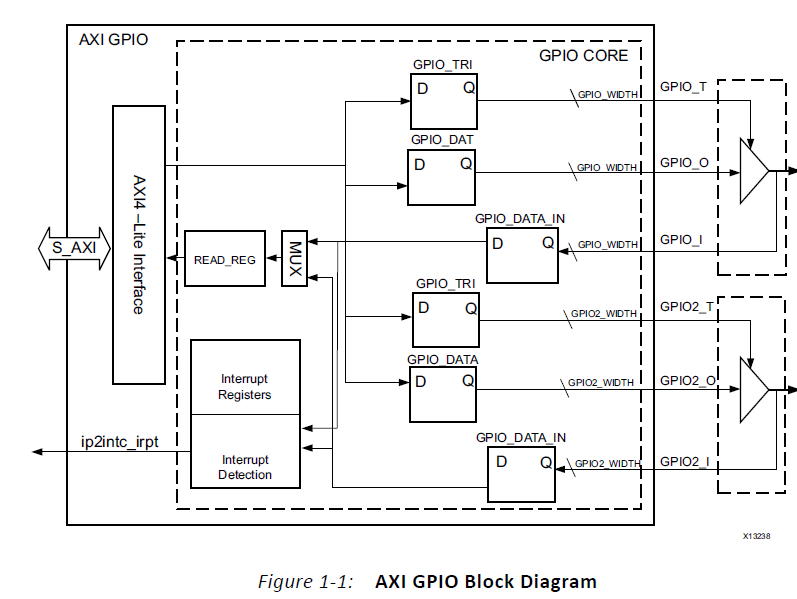
# Using GPIO With MicroBlaze

# Overview

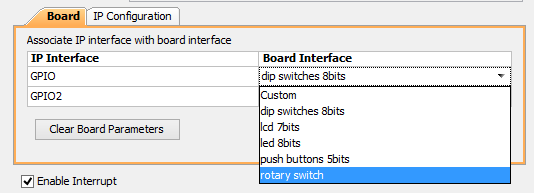
The AXI GPIO design provides a general-purpose input/output interface to an AXI4-Lite interface. The AXI GPIO can be configured as either a single or a dual-channel device.

Ports direction can be changed dynamically on software unless ALL\_INPUTS and ALL\_OUTPUTS configurations set to 0, and 3-state buffer is enabled for inputs and outputs. The width of each channel is independently configurable. The channels can be configured to generate an edge triggered interrupt.



Hardware Part

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

* First click on “Add New IP” button and select “AXI GPIO”
* Double Click on the IP block.
* 
* Select the peripherals you want to use. If you want to use interrupt do not remove the “Enable Interrupts” tick. There are two channel you can use. But it also can be used as a single channel.
* If you are not using the custom interface it not possible to set inputs as outputs and outputs as input in Vivado GUI.
* Click Ok.
* Click “Run Connection Automation”
* Generate bitstream, export hardware and overwrite to existing hardware.
* Open SDK.

Software Part

The code below shows how to configure GPIO ports as output or input and how to read data from I/O and write data to I/O. After following code is executed all LEDs on VC707 board will be the same as the DIP states.

|  |
| --- |
| **#include** "xstatus.h"  **#include** "xparameters.h"  **#include** "xgpio.h"  **#define** GPIO\_LED\_DEVICE\_ID XPAR\_GPIO\_0\_DEVICE\_ID  **#define** LED\_CHANNEL 1  **#define** DIP\_CHANNEL 2  XGpio Gpio; /\* The Instance of the GPIO Driver \*/  **int** **main**(**void**){  **int** Status;  **int** DIP\_State;  Status = XGpio\_Initialize(&Gpio, GPIO\_LED\_DEVICE\_ID);  **if** (Status != XST\_SUCCESS) {  **return** XST\_FAILURE;  }  XGpio\_SetDataDirection(&Gpio, DIP\_CHANNEL, 0xff);  XGpio\_SetDataDirection(&Gpio, LED\_CHANNEL, 0x00);  DIP\_State = XGpio\_DiscreteRead(&Gpio,DIP\_CHANNEL);  XGpio\_DiscreteWrite(&Gpio, LED\_CHANNEL, DIP\_State);  **return** XST\_SUCCESS;  } |

* “xstatus.h” library contains XST\_SUCCESS and XST\_FAILURE macros for status of the functions.
* “xgpio.h” library contains all APIs, types and definitions to use GPIO IP block.
* LED\_CHANNEL can be 1 or 2 according to the connection.
* DIP \_CHANNEL can be 1 or 2 according to the connection.
* “XGpio” is struct type definition for GPIO module, and it must be declared before calling any GPIO function, because all functions get this type as a parameter.
* XGpio\_Initialize() function initializes a GPIO instance according to GPIO\_LED\_DEVICE\_ID. All initialization functions return failure or success. This can be used when program does not work correctly, to understand where is the problem
* XGpio\_SetDataDirection() sets the direction of the data. Third parameter of the function is the mask. And it is specifying which bits are input and which are outputs. Bits set to 0 are output and bits set to 1 are inputs. In this case all LEDs are output and the DIP switches are input. If all bits are output or input there is no need to use this function.
* XGpio\_DiscreteRead() function return the state of the DIP switches.
* XGpio\_DiscreteWrite() function set the LEDs state as DIP states.