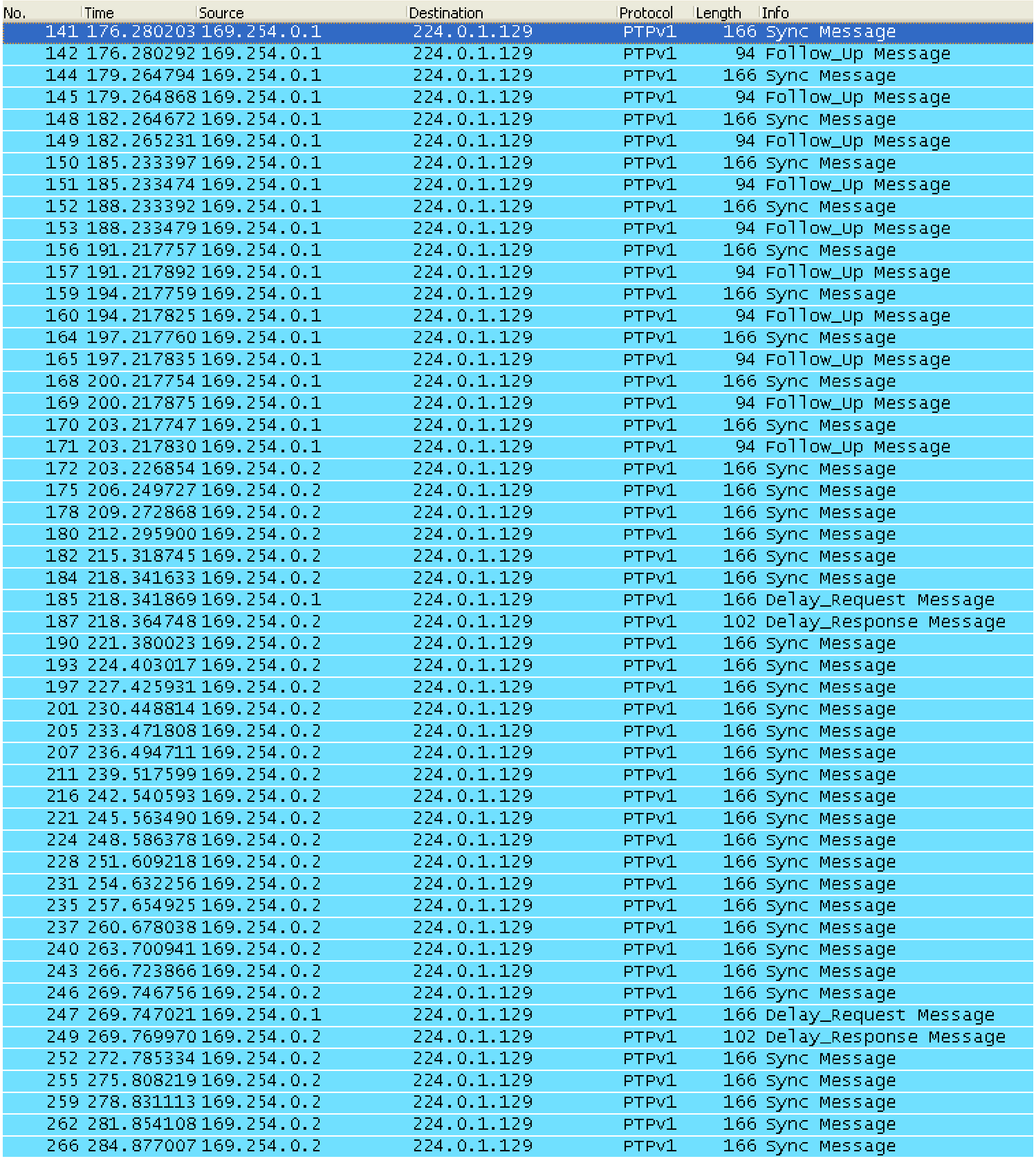
9) Capture the logs from PC, BF518 and wireshark.

5.3 Test Results

5.3.1.1 Wireshark reports

5.3.1.2 Captured packets

The following is a snapshot of all the packets transmitted over the test Ethernet link.



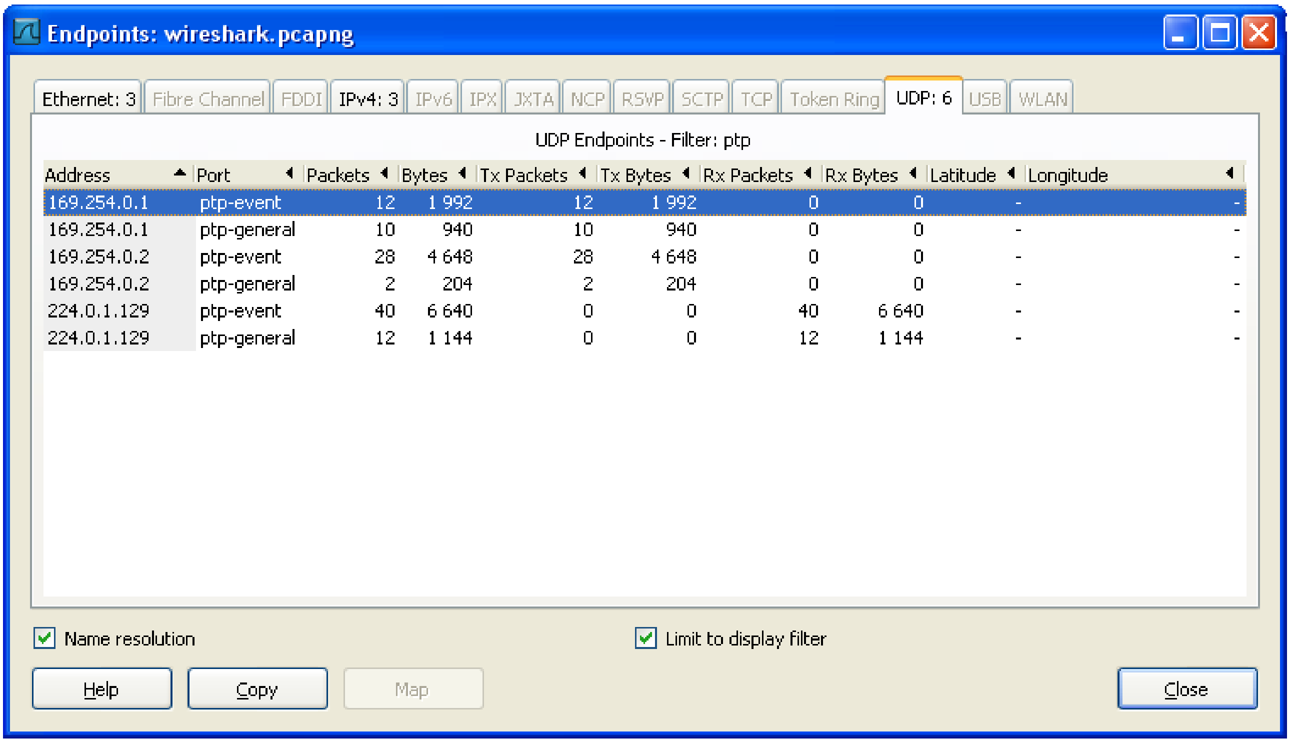
No.	Time	Source	Destination	Protocol	Length	Info
141	176.280203	169.254.0.1	224.0.1.129	PTPv1	166	Sync Message
142	176.280292	169.254.0.1	224.0.1.129	PTPv1	94	Follow_Up Message
144	179.264794	169.254.0.1	224.0.1.129	PTPv1	166	Sync Message
145	179.264868	169.254.0.1	224.0.1.129	PTPv1	94	Follow_Up Message
148	182.264672	169.254.0.1	224.0.1.129	PTPv1	166	Sync Message
149	182.265231	169.254.0.1	224.0.1.129	PTPv1	94	Follow_Up Message
150	185.233397	169.254.0.1	224.0.1.129	PTPv1	166	Sync Message
151	185.233474	169.254.0.1	224.0.1.129	PTPv1	94	Follow_Up Message
152	188.233392	169.254.0.1	224.0.1.129	PTPv1	166	Sync Message
153	188.233479	169.254.0.1	224.0.1.129	PTPv1	94	Follow_Up Message
156	191.217757	169.254.0.1	224.0.1.129	PTPv1	166	Sync Message
157	191.217892	169.254.0.1	224.0.1.129	PTPv1	94	Follow_Up Message
159	194.217759	169.254.0.1	224.0.1.129	PTPv1	166	Sync Message
160	194.217825	169.254.0.1	224.0.1.129	PTPv1	94	Follow_Up Message
164	197.217760	169.254.0.1	224.0.1.129	PTPv1	166	Sync Message
165	197.217835	169.254.0.1	224.0.1.129	PTPv1	94	Follow_Up Message
168	200.217754	169.254.0.1	224.0.1.129	PTPv1	166	Sync Message
169	200.217875	169.254.0.1	224.0.1.129	PTPv1	94	Follow_Up Message
170	203.217747	169.254.0.1	224.0.1.129	PTPv1	166	Sync Message
171	203.217830	169.254.0.1	224.0.1.129	PTPv1	94	Follow_Up Message
172	203.226854	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
175	206.249727	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
178	209.272868	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
180	212.295900	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
182	215.318745	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
184	218.341633	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
185	218.341869	169.254.0.1	224.0.1.129	PTPv1	166	Delay_Request Message
187	218.364748	169.254.0.2	224.0.1.129	PTPv1	102	Delay_Response Message
190	221.380023	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
193	224.403017	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
197	227.425931	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
201	230.448814	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
205	233.471808	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
207	236.494711	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
211	239.517599	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
216	242.540593	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
221	245.563490	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
224	248.586378	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
228	251.609218	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
231	254.632256	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
235	257.654925	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
237	260.678038	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
240	263.700941	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
243	266.723866	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
246	269.746756	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
247	269.747021	169.254.0.1	224.0.1.129	PTPv1	166	Delay_Request Message
249	269.769970	169.254.0.2	224.0.1.129	PTPv1	102	Delay_Response Message
252	272.785334	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
255	275.808219	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
259	278.831113	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
262	281.854108	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message
266	284.877007	169.254.0.2	224.0.1.129	PTPv1	166	Sync Message

