Ccsds Router

Component Design Document

1 Description

This component routes CCSDS packets to output connectors based on a static table matching APID to the output connector index. Table lookup is done by binary searching against APID. The look up returns a list of indexes which to route that packet. This component can receive packets either synchronously, or asynchronously and can be made either active or passive depending on the desired use case. If a packet is received with an APID not found in the routing table then it is forwarded out a separate CCSDS connector, if it is connected. In addition to routing, this component can be configured to check the sequence counts on incoming packets, report discontiguous sequence counts, and drop duplicates.
Note, a race condition exists if packets of the same APID come in simultaneously on both the sync and async CCSDS connectors, and sequence counts are being checked, where the sequence count checking might get corrupted. This use case is not foreseen as actually happening, so complicating the component with protected objects seems unnecessary.
Note that an autocoder exists to ease the writing of the CCSDS Router input router table. See documentation for this autocoder in the local gen/doc subdirectory.

2 Requirements

The requirements for the CCSDS Router component are specified below.

1. The component shall route CCSDS packets to zero or many predetermined output connectors based on the packet's APID.
2. The component shall be able to produce a warning event when a CCSDS packet with the same APID is received with a non-incrementing sequence identifier.
3. The component shall be able to drop duplicate CCSDS packets which contain the same APID and sequence identifier.

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** - *either*
- **Number of Connectors** - 7
- **Number of Invokee Connectors** - 2
- **Number of Invoker Connectors** - 5
- **Number of Generic Connectors** - *None*

- **Number of Generic Types** - *None*
- **Number of Unconstrained Arrayed Connectors** - 1
- **Number of Commands** - *None*
- **Number of Parameters** - *None*
- **Number of Events** - 4
- **Number of Faults** - *None*
- **Number of Data Products** - *None*
- **Number of Data Dependencies** - *None*
- **Number of Packets** - 1

3.2 Diagram

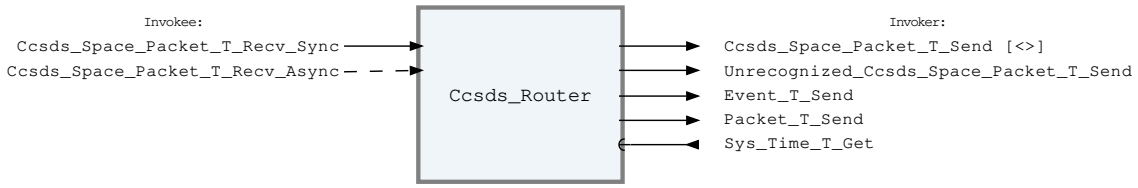

Figure 1: Ccsds Router component diagram.

The CCSDS Router can be used in a variety of different ways, each of which supports a different execution context.

The following diagram shows the most common way that the CCSDS Router component might operate in the context of an assembly.

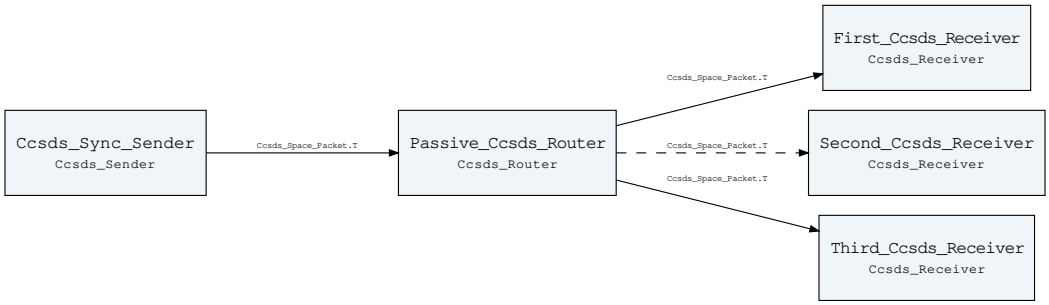

Figure 2: Example usage of a passive CCSDS Router which routes packets on the thread of its caller.

In the above context diagram the CCSDS Router is made passive. This means that any CCSDS packet passed to it will be routed on the thread of the calling component, since the calling component uses the `Ccsds_Space_Packet_Recv_Sync` connector.

Sometimes it is desirable to do the routing on a different thread of execution to decouple this processing from work going on upstream of the component. The most obvious way to accomplish this is to make the CCSDS Router component active, so it has its own thread of execution. The context diagram below shows this setup.

