TimeSlave - User Guide

Atomic Rules LLC

Document Revision / Status

2019-09-02 - Creation
2019-09-05 - Fix up Overview to reflect new single-component rendering of TimeSlave
2019-09-16 - Added description of the TimeSlave sub-cores
2019-10-16 - Added detail on the MicroBlaze embedded processor, TimeSlave Port Changes
2019-10-26 - Added detail on probing the Alveo U200 Testpoint
2019-10-28 - Added Register Descriptions for the sub-cores
2019-11-28 - Added DPLL Parameters
2019-12-06 - Added API Details
2019-12-26 - Added Test 04 Parameters and Nominal Results
2020-01-10 - Added Test Configuration Information
2021-11-03 - Remove Non-Pertinent information for User Guide

Abbreviations

ACAP - Adaptive Compute Acceleration Platform
C2H - Card to Host - Network Packet Ingress
CDC - Clock Domain Crossing
DPLL - Digital Phase Locked Loop
H2C - Host to Card - Network Packet Egress
IPI - IP Integrator, Vivado
PFD - Phase-Frequency Detector
PPS - Pulse Per Second
PTP - Precision Time Protocol (IEEE-1588)
SOP - Start of Packet
TCXO - Temperature Compensated Crystal Oscillator
TOD - Time of Day

Normative References

[1] ARM AMBA AXI4 Specification (Interface Protocols)
[2] IEEE 1364-2001 (Verilog RTL)
[3] IEEE 1588-2008 (Precision Time Protocol)
[4] Atomic Rules TimeServo User Guide (Sub-Core of TimeSlave)

Non-Normative References

[5] https://en.wikipedia.org/wiki/Precision_Time_Protocol
[6] NIST Technical Note 1867 - Time-Aware Applications, Computers and Comm Sys (TAACCS)

Preface

This document describes the Atomic Rules TimeSlave core. The audience includes technical users and potential users of the TimeSlave core.

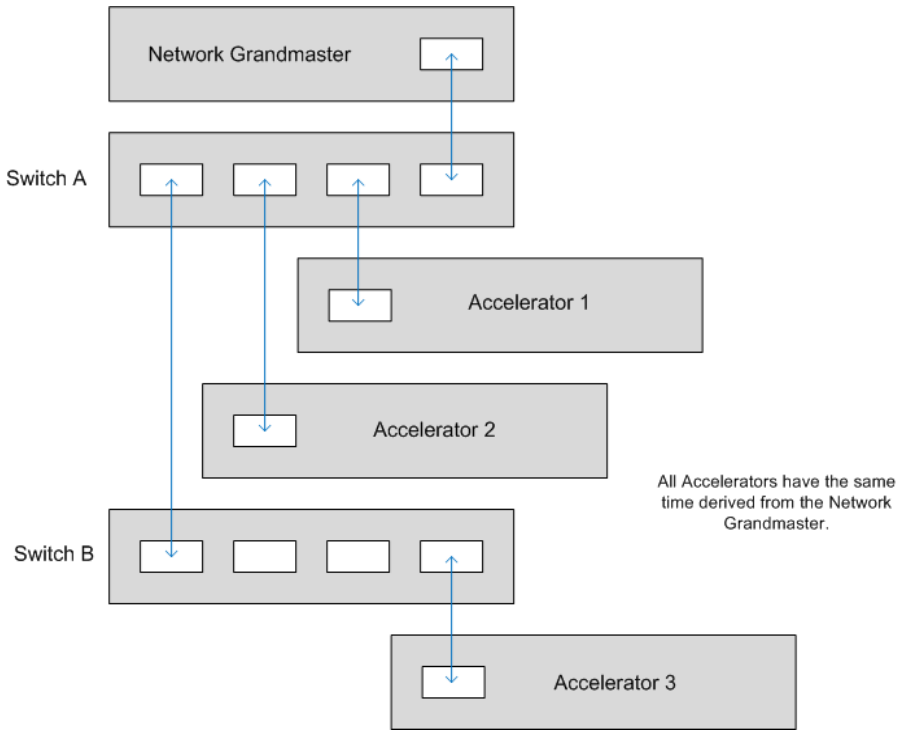
Table of Contents

Section 1 - Overview and Product Description

ACAP devices, either in the datacenter or at the edge, often need to know “what time is it?” The challenge of distributing coherent time *within* an ACAP device is addressed by the Background Technology in TimeServo from Atomic Rules. But how does a physically distinct ACAP, such as an Alveo accelerator card, stay in step with a network grandmaster? And furthermore, how would a plurality of ACAP accelerator cards all stay in sync with that same network grandmaster such that every board’s notion of “now” is the same, even when the network routes to each card may differ? These are exactly the problems that TimeSlave solves.

TimeSlave is a Xilinx IPI Component which implements and respects the IEEE 1588v2 PTP protocol as a Slave device. It periodically interacts with a network grandmaster using either a 1-step or 2-step algorithm in conjunction with timestamp logic in the ACAPs CMAC. Integral to TimeSlave is TimeServo as a sub-core. While the TimeServo sub-core acts as a timekeeper; the rest of TimeSlave acts to implement a protocol-based time-transfer engine. That is, to periodically consume and produce network messages that interact with the timestamping logic on the CMAC locally, and the network time grandmaster remotely.

The diagram below shows three physically distinct ACAP devices on a network with different path lengths from a network time master. TimeSlave IP in each ACAP allows each accelerator to be synchronized to the same value of “now”, to an accuracy of better than one microsecond, often much better.[1]



[1] Performance measurement is covered in detail in subsequent sections. While there are many factors in play, an easy way to visualize a bounds on this process is to understand that a 1 PPM TCXO on each accelerator card could drift as much as 1 microsecond over the interval of a 1 second periodic update. Other factors will contribute to make that figure worse, few if any, will make it better. That is why you can not simply use a 1 PPB (Parts-Per-Billion) TCXO and expect nanosecond accuracy in the holdover.

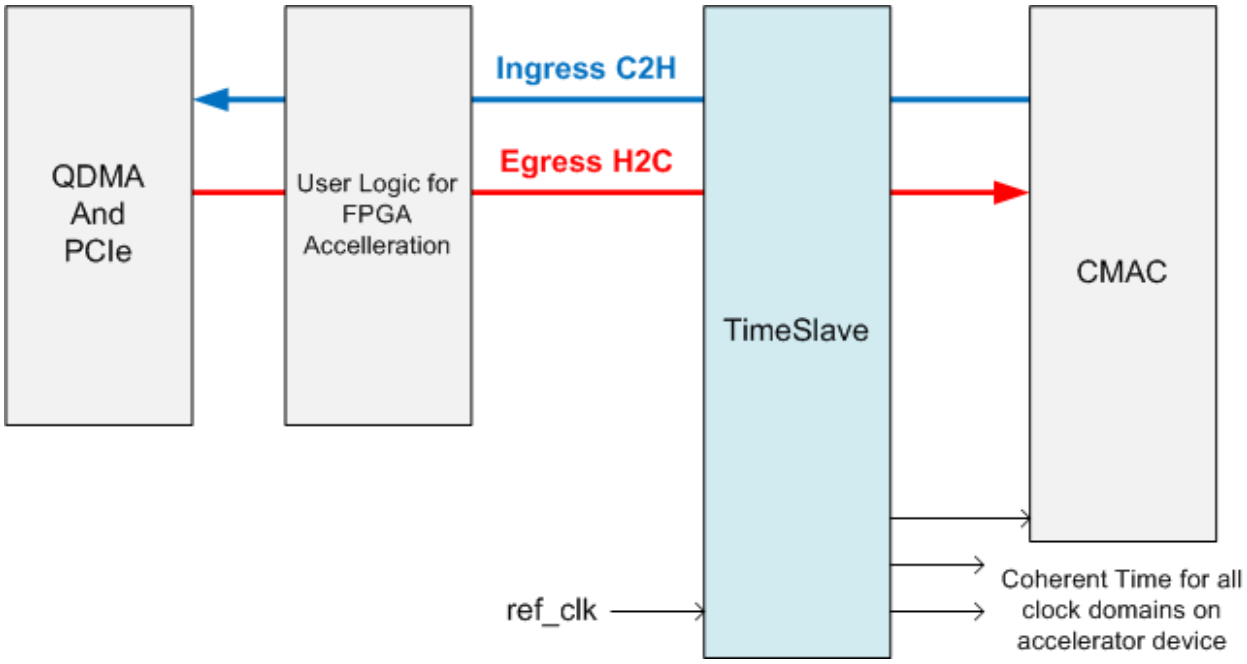
Continuing our top-down overview we look now at some key elements inside the FPGA/ACAP on each accelerator card so that we can later focus on the individual hardware and software APIs through which TimeSlave operates.

