Ccsds Command Depacketizer

Component Design Document

1 Description

This component receives CCSDS packets, validates the data within them, and converts them into Adamant commands. Note that the only internal state that this component contains is a packet accept and packet reject count. The component assumes that only a single task is attached to its CCSDS Space Packet invokee connector, and thus these counters are unprotected. If more than one task is attached to the input, a race condition arises around the counters, which may need to become protected.

2 Requirements

The requirements for the CCSDS Command Depacketizer component are specified below.

1. The component shall convert LASP CCSDS command packets to the Adamant command type.
2. The component shall reject CCSDS packets with an invalid length.
3. The component shall reject CCSDS packets that do not contain a secondary header.
4. The component shall reject CCSDS packets that are not marked as telecommand packets in the secondary header.
5. The component shall reject CCSDS packets that contain an invalid 8-bit command checksum in the secondary header.
6. The component shall calculate the actual command packet length by subtracting the number stored in the secondary header function code from the CCSDS header length.

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** - 8
- **Number of Invokee Connectors** - 2
- **Number of Invoker Connectors** - 6
- **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** - 7
- **Number of Faults** - *None*
- **Number of Data Products** - 2
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

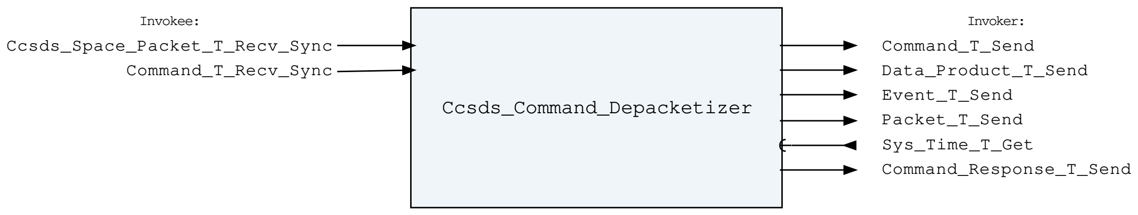

Figure 1: Ccsds Command Depacketizer component diagram.

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: Ccsds Command Depacketizer Invokee Connectors

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

Connector Descriptions:

- **Ccsds_Space_Packet_T_Recv_Sync** - The ccsds packet receive connector.
- **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: Ccsds Command Depacketizer Invoker Connectors

Name	Kind	Type	Return_Type	Count
Command_T_Send	send	Command.T	-	1
Data_Product_T_Send	send	Data_Product.T	-	1
Event_T_Send	send	Event.T	-	1

Packet_T_Send	send	Packet.T	-	1
Sys_Time_T_Get	get	-	Sys_Time.T	1
Command_Response_T_Send	send	Command_Response.T	-	1

Connector Descriptions:

- **Command_T_Send** - The packet send connector
- **Data_Product_T_Send** - Data products are sent out of this connector.
- **Event_T_Send** - Events are sent out of this connector.
- **Packet_T_Send** - Error packets are sent out of this connector.
- **Sys_Time_T_Get** - The system time is retrieved via this connector.
- **Command_Response_T_Send** - This connector is used to register the components commands with the command router component.

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: Ccsds Command Depacketizer 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

This component contains no implementation class initialization, meaning there is no `init` subprogram for this component.

3.6 Commands

These are the commands for the component.

Table 4: Ccsds Command Depacketizer Commands

Local ID	Command Name	Argument Type
0	`Reset_Counts`	-

Command Descriptions:

- **`Reset_Counts`** - This command resets the internal counts for the data products.

3.7 Parameters

The Ccsds Command Depacketizer component has no parameters.

3.8 Events

Below is a list of the events for the Ccsds Command Depacketizer component.

Table 5: Ccsds Command Depacketizer Events

Local ID	Event Name	Parameter Type
0	`Invalid_Packet_Checksum`	`Invalid_Packet_Xor8_Info.T`
1	`Invalid_Packet_Type`	`Ccsds_Primary_Header.T`
2	`Packet_Too_Small`	`Invalid_Packet_Length.T`
3	`Packet_Too_Large`	`Invalid_Packet_Length.T`
4	`No_Secondary_Header`	`Ccsds_Primary_Header.T`
5	`Counts_Reset`	-
6	`Invalid_Command_Received`	`Invalid_Command_Info.T`

Event Descriptions:

- **`Invalid_Packet_Checksum`** - A packet was received with an invalid checksum
- **`Invalid_Packet_Type`** - A packet was received with an invalid ccsds packet type. The expected packet type is a telecommand, but a telemetry packet was received.
- **`Packet_Too_Small`** - The packet received was too small to contain necessary command information.
- **`Packet_Too_Large`** - The packet received was too large and is bigger than the size of a command.
- **`No_Secondary_Header`** - A packet was received without a secondary header, but the secondary header is required.
- **`Counts_Reset`** - A command was received to reset the counts.
- **`Invalid_Command_Received`** - A command was received with invalid parameters.

