Interrupt Pender

Component Design Document

1 Description

The Interrupt Pender is used to block the execution of a calling component until an interrupt is received. When an attached component calls the Wait_On_Interrupt_Data_Type_Return connector its execution is halted. When an interrupt is received, the Interrupt Pender passes a user-defined interrupt data type, populated by a user-defined interrupt handler, to the blocked component, releasing it to continue its execution. Note, only one external component should be attached to the Wait_On_Interrupt_Data_Type_Return connector or a runtime error will be thrown by the Ravenscar runtime system due to two tasks waiting on a single protected entry.

2 Requirements

The requirements for the Interrupt Pender component are specified below.

1. The component shall attach to a single, user-defined interrupt.
2. When an interrupt is received, the component shall pass a user-defined data type to a user-defined custom interrupt handler.
3. When an interrupt is received, the component shall pass the user-defined interrupt data to an attached external component.
4. The component shall contain a connector which blocks a calling component's execution until an interrupt is received.

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** - 2
- **Number of Invokee Connectors** - 1
- **Number of Invoker Connectors** - 1
- **Number of Generic Connectors** - *None*
- **Number of Generic Types** - 2
- **Number of Unconstrained Arrayed Connectors** - *None*
- **Number of Commands** - *None*
- **Number of Parameters** - *None*

- **Number of Events** - *None*
- **Number of Faults** - *None*
- **Number of Data Products** - *None*
- **Number of Data Dependencies** - *None*
- **Number of Packets** - *None*

3.2 Diagram

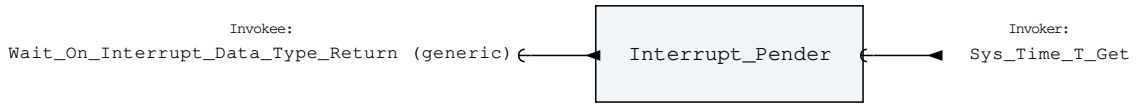

Figure 1: Interrupt Pender component diagram.

The Interrupt Pender has two connectors, one for returning a user defined (generic) data type that is filled in by the interrupt handler, and another for timestamping when the interrupt occurs. The type of the generic `Wait_On_Interrupt_Data_Type_Return` connector is determined by what data needs to be collected in the interrupt, and what data needs to be processed after an interrupt occurs.

The following diagram shows an example of how the Interrupt Pender component might operate in the context of an assembly.

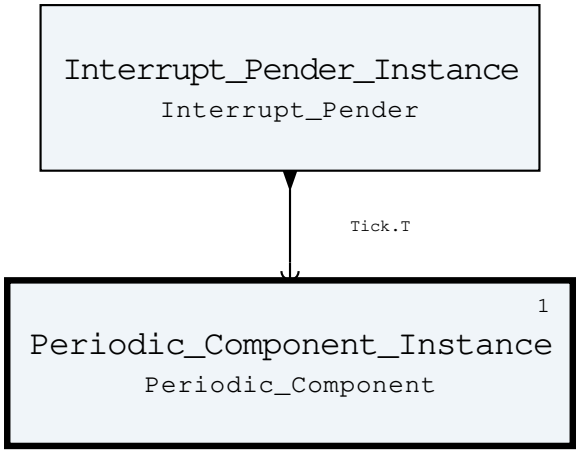

Figure 2: Example usage of the Interrupt Pender in an assembly, blocking a periodic task from executing until an interrupt is received.

The Interrupt Pender is attached to an active component that should be executed periodically. The Periodic Component blocks when invoking the `Wait_On_Interrupt_Data_Type_Return` and is released when the Interrupt Pender receives an interrupt. In this example a `Tick.T` record is sent between the component. However, since the type of this connector is a user-defined generic, it can be instantiated to be any type. When the Periodic Component finishes its execution cycle, it will call `Wait_On_Interrupt_Data_Type_Return` again, blocking until the next interrupt.

This use of the Interrupt Pender is ideal for this kind of triggering of periodic execution. Note that if an interrupt is received while the Periodic Component is still executing, then the next time that the Periodic Component calls `Wait_On_Interrupt_Data_Type_Return` it will not block, and instead will immediately receive the generic data from that interrupt. If two or more interrupts are received while the Periodic Component is still executing, then all interrupts except the last will

be effectively ignored (dropped). To avoid dropping interrupts in this situation, consider using the Interrupt Servicer component with an asynchronous connection (see its documentation).