The diagram below shows a simplified version of the TimeSlave interfaces and how they relate to other IP on the accelerator card. The TimeSlave component is instantiated in the “qap” block directly adjacent to the CMAC wrapper. This is done in part to minimize latency jitter for the PTP packets. When the TimeSlave component is in a disabled or faulted state, C2H and H2C packets move through TimeSlave unmodified. In order to intercept ingress packets dedicated to PTP and insert egress packets for the grandmaster, internal to the TimeSlave component there is an *ingress packet filter* and *egress packet merge*r. Both these are implemented as cut-through routed, and not store-and-forward, so as to constrain added system latency to a few clock cycles and not a packet’s buffer residency time.

Although TimeSlave is designed to operate autonomously; an AXI4-Lite control-plane interface is provided for initialization, operational status and debug.

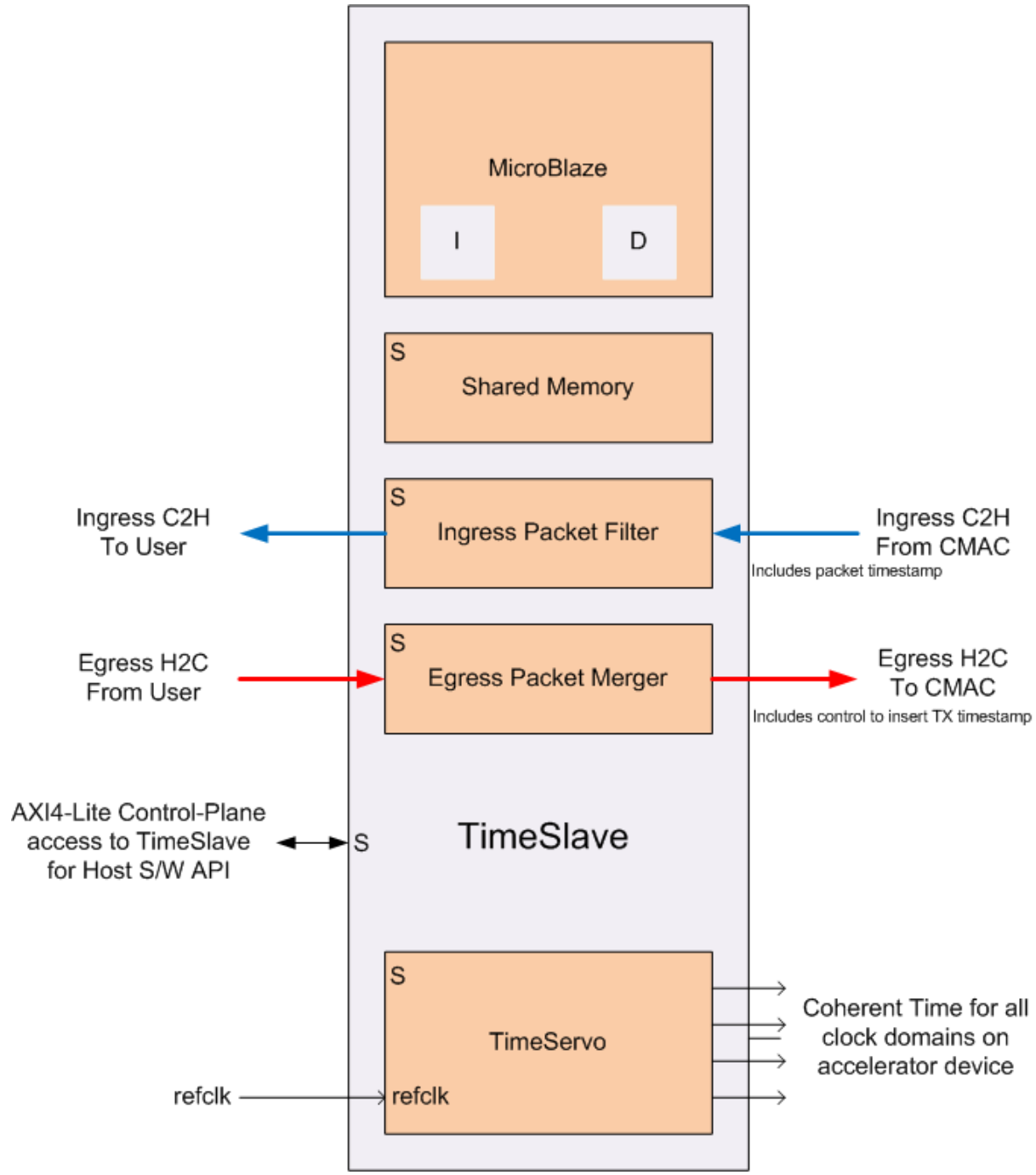
TimeSlave can create time outputs in upto 32 different clock domains. This includes feeding the CMAC timestamping logic with the time in a TOD format required to correctly timestamp both RX and TX timestamps.

If there exists a special high-stability TCXO clock for the purpose of timekeeping, it may be fed to refclk, otherwise the most appropriate clock can be used as a reference. The stable 300 MHz clock often used as a reference for DDR controllers is satisfactory.



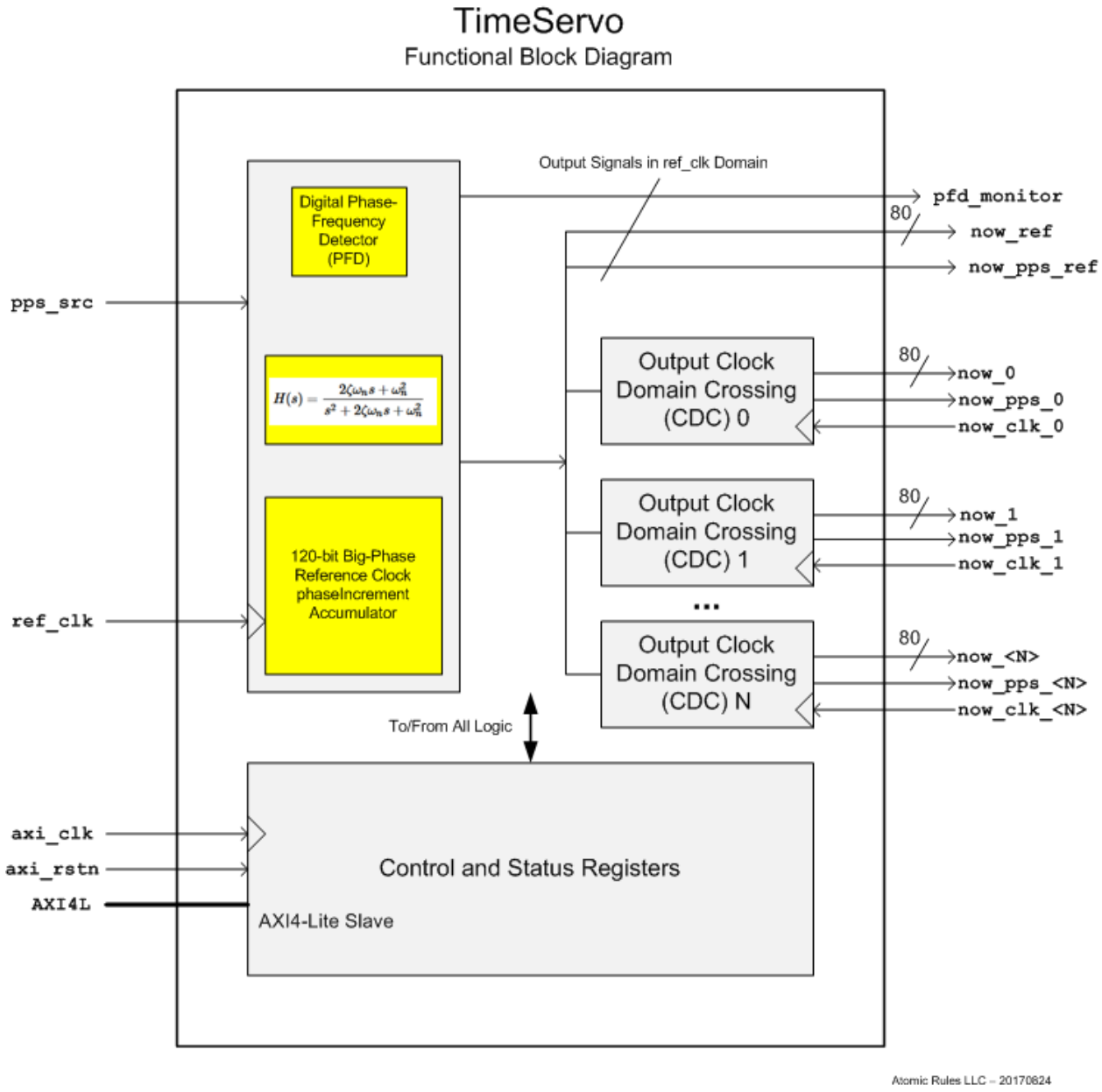
The diagram above is not representative of logic-area scale.

