**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.

|  |  |
| --- | --- |
|  |  |
| Place : Bangalore  Date : | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Vijaykumar N, Project Manager, Analog Devices India Pvt. Ltd., Bangalore |

**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, we port the open source implementation of the IEEE 1588 (which is the parent standard of IEEE 802.1AS) 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 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**

[1 Introduction 1](#_Toc352258196)

[1.1 Overview of media streaming 1](#_Toc352258197)

[1.1.1 Historical background 1](#_Toc352258198)

[1.1.2 Requirements for A/V streaming 2](#_Toc352258199)

[1.1.3 The problems with existing approaches 2](#_Toc352258200)

[1.2 The Audio and Video Bridging Standard 3](#_Toc352258201)

[1.2.1 Precise Time Synchronization 4](#_Toc352258202)

[1.2.1.1 Systems 5](#_Toc352258203)

[1.2.1.2 Architecture 5](#_Toc352258204)

[1.2.1.3 Message Classes 6](#_Toc352258205)

[1.2.1.4 Clock Synchronization Model 7](#_Toc352258206)

[1.2.1.4.1 Establishing the master-slave hierarchy 7](#_Toc352258207)

[1.2.1.4.2 Synchronizing the clocks 8](#_Toc352258208)

[1.2.1.5 Generation of message timestamps 9](#_Toc352258209)

[1.2.2 Traffic shaping for AV streams 10](#_Toc352258210)

[1.2.3 Stream Reservation or Admission controls 11](#_Toc352258211)

[1.2.4 Benefits for Automotive OEMs 12](#_Toc352258212)

[1.3 Analog Devices® ADSP-BF518F Blackfin™ Processor 13](#_Toc352258213)

[1.4 Precision Time Protocol daemon 13](#_Toc352258214)

[2 Project Plan 15](#_Toc352258215)

[2.1 Requirements 15](#_Toc352258216)

[2.1.1 Time Sync protocol requirements 15](#_Toc352258217)

[2.1.2 General Requirements 15](#_Toc352258218)

[2.1.3 Ethernet MAC requirements 16](#_Toc352258219)

[2.2 Scope 16](#_Toc352258220)

[2.3 Tools and Platforms 16](#_Toc352258221)

[2.4 Project Schedule 17](#_Toc352258222)

[3 Architectural Design 18](#_Toc352258223)

[3.1 <<driver>> emac 18](#_Toc352258224)

[3.2 <<driver>> BF Helper 19](#_Toc352258225)

[3.3 <<driver>> Device drivers 19](#_Toc352258226)

[3.4 <<stack>> lwIP 19](#_Toc352258227)

[3.5 <<application>> ptpd 19](#_Toc352258228)

[3.6 <<application>> init 20](#_Toc352258229)

[3.7 Behavioural Model 20](#_Toc352258230)

[4 Mechanistic Design 21](#_Toc352258231)

[4.1 <<driver>> BF Helper 21](#_Toc352258232)

[4.1.1 Attributes 21](#_Toc352258233)

[4.1.1.1 <<define>>PTP\_FREQUENCY 21](#_Toc352258234)

[4.1.1.2 <<defines>>PTPEVT\_\* 22](#_Toc352258235)

[4.1.2 Operations 22](#_Toc352258236)

[4.1.3 nanosec\_to\_pulse() 22](#_Toc352258237)

[4.1.4 pulse\_to\_nanosec() 22](#_Toc352258238)

[4.1.5 Init\_BF\_PTP\_Timer() 22](#_Toc352258239)

[4.1.6 Gettimeofday() 22](#_Toc352258240)

[4.1.7 settimeofday() 22](#_Toc352258241)

[4.1.8 adjtimex() 22](#_Toc352258242)

[4.1.9 rand\_r() 23](#_Toc352258243)

[4.1.10 getRxStamp () 23](#_Toc352258244)

[4.1.11 getTxStamp 24](#_Toc352258245)

[4.1.12 checkMsgStamp 24](#_Toc352258246)

[4.2 <<application>> Ptpd 25](#_Toc352258247)

[4.2.1 Behavioural model 26](#_Toc352258248)

[4.2.1.1 Initialization 26](#_Toc352258249)

[4.2.1.2 State machine 27](#_Toc352258250)

[4.2.1.3 Event handling 29](#_Toc352258251)

[4.2.2 bmca (Best Master Clock Algorithm) 29](#_Toc352258252)

[4.2.3 protocol (Protocol implementation) 30](#_Toc352258253)

[4.2.3.1 Operations 30](#_Toc352258254)

[4.2.3.1.1 protocol() 30](#_Toc352258255)

[4.2.3.1.2 handleSync() 31](#_Toc352258256)

[4.2.3.1.3 handleDelayReq() 32](#_Toc352258257)

[4.2.3.1.4 handleDelayResp () 32](#_Toc352258258)

[4.2.4 msg (Message handling) 32](#_Toc352258259)

[4.2.5 net (Net services) 33](#_Toc352258260)

[4.2.5.1 Attributes 33](#_Toc352258261)

[4.2.5.1.1 Hwaddr 33](#_Toc352258262)

[4.2.5.1.2 myIP 33](#_Toc352258263)

[4.2.5.2 Operations 34](#_Toc352258264)

[4.2.5.2.1 lookupCommunicationTechnology() 34](#_Toc352258265)

