Memory Stuffer

Component Design Document

1 Description

The memory stuffer component is an active component that can stuff (write to) memory regions. It reports an error if an action is requested on a memory region outside of the address space that it is configured with during initialization. The component can manage both protected memory regions (which require an arm command prior to stuffing) and unprotected regions (which require no arm prior to stuffing). In addition, the component has a connector to accept a memory region copy request, which will stuff memory with data from another system address. The memory region copy and release conntectors may be disconnected if this feature is not needed.

2 Requirements

The requirements for the Memory Stuffer component are specified below.

1. The component shall write to a memory region on command.
2. The component shall reject commands to write to memory in off-limit regions.
3. The component shall reject commands to write to protected memory regions if not armed.
4. The component shall enter an armed state (capable of writing to protected memory regions) if an arm command is received.
5. The component shall exit the armed state upon the receipt of any subsequent command (unless it is another arm command).
6. The component shall exit the armed state after a timeout.

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** - 8
- **Number of Invokee Connectors** - 3
- **Number of Invoker Connectors** - 5
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - 2

- **Number of Parameters** - *None*
- **Number of Events** - 11
- **Number of Faults** - *None*
- **Number of Data Products** - 2
- **Number of Data Dependencies** - *None*
- **Number of Packets** - *None*

3.2 Diagram

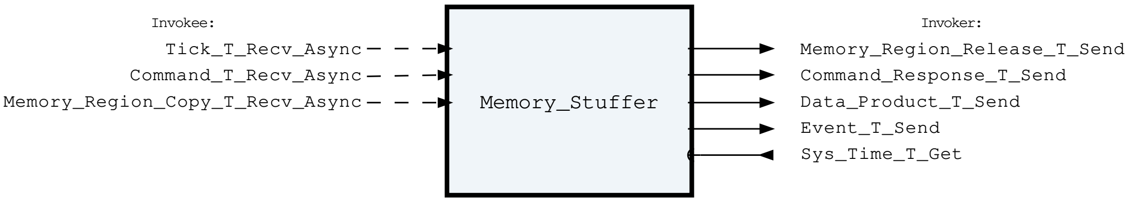

Figure 1: Memory Stuffer component diagram.

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 Stuffer 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
`Memory_Region_Copy_T_Recv_Async`	`recv_async`	`Memory_Region_Copy.T`	-	1

Connector Descriptions:

- **`Tick_T_Recv_Async`** - This tick is used to keep track of the armed state timeout and send the data product relating the current timeout value.
- **`Command_T_Recv_Async`** - This is the command recieve connector.
- **`Memory_Region_Copy_T_Recv_Async`** - A memory region is received on this connector and stuffed to a different memory region, a memory copy.

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 Stuffer Asynchronous Connectors

Name	Type	Max Size (bytes)
`Tick_T_Recv_Async`	`Tick.T`	17
`Command_T_Recv_Async`	`Command.T`	106
`Memory_Region_Copy_T_Recv_Async`	`Memory_Region_Copy.T`	25

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 Stuffer Invoker Connectors

Name	Kind	Type	Return_Type	Count
`Memory_Region_Release_T_Send`	send	`Memory_Region_Release.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:

- **`Memory_Region_Release_T_Send`** - This connector is used to release the received memory region after a copy has occured.
- **`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 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: Memory Stuffer 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: Memory Stuffer Set Id Bases Parameters

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

Parameter Descriptions:

- **`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.
- **`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. This component requires a list of memory regions which it can write to. These regions can either be protected (requiring and arm command prior to execution) or unprotected, as specified by the second parameter. The `init` subprogram requires the following parameters:

Table 6: Memory Stuffer Implementation Initialization Parameters

Name	Type	Default Value
`Memory_Regions`	`Memory_Manager_Types.Memory_Region_Array_Access`	*None provided*
`Memory_Region_Protection_List`	`Memory_Manager_Types.Memory_Protection_Array_Access`	`null`

Parameter Descriptions:

- **Memory_Regions** - An access to a list of memory regions.
- **Memory_Region_Protection_List** - An access to a list of the protected/unprotected state of each memory region. The index in this array corresponds to the index of the memory region affected in the previous parameter. If the array is null, then it is assumed that all memory regions are unprotected.