Continuing now on to the overview of the TimeSlave component itself. The diagram below shows a simplified view of TimeSlave and some of the significant internal function units it contains. An understanding at this level or lower is not essential to using TimeSlave; but it is to understand the theory of operation. TimeSlave uses a lightweight bare-metal OS configuration of a MicroBlaze processor to periodically communicate with the network grandmaster, periodically adjust the time, and periodically communicate with the host API. Most of these periodic operations are asynchronous and happen on the order of once per second.



The diagram above is not representative of logic-area scale.

Lastly, for continuity we show the simplified block diagram of the TimeServo sub-core instanced inside of TimeSlave. For more details on TimeServo see [4]. Note that the Pulse Per Second signal “pps_src” is not required or used with TimeSlave as the network time grandmaster, and not a physical 1 PPS signal, is used as the slave time TOD phase reference.



The diagram above is not representative of the logic area scale.

TimeSlave Features - Data Sheet Bullets

- An IEEE 1588v2 PTP Ordinary Clock (OC) Slave Implementation for FPGA/ACAP
- Supports both 1-Step and 2-Step synchronization with an external network time grandmaster
    - TimeSlave Delay Requests are 1-Step using MAC TX Hardware Time Insertion
- End to End (E2E) Delay Mechanism (not P2P)
- Single-component solution for providing coherent time within an FPGA/ACAP
- Communicates with PTP Master via Ethernet L2 PTP/1588 EtherType frames
- Flexible and independent clocks for control-plane and reference clock
- Four (4) time “now” outputs with Clock Domain Crossing (CDC) Logic
    - Each in their own Clock Domain from user-supplied clock
    - Each individually selectable 80b output format (Binary, IEEE Ordinary, IEEE Transparent)
    - Each with a Pulse Per Second (PPS) output pulse in output clock domain
- Software control and observability from AXI control plane
- Following Initialization, no interaction from host is required
- Atomic Rules implementation of a Gardner Type-2 Digital Phase Locked Loop (DPLL)
    - Double-Precision Floating-Point implementation
    - Sample Rate F_{sample}= 1 Hz
    - Nyquist Rate $F_{Nyquist}$= 0.5 Hz
    - Damping (zeta) ξ= 1.0
    - Noise Bandwidth ω_{BW}= 0.1 Hz
    - Natural Frequency ω_{n}= 0.025 Hz
    - Tau τ= 10 Seconds
- FPGA Resources Used (Approximate)
    - LUTs: 13K
    - BRAMs: 17
    - URAMs: 2
    - DSPs: 6

TimeSlave Simplified Functional Description

Following from the Overview section which describes what TimeSlave does, this section describes how TimeSlave does it.

Following FPGA configuration, the ACAP is released from reset and user logic may drive and respond to signals from the TimeSlave component. It is required that all clocks are stable before releasing reset. Network connectivity is not required at this early stage.

The AXI4-L Control plane interface is used for accessing TimeSlave control and status information. Following the release of AXI reset, the control and status registers may be accessed to control and observe the state of the module. See the Register Description and Software API sections for additional details.

In order for TimeSlave to keep time, there must be a stable `ref_clk` signal applied from the accelerator card platform. The source of `ref_clk` is external to TimeSlave. It is common to use an on-chip PLL to generate a nominal `ref_clk` of 300 MHz from whatever clock is the most-stable oscillator available to the device, if a stable 300 MHz soursew is not available. A TCXO is fine. A reference frequency from an atomic clock is excellent. Sources with any form of frequency modulation, such as spread-spectrum, must be avoided for `ref_clk`.

In the user design, the 1-32 “now” outputs of time are connected to their user-design loads in their respective clock domains. Each output’s CDC module has an input that the user drives with the clock for that output. One or more of these outputs will be connected to the CMACs timestamping logic inputs to facilitate both RX and TX packet timestamping.

At this point with AXI4-L Control Plane, Reference Clock applied, and Outputs connected; TimeSlave can be used in open-loop NO_PTP mode as a system timer. Time can be set, trimmed, and made to run faster or slower entirely under software control of the host API without any network connection.

Once the CMAC is initialized with a link to its link partner the embedded processor interchanges messages with the grandmaster and implements a servo algorithm in firmware compliant with 1588v2/PTP.

The steps to keep time are

- Bring TimeSlave out of reset
- Perform the initialization sequence
- Query (Poll) the registers to determine when a grandmaster has been found
- TimeSlave is now running. It can be read by software and the 4 hardware “now” ports
- Optionally, Query (Poll) the registers for diagnostics

Section 2 - Port Descriptions

This section describes the RTL signals which make up the interface to TimeSlave.

Table 2.1 - **Signals for AXI4-L Control Plane and MicroBlaze Clock and Reset**

Signal Name	Direction	Description
s_axi_aclk	In	Control Plane Clock [1]
s_axi_aresetn	In	Control Plane Reset, Active-Low, Synchronous

[1] axi_clk: Nominally 100 MHz; any stable, known-frequency above 50 MHz that meets timing.

Table 2.2 - **Signals for Time Keeping Reference**

Signal Name	Direction	Description
ref_clk	In	Reference Clock [2]

[2] ref_clk: Nominally 300 MHz; any stable, known-frequency above 100 MHz that meets timing.

Table 2.3 - **Signals for Observing Time in User Supplied Clock Domains**

Signal Name	Direction	Description
now_clk_<*N*>	In	Clock Domain *N* to Observe Time In
now_pps_<*N*>	Out	Pulse Per Second (PPS) Output [4]
now_<*N*>[79:0]	Out	“Now” Time Output, 80 bit (see text for formats)

[4] now_pps_<N>: Rising edge is the reference PPS out, this signal is asserted for 16 (sixteen) now_clk_<*N*> cycles.

Table 2.4 - **Signals for Observing Time in the Reference Clock Domain [7]**

Signal Name	Direction	Description
now_pps_ref	Out	Pulse Per Second (PPS) Output [5]
now_ref[79:0]	Out	“Now” Time Output, 80 bit (see text for formats) [6]

[5] now_pps_ref: Rising edge is the reference PPS out, this signal is asserted for 16 (sixteen) ref_clk cycles.
[6] The time format of the now_ref output is set by bits[15:14] of the ts_timeControl register.
[7] The now_pps_ref and now_ref[] outputs are exactly two ref_clk periods delayed from the internal reference.

Table 2.5 - **Signals for Clocking the Packet Processors [8]**

Signal Name	Direction	Description
axis_clk	In	Clock for all AXI-Stream Ports [9]
axis_rstn	In	Reset for all AXI-Stream Ports [10]

[8] The axis_clk provides the clock for the four AXI Stream interfaces: CMAC-Facing C2H Ingress, User-Facing C2H Ingress, CMAC-Facing H2C Egress, and User-Facing H2C Egress. As well the axis_clk is used as the processor clock for the MicroBlaze and other packet-centric internal circuitry to avoid CDC where possible.

[9] The axis_clk shall be a stable 322.265625 MHz. This is typically identical to the CMAC TX `userclk2` for 100 GbE.

[10] This is an AXI compliant active-low reset in the clock domain of axis_clk. It must remain asserted until axis_clk is stable.

Table 2.5 - **Signals for PTP/1588v2 to/from CMAC[8]**

Signal Name	Direction	Description
tx_1588_op	Out	Operation presented to the CMAC, the CMAC samples this on tx_axis_tvalid
ptp_rx_tstamp[79:0]	In	Timestamp of the received packet SOP (Valid during first tvalid signal of a packet)
ptp_tx_tstamp[79:0]	In	Timestamp of the transmitted packet SOP
ptp_tx_tstamp_valid	In	Indicates that a ptp_tx_tstamp is valid

Port Groups

The signal names of the core are prefixed with names which indicate to which port group they belong. Port groups contain bundles of signals that are functionally related. The table below lists the four port groups of TimeSlave.

