Connector Delayer

Component Design Document

1 Description

This component is similar to the Connector Queuer component except that it delays (via sleep) the transmission of the sent data for a configurable (at initialization) amount of time prior. When configured to sleep for zero microseconds, this component behaves identically to the Connector Queuer. This component can be used for a variety of purposes, such as 1) serving as an alarm, transmitting data N us after receipt or 2) spacing out the transmission of data, sending out new data every N us. Like the Connector Queuer, this component serves as a multi-tasking safe synchronization point for multiple callers. When the T_Recv_Async connector is called, the data is queued. Based on the priority of this component, the data will be safely dequeued in the future, the component will then sleep for the configurable amount of microseconds, and then, finally, the T_Send connector will be called. The queue is implemented using the standard Adamant queue, and thus calls are serviced in FIFO order. The queue protection mechanism effectively makes all downstream connector calls of this component thread-safe.

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** - *active*
- **Number of Connectors** - 4
- **Number of Invokee Connectors** - 1
- **Number of Invoker Connectors** - 3
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - 2
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - *None*
- **Number of Parameters** - *None*
- **Number of Events** - 1
- **Number of Faults** - *None*
- **Number of Data Products** - *None*
- **Number of Data Dependencies** - *None*

- **Number of Packets** - *None*

3.2 Diagram

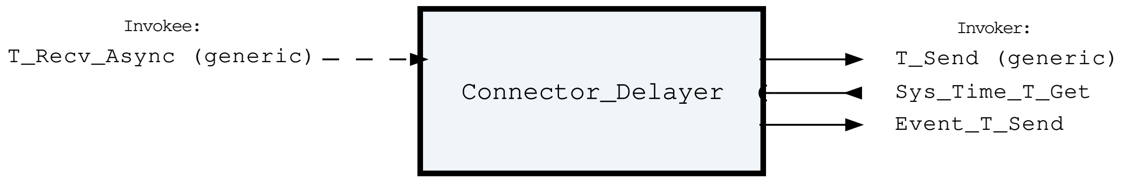

Figure 1: Connector Delayer component diagram.

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: Connector Delayer Invokee Connectors

Name	Kind	Type	Return_Type	Count
`T_Recv_Async`	`recv_async`	`T` (generic)	-	1

Connector Descriptions:
- **`T_Recv_Async`** - The generic invokee 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: Connector Delayer Asynchronous Connectors

Name	Type	Max Size (bytes)
`T_Recv_Async`	`T` (generic)	*Unconstrained*

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: Connector Delayer Invoker Connectors

Name	Kind	Type	Return_Type	Count
`T_Send`	`send`	`T` (generic)	-	1

Sys_Time_T_Get	get	-	Sys_Time.T	1
Event_T_Send	send	Event.T	-	1

Connector Descriptions:

- **`T_Send`** - The generic invoker connector. Calls originating from this connector are serviced from the component's queue and thus will be executed in FIFO order in a thread-safe, atomic manner.
- **`Sys_Time_T_Get`** - The system time is retrieved via this connector.
- **`Event_T_Send`** - Events are sent out of 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 Generic Component Instantiation

This is a generic component in that it can be instantiated to delay a connector of any type at compile time. This component contains generic formal types. These generic formal types must be instantiated with a valid actual type prior to component initialization. This is done by specifying types for the following generic formal parameters:

Table 4: Connector Delayer Generic Formal Types

Name	**Formal Type Definition**
T	`type T is private;`
Serialized_Length	`with function Serialized_Length (Src : in T; Num_Bytes_Serialized : out Natural) return Serializer_Types.Serialization_Status;`

Generic Formal Type Descriptions:

- **`T`** - The generic type of data passed in and out of the component.
- **`Serialized_Length`** - A method that returns the serialized length of an item of type T. This is useful for serializing variable length packed types onto the queue.

3.5.2 Component Instantiation

This component contains no instantiation parameters in its discriminant.

3.5.3 Component Base Initialization

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

Table 5: Connector Delayer 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.

3.5.4 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 6: Connector Delayer Set Id Bases Parameters

Name	**Type**
`Event_Id_Base`	`Event_Types.Event_Id_Base`

Parameter Descriptions:

- **`Event_Id_Base`** - The value at which the component's event identifiers begin.

3.5.5 Component Map Data Dependencies

This component contains no data dependencies.

3.5.6 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. Configure how long the component will delay prior to sending out any queued data. The `init` subprogram requires the following parameters:

Table 7: Connector Delayer Implementation Initialization Parameters

Name	**Type**	**Default Value**
`Delay_Us`	`Natural`	*None provided*

Parameter Descriptions:

- **`Delay_Us`** - The amount of time to delay prior to transmission in microseconds. The delay time begins right after the element is dequeued from the Adamant queue.

3.6 Commands

The Connector Delayer component has no commands.

3.7 Parameters

The Connector Delayer component has no parameters.

3.8 Events

Below is a list of the events for the Connector Delayer component.

Table 8: Connector Delayer Events

Local ID	Event Name	Parameter Type
0	`Dropped_Message`	-

Event Descriptions:

- **`Dropped_Message`** - The queue overflowed and the incoming data was dropped.

3.9 Data Products

The Connector Delayer component has no data products.

3.10 Data Dependencies

The Connector Delayer component has no data dependencies.

3.11 Packets

The Connector Delayer component has no packets.

3.12 Faults

The Connector Delayer component has no faults.

4 Unit Tests

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

4.1 *Connector_ Delayer_ Tests* Test Suite

This is a unit test suite for the Connector Delayer.

Test Descriptions:

- **`Test_Queued_Call`** - This unit test invokes the async connector and makes sure the arguments are passed through asynchronously, as expected.
- **`Test_Full_Queue`** - This unit test fills the queue and makes sure that dropped messages are 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..

Event.T:

Generic event packet for holding arbitrary events

Table 9: 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 10: 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

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 11: 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.

5.3 Enumerations

No enumerations found in component.