ENT-AN1132
Application Note
Transparent Clock Mode Support

September 2018

Microsemi

a Microchip company

Contents

1 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 1.1

Revision 1.1 of this document was published in September 2018. In revision 1.1, the document title and template were updated.

1.2 Revision 1.0

Revision 1.0 of this document was published in September 2014. This was the first publication.

2 Transparent Clock Mode Support

The IEEE 1588v2 specification introduces transparent clocks as an alternative to implementing boundary clocks for multiport devices such as bridges, switches, or routers. Transparent clocks update a newly introduced time-interval field (correction field) within PTP event messages. This 64-bit correction field allows for switch delay compensation to a potential accuracy of less than one picosecond.

There are two types of transparent clocks: end-to-end and peer-to-peer. End-to-end transparent clocks update the time interval field for the delay associated with individual packet transfers. Peer-to-peer transparent clocks measure the line delay associated with the ingress transmission path and include this delay in the correction field.

2.1 Residence Time Update

Both types of transparent clocks need to calculate and update the residence time to the PTP event messages according to IEEE 1588v2. The procedures to calculate the residence time in Microsemi physical layer devices are discussed in the following sections.

Microsemi's 1588 IP supports three methods to update residence time in PTP event messages for transparent clocks. These three methods are named Mode A, Mode B, and Mode C. The 1588 block can be configured to operate in any of these modes to perform residence time calculations for transmit clocks. Mode A and Mode B are supported in Microsemi's 1588 Generation 1 devices, while Generation 2 devices support only Mode B and Mode C.

2.1.1 Mode A

Residence time calculation in Mode A involves subtracting the ingress time stamp from the correction field upon ingress and adding the egress time stamp to the correction field at egress. In Mode A operation, ingress and egress time stamps are 44-bit operands.

Ingress: CF = CF – Ingress Time Stamp in ns

Egress: CF = CF + Egress Time Stamp in ns

To protect against rollovers in Mode A operation, the MSB of the correction field (after the sign bit – the correction field is a 64-bit signed integer fractional ns; bit 63 is the sign bit) is used during ADD and SUBTRACT operations to signal if a rollover has occurred between the physical layer devices.

Generation 1 IEEE 1588 IP operates in 44-bit mode whereas Generation 2 Revision A IP operates in 46-bit mode. These modes are not compatible.

2.1.1.1 Generation 1 IP

The Rx and Tx time stamp is converted to ns.

[ToD_s*10^9+ToD_ns] = TS[47:0], which is taken with a resolution of 44 bits (for instance, TS[43:0]) for timestamping in the CF. TS[44] is kept in CF[62] at ingress and is compared with egress TS[44] to indicate a rollover. Stated in pseudo code, rollover means:

```
if (ingress_CF[62] != egress_TS[44]) // add 0x1000_0000_0000_0000 to egress CF
   egress_CF = ingress_CF + egress_TS[43:0] + 0x1000_0000_0000_0000
else // if no rollover
   egress_CF = ingress_CF + egress_TS[43:0]
```

2.1.1.2 Generation 2, Revision A 1588 IP

In Generation 2 IP, 1588 resolution is increased to 46 bits.

[ToD_s*10^9+ToD_ns] = TS[47:0], which is taken with a resolution of 46 bits (for instance, TS[45:0]) in the calculations. TS[46] is kept in CF[62] at ingress and is compared with egress TS[46] to indicate a rollover. Stated in pseudo code, rollover means:

```
if (ingress_CF[62] != egress_TS[46]) // add 0x4000_0000_0000_0000 to egress CF
   egress_CF = ingress_CF + egress_TS[45:0] + 0x4000_0000_0000_0000
else // if no rollover
   egress_CF = ingress_CF + egress_TS[45:0]
```

2.1.2 Mode B

This mode saves the ingress time stamp in the reserved bytes of the PTP header, just as is done in boundary clock and ordinary clock modes. This mode also performs the residence time calculation using the ingress time stamp stored in reserved bytes at the egress PHY. The calculated residence time is added to the correction field of the PTP frame and the reserved bytes are cleared back to 0.

Ingress: Reserved Bytes = Sub Seconds part of Ingress Time Stamp in ns

Egress: CF = CF + Egress Time Stamp in ns – Reserved Bytes, after which the Reserved Bytes = 0

2.1.3 Mode C

Residence time calculations in Mode C are similar to Mode A, but perform a 48-bit nanosecond operation according to the IEEE 1588v2 standard. All 1588 Generation 2, Revision B IP systems support this method.

Residence time calculation in Mode C involves subtracting the ingress time stamp from the correction field at ingress and adding the egress time stamp to the correction field at egress.

Ingress: CF = CF – Ingress Time Stamp in ns

Egress: CF = CF + Egress Time Stamp in ns

Note: When running in TC Mode C, there will be no need to perform any rollover protection, as the addition and subtraction are done in a 2-complement calculation.

2.2 End-to-End Transparent Clocks

End-to-end transparent clocks add the residence time (the time it takes to traverse the system from the input to the output port(s)) to all Sync and Delay_req frames. Corrections will be made for the propagation of both sync and Delay_Req messages.

The following table lists the actions of end-to-end (E2E) one step transparent clock for Mode A, Mode B, and Mode C.