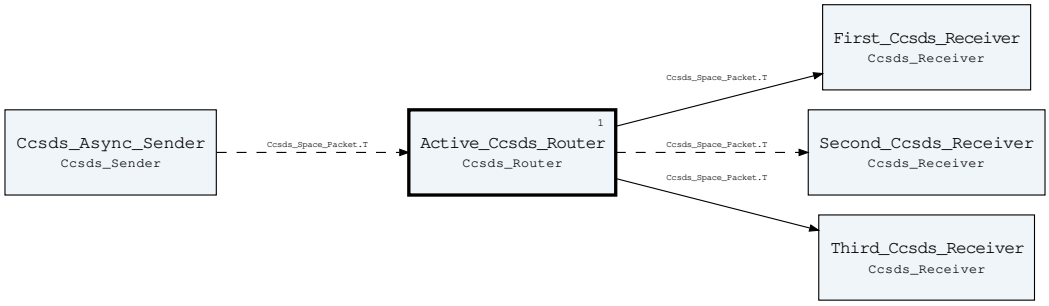

Figure 3: Example usage of an active CCSDS Router which routes any packets found on its queue using its own internal thread of execution.

In this case, the calling component passes CCSDS packets to the CCSDS Router via the `Ccsds_Space_Packet_Recv_` connector, which puts the CCSDS packets on the CCSDS Router's internal queue. When the CCSDS Router is given execution time, it pops the CCSDS packets off of the queue and routes them downstream.

It is also possible to achieve both synchronous and asynchronous routing using the same CCSDS Router component (and thus the same internal routing table data structure). This might be useful if you have a normal data path which takes the asynchronous path, but a time critical data path which requires synchronous routing. This might be the case when routing time critical packets related to control or fault protection. The following context diagram shows this case.

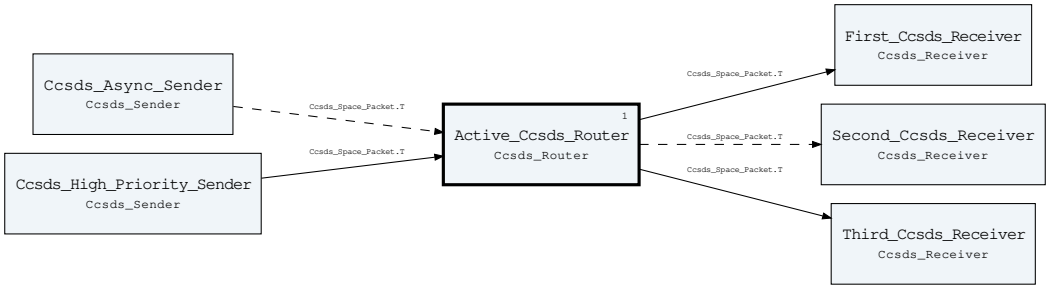

Figure 4: Example usage of a mixed CCSDS Router which supports both synchronous and asynchronous routing.

Note, there is a limitation with this use case. A CCSDS packet with the same APID should not be sent through both the synchronous and asynchronous invokee connectors at the same time if sequence number checking is enabled. Since the router table is not a protected object this could produce errant warnings about sequence number tracking, since it is expected that two updates to the sequence numbers tracked in this table will not occur to the same APID at the same time. There has been no real world use case identified which would require that the internal table become a protected object. If a use case is identified, this issue can be revisited.

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 Router Invokee Connectors

Name	Kind	Type	Return_Type	Count

Ccsds_Space_Packet_T_Recv_Sync	recv_sync	Ccsds_Space_Packet.T	-	1
Ccsds_Space_Packet_T_Recv_Async	recv_async	Ccsds_Space_Packet.T	-	1

Connector Descriptions:

- **Ccsds_Space_Packet_T_Recv_Sync** - The synchronous ccsds packet receive connector.
- **Ccsds_Space_Packet_T_Recv_Async** - The asynchronous ccsds packet receive 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: Ccsds Router Asynchronous Connectors

Name	Type	Max Size (bytes)
Ccsds_Space_Packet_T_Recv_Async	Ccsds_Space_Packet.T	1285

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: Ccsds Router Invoker Connectors

Name	Kind	Type	Return_Type	Count
Ccsds_Space_Packet_T_Send	send	Ccsds_Space_Packet.T	-	<>
Unrecognized_Ccsds_Space_Packet_T_Send	send	Ccsds_Space_Packet.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

Connector Descriptions:

- **Ccsds_Space_Packet_T_Send** - The ccsds packet send connector.
- **Unrecognized_Ccsds_Space_Packet_T_Send** - Ccsds packets not found in the routing table are forwarded out this connector if it is connected.
- **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.