Figure 31: Wireshark captured packets

5.3.1.3 Endpoints

As seen in the figure below, there are two end points on the system 169.254.0.1 (PC) and 169.254.0.2 (ADSP-BF518). The 224.0.1.129 is a multicast reception address as noted by the complete lack of TX packets from it.

It can be noticed that the two nodes, each have a ptp-event and a ptp-general port which has transmitted multiple packets.



UDP Endpoints - Filter: ptp

Address	Port	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Latitude	Longitude
169.254.0.1	ptp-event	12	1 992	12	1 992	0	0	-	-
169.254.0.1	ptp-general	10	940	10	940	0	0	-	-
169.254.0.2	ptp-event	28	4 648	28	4 648	0	0	-	-
169.254.0.2	ptp-general	2	204	2	204	0	0	-	-
224.0.1.129	ptp-event	40	6 640	0	0	40	6 640	-	-
224.0.1.129	ptp-general	12	1 144	0	0	12	1 144	-	-

☑ Name resolution

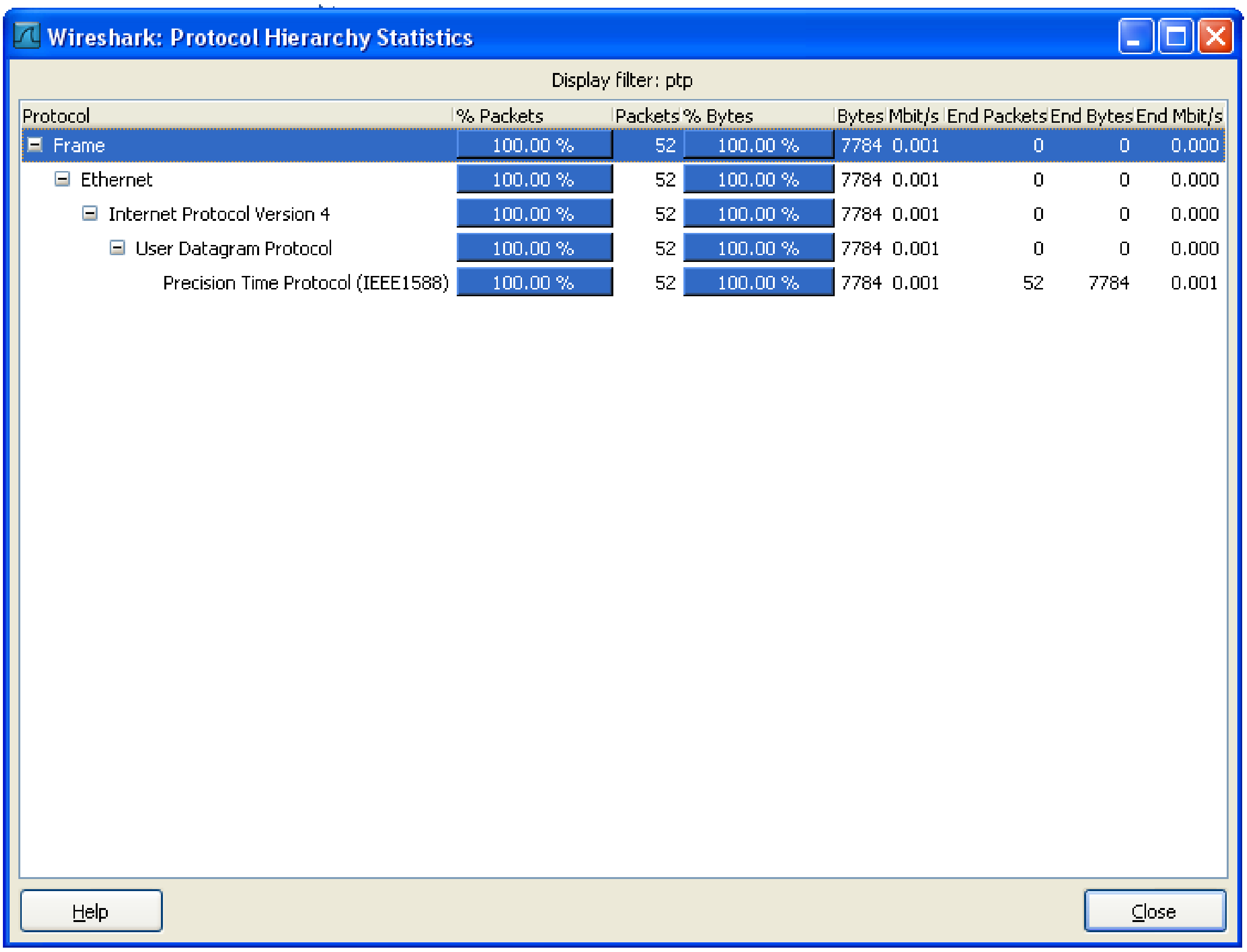
☑ Limit to display filter

Figure 32: Wireshark endpoints

5.3.1.4 Protocol Hierarchy

The figure below shows the protocol and the protocol hierarchy among all the packets sent over the Ethernet link.

As can be seen, all packets sent over the link confirm to IEEE std.1588 and are sent over UDP as stated in the requirements.



