# Using Interrupt Controller with MicroBlaze

## Overview of Interrupt Controller IP

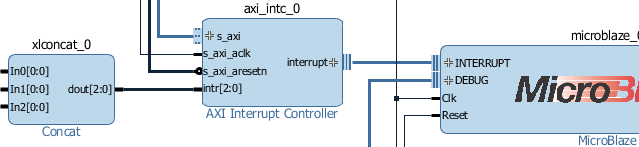
Some feature of Interrupt Controller is listed below:

* Supports up to 32 interrupts. It is cascadable to provide additional interrupt input.
* Priority between interrupt requests is determined by vector position. The LSB has the highest priority.
* It has Interrupt Enable Register for selectively enabling individual interrupts and Master Enable Register for enabling interrupt request output.
* Software interrupts and nested interrupts are supported.
* Each device connected to the AXI INTC core can use either normal or fast interrupt mode, based on the latency requirement. Fast interrupt mode can be chosen for designs requiring lower latency like Ethernet interrupts.
* Each input is configurable for edge or level sensitivity.

## Hardware Part

In this part adding a Interrupt Controller IP to a MicroBlaze project will be discussed.

* First click on “Add New IP” button and select “AXI Interrupt Controller”
* If you have more than one interrupt source(most probably you have), then Concat IP is needed. Select Concat IP and configure the number of inputs according to the number of interrupt sources.
* Connect dout[] port of the Concat to intr[] port of Interrupt Controller.
* Connect the interrupt port of the Interrupt Controller to INTERRUPT port of the MicroBlaze.
* Connect the interrupt sources to the inputs of Concat IP.



* Then “Run Connection Automaton”.
* Generate bitstream, export hardware and overwrite to existing hardware.
* Open SDK.

## Software Part

Following code shows how to use GPIO input interrupt with MicroBlaze.

|  |
| --- |
| **#include** "xparameters.h"  **#include** "xgpio.h"  **#include** "xil\_exception.h"  **#include** "xintc.h"  // Definitions for GPIO  **#define** GPIO\_DEVICE\_ID XPAR\_GPIO\_0\_DEVICE\_ID  **#define** BUTTON\_CHANNEL 1  //Definitions for GPIO interrupt  **#define** INTC\_GPIO\_INTERRUPT\_ID XPAR\_INTC\_0\_GPIO\_0\_VEC\_ID  **#define** BUTTON\_INTERRUPT XGPIO\_IR\_CH1\_MASK  // Definitions for Interrupt controller  **#define** INTC\_DEVICE\_ID XPAR\_INTC\_0\_DEVICE\_ID  //Function Prototypes  **void** **GpioHandler**(**void** \*CallbackRef);  **int** **main**(**void**)  {  XIntc IntcInstance; // Interrupt Controller Instance  XGpio GPIOInstance;// GPIO Instance  XGpio\_Initialize(&GPIOInstance, GPIO\_DEVICE\_ID);    XIntc\_Initialize(&IntcInstance, INTC\_DEVICE\_ID);  XIntc\_Connect(&IntcInstance, INTC\_GPIO\_INTERRUPT\_ID,(Xil\_ExceptionHandler)GpioHandler, &GPIOInstance);  XIntc\_Enable(&IntcInstance, INTC\_GPIO\_INTERRUPT\_ID);  XIntc\_Start(&IntcInstance, XIN\_REAL\_MODE);    XGpio\_InterruptEnable(&GPIOInstance, BUTTON\_INTERRUPT);  XGpio\_InterruptGlobalEnable(&GPIOInstance);    Xil\_ExceptionInit();  Xil\_ExceptionRegisterHandler(XIL\_EXCEPTION\_ID\_INT,(void \*)XIntc\_InterruptHandler, &GPIOInstance);  Xil\_ExceptionEnable();  **while**(1){  }  }  // Interrupt Function  **void** **GpioHandler**(**void** \*CallbackRef)  {  XGpio \*GpioPtr = (XGpio \*)CallbackRef;  /\*  \* USER CODES...  \*/  /\* Clear the Interrupt \*/  XGpio\_InterruptClear(GpioPtr, BUTTON\_INTERRUPT);  } |

* “” and “” libraries contain APIs, types and definitions for interrupt and exceptions.
* INTC\_DEVICE\_ID is the device ID of the Interrupt Controller IP.
* INTC\_GPIO\_INTERRUPT\_ID is the device ID of the GPIO interrupt.
* BUTTON\_INTERRUPT the interrupt mask for GPIO channel. Interrupt request can come one of the two channels. It determines which channels interrupt is used.
* “XIntc” is a struct type for interrupt controller instances.
* XIntc\_Initialize() function initializes the interrupt controller according to the INTC\_DEVICE\_ID.
* XIntc\_Connect() function makes the connection between the ID of the interrupt source and the associated handler that is to run when the interrupt is recognized. Callbackref is used as the argument for the handler when it is called.
* XIntc\_Enable () enables the interrupt source provided as the argument Id.
* XIntc\_Start()starts the interrupt controller by enabling the output from the controller to the processor.
* Xil\_ExceptionInit() initializes exception handling for the processor.
* XGpio\_InterruptEnable() enables interrupts.
* XGpio\_InterruptGlobalEnable() enables the interrupt output signal.
* Xil\_ExceptionRegisterHandler() makes the connection between the Id of the exception source and the associated handler that is to run when the exception is recognized.
* Xil\_ExceptionEnable() enables exceptions.