ENT-AN1189

Application Note

PTP Clock Implementation Using the Caracal Switch

Microsemi
Power Matters.™

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

VPPD-02910

Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 1.0

Revision 1.0 was the first publication of this document.

Contents

Figures

Tables

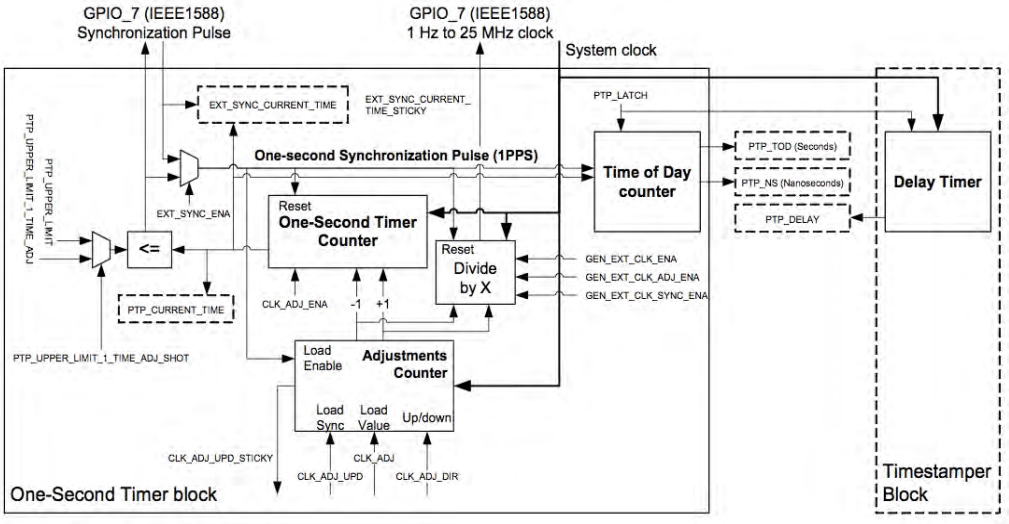
2 Caracal PTP Hardware

The Precision Time Protocol (PTP) is defined by IEEE 1588. The protocol and technology allows for the synchronization of precise time of day and network timing through packet networks. The Caracal Ethernet switch family from Microsemi Semiconductor provides the advanced PTP hardware engine. Enabling customers to implement both one-step and two-step transparent clocks (end-to-end and peer-to-peer) as well as cost-effective IEEE 1588 master and slave devices (ordinary clock).

2.1 ToD Counter and Delay Timer

The one-second timer counter as shown in the following illustration is the center of the Caracal local clock.

Figure 1 One-Second Timer Counter



The one-second timer counter is a 28-bit counter running on a 250 MHz system clock (in steps of 4 ns). The time of day (ToD) information is comprised of a 32-bit second counter and a 28-bit nanosecond counter. The nanosecond part of ToD is derived directly from the one-second timer counter, the second part of ToD is obtained from a 32-bit second counter which increments on the 1PPS synchronization pulse generated by the one-second timer counter (or external 1PPS pulse).

The goal of PTP operation is for allowing a PTP slave clock device to calculate the time offset from the master clock and correct it's local ToD counter to synchronize it with the master clock device. A one-time correction to the slave device's ToD counter can be used for that purpose, once the time offset has been calculated. If the slave clock device runs on a free running reference clock, it's ToD counter will drift away after the one-time correction. To solve this problem, the Caracal hardware provides an internal clock adjustment method by using an adjustment counter. The adjustment counter can be programmed to issue +/-1 correction to the one-second timer counter in a programmable interval. The interval is determined by the load value of the adjustment counter, which ranges from nanoseconds to one-second.

Each Caracal port module contains a hardware time stamping module in the MAC, this measures and records the arrival and departure times for any PTP event frame. The time stamping is not done using the ToD counter, it is done using a free-running delay timer. The delay timer is a 28-bit

nanosecond counter running on the 250 MHz system clock. All ports share the same delay timer. The delay timer and the one-second timer run on the same system clock, their values are not equal because the adjustment counter only corrects the one-second timer. There are two time domains in the Caracal switch, the one-second timer domain and the delay timer domain. Only the one-second timer is in sync with the master clock. The two timing domains are correlated using the PTP latch. Software can trigger a PTP latch which enables the ToD counter value and the delay timer value to be latched into specific registers at the same time. This makes it possible for software to translate from the delay timer domain to the one-second timer domain (or ToD domain).

For example, when a PTP frame arrives, it is time stamped by the hardware time stamping module in the MAC with the current value of the delay timer. The Rx_timestamp is stored in a time stamp FIFO for software to read, the PTP frame itself is redirected or copied to the CPU. When the software obtains the PTP frame and the associated Rx_timestamp from the FIFO it only knows the delay timer value when the frame was time stamped. The software can perform a PTP latch to obtain the ToD counter and the delay timer. There is a small time gap from when the frame is hardware time stamped and when the software does the PTP latch. The software can still calculate the ToD counter when the frame was hardware time stamped by applying the formula latched ToD counter – (latched delay timer – Rx_timestamp).

