ICSS PTP 1588 Developer Guide

Contents

About this Guide

This wiki is meant to aid developers in understanding and using the PTP-1588 implementation on PRU-ICSS that comes as a part of Industrial SDK (http://www.ti.com/tool/sysbiossdk-ind-sitara) and Processor SDK (http://www.ti.com/tool/PROCESSOR-SDK-AM335X).

PRU-ICSS (http://processors.wiki.ti.com/index.php/PRU-ICSS) is an IP that is available on many TI SoC's, this wiki is about PRU-ICSS on Sitara devices, particularly AM335x, AM437x and AM57x. Developers reading this wiki are expected to familiarize themselves with the Industrial SDK and EMAC LLD architecture.

Note : PTP-1588 is also available on all existing Sitara devices on CPSW (The ethernet MAC on a Beaglebone) through a module known as CPTS (Common Platform Time switch) which is a hardware IP. CPSW is a gigabit switch which supports PTP 1588 v2. It's features are documented in all the respective Sitara TRM's. This wiki does not cover that aspect. Please do a comparison of both the IP's based on your requirements.

Versions

The newer version of PTP-1588 on ICSS is called TimeSync while the older version was called PTP, the basic implementation is PTP-1588 only but the name has been changed to reflect the various annexes and derivatives of PTP that will be supported like 802.1 AS. Going forward when referring to the driver/stack as a component PTP will refer to the older design while TimeSync will refer to the new design.

The change was done to accommodate the growing number of

For developers who are using the older PTP library there is a migration guide provided here

The older version is archived here (http://processors.wiki.ti.com/index.php?title=ICSS_PTP_1588_Developer_Guide&oldid=223489)

Changes in New Release

Comparison between PTP and TimeSync modules

Changes	TimeSync	PTP
TC	Transparent Clock	
OC	Ordinary clock	
Master	Master clock	
Slave	Ordinary clock as slave	
EMAC	Two port EMAC	
Switch	Either EtherNet/IP or HSR/PRP	

Glossary of terms

Glossary of PTP/1588 terms

Terms	Abbreviation
TC	Transparent Clock
OC	Ordinary clock
Master	Master clock
Slave	Ordinary clock as slave
EMAC	Two port EMAC
Switch	Either EtherNet/IP or HSR/PRP

About PTP on PRU-ICSS

PTP comes in a variety of standards and flavors. Every industry has it's own set of sub rules and configurations which are defined by Annexes and Profiles. For example the telecom industry uses the telecom profile while power and substation automation requires the power profile. CIP sync requires the Drive profile. HSR + PRP has it's own set of rules for interoperating with PTP/1588 and then there is the plain vanilla 1588 implementation for EMAC's and switches. All of this however is based on the basic PTP-1588 standard published by IEEE so there are a lot of similarities to the implementation.

Keeping these in mind PTP/1588 implementation on PRU-ICSS serves the following protocols/domains/markets

- EMAC - Not supported right now.
- EtherNet/IP CIP Sync - Supported
- Smart Grid and Home Automation (HSR/PRP networks) - Supported

For EMAC PTP will add generic time sync capabilities. CIP Sync is geared towards the drives markets while Smart Grid and Home Automation rely on PTP along with other protocols like IEC 61850 and IEC 62439 for reliability and precision.

PTP on PRU-ICSS comes in three flavors

1. Delay Request/Response based TC and OC – Drives Profile for CIP Sync. Annex D
2. Peer Delay Request/Response based TC and OC for HSR/PRP – Power profile with HSR/PRP. Annex F with modifications for IEC 62439-3

The implementation is firmware specific. Any two implementations will not be supported at the same time. For example EtherNet/IP only has End-to-End (E2E) version of PTP, it will not support Peer-to-Peer (P2P). HSR/PRP implementation supports only P2P and the implementation is unique to HSR/PRP. Standard Peer to Peer networks will not interoperate with this implementation. There are some additional differences imposed by the standards which are applicable.

In spite of the differences there are many similarities to the PTP design across all three implementations.

The requirements for Delay Request/Response based TC and OC is coming from CIP Sync which requires support for Annex D of PTP/1588 (PTP over UDP) while Peer Delay Request/Response based TC and OC is coming from IEC 61588 and IEC 62439-3 (HSR/PRP) which requires support for Annex F of PTP/1588 (PTP over 802.3). So the support is provided as such i.e. E2E TC is implemented to be over UDP (Annex D) while P2P TC is over 802.3 (Annex F). Any profile which requires E2E in Annex F and P2P over Annex D is not supported at this point of time. Besides these there is an additional section that discusses PTP requirements related to HSR/PRP.

IEC 62439-3

PTP for HSR/PRP requires that HSR header and PRP tag be added to Sync, Follow Up and Announce packets. Peer Delay Request and Response frames are considered link local and need not have the headers. In addition to this a 1588 implementation need not add or honor redundancy tags in PRP so tags are only applicable with respect to HSR. The standard is still evolving on this one so the design is subject to change. This design doc follows Ed.2 of the IEC 62439-3 standard.

Features & Capabilities

As mentioned before PTP/1588 on PRU-ICSS supports two flavors (right now)

1. Annex D based solution on EtherNet/IP
2. Annex F (with adaptation for HSR/PRP) on IEC 62439-3 Ed.2

Single/Dual Step The slave implementation is capable of handling both single and two step PTP messages, PRU-ICSS is also capable of generating two step messages, although currently it only generates single step frames. This feature will be enabled in future.

The basic features supported are

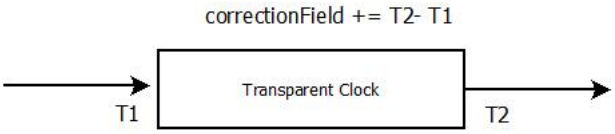
1. Single step Transparent clock
2. Single step Ordinary clock
3. BMCA (Best Master clock algorithm)
4. Syntonization factor calculation
5. Simple design for time base (More explanations in design section)
6. 1PPS output for Sync. Configurable time period for sync generation
7. Statistics for debugging and monitoring

Mechanism

End to End Transparent Clock Operation

The End to End TC works by adding the resident bridge delay correction to certain time critical packets, these are.

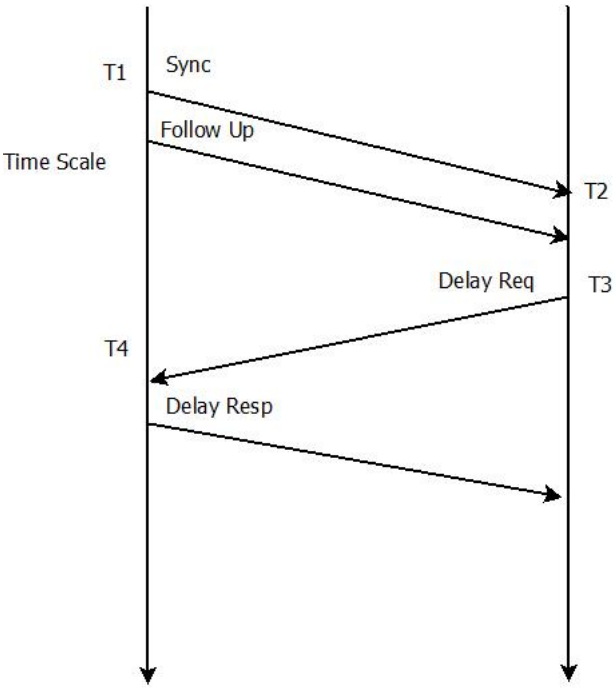
1. Sync
2. Delay Request



Line Delay Calculation (Delay Request/Response)

The calculation of line delay (time distance between two nodes) although not required for doing basic transparent clocking operation is required for finding out other items like clock drift and frequency syntonization factor (rcf). The mechanism for Line Delay is roughly the same for both Delay Request and Path Delay packets, hence this section will be referenced again for dealing with Path Delay Frames. The basic concept to understand here is that we want to find out the time it takes for a packet to traverse between the nodes. This is done by sending a packet from one node to another (usually from master to slave); this is called a sync packet. The sync packet contains a timestamp that indicates when it left the master. This is present either in the correction field or the timestamp. If the device is incapable of providing an accurate timestamp in the sync packet, a follow up packet is sent which contains this information. Upon receipt of the follow up information the slave sends a Delay Request packet which again is timestamped like the sync packet with the exit timestamp. (Delay Request is actually sent periodically and independent of follow up or Sync). When a delay request is received at the Master, a Delay Response is sent meant only for that slave which issued the Delay Request (Delay Response is tagged with Slave information). The Delay Response contains the time at which Delay Request was received at the Master.

The entire process is depicted in the diagram below.



```
The timestamps are respectively indicated by t1, t2, t3 and t4.
Line delay is calculated as
Mean Path Delay = (Forward Delay + Reverse Delay) / 2 = ((t2 – t1) + (t4 – t3))/ 2
Since on both slave and master the counters are free counters, it makes more sense to re-arrange the computation like this.
Mean Path Delay = ((t4 - t1) – (t3 – t2))/2
```

Clock Syntonization

Two clocks need not run at same frequency and minor as well as major changes in PLL should be accounted for, this is done using the syntonization factor (rcf). For example if master clock is running at half the frequency of slave then a 1s delay measured on slave is actually only 0.5s on the master. The RCF in this case would be 0.5.

```
RCF is calculated as:
Here T3 is the sync transmit timestamp from master and T4 is the sync receive timestamp on slave. T3’ is T3 measured again in future (same for T4).
Rcf = (T3’ – T3) / (T4’ – T4)
Mean Path Delay (compensated) = Mean Path Delay * rcf;
```

Clock Synchronization

Clock Synchronization is the task of adjusting the timers to correct for large changes in time base or changing the value of the timer.

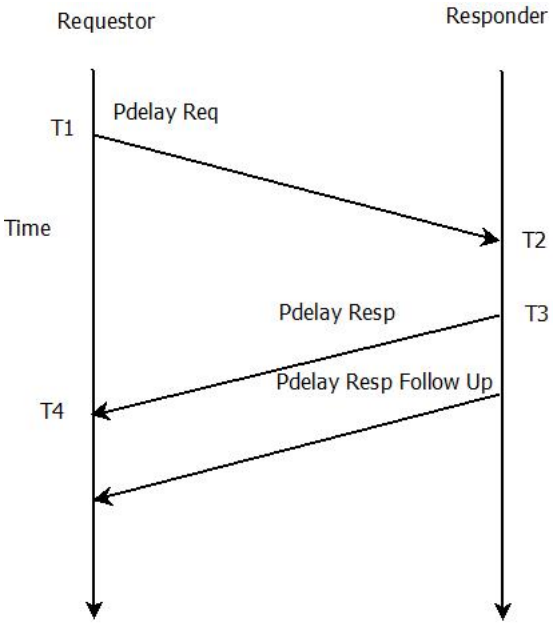
1. When the first sync packet arrives the seconds and nanoseconds field are copied directly to the corresponding fields in the PTP implementation. This is done by firmware when state machine is in BMCA complete state.
2. Once the local clock is synchronized IEP counter value and nanoseconds timestamp from master is compared to find the drift. Value calculated in nanoseconds is used to divide the sync interval to find the ECAP Period. An adjustment value is chosen and programmed into the EDMA

Peer Delay Calculation (P2P Networks)

Peer delay is calculated in a manner somewhat similar to Line delay but the frame types used are different. In this case the packets are

1. Peer Delay Request
2. Peer Delay Response
3. Peer Delay Response Follow Up

```
The nominal value of the <meanPathDelay> is computed as <meanPathDelay> = [(t2 – t1) + (t4 – t3)]/2 = [(t2 –t3) + (t4 – t1)]/2
For a one step clock like ours we return the resident bridge delay (t3-t2) in the correction field of the Pdelay response frame sent out.
For a 2-step clock the Tx timestamp t3 is sent out in a follow up frame.
```

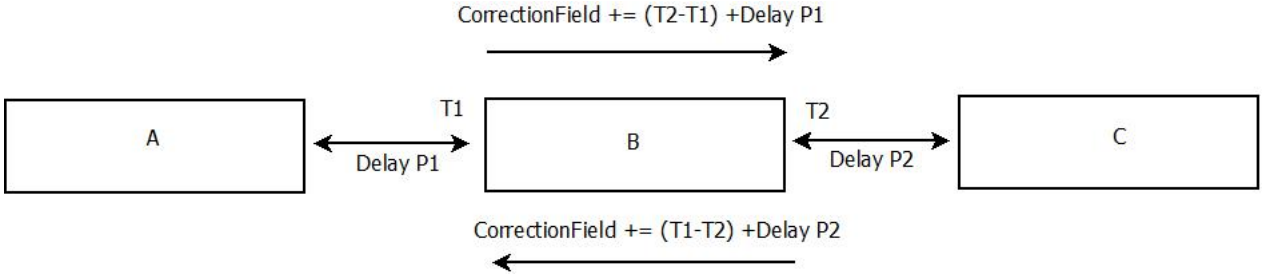


Peer to Peer Transparent Clock Operation

In Peer to Peer Transparent clock operation the four big differences compared to End to End TC are:

1. Path Delay Request/Response messages are used
2. Sync packet is not required for delay calculation
3. The clock adds resident bridge delay + mean path delay to the correction field (instead of just the resident bridge delay)
4. Delay isn’t necessarily measured between master and slave, it can be between any two devices hence the name peer to peer.

Its operation is shown in the diagram below. The correction field is updated with the value of CorrectionField where Correction Field = Mean Path Delay (lambda in the diagram) + Bridge Delay (rho in the diagram) + Correction Field of the Packet (to account for earlier delays).



Since Peer to Peer Requests are meant for adjacent nodes, it's important that these packets be dropped and not be forwarded.

Design

The processing load is shared between firmware (PRU) and Host (ARM) with the firmware doing most of the time critical activities. The generic division is as follows

- firmware (PRU)
  - Timestamping
  - Packet parsing and storing fragments to memory
  - Running checks and generating fast interrupts
- Host (ARM)
  - BMCA algorithm
  - Synchronization (clock adjustment using timestamps provided by firmware)
  - Checks for Sync timeout
  - Syntonization factor calculation
  - Line/peer delay calculation (using firmware timestamps)
  - Sending out delay request frames
  - Responding to peer delay requests
  - Management messages

The task on Host is further subdivided into Driver and Stack

- Driver - Performs all time critical tasks like packet transmit, delay calculations and synchronization
- Stack - Performs all non critical tasks like BMCA and responding to management messages. Additionally the stack checks for Announce timeout.

Clock

Every PTP implementation requires the following

1. An underlying hardware clock powering the ordinary clock which can be thought of as a wall clock
2. Ability to time stamp packets on entry and exit

In case of PRU-ICSS this clock is provided by the IEP. IEP stands for Industrial Ethernet Peripheral, it has a 32 bit timer powered by a 200 Mhz derived from the main SoC Crystal-PLL combination. The timer can be programmed to automatically wraparound on a preset value. It also has different compare and capture registers to capture different kinds of events like receive and transmit events.

More details on the HW module can be found in the ICSS section of respective TRM's. For further discussion this guide will assume that the user is familiar with IEP.

A standard PTP implementation requires two clock values.

1. The seconds value : Corresponding to number of seconds since epoch in Unix, this is a 64 bit (48 bits are used) timer in software and corresponds to the second's field of PTP origin timestamp. Firmware checks for the CMP0 event and increments this counter on every reset event. The value resides in shared memory. (See Memory Map)
2. The nanoseconds value : This is the 32 bit IEP hardware timer. All timestamps are with reference to this. CMP0 event is programmed to 1 second in PTP driver and reset on wraparound event is enabled in CMP_CFG register so this counter wraps around when it reaches 1s value. It can be read directly from the hardware, no adjustment is required.

PTP does not maintain any other timer base other than these. When a query is made to find out the local time then these two values are returned.

Note : Since one ICSS has only one IEP, one ICSS can only do one ordinary clock, because of this the two EMAC ports cannot have two independent ordinary clocks.

1 PPS Output

IEP has an additional hardware to generate a programmable sync output which is tied to the IEP counter. This is called the Sync0 unit. For this signal generation CMP1 is programmed to a value ranging from 1ms to 1 second. PRU0 checks this event in firmware and re-programs it after every hit to ensure that accurate sync pulses are generated. This sync is equivalent to the 1PPS output and should not be confused with PTP Sync frame.

Timestamping

Timestamping policy for different packets is given below

- Sync – Both nanosecond and second Rx timestamps are stored and reported to driver for the purpose of RCF calculations. Tx timestamp is taken and sent out in the Sync frame.
- Delay Response – Rx timestamp is stored and used for Delay calculation.
- Follow Up – No timestamping record is kept

- Announce – No timestamping record is kept
- Delay Req – Entry and exit timestamping is done in nanoseconds. Exit timestamp is used in delay calculations while Entry timestamp is used for bridge delay calculation.
- Pdelay Req – Rx and Tx timestamping is done in nanoseconds. Tx timestamp is used for delay calculations while Rx timestamp is used to update the correction field.
- Pdelay Res – Rx and Tx timestamps are taken in nanoseconds. Rx timestamp is used for delay calculations while Tx timestamp is used to update the correction field.
- Pdelay Res Follow Up – No timestamp record is kept.

Timestamps are stored in shared memory and not provided with the packets since other than Announce and Follow Up most packets are not even sent to the Host. The timestamps are consumed by the PTP driver as and when the interrupts are triggered or PTP Task computes peer/line delay.

Timestamping in Hardware: IEP module in PRU-ICSS timestamps packets on Receive and Transmit.

- For receive both Start of Frame (SOF) and Start of Frame Delimiter (SFD) timestamps are available.
- For transmit only SOF timestamp is available.

Since PTP-1588 specifies that SFD time stamps are to be used we store the RX SFD time stamp for all frames and add 640ns to the Tx SOF time stamp, the underlying assumption being that the preamble length is 8 bytes. This is ok since PTP-1588 frames are delayed cut-through (i.e. PRU-ICSS inserts it's own preamble and CRC instead of copying the received frames)

PHY delay correction of timestamps It's important to compensate of PHY delay and other known delay parameters in timestamps for accurate timing. This is done in the firmware by reading pre-programmed PHY delays written into the shared RAM. The memory offsets in shared RAM for this are.

- **MII_RX_CORRECTION** - Rx delay in nanoseconds
- **MII_TX_CORRECTION** - Tx delay in nanoseconds

The Tx correction value should be the PHY delay + 640ns to accommodate SOF-SFD timestamp conversion. The driver writes the latency into these locations in the function *ptpDramInit()* using the pre-defined macros RX_PHY_LATENCY and TX_PHY_LATENCY.

For TLK110 the respective values for Rx and Tx PHY delay are 186ns and 86ns. Since for Tx 640ns is added to delay the actual value comes to 726 ns.

State Machine

The PTP has the following states

1. Initialized.
2. BMCA has run and master has been determined
3. Line delay computed
4. Sync interval computed
5. Ready for adjustment. This happens once 2 and 3 occur.
6. Synchronized. Average clock drift has gone below a specified threshold.
7. Error (Announce timeout, very large adjustment, sync interval too large, missed sync frame) leads to 1.

When the Firmware initializes it goes into Initialized state and it puts all Announce and Management frames in highest priority queue and no sync interrupt is generated, TC still runs. RCF value used is the default 1.0. Once BMCA has determined the master, it's MAC ID is written into memory; firmware checks against this MAC ID to generate Fast ISR for Sync and other frames. Clock Synchronization starts and RCF is computed along with other values.

Interrupts

The EMAC-LLD has an Rx interrupt (as discussed in EMAC LLD Design Guide) but it's too slow to handle critical PTP tasks so we use a mixed approach where we dump PTP information in shared memory and issue a specific interrupt to ARM to process the data. We will refer to this as the fast ISR henceforth.

The following fast ISR's will be configured for PTP

1. Single interrupt for Sync where time synchronization will be done. Sync ISR
2. Single interrupt for Delay Response where line delay is be calculated. Delay Response ISR
3. Two interrupts – One each for Pdelay Request. Pdelay Req ISR

Announce frames and PTP management messages are handled through the regular Rx interrupt on high priority queue from where they are forwarded to PTP stack for further processing.

Synchronization

Synchronization is about making sure that the local clock on both master and slave reflect the same value (after adjustment for path delay). This is done by

1. Copying the value of clock directly (seconds and nanoseconds) at first also called the Initial adjustment.
2. Calculating the drift every time a sync frame comes and then adjusting the local clock to take care of the drift.

Synchronization is done in the Sync ISR context and adjustment must be complete by the time next Sync packet arrives. To calculate the drift the assumption is that second's field is synchronized and hence only nanoseconds field is compared to find the drift irrespective of the sync interval. Exception is when there is a wraparound and Seconds field is not the same, this can happen when the slave is so far from the master that seconds field increments by the time Sync frame reaches it.

Syntonization

Syntonization is accounting for the frequency difference between Master and Slave. This is done by

1. Keeping track of timestamps of two alternate Sync frames (not consecutive)...1st & 3rd, 2nd and 4th.
2. Taking the difference in arrival timestamp (as recorded on slave), let's call this "slave time"
3. Taking the difference in origin timestamp (as recorded on master), let's call this "master time"

The RCF or syntonization factor is computed as RCF = master time/slave time. Any delay computed on slave is then multiplied by this factor. For example if master is running twice as fast as slave then RCF will be 2 and any delay computed on slave will get multiplied by this value to reflect time in terms of master. In reality the RCF rarely goes out of the range 0.99-1.01 and any value outside this should be interpreted as an error. Timestamps are recorded and Syntonization is done in a PTP task and not inside Sync ISR. The function call for calculating RCF is calcRcfAndSyncInterval()

Determination of Sync Period

- Sync frame interval is determined in the driver from Sync frames. The driver waits for two sync interrupts to do this.
- Driver also checks if there is a large difference in the known Sync Period and the observed value, if it is determined that Sync interval has changed then state machine is initialized to Initializing and PTP syncs all over again.
- Driver monitors if the time base has changed significantly (by 1 s or more). This excludes special cases such as leap59 and leap60. Such a condition triggers state machine to go to initializing and PTP synchronization happens all over again.
- The sync frame interval is determined in the same function which calculates RCF. calcRcfAndSyncInterval()

Resetting the Driver

A driver reset might be required in the event of link loss or network reconfiguration. The function *PTP_reset()* performs this but for porting requirements user might need to implement their own version of the reset.

The following steps are needed to perform a full reset.

- Resetting the driver requires resetting certain memory locations in the shared RAM like

  - Seconds counter offset - The current UTC offset
  - *PTP_INITIAL_ADJUSTMENT_DONE* - Resetting this causes firmware to do first adjustment once again.
  - *PTP_MASTER_SRC_MAC_ID* - Sync interrupts cease to arrive once this is reset.
  - *PTP_CONTROL_FLAGS_P1* and *PTP_CONTROL_FLAGS_P1* - These are internal flags used by PTP
  - Value of 1024 should be written to *PTP_TC_RCF*

- The following structures must be reset

  - *lastSyncParam* & *syncParam* - Contains sync parameters
  - *delayParams* - Delay request and response TS parameters
  - *pDelayParams* - Pdelay request and response TS and parameters
  - *ptpSyntInfo* - Syntonization Info
  - *ptpVar* - PTP runtime variables
  - member *clockDrift* of structure *ptpVar* should be set to a non-zero value, preferably the default PPM of the crystal.
  - member *rcf* of structure *ptpSyntInfo* should be set to 1.0
  - members *firstOffset* and *initialOffset* of structure *ptpSyntInfo* should be set to 0.

- The sync timeout clock needs to be stopped.
- and finally the stack must be reset so that BMCA runs once more

Memory Map

The offset location is relative to PTP Base in Shared RAM. Currently the base is set to 0x8

Common Memory Map

Name of Offs	Description Refer to *icss_switch_ptp.h*	Offset in Shared RAM
PTP_CTRL_VAR	PTP Start/Stop (1/0) info.	*0x0*
PTP_CONTROL_FLAGS_P1	PTP flags (internal to firmware) for Port 1	*0x1*
PTP_CONTROL_FLAGS_P2	PTP flags (internal to firmware) for Port 2	*0x3*
PTP_MASTER_PORT_NUM	If Master is on Port 1, write 0 else write 1 here Only used by HSR protocol	*0x5*
PTP_RX_TS_P1	Timestamp in nanoseconds for Port 1. This is a temporary place holder before this value is copied elsewhere	*0x6*
PTP_RX_TS_P2	Same as above but for Port 2	*0xA*
PTP_FOLLOW_UP_CORRECTION_FIELD_P1	Correction field value for Follow Up Frame for Port 1	*0xE*
PTP_RX_TS_SEC_P1	Timestamp in seconds for Port 1. This is a temporary place holder before this value is copied elsewhere	*0x16*
PTP_RX_TS_SEC_P2	Same as above but for Port 2	*0x1A*
PTP_INITIAL_ADJUSTMENT_DONE	Write 1 here once Initial adjustment has been done by firmware	*0x22*
MII_TX_CORRECTION	PHY Tx correction value in nanoseconds	*0x24*
MII_RX_CORRECTION	PHY Rx correction value in nanoseconds	*0x26*

PTP_SYNC_CORRECTION_FIELD_P1	Correction field value for Sync Frame for Port 1	*0x2A*
PTP_SYNC_CORRECTION_FIELD_P2	Same as above but for Port 2	*0x30*
PTP_MASTER_SRC_MAC_ID	BMCA algorithm writes Master MAC ID here	*0x36*
PTP_SYNC_RX_TS_P1	Nanoseconds TS for Sync frame ingress on Port1	*0x3C*
PTP_SYNC_RX_TS_P2	Same as above but for Port 2	*0x40*
PTP_FOLLOW_UP_CORRECTION_FIELD_P2	Correction field value for Follow Up Frame for Port 1	*0x44*
PTP_SYNC_ORIGIN_TIMESTAMP	TS from PTP Master present on the Sync frame	*0x4A*
PTP_SYNC_SEQ_ID	Sequence ID for Sync frame	*0x54*
PTP_FOLLOW_UP_SEQ_ID	Sequence ID for Sync frame	*0x56*
PTP_SYNC_W_FUP	If it's a two step Sync, write 1 here	*0x58*
PTP_SYNC_PORT_NUM	If PTP master is on port 1, firmware writes 1 here else writes 2. Driver reads this value	*0x59*
PTP_CLK_IDENTITY_OFFSET	Clock identity of the device. Driver initializes this value	*0x5A*
PTP_SYNC0_WIDTH	Pulse width of 1PPS signal. Driver initializes this value	*0x64*
PTP_IEP_VAL_CYCLE_COUNTER	For Firmware internal usage	*0x68*
PTP_SYNC0_PERIOD	For Firmware internal usage	*0x6C*
PTP_SECONDS_COUNT_OFFSET	Current UTC offset in seconds.	*0x70*
PTP_SYNC0_CMP_VALUE	Time period of Sync pulse is written here. Driver initializes.	*0x112*
PTP_SYNC_RX_TS_SEC_P1	Sync ingress TS for Port 1 in seconds	*0x116*
PTP_SYNC_RX_TS_SEC_P2	Sync ingress TS for Port 2 in seconds	*0x11C*

E2E specific Memory Map

Name of Offs	Description **Refer to *icss_switch_ptp.h***	Offset in Shared RAM
PTP_DLY_REQ_RX_TS_P1	Delay request Rx timestamp for Port 1	*0x78*
PTP_DLY_REQ_RX_TS_P2	Same as above for Port 2	*0x7C*
PTP_DELAY_RES_RCVD_TIMESTAMP_P1	Timestamp embedded in delay response rcvd from PTP Master	*0x80*
PTP_DELAY_RES_RCVD_TIMESTAMP_P2	Same as above but for Port 2	*0x8A*
PTP_DEL_RESP_CORRECTION_FIELD_P2	Correction field embedded in Delay response frame rcvd on Port 2	*0x94*
PTP_DEL_RESP_CORRECTION_FIELD_P1	Same as above but for Port 1	*0x9A*
PTP_DEL_REQ_SEQ_ID	Delay Request Sequence ID	*0xA0*
PTP_DELAY_RES_SEQ_ID	Delay Response Sequence ID	*0xA2*
PTP_DEL_REQ_TIMESTAMP	Rx timestamp for delay request frame	*0xA4*
PTP_PDLY_REQ_RX_TS_P1	Pdelay request Rx timestamp for Port 1	*0xA8*
PTP_PDLY_REQ_RX_TS_P2	Same as above but for Port 2	*0xAC*
PTP_PDLY_RSP_RX_TS_P1	Pdelay Response Rx timestamp for Port 1	*0xB0*
PTP_PDLY_RSP_RX_TS_P2	Same as above but for Port 2	*0xB4*
PTP_DELAY_RES_PORT	Port on which Delay response was received	*0xB8*

P2P specific Memory Map

Name of Offs	Description **Refer to *icss_switch_ptp.h***	Offset in Shared RAM
PTP_CALC_LINE_DELAY_OK_P1	Signal for driver that Pdelay response has been received. Proceed with delay calculation on Port 1	*0x78*
PTP_RESP_WITH_FLWUP_P1	Write 1 if it's a 2-step Delay Response on Port 1	*0x79*
PTP_P2P_DELAY_P1	Peer delay on Port 1 in nanoseconds	*0x7A*
PTP_PDEL_REQ_RCPT_TS_P1	Pdelay Req Rx TS on Port 1	*0x7E*
PTP_PDELAY_RES_FLW_UP_CORRECTION_FIELD_P1	Correction field in Pdelay Response Follow Up frame for Port 1	*0x88*
PTP_PDELAY_RES_CORRECTION_FIELD_P1	Correction field in Pdelay Response frame for Port 1	*0x8E*

PTP_PDEL_REQ_TX_TS_P1	Pdelay Request egress TS for Port 1	*0x94*
PTP_PDEL_REQ_RX_TS_P1	Delay Response Sequence ID	*0x98*
PTP_PDEL_RES_RX_TS_P1	Pdelay response ingress TS for Port 1	*0x9C*
PTP_PDEL_RES_ORG_TS_P1	TS from peer embedded in Pdelay Response for Port 1	*0xA0*
PTP_CALC_LINE_DELAY_OK_P2	Signal for driver that Pdelay response has been received. Proceed with delay calculation on Port 2	*0xAA*
PTP_RESP_WITH_FLWUP_P2	Write 1 if it's a 2-step Delay Response on Port 2	*0xAB*
PTP_P2P_DELAY_P2	Peer delay on Port 2 in nanoseconds	*0xAC*
PTP_PDEL_REQ_RCPT_TS_P2	Pdelay Req Rx TS on Port 2	*0xB0*
PTP_PDELAY_RES_FLW_UP_CORRECTION_FIELD_P2	Correction field in Pdelay Response Follow Up frame for Port 2	*0xBA*
PTP_PDELAY_RES_CORRECTION_FIELD_P2	Correction field in Pdelay Response frame for Port 2	*0xC0*
PTP_PDEL_REQ_TX_TS_P2	Pdelay Request egress TS for Port 2	*0xC6*
PTP_PDEL_REQ_RX_TS_P2	Delay Response Sequence ID	*0xCA*
PTP_PDEL_RES_RX_TS_P2	Pdelay response ingress TS for Port 2	*0xCE*
PTP_PDEL_RES_ORG_TS_P2	TS from peer embedded in Pdelay Response for Port 2	*0xD2*
PTP_TC_RCF	Syntonization factor calculated by driver.	*0xDC*
PTP_PDEL_REQ_DOMAIN_NUM_P1	Domain Number as present in Pdelay Req frame for Port 1.	*0xE0*
PTP_PDEL_REQ_DOMAIN_NUM_P2	Same as above but for Port 2	*0xE1*
PTP_PDEL_RES_SEQ_ID_P1	Sequence ID in Pdelay Response for Port 1	*0xE2*
PTP_PDEL_RES_SEQ_ID_P2	Same as above but for Port 2	*0xE4*
PTP_PDELAY_REQ_CORRECTION_FIELD_P1	Correction field in Pdelay Req frame received on Port 1	*0xE6*
PTP_PDELAY_REQ_CORRECTION_FIELD_P2	Same as above but for Port 2	*0xEC*
PTP_PDEL_REQ_SEQ_ID_P1	Sequence ID of Pdelay Req frame rcvd on P1	*0xF2*
PTP_PDEL_REQ_SEQ_ID_P2	Same as above but for Port 2	*0xF4*
PTP_PDEL_REQ_RX_CLK_IDENTITY_OFFSET_P1	Clock identity of neighboring node on Port 1	*0xFA*
PTP_PDEL_REQ_RX_PORT_ID_P1	Port ID of neighboring node on Port 1	*0x102*
PTP_PDEL_REQ_RX_SEQ_ID_P1	Sequence ID of Pdelay request rcvd on Port 1	*0x104*
PTP_PDEL_REQ_RX_CLK_IDENTITY_OFFSET_P2	Clock identity of neighboring node on Port 2	*0x106*
PTP_PDEL_REQ_RX_PORT_ID_P2	Port ID of neighboring node on Port 2	*0x10E*
PTP_PDEL_REQ_RX_SEQ_ID_P2	Sequence ID of Pdelay request rcvd on Port 2	*0x110*

Resource Usage

Interrupts

PTP uses three interrupts events driven by the PRUSS INTC

- Sync interrupt (common to both E2E and P2P implementations) - It's functions are
  - Posting *syncIntPendSemHandle* semaphore upon which Delay Req frame is sent out and Syntonization factor and Sync Interval is calculated
- Delay response interrupt (E2E) - Functions are
  - Line delay is calculated in this ISR
- Pdelay request interrupt on Port 1 (P2P)
  - Copy the parameters from Delay Request on Port 1 and reply with a PDelay Response
- Pdelay request interrupt on Port 2 (P2P)
  - Identical to ISR for Port 1

Delay response and Pdelay request interrupts are mutually exclusive so Delay response interrupt and Pdelay request ISR on Port 1 share the same interrupt number and ISR.

The table below shows the mapping

PTP/1588 Interrupt Usage

Interrupt Name and ISR	PRU ICSS INTC number (AM335x)	Description
syncIntNum - ptpSyncIsr()	PRU_ICSS_EVTOUT3	Interrupt for Sync frame on any port
genericIntNum - ptpGenericIsr()	PRU_ICSS_EVTOUT4	Shared interrupt for Delay Response and Pdelay Request on Port 0
pDelayRespP1IntNum - ptpDelReqIsrP1()	PRU_ICSS_EVTOUT5	Interrupt for Pdelay Request on Port 1

Interrupt mapping is described in the EMAC dev guide here

Tasks

PTP/1588 driver uses two tasks

- *ptpGenericTask()* - It has different roles in case of E2E and P2P
  - End 2 End - Pend on semaphore posted by Sync interrupt, prepare a Delay response and send it out on the same port on which Sync was received. Also calculate syntonization factor and sync interval
  - Peer 2 Peer - Check on each port if Pdelay response has been received and calculate peer delay.
- *ptpCalcRCFSyncIntTaskP2P()* - This is exclusive to P2P and pends on the Sync semaphore to calculate syntonization factor and sync interval

PTPd stack internally uses timers to implement tasks like

- Checking for PTP frame arrival
- Checking Announce timeouts

Semaphore

PTP/1588 uses one semaphore *syncIntPendSemHandle* to indicate that Sync frame has arrived and it's ok to calculate RCF (Syntonization factor) and Sync Interval.

Timers

Two timers are used for overall implementation. These can be software timers since the resolution is not very high.

- PTP Driver uses one timer for detecting a Sync timeout and implementing redundant clock functionality. The clock resolution is the minimum sync interval for the system.
  - The name of clock handle is syncISRClockHandle
  - The name of ISR is *PTP_syncTimeoutIsr()*
  - The timer is a manual start timer and is started once sync interval is calculated.
- PTPd stack uses one timer to detect Announce timeouts.

Porting your own PTP stack

Data Flow

Porting your own PTP/1588 stack is very easy because most of the tasks. esp. time critical tasks are performed by the driver. The stack is only responsible for

1. Running BMCA and finding out the PTP Master.
2. Checking for Announce timeouts
3. Handling management messages

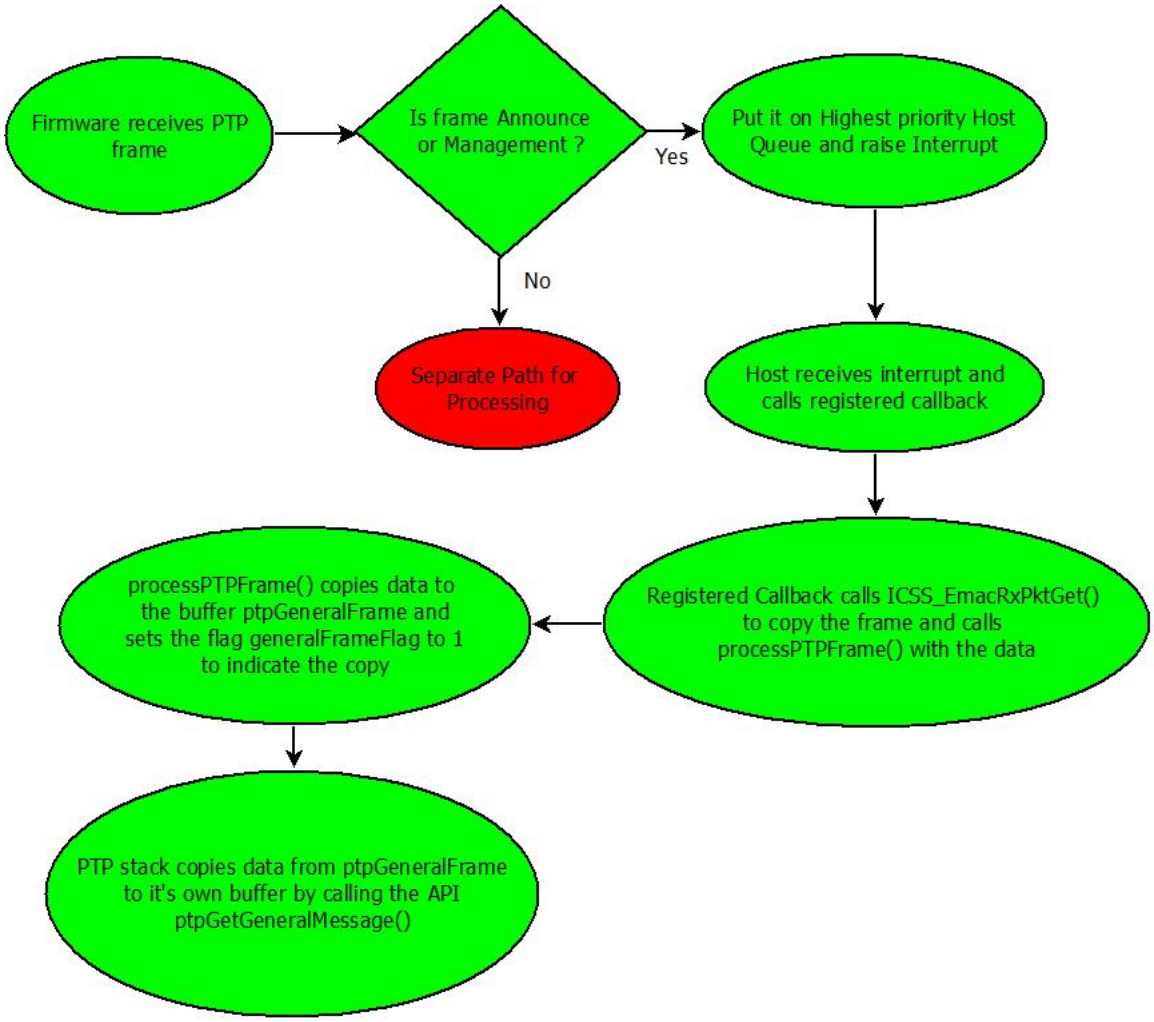
Once *ptpEnable()* API is called from the application, all PTP announce and management messages are sent on the highest priority queue to the Host.

A callback function registered to the highest priority queue then handles these frames. The callback API for PTP/1588 is called *processPTPFrame()*, it expects the entire packet in a buffer *pktBuffer*. For E2E(EtherNet/IP) the callback is called inside the EtherNet/IP callback *processProtocolFrames()* which it shares with DLR. In case of P2P, the callback can be called directly after copying the frame from the queue.

Once the frame has been received in *pktBuffer* the frame type is checked and if it's an Announce or Management message the data is copied to the buffer *ptpGeneralFrame*. Once the frame has been copied a flag *generalFrameFlag* is set. Another API called *ptpGetGeneralMessage()* checks the flag and copies the data from the buffer *ptpGeneralFrame* to another buffer *buff* which belongs to the PTP stack. The API *ptpGetGeneralMessage()* strips the header from the frame based on whether the mode is E2E or P2P and only copies the relevant data to the Stack buffer, this way only the relevant data is sent to stack and stack need not worry whether the mode is E2E or P2P.

The stack can call the API *ptpGetGeneralMessage()* as a callback to the PTP callback or in a loop in a task. In PTPd this is done in a task.

The sequence is depicted in the flowchart below



Running Best Master Clock Algorithm

PTP/1588 firmware compares the Source MAC ID of Sync and Follow Up frames with the value stored at the location **PTP_MASTER_SRC_MAC_ID** in shared RAM, and if it does not match then a Sync Interrupt is not raised so for synchronization to happen once BMCA is complete and source MAC ID of a master is available. This is done by the API *updateParentAddress()*.

Getting the current time

As explained previously in Design section, PTP has a very simple design where the IEP counter acts as the nanoseconds counter and the current UTC offset in seconds is stored in a memory location in the shared RAM. A simple read of the register and memory location gives the current time. This is done by the API *ptpGetCurrentTime()*. Refer to the API guide for more information.

Resetting the driver

In case of a timeout, link break etc the driver requires a reset. This is done using the API *resetPTP()*. The API also calls the PTPd stack de-init and init API calls *ptpDrvStackDeInit()* and *ptpDrvStackInit()*

Getting other parameters

Most of the information related to PTP is encapsulated in Announce and Management messages however certain information is part of the PTP/1588 driver, the stack may need access to these for it's operation. Some of these values are part of the PTP runtime structure *ptpRuntimeVar_t*.

- Sync Interval - *currSyncInterval* contains the running average of interval between two sync frames in nanoseconds
- Port number on which Master is connected - *syncPortNum* contains this value. Value of 1 represents PORT 1 and value of 2 represents PORT 2
- Peer/Line delay - member array *pathDelay* contains the calculated delay. First member of the array is the delay for PORT 1 and second member contains delay for PORT 2. For E2E only one delay is calculated and it's available in the first member
- Offset from master - *avgRunningOffset* contains the running average of this value.
- Sequence ID - *sequenceID* contains the sequence ID value populated in the delay request frames sent out
- Sequence ID of current sync frame - *curSyncSeqId* contains this value.

PTPd Stack integration in the example

Initialization

PTP Initialization requires in the following order

- Memory Allocation
- Configuration and Interrupt assignment
- Callback registration
- PTP initialization
- Stack initialization - Once IP has been initialized
- Enabling PTP
- IP assignment

Memory Allocation

ptpObj_t and *ptpUserConfig_t* structures must be allocated in *main.c*. Rest of the structures are allocated within PTP driver with the PTP initialization API call *ptpAllocMem()*.

Configuration and Interrupt assignment

PTP must be configured as either Ordinary Clock or Transparent Clock and as either End 2 End or Peer 2 Peer. Below is an example from EtherNet/IP adapter example

```
/*Configure PTP*/
ptpConfig.config = BOTH;
ptpConfig.type = E2E;
ptpConfig.protocol = UDP_IPV4;
```

Here PTP/1588 is configured as both OC and TC and in E2E mode.

Limitation : Currently there is no way to turn off OC and TC features separately. All PTP implementations are configured as both OC and TC by default.

Interrupt assignment depends on how the mapping has been done. Following is the excerpt from EtherNet/IP adapter

```
ptpConfig.syncIntNum = 23;
ptpConfig.genericIntNum = 24;
```

```
/*This is unused if E2E mode is used*/
if(P2P == ptpConfig.type)
{
    ptpConfig.pDelayRespP1IntNum = 25;
}
```

Callback registration

As explained previously a callback needs to be registered to handle PTP frames which go to the highest priority queue. The callback can be exclusively for PTP as is the case for HSR/PRP or it can be mixed with a protocol like DLR like in EtherNet/IP. Here we take the example of a callback registration in HSR

```
/*Packet processing callback*/
((((ICSSEMAC_Object *)
   emachandle->object)->callBackHandle)->rxRTCallBack)->callBack =
       (ICSS_EmacCallBack)processHighPrioPackets;
((((ICSSEMAC_Object *)
   emachandle->object)->callBackHandle)->rxRTCallBack)->userArg = emachandle;
```

PTP Initialization

PTP/1588 is initialized with the API call *ptpInit()*

Stack Initialization

This will vary depending on the stack in usage. For PTPd this is done with the call *ptpDrvStackInit()*

Enabling PTP

PTP is enabled on firmware with the call *ptpEnable()*

IP assignment

For E2E + UDP IP must be assigned so it's part of frames being sent out. *addIPtoPTP()* is called with the IP address once IP address is available.

Basic self debugging

Basic PTP/1588 self-check is built into the UART console of the application. For both HSR/PRP and EtherNet/IP the UART menu option for viewing PTP is the letter "**P** (case insensitive)". The screenshot for both applications UART menu is shown below.

```
IP Address      : 192.168.1.17
Mac Id          : 84:eb:18:ad:dd:c1
Device config : PRP

HSR/PRP Application Menu Options. Press key (Upper/Lower) and hit Enter
*****************************************
S : Show Statistics
C : Show HSR/PRP Configuration
N : Show Ring members/Node Table
I : Assign IP address
P : Show PTP/1588 status
R : Run Rx/Tx test
H : Help menu. Shows details on all the options
*****************************************
p

Peer Delay on P1 :                      0 ns
Peer Delay on P2 :                      0 ns
Not receiving any Sync frames, are you sure device is connected to PTP Master ?
```

The menu shows the current offset from master, line/peer delay (for both ports) and current time (UTC offset).

If no master is connected then UART prompts with a question **"Not receiving any Sync frames, are you sure device is connected to PTP Master ?"**.

Advanced Debug

There are two ways of debugging.

- PTP/1588 stack supports all management messages and getting and setting of standard parameters subject to certain conditions (for example PRU-ICSS PTP/1588 devices are slave only so certain parameters are not applicable). Users can query the parameters and figure out what's going on.
- Take a look at the internal variables and structures in CCS while debugging with JTAG

Relevant structures and their members are listed below. Not all members are listed as many of them have no debug value.

- **ptpRuntimeVar_t ptpVar** - PTP Runtime Variable : It contains all the run time variables required by PTP/1588 application.

PTP Runtime variables

Variable Name	Debug Value
sequenceID	Sequence ID encoded into the outgoing delay request and Pdelay request/response frames
curSyncSeqId	sequence ID of current Sync frame
prevSyncSeqId	sequence ID of previous Sync frame A difference of more than 1 between current and previous sync indicates a missed frame
currOffset	Current value of offset from Master
prevOffset	Previous value of offset from Master Difference between current and previous offset indicates jitter
adjustmentIsStable	Once jitter/drift goes below certain threshold this gets set to 1 If this isn't set even after settling time, indicates some issue with synchronization
Switch	Either EtherNet/IP or HSR/PRP
syncPortNum	Port number on which sync is being received. Values 1/2 for Port 1 and Port 2
syncTimeoutEvt	Is set to 1 if a sync timeout is detected
pathDelay	Line delay in case of E2E and Peer delay in case of P2P Array which contains delay for both ports. 0th index indicates line delay
meanPathDelay	Delay on the port connected to master In case of E2E this and pathDelay[0] are same
clockDrift	Jitter or drift of the clock
ltaOffset	Long Term Offset, is the running offset from master. Computed using an exponential averaging filter
avgCorrectionField	Average value of correction field in Sync frame when it reaches device
currSyncInterval	Running average of Sync Interval. Sync interval is computed as a running average

	This value is used for IEP adjustment.

- **ptpSyntInfo_t ptpSyntInfo** : Syntonization variables

PTP Syntonization variables

Variable Name	Debug Value
rcf	The RCF or syntonization value in decimals. Value should be within 0.1% of 1 generally.
syncIngressTs	An array of corrected Sync Rx timestamps Alternate values form a pair for purpose of rcf calculation
correctedMasterTs	An array of master origin timestamps (corrected) A difference of more than 1 between current and previous sync indicates a missed frame

- **ptpISRCount_t ptpISRCount :** Interrupt count for different PTP message types. Variable names are self explanatory so not being explained separately
- **ptpStats_t ptpStats :** PTP/1588 Statistics. Records events like reset events etc.

PTP Stats variables

Variable Name	Debug Value
numOffsetTooLarge	Number of times the offset from master is above the threshold OFFSET_THRESHOLD_FOR_RESET Persistent, does not reset with PTP Reset event.
numSyncIntTooLarge	Every time sync interval variation is above the threshold SYNC_INTERVAL_PERIOD_CHANGE_THRESHOLD this is incremented
numSyncMissed	Number of missed sync frames. Difference between current and previous sync frame sequence ID
numSyncTimeouts	Number of times sync timeout timer has been triggered.
negativeLineDelay	Certain conditions can trigger a negative line delay. This variable records number of such instances
firstOffset	Offset calculated as soon the first sync interrupt is received This happens after first adjustment so this value is equal to crystal PPM
initialOffset	It is the offset from master when current offset goes below the threshold STABLE_FILTER_THRESHOLD It also represents the crystal PPM closely

{{

1. switchcategory:MultiCore=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **ICSS PTP 1588 Developer Guide** here.

Keystone=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **ICSS PTP 1588 Developer Guide** here.

C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article **ICSS PTP 1588 Developer Guide** here.*

DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article **ICSS PTP 1588 Developer Guide** here.*

MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article **ICSS PTP 1588 Developer Guide** here.*

OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **ICSS PTP 1588 Developer Guide** here.*

OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **ICSS PTP 1588 Developer Guide** here.*

MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **ICSS PTP 1588 Developer Guide** here.*

*For technical su please post you questions at http://e2e.ti.cor Please post on comments abo article **ICSS PT 1588 Develope Guide** here.*

}}

Links

Amplifiers & Linear
Audio
Broadband RF/IF & Digital Radio
Clocks & Timers
Data Converters
DLP & MEMS
High-Reliability
Interface
Logic
Power Management
Processors

- ARM Processors
- Digital Signal Processors (DSP)
- Microcontrollers (MCU)
- OMAP Applications Processors

Switches & Multiplexers
Temperature Sensors & Control ICs
Wireless Connectivity