Precision Time Protocol Master

Component Design Document

1 Description

This is the Precision Time Protocol (PTP) Master component. This component implements the master portion of the protocol. Any PTP slaves can use the messages from this component to measure their system time relative to the master, or use the master to synchronize their clocks. Additional documentation on the PTP protocol can be found in the doc/research/ directory for this component.

2 Requirements

The requirements for the Precision Time Protocol Master are specified below.

1. The component shall send the Sync and Follow-Up PTP time messages according to the PTP protocol.
2. The component shall send a Delay-Response PTP time message in response to a received Delay-Request message according to the PTP protocol.
3. The component shall conduct the PTP transactions at a compile-time configurable rate.
4. The component shall send a time PTP Sync message one time on command.
5. The component shall produce data products relating the number times the Sync, Follow-Up, and Delay-Response messages have been sent since startup.
6. The component shall produce a data product relating the number times the Delay-Request message have been received since startup.

3 Design

3.1 At a Glance

Below is a list of useful parameters and statistics that give a quick look into the makeup of the component.

- **Execution** - *active*
- **Number of Connectors** - 9
- **Number of Invokee Connectors** - 4
- **Number of Invoker Connectors** - 5
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 3

- **Number of Parameters** - *None*
- **Number of Events** - 7
- **Number of Faults** - *None*
- **Number of Data Products** - 5
- **Number of Data Dependencies** - *None*
- **Number of Packets** - *None*

3.2 Diagram

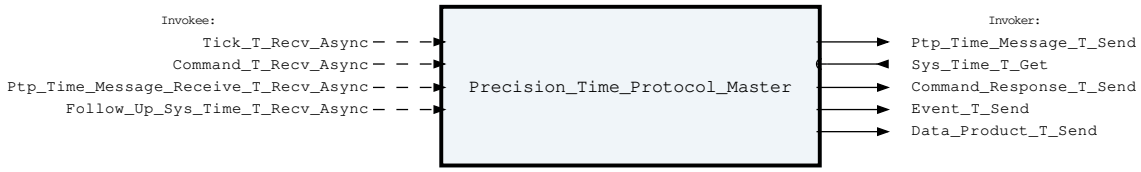

Figure 1: Precision Time Protocol Master component diagram.

3.3 Connectors

Below are tables listing the component's connectors.

3.3.1 Invokee Connectors

The following is a list of the component's *invokee* connectors:

Table 1: Precision Time Protocol Master Invokee Connectors

Name	Kind	Type	Return_Type	Count
Tick_T_Recv_Async	recv_async	Tick.T	-	1
Command_T_Recv_Async	recv_async	Command.T	-	1
Ptp_Time_Message_Receive_T_Recv_Async	recv_async	Ptp_Time_Message_Receive.T	-	1
Follow_Up_Sys_Time_T_Recv_Async	recv_async	Sys_Time.T	-	1

Connector Descriptions:

- **Tick_T_Recv_Async** - Tick input that is used to calculate time message frequency. The sync_period can be used to set how many ticks should be received before the master clock starts the update process.
- **Command_T_Recv_Async** - The command receive connector.
- **Ptp_Time_Message_Receive_T_Recv_Async** - Used to receive the Delay_Request messages from PTP slaves. These messages can be timestamped with their receive time by a component external to this system. If not timestamped (ie. seconds and microseconds set to zero) the component will timestamp the messages as it receives them. This timestamp will be relayed back to the slaves in a Delay_Response message.

- **Follow_Up_Sys_Time_T_Recv_Async** - Receives and forwards the PTP time for a follow-up message (if connected). The time received is an accurate time stamp of when the Sync message was sent. This time can be provided to this component by a lower level component which actually records the time the Sync message leaves the system. If left unconnected, then no follow-up message will be sent by this component. In this case the PTP slave should use the timestamp provided in the Sync message instead of the follow-up message.

