Last Chance Manager

Component Design Document

1 Description

The purpose of this component is to manage a region of non-volatile memory where the last chance handler saves exception information, should one be thrown. This component provides commands to dump this region of memory and reset the contents of the memory to all zeros. The component provides a data product that reports the first address of the stack trace, which can be used as confirmation that the LCH got called (if the value is nonzero).

2 Requirements

The requirements for the Last Chance Handler component.

1. The component shall provide a nonvolatile last chance handler memory region, where exception occurrence data can be stored by the last chance handler.
2. The component shall provide a command to dump the last chance handler memory region.
3. The component shall provide a command to clear the last chance handler memory region.

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

- **Number of Packets** - 1

3.2 Diagram

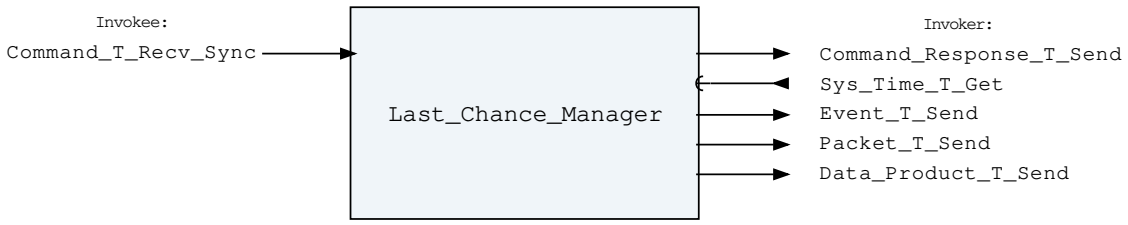

Figure 1: Last Chance Manager component diagram.

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: Last Chance Manager Invokee Connectors

Name	Kind	Type	Return_Type	Count
Command_T_Recv_Sync	recv_sync	Command.T	-	1

Connector Descriptions:

- **Command_T_Recv_Sync** - The command receive connector

3.3.2 Invoker Connectors

The following is a list of the component's *invoker* connectors:

Table 2: Last Chance Manager Invoker Connectors

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

Connector Descriptions:

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

- **Data_Product_T_Send** - Data products 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 Component Instantiation

This component contains no instantiation parameters in its discriminant.

3.5.2 Component Base Initialization

This component contains no base class initialization, meaning there is no `init_Base` subprogram for this component.

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 3: Last Chance Manager 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`
`Packet_Id_Base`	`Packet_Types.Packet_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.
- **Packet_Id_Base** - The value at which the component's unresolved packet 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 the memory region which the last chance handler data will be stored. The `init` subprogram requires the following parameters:

Table 4: Last Chance Manager Implementation Initialization Parameters

Name	Type	Default Value

Exception_Data	Packed_Exception_Occurrence.T_Access	*None provided*
Dump_Exception_Data_At_Startup	Boolean	*None provided*

Parameter Descriptions:

- **Exception_Data** - The copy of the exception data that is updated by the last chance handler, presumably in a nonvolatile memory region.
- **Dump_Exception_Data_At_Startup** - If True, then the exception data will be dumped in packet at startup.

3.6 Commands

Commands for the Last Chance Manager component.

Table 5: Last Chance Manager Commands

Local ID	Command Name	Argument Type
0	Dump_Last_Chance_Handler_Region	-
1	Clear_Last_Chance_Handler_Region	-

Command Descriptions:

- **Dump_Last_Chance_Handler_Region** - Dump the last chance handler memory region into a packet for downlink.
- **Clear_Last_Chance_Handler_Region** - Clear the last chance handler memory region by writing all zeros to it.

3.7 Parameters

The Last Chance Manager component has no parameters.

3.8 Events

Below is a list of the events for the Last Chance Manager component.

Table 6: Last Chance Manager Events

Local ID	Event Name	Parameter Type
0	Last_Chance_Handler_Called	Packed_Stack_Trace_Info.T
1	Dumped_Last_Chance_Handler_Region	-
2	Cleared_Last_Chance_Handler_Region	-
3	Invalid_Command_Received	Invalid_Command_Info.T

Event Descriptions:

- **Last_Chance_Handler_Called** - The component detected that the LCH was called by looking at the data in nonvolatile memory. The lowest level address of the stack trace is reported.
- **Dumped_Last_Chance_Handler_Region** - The component dumped the last chance handler memory region into a packet for downlink.