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: Ccsds Router Base Initialization Parameters

Name	Type
Ccsds_Space_Packet_T_Send_Count	Connector_Count_Type
Queue_Size	Natural

Parameter Descriptions:

- **Ccsds_Space_Packet_T_Send_Count** - The size of the Ccsds_Space_Packet_T_Send invoker connector array.
- **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: Ccsds Router Set Id Bases Parameters

Name	Type
Event_Id_Base	Event_Types.Event_Id_Base
Packet_Id_Base	Packet_Types.Packet_Id_Base

Parameter Descriptions:

- **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.

3.4.4 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 routing table which maps CCSDS packet APIDs to a list of output connector indexes. This is provided as part of the initialization function. The `init` subprogram requires the following parameters:

Table 6: Ccsds Router Implementation Initialization Parameters

Name	Type	Default Value
Table	Ccsds_Router_Types. Router_Table_Entry_ Array	*None provided*
Report_Unrecognized_Apids	Boolean	True

Parameter Descriptions:

- **Table** - An array of router table entries which include routing and sequence count checking information.
- **Report_Unrecognized_Apids** - Should the component report unrecognized APIDs by sending out an error packet and event, True, or should it not report them at all, False.

3.5 Events

Below is a list of the events for the Ccsds Router component.

Table 7: Ccsds Router Events

Local ID	Event Name	Parameter Type
0	Unrecognized_Apid	Ccsds_Primary_Header.T
1	Dropped_Packet	Ccsds_Primary_Header.T
2	Unexpected_Sequence_Count_Received	Unexpected_Sequence_ Count.T
3	Dropped_Duplicate_Packet	Ccsds_Primary_Header.T

Event Descriptions:

- **Unrecognized_Apid** - A packet was received with an APID that was not found in the routing table. The packet was dropped.
- **Dropped_Packet** - The component's queue overflowed and a packet with the following header was dropped.
- **Unexpected_Sequence_Count_Received** - A packet with an unexpected sequence count was received.
- **Dropped_Duplicate_Packet** - The component's received two or more packets in a row with identical sequence counts. The duplicate packet was dropped.

3.6 Packets

Packets for the CCSDS Router component.

Table 8: Ccsds Router 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.

4 Unit Tests

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

4.1 *Unrecognized_Apid_Tests* Test Suite

This is a unit test suite for the CCSDS Router, when the unrecognized apid connector is connected.

Test Descriptions:

- **`Test_Unrecognized_Id`** - This unit test makes sure that packets with APIDs not found in the routing table are forwarded and reported appropriately.
- **`Test_Unrecognized_Id_No_Report`** - This unit test makes sure that packets with APIDs not found in the routing table are forwarded and not reported appropriately.

4.2 *Ccsds_Router_Tests* Test Suite

This is a unit test suite for the CCSDS Router.

Test Descriptions:

- **`Test_Initialization`** - This unit test tests all permutations of initializing the component and makes sure improper initialization results in a runtime assertion.
- **`Test_Nominal_Routing`** - This unit test tests that CCSDS packets are routed to the expected destination component based on the routing table.
- **`Test_Unrecognized_Id`** - This unit test makes sure that packets with APIDs not found in the routing table are dropped and reported appropriately.
- **`Test_Dropped_Packet`** - This unit test tests that when the component's internal queue is overflowed, that the packet is dropped and reported appropriately.
- **`Test_Sequence_Count_Warning`** - This unit test tests that CCSDS packets of non-contiguous sequence counts produce a warning event.
- **`Test_Duplicate_Packet_Drop`** - This unit test tests that CCSDS packets of identical subsequent sequence counts get dropped.

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..

Ccsds_Primary_Header.T:

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

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

Table 9: 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 10: 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

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

Packet.T:

Generic packet for holding arbitrary data

Table 13: 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 14: 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 15: 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.

Unexpected_Sequence_Count.T:

A packed record which holds data related to an unexpected sequence count.

Table 16: Unexpected_Sequence_Count Packed Record : 80 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Ccsds_Header	Ccsds_Primary_ Header.T	-	48	0	47
Received_Sequence_ Count	Interfaces. Unsigned_16	0 to 65535	16	48	63
Expected_Sequence_ Count	Interfaces. Unsigned_16	0 to 65535	16	64	79

Field Descriptions:

- **Ccsds_Header** - The packet identifier
- **Received_Sequence_Count** - The sequence count received in the ccsds command.
- **Expected_Sequence_Count** - The sequence count that was expected to see in the command. +1 from the last sequence count received.

5.2 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 17: 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 18: 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 19: 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