3.3.2 Internal Queue

This component contains an internal first-in-first-out (FIFO) queue to handle asynchronous messages. This queue is sized at initialization as a configurable number of bytes. Determining the size of the component queue can be difficult. The following table lists the connectors that will put asynchronous messages onto the queue, and the maximum sizes of each of those messages on the queue. Note that each message put onto the queue also incurs an overhead on the queue of 5 additional bytes, which is included in the max message size below:

Table 2: Precision Time Protocol Master Asynchronous Connectors

Name	Type	Max Size (bytes)
Tick_T_Recv_Async	Tick.T	17
Command_T_Recv_Async	Command.T	265
Ptp_Time_Message_Receive_T_Recv_Async	Ptp_Time_Message_Receive.T	24
Follow_Up_Sys_Time_T_Recv_Async	Sys_Time.T	13

If you are unsure how to size the queue of this component, it is recommended that you make the queue size a multiple of the largest size found above.

3.3.3 Invoker Connectors

The following is a list of the component's *invoker* connectors:

Table 3: Precision Time Protocol Master Invoker Connectors

Name	Kind	Type	Return_Type	Count
Ptp_Time_Message_T_Send	send	Ptp_Time_Message.T	-	1
Sys_Time_T_Get	get	-	Sys_Time.T	1
Command_Response_T_Send	send	Command_Response.T	-	1
Event_T_Send	send	Event.T	-	1
Data_Product_T_Send	send	Data_Product.T	-	1

Connector Descriptions:

- **Ptp_Time_Message_T_Send** - Sends PTP Sync messages, starting a PTP transaction.
- **Sys_Time_T_Get** - Used to get system time.
- **Command_Response_T_Send** - This connector is used to register and respond to the components commands.
- **Event_T_Send** - The event send connector, sends events.
- **Data_Product_T_Send** - The data product invoker connector.

3.4 Interrupts

This component contains no interrupts.

3.5 Initialization

Below are details on how the component should be initialized in an assembly.

3.5.1 Component Instantiation

This component contains no instantiation parameters in its discriminant.

3.5.2 Component Base Initialization

This component achieves base class initialization using the `init_Base` subprogram. This subprogram requires the following parameters:

Table 4: Precision Time Protocol Master Base Initialization Parameters

Name	Type
`Queue_Size`	`Natural`

Parameter Descriptions:

- **`Queue_Size`** - The number of bytes that can be stored in the component's internal queue.

3.5.3 Component Set ID Bases

This component contains commands, events, packets, faults, or data products that require a base identifier to be set at initialization. The `set_Id_Bases` procedure must be called with the following parameters:

Table 5: Precision Time Protocol Master Set Id Bases Parameters

Name	Type
`Command_Id_Base`	`Command_Types.Command_Id_Base`
`Data_Product_Id_Base`	`Data_Product_Types.Data_Product_Id_Base`
`Event_Id_Base`	`Event_Types.Event_Id_Base`

Parameter Descriptions:

- **`Command_Id_Base`** - The value at which the component's command identifiers begin.
- **`Data_Product_Id_Base`** - The value at which the component's data product identifiers begin.
- **`Event_Id_Base`** - The value at which the component's event identifiers begin.

3.5.4 Component Map Data Dependencies

This component contains no data dependencies.

3.5.5 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. The component achieves implementation class initialization using the `init` subprogram. The `init` subprogram requires the following parameters:

Table 6: Precision Time Protocol Master Implementation Initialization Parameters

Name	Type	Default Value
Sync_Period	Positive	1
Enabled_State	Ptp_State.Ptp_State_Type	Ptp_State.Enabled

Parameter Descriptions:

- **Sync_Period** - The number of ticks between sending precision time protocol messages.
- **Enabled_State** - Is precision time protocol enabled or disabled by default at startup.

3.6 Commands

Commands for the Precision Time Protocol Master component.

Table 7: Precision Time Protocol Master Commands

Local ID	Command Name	Argument Type
0	Enable_Precision_Time_Protocol	-
1	Disable_Precision_Time_Protocol	-
2	Sync_Once	-

Command Descriptions:

- **Enable_Precision_Time_Protocol** - This enables the sending of PTP messages.
- **Disable_Precision_Time_Protocol** - This disables the sending of PTP messages.
- **Sync_Once** - This sends a PTP sync message at the next tick, regardless of the current sync period. This is useful during testing to send a sync one time.

3.7 Parameters

The Precision Time Protocol Master component has no parameters.