3.6 Commands

These are the commands for the Memory Stuffer component.

Table 7: Memory Stuffer Commands

Local ID	Command Name	Argument Type
0	Write_Memory	Memory_Region_Write.T
1	Arm_Protected_Write	Packed_Arm_Timeout.T

Command Descriptions:

- **Write_Memory** - Write bytes to a region in memory.
- **Arm_Protected_Write** - An arm command which enables the next write command to a protected memory to be accepted. The armed state of the component will expire on the next command to this component no matter what it is or after the configurable timeout.

3.7 Parameters

The Memory Stuffer component has no parameters.

3.8 Events

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

Table 8: Memory Stuffer Events

Local ID	Event Name	Parameter Type
0	Invalid_Memory_Region	Memory_Region.T
1	Invalid_Copy_Destination	Memory_Region.T
2	Protected_Write_Enabled	Packed_Arm_Timeout.T
3	Protected_Write_Disabled	-
4	Writing_Memory	Memory_Region.T
5	Memory_Written	Memory_Region.T
6	Copying_Memory	Memory_Region_Copy.T
7	Memory_Copied	Memory_Region_Copy.T
8	Protected_Write_Denied	Memory_Region.T
9	Invalid_Command_Received	Invalid_Command_Info.T
10	Protected_Write_Disabled_Timeout	-

Event Descriptions:

- **Invalid_Memory_Region** - A command was sent to access a memory region with an invalid address and/or length.
- **Invalid_Copy_Destination** - A copy request was received with an invalid destination address and length.

- **`Protected_Write_Enabled`** - An arm command was received and the protected write state is enabled.
- **`Protected_Write_Disabled`** - The protected write state was disabled either by timeout or receiving a subsequent command.
- **`Writing_Memory`** - The component is currently writing the memory location for the following region.
- **`Memory_Written`** - The component has finished writing the memory location for the following region.
- **`Copying_Memory`** - The component is currently copying memory from one address to another.
- **`Memory_Copied`** - The component has finished copying memory from one address to another.
- **`Protected_Write_Denied`** - A command was received to write to a protected region, but the component was not armed so the command is being rejected.
- **`Invalid_Command_Received`** - A command was received with invalid parameters.
- **`Protected_Write_Disabled_Timeout`** - The component armed state timed out and is now unarmed.

3.9 Data Products

Data products for the Memory Stuffer component.

Table 9: Memory Stuffer Data Products

Local ID	Data Product Name	Type
`0x0000 (0)`	`Armed_State`	`Packed_Arm_State.T`
`0x0001 (1)`	`Armed_State_Timeout`	`Packed_Arm_Timeout.T`

Data Product Descriptions:

- **`Armed_State`** - The current armed/unarmed state of the component.
- **`Armed_State_Timeout`** - The time remaining (in ticks) until the armed state expires.

3.10 Packets

The Memory Stuffer component has no packets.

4 Unit Tests

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

4.1 *Memory_ Stuffer_ Tests* Test Suite

This is a unit test suite for the Memory Stuffer component

Test Descriptions:

- **`Test_Invalid_Initialization`** - This unit test makes sure that an invalid initialization results in a runtime assertion.
- **`Test_Unprotected_Stuffing`** - This unit test excersizes stuffing a region that is unprotected.