Display filter: ptp

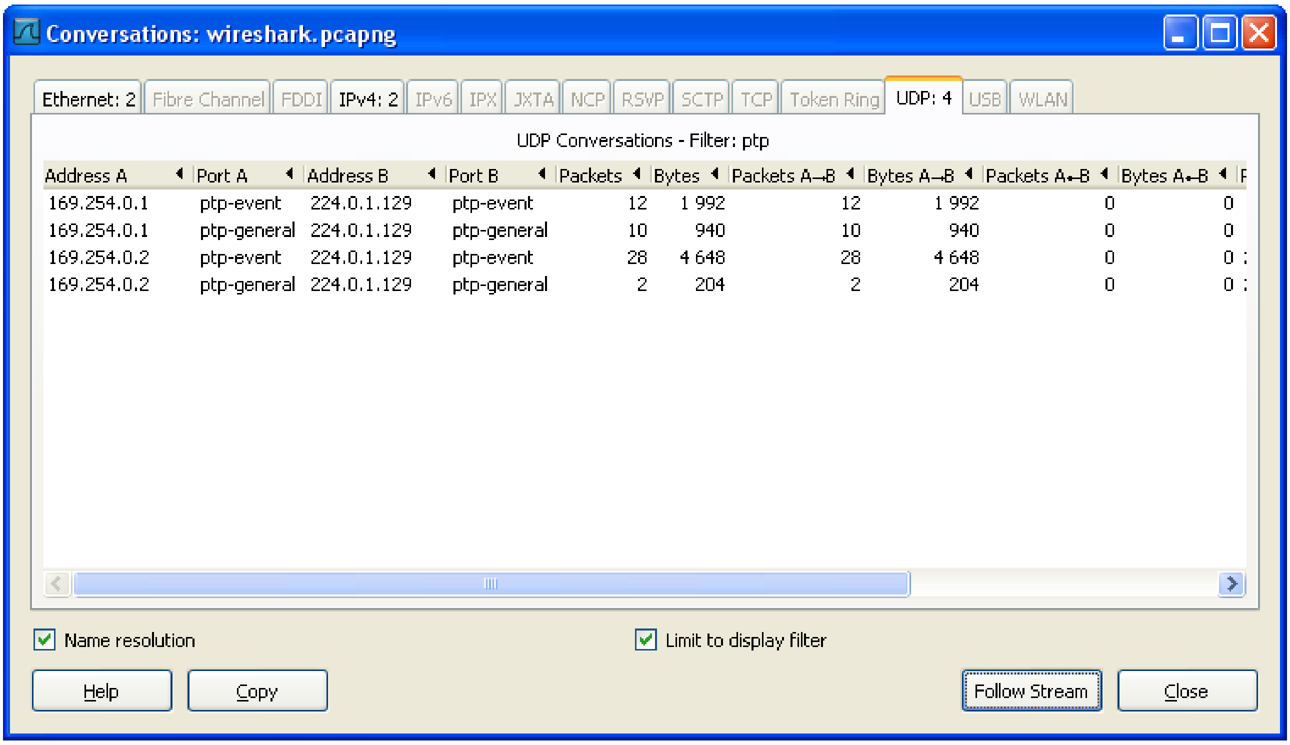
Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
⊟ Frame	100.00 %	52	100.00 %	7784	0.001	0	0	0.000
⊟ Ethernet	100.00 %	52	100.00 %	7784	0.001	0	0	0.000
⊟ Internet Protocol Version 4	100.00 %	52	100.00 %	7784	0.001	0	0	0.000
⊟ User Datagram Protocol	100.00 %	52	100.00 %	7784	0.001	0	0	0.000
Precision Time Protocol (IEEE1588)	100.00 %	52	100.00 %	7784	0.001	52	7784	0.001

Figure 33: Wireshark protocol hierarchy

5.3.1.5 Conversations

The following image shows the PTP conversations between the two nodes.

This is just an extension of the end point and it can be noticed the number of to and fro messages from each node and port.



UDP Conversations - Filter: ptp

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A→B	Bytes A→B	Packets A←B	Bytes A←B	F
169.254.0.1	ptp-event	224.0.1.129	ptp-event	12	1 992	12	1 992	0	0	
169.254.0.1	ptp-general	224.0.1.129	ptp-general	10	940	10	940	0	0	
169.254.0.2	ptp-event	224.0.1.129	ptp-event	28	4 648	28	4 648	0	0	:
169.254.0.2	ptp-general	224.0.1.129	ptp-general	2	204	2	204	0	0	:

☑ Name resolution

☑ Limit to display filter

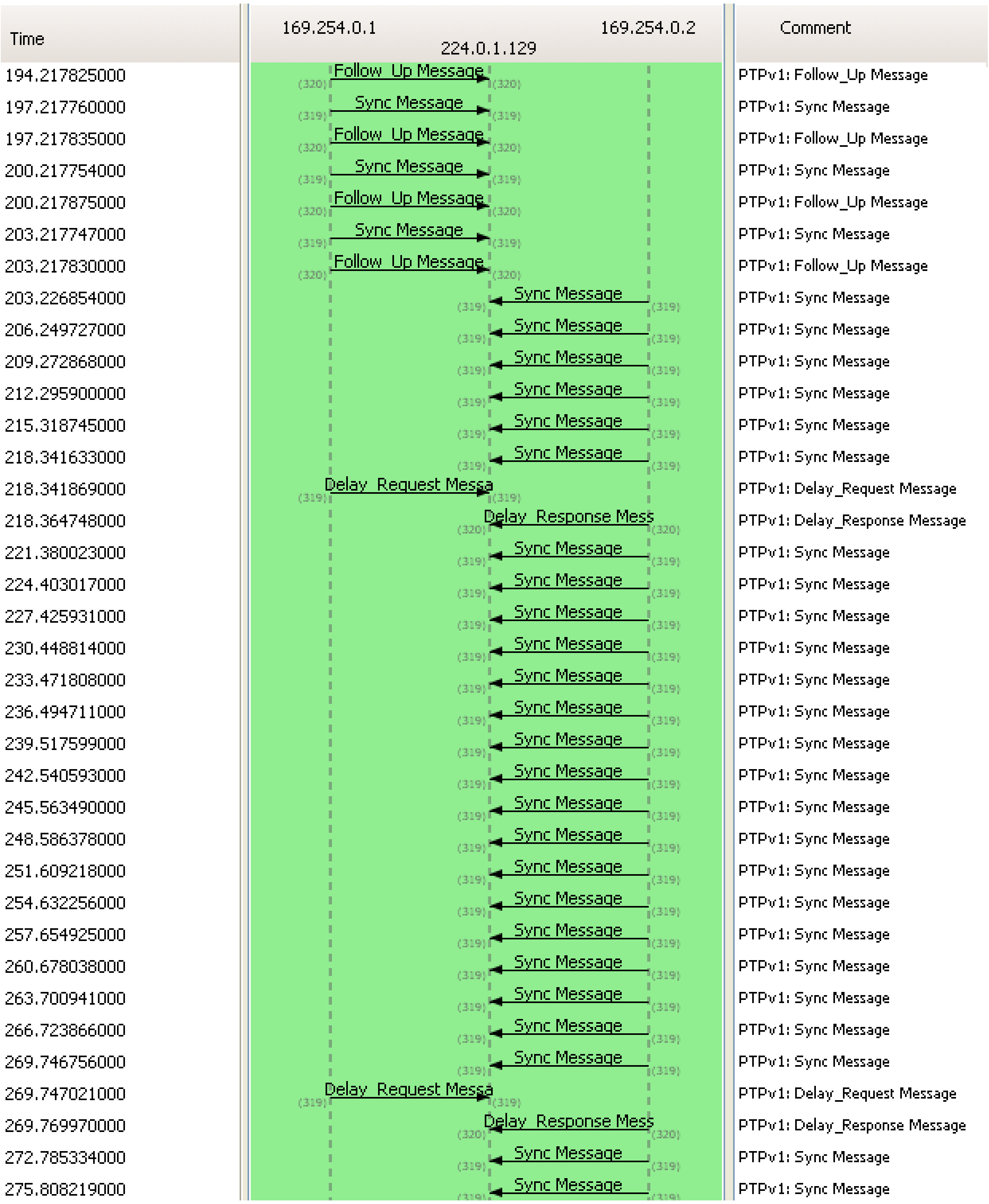
Figure 34: Wireshark PTP conversations

5.3.1.6 Flow

The following image describes the flow of packets over the Ethernet link.

