Oscillator

Component Design Document

1 Description

This is the oscillator component.

2 Requirements

No requirements have been specified for this component.

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** - *passive*
- **Number of Connectors** - 7
- **Number of Invokee Connectors** - 3
- **Number of Invoker Connectors** - 4
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 3
- **Number of Parameters** - 3
- **Number of Events** - 6
- **Number of Faults** - *None*
- **Number of Data Products** - 1
- **Number of Data Dependencies** - *None*
- **Number of Packets** - *None*

3.2 Diagram

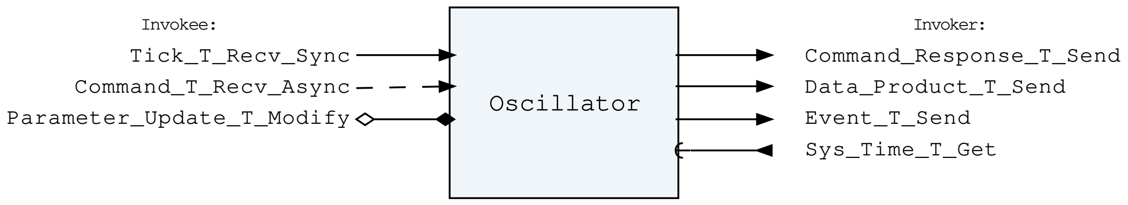

Figure 1: Oscillator component diagram.

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: Oscillator Invokee Connectors

Name	Kind	Type	Return_Type	Count
Tick_T_Recv_Sync	recv_sync	Tick.T	-	1
Command_T_Recv_Async	recv_async	Command.T	-	1
Parameter_Update_T_Modify	modify	Parameter_Update.T	-	1

Connector Descriptions:

- **`Tick_T_Recv_Sync`** - The schedule invokee connector
- **`Command_T_Recv_Async`** - The command receive connector
- **`Parameter_Update_T_Modify`** - The parameter update connector.

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: Oscillator Asynchronous Connectors

Name	Type	Max Size (bytes)
Command_T_Recv_Async	Command.T	265

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: Oscillator Invoker Connectors

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

Connector Descriptions:

- **`Command_Response_T_Send`** - This connector is used to register the components commands with the command router component.
- **`Data_Product_T_Send`** - The data product invoker connector
- **`Event_T_Send`** - The event send connector
- **`Sys_Time_T_Get`** - The system time is retrieved via this 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: Oscillator 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: Oscillator Set Id Bases Parameters

Name	Type
Parameter_Id_Base	Parameter_Types.Parameter_Id_Base
Event_Id_Base	Event_Types.Event_Id_Base
Data_Product_Id_Base	Data_Product_Types.Data_Product_Id_Base
Command_Id_Base	Command_Types.Command_Id_Base

Parameter Descriptions:

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

3.5.4 Component Map Data Dependencies

This component contains no data dependencies.

3.5.5 Component Implementation Initialization

This component contains no implementation class initialization, meaning there is no `init` subprogram for this component.

3.6 Commands

Commands for the Oscillator component

Table 6: Oscillator Commands

Local ID	Command Name	Argument Type
0	Set_Frequency	Packed_F32.T
1	Set_Amplitude	Packed_F32.T
2	Set_Offset	Packed_F32.T

Command Descriptions:

- **Set_Frequency** - Set the frequency of the oscillator in Hz
- **Set_Amplitude** - Set the amplitude of the oscillator
- **Set_Offset** - Set the Y offset of the oscillator

3.7 Parameters

Parameters for the Oscillator component

Table 7: Oscillator Parameters

Local ID	Parameter Name	Type	Default Value
0x0000 (0)	Frequency	Packed_F32.T	(Value=>0.175)
0x0001 (1)	Amplitude	Packed_F32.T	(Value=>5.0)
0x0002 (2)	Offset	Packed_F32.T	(Value=>0.0)

Parameter Descriptions:

- **Frequency** - The frequency of the oscillator in Hz
- **Amplitude** - The amplitude of the oscillator
- **Offset** - The Y offset of the oscillator

3.8 Events

Events for the oscillator component

Table 8: Oscillator Events

Local ID	Event Name	Parameter Type
0	Frequency_Value_Set	Packed_F32.T
1	Amplitude_Value_Set	Packed_F32.T
2	Offset_Value_Set	Packed_F32.T
3	Dropped_Command	Command_Header.T
4	Invalid_Command_Received	Invalid_Command_Info.T
5	Invalid_Parameter_Received	Invalid_Parameter_Info.T

Event Descriptions:

- **Frequency_Value_Set** - A new frequency value was set by command
- **Amplitude_Value_Set** - A new amplitude value was set by command
- **Offset_Value_Set** - A new offset value was set by command
- **Dropped_Command** - The component's queue overflowed and the command was dropped.
- **Invalid_Command_Received** - A command was received with invalid parameters.
- **Invalid_Parameter_Received** - A parameter was received with invalid parameters.