Table 1 • E2E One Step Transparent Clocks

Clock Type/Mode	Ingress	Egress
TC 1 step (Mode A/Mode C)	**Sync, PdelayResp:** sub(Rx_timestamp, correctionField); add(Asymmetry, correctionField)	**Sync, PdelayResp:** add(Tx_timestamp, correctionField)
	DelayReq, PdelayReq: sub(Rx_timestamp, correctionField)	**DelayReq, PdelayReq:** add(Tx_timestamp, correctionField); sub(Asymmetry, correctionField)
TC 1 step (Mode B)	**Sync, PdelayResp:** write(Rx_timestamp, Reserved); add(Asymmetry, correctionField)	**Sync, PdelayResp:** Subtract_add(Tx_timestamp, Reserved, correctionField)
	DelayReq, PdelayReq: write(Rx_timestamp, Reserved)	**DelayReq, PdelayReq:** Subtract_add (Tx_timestamp - Asymmetry, Reserved, correctionField)
TC 1 step (Mode B)	**Sync, PdelayResp:** write(Rx_timestamp, Reserved); add(Asymmetry, correctionField)	**Sync, PdelayResp:** Subtract_add(Tx_timestamp, Reserved, correctionField)
	DelayReq, PdelayReq: write(Rx_timestamp, Reserved)	**DelayReq, PdelayReq:** Subtract_add(Tx_timestamp - Asymmetry, Reserved, correctionField)

2.3 Peer-to-Peer Transparent Clocks

Peer-to-peer transparent clocks measure the local link delays using the peer delay mechanism rather than using the delay request mechanism to measure full path delay. Peer-to-peer transparent clocks must determine the residence time and make corrections to sync messages. In addition, corrections must be included for inbound path delays as measured using the peer delay mechanism.

The following table lists the actions of a peer-to-peer (P2P) one step transparent clock for Mode A, Mode B, and Mode C.

Table 2 • P2P One Step Transparent Clocks

Clock Type/Mode	Ingress	Egress

TC 1 step
 	
(Mode A/Mode C)
	Sync:
	
	sub(Rx_timestamp, correctionField); add (PathDelay + Asymmetry, correctionField);
	
	Pdelay_Req:
	
	sub(Rx_timestamp, correctionField)
	
	Pdelay_Resp:
	
	write(Rx_timestamp, Reserved); add (Asymmetry, correctionField);
		Sync, Pdelay_Resp:
		
		add(Tx_timestamp, correctionField)
		
		Pdelay_Req:
		
		save(Tx_timestamp - Asymmetry, TxFiFo)

TC 1 step (Mode B)	**Sync:**
	
	write(Rx_timestamp, Reserved); add(PathDelay + Asymmetry, correctionField);
	
	Pdelay_Req:
	
	write(Rx_timestamp, Reserved)
	
	Pdelay_Resp: write(Rx_timestamp, Reserved); add (Asymmetry, correctionField);
		Sync, Pdelay_Resp: Subtract_add (Tx_timestamp, Reserved, correctionField)
		
		Pdelay_Req:
		
		save(Tx_timestamp - Asymmetry, TxFiFo)

Note: Because the Delay_Req mechanism is not supported on P2P TCs, no special processing is required for Delay_Req messages.

2.4 Initializing the 1588 Block to Operate TCs in Mode C

The following configuration example shows how to initialize the 1588 IP block to operate in Mode C for transparent clock applications. `vtss_phy_ts_tc_op_mode_t`, a member of `vtss_phy_ts_init_conf_t`, can be assigned to a desired mode during the 1588 block initialization.

```
vtss_rc vtss_1588_init_for_mode_C(const vtss_inst_t inst, const vtss_port_no_t port_no )
{
   vtss_phy_ts_init_conf_t conf; vtss_rc rc;
   do {
      /* reference clock frequency */
      conf.clk_freq          = VTSS_PHY_TS_CLOCK_FREQ_15625M;
      /* Clock Source */
      conf.clk_src           = VTSS_PHY_TS_CLOCK_SRC_EXTERNAL;
      /* 10byte Full Tx timestamp */
      conf.tx_ts_len         = VTSS_PHY_TS_FIFO_TIMESTAMP_LEN_10BYTE;
      /* Timestamps to pushed out on the SPI interface */
      conf.tx_fifo_mode      = VTSS_PHY_TS_FIFO_MODE_SPI;
      /* Rx timestamp in PTP reserved bytes */
      conf.rx_ts_pos         = VTSS_PHY_TS_RX_TIMESTAMP_POS_IN_PTP;
      /* Rx timestamp update nano seconds part */
      conf.rx_ts_len         = VTSS_PHY_TS_RX_TIMESTAMP_LEN_30BIT;
      /* TC operating mode is Mode C */
      conf.tc_op_mode        = VTSS_PHY_TS_TC_OP_MODE_C;

      rc = vtss_phy_ts_init(NULL, iport_no, &conf);
      if (rc == VTSS_RC_OK) {
         printf("PHY TS Init Success  \n");
      } else {
         printf("PHY TS Init Failed   \n");
         break;
      }
      rc = vtss_phy_ts_mode_set (NULL, port_no, TRUE);
      if (rc != VTSS_RC_OK) {
         printf("PHY TS Block Enable Failed \n");
         break;
      }
   }
   return rc;
}
```

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

VPPD-03802