Register Stuffer

Component Design Document

1 Description

This component services commands to stuff and dump registers. This component is different than the memory stuffer/dumper in that it atomically sets 32-bit little endian registers, which is a requirement on some hardware. It rejects commands to stuff or dump addresses that are not on a 4-byte boundary. Note that this component assumes all registers it accesses are little endian. Another version of this component needs to be used to access registers that are big endian.

2 Requirements

The requirements for the Register Stuffer component are specified below.

1. The component shall respond to commands to set a 32-bit little endian register.
2. The component shall respond to commands to read a 32-bit little endian register.
3. The component shall publish a data product that reflects the last written register address and value.
4. The component shall publish a data product that reflects the last read register address and value.

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** - 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** - *None*
- **Number of Commands** - 4
- **Number of Parameters** - *None*
- **Number of Events** - 10
- **Number of Faults** - *None*

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

3.2 Diagram

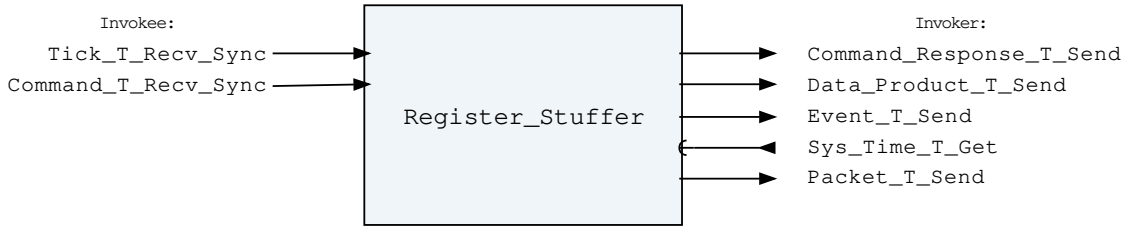

Figure 1: Register Stuffer component diagram.

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: Register Stuffer Invokee Connectors

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

Connector Descriptions:

- **Tick_T_Recv_Sync** - This tick is used to keep track of the armed state timeout and send the data product relating the current timeout value.
- **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: Register Stuffer Invoker Connectors

Name	Kind	Type	Return_Type	Count
Command_Response_T_Send	send	Command_Response.T	-	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
Packet_T_Send	send	Packet.T	-	1

Connector Descriptions:

- **Command_Response_T_Send** - This connector is used to send the command response back to the command router.
- **Data_Product_T_Send** - Data products are sent out of this connector.
- **Event_T_Send** - The event send connector.
- **Sys_Time_T_Get** - The system time is retrieved via this connector.
- **Packet_T_Send** - Packets 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: Register Stuffer 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. Configuration for the register stuffer component. The `init` subprogram requires the following parameters:

Table 4: Register Stuffer Implementation Initialization Parameters

Name	Type	Default Value
Protect_Registers	Boolean	*None provided*

Parameter Descriptions:

- **Protect_Registers** - If set to True, the arm command will be required before each register write command. This does not affect register reads. If set to False, an arm command is not required before each register write command.

3.6 Commands

These are the commands for the Register Stuffer component.

Table 5: Register Stuffer Commands

Local ID	Command Name	Argument Type
0	Write_Register	Register_Value.T
1	Read_Register	Packed_Address.T
2	Arm_Protected_Write	Packed_Arm_Timeout.T
3	Dump_Registers	Register_Dump_Packet_Header.T

Command Descriptions:

- **Write_Register** - Write the value of a register.
- **Read_Register** - Read the value of a register and reflect it in a data product.
- **Arm_Protected_Write** - An arm command which enables the next write command to a register to be accepted. The armed state of the component will expire on the next command to this component no matter what it is or after the configurable timeout.
- **Dump_Registers** - Read the value of multiple registers and dump them into a packet.

3.7 Parameters

The Register Stuffer component has no parameters.

3.8 Events

Events for the Register Stuffer component.

Table 6: Register Stuffer Events

Local ID	Event Name	Parameter Type
0	Invalid_Register_Address	Packed_Address.T
1	Register_Written	Register_Value.T
2	Register_Read	Register_Value.T
3	Invalid_Command_Received	Invalid_Command_Info.T
4	Rejected_Protected_Register_Write	Register_Value.T
5	Armed	Packed_Arm_Timeout.T
6	Unarmed	-
7	Unarmed_Timeout	-

8	Registers_Dumped	Register_Dump_Packet_Header.T
9	Address_Range_Overflow	Register_Dump_Packet_Header.T

Event Descriptions:

- **Invalid_Register_Address** - The register address provided does not start on a 32-bit boundary.
- **Register_Written** - The specified register was written to the commanded value.
- **Register_Read** - The specified register was read from.
- **Invalid_Command_Received** - A command was received with invalid parameters.
- **Rejected_Protected_Register_Write** - The specified register could not be written because the component was not armed first.
- **Armed** - The component received the arm command and is now armed.
- **Unarmed** - The component received a command and is now unarmed.
- **Unarmed_Timeout** - The component armed state timed out and is now unarmed.
- **Registers_Dumped** - The specified registers were dumped.
- **Address_Range_Overflow** - The specified register address range overflows the address space.

3.9 Data Products

Data products for the Register Stuffer component.