2.2 Time Stamp Identifier

In the example above, a PTP frame is received and forwarded to the CPU, the hardware Rx_timestamp is stored in a time stamp FIFO for the software to read. In another case, when the software sends out a PTP frame in two-step mode it also expects the hardware time stamper to store the Tx_timestamp in the FIFO. There must be a mechanism for the software to correlate a PTP frame with the associated time stamp in the FIFO. This is done through the time stamp identifier field in the internal frame header (IFH) and also in the time stamp FIFO.

An IFH is generated and inserted into the original frame by hardware for every frame received. It provides additional information about the frame (for example, source port information) for software to process the frame. When the CPU injects a frame into the switch an IFH is also required to inform the switch on how to handle the frame.

A Time stamp identifier is a 6-bit field in the IFH which overloads the Differentiated Service Code Point (DSCP) value. The time stamp identifiers can take values between 0 and 63. Values of 0 through 3 are pre-assigned to the CPU for injection of frames while values 4 through 62 are used by the hardware. A value of 63 implies that all values 0-62 are in use.

When a PTP frame arrives a free time stamp identifier is assigned and inserted into the IFH by hardware. The same time stamp identifier is also stored in the time stamp FIFO with the Rx_timestamp. If the frame is destined for the CPU, the software can read the time stamp identifier in the IFH to associate the frame with the Rx_timestamp in the FIFO. If the frame is hardware forwarded to the egress, the IFH is also followed all the way to the egress. When the frame is Tx_timestamped the time stamp identifier is extracted from the IFH and stored in the FIFO queue together with the hardware calculated residence time. When the software reads the residence time from the FIFO queue, it is able to correlate the residence time with the frame. For CPU injection of a PTP frame the time stamp identifier is assigned by the software and inserted into the IFH. The hardware Tx_timestamper obtains it from the IFH and stores the time stamp identifier together with the Tx_timestamp in the same way, the software can then correlate Tx_timestamp to the correct CPU injection frame.

2.3 VCAP-II engine and PTP actions

VCAP-II is the second generation of the Microsemi Content-Aware Packet processor engine for wire-speed packet inspection. It can be used for advanced VLAN and QoS classifications/manipulations, IP source guarding, and security features for wireline and wireless applications. There are three stages of VCAP-II matches implemented in Caracal: ingress stage 1 (IS1), ingress stage 2 (IS2), and egress stage 0 (ES0). PTP frame identification with time stamping actions can be decided at IS2. PTP actions are indicated by two bits (PTP_ENA[1:0]) in the action vectors.

Each frame will be subject to two IS2 lookups. There are numerous action fields within the action vector indicating different actions such as: CPU copy (CPU_COPY_ENA in the action vector), redirect to CPU (MASK_MODE and PORT_MASK in the action vector) and one-step/two-step time stamping (PTP_ENA[1] and PTP_ENA[0]) are of particular interest for PTP implementation.

PTP_ENA[1:0] specifies whether one-step or two-step PTP time stamping must be done at egress for a frame. The Tx time stamper generates a time stamp only if the frame has matched a VCAP-II IS2 entry with either PTP_ENA[1] or PTP_ENA[0] set.

If PTP_ENA[0] is set, the Tx time stamper will perform the following one-step PTP time stamping process:

1. Obtain the delay timer value when the fist byte of the frame begins transmission.
2. Perform the calculation: residence time = current delay timer value – Rx_timestamp.
3. Read the correction field in the PTP frame's header and add it with the residence time; then write the result back into the frame's correction field.
4. Make any necessary changes to the checksum if the PTP frame is over UDP, IPv4, or IPv6. Update the FCS (this is actually done by the rewriter).

If PTP_ENA[1] is set, the Tx time stamper will perform the following two-step PTP time stamping process:

1. Obtain the delay timer value when the first byte of the frame begins transmission.
2. Perform the calculation: residence time = current delay timer value - Rx_timestamp.
3. Store the residence time into the time stamp FIFO together with other information associated with that frame (for example, the time stamp identifier obtained from the IFH and port number frame is sent out).

Note: the term one-step or two-step PTP time stamping only specifies the behavior of the egress time stamp module. It does not imply one-step time stamping only works for one-step PTP clocks and two-step time stamping only works for two-step PTP clocks.

There are some exceptions to the PTP_ENA[1:0] actions as follows:

- When a PTP frame's destination port is the CPU port, and two-step time stamping is enabled for this PTP frame, the Rx_timestamp is stored in the FIFO queue instead of the residence time. This is useful for software to obtain the arrival time for any PTP frame.
- When a PTP frame is inserted into the switch by software, there is no Rx_timestamp associated with that PTP frame, thus the hardware calculated residence time will only be the current delay timer value (Tx_timestamp). This is useful for a two-step device to get the departure time of a PTP frame. For a one-step device, Rx_timestamp must be subtracted from the correction field prior to the frame being injected. This will allow the residence time to be calculated correctly.