- **Cleared_Last_Chance_Handler_Region** - The component cleared the last chance handler memory region by writing all zeros to it.
- **Invalid_Command_Received** - A command was received with invalid parameters.

3.9 Data Products

Data products for the Last Chance Manager component.

Table 7: Last Chance Manager Data Products

Local ID	Data Product Name	Type
0x0000 (0)	Lch_Stack_Trace_Info	Packed_Stack_Trace_Info.T

Data Product Descriptions:
- **Lch_Stack_Trace_Info** - Information on the current stack trace stored in the last chance handler memory store.

3.10 Packets

The second packet listed here is not actually produced by the Last Chance Manager component, but instead should be produced by the implementation of the Last_Chance_Handler. This packet definition exists to ensure that the packet gets reflected in the documentation and ground system definitions.

Table 8: Last Chance Manager Packets

Local ID	Packet Name	Type
0x0000 (0)	Lch_Memory_Region_Dump	Packed_Exception_Occurrence.T

Packet Descriptions:
- **Lch_Memory_Region_Dump** - This packet contains a dump of the LCH nonvolatile memory region where exception information is thrown.

4 Unit Tests

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

4.1 *Last_Chance_Manager_Tests* Test Suite

This is a unit test suite for the Last Chance Manager component.

Test Descriptions:
- **Test_Region_Dump** - This unit test makes sure the region dump command executes successfully.
- **Test_Region_Clear** - This unit test makes sure the region clear command executes successfully.
- **Test_Invalid_Command** - This unit test makes sure an invalid command is rejected.

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 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.

Packed_Address.T:

A packed system address.

Table 17: Packed_Address Packed Record : 64 bits

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

Field Descriptions:

- **Address** - The starting address of the memory region.

Packed_Exception_Occurrence.T:

Packed record which holds information from an Ada Exception Occurrence type. This is the type passed into the Last Chance Handler when running a full runtime. *Preamble (inline Ada definitions):*

```
type Exception_Name_Buffer is new Basic_Types.Byte_Array (0 .. 99)
  with Size => 100 * 8,
       Object_Size => 100 * 8;
type Exception_Message_Buffer is new Basic_Types.Byte_Array (0 .. 299)
  with Size => 300 * 8,
       Object_Size => 300 * 8;
```

Table 18: Packed_Exception_Occurrence Packed Record : 7712 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Exception_Name	Exception_Name_Buffer	-	800	0	799

Exception_ Message	Exception_ Message_Buffer	-	2400	800	3199
Stack_Trace_ Depth	Interfaces. Unsigned_32	0 to 4294967295	32	3200	3231
Stack_Trace	Stack_Trace_ Addresses.T	-	4480	3232	7711

Field Descriptions:

- **Exception_Name** - The exception name.
- **Exception_Message** - The exception message.
- **Stack_Trace_Depth** - The depth of the reported stack trace.
- **Stack_Trace** - The stack trace addresses.

Packed_Stack_Trace_Info.T:

Packed record which holds summary information about a stored stack trace.

Table 19: Packed_Stack_Trace_Info Packed Record : 96 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Stack_Trace_Depth	Interfaces. Unsigned_32	0 to 4294967295	32	0	31
Stack_Trace_ Bottom_Address	Packed_Address.T	-	64	32	95

Field Descriptions:

- **Stack_Trace_Depth** - The depth of the reported stack trace.
- **Stack_Trace_Bottom_Address** - The bottom stack trace address.

Packet.T:

Generic packet for holding arbitrary data

Table 20: 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 21: 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

Stack_Trace_Addresses.T:

An array of packed addresses in big endian. This is sized to easily fit a normal stack trace.

Table 22: Stack_Trace_Addresses Packed Array : 4480 bits

Type	Range	Element Size (Bits)	Length	Total Size (Bits)
Packed_Address.T	-	64	70	4480

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.

5.3 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 24: 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.