Zero Divider

Component Design Document

1 Description

The purpose of this component is to provide a safe, commandable way to cause the Ada Last Chance Handler to be called. To accomplish this, this component provides a Divide_By_Zero command which divides an integer by zero, which causes an Ada exception to be thrown, which is purposely not handled. The Divide_By_Zero command must be passed a magic number as an argument. If the magic number does not match the number that this component is instantiated with at initialization, then the Divide_By_Zero is not executed. This feature prevents inadvertant execution of this command. This component also supplies the packet definition for the assembly for a Last Chance Handler (LCH) packet that is created by the last chance handler itself (which is not usually implemented as an Adamant component). This provides the ground system the LCH packet definition so it can be parsed and stored. The component does not contain a Packet.T send connector, so will not send out this packet itself. You Last Chance Handler should produce a packet with this packet definition.

2 Requirements

The requirements for the Zero Divider component.

1. The component shall provide a command, that when executed, causes an unhandled exception to be thrown.
2. The component shall provide a protection mechanism that protects the command from being executed accidentally.

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** - 4
- **Number of Invokee Connectors** - 1
- **Number of Invoker Connectors** - 3
- **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** - 3
- **Number of Faults** - *None*
- **Number of Data Products** - *None*
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

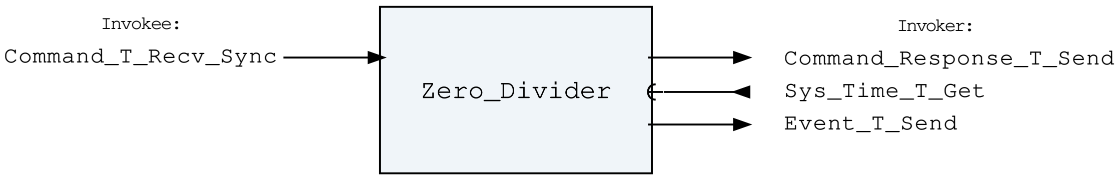

Figure 1: Zero Divider component diagram.

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: Zero Divider 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: Zero Divider 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

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.

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: Zero Divider Set Id Bases Parameters

Name	**Type**
`Command_Id_Base`	`Command_Types.Command_Id_Base`
`Packet_Id_Base`	`Packet_Types.Packet_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.
- **`Packet_Id_Base`** - The value at which the component's unresolved packet 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. The magic number is provided at instantiation. The `init` subprogram requires the following parameters:

Table 4: Zero Divider Implementation Initialization Parameters

Name	**Type**	**Default Value**
`Magic_Number`	`Magic_Number_Type`	*None provided*
`Sleep_Before_Divide_Ms`	`Natural`	`1000`

Parameter Descriptions:

- **Magic_Number** - Pick a number that must be provided with the Divide_By_Zero command for it to be executed. If any other number is provided, the command is failed and no divide by zero instruction is executed. Note - The values of 0 and 1 are not accepted as magic numbers.
- **Sleep_Before_Divide_Ms** - The number of milliseconds to sleep after receiving the command but before performing the divide by zero. This allows time for any events to be written by the component, if desired.

3.6 Commands

Commands for the Zero Divider component.

Table 5: Zero Divider Commands

Local ID	Command Name	Argument Type
0	Divide_By_Zero	Packed_U32.T

Command Descriptions:

- **Divide_By_Zero** - You must provide the correct magic number as argument to this command for it to be executed.

3.7 Parameters

The Zero Divider component has no parameters.

3.8 Events

Below is a list of the events for the Zero Divider component.

Table 6: Zero Divider Events

Local ID	Event Name	Parameter Type
0	Dividing_By_Zero	Packed_Natural.T
1	Invalid_Magic_Number	Packed_U32.T
2	Invalid_Command_Received	Invalid_Command_Info.T

Event Descriptions:

- **Dividing_By_Zero** - A divide by zero command was received, and the magic number was correct. The division will occur in N milliseconds, where N is provided as the event parameter.
- **Invalid_Magic_Number** - A divide by zero command was received, but the magic number was incorrect. The division will not occur.
- **Invalid_Command_Received** - A command was received with invalid parameters.

3.9 Data Products

The Zero Divider component has no data products.

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 7: Zero Divider Packets

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

Packet Descriptions:

- **`Last_Chance_Handler_Packet`** - This packet contains information regarding an exception occurrence that triggers the Last_Chance_Handler to get invoked. This packet is not produced directly by this component, and should be produced by the last chance handler implementation. This packet definition exists to ensure that the packet gets reflected in the documentation and ground system definitions.

4 Unit Tests

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

4.1 *Zero_ Divider_ Tests* Test Suite

This is a unit test suite for the Zero Divider component.

Test Descriptions:

- **`Test_Bad_Magic_Number`** - This unit test makes sure the Divide_By_Zero command does not execute if the correct magic number is not provided.
- **`Test_Divide_By_Zero`** - This unit test makes sure a constraint error is thrown when the divide by zero command executes.
- **`Test_Invalid_Command`** - This unit test makes sure an invalid command is rejected.

5 Appendix

5.1 Preamble

This component contains the following preamble code. This is inline Ada code included in the component model that is usually used to define types or instantiate generic packages used by the component. Preamble code is inserted as the top line of the component base package specification.

```ada
subtype Magic_Number_Type is Interfaces.Unsigned_32 range 2 ..
↪  Interfaces.Unsigned_32'Last;
```

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 8: 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 to that contains the command arguments

Command_Header.T:

Generic command header for holding arbitrary commands

Table 9: 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 10: 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.

Event.T:

Generic event packet for holding arbitrary events

Table 11: 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 12: 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 13: 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.

Packed_Address.T:

A packed system address.

Table 14: 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):

```ada
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 15: Packed_Exception_Occurrence Packed Record : 9632 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	-	6400	3232	9631

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_Natural.T:

Single component record for holding packed Natural value.

Table 16: Packed_Natural Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Value	Natural	0 to 2147483647	32	0	31

Field Descriptions:

- **Value** - The 32-bit Natural Integer.

Packed_U32.T:

Single component record for holding packed unsigned 32-bit value.

Table 17: Packed_U32 Packed Record : 32 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Value	Interfaces. Unsigned_32	0 to 4294967295	32	0	31

Field Descriptions:

- **Value** - The 32-bit unsigned integer.

Stack_Trace_Addresses.T:

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

Table 18: Stack_Trace_Addresses Packed Array : 6400 bits

Type	Range	Element Size (Bits)	Length	Total Size (Bits)
Packed_Address.T	-	64	100	6400

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 19: 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 enumerations provides information on the success/failure of a command through the command response connector.

Table 20: 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.