Table 1.2 - Table of TimeSlave Port Groups

Port Group	Prefix	Function
Control Plane	`s_axi`	Core Control as an AXI4Lite Slave
Ingress C2H from MAC	`mac_s_axis`	C2H Ingress Stream from MAC to TimeSlave
Egress H2C to MAC	`mac_m_axis`	H2C Egress Stream to MAC from TimeSlave
Ingress C2H to USER	`usr_m_axis`	C2H Ingress Stream to User from TimeSlave
Egress H2C from USER	`usr_s_axis`	H2C Egress Stream from User to TimeSlave

Unless stated otherwise, each of the port groups are made up of one or more AXI channels with fully-guarded, bi-directional flow control. Data advances on a channel *only* when the source's VALID and the sink's READY are both asserted True in the same cycle.

TimeSlave Control Plane Signals

The TimeSlave Control Plane (CP) Port Group signals are listed in the following table. All the signals in this group begin with the prefix `s_axi` .The encoding of these signals complies with the AXI4 specification and protocol. Note there are 20 bits of address used to provide the 1 MByte address space that TimeSlave requires. See the next section “Register Description” to understand the internal state accessed through this control-plane interface.

Table 1.3 - TimeSlave Control Plane Signals

Signal Name	Direction	Description
`s_axi_awvalid`	In	AXI Write Address Channel Valid
`s_axi_awready`	Out	AXI Write Address Channel Ready
`s_axi_awaddr[19:0]`	In	AXI Write Address Channel Write Address (1 MB)
`s_axi_awprot[2:0]`	In	AXI Write Address Channel Write Protection
`s_axi_wvalid`	In	AXI Write Data Channel Valid
`s_axi_wready`	Out	AXI Write Data Channel Ready
`s_axi_wdata[31:0]`	In	AXI Write Data Channel Data
`s_axi_wstrb[3:0]`	In	AXI Write Data Channel Data Strobe
`s_axi_bvalid`	Out	AXI Write Response Channel Valid
`s_axi_bready`	In	AXI Write Response Channel Ready
`s_axi_bresp[1:0]`	Out	AXI Write Response Channel Response Code
`s_axi_arvalid`	In	AXI Read Address Channel Valid
`s_axi_arready`	Out	AXI Read Address Channel Ready
`s_axi_araddr[19:0]`	In	AXI Read Address Channel Read Address (1 MB)
`s_axi_arprot[2:0]`	In	AXI Read Address Channel Read Protection
`s_axi_rvalid`	Out	AXI Read Response Channel Valid
`s_axi_rready`	In	AXI Read Response Channel Ready
`s_axi_rdata[31:0]`	Out	AXI Read Response Channel Read Data
`s_axi_rresp[1:0]`	Out	AXI Read Response Channel Read Response Code

MAC Facing Signals

The signals interfacing the MAC Facing Ingress and Egress streams are listed in the following two tables. These two interfaces are how TimeSlave communicates with the MAC. The encoding of these signals complies with the AXI4-Stream specification and protocol. The 64 Bytes of data on TDATA corresponds to 64 consecutive octets on the wire.

Table 1.4 - MAC Facing C2H Ingress Stream Signals - `mac_s_axis`

Signal Name	Direction	Description
`mac_s_axis_tvalid`	In	C2H Ingress AXI Stream MAC Facing Valid
`mac_s_axis_tready`	Out	C2H Ingress AXI Stream MAC Facing Ready
`mac_s_axis_tdata[511:0]`	In	C2H Ingress AXI Stream MAC Facing Data
`mac_s_axis_tkeep[63:0]`	In	C2H Ingress AXI Stream MAC FacingKeep
`mac_s_axis_tlast`	In	C2H Ingress AXI Stream MAC Facing Last

The signal `mac_saxis_tuser` carries the RX timestamp on the 80b field `[128+80-1:128]` . See the section on Streaming Tuser Signals following for more details.

Table 1.5 - MAC Facing H2C Egress Stream Signals - `mac_m_axis`

Signal Name	Direction	Description
`mac_m_axis_tvalid`	Out	H2C Egress AXI Stream MAC Facing Valid
`mac_m_axis_tready`	In	H2C Egress AXI Stream MAC Facing Ready
`mac_m_axis_tdata[511:0]`	Out	H2C Egress AXI Stream MAC Facing Data
`mac_m_axis_tkeep[63:0]`	Out	H2C Egress AXI Stream MAC Facing Keep
`mac_m_axis_tlast`	Out	H2C Egress AXI Stream MAC Facing Last

The signal `mac_maxis_tuser` carries the TX time insertion bit on bit 129. See the section on Streaming Tuser Signals following for more details.

The following two ports provide the timestamp of the departure time of PTP packets that leave the TX MAC port.

User Facing Signals

The signals interfacing the User Facing Ingress and Egress streams are listed in the following two tables. These two interfaces are how TimeSlave communicates with the User logic. The encoding of these signals complies with the AXI4-Stream specification and protocol. The 64 Bytes of data on TDATA corresponds to 64 consecutive octets on the wire.

Table 1.5 - User Facing C2H Ingress Stream Signals - `usr_m_axis`

Signal Name	Direction	Description
`usr_m_axis_tvalid`	Out	C2H Ingress AXI Stream User Facing Valid
`usr_m_axis_tready*`	In	C2H Ingress AXI Stream User Facing Ready
`usr_m_axis_tdata[511:0]`	Out	C2H Ingress AXI Stream User Facing Data
`usr_m_axis_tuser[79:0]`	Out	C2H Ingress AXI Stream User Facing User
`usr_m_axis_tkeep[63:0]`	Out	C2H Ingress AXI Stream User Facing Keep
`usr_m_axis_tlast`	Out	C2H Ingress AXI Stream User Facing Last

* C2H Back pressure will be passed from the user-facing to the CMAC-facing interface. Note that the CMAC does not allow RX/ingress backpressure. If ingress back pressure support is required, appropriate buffering must be added between timeslave and user logic.

Table 1.4 - User Facing H2C Egress Stream Signals - `usr_s_axis`

Signal Name	Direction	Description
`usr_s_axis_tvalid`	In	H2C Egress AXI Stream User Facing Valid
`usr_s_axis_tready`	Out	H2C Egress AXI Stream User Facing Ready
`usr_s_axis_tdata[511:0]`	In	H2C Egress AXI Stream User Facing Data
`usr_s_axis_tkeep[63:0]`	In	H2C Egress AXI Stream User Facing Keep
`usr_s_axis_tlast`	In	H2C Egress AXI Stream User Facing Last

Streaming TUSER Signals

One of the preceding port groups (`usr_m_axis`) is a AXI Stream interface with TUSER signals.

User Facing TUSER Bits

Table 1.4 - User Facing C2H Ingress Stream Signals - `usr_m_axis`

Signal Name	Direction	Description
`usr_m_axis_tuser[79:0]`	Out	Timestamp of current packet

This 80 bit TUSER field contains the RX/ingress timestamp of the current packet. User application logic may pick up this signal to know at what time this packet arrived.

Section 3 - Register Descriptions

TimeSlave's Control Plane (CP) interface allows read/write access to state within the core. This section describes the memory-mapped register resources accessible through the CP. In the column labeled "Type" below a value of 4 means a 4-Byte (e.g. `UInt32_t`) and a value of 8 means an 8-Byte (e.g. `UInt64_t`) type. See accompanying text for further details of individual bit fields.

On access "Type" (column on table on next page):

4: These storage locations are logically 32 bit, 4B, little-endian stores. There are no Byte enables to permit access to less than DWORD (4B) granularity. All addresses are aligned on 0x4 Byte bounds.
Observation: This is a single-data transfer cycle over a 32-bit AXI4 interface.

8: These storage locations are logically 64 bit, 8B, little-endian stores. A 4B write to the LS *stages* the lower 32b, while the subsequent 4B write to the MS *commits* the entire 64b. A 4B read to the LS returns the LS and stages the MS, the subsequent 4B read of the MS returns the staged value. There are no Byte enables to permit access to less than 2-DWORD (8B) granularity. All addresses are aligned on 0x8 Byte bounds.
Observation: This is a two data-transfer cycles over a 32-bit AXI4 interface. Since the AXI4-Lite subset does not allow burst INCR, a processor de-referencing `* UInt64_t wfoo` to store a value would generate two sequential stores at address `wfoo` (for the LS 32-bits) and `wfoo+0x4` (for the MS 32-bits).

