Memory Packetizer

Component Design Document

1 Description

This active component receives memory pointer information on an asynchronous queue. It then reads the data that these pointers reference into packets, producing multiple maximum sized packets, if necessary, to packetize the entire memory region. Note that it does the packetization process on its own task. The priority of this task can be tuned by the user. Usually, this component will be made low priority, so packetization can happen in the background while nothing more important is running.

2 Requirements

The requirements for the Memory Packetizer are specified below.

1. The component shall receive pointers to contiguous data regions in memory and create packets containing the data stored at this region.
2. The component shall receive packet identifiers with each pointer and emit packets with this identifier when sending the pointer data.
3. The component shall produce packets with increasing sequence numbers for each unique packet identifier it receives, up to a configurable maximum.
4. The component shall be able to meter the production of packets to not exceed a commandable maximum rate.

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** - 7
- **Number of Invokee Connectors** - 2
- **Number of Invoker Connectors** - 5
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 1
- **Number of Parameters** - *None*

- **Number of Events** - 4
- **Number of Faults** - *None*
- **Number of Data Products** - 1
- **Number of Data Dependencies** - *None*
- **Number of Packets** - *None*

3.2 Diagram

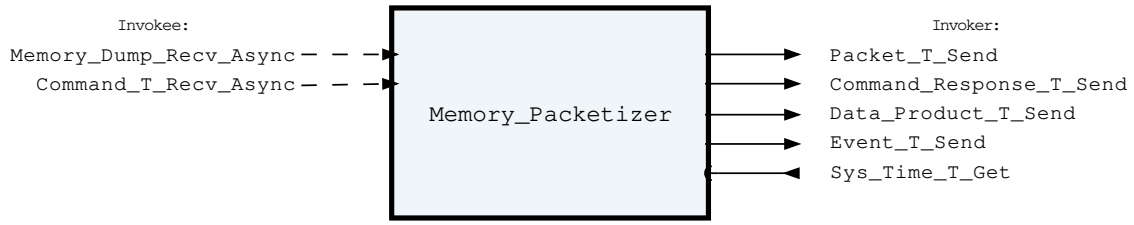

Figure 1: Memory Packetizer component diagram.

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: Memory Packetizer Invokee Connectors

Name	Kind	Type	Return_Type	Count
Memory_Dump_Recv_Async	recv_async	Memory_Packetizer_Types.Memory_Dump	-	1
Command_T_Recv_Async	recv_async	Command.T	-	1

Connector Descriptions:

- **Memory_Dump_Recv_Async** - A memory dump pointer and id queued up for packetization on this connector.
- **Command_T_Recv_Async** - This is the command receive 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: Memory Packetizer Asynchronous Connectors

Name	Type	Max Size (bytes)
Memory_Dump_Recv_Async	Memory_Packetizer_Types.Memory_Dump	*Platform Dependent*
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: Memory Packetizer Invoker Connectors

Name	Kind	Type	Return_Type	Count
Packet_T_Send	send	Packet.T	-	1
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:

- **Packet_T_Send** - Send a packet of data.
- **Command_Response_T_Send** - This connector is used to register and respond to the component's commands.
- **Data_Product_T_Send** - Data products are sent out of this connector.
- **Event_T_Send** - Events are sent out of this connector.
- **Sys_Time_T_Get** - The system time is retrieved via this connector.

3.4 Initialization

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

3.4.1 Component Instantiation

This component contains no instantiation parameters in its discriminant.

3.4.2 Component Base Initialization

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

Table 4: Memory Packetizer 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.

3.4.3 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: Memory Packetizer 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.

3.4.4 Component Map Data Dependencies

This component contains no data dependencies.

3.4.5 Component Implementation Initialization

The calling of this implementation class initialization procedure is mandatory. This initialization function is used to set a threshold for the maximum number of packets that the component will produce in a single time period. A time period is measured in an integer number of seconds. The component also needs to keep track of the sequence counts for each packet ID that it receives. To do this, it needs to allocate internal memory to keep track of the last sequence count for each packet. A maximum number of unique packet ids is provided in this function to allocate the necessary memory to keep track of this information. The `init` subprogram requires the following parameters:

Table 6: Memory Packetizer Implementation Initialization Parameters

Name	Type	Default Value
`Max_Packets_Per_Time_Period`	`Natural`	*None provided*
`Time_Period_In_Seconds`	`Positive`	`1`
`Max_Packet_Ids`	`Positive`	`10`

Parameter Descriptions:

- **`Max_Packets_Per_Time_Period`** - The maximum number of packets that this component will produce in a single second. The component will stop producing packets if the threshold is met, until the end of a second period has elapsed.
- **`Time_Period_In_Seconds`** - The time period in seconds over which to measure the number of packets produced.
- **`Max_Packet_Ids`** - The maximum number of unique packet ids that this component is expected to receive during operations. This value is used to allocate a small amount of memory at initialization to keep track of the sequence count for each produced packet. If this memory becomes fully used, any new unique packet ids received will trigger an event and will be emitted with a sequence count of zero. This misconfiguration should easily be detectable during test.