3.9 Data Products

Data products for the CCSDS Command Depacketizer component.

Table 6: Ccsds Command Depacketizer Data Products

Local ID	Data Product Name	Type
`0x0000 (0)`	`Rejected_Packet_Count`	`Packed_U16.T`
`0x0001 (1)`	`Accepted_Packet_Count`	`Packed_U16.T`

Data Product Descriptions:

- **`Rejected_Packet_Count`** - The number of packets rejected by the component due to invalid data
- **`Accepted_Packet_Count`** - The number of packets accepted by the component

3.10 Data Dependencies

The Ccsds Command Depacketizer component has no data dependencies.

3.11 Packets

Packets for the CCSDS Command Depacketizer component.

Table 7: Ccsds Command Depacketizer Packets

Local ID	Packet Name	Type
`0x0000 (0)`	`Error_Packet`	`Ccsds_Space_Packet.T`

Packet Descriptions:

- **`Error_Packet`** - This packet contains a CCSDS packet that was dropped due to error.

3.12 Faults

The Ccsds Command Depacketizer component has no faults.

4 Unit Tests

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

4.1 *Ccsds_ Command_ Depacketizer_ Tests* Test Suite

This is a unit test suite for the CCSDS Command Depacketizer component

Test Descriptions:

- **`Test_Nominal_Depacketization`** - This unit test exercises the nominal behavior of the ccsds command depacketizer.
- **`Test_Invalid_Packet_Checksum`** - This unit test makes sure that packets with invalid checksums are reported and dropped.
- **`Test_Invalid_Packet_Type`** - This unit test makes sure that packets with invalid packet types are reported and dropped.

- **Test_Packet_Too_Small** - This unit test makes sure that packets that are too small to hold a valid command are reported and dropped.
- **Test_Packet_Too_Large** - This unit test makes sure that packets that are too large to hold a valid command are reported and dropped.
- **Test_Packet_Without_Secondary_Header** - This unit test makes sure that packets that do not include a secondary header are reported and dropped.
- **Test_Pad_Bytes** - This unit test makes use of the function code in the secondary header to denote a different number of pad bytes. It makes sure the component responds appropriately.
- **Test_Reset_Counts** - This unit test tests the reset data products command.
- **Test_Invalid_Command** - This unit test makes sure the component handles an invalid command appropriately.

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..

Ccsds_Command_Header.T:

Record for a LASP-specific CCSDS command header.

Table 8: Ccsds_Command_Header Packed Record : 64 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Primary_Header	Ccsds_Primary_Header.T	-	48	0	47
Secondary_Header	Ccsds_Command_Secondary_Header.T	-	16	48	63

Field Descriptions:
- **Primary_Header** - The CCSDS primary header
- **Secondary_Header** - The command secondary header

Ccsds_Command_Secondary_Header.T:

Record for the LASP-specific command secondary header. *Preamble (inline Ada definitions):*

```
type Function_Code_Type is mod 2**7;
type One_Bit_Pad_Type is mod 2**1;
```

Table 9: Ccsds_Command_Secondary_Header Packed Record : 16 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
Reserved	One_Bit_Pad_Type	0 to 1	1	0	0
Function_Code	Function_Code_Type	0 to 127	7	1	7
Checksum	Interfaces.Unsigned_8	0 to 255	8	8	15

Field Descriptions:

- **Reserved** - Reserve bit.
- **Function_Code** - The command function code.
- **Checksum** - An 8 bit checksum over the entire command packet

Ccsds_Primary_Header.T:

Record for the CCSDS Packet Primary Header *Preamble (inline Ada definitions):*

```
subtype Three_Bit_Version_Type is Interfaces.Unsigned_8 range 0 .. 7;
type Ccsds_Apid_Type is mod 2**11;
type Ccsds_Sequence_Count_Type is mod 2**14;
```

Table 10: Ccsds_Primary_Header Packed Record : 48 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
Version	Three_Bit_Version_Type	0 to 7	3	0	2
Packet_Type	Ccsds_Enums.Ccsds_Packet_Type.E	0 => Telemetry 1 => Telecommand	1	3	3
Secondary_Header	Ccsds_Enums.Ccsds_Secondary_Header_Indicator.E	0 => Secondary_Header_Not_Present 1 => Secondary_Header_Present	1	4	4
Apid	Ccsds_Apid_Type	0 to 2047	11	5	15
Sequence_Flag	Ccsds_Enums.Ccsds_Sequence_Flag.E	0 => Continuationsegment 1 => Firstsegment 2 => Lastsegment 3 => Unsegmented	2	16	17

Sequence_Count	Ccsds_Sequence_Count_Type	0 to 16383	14	18	31
Packet_Length	Interfaces.Unsigned_16	0 to 65535	16	32	47

Field Descriptions:

- **Version** - Packet Version Number
- **Packet_Type** - Packet Type
- **Secondary_Header** - Does packet have CCSDS secondary header
- **Apid** - Application process identifier
- **Sequence_Flag** - Sequence Flag
- **Sequence_Count** - Packet Sequence Count
- **Packet_Length** - This is the packet data length. One added to this number corresponds to the number of bytes included in the data section of the CCSDS Space Packet.