- **Test_Protected_Stuffing** - This unit test excersizes stuffing a region that is protected.
- **Test_Arm_Unarm** - This unit test excersizes all of the ways that the arm command can be invalidated prior to a write (except via timeout).
- **Test_Arm_Timeout** - This unit test excersizes the unarming of the arm command via timeout.
- **Test_Invalid_Address** - This unit test excersizes writing to an invalid region of memory
- **Test_Invalid_Command** - This unit test makes sure an invalid command is reported and ignored.
- **Test_Memory_Region_Copy** - This unit test excersizes the memory region copy and release connectors.
- **Test_Memory_Region_Copy_Invalid_Address** - This unit test excersizes the memory region copy and release connectors with an invalid destination address.

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 : 808 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	-	768	40	807	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 96	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 unkown 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.

Memory_Region.T:

A memory region described by a system address and length (in bytes).

Table 18: Memory_Region Packed Record : 96 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Address	System.Address	-	64	0	63
Length	Natural	0 to 2147483647	32	64	95

Field Descriptions:

- **Address** - The starting address of the memory region.
- **Length** - The number of bytes at the given address to associate with this memory region.

Memory_Region_Copy.T:

A memory region copy record.

Table 19: Memory_Region_Copy Packed Record : 160 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Source_Region	Memory_Region.T	-	96	0	95
Destination_Address	System.Address	-	64	96	159

Field Descriptions:

- **Source_Region** - The source address and length to copy from.
- **Destination_Address** - The destination address of the memory region copy.

Memory_Region_Release.T:

A memory region copy release record. This is returned from a component who has completed a copy operation.

Table 20: Memory_Region_Release Packed Record : 104 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Region	Memory_Region.T	-	96	0	95
Status	Memory_Enums.Memory_Copy_Status.E	0 => Success 1 => Failure	8	96	103

Field Descriptions:

- **Region** - The address and length to release.
- **Status** - The return status from the operation.

Memory_Region_Write.T:

A memory region CRC report.

Preamble (inline Ada definitions):

```ada
use Command_Types;
subtype Byte_Buffer_Index_Type is Command_Arg_Buffer_Index_Type range Command_Arg_Buffer_Index_Type'First .. Command_Arg_Buffer_Index_Type'Last - 8 - 2;
subtype Byte_Buffer_Type is Basic_Types.Byte_Array (Byte_Buffer_Index_Type);
subtype Region_Length_Type is Natural range 0 .. Byte_Buffer_Type'Length;
```

Table 21: Memory_Region_Write Packed Record : 768 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Address	System.Address	-	64	0	63	-
Length	Region_Length_Type	0 to 86	16	64	79	-

Data	Byte_Buffer_Type	-	688	80	767	Length

Field Descriptions:

- **Address** - The starting address of the memory region.
- **Length** - The number of bytes at the given address to associate with this memory region.
- **Data** - The bytes to write to the memory region

Packed_Arm_State.T:

Holds the armed state.

Table 22: Packed_Arm_State Packed Record : 8 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
State	Command_Protector_Enums.Armed_State.E	0 => Unarmed 1 => Armed	8	0	7

Field Descriptions:

- **State** - The armed/unarmed status.

Packed_Arm_Timeout.T:

Holds the armed state timout.

Preamble (inline Ada definitions):

```
type Arm_Timeout_Type is new Natural range 0 .. 255;
```

Table 23: Packed_Arm_Timeout Packed Record : 8 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Timeout	Arm_Timeout_Type	0 to 255	8	0	7

Field Descriptions:

- **Timeout** - The timeout value (in ticks).

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 24: 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 25: 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 26: 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.

Command_Protector_Enums.Armed_State.E:

This type enumerates the armed state for the component.

Table 27: Armed_State Literals:

Name	Value	Description
Unarmed	0	The component is unarmed. Any protected commands received will be rejected.
Armed	1	The component is armed. If the next command received is a protected command, it will be forwarded.

Memory_Enums.Memory_Copy_Status.E:

This status enumerations provides information on the success/failure of a memory copy.

Table 28: Memory_Copy_Status Literals:

Name	Value	Description
Success	0	The copy command succeeded.
Failure	1	The copy command failed.