[4.2.5.2.2 findIface() 34](#_Toc352258266)

[4.2.5.2.3 netInit() 34](#_Toc352258267)

[4.2.5.2.4 netRecvEvent() 35](#_Toc352258268)

[4.2.6 servo (Clock syntonization functions) 36](#_Toc352258269)

[4.2.7 sys (utility functions) 37](#_Toc352258270)

[4.2.7.1 Operations 37](#_Toc352258271)

[4.2.8 timer (Timer functions) 38](#_Toc352258272)

[4.2.8.1 Operations 38](#_Toc352258273)

[4.2.8.1.1 initTimer() 38](#_Toc352258274)

[5 Testing 39](#_Toc352258275)

[5.1 Test Setup 39](#_Toc352258276)

[5.2 Test Sequence 39](#_Toc352258277)

[5.3 Test Results 41](#_Toc352258278)

[5.3.1.1 Wireshark reports 41](#_Toc352258279)

[5.3.1.2 Captured packets 41](#_Toc352258280)

[5.3.1.3 Endpoints 42](#_Toc352258281)

[5.3.1.4 Protocol Hierarchy 43](#_Toc352258282)

[5.3.1.5 Conversations 44](#_Toc352258283)

[5.3.1.6 Flow 44](#_Toc352258284)

[5.3.1.7 IO rate 46](#_Toc352258285)

[5.3.1.8 Sync message captured packet 47](#_Toc352258286)

[5.3.1.9 Delay response captured packet 48](#_Toc352258287)

[5.3.2 Debug logs 48](#_Toc352258288)

[5.3.3 Conclusions 48](#_Toc352258289)

[6 Summary 49](#_Toc352258290)

[7 Directions for future work 51](#_Toc352258291)

[8 Bibliography 52](#_Toc352258292)

[9 References 53](#_Toc352258293)

[Appendix A: ADSP-BF518 debug log 54](#_Toc352258294)

[Appendix B: PC debug log 56](#_Toc352258295)

List of Figures

[Figure 1: AVB connections [1] 4](#_Toc352258296)

[Figure 2: Simple master−slave clock hierarchy [2] 6](#_Toc352258297)

[Figure 3: Basic synchronization message exchange [2] 8](#_Toc352258298)

[Figure 4: Timestamp generation model [2] 9](#_Toc352258299)

[Figure 5: Example Qav traffic shaping [1] 10](#_Toc352258300)

[Figure 6: Successful reservation (talker advertise) [1] 11](#_Toc352258301)

[Figure 7: ADSP-BF518F Block Diagram [7] 13](#_Toc352258302)

[Figure 8: Software Architecture 18](#_Toc352258303)

[Figure 9: High level software sequence 20](#_Toc352258304)

[Figure 10: BF Helper list of attributes 21](#_Toc352258305)

[Figure 11: BF Helper list of operations 21](#_Toc352258306)

[Figure 12: getRxStamp activity 23](#_Toc352258307)

[Figure 13: getTxStamp activity 24](#_Toc352258308)

[Figure 14: ptpd class diagram 25](#_Toc352258309)

[Figure 15: ptpd initialization 26](#_Toc352258310)

[Figure 16: ptpd state machine 28](#_Toc352258311)

[Figure 17: ptpd event handling 29](#_Toc352258312)

[Figure 18: ptpd::bmca operations 30](#_Toc352258313)

[Figure 19: ptpd::protocol operations 30](#_Toc352258314)

[Figure 20: ptpd::protocol::protocol() sequence 31](#_Toc352258315)

[Figure 21: ptpd::msg handling operations 33](#_Toc352258316)

[Figure 22: ptpd::net attributes 33](#_Toc352258317)

[Figure 23: ptpd::net operations 34](#_Toc352258318)

[Figure 24: ptpd::net::netinit() flow 35](#_Toc352258319)

[Figure 25: net::ptpd::netRecvEvent() Sequence 36](#_Toc352258320)

[Figure 26: ptpd::servo operations 36](#_Toc352258321)

[Figure 27: ptpd::sys operations 37](#_Toc352258322)

[Figure 28: ptpd::sys sequence 37](#_Toc352258323)

[Figure 29: ptpd::timer operations 38](#_Toc352258324)

[Figure 30: Test Setup 39](#_Toc352258325)

[Figure 31: Wireshark captured packets 41](#_Toc352258326)

[Figure 32: Wireshark endpoints 42](#_Toc352258327)

[Figure 33: Wireshark protocol hierarchy 43](#_Toc352258328)

[Figure 34: Wireshark PTP conversations 44](#_Toc352258329)

[Figure 35: Wireshark flow 45](#_Toc352258330)

[Figure 36: Wireshark IO rate 46](#_Toc352258331)

[Figure 37: Wireshark sync message 47](#_Toc352258332)

[Figure 38: Wireshark delay response packet 48](#_Toc352258333)

List of Tables

[Table 1: Tools used in Project 16](#_Toc352258334)

[Table 2: Project Schedule 17](#_Toc352258335)

[Table 3: PTP states [2] 27](#_Toc352258336)