  For example, when a one-step master sends out a Sync message it can do a PTP latch to obtain the ToD and delay timer. The delay timer value can be regarded as the Rx_timestamp. The software can put the ToD value into the original time stamp field of the Sync message and place

the Rx_timestamp into the correction field of that Sync message. When the Sync message arrives at the egress port, the Tx_timestamp will be added to the correction field so the correction field becomes Tx timestamp – Rx timestamp, to reflect the precise Sync message departure time.

3 Different PTP Clock Types

Caracal supports different PTP clocks with proper software—one-step/two-step ordinary clocks (master and slave) and transparent clocks (end-to-end and peer-to peer). In this section reference flowcharts for different PTP clocks are described one by one in detail. The reader should be aware of the following nuances.

- There is more than one way to implement each PTP clock type. Only one solution is presented in this section.
- Clock adjustment (both internally by programming the adjustment counter, or externally by programming an external programmable clock) and PTP frame filtering algorithms are not covered in this document.

Table 1 Delay Request-Response Mechanism with One-Step Master

Step	Caracal as an Ordinary Clock (one-step master)		Caracal as an Ordinary Clock (slave)
1	Latch ToD counter and delay timer.		

2	Prepare Sync message and send to egress.
	twoStepFlag = FALSE;
	originTime stamp = ToD counter from step1;
	Correction = - latched delay timer from step1.		

3	IS2 set to enable one step PTP time stamping for the Sync frame, IS2_ACTION.PTP_ENA[0] is set.		
4	Hardware time stamping is done on the egress when the first byte of the Sync frame begins transmission. Residence time is calculated to be the current delay time. Residence time is then added to the correction field of the Sync message. The resulting correction field = current delay timer – latched delay time from step1. Send out the sync frame with updated FCS.		
5		Sync message transmission.	
6			Hardware Rx time stamping is done upon reception of the Sync message.
7			IS2 is set to redirect the received Sync message to the CPU and two-step PTP time stamping is enabled.

Step	Caracal as an Ordinary Clock (one-step master)		Caracal as an Ordinary Clock (slave)
8			Software gets T1 from the Sync message and Rx_timestamp from the FIFO queue. Note: Rx_timestamp is stored into the FIFO instead of residence time when the frame is destined to the CPU.
9			Software latches ToD counter and delay timer. T2 is calculated by latched ToD counter – (latched delay timer – Rx_timestamp from step 8).
10			Software latches ToD counter and delay timer and prepares Delay_Req message with correctionField = 0. originTime stamp = 0 or the latched ToD counter value.
11			CPU sends out the Delay_Req message and IS2 is set to enable the Delay_Req frame for two step processing, IS2_ACTION.PTP_ENA[1] is set.
12			Hardware Tx_timestamping is done when the Delay_Req is sent out and residence time is stored in the time stamp FIFO queue. Residence time is the delay timer when Delay_Req starts to transmit. Note: Tx_timestamp is stored in the FIFO instead of the residence time when the frame is sourced from the CPU.
13			Software reads Tx_timestamp for the Delay_Req from the time stamp FIFO queue and calculates T3 = latched ToD counter (from step 10) + Tx_timestamp – latched delay timer (from step 10).
14		Delay_Req Transmission.	
15	Hardware Rx time stamping is done upon reception of the Delay_Req message.		
16	IS2 is set to redirect the Delay_Req to the CPU and two-step PTP time stamping is enabled.		

Step	Caracal as an Ordinary Clock (one-step master)		Caracal as an Ordinary Clock (slave)
17	Software gets the Rx_timestamp from the FIFO and latches ToD counter and delay_timer. Calculate T4 = latched ToD counter – (latched delay_timer – Rx_timestamp).		
18	Software prepares a Delay_resp message with recieveTime stamp = T4(from step 17) and correctField = correctionField copied from the received Delay_Req (the correction field could be added by any TC between slave and master).		
19	Send out the Delay_Resp message without IS2 action.		
20		Delay_Resp transmission.	
21			IS2 is set to redirect the Delay_Resp message to the CPU with no PTP action. Software calculates T4 = receiveTime stamp of Delay_Resp – correctionField of Delay_Resp.
22			Software calculates meanPathDelay =((T2-T3) + (T4-T1))/2 and timeOffset = ((T2-T1) - (T4-T3))/2

Table 2 Delay Request-Response Mechanism with Two-Step Master

Step	Caracal as an Ordinary Clock (two-step master)		Caracal as an Ordinary Clock (slave)
1	Latch ToD counter and delay timer		
2	Prepare Sync message and send to egress. originTime stamp = latched ToD from step1; Correction = 0; twoStepFlag = TRUE.		
3	IS2 is set to enable two step PTP time stamping for the Sync frame, IS2_ACTION.PTP_ENA[1] is set.		
4	Hardware Tx time stamping is done on the egress when the first byte of the Sync frame starts transmission. Tx_timestamp is stored in the time stamp FIFO queue while the Sync message is sent out unchanged.		