The following details are noticeable. Since the PC was started first, it assumed itself to be the PTP_MASTER and began to send sync and follow up messages (since it does not have hardware timestamping, the PC performs a two-step sync routine).

Once the BF518 boots up and announces itself, the BMCA on both the nodes recognizes the BF518 as the better clock and the Blackfin assumes the PTP_MASTER role. It starts to send sync messages (1 step sync routine with hardware support). The PC now periodically requests for delay using the delay_req packets and continues to synchronize with the BF518.



Time	169.254.0.1 224.0.1.129 169.254.0.2	Comment
194.217825000	(320) Follow Up Message (320)	PTPv1: Follow_Up Message
197.217760000	(319) Sync Message (319)	PTPv1: Sync Message
197.217835000	(320) Follow Up Message (320)	PTPv1: Follow_Up Message
200.217754000	(319) Sync Message (319)	PTPv1: Sync Message
200.217875000	(320) Follow Up Message (320)	PTPv1: Follow_Up Message
203.217747000	(319) Sync Message (319)	PTPv1: Sync Message
203.217830000	(320) Follow Up Message (320)	PTPv1: Follow_Up Message
203.226854000	(319) Sync Message (319)	PTPv1: Sync Message
206.249727000	(319) Sync Message (319)	PTPv1: Sync Message
209.272868000	(319) Sync Message (319)	PTPv1: Sync Message
212.295900000	(319) Sync Message (319)	PTPv1: Sync Message
215.318745000	(319) Sync Message (319)	PTPv1: Sync Message
218.341633000	(319) Sync Message (319)	PTPv1: Sync Message
218.341869000	(319) Delay Request Messa (319)	PTPv1: Delay_Request Message
218.364748000	(320) Delay Response Mess (320)	PTPv1: Delay_Response Message
221.380023000	(319) Sync Message (319)	PTPv1: Sync Message
224.403017000	(319) Sync Message (319)	PTPv1: Sync Message
227.425931000	(319) Sync Message (319)	PTPv1: Sync Message
230.448814000	(319) Sync Message (319)	PTPv1: Sync Message
233.471808000	(319) Sync Message (319)	PTPv1: Sync Message
236.494711000	(319) Sync Message (319)	PTPv1: Sync Message
239.517599000	(319) Sync Message (319)	PTPv1: Sync Message
242.540593000	(319) Sync Message (319)	PTPv1: Sync Message
245.563490000	(319) Sync Message (319)	PTPv1: Sync Message
248.586378000	(319) Sync Message (319)	PTPv1: Sync Message
251.609218000	(319) Sync Message (319)	PTPv1: Sync Message
254.632256000	(319) Sync Message (319)	PTPv1: Sync Message
257.654925000	(319) Sync Message (319)	PTPv1: Sync Message
260.678038000	(319) Sync Message (319)	PTPv1: Sync Message
263.700941000	(319) Sync Message (319)	PTPv1: Sync Message
266.723866000	(319) Sync Message (319)	PTPv1: Sync Message
269.746756000	(319) Sync Message (319)	PTPv1: Sync Message
269.747021000	(319) Delay Request Messa (319)	PTPv1: Delay_Request Message
269.769970000	(320) Delay Response Mess (320)	PTPv1: Delay_Response Message
272.785334000	(319) Sync Message (319)	PTPv1: Sync Message
275.808219000	(319) Sync Message (319)	PTPv1: Sync Message

Figure 35: Wireshark flow

5.3.1.7 IO rate

The following figure describes the rate of packet transmission over the Ethernet link.

As can be seen, the initial values denote the dual packets sent by the PC (sync and follow up). The first spike corresponds to the consecutive sync message from both the PC and the Blackfin. The subsequent spikes correspond to the delay request and response.

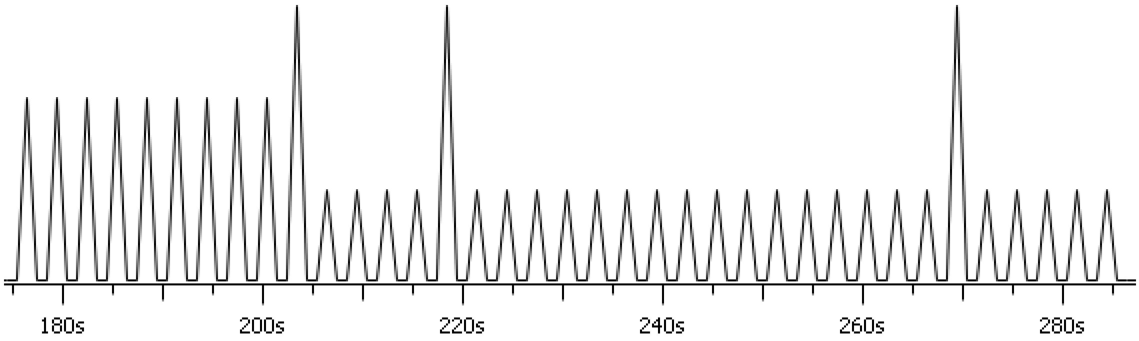


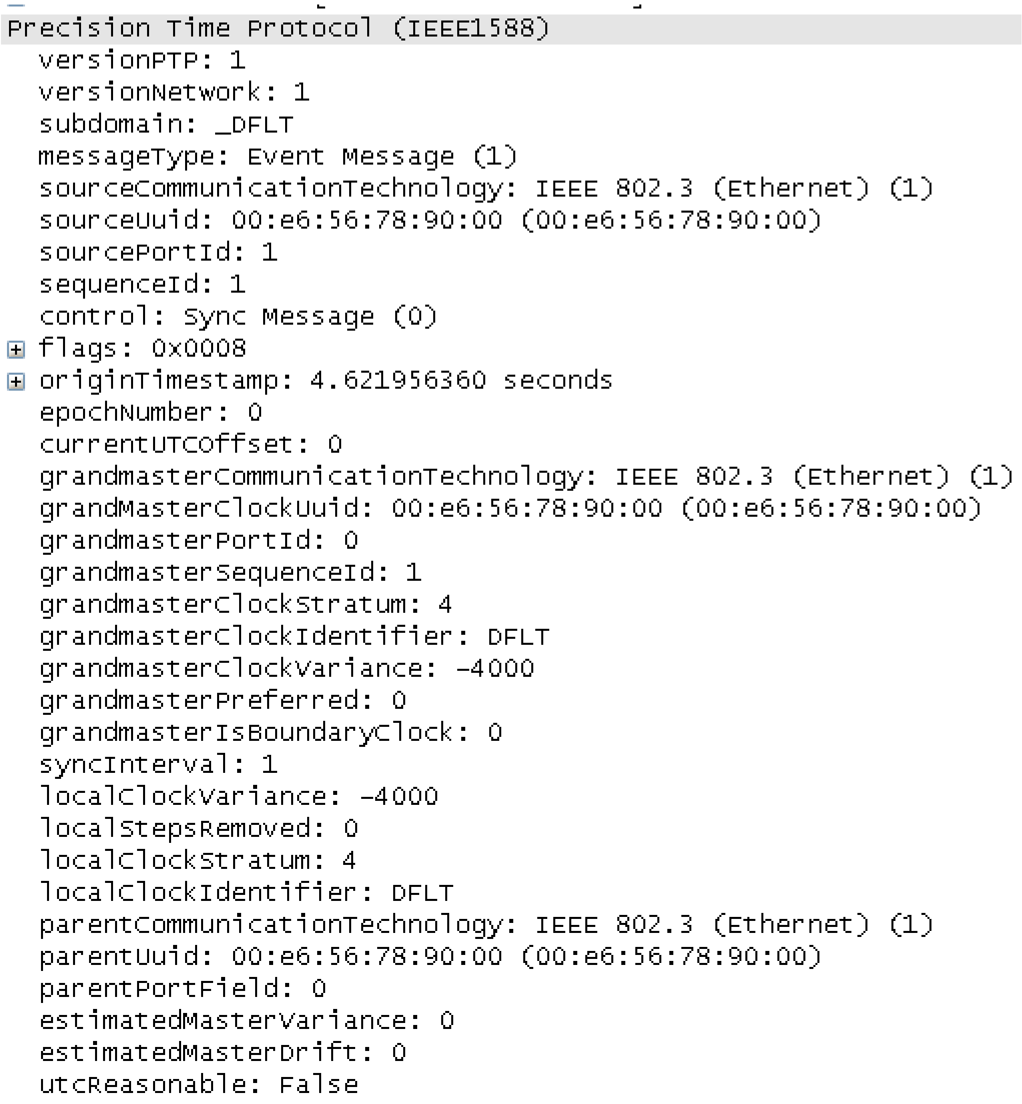
Figure 36: Wireshark IO rate

5.3.1.8 Sync message captured packet

The following figure is an extract from wireshark of the sync message packet being sent from Blackfin.