# 

# Introduction

## Overview of media streaming

### 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.

### 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.

### 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.

## 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.

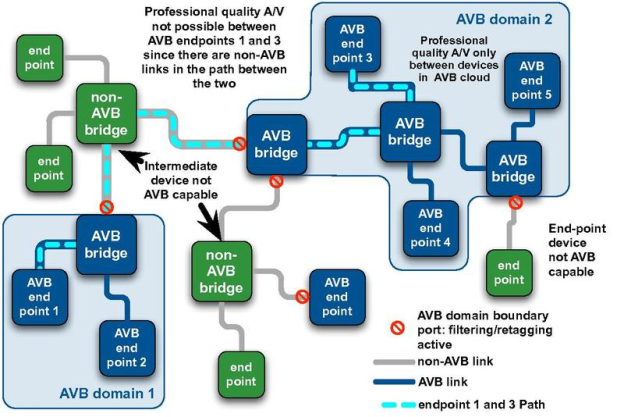


Figure 1: AVB connections [1]

These updates are implemented using rather small additions to the standard layer-2 MACs[[1]](#footnote-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:

1. Reserving a portion of network resources for streaming media
2. Sending and receiving the new time-sensitive frames.

### 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. 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.

#### 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.

#### Architecture

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

1. Ordinary clock
2. Boundary clock
3. End-to-end transparent clock
4. Peer-to-peer transparent clock
5. Management node

All five types implement one or more aspects of the protocol. In this project we will use on Ordinary clocks.

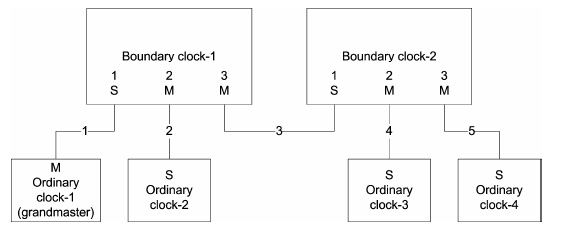


Figure 2: Simple master−slave clock hierarchy [2]

#### 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:

1. Sync
2. Delay\_Req
3. Pdelay\_Req
4. Pdelay\_Resp

The set of general messages consists of:

1. Announce
2. Follow\_Up
3. Delay\_Resp
4. Pdelay\_Resp\_Follow\_Up
5. Management
6. Signalling

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

#### Clock Synchronization Model

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

##### 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 3.

###### 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.

##### 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.

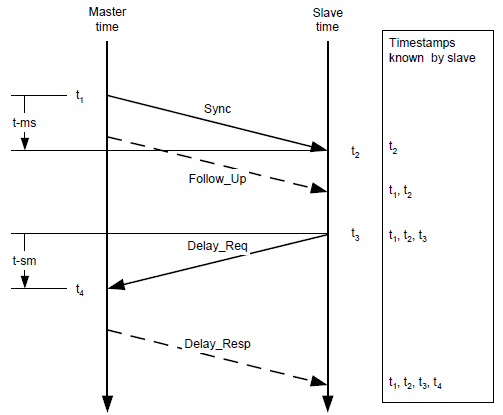


Figure 3: Basic synchronization message exchange [2]

The message exchange pattern is as follows:

1. The master initiates a Sync message to the slave and notes the time t1 at which it was sent.
2. The slave receives the Sync message and notes the time of reception t2.
3. 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)
4. The slave sends a Delay\_Req message to the master and notes the time t3 at which it was sent.
5. The master receives the Delay\_Req message and notes the time of reception t4.
6. 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.

#### 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.

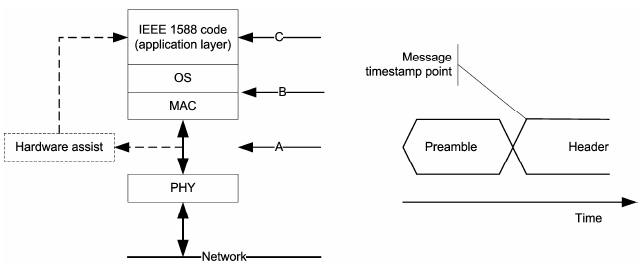


Figure 4: Timestamp generation model [2]

### 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.

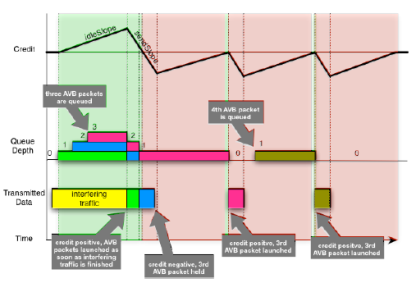


Figure 5: Example Qav traffic shaping [1]

### 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.

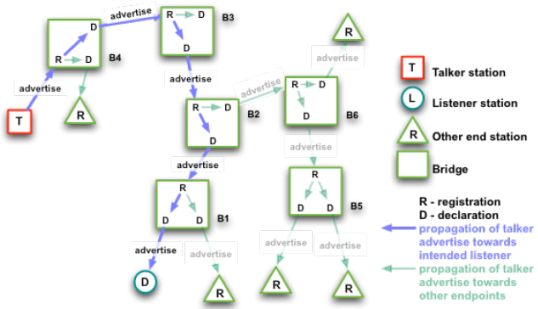


Figure 6: Successful reservation (talker advertise) [1]

### 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].

## 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.

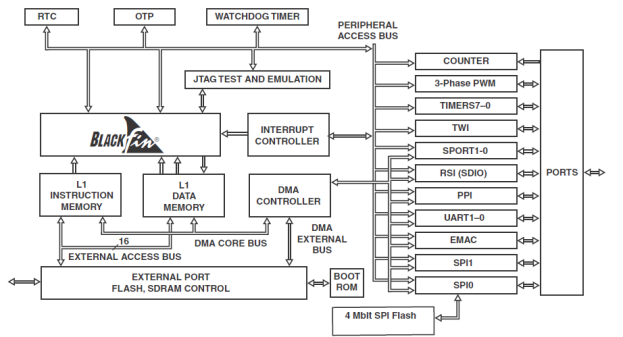


Figure 7: ADSP-BF518F Block Diagram [7]

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].