3.8 Events

Events for the Precision Time Protocol Master component.

Table 8: Precision Time Protocol Master Events

Local ID	Event Name	Parameter Type
0	Unexpected_Message_Type	Ptp_Time_Message.T
1	Unexpected_Transaction_Count	Unexpected_Ptp_Transaction_Count.T
2	Ptp_Enabled	-
3	Ptp_Disabled	-
4	Syncing_Once	-
5	Queue_Overflowed	-
6	Invalid_Command_Received	Invalid_Command_Info.T

Event Descriptions:

- **Unexpected_Message_Type** - Received a message of unexpected type, message is sent as parameter.

- **Unexpected_Transaction_Count** - Received a message with an unexpected transaction count; message is sent as a parameter along with the expected transaction number.
- **Ptp_Enabled** - The PTP has been enabled by command.
- **Ptp_Disabled** - The PTP has been disabled by command.
- **Syncing_Once** - A command was received to complete a single PTP transaction at the next Tick.
- **Queue_Overflowed** - An incoming message was dropped due to the queue overflowing. The queue needs to be made larger.
- **Invalid_Command_Received** - A command was received with invalid parameters.

3.9 Data Products

Data products for the Precision Time Protocol Master component.

Table 9: Precision Time Protocol Master Data Products

Local ID	Data Product Name	Type
0x0000 (0)	Transaction_Number	Packed_U16.T
0x0001 (1)	Follow_Up_Messages_Sent	Packed_U16.T
0x0002 (2)	Delay_Request_Messages_Received	Packed_U16.T
0x0003 (3)	Unexpected_Messages_Received	Packed_U16.T
0x0004 (4)	Precision_Time_Protocol_State	Ptp_State.T

Data Product Descriptions:
- **Transaction_Number** - The transaction number of the last sent Sync message.
- **Follow_Up_Messages_Sent** - The number of follow up messages sent.
- **Delay_Request_Messages_Received** - The number of delay request messages received.
- **Unexpected_Messages_Received** - The number of received messages that had unexpected transaction numbers or message types.
- **Precision_Time_Protocol_State** - The disable/enable state of the precision time protocol within the component.

3.10 Packets

The Precision Time Protocol Master component has no packets.

4 Unit Tests

The following section describes the unit test suites written to test the component.

4.1 *Precision_Time_Protocol_Master_Tests* Test Suite

This is a unit test suite for the Precision Time Protocol Master component.

Test Descriptions:
- **Test_Time_Sync** - This test ensures that time syncing messages are sent out appropriately when Ticks are sent to the component.

- **Test_Follow_Up** - This test ensures that the Follow_Up message is sent out appropriately.
- **Test_Delay_Request** - This test ensures that the Delay_Response message is sent out when a Delay_Request is received.
- **Test_Unexpected_Message_Received** - This test ensures that the unexpected received messages are reported and not processed.
- **Test_Enable_Disable** - This test ensures that the enable/disable commands work as intended.
- **Test_Sync_Once** - This test ensures that the Sync_Once command works.
- **Test_Invalid_Command** - This test ensures that an invalid command is rejected and reported.
- **Test_Queue_Overflow** - This test ensures that a queue overflow is reported.

5 Appendix

5.1 Preamble

This component contains no preamble code.

5.2 Packed Types

The following section outlines any complex data types used in the component in alphabetical order. This includes packed records and packed arrays that might be used as connector types, command arguments, event parameters, etc..

Command.T:

Generic command packet for holding arbitrary commands

Table 10: Command Packed Record : 2080 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Command_ Header.T	-	40	0	39	-
Arg_Buffer	Command_ Types. Command_Arg_ Buffer_Type	-	2040	40	2079	Header.Arg_ Buffer_Length

Field Descriptions:
- **Header** - The command header
- **Arg_Buffer** - A buffer that contains the command arguments

Command_Header.T:

Generic command header for holding arbitrary commands

Table 11: Command_Header Packed Record : 40 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit

Source_Id	Command_Types. Command_Source_Id	0 to 65535	16	0	15
Id	Command_Types. Command_Id	0 to 65535	16	16	31
Arg_Buffer_Length	Command_Types. Command_Arg_Buffer_ Length_Type	0 to 255	8	32	39

Field Descriptions:

- **Source_Id** - The source ID. An ID assigned to a command sending component.
- **Id** - The command identifier
- **Arg_Buffer_Length** - The number of bytes used in the command argument buffer

Command_Response.T:

Record for holding command response data.