16: These storage locations are logically 128bit, 16B, little-endian stores. A 4B write to offset 0x0 *stages* the lower 32b, another 4B write to offset 0x4 stages the next 32b, another 4B write to offset 0x8 stages the next 32b, finally the subsequent 4B write to the MS *commits* the entire 128b. A 4B read to the LS returns the LS and stages the MS, the subsequent 4B reads of the MS returns the staged value. There is no logic enabling access to less than 8-DWORD (16B) granularity. All addresses are aligned on 0x10 Byte bounds.
Observation: This is four data-transfer cycles over a 32-bit AXI4 interface.

64: This a 64 Byte cache-line aligned data structure with special read-access side-effects. The only way to read this data structure is to either perform 16 sequential 4B accesses or 8 sequential 8B accesses. In either case, a special-behavior happens with read side-effect on the first read at offset zero: The first read freezes all 64B of the data structure at the time of the first read so that all 64B are coherent. This behavior allows all 64B returned to have come from the same axi_clk cycle sample, without any uncertainty, latency, or jitter due to the time between successive AXI reads. Software processing the 64 Bytes retrieved in this manner can rely on the fact that all the data was captured on the same `axi_clk` cycle; even though loading it will have taken 10s, 100s or (over PCIe) possibly 1000s of cycles.
Observation: Reading the `ts_coherentState` data structure in this manner provides software a view of 64 Bytes of TimeSlave state as if there was a 64B cache-line atomic read operation.

Summary of TimeSlave Sub-Core Address Regions

TimeSlave is composed of various sub-cores, some of them, such as TimeServo have their own normative and complete datasheets and register descriptions. Where the registers are not fully-described here, the table below describes their relative offsets.

This table shows how the 18 bit address space is segmented:

Table 2.1 - TimeSlave Sub-Core Offset Summary

Offset	Sub-Core or Region	Size	Function
0x0_0000	TimeSlave Reserved	128KB	Reserved - No User Access (text/data/bss for uBlaze)
0x3_1000	Egress Merger	4KB	Egress Merger Sub-Core
0x3_2000	Ingress Filter	4KB	Ingress Filter Sub-Core
0x3_3000	TimeServo	4KB	TimeServo Sub-Core
0x3_4000	MDM	4KB	MicroBlaze Debug Module
0x2_0000	Shared Memory	32KB	Shared Memory for MicroBlaze and User Communication

Summary of TimeSlave Control Registers

Table 2.1 - Control Plane Register Summary

Offset	Name	Type	Access	Function
0x3_1000	egress_id	4	RO	EXPECT_TSE_ID = 0x564c5354
0x3_1004	egress_version	4	RO	EXPECT_TSE_VER = 0x302e3145
0x3_1008	egress_config	4	RW	
0x3_1010	egress_status	4	RO	
0x3_1014	egress_send	4	WO	Write this register with packet length in Bytes to Send
0x3_1018	egress_control	4	RW	
0x3_1020	egress_pktCount	4	RO	Rolling count of packets making egress
0x3_1024	egress_insCount	4	RO	Rolling count of packets inserted
0x3_1030	egress_ts_status	4	RO	0x1 indicates captured TX Timestamp
0x3_1034	egress_ts_nsec	4	RO	TX Timestamp nanoseconds (32b)
0x3_1038	egress_ts_sec	8	RO	TX Timestamp seconds (48b)
0x3_1200	egress_tuser	16	WO	128b TUSER to MAC. Must have len in [11:0], set bit 64
0x3_1400	egress_packets	512	WO	Egress insert packet buffer (up to 512B)
0x3_2000	ingress_id	4	RO	EXPECT_TSI_ID = 0x564c5354
0x3_2004	ingress_version	4	RO	EXPECT_TSI_VER = 0x302e3149
0x3_2010	ingress_status	4	RO	
0x3_2014	ingress_pkt_index	4	RO	Pointer to where the last received packet is [0-7]
0x3_2018	ingress_control	4	RW	
0x3_201C	ingress_ack_pkt	4	WO	
0x3_2020	ingress_pktCount	4	RO	Rolling count of packets making ingress
0x3_2024	ingress_ptpCount	4	RO	Rolling count of PTP packets captured
0x3_2028	ingress_ptpDrop	4	RO	Rolling count of PTP packets dropped (buf overflow)
0x3_202C	ingress_ptpError	4	RO	Rolling count of PTP errors.
0x3_2200	ingress_stamps	512	RO	Circular Buffer of 8 PTP RX timestamps

0x3_2400	ingress_packets	512	RO	Circular Buffer of 8 PTP RX packets (64B per packet)
0x3_3XXX	timeservo_	4K	RW	See Atomic Rules TimeServo User Guide
0x3_4XXX	UART	4K	RW	See Xilinx UART User Guide
0x9_XXXX	Shared Memory	16K	RW	See Shared Memory Description

Section 4 - Detailed Functional Description

This section describes the detailed functional behavior of TimeSlave. Clauses in this section describe specific features, behaviors, control register usage and internal operations required for common use-cases.

Reset Behavior

Hardware Reset

Hardware reset is accomplished by asserting the *active-low* `axi_resetn` signal. This reset signal is synchronous and must be asserted and deasserted at the rising edge of the clock, `axi_clock`. The minimum assertion time is 1 clock cycle. Asserting reset brings TimeSlave to a known state for normal operation. Note that reset does not clear all memory buffers and registers; just the internal state necessary to guarantee correct and deterministic operation. Prior data may remain in some memory locations (e.g. DFFs, BRAMs, and other stateful elements). Secure clearing of this data may be accomplished by removing power from the FPGA.

Software Reset

Portions of TimeSlave may be reset via software (i.e.. through the use of control plane writes).

Section 5 - Functional Simulation

Unit testing of TimeSlave and it's sub cores are limited to sub-module unit tests. The entire TimeSlave code may be simulated with VCS within the Xilinx simulation environment.

Section 6 - Example Design

The vivado project is designed to be the simplest possible design with TimeSlave and the ultrascale+ CMAC. It contains 1 IPI block design that shows how to connect the various busses between the cmac and the TimeSlave. It also demonstrates how to connect the clocks and resets.

To get this example running do the following steps

1. Create and open the project with `make proj`
2. Build the design by clicking on "Generate Bitstream"
3. Program the design on the board
4. Connect the ethernet to a network that also has a grand master on it
5. Run `xsdb ../../common/xsdb/run_example.tcl` to initialize the mac and display timeslave statistics.

If all goes well you should see output on the console that looks like this:

```
Date: Wed Dec 31 19:00:02 EST 1969
Epoch Second : 2
Fractional Seconds : 65536
Last  PTP Offset  (ns) 0
Mean  PTP Offset  (ns) 0.0
Sigma PTP Offset  (ns) 0.00

Date: Tue Nov 02 19:01:48 EDT 2021
Epoch Second : 1635894108
Fractional Seconds : 752812032
Last  PTP Offset  (ns) 1499075828

Date: Tue Nov 02 19:01:49 EDT 2021
Epoch Second : 1635894109
Fractional Seconds : 752746496
Last  PTP Offset  (ns) 1499075828
```

After the timeslave "locks" onto the grandmasters clock (PTP offset <100ns) then more statistics start being displayed:

```
Date: Tue Nov 02 19:03:02 EDT 2021
Epoch Second : 1635894182
Fractional Seconds : 752680960
Last  PTP Offset  (ns) 71
Mean  PTP Offset  (ns) 28.0
Sigma PTP Offset  (ns) 36.22
```

In addition to printing this information on the screen, the data is also saved to a file called “time.log” in the form of `time_seconds frac_seconds last_delta`. This is useful for doing data analysis on it.

Section 7 - Time Output Formats

In any format, “Now” represents a positive time with respect to the epoch.

Pulse Per Second Output

Each CDC output will generate a 1-bit PPS output signal in the user-supplied CDC output clock domain that is a pulse which goes high for sixteen cycles following a time crossing of each one-second top-dead-center crossing of bigPhase. The rising edge of this pulse is the PPS event. It is asserted for 16 cycles as a design and debugging visibility convenience.

Output Formats Available

There are three available 80-bit output formats selectable individually for each CDC Output block:

- Binary Time 48.32 Format - Top 80-bits of bigPhase in binary seconds 48.32 format
- IEEE Ordinary Format - 48-bits of seconds and 32-bits of integer nanoseconds per IEEE spec
- IEEE Transparent Format - Binary nanoseconds in 48.16 format per IEEE spec

Binary Time 48.32 Format

In “Binary Time” time output mode, the time format is output as a 80-bit unsigned number with the binary point for the seconds located to the right of bit 32. The 80b ordinary type `NowBinary_T` represents a positive time with respect to the epoch expressed in integer seconds and binary seconds.

```
struct NowBinary_T
{
  UInteger48 secondsField;
  UInteger32 fractSecondsField;
};
```

The secondsField member is the integer portion of NowBinary_T in units of seconds.
The fractSecondsField member is the fractional portion of NowBinary_T in units of fractional seconds.