## 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.

# Project Plan

This section talks about the overall plan for the project execution, which includes the objectives, schedule and dependencies.

## Requirements

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

### 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.

### 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.

### 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.

## Scope

The following are in the scope of this project

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

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

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

## Tools and Platforms

These are the list of tools that were used during the execution of this project.

|  |  |
| --- | --- |
| **Tools Used** | **Purpose** |
| Visual DSP 5.0, Update 10 | This is the IDE used for the development of the Project source code. |
| Enterprise Architect | This was used for all the UML modelling from Use cases, architectural designs and detailed designs. |
| Github and SVN | This was the version control software used to manage the software modifications. |
| Wireshark | Testing and validation of the implementation. |

Table 1: Tools used in Project

## Project Schedule

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Description of Work** | **Start date** | **End date** |
| 1. | Ethernet AVB specification study   * Study of AVB specifications * Study of existing literature. | 01-01-13 | 13-01-13 |
| 2. | Implement the framework   * Integrate low level drivers | 14-01-13 | 27-01-13 |
| 3. | Implement the best master clock protocol | 28-01-13 | 10-02-13 |
| 4. | Implement clock synchronization | 11-02-13 | 24-02-13 |
| 5. | Implement clock syntonization | 25-02-13 | 10-03-13 |
| 6. | Testing the application with talker and a listener   * Testing for functionality * Testing for timing accuracy | 11-03-13 | 24-03-13 |
| 7. | Documentation | 25-03-13 | 31-03-13 |
| 8. | Project Submission | 01-04-13 | 01-04-13 |

Table 2: Project Schedule

# 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.

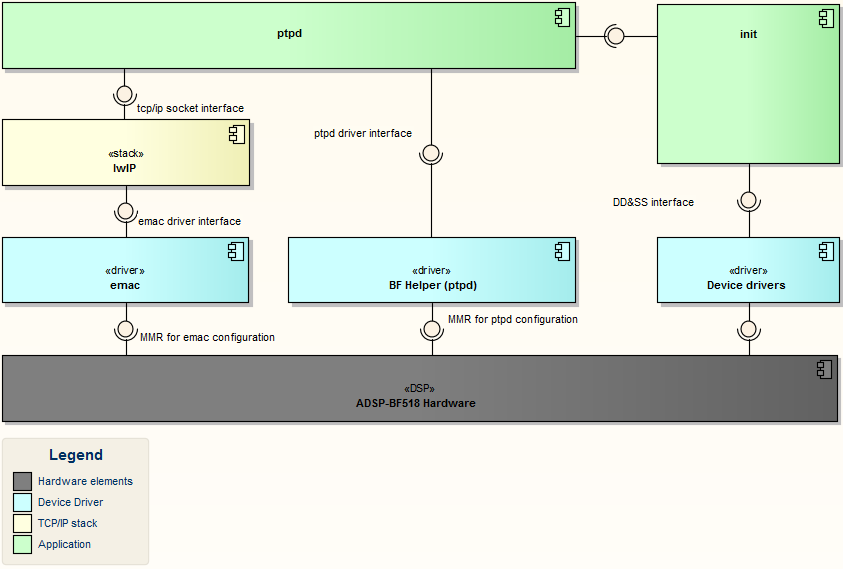


Figure 8: Software Architecture

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.

## <<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.

## <<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.

## <<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.

## <<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.

## <<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].

## <<application>> init

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

## 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.

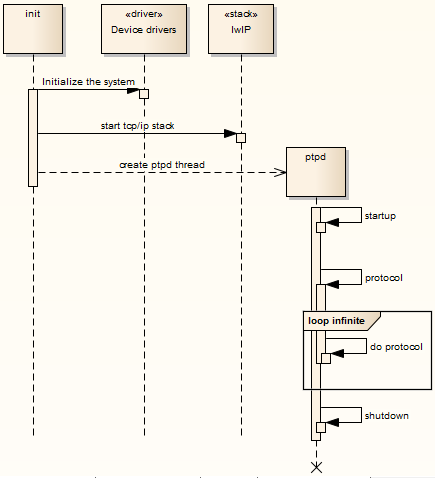


Figure 9: High level software sequence

# 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.

## <<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.

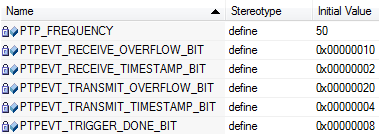


Figure 10: BF Helper list of attributes

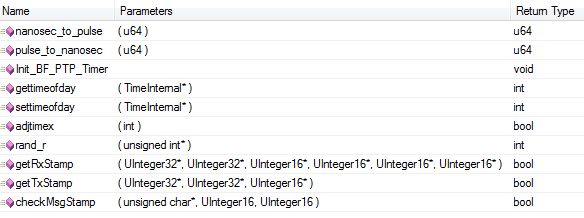


Figure 11: BF Helper list of operations

### Attributes

#### <<define>>PTP\_FREQUENCY

This is the PTP input clock frequency.

#### <<defines>>PTPEVT\_\*

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

### Operations

### [nanosec\_to\_pulse](mk:@MSITStore:E:\My%20Documents\Office\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/_b_f__helper_8c.html#a3e5edf30adf75222b80b1f904d52b3d0)()

Input: u64 nanosec

Return: u64 (pulse value)

Converts a given input nanosecond value into a pulse with reference to the PTP\_FREQUENCY.

### [pulse\_to\_nanosec](mk:@MSITStore:E:\My%20Documents\Office\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/_b_f__helper_8c.html#af80ec200016f76329d66b149e387beba)()

Input: u64 pulse

Return: u64 (nanosecond value)

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

### [Init\_BF\_PTP\_Timer](mk:@MSITStore:E:\My%20Documents\Office\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/_b_f__helper_8c.html#ab57de863bac156b0531d9c25cf255dcc)()

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.

### Gettimeofday()

Output: [TimeInternal](mk:@MSITStore:E:\My%20Documents\Office\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/struct_time_internal.html) \*tv

Return: int (success / failure)

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

### [settimeofday](mk:@MSITStore:E:\My%20Documents\Office\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/_b_f__helper_8c.html#adf8acd1f0ed87abf8da5be9370b5fefa)()

Input: [TimeInternal](mk:@MSITStore:E:\My%20Documents\Office\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/struct_time_internal.html) \*tv

Return: int (success / failure)

Sets the PTP clock time to the value in tv.

### [adjtimex](mk:@MSITStore:E:\My%20Documents\Office\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/_b_f__helper_8c.html#a2a962fd87eae51360e72969422210867)()

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.

### [rand\_r](mk:@MSITStore:E:\My%20Documents\Office\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/_b_f__helper_8c.html#adaac597461f394d44103a786c9ba0441)()

Input: unsigned int \*seedp

Return: int (random runmber)

A random number generation function.

### getRxStamp ()

Output: UInteger32 \*retNumberOfSeconds

Output: UInteger32 \*retNumberOfNanoSeconds

Output: UInteger16 \*overflowCount

Output: UInteger16 \*sequenceId

Output: UInteger16 \*messageType

Output: UInteger16 \*hashValue

Return: bool (success/failure)

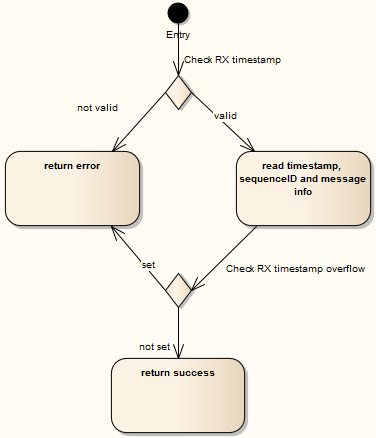


Figure 12: getRxStamp activity

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).

### getTxStamp

Output: UInteger32 \*retNumberOfSeconds

Output: UInteger32 \*retNumberOfNanoSeconds

Output: UInteger16 \*overflowCount

Return: bool (success / failure)

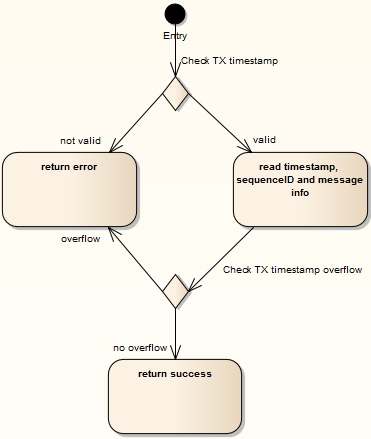


Figure 13: getTxStamp activity

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).

### 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.

## <<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

### Behavioural model

#### 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 beings 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

#### State machine

The PTP states are described in Table 3.

|  |  |
| --- | --- |
| 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 3: 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

#### Event handling

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



Figure 17: ptpd event handling

### 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].



Figure 18: ptpd::bmca operations

### 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].

#### Operations

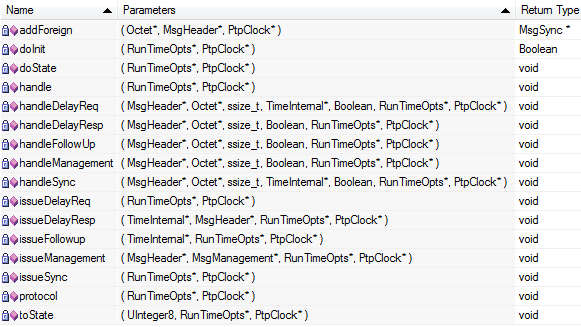


Figure 19: ptpd::protocol operations

##### 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.



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

##### 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.

##### 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.

##### 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.

### 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].

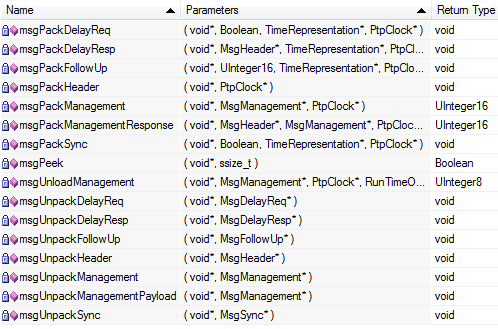


Figure 21: ptpd::msg handling operations

### 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.

#### Attributes



Figure 22: ptpd::net attributes

##### Hwaddr

This attribute stores the MAC address of the device.