Table 12: Command_Response Packed Record : 56 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Source_Id	Command_ Types.Command_ Source_Id	0 to 65535	16	0	15
Registration_ Id	Command_ Types.Command_ Registration_ Id	0 to 65535	16	16	31
Command_Id	Command_Types. Command_Id	0 to 65535	16	32	47
Status	Command_Enums. Command_ Response_ Status.E	0 => Success 1 => Failure 2 => Id_Error 3 => Validation_Error 4 => Length_Error 5 => Dropped 6 => Register 7 => Register_Source	8	48	55

Field Descriptions:

- **Source_Id** - The source ID. An ID assigned to a command sending component.
- **Registration_Id** - The registration ID. An ID assigned to each registered component at initialization.
- **Command_Id** - The command ID for the command response.
- **Status** - The command execution status.

Data_Product.T:

Generic data product packet for holding arbitrary data types

Table 13: Data_Product Packed Record : 344 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Data_Product_ Header.T	-	88	0	87	-
Buffer	Data_Product_ Types.Data_ Product_ Buffer_Type	-	256	88	343	Header.Buffer_ Length

Field Descriptions:

- **Header** - The data product header
- **Buffer** - A buffer that contains the data product type

Data_Product_Header.T:

Generic data_product packet for holding arbitrary data_product types

Table 14: Data_Product_Header Packed Record : 88 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Id	Data_Product_Types. Data_Product_Id	0 to 65535	16	64	79
Buffer_Length	Data_Product_ Types.Data_Product_ Buffer_Length_Type	0 to 32	8	80	87

Field Descriptions:

- **Time** - The timestamp for the data product item.
- **Id** - The data product identifier
- **Buffer_Length** - The number of bytes used in the data product buffer

Event.T:

Generic event packet for holding arbitrary events

Table 15: Event Packed Record : 344 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Event_Header.T	-	88	0	87	-
Param_Buffer	Event_Types. Parameter_ Buffer_Type	-	256	88	343	Header.Param_ Buffer_Length

Field Descriptions:

- **Header** - The event header
- **Param_Buffer** - A buffer that contains the event parameters

Event_Header.T:

Generic event packet for holding arbitrary events

Table 16: Event_Header Packed Record : 88 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Id	Event_Types.Event_Id	0 to 65535	16	64	79
Param_Buffer_Length	Event_Types.Parameter_Buffer_Length_Type	0 to 32	8	80	87

Field Descriptions:

- **Time** - The timestamp for the event.
- **Id** - The event identifier
- **Param_Buffer_Length** - The number of bytes used in the param buffer

Invalid_Command_Info.T:

Record for holding information about an invalid command

Table 17: Invalid_Command_Info Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Command_Types.Command_Id	0 to 65535	16	0	15
Errant_Field_Number	Interfaces.Unsigned_32	0 to 4294967295	32	16	47
Errant_Field	Basic_Types.Poly_Type	-	64	48	111

Field Descriptions:

- **Id** - The command Id received.
- **Errant_Field_Number** - The field that was invalid. 1 is the first field, 0 means unknown field, 2**32 means that the length field of the command was invalid.
- **Errant_Field** - A polymorphic type containing the bad field data, or length when Errant_Field_Number is 2**32.

Packed_U16.T:

Single component record for holding packed unsigned 16-bit value.

Table 18: Packed_U16 Packed Record : 16 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Value	Interfaces.Unsigned_16	0 to 65535	16	0	15

Field Descriptions:

- **Value** - The 16-bit unsigned integer.

Ptp_State.T:

The precision time protocol disable/enable state. *Preamble (inline Ada definitions):*

```
type Ptp_State_Type is (Disabled, Enabled);
for Ptp_State_Type use (Disabled => 0, Enabled => 1);
```

Table 19: Ptp_State Packed Record : 8 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
State	Ptp_State_Type	0 => Disabled 1 => Enabled	8	0	7

Field Descriptions:

- **State** - Is precision time protocol enabled or disabled.