3.5 Commands

These are the commands for the memory packetizer component.

Table 7: Memory Packetizer Commands

Local ID	Command Name	Argument Type
0	Set_Max_Packet_Rate	Packets_Per_Period.T

Command Descriptions:

- **Set_Max_Packet_Rate** - Set a new value for the Max_Packets_Per_Time_Period and the Time_Period_In_Seconds to control the output rate of the emitted packets.

3.6 Events

Below is a list of the events for the Memory Packetizer component.

Table 8: Memory Packetizer Events

Local ID	Event Name	Parameter Type
0	Max_Packet_Id_Exceeded	Packet_Id.T
1	Memory_Dump_Request_Dropped	Packet_Id.T
2	Max_Packet_Rate_Set	Packets_Per_Period.T
3	Invalid_Command_Received	Invalid_Command_Info.T

Event Descriptions:

- **Max_Packet_Id_Exceeded** - The maximum number of packet ids that the component can keep track of sequence numbers for has been exceeded. Packets of this id will be emitted with a sequence number of 0.
- **Memory_Dump_Request_Dropped** - The queue for memory dump requests overflowed and a request to dump memory with the given packet id was dropped.
- **Max_Packet_Rate_Set** - A new maximum rate has been set for the packetizer.
- **Invalid_Command_Received** - A command was received with invalid parameters.

4 Unit Tests

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

4.1 *Memory_Packetizer_Tests* Test Suite

This is a unit test suite for the Memory Packetizer component.

Test Descriptions:

- **Test_Nominal_Packetization** - This unit test tests the packetizer's normal behavior and makes sure it packetizes memory properly, metering out the packets according to its rate.
- **Test_Set_Max_Packet_Rate** - This unit test tests the Set_Max_Packet_Rate command, and ensures that the rate changes appropriately.
- **Test_Invalid_Command** - This unit test tests a bad Set_Max_Packet_Rate command, and ensures that an appropriate event is thrown.

- **Test_Memory_Dump_Dropped** - This unit test tests the behavior when the component's queue becomes full and must drop a memory dump request.
- **Test_Max_Packet_Id_Exceeded** - This unit test tests what happens when more packet ids are sent to the component than the component can track sequence counts for.

5 Appendix

5.1 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 9: 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 10: 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 11: 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 12: 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 13: 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 14: 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 15: 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 16: 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.

Packet.T:

Generic packet for holding arbitrary data

Table 17: Packet Packed Record : 10080 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Packet_ Header.T	-	112	0	111	-
Buffer	Packet_ Types.Packet_ Buffer_Type	-	9968	112	10079	Header. Buffer_Length

Field Descriptions:

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

Packet_Header.T:

Generic packet header for holding arbitrary data

Table 18: Packet_Header Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63
Id	Packet_Types. Packet_Id	0 to 65535	16	64	79
Sequence_Count	Packet_Types. Sequence_Count_Mod_ Type	0 to 16383	16	80	95

Buffer_Length	Packet_Types. Packet_Buffer_ Length_Type	0 to 1246	16	96	111

Field Descriptions:

- **Time** - The timestamp for the packet item.
- **Id** - The packet identifier
- **Sequence_Count** - Packet Sequence Count
- **Buffer_Length** - The number of bytes used in the packet buffer

Packet_Id.T:

A packed record which holds a packet identifier.

Table 19: Packet_Id Packed Record : 16 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Id	Packet_Types. Packet_Id	0 to 65535	16	0	15

Field Descriptions:

- **Id** - The packet identifier

Packets_Per_Period.T:

A record which holds information on how to set a maximum packet rate.

Table 20: Packets_Per_Period Packed Record : 64 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Max_Packets	Natural	0 to 2147483647	32	0	31
Period	Positive	1 to 2147483647	32	32	63

Field Descriptions:

- **Max_Packets** - The maximum number of packets to send out in one time period.
- **Period** - The length of a time period, defined in seconds.

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 21: 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.

5.2 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 22: 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.