Interrupt
1.20

Features

- Define hardware triggered interrupts

General Description

The Interrupt component is an integral part of the Interrupt Design-Wide Resource system (see PSoC Creator Online Help, Design-Wide Resources section). The interrupt component is the means by which hardware triggered interrupts are defined.

When to use an Interrupt

Use an interrupt component whenever a hardware triggered interrupt is required.

Input/Output Connections

This section describes the various input and output connections for the Interrupt. An asterisk (*) in the list of I/O's states that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

int_signal – Input

The signal which generates the interrupt should be connected to this input. When the signal value becomes logic high, the interrupt will be triggered.

Component Parameters

The interrupt component does not have any parameters that are intended for general use.

Placement

Each interrupt component consumes one entry in the devices interrupt vector.

Resources

The Interrupt component uses the following device resources:

- Flash space for ISR code.
- An entry in the device interrupt vector.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "ISR_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "ISR".

Function	Description
ISR_Start	Sets up the interrupt to function.
ISR_Stop	Disables and removes the interrupt.
ISR_Interrupt	The default Interrupt handler for ISR.
ISR_SetVector	Sets "address" as the new ISR vector for the Interrupt.
ISR_GetVector	Gets the "address" of the current ISR vector for the Interrupt.
ISR_SetPriority	Sets the Priority of the Interrupt.
ISR_GetPriority	Gets the Priority of the Interrupt.
ISR_Enable	Enables the interrupt.
ISR_GetState	Gets the state (enabled, disabled) of the Interrupt.
ISR_Disable	Disables the Interrupt.
ISR_SetPending	Causes the Interrupt to enter the pending state, a software method of generating the interrupt.
ISR_ClearPending	Clears a pending interrupt.

void ISR_Start(void)

Description: Set up the interrupt and enable it.

Parameters: void.

Return Value: void.

Side Effects:

void ISR_Stop(void)

Description: Disables and removes the interrupt.

Parameters: void

Return Value: void

Side Effects:

void ISR_Interrupt(void)

Description: The default Interrupt Service Routine for the component. The user can add code between the START and END comments.

Parameters: void

Return Value: void

Side Effects:

void ISR_SetVector(cyisraddress address)

Description: Change the ISR vector for the Interrupt. The user can write his own ISR and use this function to change the vector to his own function.

Parameters: address: Address of the ISR to set in the interrupt vector table.

Return Value: void

Side Effects: The caller should disable this interrupt before calling this function and re enable it after.

cyisraddress ISR_GetVector(void)

Description: Gets the "address" of the current ISR vector for the Interrupt.

Parameters: void.

Return Value: cyisraddress, Address of the current ISR.

Side Effects:

void ISR_SetPriority(uint8 priority)

Description: Sets the Priority of the Interrupt.

Parameters: priority: Priority of the interrupt. 0 - 7, 0 being the highest.

Return Value: void

Side Effects:

uint8 ISR_GetPriority(void)

Description: Gets the Priority of the Interrupt.

Parameters: V

Return Value: Priority of the interrupt. 0 - 7, 0 being the highest.

Side Effects:

void ISR_Enable(void)

Description: Enables the interrupt with the interrupt controller.

Parameters: void

Return Value: void

Side Effects:

uint8 ISR_SetState(void)

Description: Gets the state (enabled, disabled) of the Interrupt.

Parameters: void

Return Value: 1 if enabled, 0 if disabled.

Side Effects:

void ISR_Disable(void)

Description: Disables the Interrupt with the interrupt controller.

Parameters: void

Return Value: void

Side Effects:

void ISR_SetPending(void)

Description: Causes the Interrupt to enter the pending state, a software method of generating the interrupt.

Parameters: void

Return Value: void

Side Effects: If interrupts are enabled and the interrupt is setup properly, the ISR will be entered (depending on the priority of this interrupt and other pending interrupts).

void ISR_ClearPending(void)

Description: Clears a pending interrupt with the interrupt controller.

Parameters: void

Return Value: void

Side Effects: Some interrupt sources will also need to be cleared with the block (GPIO, UART, …) or they will just re pend the interrupt. Entering the ISR will clear the pending bit for some interrupt sources.

Sample Firmware Source Code

The following is a C language example demonstrating the basic functionality of the Interrupt component. This example assumes the component has been placed in a design with the default name "ISR_1."

Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

```
#include <device.h>
void main()
{
        /* Dissable all interrupts by the processor. */
        CYGlobalIntDisable;

        /* Interrupt will be active with the Interrupt controller after this
function. */
        ISR_1_Start();

        /* Enable all interrupts by the processor. */
        CYGlobalIntEnable;

        /* Interrupt will be active at this point. */


    for(;;)
        ;
}
```

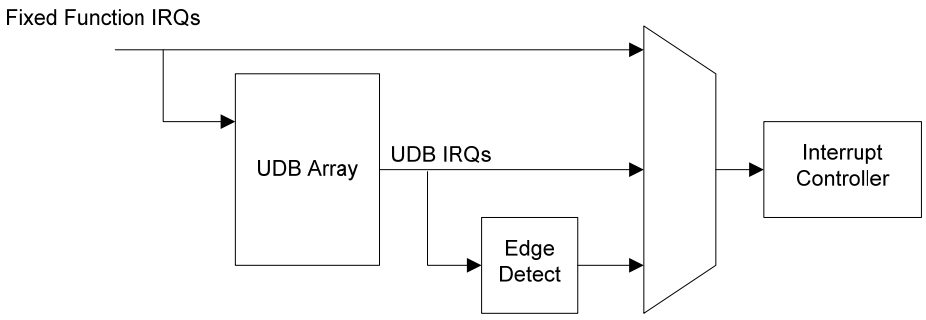
Interrupt Service Routine

The following is a C language example of the generated Interrupt Service Routine locations where the user should enter custom ISR code:

```
/*******************************************************************************
*  Place your includes, defines and code here
********************************************************************************/
/* `#START ISR_1_intc` */

/* `#END` */


CY_ISR(ISR_1_Interrupt)
{
    /*  Place your Interrupt code here. */
    /* `#START ISR_1_Interrupt` */

    /* `#END` */
}
```

Functional Description

Interrupt routing is flexible in the PSoC 3 programmable architecture. In addition to the fixed function peripherals, any data signal in the UDB array routing can also be used to generate an interrupt. A high-level view of the Interrupt mux (IDMUX) routing is shown in the figure below. The IDMUX selects from the available sources of interrupt requests.

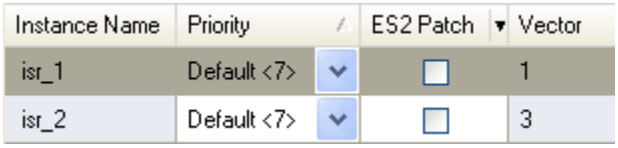


There are three types of system interrupt waveforms that can be processed by the interrupt controller:

- **Level –** IRQ source is sticky and remains active until firmware clears the source of the request with an action (e.g., clear on read). Most fixed function peripherals have level sensitive interrupts, including the UDB FIFO(s) and status registers

- **Pulse –** Ideally, a pulse IRQ is a single bus clock, which logs a pending action and insures that the ISR action will only be executed once. No firmware action to the peripheral is required.
- **Edge –** An arbitrary synchronous waveform is the input to an edge-detect circuit and the positive edge of that waveform becomes a synchronous one-cycle pulse (Pulse mode).

The use of an interrupt instance results in an entry in the Design-Wide Resource editor. The Interrupts Tab contains the following parameters:



The Priority entry may be updated from this editor.

DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data.

Interrupt AC/DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	Delay from Interrupt signal input to ISR code execution from main line code	Includes worse case completion of longest instruction DIV with 6 cycles	-	-	20	Tcy CPU
	Delay from Interrupt signal input to ISR code execution from ISR code	Includes worse case completion of longest instruction DIV with 6 cycles	-	-	20	Tcy CPU

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes
1.20	ES2 ISR patch.