The following points can be noted on this packet.

- It has an origin Timestamp, hence the packet has been hardware time stamped complying with a 1 step sync routine.
- The grandMasterClockUuid matches the sourceUuid hence confirming that the Blackfin is the grand master in this PTP domain.



```
Precision Time Protocol (IEEE1588)
  versionPTP: 1
  versionNetwork: 1
  subdomain: _DFLT
  messageType: Event Message (1)
  sourceCommunicationTechnology: IEEE 802.3 (Ethernet) (1)
  sourceUuid: 00:e6:56:78:90:00 (00:e6:56:78:90:00)
  sourcePortId: 1
  sequenceId: 1
  control: Sync Message (0)
⊞ flags: 0x0008
⊞ originTimestamp: 4.621956360 seconds
  epochNumber: 0
  currentUTCoffset: 0
  grandmasterCommunicationTechnology: IEEE 802.3 (Ethernet) (1)
  grandMasterClockUuid: 00:e6:56:78:90:00 (00:e6:56:78:90:00)
  grandmasterPortId: 0
  grandmasterSequenceId: 1
  grandmasterClockStratum: 4
  grandmasterClockIdentifier: DFLT
  grandmasterClockVariance: -4000
  grandmasterPreferred: 0
  grandmasterIsBoundaryClock: 0
  syncInterval: 1
  localClockVariance: -4000
  localStepsRemoved: 0
  localClockStratum: 4
  localClockIdentifier: DFLT
  parentCommunicationTechnology: IEEE 802.3 (Ethernet) (1)
  parentUuid: 00:e6:56:78:90:00 (00:e6:56:78:90:00)
  parentPortField: 0
  estimatedMasterVariance: 0
  estimatedMasterDrift: 0
  utcReasonable: False
```

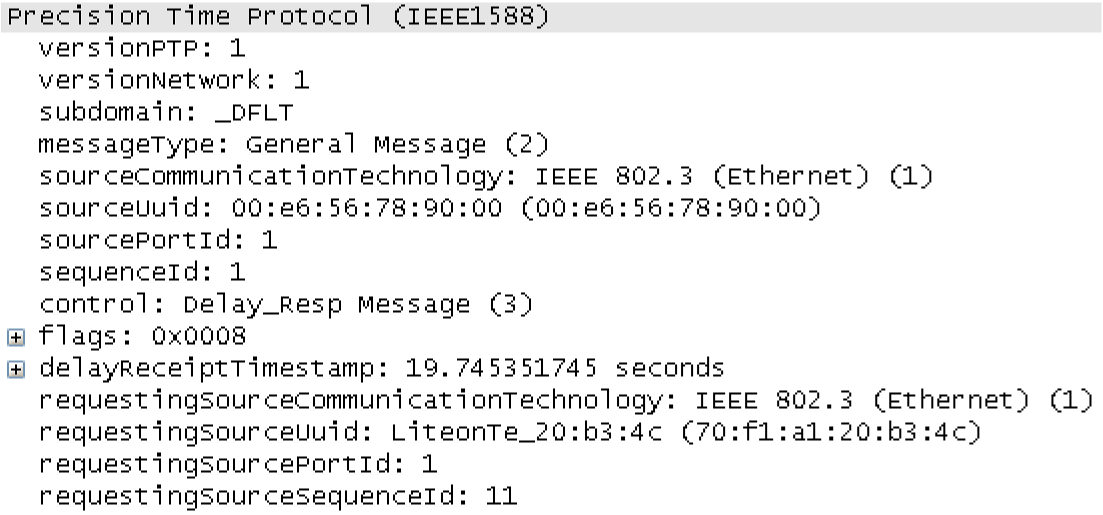
Figure 37: Wireshark sync message

5.3.1.9 Delay response captured packet

The following figure is an extract from wireshark of the delay response message packet being sent from Blackfin.

The following points can be noted on this packet.

- The source Uuid matches that of the previous message, thus confirming that this packet is originating from Blackfin.
- It contains the delayReceiptTimestamp with the appropriate timestamp.
- The requestingSourceUuid contains the PC's Ethernet interface with is a Liteon one.



```
Precision Time Protocol (IEEE1588)
  versionPTP: 1
  versionNetwork: 1
  subdomain: _DFLT
  messageType: General Message (2)
  sourceCommunicationTechnology: IEEE 802.3 (Ethernet) (1)
  sourceUuid: 00:e6:56:78:90:00 (00:e6:56:78:90:00)
  sourcePortId: 1
  sequenceId: 1
  control: Delay_Resp Message (3)
⊞ flags: 0x0008
⊞ delayReceiptTimestamp: 19.745351745 seconds
  requestingSourceCommunicationTechnology: IEEE 802.3 (Ethernet) (1)
  requestingSourceUuid: LiteonTe_20:b3:4c (70:f1:a1:20:b3:4c)
  requestingSourcePortId: 1
  requestingSourceSequenceId: 11
```

Figure 38: Wireshark delay response packet

5.3.2 Debug logs

The actual debug log from the two nodes can be found in the appendix. Commentary on the log can be found in **bold** next to each log entry.

5.3.3 Conclusions

From the wireshark analysis and the debug logs generated from both the nodes, it is concluded that the software is compliant to IEEE 1588.

6 Summary

Traditionally, the implementation of packet-switched network has been avoided for vehicle multimedia applications due to its nondeterministic nature. The recent work of the IEEE Audio Video Bridging (AVB) task group offers draft standards-based approach for highly consistent networked transmission for low latency applications like those found within an automobile.

An "Audio Video Bridging" network is one that implements a set of protocols being developed by the IEEE 802.1 Audio/Video Bridging Task Group. There are four primary updates between the proposed Audio Video Bridging architecture and existing 802 architectures:

1. Precise synchronization,
2. Traffic shaping for media streams,
3. Admission controls (or stream reservation), and
4. Identification of non-participating devices.

AVB nodes periodically synchronize their reference clocks very precisely by exchanging timing information across the links that connect them. This nano-precision synchronization allows a common time base across the various connected nodes and allows the transmission and reception of multiple synchronized media streams.

This project attempts to Design and Implement the precise time synchronization protocol (defined in IEEE Std. 1588) as a foundation for an automotive audio amplifier. The implementation is attempted on the Analog Devices® ADSP-BF518F Blackfin™ Processor. The ADSP-BF518F was chosen for its hardware capabilities in precisely timestamping Ethernet messages thus greatly boosting the precision of the time synchronization protocol.

This software is architected using the layered software architecture design pattern. The hardware elements lie at the bottom. The next layer is made up of device drivers that would configure the hardware. The next layer is the LWip (light weight TCP/IP) stack and on top of that rests the ptpd application. The ptpd application is a spin off from the open source ptpd implementation by Steven Kreuzer, George Neville-Neil [9]. The design borrows concepts from the original implementation but architects the system around it. It also adds the required hardware support to ensure the high precision requirements.