3.9 Data Products

Data products for the Oscillator component

Table 9: Oscillator Data Products

Local ID	Data Product Name	Type
0x0000 (0)	Oscillator_Value	Packed_F32.T

Data Product Descriptions:

- **Oscillator_Value** - The current value of the oscillator.

3.10 Data Dependencies

The Oscillator component has no data dependencies.

3.11 Packets

The Oscillator component has no packets.

3.12 Faults

The Oscillator component has no faults.

4 Unit Tests

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

4.1 *Oscillator_ Tests* Test Suite

This is a unit test suite for the oscillator component

Test Descriptions:

- **`Test_Parameters`** - This unit test exercises the parameters within the component
- **`Test_Parameter_Validation`** - This unit test exercises the implementable validation function

4.2 *Oscillator_ Tests* Test Suite

This is a unit test suite for the oscillator component

Test Descriptions:

- **`Test_Parameters`** - This unit test exercises the parameters within the component
- **`Test_Parameter_Validation`** - This unit test exercises the implementable validation function

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 to 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.

Invalid_Parameter_Info.T:

Record for holding information about an invalid parameter

Table 18: Invalid_Parameter_Info Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit

Id	Parameter_Types.Parameter_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 parameter 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 parameter was invalid.
- **`Errant_Field`** - A polymorphic type containing the bad field data, or length when Errant_Field_Number is 2**32.

Packed_F32.T:

Single component record for holding packed 32-bit floating point number.

Table 19: Packed_F32 Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Value	Short_Float	-3.40282e+38 to 3.40282e+38	32	0	31

Field Descriptions:

- **`Value`** - The 32-bit floating point number.

Parameter.T:

Generic parameter packet for holding a generic parameter

Table 20: Parameter Packed Record : 280 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Parameter_Header.T	-	24	0	23	-
Buffer	Parameter_Types.Parameter_Buffer_Type	-	256	24	279	Header.Buffer_Length

Field Descriptions:

- **`Header`** - The parameter header
- **`Buffer`** - A buffer to that contains the parameter type

Parameter_Header.T:

Generic parameter header for holding arbitrary parameters

Table 21: Parameter_Header Packed Record : 24 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Parameter_Types. Parameter_Id	0 to 65535	16	0	15
Buffer_Length	Parameter_Types. Parameter_Buffer_ Length_Type	0 to 32	8	16	23

Field Descriptions:

- **Id** - The parameter identifier
- **Buffer_Length** - The number of bytes used in the parameter type buffer

Parameter_Update.T:

A record intended to be used as a provide/modify connector type for updating/fetching parameters.

Table 22: Parameter_Update Packed Record : 296 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Operation	Parameter_ Enums. Parameter_ Operation_ Type.E	0 => Stage 1 => Update 2 => Fetch 3 => Validate	8	0	7	-
Status	Parameter_ Enums. Parameter_ Update_ Status.E	0 => Success 1 => Id_Error 2 => Validation_Error 3 => Length_Error	8	8	15	-
Param	Parameter. T	-	280	16	295	-

Field Descriptions:

- **Operation** - The parameter operation to perform.
- **Status** - The parameter return status.
- **Param** - The parameter that has been updated or fetched.

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 23: 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 24: 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.

5.3 Enumerations

The following section outlines any enumerations used in the component.

Command_Enums.Command_Response_Status.E:

This status enumerations 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.

Parameter_Enums.Parameter_Operation_Type.E:

This enumeration lists the different parameter operations that can be performed.

Table 26: Parameter_Operation_Type Literals:

Name	Value	Description
Stage	0	Stage the parameter.
Update	1	All parameters are staged, it is ok to update all parameters now.
Fetch	2	Fetch the parameter.
Validate	3	Validate the parameter.

Parameter_Enums.Parameter_Update_Status.E:

This status enumeration provides information on the success/failure of a parameter operation.

Table 27: Parameter_Update_Status Literals:

Name	Value	Description
Success	0	Parameter was successfully staged.
Id_Error	1	Parameter id was not valid.
Validation_Error	2	Parameter values were not successfully validated.
Length_Error	3	Parameter length was not correct.