Step	Caracal as an Ordinary Clock (two-step master)		Caracal as an Ordinary Clock (slave)
5		Sync transmission.	
6			Hardware Rx time stamping is done upon reception of the Sync message.
7			IS2 is set to redirect the received Sync message to the CPU and two-step PTP time stamping is enabled. Software gets the Rx_timestamp from the FIFO queue.
8			Software latches ToD counter and delay timer. T2 is calculated by latched ToD counter – (latched delay timer – Rx_timestamp from step 7).
9	Software reads the Tx_timestamp of the Sync message from the FIFO queue and calculates T1 = ToD counter (from step1) + Tx_timestamp – latched delay timer (from step 1)		
10	Prepare and send a Follow_Up message with PreciseOriginTimestamp = T1 (from step 9); correctionField = 0.		
11		Follow_Up transmission.	
12			IS2 is setup so that the Follow_Up message is redirected to the CPU.
13			Software calculates T1 from fields in the Follow_Up message combined with the Sync message received in step 7. T1 = PreciseOriginTime stamp of Follow_Up + correctionField of Follow_Up + correctionField of Sync.
14			Software latches ToD counter and delay timer and prepares Delay_Req message with correctionField = 0, originTime stamp = 0 or the latched ToD counter value.
15			CPU sends out the Delay_Req message and IS2 is set to enable the Delay_Req frame for two step processing, IS2_ACTION.PTP_ENA[1]is set.
16			Hardware Tx_timestamping is done when the Delay_Req is sent out and Tx_timestamp is stored in the time stamp FIFO queue.
17			Software reads the Tx_timestamp for the Delay_Req from the time stamp FIFO queue and calculate T3 = latched ToD counter (from step 14) + Tx_timestamp from FIFO – latched delay timer (from step 14).

Step	Caracal as an Ordinary Clock (two-step master)		Caracal as an Ordinary Clock (slave)
18		Delay_Req transmission.	
19	Hardware Rx time stamping is done upon reception of the Delay_Req message.		
20	IS2 is set to redirect the Delay_Req to the CPU and two-step PTP time stamping enabled.		
21	Software gets the Rx_timestamp from the FIFO and latches the ToD counter and delay_timer. Calculate T4 = latched ToD counter – (latched delay_timer – Rx_timestamp).		
22	Software prepares a Delay_resp message with recieveTime stamp = T4 (from step 21); correctField = correctionField copied from the received Delay_Req.		
23	Send out the Delay_Resp message without IS2 action.		
24		Delay_Resp transmission.	
25			IS2 is set to redirect the Delay_Resp message to the CPU with no PTP action. Software calculates T4 = receiveTime stamp of Delay_Resp – correctionField of Delay_Resp.
26			Software calculates meanPathDelay = ((T2-T3) + (T4-T1))/2 and timeOffset = ((T2-T1) - (T4-T3))/2.

Table 3 One-Step End-to-End Transparent Clock with One-Step Master

Step	Ordinary Clock (one-step master)	Caracal as One-Step E2E Transparent Clock	Ordinary Clock (slave)
1	Master sends out Sync with twoStepFlag = FALSE.		
2		Hardware Rx time stamping is done upon reception of the Sync message.	

Step	Ordinary Clock (one-step master)	Caracal as One-Step E2E Transparent Clock	Ordinary Clock (slave)
3		Sync message is forwarded to the egress together with the Rx_timestamp. IS2 is set so that one step PTP time stamping will be enabled.	
4		Hardware Tx time stamping is done. Residence time is calculated by the current delay_timer – Rx_timestamp.	
5		Residence time is added to the correction field of the Sync message. Send out the sync frame with updated FCS	
6			Get T1 and T2.
7			Send out Delay_Req message with correction field cleared and store T3.
8		Hardware Rx time stamping is done upon reception of the Delay_Req message.	
9		Delay_Req message is forwarded to the egress together with the Rx_timestamp. IS2 is set so that one step PTP time stamping will be enabled.	
10		Hardware Tx time stamping is done. Residence time is calculated by the current delay_timer – Rx_timestamp	
11		Residence time is added to the correction field of the Delay_Req message. Send out the Delay_Req frame with updated FCS	
12	Get T4 upon reception of the Delay_Req message		
13	Send Delay_Resp message with recieveTime stamp = T4 (from step 12); correctField = correctionField copied from the received Delay_Req (from step 12).		