Ccsds_Space_Packet.T:

Record for the CCSDS Space Packet *Preamble (inline Ada definitions):*

```
use Basic_Types;
subtype Ccsds_Data_Type is Byte_Array (0 ..
↪  Configuration.Ccsds_Packet_Buffer_Size - 1);
```

Table 11: Ccsds_Space_Packet Packed Record : 10240 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Ccsds_Primary_Header.T	-	48	0	47	-
Data	Ccsds_Data_Type	-	10192	48	10239	Header.Packet_Length

Field Descriptions:

- **Header** - The CCSDS Primary Header
- **Data** - User Data Field

Command.T:

Generic command packet for holding arbitrary commands

Table 12: 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 13: 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 14: 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 15: 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 16: 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 17: 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 18: 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 19: 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.

Invalid_Packet_Length.T:

A packed record which holds data related to an invalid command packet length.

Table 20: Invalid_Packet_Length Packed Record : 112 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
`Ccsds_Header`	`Ccsds_Primary_Header.T`	-	48	0	47
`Length`	`Integer`	`-2147483648 to 2147483647`	32	48	79
`Length_Bound`	`Integer`	`-2147483648 to 2147483647`	32	80	111

Field Descriptions:

- **`Ccsds_Header`** - The packet identifier
- **`Length`** - The packet length
- **`Length_Bound`** - The packet length bound that the length failed to meet.

Invalid_Packet_Xor8_Info.T:

A packed record which holds data related to an invalid checksummed CCSDS command packet.

Table 21: Invalid_Packet_Xor8_Info Packed Record : 80 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
`Ccsds_Header`	`Ccsds_Command_Header.T`	-	64	0	63
`Computed_Checksum`	`Xor_8.Xor_8_Type`	`0 to 255`	8	64	71
`Expected_Checksum`	`Xor_8.Xor_8_Type`	`0 to 255`	8	72	79

Field Descriptions:

- **`Ccsds_Header`** - The CCSDS command header.
- **`Computed_Checksum`** - The computed XOR of the entire packet. This should be 0 if the packet passes.
- **`Expected_Checksum`** - The XOR included in the CCSDS packet secondary header.

Packed_U16.T:

Single component record for holding packed unsigned 16-bit value.

Table 22: Packed_U16 Packed Record : 16 bits

Name	**Type**	**Range**	**Size (Bits)**	**Start Bit**	**End Bit**
Value	Interfaces. Unsigned_16	0 to 65535	16	0	15

Field Descriptions:

- **`Value`** - The 16-bit unsigned integer.

Packet.T:

Generic packet for holding arbitrary data

Table 23: 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 24: 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

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 25: 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.

Ccsds_Enums.Ccsds_Packet_Type.E:

This single bit is used to identify that this is a Telecommand Packet or a Telemetry Packet. A Telemetry Packet has this bit set to value 0; therefore, for all Telecommand Packets Bit 3 shall be set to value 1.

Table 26: Ccsds_Packet_Type Literals:

Name	Value	Description
Telemetry	0	Indicates a telemetry packet
Telecommand	1	Indicates a telecommand packet

Ccsds_Enums.Ccsds_Secondary_Header_Indicator.E:

This one bit flag signals the presence (Bit 4 = 1) or absence (Bit 4 = 0) of a Secondary Header data structure within the packet.

Table 27: Ccsds_Secondary_Header_Indicator Literals:

Name	Value	Description
Secondary_Header_Not_Present	0	Indicates that the secondary header is not present within the packet
Secondary_Header_Present	1	Indicates that the secondary header is present within the packet

Ccsds_Enums.Ccsds_Sequence_Flag.E:

This flag provides a method for defining whether this packet is a first, last, or intermediate component of a higher layer data structure.

Table 28: Ccsds_Sequence_Flag Literals:

Name	Value	Description
Continuationsegment	0	Continuation component of higher data structure
Firstsegment	1	First component of higher data structure
Lastsegment	2	Last component of higher data structure
Unsegmented	3	Standalone packet

Command_Enums.Command_Response_Status.E:

This status enumeration provides information on the success/failure of a command through the command response connector.

Table 29: 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.