##### myIP

This attribute stores the IP address of the device.

#### Operations

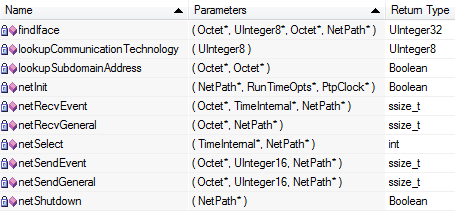


Figure 23: ptpd::net operations

##### 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.

##### 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, we statically return the following

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

##### 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.



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

##### [netRecvEvent](mk:@MSITStore:D:\Work\BITS\Project\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/net_8c.html#a064f012661c9cad539d05c1c2e67418f)()

Input: [Octet](mk:@MSITStore:D:\Work\BITS\Project\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/datatypes__dep_8h.html#af8e1bf583f840a555f39a222bc5d3ec0) \*buf

Input: [TimeInternal](mk:@MSITStore:D:\Work\BITS\Project\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/struct_time_internal.html) \*time

Input: [NetPath](mk:@MSITStore:D:\Work\BITS\Project\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/struct_net_path.html) \*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.



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

### 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].

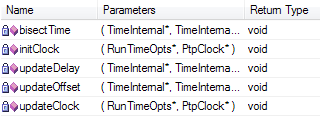


Figure 26: ptpd::servo operations

### 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.

#### Operations

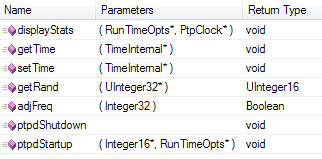


Figure 27: ptpd::sys operations

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



Figure 28: ptpd::sys sequence

### 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.

#### Operations

The following are the operations within the timer class.

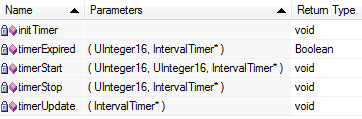


Figure 29: ptpd::timer operations

##### [initTimer](mk:@MSITStore:D:\Work\BITS\Project\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/ptpd__dep_8h.html#a02922529b60419a94afe8f369dda070b)()

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](mk:@MSITStore:D:\Work\BITS\Project\BitsPTP\trunk\Design\2011HZ12013-ClockSynchronization.chm::/_b_f__helper_8c.html#ab57de863bac156b0531d9c25cf255dcc)().

# Testing

## Test Setup

The following is the test setup that was used.

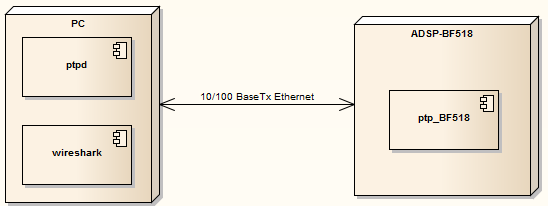


Figure 30: Test Setup

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.

## 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.

## Test Results

#### Wireshark reports

#### Captured packets

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

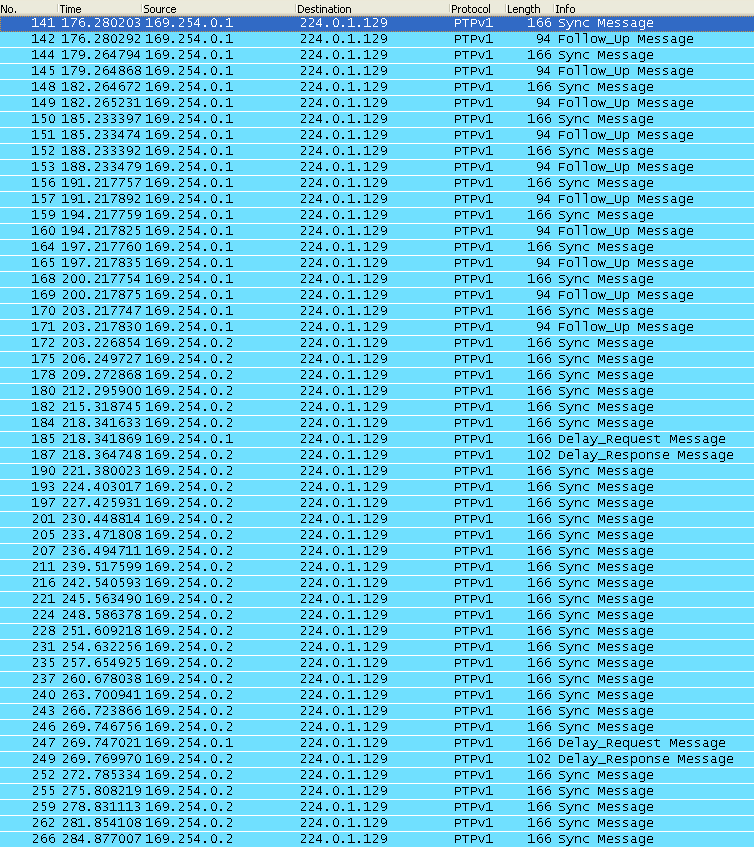


Figure 31: Wireshark captured packets

#### 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.

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

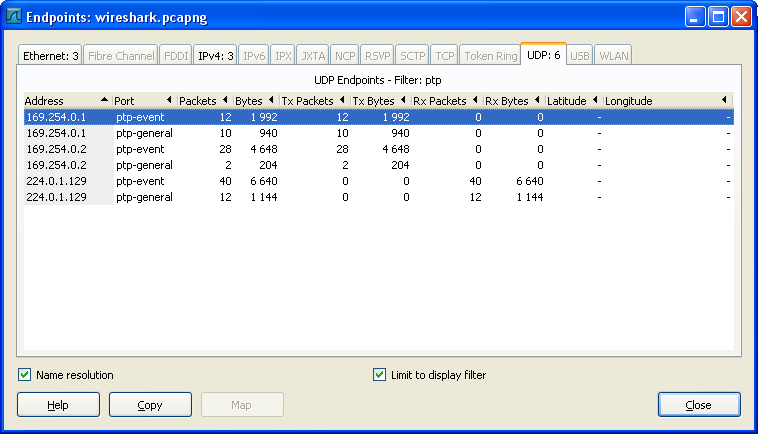


Figure 32: Wireshark endpoints

#### 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.

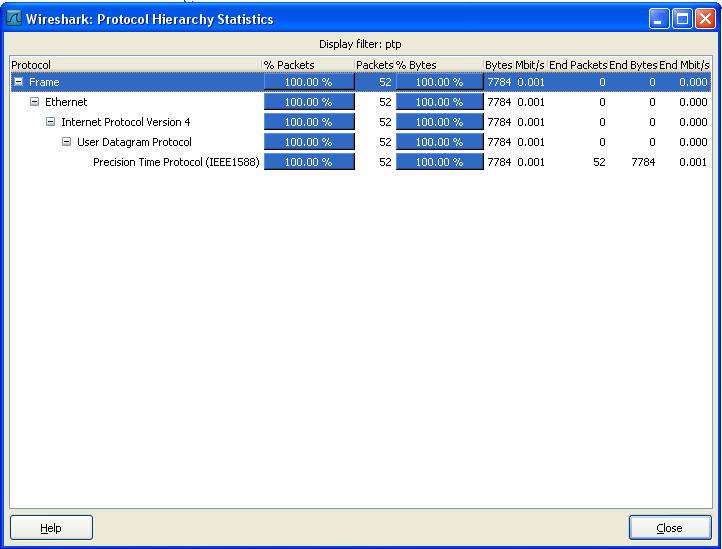


Figure 33: Wireshark protocol hierarchy

#### Conversations

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

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

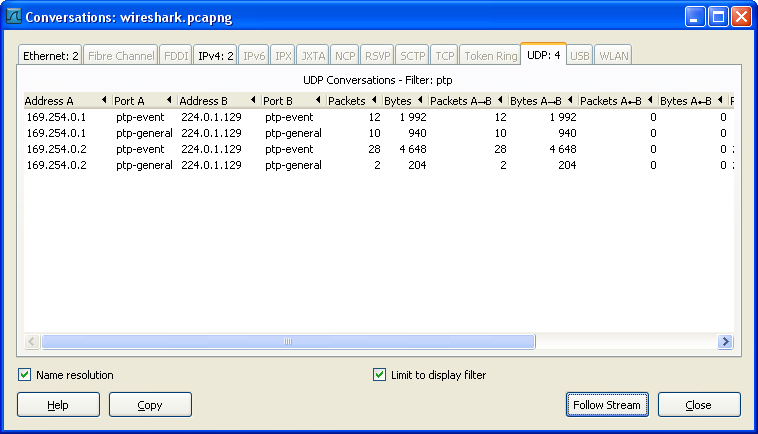


Figure 34: Wireshark PTP conversations

#### 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.

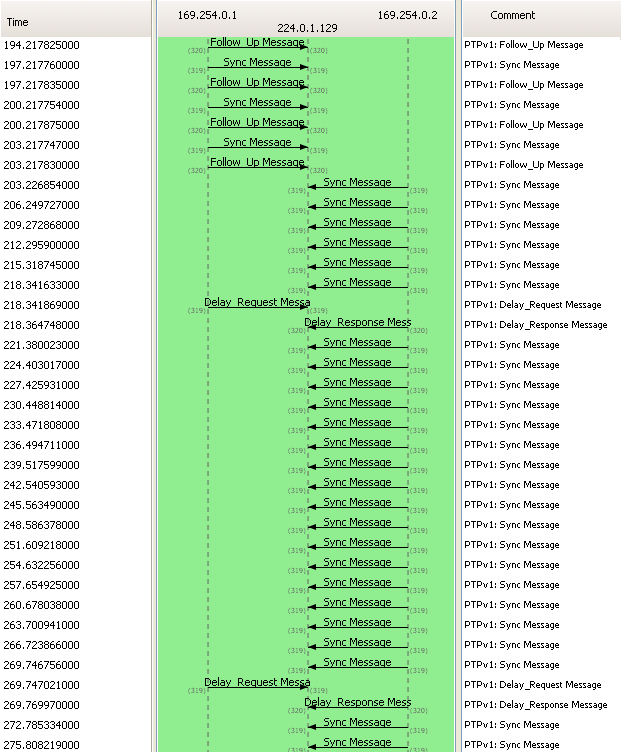


Figure 35: Wireshark flow

#### 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.

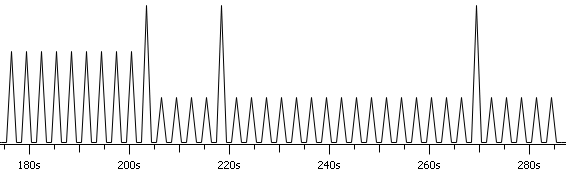


Figure 36: Wireshark IO rate

#### 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.

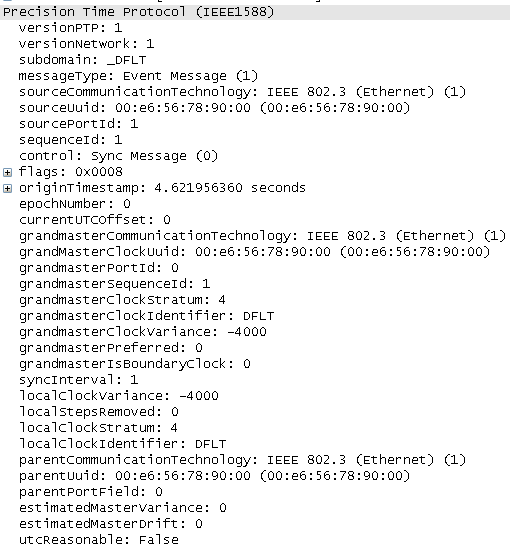


Figure 37: Wireshark sync message

#### 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.

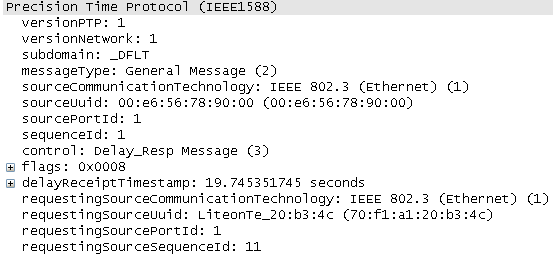


Figure 38: Wireshark delay response packet

### 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.

### Conclusions

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

# Summary

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

In **Chapter 1**, I look at an overview of the existing media streaming schemes. And then introduce the Audio Video Bridging standard. The “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 differences 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”):

Precise synchronization,

1. Traffic shaping for media streams,
2. Admission controls, and
3. Identification of non-participating devices.

In **Chapter 2,** I look at a plan for executing this project. I discuss the requirements that were drafted at the beginning of this project. I then discuss the scope of the project and clarify the items that are in and out of the scope of this work. I then explain the environment (tools and platforms) used during the execution of this project. And the chapter concludes with a detailed schedule of the project.

In **Chapter 3**, I look at the architectural design. I have chosen the “Layered” architectural design pattern for this work because of its significance for networking protocols. I explain the various components in the system briefly. I also explain the dynamic behaviour at the architecture level between the various components.

In **Chapter 4**, I delve deep into the mechanistic designs for the project. Here, we discuss in detail the two components that were implemented, the BF Helper (that provides PTP\_TSYNC drivers and other utility functions) and the ptpd which implements the PTP protocol. Detailed class and behavioural diagrams such as the activity diagram and the sequence diagrams are presented for the various classes. And the designs for each of the classes within these components are discussed.

In **Chapter 5**, I look at the test methodology, the setup and then publish the various results that were obtained during the testing. I conclude the chapter with an inference that the implementation adheres to the IEEE Std.1588 protocol as the requirements had started.

# 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, we can now build a complete media framework on top of this time synchronization. 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

# 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)

# 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 queueing (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.

1. 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

1. 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.**

|  |  |  |
| --- | --- | --- |
|  | **Is the final report properly** **hard bound?** (**Spiral bound or Soft bound or Perfect bound reports are not acceptable.)** | Yes / No |
|  | Is the Cover page in proper format as given in Annexure A? | Yes / No |
|  | Is the Title page (Inner cover page) in proper format? | Yes / No |
|  | (a) Is the Certificate from the Supervisor in proper format?  (b) Has it been signed by the Supervisor? | Yes / No  Yes / No |
|  | Is the Abstract included in the report properly written within one page? Have the technical keywords been specified properly? | Yes / No  Yes / No |
|  | 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 |
|  | Have you included the List of abbreviations / Acronyms?  Uncommon abbreviations / Acronyms should not be used in the title. | Yes / No |
|  | Does the Report contain a summary of the literature survey? | Yes / No |
|  | Does the Table of Contents include page numbers?   1. Are the Pages numbered properly? (Ch. 1 should start on Page # 1) 2. Are the Figures numbered properly? (Figure Numbers and Figure Titles should be at the bottom of the figures) 3. Are the Tables numbered properly? (Table Numbers and Table Titles should be at the top of the tables) 4. Are the Captions for the Figures and Tables proper? 5. Are the Appendices numbered properly? Are their titles appropriate | Yes / No  Yes / No  Yes / No  Yes / No  Yes / No |
|  | Is the conclusion of the Report based on discussion of the work? | Yes / No |
|  | 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 |
|  | 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** |
| **Date: 28th March 2013** | **Name: Mohan Karthik** |

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”. [↑](#footnote-ref-1)