Table 7: Register Stuffer Data Products

Local ID	Data Product Name	Type
0x0000 (0)	Last_Register_Written	Register_Value.T
0x0001 (1)	Last_Register_Read	Register_Value.T
0x0002 (2)	Armed_State	Packed_Arm_State.T
0x0003 (3)	Armed_State_Timeout	Packed_Arm_Timeout.T

Data Product Descriptions:

- **Last_Register_Written** - The address and value of the last written register.
- **Last_Register_Read** - The address and value of the last read register.
- **Armed_State** - The current armed/unarmed state of the component.
- **Armed_State_Timeout** - The time remaining (in ticks) until the armed state expires.

3.10 Data Dependencies

The Register Stuffer component has no data dependencies.

3.11 Packets

Packets for the register stuffer.

Table 8: Register Stuffer Packets

Local ID	Packet Name	Type
0x0000 (0)	Register_Packet	Register_Dump_Packet.T

Packet Descriptions:

- **Register_Packet** - This packet contains dumped register values.

3.12 Faults

The Register Stuffer component has no faults.

4 Unit Tests

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

4.1 *Register_ Stuffer_ Tests* Test Suite

This is a unit test suite for the Register Stuffer component

Test Descriptions:

- **Test_Nominal_Register_Write** - This unit test makes sure the component can write registers by command.
- **Test_Nominal_Register_Read** - This unit test makes sure the component can read registers by command.
- **Test_Bad_Address** - This unit test makes sure the component rejects reading or writing registers that are not 4-byte aligned.
- **Test_Invalid_Command** - This unit test makes sure a malformed command is rejected.
- **Test_Protected_Register_Write** - This unit test makes sure the protected register write feature works as intended.
- **Test_Nominal_Dump_One_Registers** - This unit test makes sure the component can dump one register by command.
- **Test_Nominal_Dump_Max_Registers** - This unit test makes sure the component can dump the maximum number of registers by command.
- **Test_Dump_Four_Registers** - This unit test makes sure the component can dump four registers by command.

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_Arm_State.T:

Holds the armed state.

Table 18: Packed_Arm_State Packed Record : 8 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
State	Command_Protector_Enums.Armed_State.E	0 => Unarmed 1 => Armed	8	0	7

Field Descriptions:

- **State** - The armed/unarmed status.

Packed_Arm_Timeout.T:

Holds the armed state timeout. *Preamble (inline Ada definitions):*

```
1 type Arm_Timeout_Type is new Natural range 0 .. 255;
```

Table 19: Packed_Arm_Timeout Packed Record : 8 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Timeout	Arm_Timeout_Type	0 to 255	8	0	7

Field Descriptions:

- **`Timeout`** - The timeout value (in ticks).

Packed_U32.T:

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

Table 20: 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.

Packet.T:

Generic packet for holding arbitrary data

Table 21: 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 22: 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

Register_Dump_Packet.T:

Packed Record for N Dumped Registers

Table 23: Register_Dump_Packet Packed Record : 9968 bits *(maximum)*

Name	Type	Range	Size (Bits)	Start Bit	End Bit	Variable Length
Header	Register_Dump_Packet_Header.T	-	80	0	79	-
Buffer	Register_Dump_Packet_Array.T	-	9888	80	9967	Header.Num_Registers

Field Descriptions:

- **`Header`** - *No description provided.*
- **`Buffer`** - *No description provided.*

Register_Dump_Packet_Array.T:

An array of Packed U32 Register Values.

Table 24: Register_Dump_Packet_Array Packed Array : 9888 bits

Type	Range	Element Size (Bits)	Length	Total Size (Bits)
`Packed_U32.T`	-	32	309	9888

Register_Dump_Packet_Header.T:

Packet Header for Register Stuffer Packets *Preamble (inline Ada definitions):*

```
1 subtype N_Registers is Integer range 1 .. 309;
```

Table 25: Register_Dump_Packet_Header Packed Record : 80 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Start_Address	System.Address	-	64	0	63
Num_Registers	N_Registers	1 to 309	16	64	79

Field Descriptions:

- **`Start_Address`** - Starting address of the N register dump
- **`Num_Registers`** - Number of Registers to Dump

Register_Value.T:

A register value packed record. *Preamble (inline Ada definitions):*

```
-- Add subtype for conversion of System.Address type so we can check alignment
   and prevent overflow.
subtype Address_Mod_Type is Interfaces.Unsigned_64;
```

Table 26: Register_Value Packed Record : 96 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Address	System.Address	-	64	0	63
Value	Interfaces. Unsigned_32	0 to 4294967295	32	64	95

Field Descriptions:

- **`Address`** - The address of the register.
- **`Value`** - The value to write to or read from the register

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 27: 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.

Tick.T:

The tick datatype used for periodic scheduling. Included in this type is the Time associated with a tick and a count.

Table 28: Tick Packed Record : 96 bits

Name	Type	Range	Size (Bits)	Start Bit	End Bit
Time	Sys_Time.T	-	64	0	63

Count	Interfaces. Unsigned_32	0 to 4294967295	32	64	95

Field Descriptions:

- **Time** - The timestamp associated with the tick.
- **Count** - The cycle number of the tick.

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 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.

Command_Protector_Enums.Armed_State.E:

This type enumerates the armed state for the component.

Table 30: Armed_State Literals:

Name	Value	Description
Unarmed	0	The component is unarmed. Any protected commands received will be rejected.
Armed	1	The component is armed. If the next command received is a protected command, it will be forwarded.