This design was then implemented and tested on the ADSP-BF518F processor. The test setup used for testing the ptp_BF518 software consists of the ADSP-BF518F and a generic PC executing the open source implementation of ptpd on the other side. The PC also runs wireshark to capture the network packets as they sent / received. The capture packets along with the analysis of the captured packets are presented here in this dissertation.

The analysis concludes that the implementation is compliant to the IEEE Std. 1588. This lays the foundation for a complete audio video bridging networked amplifier on the ADSP-BF518F with the implementation of the precise time stamping protocol. It also establishes the capability of the Analog Devices® ADSP-BF518F Blackfin™ Processor in the Ethernet AVB space.

7 Directions for future work

This is just the ice berg of the possibilities in the new world of real time media over the Ethernet. With this important corner stone laid, a complete media framework on top of this time synchronization can be established. The following would be the activities:

1) Implementing the IEEE Std.802.1Qat for stream reservation protocol (SRP)

2) Implementing the IEEE Std.802.1Qav for forwarding and queuing for time sensitive streams (FQTSS)

3) Implementing the IEEE Std.1722 for the transport stream protocol.

4) Implementing the IEC 61883 for compatibility with IEEE Std.1722.

5) Implementing a media manager over these protocols to manage multiple Ethernet AVB streams and synchronizing them.

6) Implementing a media clock recovery and a generation scheme for each of the media streams on the network.

With the above implementations, there would be a complete solution of Ethernet enabled multimedia communications that provide precise time synchronization and professional quality of service.

Currently, the following DSPs from Analog Devices® have hardware support for Ethernet AVB:

- ADSP-BF518
- ADSP-BF609
- ADZS-CM408F

8 Bibliography

Rick Kreifeldt. AVB for Automotive Use. AVnu Alliance, 2009

Michael D Johas Teener, Garner, Geoffrey M. Overview and timing performance of IEEE 802.1AS. Precision Clock Synchronization for Measurement, Control and Communication, 2008. ISPCS 2008. 22-26 Sept. 2008; Ann Arbor, MI.

Geoffrey M Garner. New simulation and test results for IEEE 802.1AS timing performance. Precision Clock Synchronization for Measurement, Control and Communication, 2009. ISPCS 2009. 12-16 Oct. 2009; Brescia.

Geoffrey M Garner. Synchronization of audio/video bridging networks using IEEE 802.1AS. Communications Magazine, IEEE Volume: 49, Issue: 2 (2011): 140 – 147

Hyung-Taek Lim. IEEE 802.1AS time synchronization in a switched Ethernet based in-car network. Vehicular Networking Conference (VNC), 2011 IEEE. 14-16 Nov. 2011; Amsterdam.

Jahanzaib Imtiaz. A performance study of Ethernet Audio Video Bridging (AVB) for Industrial real-time communication. Emerging Technologies & Factory Automation, 2009. ETFA 2009. 22-25 Sept. 2009; Mallorca.

Andrew S. Tanenbaum, David J. Wetherall. Computer Networks. Prentice Hall; 5 edition (October 7, 2010)

James Kurose, Keith Ross. Computer Networking: A Top-Down Approach (6th Edition). Pearson; 6th edition (March 5, 2012)

W Richard Stevens, Bill Fenner, Andrew M Rudoff. UNIX Network Programming, Volume 1: The Sockets Networking API (3rd Edition). Addison-Wesley Professional; 3 edition (November 24, 2003)

W Richard Stevens. UNIX Network Programming, Volume 2: Interprocess Communications, Second Edition. Prentice Hall; 2nd edition (September 4, 1998)

9 References

[1] – Michael Johas Teener et al. "No-excuses Audio/Video Networking: the Technology behind AVnu". AVnu Alliance, 2009.

[2] – "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems". IEEE Std 1588-2008. pp. i-269, 2008.

[3] – "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams". IEEE Std 802.1Qav-2009. pp. i-79, 2009.

[4] – "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks". IEEE Std 802.1AS-2011. pp. i-274, 2011.

[5] – "IEEE Standard for Local and Metropolitan Area Networks---Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)". IEEE Std 802.1Qat-2010. pp. i-103, 2010.

[6] - Brahim Bensaou, KT Chan, HK Danny Tsang. Credit-based fair queuing (CBFQ): a simple and feasible scheduling algorithm for packet networks. IEEE ATM Workshop 1997. Proceedings. 25-28 May 1997; Lisboa, 1997.

[7] – Analog Devices, Inc. ADSP-BF51x Blackfin® Processor Hardware Reference. USA. Analog Devices, Inc., 2010.

[8] - Barry Rubel. "Patterns for Generating a Layered Architecture". Pattern languages of program design (1995): 119-128.

[9] – Steven Kreuzer, George Neville-Neil. http://sourceforge.net/p/ptpd/wiki/Home/.

[10] - Analog Devices, Inc. VisualDSP++ 5.0 Device Drivers and System Services Manual for Blackfin Processors. USA. Analog Devices, Inc., 2007.

[11] - Kaushal Sanghi. "Building Complex VDK/LwIP Applications Using Blackfin® Processors". Analog Devices® Engineer-To-Engineer Note. September 16, 2008.