Step	Ordinary Clock (one-step master)	Caracal as One-Step E2E Transparent Clock	Ordinary Clock (slave)
14		Forward Delay_Resp message to egress without IS2 action.	
15			Software calculates T4 = receiveTime stamp of Delay_Resp – correctionField of Delay_Resp.
16			Software calculates meanPathDelay = ((T2-T3) + (T4-T1))/2 and timeOffset = ((T2-T1) - (T4-T3))/2

Table 4 One-step End-to-End Transparent Clock with Two-Step Master

Step	Ordinary Clock (Two-Step Master)	Caracal as a One-Step E2E Transparent Clock	Ordinary Clock (Slave)
1	Master sends out Sync with twoStepFlag = TRUE.		
2		Hardware Rx time stamping is done upon reception of the Sync message.	
3		Sync message is forwarded to the egress together with the Rx_timestamp. IS2 is set so that one step PTP time stamping will be performed.	
4		Hardware Tx time stamping is done at the egress. Residence time is calculated by the current delay_timer – Rx_timestamp.	
5		Residence time is added to the correction field of the Sync message. Send out the sync frame with updated FCS	
6			Redirect Sync to CPU and get T2
7	Master sends out Follow_Up message		
8		Forward Follow_Up message to egress without IS2 action.	
9			Redirect Follow_Up message to the CPU and calculate T1 combined with information from the Sync message (Step 6) and the Follow_Up message.
10			Send out Delay_Req message with correction field cleared and store T3.

Step	Ordinary Clock (Two-Step Master)	Caracal as a One-Step E2E Transparent Clock	Ordinary Clock (Slave)
11		Hardware Rx time stamping is done upon reception of the Delay_Req message.	
12		Delay_Req message is forwarded to the egress together with the Rx_timestamp. IS2 is set so that one step PTP processing will be performed.	
13		Hardware Tx time stamping is done at the egress. Residence time is calculated by the current delay_timer – Rx_timestamp.	
14		Residence time is added to the correction field of the Delay_Req message. Send out the Delay_Req frame with updated FCS.	
15	Get T4 upon reception of the Delay_Req message.		
16	Send Delay_Resp message with recieveTime stamp = T4 (from step 15); correctionField = correctionField copied from the received Delay_Req (from step 15).		
17		Forward Delay_Resp message to the egress without IS2 actions.	
18			Software calculates T4 = receiveTime stamp of Delay_Resp – correctionField of Delay_Resp.
19			Software calculates meanPathDelay = ((T2-T3) + (T4-T1))/2 and timeOffset = ((T2-T1) - (T4-T3))/2.

Table 5 Two-Step End-to-End Transparent Clock with One-Step Master

Step	Ordinary Clock (one-step master)	Caracal as Two-Step E2E Transparent Clock	Ordinary Clock (slave)
1	Master sends out Sync with twoStepFlag = FALSE		

Step	Ordinary Clock (one-step master)	Caracal as Two-Step E2E Transparent Clock	Ordinary Clock (slave)
2		Hardware Rx time stamping is done upon reception of the Sync message. IS2 is set so that the Sync message is redirected to the CPU with Rx_timestamp in the FIFO.	
3		Software checks the twoStepFlag and prepares to send a new Sync message with originTime stamp and correctionField copied from the received Sync (step 2) but with twoStepFlag = TRUE.	
4		Forward the new Sync message as soon as possible to the egress. IS2 is set so that two step PTP processing will be enabled.	
5		Hardware Tx time stamping is done and the Tx_timestamp is stored in the FIFO queue.	
6			Redirect Sync to the CPU and store T2.
7		Get Tx_timestamp from the FIFO queue and prepare a Follow_Up message with PreciseTime stamp = OriginTime stamp copied from the received Sync message (from step 2); correctionField = Tx_timestamp – Rx_timestamp (from step 2). Send out the Follow_Up message with no IS2 action.	
8			Redirect Follow_Up message to the CPU and calculate T1 based on the Sync message (Step 6) and the Follow_Up message time fields.
9			Send out Delay_Req message with correction field cleared and store T3.
10		Hardware Rx time stamping is done upon reception of the Delay_Req message.	
11		IS2 set to forward the Delay_Req to the egress while at the same time copy Delay_Req message to CPU. Two-step PTP time stamping enabled.	

Step	Ordinary Clock (one-step master)	Caracal as Two-Step E2E Transparent Clock	Ordinary Clock (slave)
12		IS2 set to forward the Delay_Req to the egress while at the same time copy Delay_Req message to CPU. Two-step PTP time stamping enabled.	
13		Software reads the residence time from the FIFO queue and waits for the associated Delay_Resp to come.	
14	Gets T4 upon reception of the Delay_Req message.		
15	Send Delay_Resp message with recieveTimestamp = T4 (from step 14). correctField = correctionField copied from the received Delay_Req (from step 14).		
16		Redirect Delay_Resp message to the CPU. Add residence time (step 13) to the correctionField of the received Delay_Resp and send it to the egress without IS2 action.	
17			Software calculates T4 = receiveTimestamp of Delay_Resp – correctionField of Delay_Resp.
18			Software calculates meanPathDelay = ((T2-T3) + (T4-T1))/2 and timeOffset = ((T2-T1) - (T4-T3))/2

Table 6 Two-Step End-to-End Transparent Clock with Two-Step Master

Step	Ordinary Clock (two-step master)	Caracal as a Two-Step E2E Transparent Clock	Ordinary Clock (slave)
1	Master sends out Sync with twoStepFlag = TRUE.		
2		Hardware Rx time stamping is done upon reception of the Sync message. IS2 is set so that the Sync message is redirected to the CPU with Rx_timestamp in the FIFO.	

Step	Ordinary Clock (two-step master)	Caracal as a Two-Step E2E Transparent Clock	Ordinary Clock (slave)
3		Software checks the twoStepFlag. When twoStepFlag = TRUE there is no need to generate a new Sync message. Software reads the Rx_timestamp from the FIFO and forwards the Sync message unchanged to the egress. IS2 is set to enable two-step PTP time stamping.	
4		Hardware Tx time stamping is done when the frame is sent out. Tx_timestamp is stored in the FIFO queue.	
5		Software reads the Tx_timestamp from the FIFO queue and calculates residence time = Tx_timestamp – Rx_timestamp (from step 3) and waits for the associated Follow_Up message.	
6			Redirect Sync to CPU and store T2.
7	Master sends out the Follow_Up message.		
8		IS2 is set so that the Follow_Up is redirected to the CPU. Software adds residence time (step 5) to the correctionField of the Follow_Up and sends it out to the egress without IS2 actions.	
9			Redirect the Follow_Up message to the CPU and calculate T1 based on the Sync message (Step 6) and the Follow_Up message.
10			Send out Delay_Req message with correction field cleared and store T3.
11		Hardware Rx time stamping is done upon reception of the Delay_Req message.	
12		IS2 is set to forward the Delay_Req to the egress while at the same time copying the Delay_Req message to the CPU. Two-step PTP time stamping is enabled.	

Step	Ordinary Clock (two-step master)	Caracal as a Two-Step E2E Transparent Clock	Ordinary Clock (slave)
13		Hardware Tx timestamping is done on the egress. Residence time is calculated to be the current delay timer - Rx_timestamp (from step 11) and stored in the FIFO.	
14		Software reads the residence time from the FIFO queue and waits for the associated Delay_Resp to come.	
15	Get T4 upon reception of the Delay_Req message.		
16	Send Delay_Resp message with recieveTime stamp = T4 (from step 15); correctionField = correctionField copied from the received Delay_Req (from step 15).		
17		Redirect Delay_Resp message to CPU. Add residence time (step 14) to the correctionField of the received Delay_Resp and send it to the egress without IS2 action.	
18			Software calculates T4 = receiveTime stamp of Delay_Resp – correctionField of Delay_Resp.
19			Software calculates meanPathDelay = ((T2-T3) + (T4-T1))/2 and timeOffset = ((T2-T1) - (T4-T3))/2.

Table 7 Peer Delay Mechanism with One-Step Responder

Step	Caracal as a Delay Requestor		Caracal as a One-Step Delay Responder
1	Latch ToD counter and delay timer.		
2	Prepare Pdelay_Req message and send to the egress. originTime stamp = 0 or ToD counter from step1; CorrectionField = 0;		
3	IS2 should be set to enable two-step PTP time stamping for the Pdelay_Req frame, IS2_ACTION.PTP_ENA[1] is set.		

Step	Caracal as a Delay Requestor		Caracal as a One-Step Delay Responder
4	Hardware time stamping is done on the egress when the first byte of the Pdelay_Req starts transmission. Tx_timestamp is stored in the time stamp FIFO queue.		
5	Software reads Tx_timestamp from the FIFO. T1 is calculated to be ToD counter (from step 1) + Tx_timestamp (from step 4) – latched delay timer (from step 1).		
6		Pdelay_Req transmission.	
7			Hardware Rx time stamping is done and IS2 is set to redirect Pdelay_Req to the CPU and two step PTP time stamping is performed so that Rx_timestamp is stored in the FIFO.
8			Software reads the Rx_timestamp from the time stamp FIFO queue.

9			Software prepares and sends Pdelay_Resp message as soon as possible with:
			twoStepFlag = FALSE;
			requestReceiptTime stamp = 0;
			correctionField = correctionField copied from Pdelay_Req (step 7) – Rx_timestamp (from step 8).

10			IS2 is set to enable one step PTP processing for the Pdelay_Resp frame.
11			Hardware Tx time stamping is done when the Pdelay_Resp is starts transmission and residence time is calculated to be the Tx_timestamp because it is a CPU inserted PTP frame.
12			Hardware adds the residence time to the correctionField of the Pdelay_resp message so the correction field becomes the original correctionField + (Tx_timestamp – Rx_timestamp) – which is the turnaround time of T3-T2. Hardware makes any needed corrections to the FCS and completes transmission of the Pdelay_Resp message.
13		Pdelay_Resp transmission	

Step	Caracal as a Delay Requestor		Caracal as a One-Step Delay Responder
14	Hardware Rx time stamping is done upon reception of the Pdelay_Resp message.		
15	IS2 is set so that the Pdelay_Resp is redirected to the CPU with two step PTP time stamping so the Rx_timestamp will be stored in the FIFO.		
16	Software reads the Rx_timestamp from the FIFO and latches the ToD counter and delay_timer. Calculate, T4 = latched ToD counter – (latched delay_timer – Rx_timestamp).		
17	Software calculates meanPathDelay (link delay) = ((T4-T1) - correctionField of Pdelay_Resp)/2.		

When the Peer Delay mechanism is used, there is no need for a slave clock to send delay request messages anymore. The slave can calculate: the timeOffset by (T2-T1) – meanPathDelay (derived through the peer delay mechanism). T2 and T1 can be obtained following the same steps as described in Table 1 and Table 2.

Table 8 Peer Delay Mechanism with Two-Step Responder

Step	Caracal as a Delay Requestor		Caracal as a Two-Step Delay Responder
1	Software latches the ToD counter and delay timer.		
2	Prepare Pdelay_Req message and send it to the egress. originTime stamp = 0 or ToD counter from step1; Correction = 0.		
3	IS2 should be set to enable two-step PTP time stamping for the Pdelay_Req frame, IS2_ACTION.PTP_ENA[1] is set.		
4	Hardware time stamping is done on the egress when the first byte of the Pdelay_Req starts transmission. Tx_timestamp is stored in the time stamp FIFO queue.		
5	Software reads the Tx_timestamp from the FIFO. T1 is calculated = ToD counter (from step 1) + Tx_timestamp (from step 4) – latched delay timer (from step 1).		

Step	Caracal as a Delay Requestor		Caracal as a Two-Step Delay Responder
6		Pdelay_Req transmission.	
7			Hardware RX time stamping is done and IS2 is set to redirect Pdelay_Req to the CPU and two step PTP time stamping is performed so the Rx_timestamp is stored in the FIFO.
8			Software reads the Rx_timestamp from time stamp FIFO queue.
9			Software latches the ToD counter and delay_timer. T2 = latched ToD counter – (latched delay timer – Rx_timestamp).

10			Prepare and send Pdelay_Resp message as soon as possible with:
			twoStepFlag = TRUE;
			requestReceiptTime stamp = 0;
			correctionField = 0.

11			IS2 is set to enable two step PTP processing for the Pdelay_Resp frame.
12			Hardware Tx time stamping is done when the Pdelay_Resp starts transmission and the Tx_timestamp is captured by hardware and stored in the FIFO queue.
13			Software reads the Tx_timestamp from the FIFO queue, latches ToD counter and delay_timer. Calculate, T3 = latched ToD counter + (Tx_timestamp – latched delay timer).
14		Pdelay_Resp transmission.	
15	Hardware RX time stamping is done upon reception of the Pdelay_Resp message.		
16	IS2 is set so that the Pdelay_Resp is redirected to the CPU with two step PTP time stamping so the Rx_timestamp will be stored in the FIFO.		

Step	Caracal as a Delay Requestor		Caracal as a Two-Step Delay Responder
17	Software reads the Rx_timestamp and latches the ToD counter and delay_timer. Calculate T4 = latched ToD counter – (latched delay_timer – Rx_timestamp).		
18			Software prepares and sends the Pdelay_Resp_Follow_Up massage with: responseOriginTime stamp = 0; correctionField = correctionField copied from Pdelay_Req (step 7) + (T3-T2). No IS2 actions are set for the Pdelay_Resp_Follow_Up message.
19		Pdelay_Resp_Follow_Up transmission.	
20	Software calculates meanPathDelay (link delay) = ((T4-T1)-correctionField of Pdelay_Resp)/2.		

When the Peer Delay mechanism is used there is no need for a slave clock to send delay request messages anymore. The slave can calculate: the timeOffset by (T2-T1) – meanPathDelay (derived through the peer delay mechanism). T2 and T1 can be obtained following the same steps as described in Table 1 and Table 2.

Table 9 One-Step Peer-to-Peer Transparent Clock with One-Step Master

Step	Ordinary Clock (one-step master)	Caracal as a One-Step P2P Transparent Clock	Ordinary Clock (slave)
1	Master sends out Sync with twoStepFlag = FALSE.		
2		IS2 is set to redirect the Sync message to the CPU and two step PTP time stamping is enabled for the Sync message.	
3		Software reads the Rx_timestamp from the FIFO and forwards the Sync message to the egress port with link_Delay_TC (from the peer delay mechanism) – Rx_timestamp added to the correctionField of the Sync message.	
4		Sync message is forwarded to the egress together with one step PTP time stamping enabled.	
5		Hardware Tx time stamping is done. Residence time is calculated to be the Tx_timestamp.	

Step	Ordinary Clock (one-step master)	Caracal as a One-Step P2P Transparent Clock	Ordinary Clock (slave)
6		Tx_timestamp is added to the correction field of the Sync message. So the correction field becomes: Original correction field + link_Delay_TC + Tx_timestamp – Rx_timestamp. Send out the sync frame with an updated FCS.	
7			Get T1 and T2 which are stored by software.
8			Software calculates timeOffset = (T2-T1) - link_Delay_Slave (from peer delay mechanism).

One-step/two-step peer-to-peer transparent clock responses to Sync messages are similar to one-step/two-step end-to-end transparent clock messages, the only difference being that the residence time should also include the link delay (or the peer-to-peer meanPathDelay) from peer delay mechanism described in Table 7and Table 8. The link delay process operates in parallel with flowcharts seen in Table 9 and Table 12.

Table 10 One-Step Peer-to-Peer Transparent Clock with Two-Step Master

Step	Ordinary Clock (two-step master)	Caracal as a One-Step P2P Transparent Clock	Ordinary Clock (slave)
1	Master sends out Sync with twoStepFlag = TRUE.		
2		IS2 is set to redirect the Sync message to the CPU and two step PTP time stamping is enabled for the Sync message.	
3		Software reads the Rx_timestamp from the FIFO and forwards the Sync message to the egress port with link_Delay_TC (from peer delay mechanism) – Rx_timestamp added to the correctionField of the Sync message.	
4		Sync message is forwarded to the egress together with one step PTP time stamping enabled.	
5		Hardware Tx time stamping is done., Residence time is calculated to be the Tx_timestamp.	
6		Tx_timestamp is added to the correction field of the Sync message. So the correction field becomes: Original correction field + link_Delay_TC + Tx_timestamp – Rx_timestamp. Send out the sync frame with an updated FCS.	

Step	Ordinary Clock (two-step master)	Caracal as a One-Step P2P Transparent Clock	Ordinary Clock (slave)
7			Redirect Sync to the CPU and get T2.
8	Master sends out a Follow_Up message.		
9		Forward Follow_Up message to the egress without IS2 action.	
10			Redirect Follow_Up message to the CPU and calculate T1 based on the Sync message (Step 7) and the Follow_Up message time fields.
11			Software calculates, timeOffset = (T2-T1)- link_Delay_Slave (from peer delay mechanism)

Table 11 Two-Step Peer-to-Peer Transparent Clock with One-Step Master

Step	Ordinary Clock (one-step master)	Caracal as a Two-Step P2P Transparent Clock	Ordinary Clock (slave)
1	Master sends out Sync with twoStepFlag = FALSE.		
2		Hardware Rx time stamping is done upon reception of the Sync message. IS2 is set so that the Sync message is redirected to the CPU with Rx_timestamp in the FIFO.	
3		Software checks the twoStepFlag and prepares and sends a new Sync message with originTime stamp and correctionField copied from the received Sync (step 2) but with twoStepFlag = TRUE.	
4		Forward the new Sync message as soon as possible to the egress. IS2 is set so that two-step PTP processing will be enabled.	
5		Hardware Tx time stamping is done. The Tx_timestamp is stored in the FIFO queue.	
6			Redirect Sync to CPU and store T2.

Step	Ordinary Clock (one-step master)	Caracal as a Two-Step P2P Transparent Clock	Ordinary Clock (slave)
7		Get the Tx_timestamp from the FIFO queue and prepare a Follow_Up message with: PresiceTime stamp = OriginTime stamp copied from the received Sync message (step 2); correctionField = link_Delay_TC (from peer delay mechanism) + Tx_timestamp – Rx_timestamp (from step 2). Send out the Follow_Up message with no IS2 action.	
8			Redirect the Follow_Up message to the CPU and calculate T1 based on the Sync message (Step 6) and the Follow_Up message time fields.
9			Software calculates timeOffset = (T2-T1)- link_Delay_Slave (from peer delay mechanism)

Table 12 Two-Step Peer-to-Peer Transparent Clock with Two-Step Master

Step	Ordinary Clock (two-step master)	Caracal as a Two-Step P2P Transparent clock	Ordinary Clock (slave)
1	Master sends out Sync with twoStepFlag = TRUE.		
2		Hardware Rx time stamping is done upon reception of the Sync message. IS2 is set so that the Sync message is redirected to the CPU with Rx_timestamp in the FIFO.	
3		When twoStepFlag = TRUE there is no need to generate a new Sync message. Software reads the Rx_timestamp from the FIFO and forwards the Sync message unchanged to the egress. IS2 is set to enable two-step PTP time stamping.	
4		Hardware Tx time stamping is done. The Tx_timestamp is stored in the FIFO queue.	
5			Redirect Sync to the CPU and store T2.
6		Software reads the Tx_timestamp from the FIFO queue and calculates the residence time = Tx_timestamp – Rx_timestamp (Step 3) and waits for the associated Follow_Up message.	
7	Master sends out Follow_Up.		

Step	Ordinary Clock (two-step master)	Caracal as a Two-Step P2P Transparent clock	Ordinary Clock (slave)
8		IS2 is set so that the Follow_Up is redirected to the CPU. Software adds residence time (step 6) + link_Delay_TC (from peer delay mechanism) to the correctionField of the Follow_Up and sends it out to the egress without IS2 actions.	
9			Redirect Follow_Up message to the CPU and calculate T1 based on the Sync message (step 5) and the Follow_Up message time fields.
10			Software calculates timeOffset = (T2-T1)- link_Delay_Slave (from peer delay mechanism).