For example:
+2.000000001 seconds is represented in Ordinary Mode by:
secondsField = 0000 0000 0002_{16} and fractSecondsField= 0000 0004_{16}.
See table on next page for detail

Table of Bits [34:0] from Binary Time Format

Bits [79:35] not shown to save space on the page.

Bit	Value in Seconds	Base 10	Description
34	2^{2}	4.000 000 000	(4 s)
33	2^{1}	2.000 000 000	(2 s)
32	2^{0}	1.000 000 000	(1 s)
31	2^{-1}	0.500 000 000	(0.5 s)
30	2^{-2}	0.250 000 000	(0.25 s)
29	2^{-3}		
28	2^{-4}		
27	2^{-5}		
26	2^{-6}		
25	2^{-7}		
24	2^{-8}		
23	2^{-9}		
22	2^{-10}		
21	2^{-11}		
20	2^{-12}		
19	2^{-13}		
18	2^{-14}		
17	2^{-15}		
16	2^{-16}	0.000 015 258 = $1/2^{16}$	(15.258 us)
15	2^{-17}		
14	2^{-18}		
13	2^{-19}		
12	2^{-20}		
11	2^{-21}		
10	2^{-22}		
9	2^{-23}		
8	2^{-24}		
7	2^{-25}		
6	2^{-26}		
5	2^{-27}		
4	2^{-28}		
3	2^{-29}		
2	2^{-30}	0.000 000 000 931= $1/2^{30}$	(931.322 ps)
1	2^{-31}		
0	2^{-32}	0.000 000 000 232 = $1/2^{32}$	(232.831 ps)

IEEE Ordinary Format

In “ordinary” time output mode, the time format is according to the IEEE 1588 format, with 48 bits for seconds and 32 bits for nanoseconds. The 80b ordinary type `NowOrdinary_T` represents a positive time with respect to the epoch expressed in seconds and nanoseconds.

```
struct NowOrdinary_T
{
  UInteger48 secondsField;
  UInteger32 nanosecondsField;
};
```

The secondsField member is the integer portion of NowOrdinary_T in units of seconds.
The nanosecondsField member is the fractional portion of NowOrdinary_T in units of nanoseconds.
The nanosecondsField member is always less than 10^9.

For example:
+2.000000001 seconds is represented in Ordinary Mode by:
secondsField = 0000 0000 0002_{16} and nanosecondsField= 0000 0001_{16}.

IEEE Transparent Format

In “transparent” time output mode, bit 63 represents the sign bit, bits 62:16 carry nanoseconds, and bits 15:0 carry fractional nanoseconds. The 64b transparent type `NowTransparent_T` represents a positive time with respect to the epoch expressed in signed nanoseconds with 48 bits (including sign) to the left of the nanosecond binary point and 16 bits of fractional nanoseconds to the right of the binary point.

As bit 63 is a sign-bit (0 for positive time), bits [79:64] will include a replica of the sign bit.

```
struct NowTransparent_T
{
  Integer48  signedNanosecondsField;
  Integer16  nanosecondsFractionField;
};
```

For example:
+2.000000001 seconds is represented in Transparent Mode by:
signedNanosecondsField= 0000 7735 9401_{16} and nanosecondsFractionField= 0000 0000_{16}.
signedNanosecondsField= 2 000 000 001_{10} and nanosecondsFractionField= 0000 0000_{16}.

Section 9 - Usage Notes

Relationship of Reference Clock to Output CDC Clocks

TimeSlave contains a digital accumulator with 120 bits of internal precision. To achieve greater accuracy, it is desirable to operate the Reference Clock (ref_clk) at the highest practical frequency. Higher ref_clk frequencies mean that the digital accumulator is updated more often. At a 400 MHz ref_clk frequency the accumulator is updated every 2.5 ns.

There is another benefit to a high frequency ref_clk, which impacts the quality of time for external devices observing time in their own clock domain, target_clk. In cases were

$$F_{ref\_clk} >> F_{target\_clk}$$

It is assured that each output from the Clock Domain Crossing (CDC) will present a new, unique, monotonically increasing time value. However, as ref_clk and target_clk become close in frequency; and certainly for cases where

$$F_{ref\_clk} << F_{target\_clk}$$

There will be samples in the target_clk domain where the observed time repeats for one or more cycles. The CDC is not an interpolator. The times are valid: It is just when you oversample the ref_clk in a target domain, you will see “time repeat”. This may or may not be an issue for your application. Clearly, the differential linearity of time observed in the target domain suffers in this case where the target_clk is faster than the reference_clk.

Some specific time-consuming devices, like MAC timestamp logic, may actually depend on a time input being monotonically increasing to correctly operate. We have observed in our lab the situation of time appearing to go backwards on timestamped packets (even if only tens of nanoseconds) when time is not monotonically increasing. This is not a defect of TimeSlave; but rather a consequence of the time oversampling. In these special cases, consider if ref_clk can me made higher than target_clk.

The non-CDC direct ref_clk output *always* has a perfect 1:1 relationship with the phase accumulator. In special situations that must have the *best possible differential linearity* of the slope of the output time, it is suggested that the ref_clk based direct outputs be used (and the CDC outputs avoided).

Section 11 - IEEE 1588-2008 / PTP Protocol Compliance

The Atomic Rules TimeSlave IP is an IEEE 1588 Ordinary Clock (OC) and Slave Only (SO). It uses a PTP Profile called the “Atomic Rules TimeSlave PTP Profile”.

Atomic Rules TimeSlave PTP Profile

As per IEEE 1588-2008 Clause 15.1.1 - The PTP management mechanism used is “No specified management mechanism”. A combination of fixed-values and implementation-specific means are used to address any configurable variables.

User Test Equipment Recommendations and Configuration

This section details some of the test equipment used as a non-normative convenience for replicating the Atomic Rules test setups used to produce the results herein.

Trimble Thunderbolt PTP GM200 GrandMaster
https://www.trimble.com/Timing/PTP-grandmaster-clock-GM200.aspx

We borrowed one of these units, were pleased with the performance, and so we bought a second unit for ourselves for about $1700 from the online store. The Trimble team keeps the firmware up to date. Be sure to order a 48V power supply and GPS antenna as per your requirements. We routinely see a 6 ns sigma when all satellites are in view. It appears to be an excellent implementation of a PTP GrandMaster, and so it is well-suited for our needs. We use a second identical unit to do statistical experiments on the second-to-second error; but that is a luxury, not a requirement. It is the PTP GM we used in this video: https://youtu.be/_-yyUa7iqmU

Mellanox 1/10/25/50/100 GbE SN2010 Ethernet Switch
https://store.mellanox.com/categories/switches/ethernet-switches/sn2000/sn2010.html

This is a “swiss army knife” for interconnecting Ethernet devices of different line rates. It is too loud for bench work, and at $9600, more of an investment that most need for PPT testing. Atomic Rules has developed a method to, in some cases, obviate the need for this switch. It is listed here for reference only. *This switch is not necessary, although it is desirable, for PTP testing.* At this writing, Mellanox is enhancing the Boundary Clock software in their Onyx Switch OS. This is the Ethernet switch we used in this video: https://youtu.be/_-yyUa7iqmU

Mellanox ConnectX-5 2-Port Ethernet NIC (MCX516A-CCAT)
https://store.mellanox.com/products/mellanox-mcx516a-ccat-connectx-5-en-network-interface-card-100gbe-dual-port-qsfp28-pcie3-0-x16-tall-bracket-rohs-r6.html

This is an excellent 100 GbE NIC with robust and ongoing software support. Note that there are many variations available, including Gen4 PCIe which we do not use yet. At $955, the MCX516A-CCAT is a terrific value; ubiquitous in our lab. There is a PCIe Gen4x16 variant of this card to consider as well. It’s worth noting that after four full years of working Xilinx Ethernet interop issues at 25 GbE and 100 GbE; having a Mellanox link-partner has been observed to be the most-reliable, without PHY-level headaches. This is true for both copper and fibre links. This is the NIC used in this video: https://youtu.be/_-yyUa7iqmU One great feature of the CX-5 NIC is so-called “Hairpin Mode”, described next.

