



This Application Note describes the overview concept of IEEE 1588v2 standard and Precision Time Protocol as well as the procedure and architecture of Altera 1588 system solution reference design using Altera Arria V SoC, 10G Ethernet MAC with 10G BASE-R PHY hardware IP and software stack which is build based on Linux kernel v3.16, consists of PTP stack LinuxPTP v1.5, a preloader, 10G-bps Ethernet MAC driver and a PTP driver.

The objectives of the reference design include the following:

- 6.99ns timestamp accuracy using Altera 1588 hardware IP and Linux PTP software stack for 10Gbps speed.
- Offset adjustment capability from the system perspective for an ordinary clock (OC) in slave mode.
- Altera 1588 IP usability from the Linux system perspective for an ordinary clock in master mode.

This reference design includes hardware system design and software stack which allow you to:

- Perceive the usability of the hardware IP from various perspectives of the system solution.
- Assess the accuracy requirements in relation to the capability of this reference design.
- Replace the existing Linux PTP application with other PTP applications.

Related Information

- [AN739 Reference Design Image File](#)
- [AN739 Reference Design Files](#)

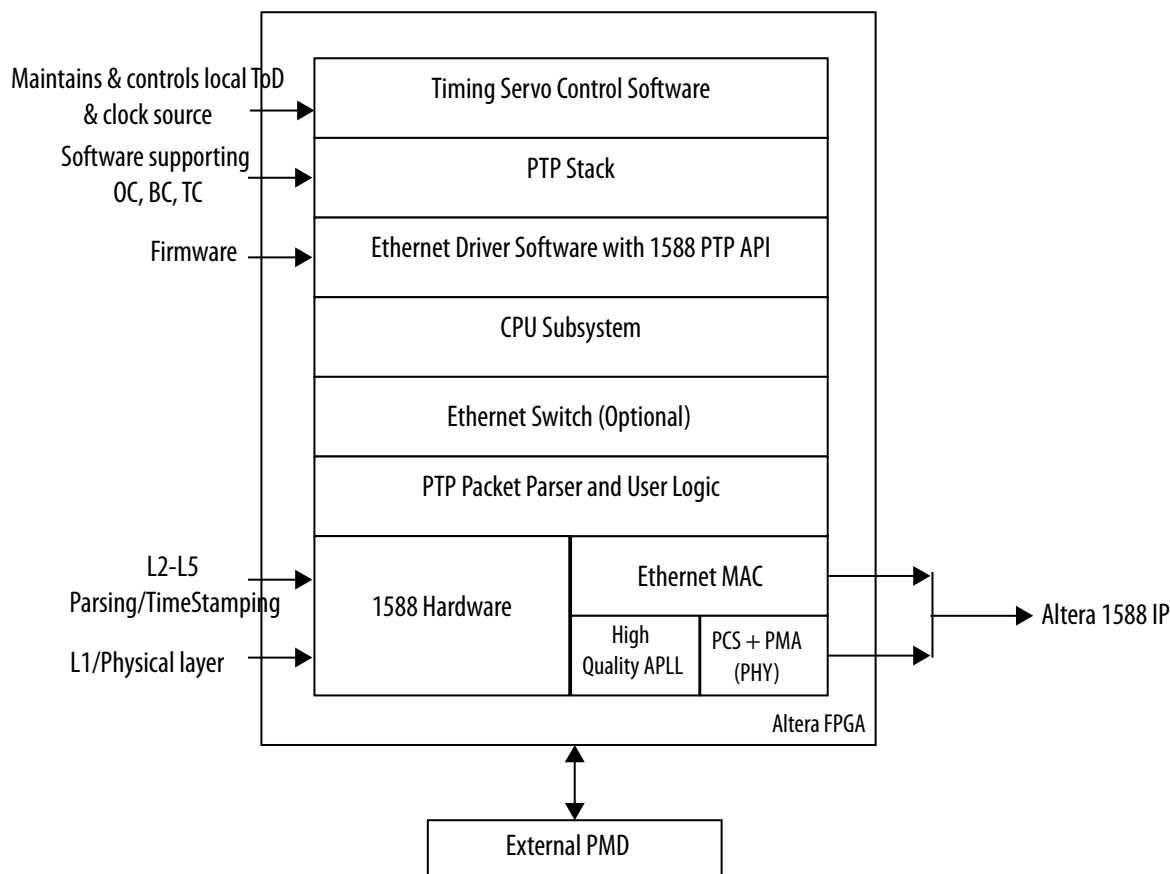
Altera 1588 System Level Overview

IEEE 1588v2 is a standard that defines a Precision Time Protocol (PTP) used in packet networking to precisely synchronize the real Time-of-Day (ToD) clocks and frequency sources in a distributed system to a master ToD clock, which is synchronized to a global clock source. Traditionally, such synchronization has been achieved in TDM (Time Division Multiplexing) networks due to their control on timing with their precise frequency and clocking hierarchy. However, with the advent of 1588v2, synchronization to nanosecond precision has become a possibility within packet networks, enabling low-cost phase and frequency synchronization in applications such as wireless, mobile backhaul, wireline and industrial instrumentation potentially replacing the higher cost of TDM networks.

The following diagram illustrates the 1588 system solution using Altera developed Ethernet driver and PTP stack running on top of Altera SoC, Ethernet MAC and transceiver with 1588 capability IPs. The software stack also provides accessibility to control and status registers for configuration using Ethtool and accessibility to the MAC statistic counters to observe the system performance using iperf utility.

Though the complete solution can be implemented in software, the hardware assisted the system to achieve the high accuracy needed in some applications such as mobile backhaul. All Altera Ethernet MAC and PHY 1588 hardware solution supports a static error of $\pm 3\text{ns}$ for throughputs of 10Gbps with random error of $\pm 1\text{ns}$ from the PHY.

Figure 1: 1588 System Solution Using Altera Ethernet MAC And PHY Hardware IP



Precision Time Protocol (PTP) Concept in A 1588 System

This section describe the below in details:

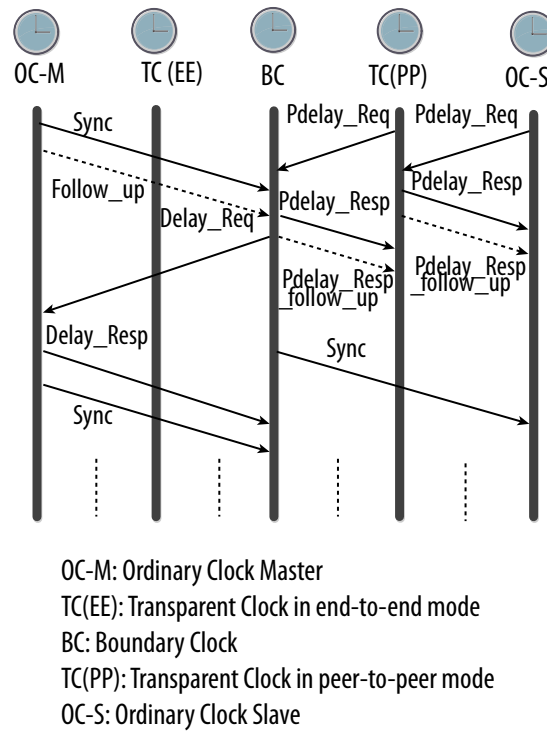
- Each type of PTP clocks used in a 1588 system.
- Synchronization process of the different clocks in a 1588 system.
- The functional flow of a 1588 Ordinary Clock in master and slave mode using Altera 1588 IP.
- The timestamp packet functional flow in a Linux driver.

Understanding the Different PTP Clocks

The following figure shows the different PTP clocks in a networked distributed system, where Altera Ethernet MAC with 1588 hardware IP can be used. The follow-up messages shown in the diagram are meant for 2-step mechanism and are not required for 1-step mechanism. The accuracy of hardware IP

time stamping for 1-step and 2-step mechanisms remains the same even with the existence of follow-up messages in 2-step mechanism.

Figure 2: Different PTP Clocks Defined in IEEE 1588v2



In addition to the clock nodes shown in the figure, there is also a management node communicating management messages to query and update the PTP datasets maintained by the clock nodes for the purposes of initialization and fault management. This node typically resides in a CPU.

The following describe the different PTP clocks present in the PTP protocol:

- **Ordinary Clock:**

An ordinary clock (OC) device can be either a master clock or a slave clock and it is a single port in a 1588 clock domain network.

- **Transparent Clock:**

A transparent clock (TC) device updates the PTP messages with the time taken by them to traverse through the network device from an ingress Ethernet port to an egress Ethernet port. In other words, the TC device updates the PTP event messages with their residence delay in the devices before the messages are transmitted out. The TC has two modes; end-to-end mode and peer-to-peer mode. The end-to-end TC uses the end-to-end delay measurement mechanism between slave clocks and the master clock. The peer-to-peer TC involves link delay correction using peer-to-peer delay measurement mechanism, in addition to the residence delay correction.

- **Boundary Clock:**

A boundary clock (BC) device has multiple ports in a 1588 clock domain with one slave clock port and possibly more than one master clock port. The BC maintains the same timescale for the slave clock port and possibly multiple master clock ports. A BC helps to avoid the long chain of transparent clocks between the network 1588 grandmaster and slave, leading to higher inaccuracy in the slave's ToD synchronization. The BC also helps to divide a larger 1588 network, thus reducing the traffic going all the way back to the original 1588 master. Typical application BC devices include routers, gateways and bridges.

Precision Time Protocol (PTP) Synchronization Process

The synchronization process involves ToD (Time of Day) offset correction and frequency correction between a master clock and a slave clock. The slave clock collects necessary data to synchronize its clock with master's clock through event messages. Below is the synchronization process flow:

1. The slave collects the timestamps of T1, T2, T3 and T4 through the event messages; Sync, Delay_Req, and Delay_Resp and calculates the mean path delay (MPD).
2. At the next sync message, the slave calculates the ToD offset by subtracting the MPD from the result of T6-T5 and adjusts its ToD counter accordingly.
3. Next, the slave clock calculates the frequency offset by comparing the time difference in frequency between 2 successively transmitted and received sync messages per the following equation:

$$\text{Frequency offset} = (F_o - F_r)/F_r$$

where $F_o = 1/(T_5 - T_1)$ and $F_r = 1/(T_6 - T_2)$,

T1 = Initial time for first transmitted sync message from master clock

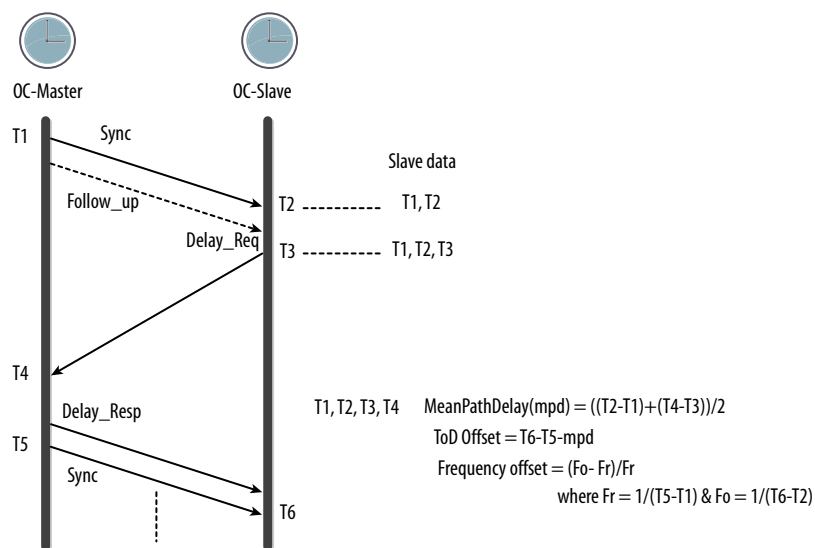
T2 = Initial time for first received sync message from slave clock

T5 = Initial time for second transmitted sync message from master clock

T6 = Initial time for second received sync message from slave clock

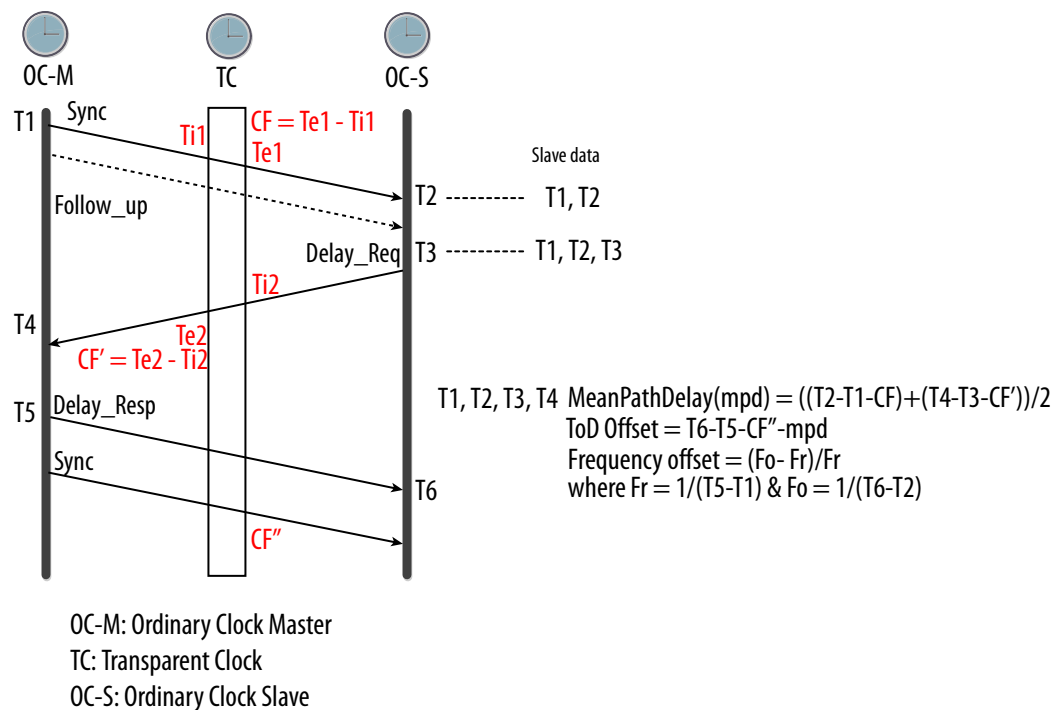
4. The slave clock calculates ToD offset and frequency offset continuously to maintain its ToD counter corresponding to the master clock to the best possible accuracy. This is most often accomplished through frequent syntonization offset adjustments after the initial ToD offset adjustment and occasional ToD offset adjustments.

Figure 3: Synchronization Between Master and Slave Clocks



The function of TC nodes is to calculate the residence delays of the PTP packets between a master clock and a slave clock to provide a more accurate offset. However, a long chain of TC nodes increases inaccuracy in synchronization.

Figure 4: Synchronization Between Master and Slave Clocks with Transparent Clock Mode



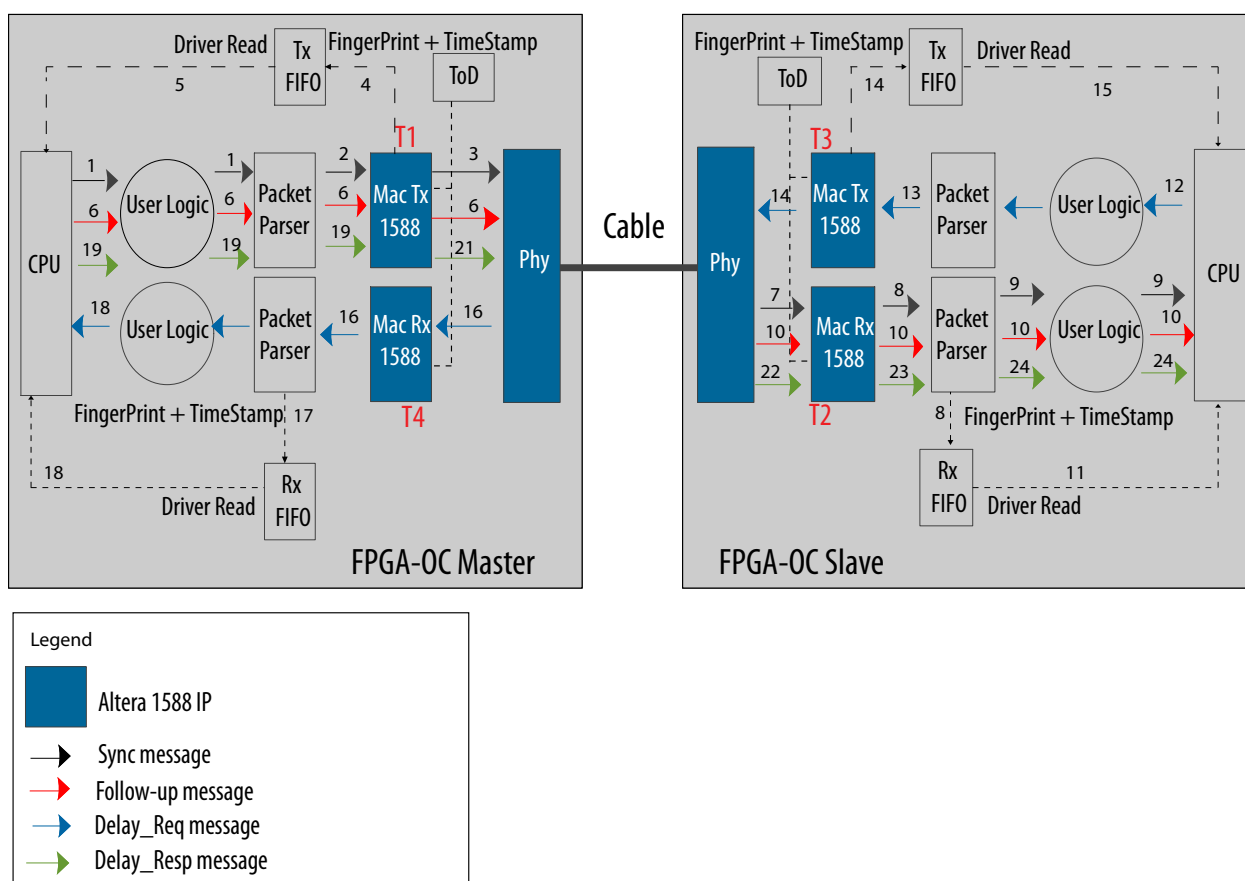
The BC device is used to break the long chain of TC nodes and maintain the accuracy of the synchronization. The BC device has a slave clock port synchronizing to a master clock external to the BC system. The master ports in a BC system use the timescale maintained by the slave clock within the same BC system.

Altera Ethernet MAC and transceiver IPs with fractional arithmetic capability can provide highest accuracy of 6.99ns for Ethernet link of 10Gbps speed, facilitating synchronization required even for stringent applications such as telecom and mobile backhaul. Further, the design facilitates visual demonstration of the achieved synchronization through PPS (pulse per second) output. The efficacy of the synchronization between the master and slave clock can be observe through the PPS output in an oscilloscope.

Functional Flow For A 1588 Ordinary Clock Master/Slave Mode System

The following figure illustrates the functional flow for Altera 1588 reference design system with Altera's Ethernet 1588 configured as ordinary clock (OC) master and slave clocks. In this flow, the CPU refers to the Linux PTP software stack and the PTP driver interacting with the stack.

Figure 5: Functional Flow For 1588 Ordinary Clock in Master & Slave Mode



1. The CPU acting as an OC-master sends a Sync packet to the network via its user logic.
2. The packet parser block extracts the fingerprint (an ID for the PTP packet) after decoding the PTP packet and sends the fingerprint to MAC Tx along with the packet.
3. For 1-step mechanism, the 1588 logic generates a timestamp T1 for the Sync packet before sending it to the PHY. This reference design provides the option to enable timestamping for every packet to integrate with LinuxPTP. However, the design can be changed to only timestamp PTP packets.
4. For 2-step mechanism, the transmitting MAC generates the timestamp T1 before sending the Sync packet to the PHY and stores the timestamp T1 along with its fingerprint in the FIFO.⁽¹⁾
5. The OC-master checks if there is a valid entry in the FIFO via interrupt or polling mechanism. It then reads the entry from the FIFO to collect the timestamp T1 and the fingerprint. The dotted arrows in OC-master indicate the blocks involved in 2-step mechanism.⁽¹⁾
6. The CPU then sends a Follow-up message with timestamp T1, which refers to the Sync packet sent in step 1.⁽¹⁾
7. The FPGA hosting the OC-slave clock receives the Sync packet and the receiving MAC generates the timestamp T2.
8. The packet parser stores the corresponding timestamp T2 and the fingerprint in the FIFO. You can design a packet parser which can validate the incoming packet as a PTP packet and stores the timestamp T2 along with its fingerprint in the RX FIFO, ignoring the timestamps for the non-PTP packets.
9. The CPU receives the Sync packet via user logic and collects timestamp T1 from the received packet.⁽²⁾
10. In 2-step mechanism, the CPU acting as the OC-slave collects timestamp T1 from the Follow-up message sent in step 6.
11. The CPU then reads the FIFO to collect the fingerprint and the timestamp T2.
12. The CPU acting as the OC-slave sends Delay_Req packet via its user logic.
13. The packet parser block after decoding the PTP packet extracts the fingerprint and sends it to the MAC block along with the packet.
14. The MAC Tx of the OC-slave clock generates timestamp T3 before sending the packet to the PHY and stores the timestamp along with its fingerprint in the FIFO.
15. The CPU acting as the OC-slave reads the FIFO to obtain the timestamp T3.
16. The FPGA hosting the OC-master clock receives the Delay_Req packet and the receiving MAC generates the timestamp T4.
17. The packet parser stores the corresponding timestamp T4 and the fingerprint in the FIFO.
18. The CPU acting as the OC-master receives the PTP packet via user logic and reads the FIFO to collect the fingerprint and timestamp T4.
19. Next, the CPU sends a Delay_Response packet containing timestamp T4 via its user logic.
20. The MAC Tx of the OC-master timestamps the Delay_Response packet but the software stack will ignore the timestamp. You can design a packet parser to instruct the MAC to not generate a timestamp for the Delay_Response packet, which already contains the timestamp T4, after decoding the PTP packet.
21. The MAC Tx of the OC-master sends the Delay_Response packet as it is.
22. The FPGA hosting the OC-slave clock receives the Delay_Response packet.
23. The receiving MAC timestamps the Delay_Response packet and stores it in the FIFO. The CPU reads the FIFO but the timestamp collected through the FIFO read is ignored by the software stack. You can

⁽¹⁾ This flow is only applicable in a 2-step mechanism.

⁽²⁾ This flow is only applicable in a 1-step mechanism.

design a packet parser block which able to identify Delay_Response packet type and sends the packet further to the user logic ignoring its timestamp.

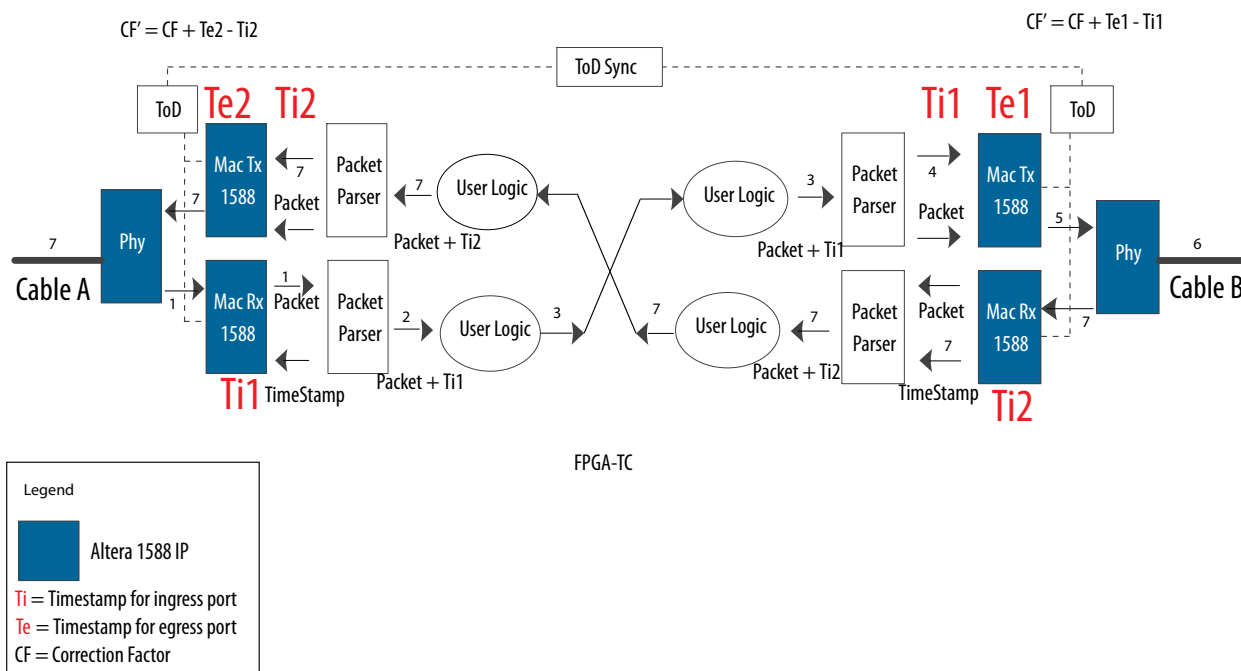
24. The CPU receives Delay_Response packet via user logic and collects the timestamp T4 from the packet.
25. At this point, the stack in the OC-slave has all the timestamps T1, T2, T3, & T4 from which the mean path delay (MPD) is calculated.
26. The OC-slave collects the timestamps T5 & T6 after receiving the next Sync packet, and calculates the ToD offset as described in [Precision Time Protocol \(PTP\) Synchronization Process](#) on page 4.
27. Next, with the time difference between two successive Sync packets, the stack in the OC-slave calculates the frequency offset as described in [Precision Time Protocol \(PTP\) Synchronization Process](#) on page 4.
28. The stack residing in the OC-slave CPU supplies the calculated ToD offset and frequency offset to Servo algorithm, which can either adjust the PLL settings supplying the PTP clock to the ToD or program the registers in the 1588 IP to synchronize the slave's ToD with master's ToD.
29. Step 1 to step 24 is repeated continuously to adjust the settings for slave's ToD with the calculated ToD and frequency offset.
30. The PPS pulse outputs from the OC-master and the OC-slave can be observed in an oscilloscope to appreciate the efficacy of synchronization process.

The correction field in the PTP header may be updated in any of the above mentioned PTP packets for any fractional arithmetic residue or asymmetry.

Functional Flow For A 1588 Transparent Clock Master/Slave Mode System

The following figure illustrates timestamp synchronization flow using 1-step mechanism transparent clock mode on the Altera 1588 system. In this figure, the transparent clock consists of two Altera 1588 IP cores to create two ports system; one port as PTP packet input port and another as PTP packet output port. The residence delay is calculated from the timestamp of the input port to the timestamp of the output port.

Figure 6: Functional Flow For 1588 Transparent Clock Mode with Multiple ToD

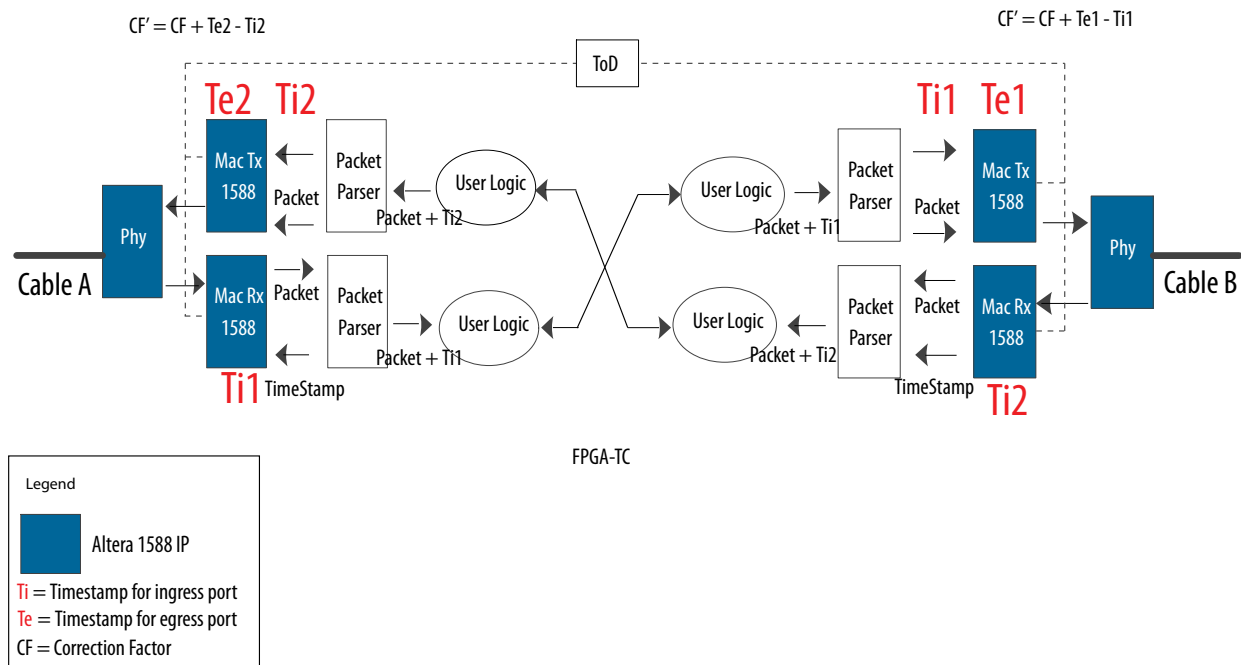


1. PTP packet enters the FPGA through cable A and the Mac Rx 1588 logic generates timestamp Ti1.
2. The packet parser validates the PTP packet before sending it to the user logic along with the timestamp information, ignoring all timestamps for non-PTP packets.
3. The user logic at the egress port send the PTP packet along with timestamp Ti1 to the packet parser of the egress port.
4. The packet parser, then validates the PTP packet and send the packet along with the timestamp Ti1 to Mac Tx 1588 logic along with the required command.
5. Mac Tx 1588 logic generates timestamp Te1 and updates the correction factor (CF) in the packet header. The CF field contains the new residence delay, calculated as $CF' = CF + Te1 - Ti1$.
6. The PHY, then transmits the PTP packet to the network.
7. The same flow from step 1 to step 6 applies to the PTP packet received by cable B, with the timestamp components of Ti2 and Te2.

For 2-step mechanism, the CPU collects the ingress and egress timestamps from FIFOs, but the PTP packet remain unchanged. The ingress port receives a follow-up packet from the network and Mac Rx 1588 logic generates a timestamp for the follow-up packet. The packet parser verifies it is a follow-up packet and send it to the CPU via user logic. The CPU updates the follow-up packet after matching the packet's fingerprints with the new correction field of $CF' = CF + Te - Ti$ before sending it to the egress port.

The ingress and egress port of a TC mode system may have a single or multiple ToD modules. For system with single ToD module, the ToD operates using the transmit clock to use a single clock to calculate the residence delay. For system with multiple ToD modules, a ToD synch module is used to synchronize the residence delay between the ToD modules.

Figure 7: Functional Flow For 1588 Transparent Clock Mode with Single ToD

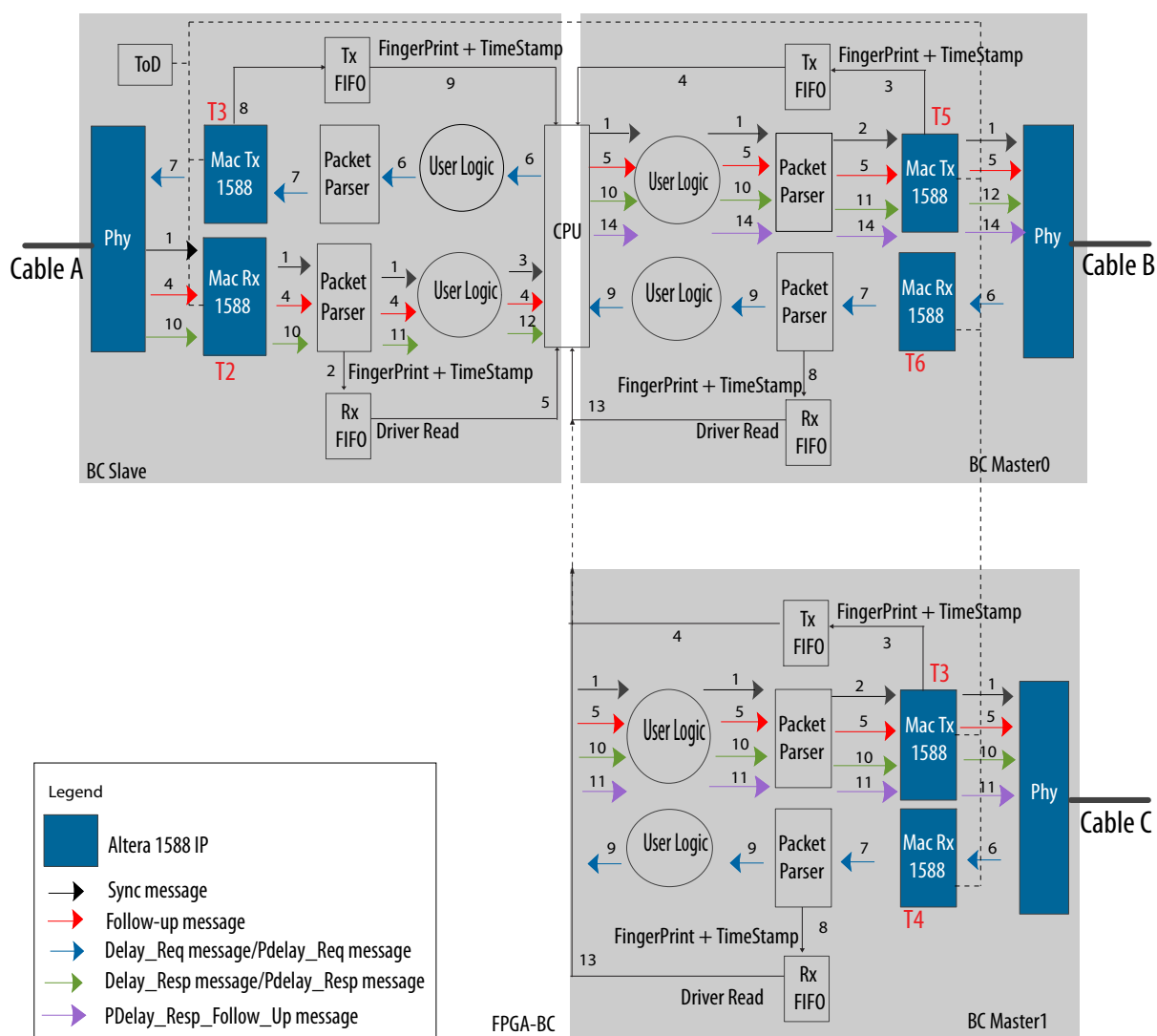


Functional Flow for A 1588 Boundary Clock Mode System

The function of a boundary clock mode system is to terminate a long chain of transparent clocks leading to high inaccuracy, and maintain a single timescale. A boundary clock commonly contains a slave clock port and at least one master clock port. However, the boundary clock has all the ports in one domain and maintain the timescale used in the domain.

The following figure illustrates a BC mode system consists of one slave port and two master ports. The BC-slave port synchronizes with an external grand master clock through the network. The BC-master ports synchronizes its clocks with the BC-slave port. In the figure, one ToD module is connected to the BC-slave port and the BC-master ports. Each of the ports shown in the figure operates independently. Timestamp T1 and T2 are synchronized to an external grand master clock while timestamp T3 and T4 are synchronizes to the BC-slave ToD. The CPU hosts the independent stacks for the BC-slave port and the two BC-master ports with the common timescale of the BC-slave ToD.

Figure 8: Functional Flow For 1588 Boundary Clock in Master & Slave Mode

**Timestamp Functional Flow for BC Slave Port:**

1. The receiving MAC of the BC-slave port receives a Sync packet with timestamp T1 from external 1588 system through cable A and generates timestamp T2.
2. A packet parser validates the incoming packet and stores timestamp T2 and its fingerprint in the FIFO.
3. The CPU receives the Sync packet with timestamp T1 via user logic.⁽³⁾
4. In 2-step mechanism, the receiving MAC of BC-slave port receives a Follow-Up message timestamp T1 which refers to the Sync message in step 1.
5. The CPU then reads the FIFO to collect the fingerprint and the timestamp T2.
6. The CPU sends Delay_Req packet via its user logic..

⁽³⁾ This flow is only applicable in a 1-step mechanism.

7. The packet parser block after decoding the PTP packet extracts the fingerprint and sends it to the MAC block along with the packet.
8. The MAC Tx of the BC-slave generates timestamp T3 before sending the packet to the PHY and stores the timestamp along with its fingerprint in the FIFO.
9. The CPU then reads the FIFO and collects fingerprint and timestamp T3.
10. The receiving MAC receives a Delay_Response packet and timestamp the packet. The MAC then stores the timestamp in the FIFO and forward the Delay_Response packet to packet parser
11. The packet parser validates the packet and extracts the fingerprint before storing it in the FIFO.
12. The CPU receives the Delay_Response packet with timestamp T4. The CPU then reads the FIFO but the timestamp collected through the FIFO read is ignored by the software stack. .

Timestamp Functional Flow for BC Master Port:

1. The CPU sends a Sync packet to the external 1588 system through MAC Tx and the PHY.
2. The packet parser block extracts the fingerprint after decoding the Sync packet and sends the fingerprint to the MAC Tx along with the packet.
3. The MAC Tx generates timestamp T5 for the Sync packet before sending it to the PHY.⁽³⁾ The MAC Tx then stores the timestamp T5 with its fingerprint in the FIFO.
4. The CPU then reads the FIFO to collect the fingerprint and the timestamp T5.
5. For 2-step mechanism, the CPU sends a Follow-up message with timestamp T5 referring to the Sync packet sent in step 1.
6. The receiving MAC of the BC-master clock receives a PDelay_Req from external 1588 system through the PHY.
7. The MAC Rx receives the PDelay_Req packet and generates timestamp T6 before storing it in the FIFO. It then sends the PDelay_Req to the packet parser.
8. The packet parser decodes the packet and extracts its fingerprint before storing in the FIFO.
9. The CPU receives the PDelay_Req packet via user logic and collects the fingerprint and the timestamp T6 from the FIFO.
10. The CPU sends a PDelay_Resp packet containing the timestamp T6 to the packet parser.
11. The packet parser validates and extracts the PDelay_Resp packet fingerprint before sending it to the MAC.⁽³⁾
12. The MAC Tx receives the packet and generates a timestamp before storing the timestamp and its fingerprint into the FIFO for CPU to read. However, this timestamp is ignore by the software stack. The MAC Tx then sends the PDelay_Resp packet as it is.
13. For 2-step mechanism, the CPU reads the entry from the FIFO to collect the timestamp T6 and its fingerprint
14. The CPU then sends a Pdelay_Resp_Follow_Up packet with timestamp T6 to the external 1588 system.

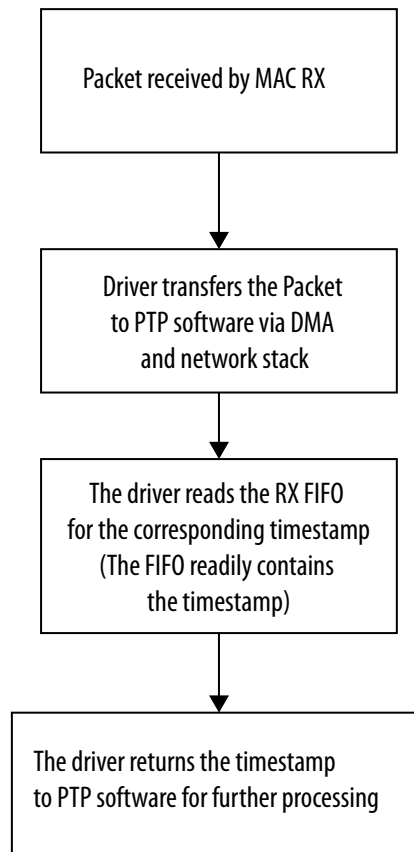
Timestamp Packet Functional Flow in Linux Driver

This section describes the timestamp packet flow between Linux driver and the system hardware.

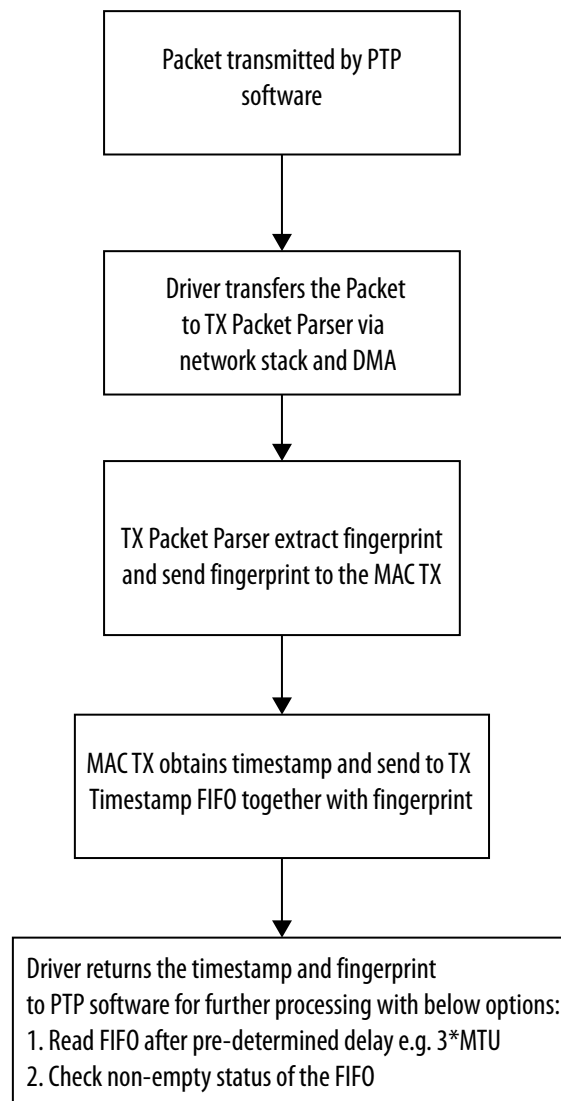
The reference design uses TX Ingress Timestamp FIFO and RX Ingress Timestamp FIFO to store the timestamp with fingerprint. The Linux PTP driver will access these FIFO to obtain the timestamp information for PTP software processing. A FIFO status read can be omitted if the FIFO read delay is determined. CPU can read the RX FIFO without any delay while for TX FIFO, a delay is necessary and most often depends on the sum of MTU delay of a scheduled packet, DMA delays and the PTP packet transmit delays. In general, 3*MTU delay is the common value for TX FIFO read delay.

The following figure shows the timestamp packet flow between Linux driver and the system hardware for receive path.

Figure 9: Timestamp Flow Between Linux Driver and System Hardware for Receive Path



The following figure shows the timestamp packet flow between Linux driver and the system hardware for transmit path.

Figure 10: Timestamp Flow Between Linux Driver and System Hardware for Transmit Path

Reference Design Walk-through

This section describes the procedure on setting up the development kit and how-to in executing the reference design.

Related Information

- [AN739 Reference Design Image File](#)
- [AN739 Reference Design Files](#)

System Requirements

This section describes the hardware and software requirements in order to use the reference design.

The following are the hardware and software requirements to run this reference design:

Hardware requirements:

- 2 units of Arria V SoC board (Arria V ST SoC-5ASTFD5K3F40I3NES (SoC))
- 6 units of SMA cables
- 2 units of micro SD card
- SD card reader
- USB II Blaster
- 2 units of UART serial cable

Software requirements:

- Altera Complete Design Suite (ACDS) version 14.1
- Arria V SoC Development Kit Installation. This installation kit only compatible with Altera Complete Design Suite (ACDS) version 13.1.
- Altera 1588 system solution reference design image file that includes:
 - soc_system.rbf - FPGA raw binary file
 - u-boot.img - u-boot image
 - u-boot.src - u-boot script
 - socfpga.dtb - device tree block
 - Ext-3 - Root File System
 - Preloader

Related Information

- [AN739 Reference Design Image File](#)
- [AN739 Reference Design Files](#)
- [Intel Quartus II Subscription Edition Design Software](#)
- [Arria V SoC Development Kit](#)

Reference Design User Guide

This section guides you through the steps required to run the reference design on Altera Arria V SoC development kit.

This section is divided into 3 parts:

- Hardware setup.
- Executing the reference design.
- Replacing RBF file.

Hardware Setup

This section provides a step by step guide in setting up the hardware for the reference design.

This section is divided into 3 parts:

1. Loading the boot image into SD card.
2. Setting up the development kit.
3. Booting up the reference design.

Loading the Boot Image into SD Card

The procedures shown in this section are performed using a Linux host. You may use different methods to load the boot image into a micro SD card.

1. Plug in SD Card Reader to your Linux host and mount the micro SD card. Reformat the micro SD card to delete any existing content in it.

```
sudo modprobe usb-storage
```

Note: Use `ls -la /dev/sd*` to identify the micro SD card device name. In the steps below, replace `<sd card device>` with the micro SD device name shown in your Linux host.

2. Download `sd_image.img.gz` image file to your computer and unzip the image file.

```
gunzip sd_image.img.gz
```

3. Program the `sd_image.img` file into the micro SD card.

```
sudo dd if=./sd_image.img of=/dev/<sd card device>
```

4. Unmount the micro SD card device before unplugging it from the SD Card Reader.

```
sudo umount /dev/<sd card device>
```

Related Information

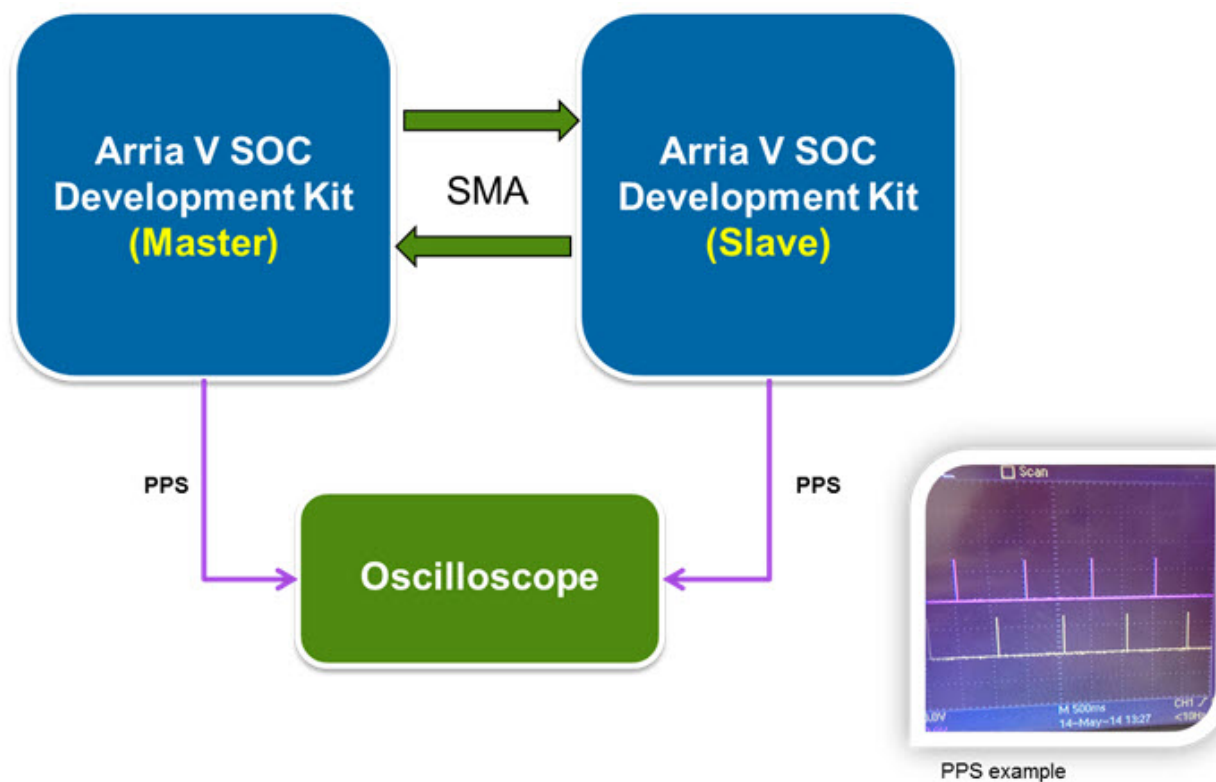
- [AN739 Reference Design Image File](#)
- [AN739 Reference Design Files](#)

Setting Up Development Kit

The following procedures are performed using a Windows host.

The figure below shows the hardware connections in order to run this reference design.

Figure 11: Reference Design Hardware Setup

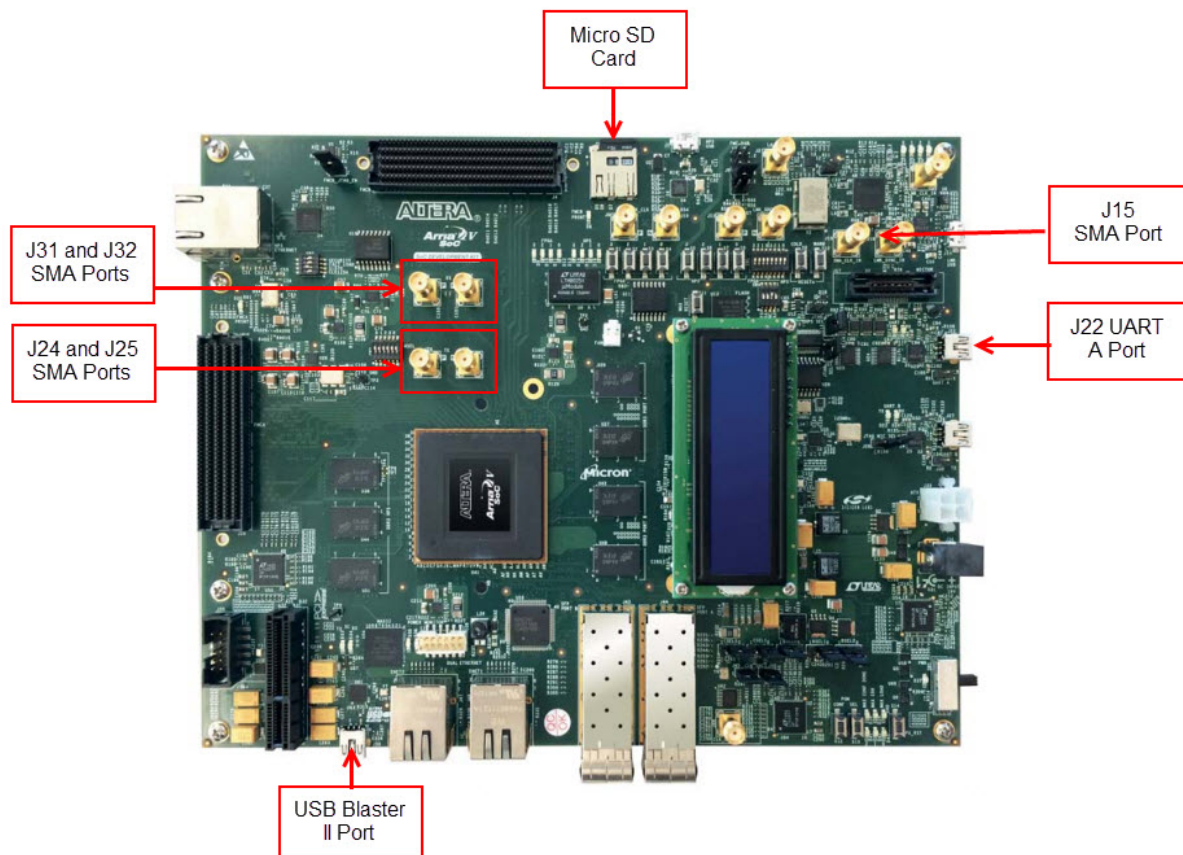


1. Download and install Quartus II Subscription Edition software v14.1 and Altera Arria V SoC Development Kit Installations on your Windows host.
2. Place two Arria V SoC development boards (5ASTFD5K3F40I3NES) side by side. In the following steps, the two Arria V SoC development boards will be referred to as Board A and Board B. Setup the SMA connections between these 2 boards per below:
 - Port J24 on Board A connects to port J31 on Board B.
 - Port J25 on Board A connects to port J32 on Board B.
 - Port J24 on Board B connects to port J31 on Board A.
 - Port J25 on Board B connects to port J32 on Board A.

Note: Refer to [Arria V SoC Development Board Reference Manual](#) for more information on connectors available on the board.

3. Set MSEL [4..0] pins (SW3) to value 00110 to enable compression feature for Board A and Board B.
4. Connect both serial cables on Board A and Board B (J22 port) to your Windows host.

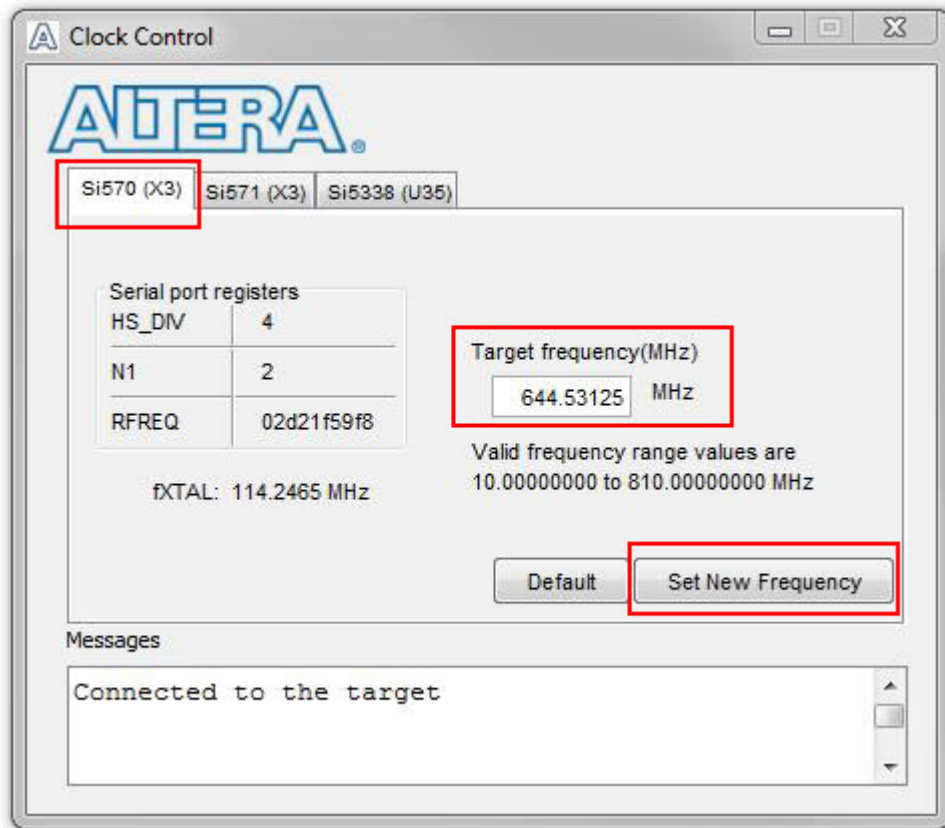
Figure 12: Arria V SoC Development Board Connections Overview



Booting Up The Reference Design

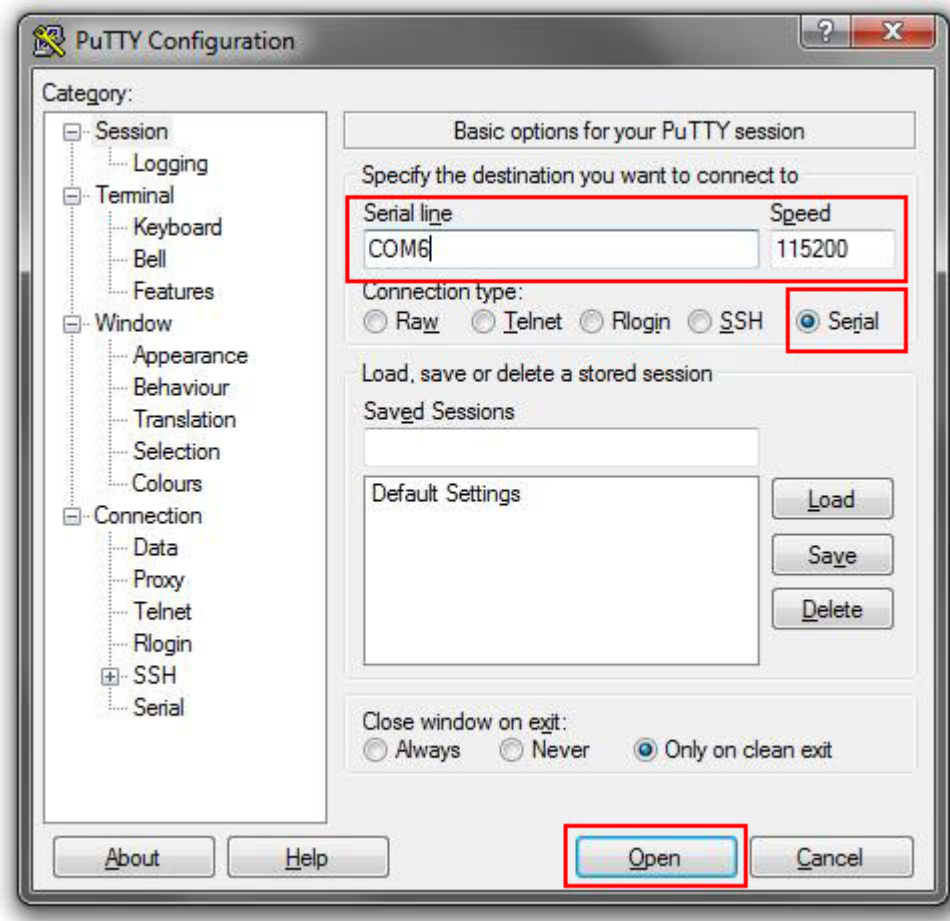
1. Connect the micro SD card with the bootable image to J5 port on board A.
2. Connect USB II blaster cable from Board A to your Windows host that has Arria V SoC development kit installed.
3. Turn on the power for Board A to load the bootable image. The **Arria V 10GbE** appears on the LCD screen on the board when the image is successfully loaded.
4. Open the Clock Control application from the Arria V SoC development kit installation on your Windows host.
5. Set Channel 0 reference clock source, Si570(X3) to 644.53125MHz. This step is required to run the reference design at 10Gbps data rate. FPGA LED [0] will blink every 2 seconds to indicate that the hardware clock is running at 10Gbps data rate if the reference clock source Si570 is successfully configured.

Figure 13: Clock Control Application Settings



- You may choose to access the Linux operating system on the Arria V SoC development board using any serial communication program such as Putty or minicom applications that are available to you. In these steps, Putty application is used as an example. You may follow the setup shown in the following figure to configure the Putty tool.

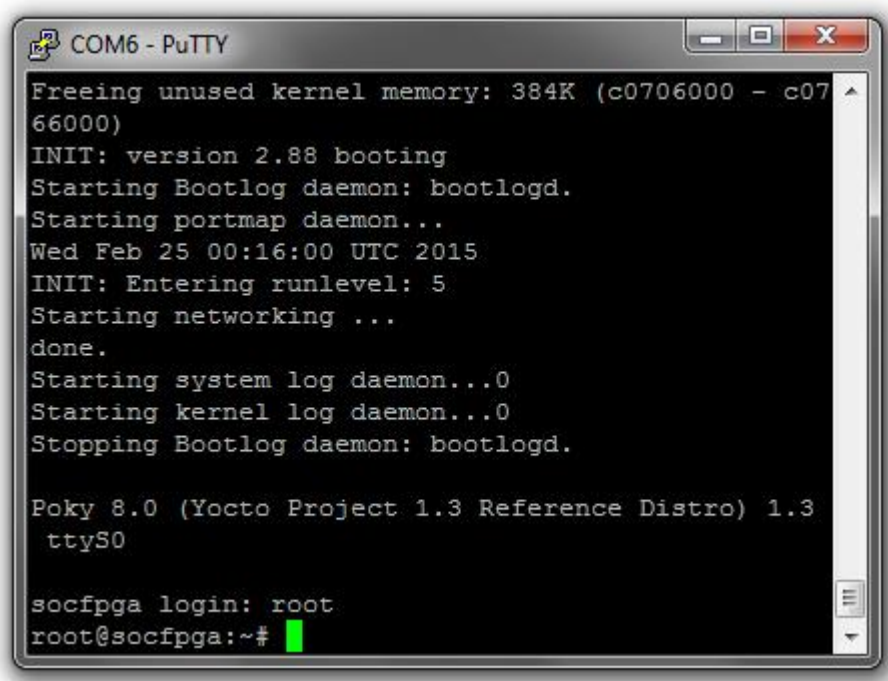
Figure 14: Putty Configuration Settings



Note: Always check the serial line ID of the COM port connected to your computer. In this example, the serial line ID is COM6.

7. Type `root` as the login name when requested.

Figure 15: Arria V SoC Operating System Login Prompt



```
COM6 - PuTTY
Freeing unused kernel memory: 384K (c0706000 - c0766000)
INIT: version 2.88 booting
Starting Bootlog daemon: bootlogd.
Starting portmap daemon...
Wed Feb 25 00:16:00 UTC 2015
INIT: Entering runlevel: 5
Starting networking ...
done.
Starting system log daemon...0
Starting kernel log daemon...0
Stopping Bootlog daemon: bootlogd.

Poky 8.0 (Yocto Project 1.3 Reference Distro) 1.3
ttyS0

socfpga login: root
root@socfpga:~#
```

8. Repeat step 1 to step 8 for Board B.
9. In the Putty console, type `reboot` to reset the system for both Board A and Board B.
10. Observe that FPGA LED [2] on board A and board B will turn off for sometime before turning on again indicating that the link between Board A and Board B is successfully connected.
11. You may connect port J15 to an oscilloscope in order to view the ToD Pulse Per Second waveform.

Executing The Reference Design

This section shows step by step guide to execute the reference design on the Altera Arria V SoC Development Board.

1. Login to the system on both Board A and Board B.
2. Assign IP address for Board A.
`ifconfig eth0 10.0.0.1 netmask 255.255.255.0 up`
3. Assign IP address for Board B.
`ifconfig eth0 10.0.0.2 netmask 255.255.255.0 up`
4. Type `ifconfig` to check the IP addresses are successfully assigned to Board A and Board B.
The following figure shows a screen capture for a successful IP address assignment.

Figure 16: Successful IP Address Assignment

```

root@socfpga:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 66:f1:d5:92:ad:f1
          inet addr:10.0.0.1  Bcast:10.0.0.255  Mask:255.255.255.0
          inet6 addr: fe80::64f1:d5ff:fe92:adf1/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7  errors:0  dropped:0  overruns:0  frame:0
          TX packets:8  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:592 (592.0 B)  TX bytes:664 (664.0 B)
          Memory:ff210000-ff217fff

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128  Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0  errors:0  dropped:0  overruns:0  frame:0
          TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

5. Test the network connectivity between the two boards by using ping command to the assigned IP addresses.

Example: Type ping 10.0.0.2 from Board A.

Type ping 10.0.0.1 from Board B.

The following figure shows successful connections between the two boards.

Figure 17: Successful Connection Screen Capture

```

root@socfpga:~# ping 10.0.0.2 -c 5 -s 1500
PING 10.0.0.2 (10.0.0.2) 1500(1528) bytes of data.
1508 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=0.367 ms
1508 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=0.216 ms
1508 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=0.198 ms
1508 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=0.192 ms
1508 bytes from 10.0.0.2: icmp_req=5 ttl=64 time=0.191 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3996ms
rtt min/avg/max/mdev = 0.191/0.232/0.367/0.070 ms

```

6. You can check the Altera 10G-bps Ethernet MAC register values and statistics using the following commands.

Use `ethtool -S eth0` to view statistic counters.

Use `ethtool -d eth0` to view MAC register settings.

7. You can use the Ptp4l software in the LinuxPTP package to measure the timestamp accuracy of the reference design. By default, the hardware timestamping is selected and time interval between SYNC messages is set to 1 second.
8. Use `testptp -c` to view the PTP hardware clock capabilities on both boards.
9. Use `ethtool -C eth0 tx-frames 1` command to send packet to the stack without any delay on both boards. This must be done before executing `ptp4l` command.
10. Type `ptp4l -i eth0 -m` to start a PTP Master on Board A with the IP address of 10.0.0.1.
11. Type `ptp4l -i eth0 -m -s` to start a PTP Client on Board B with the IP address of 10.0.0.2. The PTP measurement starts at this point.
12. Stop the PTP measurement on Board B (PTP client) by pressing CTRL + C on your keyboard.
13. Type `testptp -f 0` to reset the MAC clock period default to 6.4ns on Board B (PTP client) after the measurement has stopped.
14. Optionally, type `testptp -s` to reset PTP hardware clock time to system time at PTP client.
15. By default, the PTP Master sends 1 SYNC message per every second. You can create a configuration file to override the default settings of ptp4l. The following code shows the example of ptp4l configuration file to set PTP Master to send 512 SYNC messages every second. This setting provides a timestamp accuracy of 6.99ns for every 100 samples.

```
[global]
logSyncInterval -9
```

16. Type `ptp4l -i eth0 -m -f <filename>` to re-start the PTP Master board with the new configuration file.

The following figure shows the example PTP4L measurement log. The average timestamp accuracy demonstrated in this reference design is calculated based on the following equation:

Average timestamp accuracy = sum of max offset / number of data samples

Note: where max offset and number of data samples are only referring to the high values observed during the PTP measurement.

Figure 18: Example of PTP4L Measurement Log

```

root@socfpga:~# ptp4l -i eth0 -m -s
ptp4l[1311.810]: selected /dev/ptp0 as PTP clock
ptp4l[1311.863]: driver changed our HWTSTAMP options
ptp4l[1311.864]: tx_type 1 not 1
ptp4l[1311.864]: rx_filter 1 not 12
ptp4l[1311.864]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[1311.864]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[1312.239]: port 1: new foreign master e66140.ffff.8d7735-1
ptp4l[1316.239]: selected best master clock e66140.ffff.8d7735
ptp4l[1316.239]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[1317.066]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[1317.444]: rms 927213906538 max 1165580873180 freq -1703 +/- 2376 delay 6 +/- 0
ptp4l[1318.482]: rms 233 max 352 freq +144 +/- 973
ptp4l[1319.507]: rms 41 max 98 freq +946 +/- 8 delay 6 +/- 0
ptp4l[1320.536]: rms 3 max 8 freq +945 +/- 8 delay 7 +/- 0
ptp4l[1321.577]: rms 1 max 4 freq +944 +/- 6 delay 7 +/- 0
ptp4l[1322.614]: rms 1 max 4 freq +944 +/- 6 delay 7 +/- 0
ptp4l[1323.647]: rms 1 max 4 freq +944 +/- 7 delay 7 +/- 0
ptp4l[1324.672]: rms 1 max 3 freq +944 +/- 6
ptp4l[1325.694]: rms 1 max 3 freq +944 +/- 6 delay 7 +/- 0
ptp4l[1326.731]: rms 2 max 10 freq +944 +/- 8 delay 7 +/- 0
ptp4l[1327.752]: rms 1 max 4 freq +944 +/- 6
ptp4l[1328.777]: rms 2 max 10 freq +944 +/- 8 delay 9 +/- 0
ptp4l[1329.803]: rms 1 max 4 freq +944 +/- 6 delay 6 +/- 0
ptp4l[1330.816]: rms 1 max 3 freq +944 +/- 6 delay 7 +/- 0
ptp4l[1331.833]: rms 2 max 9 freq +944 +/- 8
ptp4l[1332.851]: rms 1 max 3 freq +944 +/- 6 delay 8 +/- 0
ptp4l[1333.864]: rms 1 max 3 freq +944 +/- 6 delay 7 +/- 0
ptp4l[1334.881]: rms 1 max 3 freq +944 +/- 7 delay 7 +/- 0
ptp4l[1335.898]: rms 1 max 3 freq +944 +/- 7
ptp4l[1336.916]: rms 1 max 4 freq +944 +/- 7 delay 6 +/- 0
ptp4l[1337.941]: rms 1 max 4 freq +944 +/- 6 delay 6 +/- 0
ptp4l[1338.974]: rms 1 max 4 freq +944 +/- 6 delay 9 +/- 0
ptp4l[1339.987]: rms 1 max 3 freq +944 +/- 6 delay 7 +/- 0
ptp4l[1341.005]: rms 1 max 4 freq +944 +/- 7 delay 7 +/- 0
ptp4l[1342.026]: rms 1 max 3 freq +944 +/- 6 delay 6 +/- 0
ptp4l[1343.063]: rms 2 max 10 freq +944 +/- 8 delay 6 +/- 0
ptp4l[1344.096]: rms 1 max 4 freq +944 +/- 7 delay 7 +/- 0
ptp4l[1345.134]: rms 1 max 3 freq +944 +/- 6 delay 8 +/- 0
ptp4l[1346.171]: rms 2 max 10 freq +944 +/- 8 delay 7 +/- 0
ptp4l[1347.184]: rms 1 max 3 freq +944 +/- 7 delay 7 +/- 0
ptp4l[1348.225]: rms 1 max 4 freq +944 +/- 7
ptp4l[1349.262]: rms 2 max 9 freq +944 +/- 8 delay 6 +/- 0
ptp4l[1350.303]: rms 2 max 9 freq +944 +/- 8 delay 6 +/- 0
ptp4l[1351.340]: rms 2 max 9 freq +943 +/- 8 delay 7 +/- 0
ptp4l[1352.373]: rms 1 max 4 freq +944 +/- 7 delay 6 +/- 0
ptp4l[1353.399]: rms 1 max 3 freq +944 +/- 6 delay 6 +/- 0

```

Related Information

[PTP4L manual page](#)

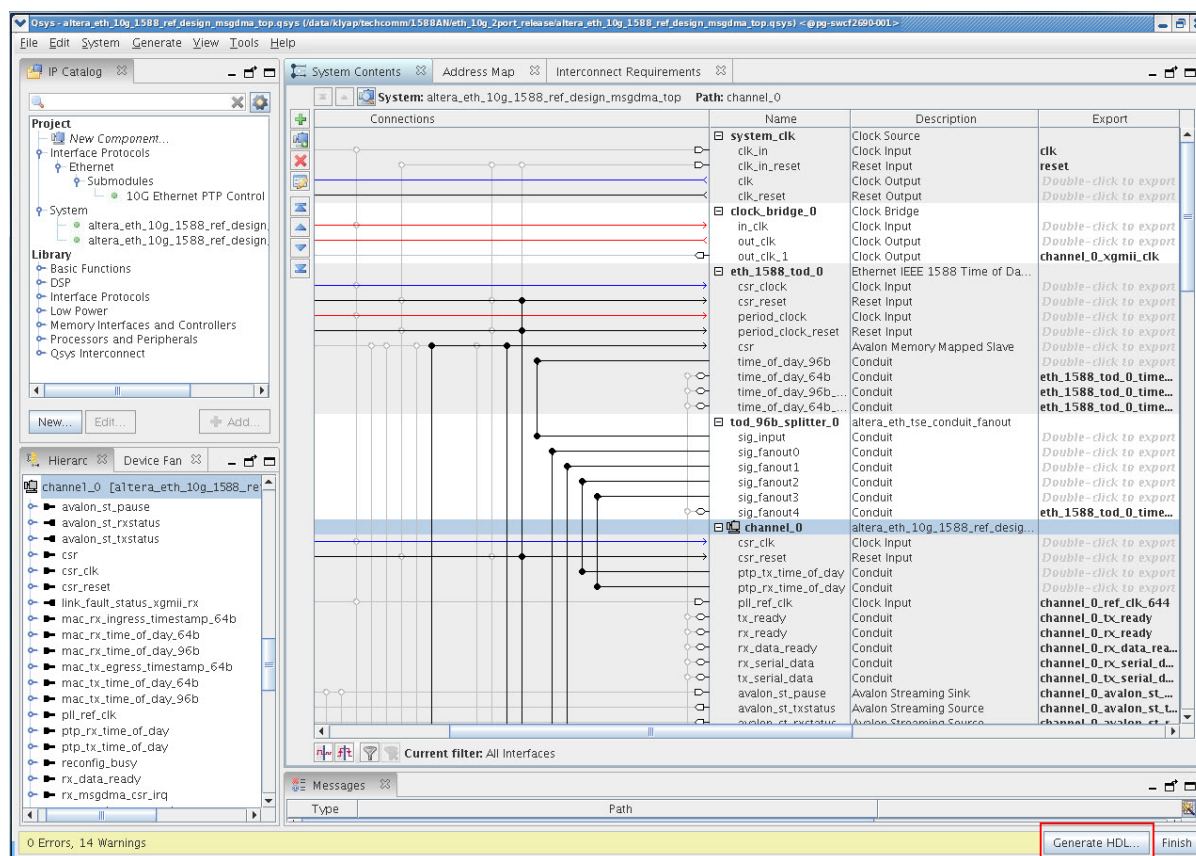
Replacing RBF File

At times, you may need to modify the reference design according to your system requirements and recompile the reference design. You are required to regenerate the `altera_eth_10g_1588_ref_design_msgdma_top.qsys` file for compilation. Following are the steps to guide

you in regenerating the design files and replacing the RBF file into the system image. You are required to use Quartus II Subscription Edition v14.1 for these procedures.

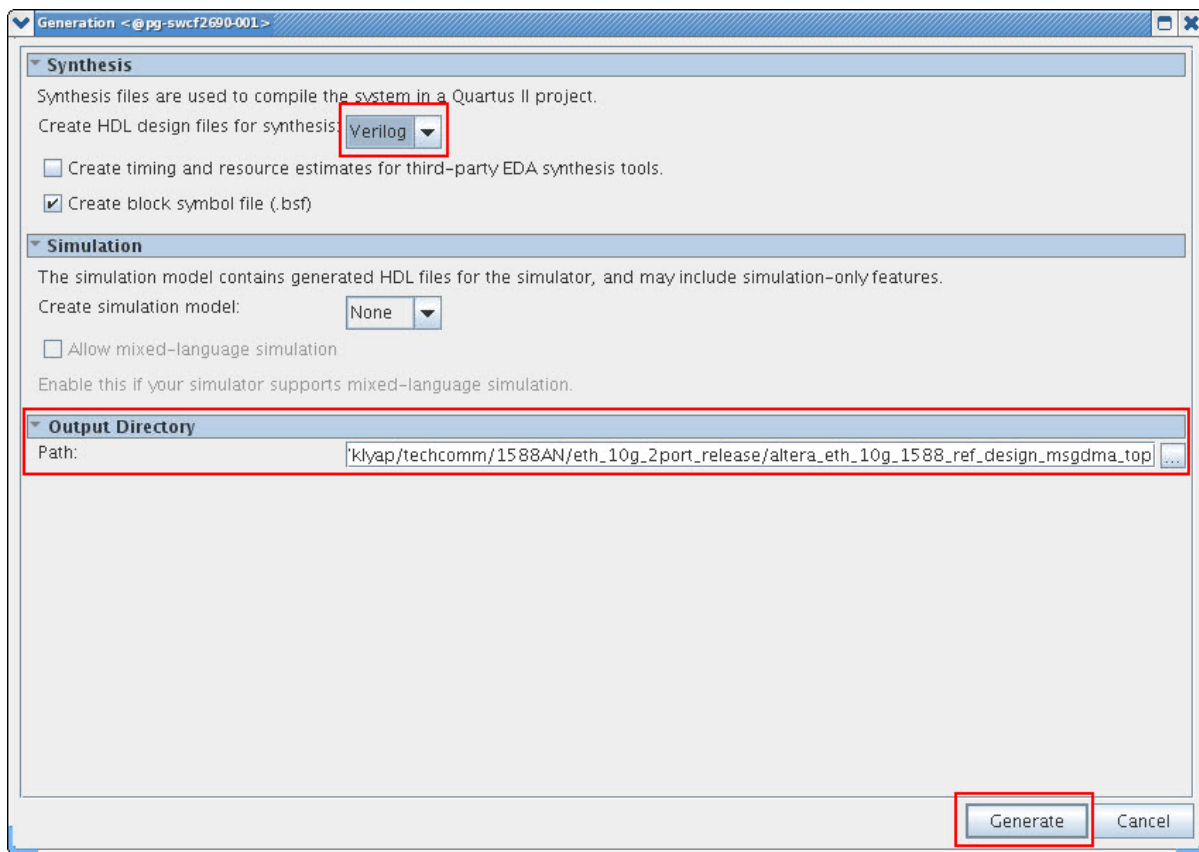
1. Download the alt_eth_1588_rd.qsys.zip file to your computer and unzip the folder.
2. Launch Quartus II software. In Quartus II software, click **File -> Open Project** and browse to **av_1588_top.qpf** file to open the reference design project.
3. Once the the project file is open, click on **Tools -> Qsys** in the Quartus II software.
4. In the Qsys interface, click **File -> Open** and browse to **altera_eth_10g_1588_ref_design_msgdma_top.qsys** file to open the reference design Qsys file.
5. Once the reference design Qsys file is opened, click **Generate HDL**.

Figure 19: Generating HDL File in Qsys



6. You will see the **Generation** dialog box appear. You may choose **Verilog** or **VHDL** for synthesis file type.
7. Browse to the location where you want to store the generated output files under the **Output Directory** and click **Generate**. Click **Close** once the files generation completed.

Figure 20: Generation Dialog Box



8. In Quartus II software, click **Processing** -> **Start Compilation** to recompile the reference design. You will see **av_1588_top.sof** file in the **output_files** folder once compilation is completed.
9. Create a new text file titled **option.txt** with the following content.