Ptp_Time_Message.T:

Record for the PTP Time Sync Message.

Table 20: Ptp_Time_Message Packed Record : 88 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Message_Type	Ptp_Enums.Ptp_Message_Type.E	0 => Sync 1 => Follow_Up 2 => Delay_Request 3 => Delay_Response	8	0	7
Transaction_Count	Interfaces.Unsigned_16	0 to 65535	16	8	23
Time_Stamp	Sys_Time.T	-	64	24	87

Field Descriptions:

- **Message_Type** - Type of message contained in this packet.
- **Transaction_Count** - Ptp transaction counter. This increments with every sync message.
- **Time_Stamp** - Message time information, meaning depends on message type.

Ptp_Time_Message_Receive.T:

Record for the PTP Time Sync Message. This record includes a time stamp which signifies when the message was received by the system. If the timestamp is set to zero, then time is retrieved at the receipt of this message by the component.

Table 21: Ptp_Time_Message_Receive Packed Record : 152 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Receive_Time	Sys_Time.T	-	64	0	63
Message	Ptp_Time_Message.T	-	88	64	151

Field Descriptions:

- **Receive_Time** - The time when the PTP message was received by the system. If this is set to zeros, then the component will grab a timestamp when receiving this message and assume that as the received time.
- **Message** - The PTP time message that was received.

Sys_Time.T:

A record which holds a time stamp using GPS format including seconds and subseconds since epoch (1-5-1980 to 1-6-1980 midnight).

Table 22: Sys_Time Packed Record : 64 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Seconds	Interfaces. Unsigned_32	0 to 4294967295	32	0	31
Subseconds	Interfaces. Unsigned_32	0 to 4294967295	32	32	63

Field Descriptions:
- **Seconds** - The number of seconds elapsed since epoch.
- **Subseconds** - The number of 1/(2^32) sub-seconds.

Tick.T:

The tick datatype used for periodic scheduling. Included in this type is the Time associated with a tick and a count.

Table 23: Tick Packed Record : 96 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Count	Interfaces. Unsigned_32	0 to 4294967295	32	64	95

Field Descriptions:
- **Time** - The timestamp associated with the tick.
- **Count** - The cycle number of the tick.

Unexpected_Ptp_Transaction_Count.T:

Record for the PTP Time Sync Message with unexpected transaction count.

Table 24: Unexpected_Ptp_Transaction_Count Packed Record : 104 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Message	Ptp_Time_Message.T	-	88	0	87
Expected_ Transaction_Count	Interfaces. Unsigned_16	0 to 65535	16	88	103

Field Descriptions:
- **Message** - The PTP message.

- **Expected_Transaction_Count** - The expected transaction count.

5.3 Enumerations

The following section outlines any enumerations used in the component.

Command_Enums.Command_Response_Status.E:

This status enumeration provides information on the success/failure of a command through the command response connector.

Table 25: Command_Response_Status Literals:

Name	Value	Description
Success	0	Command was passed to the handler and successfully executed.
Failure	1	Command was passed to the handler not successfully executed.
Id_Error	2	Command id was not valid.
Validation_Error	3	Command parameters were not successfully validated.
Length_Error	4	Command length was not correct.
Dropped	5	Command overflowed a component queue and was dropped.
Register	6	This status is used to register a command with the command routing system.
Register_Source	7	This status is used to register command sender's source id with the command router for command response forwarding.

Ptp_Enums.Ptp_Message_Type.E:

This is the message type enumeration for the Precision Time Protocol.

Table 26: Ptp_Message_Type Literals:

Name	Value	Description
Sync	0	The Sync message is sent from the master to the slave and initiates a PTP transaction.
Follow_Up	1	The Follow_Up message is sent from the master to the slave and provides a more accurate timestamp for the Sync message transmission time.
Delay_Request	2	The Delay_Request message is sent from the slave to the master to request the master send a Delay_Response message.
Delay_Response	3	The Delay_Response message is sent from the master to the slave and provides enough information for the slave to know its clock offset from the master.