Configuring “Hairpin Mode” on a Mellanox ConnectX-5 NIC

In order to provide connectivity between the 1 GbE interface on the PTP GM and the 100 GbE interface on the DUT, some device has to perform the line rate conversion. This occurs “for free” when using a switch, such as the Mellanox SN2010, or any modern switch, for that matter. But 100 GbE switches are both loud and cost-prohibitive for each bench. “Hairpin Mode”, sometimes also called “Host-Chaining”, is a technique that allows the second port of a Mellanox NIC to be used for this purpose.

With “Hairpin Mode”, packets that arrive that are not destined for the host are identified, and entirely within the Mellanox NIC ASIC sent right back out on the other NIC port. These include L2 Multicast packets such as those used in 1588v2/PTP.

In this way, the Xilinx DUT can plug at 100 GbE into port 0; and the Trimble GM can plug at 1 GbE into port 1, and the two PTP devices can “see” each other, vitaly without any software jitter.

The steps to configure “Hairpin Mode” are as follows.

1. Bring your OS up to date:
   ```
   apt-get update; apt-get upgrade; apt-get dist-upgrade
   ```

2. Bring the NIC drivers and firmware up to date
   https://www.mellanox.com/page/software_overview_eth
   E.g. `mlnx-en-4.7-3.2.9.0-ubuntu16.04-x86_64.tgz`
   Then install, reboot if needed.

3. Install Firmware Tools
   https://www.mellanox.com/page/management_tools
   E.g. `mft-4.13.3-6-x86_64-deb.tgz`
   Then install.

4. Start mst
   ```
   shep@ar-6701k:~/Downloads$ sudo mst start
   Starting MST (Mellanox Software Tools) driver set
   Loading MST PCI module - Success
   Loading MST PCI configuration module - Success
   Create devices
   Unloading MST PCI module (unused) - Success
   ```

5. Query device
   ```
   shep@ar-6701k:~/Downloads$ sudo mlxconfig -d
   /dev/mst/mt4119_pciconf0 query
   ```

```
Device #1:
----------

Device type:    ConnectX5
Name:           MCX516A-CCA_Ax
Description:    ConnectX-5 EN network interface card; 100GbE
dual-port QSFP28; PCIe3.0 x16; tall bracket; ROHS R6
Device:         /dev/mst/mt4119_pciconf0

Configurations:                                 Next Boot
         MEMIC_BAR_SIZE                         0
         MEMIC_SIZE_LIMIT                       _256KB(1)
         HOST_CHAINING_MODE                     FALSE(0)
         HOST_CHAINING_DESCRIPTORS              Array[0..7]
         HOST_CHAINING_TOTAL_BUFFER_SIZE        Array[0..7]
         FLEX_PARSER_PROFILE_ENABLE             0
         FLEX_IPV4_OVER_VXLAN_PORT              0
         ROCE_NEXT_PROTOCOL                     254
         ESWITCH_HAIRPIN_DESCRIPTORS            Array[0..7]
         ESWITCH_HAIRPIN_TOT_BUFFER_SIZE        Array[0..7]
         PF_BAR2_SIZE                           0
         NON_PREFETCHABLE_PF_BAR                False(0)
         VF_VPD_ENABLE                          False(0)
         STRICT_VF_MSIX_NUM                     False(0)
         VF_NODNIC_ENABLE                       False(0)
         NUM_OF_VFS                             0
         PF_BAR2_ENABLE                         False(0)
         SRIOV_EN                               False(0)
         PF_LOG_BAR_SIZE                        5
         VF_LOG_BAR_SIZE                        1
         NUM_PF_MSIX                            63
         NUM_VF_MSIX                            11
         INT_LOG_MAX_PAYLOAD_SIZE               AUTOMATIC(0)
         PARTIAL_RESET_EN                       False(0)
         SW_RECOVERY_ON_ERRORS                  False(0)
         RESET_WITH_HOST_ON_ERRORS              False(0)
         ADVANCED_POWER_SETTINGS                False(0)
         CQE_COMPRESSION                        BALANCED(0)
         IP_OVER_VXLAN_EN                       False(0)
         ESWITCH_IPV4_TTL_MODIFY_ENABLE         False(0)
         PRIO_TAG_REQUIRED_EN                   False(0)
         UCTX_EN                                True(1)
```

```
        PCI_ATOMIC_MODE
PCI_ATOMIC_DISABLED_EXT_ATOMIC_ENABLED(0)
        LRO_LOG_TIMEOUT0                    6
        LRO_LOG_TIMEOUT1                    7
        LRO_LOG_TIMEOUT2                    8
        LRO_LOG_TIMEOUT3                    13
        LOG_DCR_HASH_TABLE_SIZE             11
        DCR_LIFO_SIZE                       16384
        ROCE_CC_PRIO_MASK_P1                255
        ROCE_CC_ALGORITHM_P1                ECN(0)
        ROCE_CC_PRIO_MASK_P2                255
        ROCE_CC_ALGORITHM_P2                ECN(0)
        CLAMP_TGT_RATE_AFTER_TIME_INC_P1    True(1)
        CLAMP_TGT_RATE_P1                   False(0)
        RPG_TIME_RESET_P1                   300
        RPG_BYTE_RESET_P1                   32767
        RPG_THRESHOLD_P1                    1
        RPG_MAX_RATE_P1                     0
        RPG_AI_RATE_P1                      5
        RPG_HAI_RATE_P1                     50
        RPG_GD_P1                           11
        RPG_MIN_DEC_FAC_P1                  50
        RPG_MIN_RATE_P1                     1
        RATE_TO_SET_ON_FIRST_CNP_P1         0
        DCE_TCP_G_P1                        1019
        DCE_TCP_RTT_P1                      1
        RATE_REDUCE_MONITOR_PERIOD_P1       4
        INITIAL_ALPHA_VALUE_P1              1023
        MIN_TIME_BETWEEN_CNPS_P1            2
        CNP_802P_PRIO_P1                    6
        CNP_DSCP_P1                         48
        CLAMP_TGT_RATE_AFTER_TIME_INC_P2    True(1)
        CLAMP_TGT_RATE_P2                   False(0)
        RPG_TIME_RESET_P2                   300
        RPG_BYTE_RESET_P2                   32767
        RPG_THRESHOLD_P2                    1
        RPG_MAX_RATE_P2                     0
        RPG_AI_RATE_P2                      5
        RPG_HAI_RATE_P2                     50
        RPG_GD_P2                           11
        RPG_MIN_DEC_FAC_P2                  50
        RPG_MIN_RATE_P2                     1
        RATE_TO_SET_ON_FIRST_CNP_P2         0
```

```
DCE_TCP_G_P2                        1019
DCE_TCP_RTT_P2                      1
RATE_REDUCE_MONITOR_PERIOD_P2       4
INITIAL_ALPHA_VALUE_P2              1023
MIN_TIME_BETWEEN_CNPS_P2            2
CNP_802P_PRIO_P2                    6
CNP_DSCP_P2                         48
LLDP_NB_DCBX_P1                     False(0)
LLDP_NB_RX_MODE_P1                  OFF(0)
LLDP_NB_TX_MODE_P1                  OFF(0)
LLDP_NB_DCBX_P2                     False(0)
LLDP_NB_RX_MODE_P2                  OFF(0)
LLDP_NB_TX_MODE_P2                  OFF(0)
DCBX_IEEE_P1                        True(1)
DCBX_CEE_P1                         True(1)
DCBX_WILLING_P1                     True(1)
DCBX_IEEE_P2                        True(1)
DCBX_CEE_P2                         True(1)
DCBX_WILLING_P2                     True(1)
KEEP_ETH_LINK_UP_P1                 True(1)
KEEP_IB_LINK_UP_P1                  False(0)
KEEP_LINK_UP_ON_BOOT_P1             False(0)
KEEP_LINK_UP_ON_STANDBY_P1          False(0)
KEEP_ETH_LINK_UP_P2                 True(1)
KEEP_IB_LINK_UP_P2                  False(0)
KEEP_LINK_UP_ON_BOOT_P2             False(0)
KEEP_LINK_UP_ON_STANDBY_P2          False(0)
NUM_OF_VL_P1                        _4_VLs(3)
NUM_OF_TC_P1                        _8_TCs(0)
NUM_OF_PFC_P1                       8
NUM_OF_VL_P2                        _4_VLs(3)
NUM_OF_TC_P2                        _8_TCs(0)
NUM_OF_PFC_P2                       8
DUP_MAC_ACTION_P1                   LAST_CFG(0)
SRIOV_IB_ROUTING_MODE_P1            LID(1)
IB_ROUTING_MODE_P1                  LID(1)
DUP_MAC_ACTION_P2                   LAST_CFG(0)
SRIOV_IB_ROUTING_MODE_P2            LID(1)
IB_ROUTING_MODE_P2                  LID(1)
PCI_WR_ORDERING                     per_mkey(0)
MULTI_PORT_VHCA_EN                  False(0)
PORT_OWNER                          True(1)
ALLOW_RD_COUNTERS                   True(1)
```

```
RENEG_ON_CHANGE                     True(1)
TRACER_ENABLE                       True(1)
IP_VER                              IPv4(0)
BOOT_UNDI_NETWORK_WAIT              0
UEFI_HII_EN                         False(0)
BOOT_DBG_LOG                        False(0)
UEFI_LOGS                           DISABLED(0)
BOOT_VLAN                           1
LEGACY_BOOT_PROTOCOL                PXE(1)
BOOT_RETRY_CNT                      NONE(0)
BOOT_LACP_DIS                       True(1)
BOOT_VLAN_EN                        False(0)
BOOT_PKEY                           0
ATS_ENABLED                         False(0)
DYNAMIC_VF_MSIX_TABLE               False(0)
EXP_ROM_UEFI_x86_ENABLE             False(0)
EXP_ROM_PXE_ENABLE                  True(1)
ADVANCED_PCI_SETTINGS               False(0)
SAFE_MODE_THRESHOLD                 10
SAFE_MODE_ENABLE                    True(1)
```