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: Interrupt Pender Invokee Connectors

Name	Kind	Type	Return_Type	Count
Wait_On_ Interrupt_Data_ Type_Return	return	-	Interrupt_Data_ Type (generic)	1

Connector Descriptions:

- **Wait_On_Interrupt_Data_Type_Return** - Calling this connector causes execution to block until an interrupt occurs. Note, only one external component should be attached to this connector or a runtime error will be thrown by the Ravenscar runtime system due to two tasks waiting on a single protected entry.

3.3.2 Invoker Connectors

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

Table 2: Interrupt Pender Invoker Connectors

Name	Kind	Type	Return_Type	Count
Sys_Time_T_Get	get	-	Sys_Time.T	1

Connector Descriptions:

- **Sys_Time_T_Get** - The system time is retrieved via this connector.

3.4 Interrupts

Below is a list of the interrupts for the Interrupt Pender component.
Interrupt Descriptions:

- **Interrupt** - This component counts the number of times this interrupt occurs.

3.5 Initialization

Below are details on how the component should be initialized in an assembly.

3.5.1 Generic Component Instantiation

The Interrupt_Data_Type. The instantiation of this type is often determined by what data needs to be collected during an interrupt and how a connected component intends to process that data. This type is a user defined type that is passed into the user's custom interrupt handler as an in-out parameter and then passed to a downstream components out of the Wait_On_Interrupt_Data_Type_Return connector. Usually, this type will be used to capture

data related to an interrupt in the interrupt handler. Often, a timestamp, relating when the interrupt occurred, will be included in this custom type. The Interrupt Pender will automatically insert the timestamp into the custom type if the second generic parameter, the Set_Interrupt_Data_Time procedure, is provided. This component contains generic formal types. These generic formal types must be instantiated with a valid actual type prior to component initialization. This is done by specifying types for the following generic formal parameters:

Table 3: Interrupt Pender Generic Formal Types

Name	Formal Type Definition
`Interrupt_Data_Type`	`type Interrupt_Data_Type is private;`
`Set_Interrupt_Data_Time`	`with procedure Set_Interrupt_Data_Time (Interrupt_Data : in out Interrupt_Data_Type; Time : in Sys_Time.T) is null;`

Generic Formal Type Descriptions:

- **`Interrupt_Data_Type`** - The user's custom datatype that is is set in the custom interrupt handler and then passed to downstream components. If you do not foresee needing to collect specific data in the interrupt handler for the downstream component, consider using the Tick.T type, which includes a timestamp and a count.
- **`Set_Interrupt_Data_Time`** - This is an optional generic parameter that is used if the Interrupt_Data_Type includes a timestamp which should be filled in when an interrupt occurs. This parameter is a user defined procedure that, given a time, copies that time to the Interrupt_Data_Type. If this parameter is not provided, then time will not be set in the Interrupt_Data_Type. In this case, the user should not connect the Sys_Time_T_Get connector to avoid the CPU overhead of fetching a time that is never used. Note, if you are using Tick.T for the Interrupt_Data_Type, consider using the Handler procedure in the Tick_Interrupt_Handler package to instantiate this parameter.

3.5.2 Component Instantiation

This component contains the following instantiation parameters in its discriminant:

Table 4: Interrupt Pender Instantiation Parameters

Name	Type
`Custom_Interrupt_Procedure`	`Custom_Interrupt_Handler_Package. Interrupt_Procedure_Type`
`Interrupt_Id`	`Ada.Interrupts.Interrupt_Id`
`Interrupt_Priority`	`System.Interrupt_Priority`

Parameter Descriptions:

- **`Custom_Interrupt_Procedure`** - A custom procedure to be called within the interrupt handler. The null procedure can be used here if no specific behavior is desired.
- **`Interrupt_Id`** - Interrupt identifier number for Interrupt
- **`Interrupt_Priority`** - Interrupt priority for Interrupt

3.5.3 Component Base Initialization

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

3.5.4 Component Set ID Bases

This component contains no commands, events, packets, faults or data products that need base indentifiers.

3.5.5 Component Map Data Dependencies

This component contains no data dependencies.

3.5.6 Component Implementation Initialization

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

4 Unit Tests

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

4.1 *Tests* Test Suite

This is a unit test suite for the Interrupt Pender component. It tests the component in a Linux environment, responding to signals.

Test Descriptions:

- **`Test_Interrupt_Handling`** - This unit test sends many interrupts to the Interrupt Pender component and expects it to produce ticks.

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.

```
1 -- Define the custom interrupt handling package type:
2 package Custom_Interrupt_Handler_Package is new Interrupt_Handlers
  ↪  (Interrupt_Data_Type);
```

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..

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 5: 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.