```
bitstream_compression=on
```
10. At the command prompt in Linux or DOS, type `quartus_cpf -o <project_dir>/option.txt -c av_1588_top.sof soc_system.rbf` to generate the .rbf (file name tag) file.
11. You can replace the **soc_system.rbf** file in the SD card with the newly generated RBF file.

Related Information

- [AN739 Reference Design Image File](#)
- [AN739 Reference Design Files](#)

System Architecture

This section provides information on the modules used in the reference design, hardware interface signals, registers map, clock and reset scheme of the reference design.

System Modules

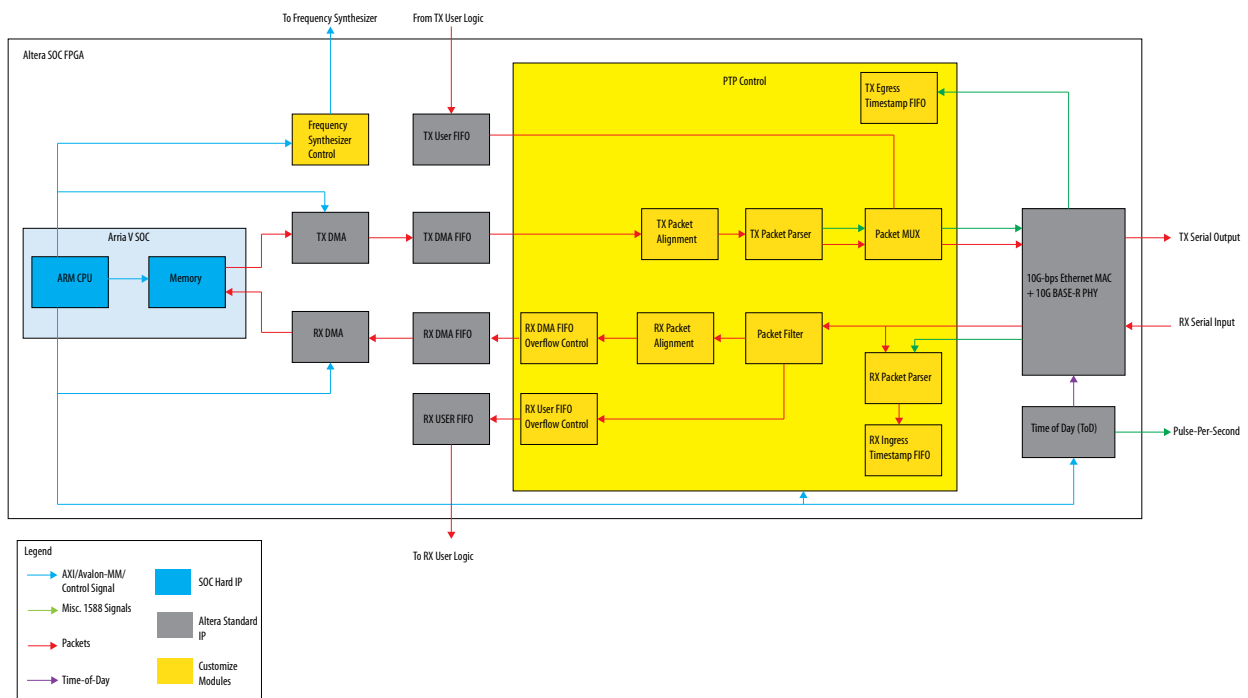
Below are the modules available in the Altera 1588 system solution reference design:

- A 1588 capable Ethernet MAC and PHY, capable of very precise time stamping of packets with ToD value and an indication on the time-stamp offset. This time stamping achieves equal accuracy in both 1-step and 2-step operations.
- A ToD clock module that supports loading of real ToD value and fine-grain update of its value and frequency.
- Clear-text packet parser that can be used to detect PTP packet types and then tell the Ethernet MAC the time-field offset for the following packet types: UDP over IPv4, UDP over IPv6, stacked VLAN. This block is made available in clear-text so that the user can easily augment the code to cover other packet types the users may need such as MPLS or MAC-in-MAC.
- ToD synchronizer to synchronize different ToDs running in different clock domains; for example, in a system of network line cards to a system master ToD.

These building blocks can be put together in a useful system combined with the user provided CPU and software stack to create a high-quality 1588 solution. It is the responsibility of the software stack, which the user either creates or obtains from a third-party, to implement the overall 1588 stack, including the corresponding logic to support different modes for synchronization process.

The following diagram illustrates the hardware modules in this 1588 system reference design.

Figure 21: 1588 Hardware Modules



MAC

The MAC transmit block adds the PHY transmit path delay to the timestamp, places the net timestamp value at the byte offset given by the packet parser and recalculates the CRC for the packet before the packet is sent to the PHY. The 1588 MAC receive block record the timestamp of the incoming packet, subtracts the PHY receive path delay from the timestamp and sends the net timestamp value to the user logic.

Related Information

10Gbps Ethernet MAC Megacore Function User Guide

Time of Day (ToD)

Altera provides a Time of Day Clock (ToD) module to ease user's implementation when using Altera Ethernet MAC-based 1588 system.

The ToD module is a counter responsible for generating the current real time of day locally. A processor with a Timing Servo Control Software updates the ToD counter, typically via fine-grain adjustments in terms of phase and frequency corrections through relevant registers. The ToD can provide both a 64-bit time useful for the correction field used in a transparent clock (TC) and a full 96-bit time useful for ordinary clock (OC) and boundary clock (BC).

You can instantiate a master ToD module to time-stamp the transmitted and received packets. One ToD module can be shared across multiple MACs. You must enable the 64-bit timestamp when using TC, otherwise use 96-bit timestamp. You may use Altera Ethernet IEEE 1588 TOD Synchronizer to synchronize multiple slave ToDs to a single master ToD.

Packet Parser

The Ethernet Packet Classifier functions as a packet parser module for Altera Ethernet MAC-based 1588 system. In the transmit direction, the Packet Parser module is used to determine the type of packet being sent, determine the appropriate location for the time stamp field and provide this information via a command to the 1588 MAC transmitter for further processing. The module is also able to provide the information of the timestamp offset and whether there is an update to the IPv6 UDP correction field.

In the receive direction, this block checks whether the local timestamp of the packet received on the Ethernet MAC receiver is indeed for a PTP packet. For example, the 1588 PTP packet's timestamp field location relative to the Start Of Packet (SOP) is different for a PTP packet encapsulated over Ethernet versus for a PTP packet encapsulated over UDP over IPv6 over Ethernet.

The Central Processing Unit (CPU)

The CPU is responsible for running timer servo control software, PTP stack, Ethernet driver software and appropriate control logic and filtering to update the ToD in addition to the 1588 network management software.

FIFO

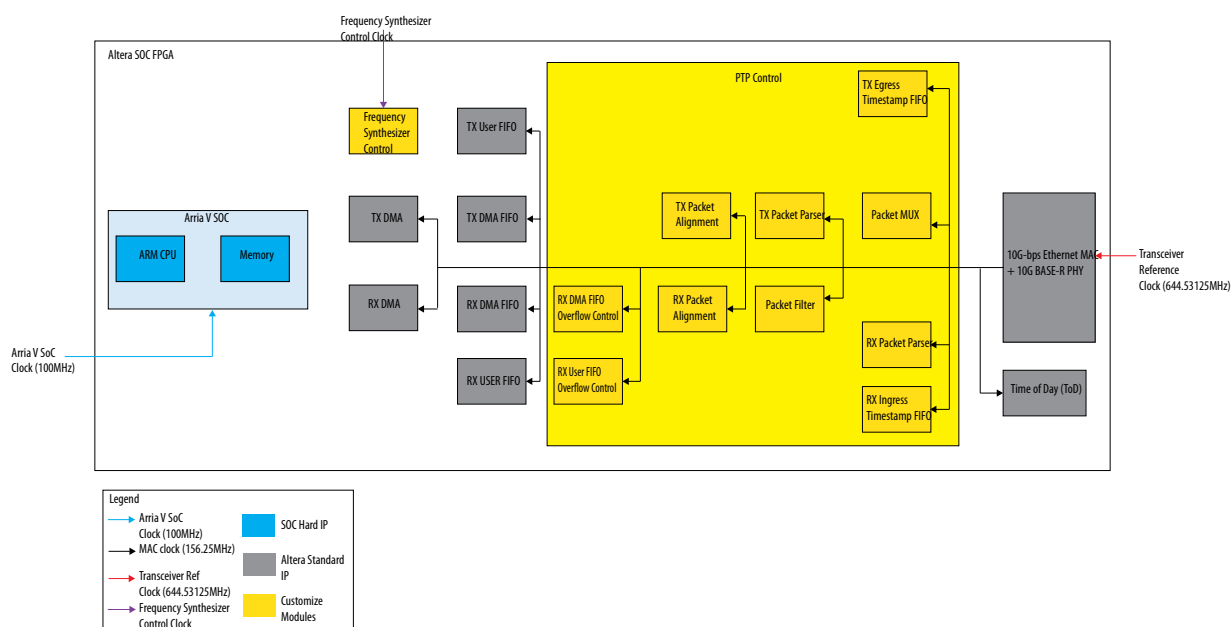
A FIFO is used to associate a PTP event with a timestamp for later retrieval. For example, in a 2-step operation, the FIFO will be used to remember the time that a packet has gone out so that it can be looked up and retrieved at the time that the CPU is ready to send out the follow-up response. Though the 1588 protocol is a slow protocol with 128 frames per second, the 1588 packets may arrive in burst or many contexts such as simultaneous flows or many PTP domains can coexist in a single clock device. Hence, a FIFO is sometimes required depending on the system performance requirements in a 2-step operation. Similarly, in 1-step operation, the timestamp T3 and optionally T2 needs to be collected by the PTP stack through a FIFO. The use cases are an Ordinary Clock-Master in 2-step, Boundary Clock-Master in 2-step, Ordinary Clock-Slave in both 1-step and 2-step, Boundary Clock-Slave in both 1-step and 2-step & Transparent Clock (TC) in 2-step. A typical size of the FIFO required is a network parameter, which can approximately range from 64 to 256 entrees.

The FIFO stores the timestamp along with the signature of the packet for the CPU to read the timestamp later. The CPU can match the timestamp's signature with its signature of the packet before accepting the timestamp. If the timestamp is read out of order, the CPU can keep a shadow copy of the entry to match with other signatures and read the FIFO for the packet in context. A FIFO works more efficiently than a CAM the timestamps are read in order.

Clocking and Reset Scheme

The following figure shows the clocking scheme implemented in the reference design. The clock crossing logic between Arria V SoC and 10Gbps Ethernet MAC clock is handled by Qsys. Refer to Clock and Reset Signals for information related to the clock signals available in the reference design.

Figure 22: Reference Design Clocking Scheme



This reference implements the Altera Transceiver PHY Reset Controller IP to handle the Ethernet system reset sequence.

Related Information

- [Clock and Reset Signals](#) on page 42
- [Altera Transceiver PHY IP Core User Guide](#)

System Parameters

This section describes the parameters used for the modules used in the reference design that are instantiated from the Quartus II software IP catalog.

The following table shows the parameter settings in the Altera 10G-bps Ethernet MAC IP for this reference design. Altera recommends keeping the specified parameter settings as set in the project when using the reference design.

Table 1: Parameters Settings for Altera 10G-bps Ethernet MAC IP

Parameter	Setting
Speed	10 Gbps
Datapath option:	TX & RX
Enable preamble pass-through mode	Turn off

Parameter	Setting
Enable priority-based flow control (PFC)	Turn off
Enable supplementary address	Turn off
Enable CRC on transmit path	Turn on
Enable statistics collection	Turn on
Statistics counters:	Memory-based
Enable time stamping	Turn on
Enable PTP one-step clock support	Turn on

The following table shows the parameter settings in the Altera 10GBASE-R PHY IP for this reference design. Altera recommends keeping the specified parameter settings as set in the project when using the reference design.

Table 2: Parameters Settings for Altera 10GBASE-R PHY IP

Parameter	Setting
Number of channels:	1
Mode of operation:	duplex
PLL type:	CMU
Reference clock frequency:	644.53125 MHz
Enable additional control and status pins	Turn off
Enable rx_recovered_clk_pin	Turn off
Enable pll_locked_status port	Turn off
Enable rx_coreclk_in_port	Turn off
Enable embedded reset controller	Turn on
Enable IEEE 1588 latency adjustment ports	Turn on

The following table shows the parameter settings in the Altera Transceiver Reconfiguration Controller IP for this reference design. Altera recommends keeping the specified parameter settings as set in the project when using the reference design.

Table 3: Parameters Settings for Altera Transceiver Reconfiguration Controller IP

Parameter	Setting
Number of reconfiguration interfaces:	2
Optional interface grouping:	Leave blank
Enable offset cancellation	Turn on by default

Parameter	Setting
Enable duty cycle calibration	Turn off
Create optional calibration status ports	Turn off
Enable Analog controls	Turn on
Enable channel/PLL reconfiguration	Turn off
Enable PLL reconfiguration support block	Turn off

The following table shows the parameter settings in the Altera Ethernet IEEE 1588 Time of Day Clock IP for this reference design. Altera recommend keeping the specified parameter settings as set in the project when using the reference design.

Table 4: Parameters Settings for Altera Ethernet IEEE 1588 Time of Day Clock IP

Parameter	Setting
DEFAULT_NSEC_PERIOD:	6
DEFAULT_FNSEC_PERIOD:	0x00006666
DEFAULT_NSEC_ADJPERIOD:	6
DEFAULT_FNSEC_ADJPERIOD:	0x00006666

Related Information

- [10Gbps Ethernet MAC Megacore Function User Guide](#)
- [Altera Transceiver PHY IP Core User Guide](#)

System Register Map

This section list the base address, the register offset and register descriptions for each modules in the reference design.

The following table shows the base address map for each block in the Altera 1588 reference design.

Table 5: Reference Design Block Register Map

Base address	Block
0xFF20_0140	Altera ToD Clock
Channel 0	
0xFF21_0000	Altera 10G-bps Ethernet MAC
0xFF21_8000	Altera 10G BASE-R PHY
0xFF21_8800	PTP Control
0xFF21_8C00	Altera Reconfiguration Bundle
Channel 1	
0xFF22_0000	Altera 10G-bps Ethernet MAC
0xFF22_8000	Altera 10G BASE-R PHY
0xFF22_8800	PTP Control
0xFF22_8C00	Altera Reconfiguration Bundle

MAC Registers

For the description of each MAC register, refer to the 10-Gbps Ethernet MAC Megacore Function User Guide. The address offset in the following tables is in byte, while the register map table in the 10-Gbps Ethernet MAC Megacore Function User Guide is in word.

Note: Altera recommends maintain the register settings for this module as set in the reference design for optimum operation.

Table 6: MAC Register Components and Offset Range

Component	Byte Offset Range
RX Packet Transfer	0x0000:0x00FF
RX Pad/CRC Remover	0x0100:0x01FF
RX CRC Checker	0x0200:0x02FF
RX Packet Overflow	0x0300:0x03FF
RX Preamble Control	0x0400:0x04FF
RX Lane Decoder	0x0500:0x1FFF
RX Frame Decoder	0x2000:0x2FFF
RX Statistics Counters	0x3000:0x3FFF
TX Packet Transfer	0x4000:0x40FF
TX Pad Inserter	0x4100:0x41FF
TX CRC Inserter	0x4200:0x42FF
TX Packet Underflow	0x4300:0x43FF

Component	Byte Offset Range
TX Preamble Control	0x4400:0x44FF
TX Pause Frame Control and Generator	0x4500:0x45FF
TX PFC Generator	0x4600:0x47FF
TX Address Inserter	0x4800:0x5FFF
TX Frame Decoder	0x6000:0x6FFF
TX Statistics Counters	0x7000:0x7FFF

Table 7: MAC Registers for IEEE 1588v2 Feature

Register	Byte Offset
rx_period	0x0440
rx_adjust_fns	0x0448
rx_adjust_ns	0x044C
tx_period	0x4440
tx_adjust_fns	0x4448
tx_adjust_ns	0x444C

Transceiver Registers

For the description of each PHY register, refer to the Altera Transceiver PHY IP Core User Guide. The address offset in the following tables is in byte addressing, while the register map table in the Altera Transceiver PHY IP Core User Guide is in word addressing.

Note: Altera recommends maintain the register settings for this module as set in the reference design for optimum operation.

Table 8: PMA Registers

Byte Offset	Bit	R/W	Name
0x0088		RO	pma_tx_pll_is_locked
0x0110	1	RW	reset_tx_digital
	2	RW	reset_rx_analog
	3	RW	reset_rx_digital
0x0184		RW	phy_serial_loopback
0x0190		RW	pma_rx_set_locktodata
0x0194		RW	pma_rx_set_locktoref
0x0198		RO	pma_rx_is_lockedtodata

Byte Offset	Bit	R/W	Name
0x019C		RO	pma_rx_is_lockedtoeref
0x02A0	0	RW	tx_invpolarity
	1	RW	rx_invpolarity
	2	RW	rx_bitreversal_enable
	3	RW	rx_bytereversal_enable
	4	RW	force_electrical_idle
0x02A4	0	R	rx_syncstatus
	1	R	rx_patterndetect
	2	R	rx_rlv
	3	R	rx_rmfifoatainserted
	4	R	rx_rmfifoatadeleted
	5	R	rx_disperr
	6	R	rx_errdetect

Table 9: PCS Registers

Byte Offset	Bit	R/W	Name
0x0200		RW	Indirect_addr
0x0204	2	RW	RCLR_ERRBLK_CNT
	3	RW	RCLR_BER_COUNT
0x0208	1	RO	HI_BER
	2	RO	BLOCK_LOCK
	3	RO	TX_FULL
	4	RO	RX_FULL
	5	RO	RX_SYNC_HEAD_ERROR
	6	RO	RX_SCRAMBLER_ERROR
	7	RO	Rx_DATA_READY

PTP Control Registers

The following table shows the list of names, offsets and description for all the registers available in the PTP Control module.

Note: Altera recommends maintain the register settings for this module as set in the reference design for optimum operation.

Table 10: Register Description and Address Offset for PTP Control Module

Word Offset	Bits	Name	Type	Reset Value	Description
0x00	1:0	txrx_pkt_parser_clock_mode	RW	2'b00	Specify the operating clock mode for PTP port. 2'b00: Ordinary Clock 2'b01: Boundary Clock 2'b10: End-to-end Transparent Clock 2'b11: Peer-to-peer Transparent Clock
	8	tx_pkt_parser_two_step_mode	RW	1'b0	Specify the operation mode for the synchronization process. 1'b0: 1-step operation mode 1'b1: 2-step operation mode.
	10	tx_pkt_parser_packet_with_crc	RW	1'b0	Enable the packet parser to indicate whether the incoming packet to the MAC includes CRC. This register is required by TX packet parser to calculate the offset location for checksum corrector for UDP/IPv6 packets.

Word Offset	Bits	Name	Type	Reset Value	Description
0x04	0	rx_pktflt_fwd_user_ucast_match	RW	1'b0	Forward unicast packet that matched the MAC address to user logic. 1'b0: Allow unicast packet which matched the MAC address to be forwarded to user logic. 1'b1: Drop unicast packet that matched the MAC address.
	1	rx_pktflt_fwd_user_ucast_xmatch	RW	1'b1	Forward unicast packet that does not matched the MAC address to user logic. 1'b0: Drop unicast packet that do not match the MAC address. 1'b1: Allow unicast packet which does not match the MAC address to be forwarded to user logic.
	2	rx_pktflt_fwd_user_mcast	RW	1'b1	Forward multicast packet to user logic. 1'b0: Drop multicast packet. 1'b1: Allow multicast packet to be forwarded to user logic.
	3	rx_pktflt_fwd_user_bcast	RW	1'b1	Forward Broadcast packet to user logic. 1'b0: Drop broadcast packet. 1'b1: Allow broadcast packet to be forwarded to user logic.
	8	rx_pktflt_fwd_sw_ucast_match	RW	1'b1	Forward unicast packet that matched the MAC address to software.
	9	rx_pktflt_fwd_sw_ucast_xmatch	RW	1'b0	Forward unicast packet that does not matched the MAC address to software.
	10	rx_pktflt_fwd_sw_mcast	RW	1'b1	Forward multicast packet to software.
	11	rx_pktflt_fwd_sw_bcast	RW	1'b1	Forward Broadcast packet to software.

Word Offset	Bits	Name	Type	Reset Value	Description
0x05	31:0	rx_pktflt_mac_addr_prim_31to0	RW	32'h0	6-byte primary MAC address. You must map the address to the registers in the following manner: <ul style="list-style-type: none"> rx_pktflt_mac_addr_prim_31to0 = Last four bytes of the address rx_pktflt_mac_addr_prim_47to32: First two bytes of the address
0x06	15:0	rx_pktflt_mac_addr_prim_47to32	RW	16'h0	
0x20	0	tx_fifo_clr	RW	1'b0	Reset TX Egress TimeStamp FIFO. 1'b0: De-assert reset. 1'b1: Assert reset.
0x21	0	tx_fifo_ts_fprint_rdy	RO	1'b0	Indicates the availability of the timestamp and the fingerprint. 1'b0: No timestamp and fingerprint available in the TX Egress Timestamp FIFO. 1'b1: Timestamp and fingerprint available in the TX Egress Timestamp FIFO.
	16:8	tx_fifo_used_words	RO	9'b0	Indicates the number of words in the FIFO.

Word Offset	Bits	Name	Type	Reset Value	Description
0x25	31:0	tx_fifo_recovered_timestamp_31to0	RO	32'h0	The recovered timestamp.
0x26	31:0	tx_fifo_recovered_timestamp_63to32	RO	32'h0	For 96-bit timestamp format, all TX FIFO recovered timestamp register-sets are used.
0x27	31:0	tx_fifo_recovered_timestamp_95to64	RO	32'h0	<p>For 64-bit timestamp format, only tx_fifo_recovered_timestamp_63to32 and tx_fifo_recovered_timestamp_31to0 registers are used.</p> <p>Read to tx_fifo_recovered_timestamp_95to64 register indicates the completion of a read transaction for timestamp and fingerprint registers.</p> <p>Altera recommend to follow the below order to read the timestamp and fingerprint registers:</p> <ol style="list-style-type: none"> 1. Read tx_fifo_recovered_fprint_31to0 register (optional). 2. Read tx_fifo_recovered_timestamp_31to0 register. 3. Read tx_fifo_recovered_timestamp_63to32 register. 4. Read tx_fifo_recovered_timestamp_95to64 register.
0x28	19:0	tx_fifo_recovered_fprint_31to0	RW	20'h0	<p>The fingerprint corresponding to the timestamp.</p> <p>Read this register if the PTP stack is require to verify the timestamp correspond to the packet.</p>
0x40	0	rx_fifo_clr	RW	1'b0	<p>Reset RX Ingress TimeStamp FIFO.</p> <p>1'b0: De-assert reset.</p> <p>1'b1: Assert reset.</p>

Word Offset	Bits	Name	Type	Reset Value	Description
0x41	0	rx_fifo_ts_fprint_rdy	RO	1'b0	Indicates the availability of the timestamp and the fingerprint. 1'b0: No timestamp and fingerprint available in the RX Ingress Timestamp FIFO. 1'b1: Timestamp and fingerprint available in the RX Ingress Timestamp FIFO.
	16:8	rx_fifo_used_words	RO	9'b0	Indicates the number of words in the FIFO.
0x45	31:0	rx_fifo_recovered_tstamp_31to0	RO	32'h0	The recovered timestamp.
0x46	31:0	rx_fifo_recovered_tstamp_63to32	RO	32'h0	For 96-bit timestamp format, all RX FIFO recovered timestamp register-sets are used.
0x47	31:0	rx_fifo_recovered_tstamp_95to64	RO	32'h0	For 64-bit timestamp format, only rx_fifo_recovered_tstamp_63to32 and rx_fifo_recovered_tstamp_31to0 registers are used. Read to rx_fifo_recovered_tstamp_95to64 register indicates the completion of a read transaction for timestamp and fingerprint registers. Altera recommend to follow the below order to read the timestamp and fingerprint registers: <ol style="list-style-type: none"> 1. Read rx_fifo_recovered_fprint_31to0 register (optional). 2. Read rx_fifo_recovered_tstamp_31to0 register. 3. Read rx_fifo_recovered_tstamp_63to32 register. 4. Read Rx_fifo_recovered_tstamp_95to64 register.
0x48	19:0	rx_fifo_recovered_fprint_31to0	RW	20'h0	The fingerprint corresponding to the timestamp. Read this register if the PTP stack is require to verify the timestamp correspond to the packet.

1588 ToD Clock Registers

The following table shows the list of names, offsets and description for all the registers available in the Altera Ethernet IEEE 1588 ToD Clock module.

Note: Altera recommends maintain the register settings for this module as set in the reference design for optimum operation.

Table 11: Register Description and Address Offset for 1588 TOD Clock

Byte Offset	R/W	Name	Description	HW Reset
0x0000	RW	SecondsH	<ul style="list-style-type: none"> Bits 0 to 15: High-order 16-bit second field Bits 16 to 31: Not used. 	0x0
0x0004	RW	SecondsL	Bits 0 to 32: Low-order 32-bit second field.	0x0
0x0008	RW	NanoSec	Bits 0 to 32: 32-bit nanosecond field.	0x0
0x0010	RW	Period	<ul style="list-style-type: none"> Bits 0 to 15: Period in fractional nanosecond Bits 16 to 19: Period in nanosecond Bits 20 to 31: Not used. 	N
0x0014	RW	AdjustPeriod	<p>The period for the offset adjustment.</p> <ul style="list-style-type: none"> Bits 0 to 15: Period in fractional nanosecond Bits 16 to 19: Period in nanosecond Bits 20 to 31: Not used. 	0x0
0x0018	RW	AdjustCount	<ul style="list-style-type: none"> Bits 0 to 19: The number of AdjustPeriod clock cycles used during offset adjustment Bits 20 to 31: Not used. 	0x0
0x001C	RW	DriftAdjust	<p>The drift of ToD adjusted periodically by adding a correction value as configured in this register space.</p> <ul style="list-style-type: none"> Bits 0 to 15: Adjustment value in fractional nanosecond (DRIFT_ADJUST_FNS). This value is added into the current ToD during the adjustment. Bits 16 to 19: Adjustment value in nanosecond (DRIFT_ADJUST_NS). This value is added into the current ToD during the adjustment. Bits 20 to 32: Not used. 	0x0
0x0020	RW	DriftAdjustRate	<p>The count of clock cycles for each ToD's drift adjustment to take effect.</p> <ul style="list-style-type: none"> Bits 0 to 15: The number of clock cycles (ADJUST_RATE). The ToD adjustment happens once after every period in number of clock cycles as indicated by this register space. Bits 20 to 32: Not used. 	0x0

Interface Signals

This section lists the signal interfaces based on the top level file in the reference design.

Clock and Reset Signals

Table 12: Clock and Reset Interface Signals

Signal	Direction	Width	Description
reset_reset_n	input	1	Reset signal for the system reference design. This is asynchronous and active low signal.
fpga_clk_100	input	1	Arria V SoC operating clock.
ref_clk_644_0	input	1	Reference clock for Altera 10GBASE-R PHY for channel 0.
ref_clk_644_1	input	1	Reference clock for Altera 10GBASE-R PHY for channel 1.
clk_644_out	output	1	Reference clock for Altera 10GBASE-R PHY output signal for debug purposes.

PHY Interface Signals

Table 13: PHY Interface Signals

Signal	Direction	Width	Description
rx_serial_0	input	1	RX serial input data for channel 0.
tx_serial_0	output	1	TX serial output data for channel 0.
rx_serial_1	input	1	RX serial input data for channel 1. Note: The current version of the reference design only use channel 0. This signal is for future enhancement.

Signal	Direction	Width	Description
tx_serial_1	output	1	TX serial output data for channel 1. Note: The current version of the reference design only use channel 0. This signal is for future enhancement.
pulse_per_second_0	output	1	Pulse Per Second output signal for channel 0. Connect this signal to oscilloscope to observe the PPS waveform for channel 0.
pulse_per_second_1	output	1	Pulse Per Second output signal for channel 1. Connect this signal to oscilloscope to observe the PPS waveform for channel 1. Note: The current version of the reference design only use channel 0. This signal is for future enhancement.

HPS Signals

Table 14: Arria V HPS Interface Signals

Signal	Direction	Width	Description
HPS DDR3 SDRAM			
hps_memory_mem_a	Output	15	Address bus.
hps_memory_mem_ba	Output	3	Bank address.
hps_memory_mem_ck mem_ck_n	Output	1	Memory clock.
mem_cke	Output	1	Clock enable.

Signal	Direction	Width	Description
mem_cs_n	Output	1	Chip select..
mem_ras_n	Output	1	Row address strobe.
mem_cas_n	Output	1	Column address strobe.
mem_we_n	Output	1	Write enable.
mem_reset_n	Output	1	Reset
mem_dq	Bidirectional	40	Data.
mem_dqs	Bidirectional	5	Data strobe.
mem_dqs_n	Bidirectional	5	Data strobe.
mem_odt	Output	1	On-die termination.
mem_dm	Output	5	Data mask.
oct_rzqin	Input	1	OCT reference resistor pins for RZQ.

HPS Peripheral

hps_uart0_TX	Output	1	Output signal for UART channel 0. This signal is required for serial console communication to host.
hps_uart0_RX	Input	1	Input signal for UART channel 0. This signal is required for serial console communication to host.

Related Information

[Arria V Hard Processor System Technical Reference Manual](#)

Altera 1588 System Solution Document Revision History

Table 15: Document Revision History

Date	Version	Changes
January 2016	2016.01.28	<ul style="list-style-type: none">Updated timestamp accuracy number and equation for timestamp accuracy calculation.Added functional flow description for transparent clock and boundary clock mode.Corrected figure Functional Flow For 1588 Ordinary Clock in Master & Slave Mode.
September 2015	2015.09.02	Added links to reference design files.
May 2015	2015.05.05	Initial release.