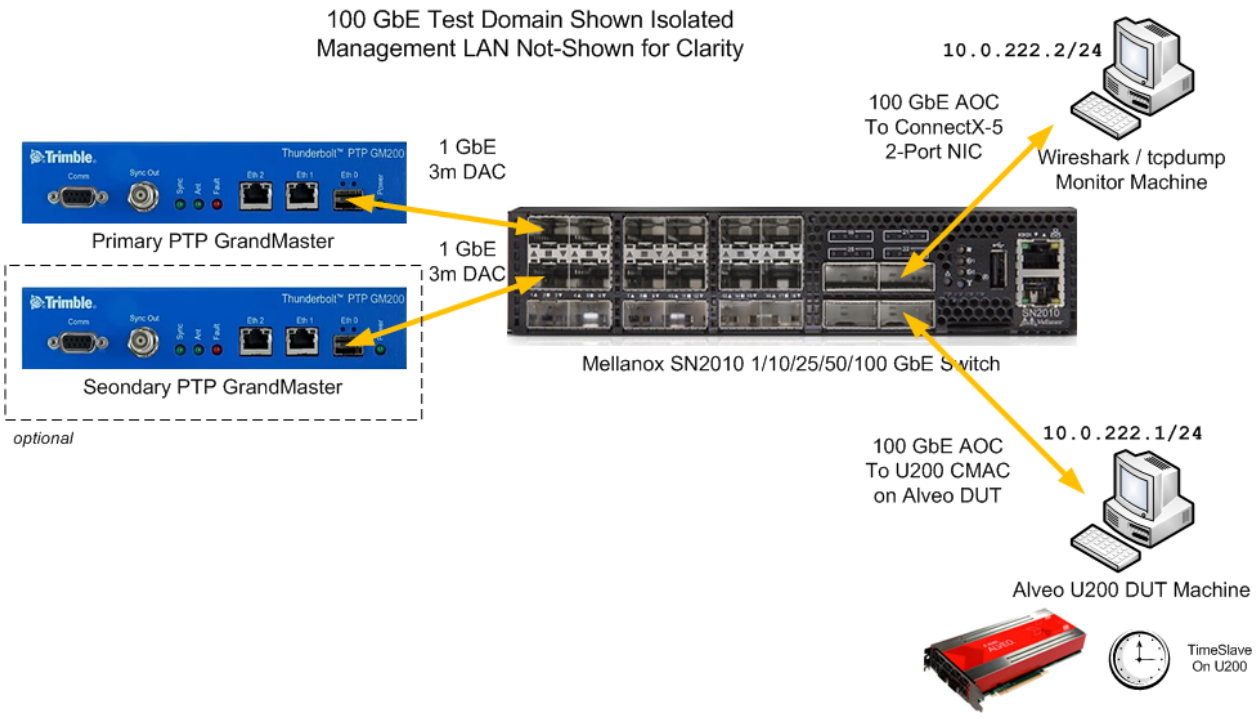
6. Set Host Chaining

```
sudo mlxconfig -d /dev/mst/mt4119_pciconf0 set HOST_CHAINING_MODE=1
```

7. Query Again

```
HOST_CHAINING_MODE                  BASIC(1)
```

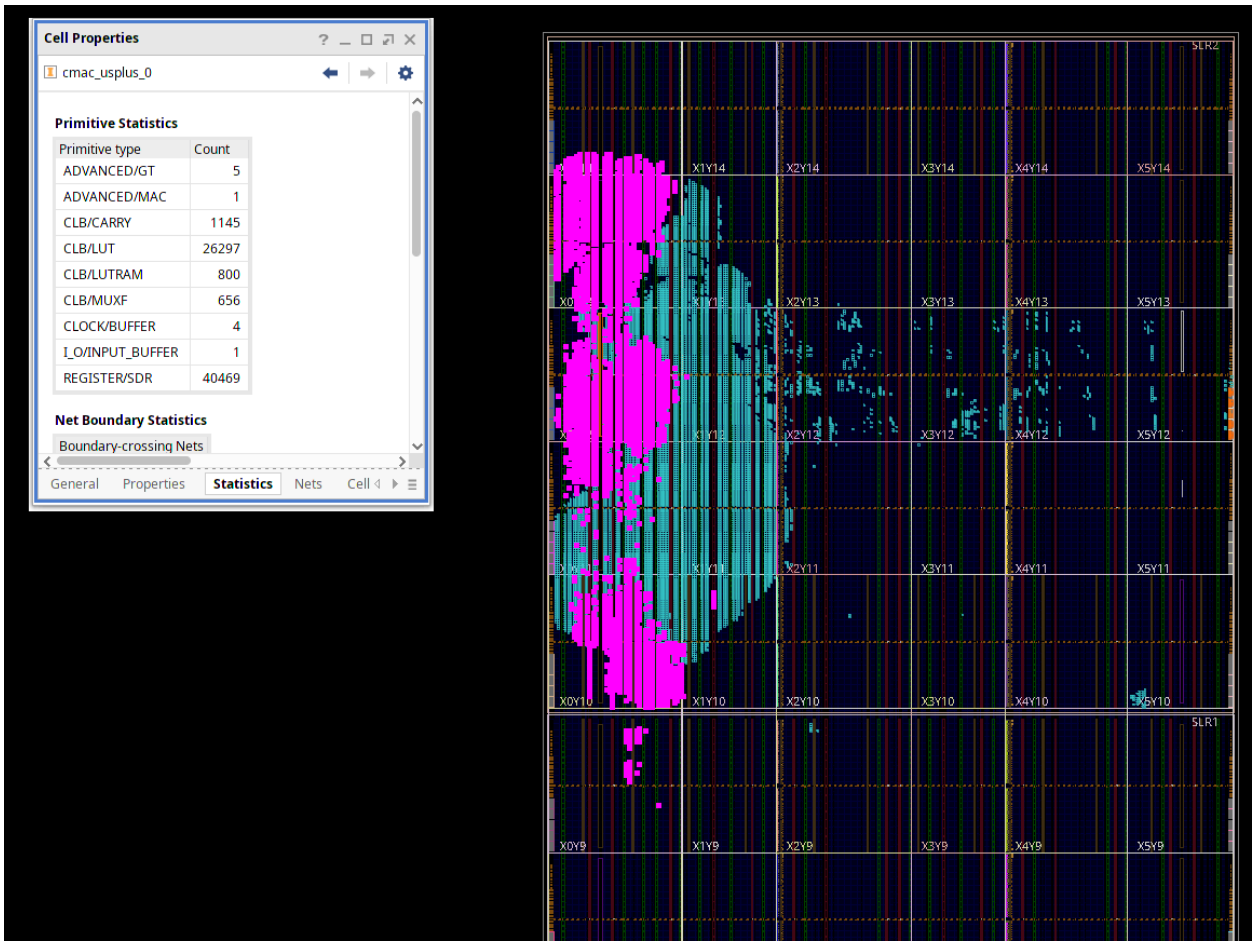
8. Test Hairpin



Atomic Rules TimeSlave
Test Setup 2019-12-27

TimeSlave highlighted in Green and CMAC highlighted in Blue in the U200 example design:

Note about 13K LUT Utilization on VU9P U200 substrate.



Hierarchy view showing breakdown of area utilization inside TimeSlave: