Memory Dumper

Component Design Document

1 Description

The memory dumper component is an active component that can dump memory regions or report the CRC of memory regions by command. 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.

2 Requirements

The requirements for the Memory Dumper component are specified below.

1. The component shall dump a memory region on command.
2. The component shall crc a memory region on command.
3. The component shall reject commands to dump or crc memory in off-limit regions.

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** - 6
- **Number of Invokee Connectors** - 1
- **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** - 5
- **Number of Faults** - *None*
- **Number of Data Products** - 1
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

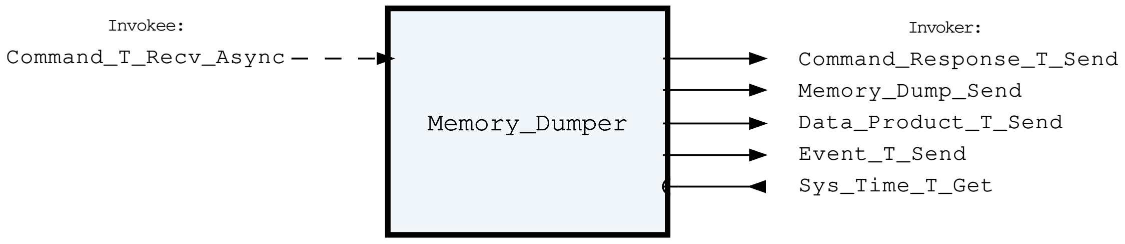

Figure 1: Memory Dumper component diagram.

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 Dumper Invokee Connectors

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

Connector Descriptions:

- **Command_T_Recv_Async** - This is the command recieve 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 Dumper Asynchronous Connectors

Name	Type	Max Size (bytes)
Command_T_Recv_Async	Command.T	106

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 Dumper Invoker Connectors

Name	Kind	Type	Return_Type	Count

Command_Response_T_Send	send	Command_Response.T	-	1
Memory_Dump_Send	send	Memory_Packetizer_Types.Memory_Dump	-	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 and respond to the component's commands.
- **Memory_Dump_Send** - The memory dump connector.
- **Data_Product_T_Send** - The data product invoker 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 Dumper 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 Dumper Set Id Bases Parameters

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

Packet_Id_Base	Packet_Types.Packet_Id_Base

Parameter Descriptions:

- **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.
- **Command_Id_Base** - The value at which the component's command identifiers begin.
- **Packet_Id_Base** - The value at which the component's unresolved packet 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 component requires a list of memory regions which it can dump and CRC. The `init` subprogram requires the following parameters:

Table 6: Memory Dumper Implementation Initialization Parameters

Name	Type	Default Value
Memory_Regions	Memory_Manager_Types. Memory_Region_Array_Access	*None provided*

Parameter Descriptions:

- **Memory_Regions** - An access to a list of memory regions.

3.5 Commands

These are the commands for the Memory Dumper component.

Table 7: Memory Dumper Commands

Local ID	Command Name	Argument Type
0	Dump_Memory	Memory_Region_Positive.T
1	Crc_Memory	Memory_Region_Positive.T

Command Descriptions:

- **Dump_Memory** - Dump a region of memory starting at a given address and of a given length.
- **Crc_Memory** - Perform a CRC on a region of memory starting at a given address and of a given length. The CRC will be reported via event and data product, if those connectors are connected.

3.6 Events

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

Table 8: Memory Dumper Events

Local ID	Event Name	Parameter Type
0	`Invalid_Memory_Region`	`Memory_Region_Positive.T`
1	`Dumping_Memory`	`Memory_Region_Positive.T`
2	`Crcing_Memory`	`Memory_Region_Positive.T`
3	`Memory_Crc`	`Memory_Region_Crc.T`
4	`Invalid_Command_Received`	`Invalid_Command_Info.T`

Event Descriptions:

- **`Invalid_Memory_Region`** - A command was sent to access a memory region with an invalid address and/or length.
- **`Dumping_Memory`** - The component is currently dumping the memory location for the following region.
- **`Crcing_Memory`** - The component is currently CRCing the memory location for the following region.
- **`Memory_Crc`** - The memory region CRC has been calculated.
- **`Invalid_Command_Received`** - A command was received with invalid parameters.

3.7 Data Products

Data products for the memory dumper component.

Table 9: Memory Dumper Data Products

Local ID	Data Product Name	Type
`0x0000 (0)`	`Crc_Report`	`Memory_Region_Crc.T`

Data Product Descriptions:

- **`Crc_Report`** - The last computed CRC by the memory dumper component.

3.8 Packets

Packets for the memory dumper.

Table 10: Memory Dumper Packets

Local ID	Packet Name	Type
`0x0000 (0)`	`Memory_Dump_Packet`	*Undefined*

Packet Descriptions:

- **`Memory_Dump_Packet`** - This packet contains memory.

4 Unit Tests

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

4.1 *Memory_ Dumper_ Tests* Test Suite

This is a unit test suite for the Memory Dumper component

Test Descriptions:

- **`Test_Nominal_Dumping`** - This unit test excersizes dumping a valid region of memory managed by the component
- **`Test_Memory_Crc`** - This unit test excersizes CRCing from a valid region of memory
- **`Test_Invalid_Address`** - This unit test excersizes dumping and CRCing from an invalid region of memory
- **`Test_Invalid_Command`** - This unit test makes sure an invalid command is reported and ignored.

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 11: 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 12: 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 13: 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 14: 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 15: 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 16: 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 17: 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 18: 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 19: 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_Crc.T:

A memory region CRC report.

Table 20: Memory_Region_Crc Packed Record : 112 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
`Region`	`Memory_Region.T`	-	96	0	95
`Crc`	`Crc_16.Crc_16_Type`	-	16	96	111

Field Descriptions:

- **`Region`** - The memory region that was CRCed
- **`Crc`** - The computed CRC

Memory_Region_Positive.T:

A memory region described by a system address and length (in bytes). The length must be positive.

Table 21: Memory_Region_Positive Packed Record : 96 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
`Address`	`System.Address`	-	64	0	63
`Length`	`Positive`	1 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.

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.

5.2 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 23: 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.