Appendix A:ADSP-BF518 debug log

```
(ptpd debug) allocated 1088 bytes for protocol engine data
(ptpd debug) allocated 600 bytes for foreign master data
(ptpd debug) event POWERUP
(ptpd debug) state PTP_INITIALIZING  - BF518 has started initializing itself
(ptpd debug) manufacturerIdentity: ADSP BF518
(ptpd debug) netInit
(ptpd debug) Unicast address: 169.254.0.2
(ptpd debug) initData
(ptpd debug) initTimer
(ptpd debug) initClock
(ptpd debug) sync message interval: 2
(ptpd debug) clock identifier: DFLT
(ptpd debug) 256*log2(clock variance): -4000
(ptpd debug) clock stratum: 4
(ptpd debug) clock preferred?: no
(ptpd debug) bound interface name: National DP83848
(ptpd debug) communication technology: 1
(ptpd debug) uuid: 00:e6:56:78:90:00
(ptpd debug) PTP subdomain name: _DFLT
(ptpd debug) subdomain address: 224.0.1.129
(ptpd debug) event port address: 3f 1
(ptpd debug) general port address: 40 1
(ptpd debug) state PTP_LISTENING     - BF518 has completed initializing and is listening for packets
(ptpd debug) updateForeign: new record (0,1) 1 1 70:f1:a1:20:b3:4c
(ptpd debug) state PTP_MASTER        - Receives a packet from the PC. Finds itself to be better
master via BMCA algorithm (since it has hardware ptp) and becomes PTP_MASTER.
BF518 now begins to transmit sync messages to the PC periodically with its internal PTP time being
displayed in seconds and nano seconds.
(ptpd debug) fromInternalTime:          4s   621956360ns ->          4s   621956360ns
(ptpd debug) fromInternalTime:          4s   621956360ns ->          4s   621956360ns
(ptpd debug) fromInternalTime:          4s   621956360ns ->          4s   621956360ns
(ptpd debug) fromInternalTime:          7s   645079440ns ->          7s   645079440ns
(ptpd debug) fromInternalTime:          7s   645079440ns ->          7s   645079440ns
(ptpd debug) fromInternalTime:          7s   645079440ns ->          7s   645079440ns
(ptpd debug) fromInternalTime:         10s   667979660ns ->         10s   667979660ns
(ptpd debug) fromInternalTime:         10s   667979660ns ->         10s   667979660ns
(ptpd debug) fromInternalTime:         10s   667979660ns ->         10s   667979660ns
(ptpd debug) fromInternalTime:         13s   690979520ns ->         13s   690979520ns
(ptpd debug) fromInternalTime:         13s   690979520ns ->         13s   690979520ns
(ptpd debug) fromInternalTime:         13s   690979520ns ->         13s   690979520ns
(ptpd debug) fromInternalTime:         16s   713879520ns ->         16s   713879520ns
(ptpd debug) fromInternalTime:         16s   713879520ns ->         16s   713879520ns
(ptpd debug) fromInternalTime:         16s   713879520ns ->         16s   713879520ns
(ptpd debug) fromInternalTime:         19s   736779700ns ->         19s   736779700ns
(ptpd debug) fromInternalTime:         19s   736779700ns ->         19s   736779700ns
(ptpd debug) fromInternalTime:         19s   736779700ns ->         19s   736779700ns
(ptpd debug) fromInternalTime:         19s   745351745ns ->         19s   745351745ns
(ptpd debug) fromInternalTime:         19s   745351745ns ->         19s   745351745ns
(ptpd debug) fromInternalTime:         22s   775179680ns ->         22s   775179680ns
(ptpd debug) fromInternalTime:         22s   775179680ns ->         22s   775179680ns
(ptpd debug) fromInternalTime:         22s   775179680ns ->         22s   775179680ns
(ptpd debug) fromInternalTime:         25s   798179580ns ->         25s   798179580ns
(ptpd debug) fromInternalTime:         25s   798179580ns ->         25s   798179580ns
(ptpd debug) fromInternalTime:         25s   798179580ns ->         25s   798179580ns
(ptpd debug) fromInternalTime:         28s   821079760ns ->         28s   821079760ns
(ptpd debug) fromInternalTime:         28s   821079760ns ->         28s   821079760ns
(ptpd debug) fromInternalTime:         28s   821079760ns ->         28s   821079760ns
(ptpd debug) fromInternalTime:         31s   843979560ns ->         31s   843979560ns
(ptpd debug) fromInternalTime:         31s   843979560ns ->         31s   843979560ns
(ptpd debug) fromInternalTime:         31s   843979560ns ->         31s   843979560ns
(ptpd debug) fromInternalTime:         34s   866979660ns ->         34s   866979660ns
(ptpd debug) fromInternalTime:         34s   866979660ns ->         34s   866979660ns
(ptpd debug) fromInternalTime:         34s   866979660ns ->         34s   866979660ns
(ptpd debug) fromInternalTime:         37s   889879640ns ->         37s   889879640ns
(ptpd debug) fromInternalTime:         37s   889879640ns ->         37s   889879640ns
(ptpd debug) fromInternalTime:         37s   889879640ns ->         37s   889879640ns
(ptpd debug) fromInternalTime:         40s   912779440ns ->         40s   912779440ns
(ptpd debug) fromInternalTime:         40s   912779440ns ->         40s   912779440ns
(ptpd debug) fromInternalTime:         40s   912779440ns ->         40s   912779440ns
(ptpd debug) fromInternalTime:         43s   935779640ns ->         43s   935779640ns
(ptpd debug) fromInternalTime:         43s   935779640ns ->         43s   935779640ns
```

```
(ptpd debug) fromInternalTime:         43s   935779640ns ->         43s   935779640ns
(ptpd debug) fromInternalTime:         46s   958679620ns ->         46s   958679620ns
(ptpd debug) fromInternalTime:         46s   958679620ns ->         46s   958679620ns
(ptpd debug) fromInternalTime:         46s   958679620ns ->         46s   958679620ns
(ptpd debug) fromInternalTime:         49s   981579680ns ->         49s   981579680ns
(ptpd debug) fromInternalTime:         49s   981579680ns ->         49s   981579680ns
(ptpd debug) fromInternalTime:         49s   981579680ns ->         49s   981579680ns
(ptpd debug) fromInternalTime:         53s     4579520ns ->         53s     4579520ns
(ptpd debug) fromInternalTime:         53s     4579520ns ->         53s     4579520ns
(ptpd debug) fromInternalTime:         53s     4579520ns ->         53s     4579520ns
(ptpd debug) fromInternalTime:         56s    27479680ns ->         56s    27479680ns
(ptpd debug) fromInternalTime:         56s    27479680ns ->         56s    27479680ns
(ptpd debug) fromInternalTime:         56s    27479680ns ->         56s    27479680ns
(ptpd debug) fromInternalTime:         59s    50379700ns ->         59s    50379700ns
(ptpd debug) fromInternalTime:         59s    50379700ns ->         59s    50379700ns
(ptpd debug) fromInternalTime:         59s    50379700ns ->         59s    50379700ns
(ptpd debug) fromInternalTime:         62s    73279520ns ->         62s    73279520ns
(ptpd debug) fromInternalTime:         62s    73279520ns ->         62s    73279520ns
(ptpd debug) fromInternalTime:         62s    73279520ns ->         62s    73279520ns
(ptpd debug) fromInternalTime:         65s    96279640ns ->         65s    96279640ns
(ptpd debug) fromInternalTime:         65s    96279640ns ->         65s    96279640ns
(ptpd debug) fromInternalTime:         65s    96279640ns ->         65s    96279640ns
(ptpd debug) fromInternalTime:         68s   119179700ns ->         68s   119179700ns
(ptpd debug) fromInternalTime:         68s   119179700ns ->         68s   119179700ns
(ptpd debug) fromInternalTime:         68s   119179700ns ->         68s   119179700ns
(ptpd debug) fromInternalTime:         71s   142079500ns ->         71s   142079500ns
(ptpd debug) fromInternalTime:         71s   142079500ns ->         71s   142079500ns
(ptpd debug) fromInternalTime:         71s   142079500ns ->         71s   142079500ns
(ptpd debug) fromInternalTime:         71s   150615745ns ->         71s   150615745ns
(ptpd debug) fromInternalTime:         71s   150615745ns ->         71s   150615745ns
(ptpd debug) fromInternalTime:         74s   180579820ns ->         74s   180579820ns
(ptpd debug) fromInternalTime:         74s   180579820ns ->         74s   180579820ns
(ptpd debug) fromInternalTime:         74s   180579820ns ->         74s   180579820ns
(ptpd debug) fromInternalTime:         77s   203479380ns ->         77s   203479380ns
(ptpd debug) fromInternalTime:         77s   203479380ns ->         77s   203479380ns
(ptpd debug) fromInternalTime:         77s   203479380ns ->         77s   203479380ns
(ptpd debug) fromInternalTime:         80s   226379660ns ->         80s   226379660ns
(ptpd debug) fromInternalTime:         80s   226379660ns ->         80s   226379660ns
(ptpd debug) fromInternalTime:         80s   226379660ns ->         80s   226379660ns
(ptpd debug) fromInternalTime:         83s   249379500ns ->         83s   249379500ns
(ptpd debug) fromInternalTime:         83s   249379500ns ->         83s   249379500ns
(ptpd debug) fromInternalTime:         83s   249379500ns ->         83s   249379500ns
(ptpd debug) fromInternalTime:         86s   272279400ns ->         86s   272279400ns
(ptpd debug) fromInternalTime:         86s   272279400ns ->         86s   272279400ns
(ptpd debug) fromInternalTime:         86s   272279400ns ->         86s   272279400ns
```

Appendix B:PC debug log

```
allocated 1088 bytes for protocol engine data
allocated 600 bytes for foreign master data
event POWERUP
state PTP_INITIALIZING        - PC has powered up and begun to initialize.
manufacturerIdentity: Analog Devices
netInit
initData
initTimer
initClock
adj:               0s
sync message interval: 2
clock identifier: DFLT
256*log2(clock variance): -4000
clock stratum: 4
clock preferred?: no
bound interface name: Dell Wireless 13
communication technology: 1
uuid: 70:f1:a1:20:b3:4c
PTP subdomain name: _DFLT
subdomain address: 224.0.1.129
event port address: 3f 1
general port address: 40 1
state PTP_LISTENING           - PC moves to state listening after initialization
timerStart: set timer 0 to 20
activity
netSelect returns 0
handle: nothing
timerUpdate: timer 0 expired
event SYNC_RECEIPT_TIMEOUT_EXPIRES - PC does not receive any master announcements, hence becomes
PTP_MASTER
state PTP_MASTER
timerStart: set timer 1 to 2
timerUpdate: timer 1 expired
event SYNC_INTERVAL_TIMEOUT_EXPIRES - PC begins to send sync message (BF518 is still booting)
fromInternalTime: 1363946436s    15625000ns -> 1363946436s    15625000ns
sent sync message
fromInternalTime: 1363946436s    16055000ns -> 1363946436s    16055000ns
sent followup message
event SYNC_INTERVAL_TIMEOUT_EXPIRES
fromInternalTime: 1363946439s    15625000ns -> 1363946439s    15625000ns
sent sync message
fromInternalTime: 1363946439s    16055000ns -> 1363946439s    16055000ns
sent followup message
event SYNC_INTERVAL_TIMEOUT_EXPIRES
fromInternalTime: 1363946442s    15625000ns -> 1363946442s    15625000ns
sent sync message
fromInternalTime: 1363946442s    16055000ns -> 1363946442s    16055000ns
sent followup message
fromInternalTime: 1363946445s    15625000ns -> 1363946445s    15625000ns
sent sync message
fromInternalTime: 1363946445s    16055000ns -> 1363946445s    16055000ns
sent followup message
event SYNC_INTERVAL_TIMEOUT_EXPIRES
fromInternalTime: 1363946448s    15625000ns -> 1363946448s    15625000ns
sent sync message
fromInternalTime: 1363946448s    16055000ns -> 1363946448s    16055000ns
sent followup message
event SYNC_INTERVAL_TIMEOUT_EXPIRES
fromInternalTime: 1363946451s    15625000ns -> 1363946451s    15625000ns
sent sync message
fromInternalTime: 1363946451s    16055000ns -> 1363946451s    16055000ns
sent followup message
event SYNC_INTERVAL_TIMEOUT_EXPIRES
fromInternalTime: 1363946454s    15625000ns -> 1363946454s    15625000ns
sent sync message
fromInternalTime: 1363946454s    16055000ns -> 1363946454s    16055000ns
sent followup message
event SYNC_INTERVAL_TIMEOUT_EXPIRES
fromInternalTime: 1363946457s    15625000ns -> 1363946457s    15625000ns
sent sync message
fromInternalTime: 1363946457s    16055000ns -> 1363946457s    16055000ns
```

```
sent followup message
event SYNC_INTERVAL_TIMEOUT_EXPIRES
fromInternalTime: 1363946460s    15625000ns -> 1363946460s    15625000ns
sent sync message
fromInternalTime: 1363946460s    16055000ns -> 1363946460s    16055000ns
sent followup message
event SYNC_INTERVAL_TIMEOUT_EXPIRES
fromInternalTime: 1363946463s    15625000ns -> 1363946463s    15625000ns
sent sync message
fromInternalTime: 1363946463s    16055000ns -> 1363946463s    16055000ns
sent followup message
updateForeign - BF518 has booted up, notices that PC has inferior clock and announces itself
bmc: best record 0
bmcDataSetComparison: start
state PTP_SLAVE       - PC realizes that BF518 has better clock and becomes PTP_SLAVE
PC begins to receive clock from the BF518 PTP master and starts to adjust its own clock to it.
initClock
adj:               0s
handleSync: looking for uuid 00:e6:56:78:90:00
msgUnpackSync: originTimestamp.seconds 7
msgUnpackSync: originTimestamp.nanoseconds 645079440
msgUnpackSync: originTimestamp.seconds 10
msgUnpackSync: originTimestamp.nanoseconds 667979660
msgUnpackSync: originTimestamp.seconds 13
msgUnpackSync: originTimestamp.nanoseconds 690979520
msgUnpackSync: originTimestamp.seconds 16
msgUnpackSync: originTimestamp.nanoseconds 713879520
msgUnpackSync: originTimestamp.seconds 19
msgUnpackSync: originTimestamp.nanoseconds 736779700
msgUnpackDelayResp: delayReceiptTimestamp.seconds 19
msgUnpackDelayResp: delayReceiptTimestamp.nanoseconds 745351745
toInternalTime:         19s   745351745ns <-         19s   745351745ns
msgUnpackSync: originTimestamp.seconds 22
msgUnpackSync: originTimestamp.nanoseconds 775179680
msgUnpackSync: originTimestamp.seconds 25
msgUnpackSync: originTimestamp.nanoseconds 798179580
msgUnpackSync: originTimestamp.seconds 28
msgUnpackSync: originTimestamp.nanoseconds 821079760
msgUnpackSync: originTimestamp.seconds 31
msgUnpackSync: originTimestamp.nanoseconds 843979560
msgUnpackSync: originTimestamp.seconds 34
msgUnpackSync: originTimestamp.nanoseconds 866979660
msgUnpackSync: originTimestamp.seconds 37
msgUnpackSync: originTimestamp.nanoseconds 889879640
msgUnpackSync: originTimestamp.seconds 40
msgUnpackSync: originTimestamp.nanoseconds 912779440
msgUnpackSync: originTimestamp.seconds 43
msgUnpackSync: originTimestamp.nanoseconds 935779640
msgUnpackSync: originTimestamp.seconds 46
msgUnpackSync: originTimestamp.nanoseconds 958679620
msgUnpackSync: originTimestamp.seconds 53
msgUnpackSync: originTimestamp.nanoseconds 4579520
msgUnpackSync: originTimestamp.seconds 56
msgUnpackSync: originTimestamp.nanoseconds 27479680
msgUnpackSync: originTimestamp.seconds 59
msgUnpackSync: originTimestamp.nanoseconds 50379700
msgUnpackSync: originTimestamp.seconds 62
msgUnpackSync: originTimestamp.nanoseconds 73279520
msgUnpackSync: originTimestamp.seconds 65
msgUnpackSync: originTimestamp.nanoseconds 96279640
msgUnpackSync: originTimestamp.seconds 68
msgUnpackSync: originTimestamp.nanoseconds 119179700
msgUnpackSync: originTimestamp.seconds 71
msgUnpackSync: originTimestamp.nanoseconds 142079500
```

Checklist of items for the Final Dissertation Report

This checklist is to be attached as the last page of the report.

This checklist is to be duly completed, verified and signed by the student.

1.	**Is the final report properly hard bound? (Spiral bound or Soft bound or Perfect bound reports are not acceptable.)**	[✓ Yes] / No
2.	Is the Cover page in proper format as given in Annexure A?	[✓ Yes] / No
3.	Is the Title page (Inner cover page) in proper format?	[✓ Yes] / No
4.	(a) Is the Certificate from the Supervisor in proper format? (b) Has it been signed by the Supervisor?	[✓ Yes] / No [✓ Yes] / No
5.	Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly?	[✓ Yes] / No [✓ Yes] / No
6.	Is the title of your report appropriate? **The title should be adequately descriptive, precise and must reflect scope of the actual work done.**	[✓ Yes] / No
7.	Have you included the List of abbreviations / Acronyms? Uncommon abbreviations / Acronyms should not be used in the title.	[✓ Yes] / No
8.	Does the Report contain a summary of the literature survey?	[✓ Yes] / No
9.	Does the Table of Contents include page numbers? (i). Are the Pages numbered properly? (Ch. 1 should start on Page # 1) (ii). Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) (iii). Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) (iv). Are the Captions for the Figures and Tables proper? (v). Are the Appendices numbered properly? Are their titles appropriate	 [✓ Yes] / No [✓ Yes] / No [✓ Yes] / No [✓ Yes] / No [✓ Yes] / No
10.	Is the conclusion of the Report based on discussion of the work?	[✓ Yes] / No
11.	Are References or Bibliography given at the end of the Report? Have the References been cited properly inside the text of the Report? Is the citation of References in proper format?	[✓ Yes] / No [✓ Yes] / No [✓ Yes] / No
12.	Is the report format and content according to the guidelines? The report should not be a mere printout of a Power Point Presentation, or a user manual. Source code of software need not be included in the report.	[✓ Yes] / No

Declaration by Student:

I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

Place: Bangalore

Signature of the Student

Name: Mohan Karthik

Date: 28th March 2013