TP 0x01	Codesign		CHERIF Bilel
	PS Gpio, timer, and interrupts manipulation		

Windows :
Execute vivado using desktop shortcut
Linux :
Open terminal
Source the vivavdo setting64.sh file using the command
source /path to vivado/settings64.sh
Type vivado on terminal window

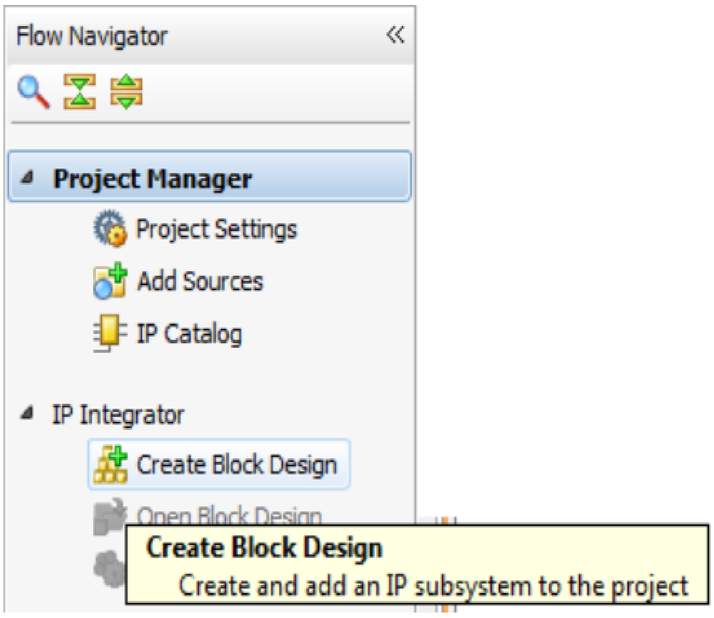
EX01 :

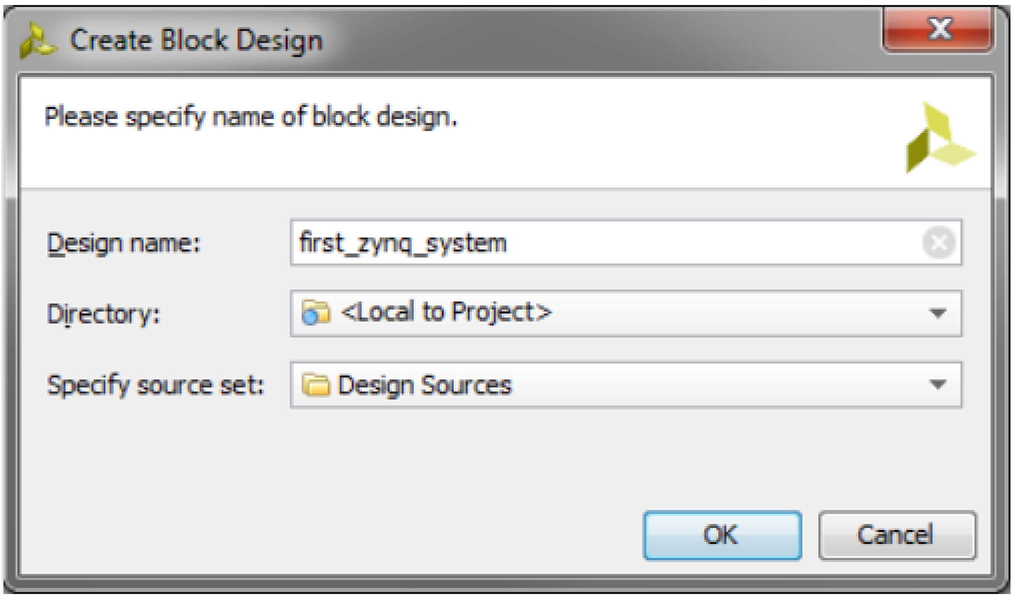
In this exercise we will use our ps gpios to blink a led as a first embedded software practice.

steps :

In the Flow Navigator window, select Create Block Design from the IP Integrator section

Enter a name for the design in the Design name box




Create Block Design

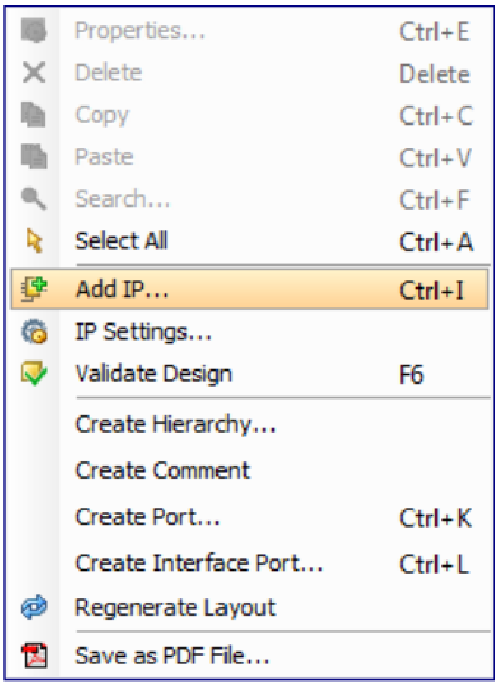
Please specify name of block design.

Design name: first_zynq_system

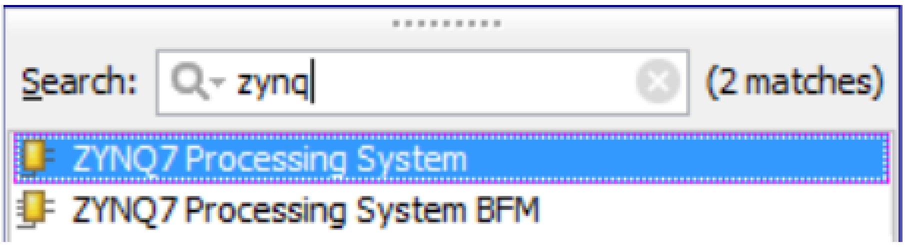
Directory: <Local to Project>

Specify source set: Design Sources

In the Vivado IP Integrator Diagram canvas, right-click anywhere and select Add IP



Enter zynq in the search field and select the ZYNQ7 Processing System

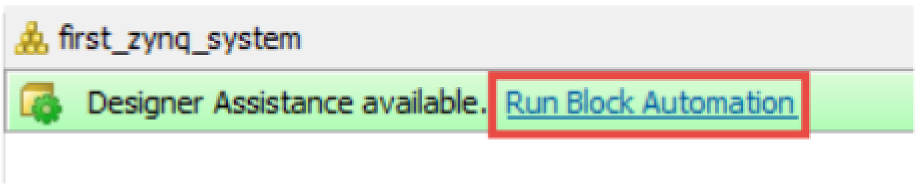

Search: zynq (2 matches)

ZYNQ7 Processing System

ZYNQ7 Processing System BFM

The next step is to connect the DDR and FIXED_IO interface ports on the Zynq PS to the top-level interface ports on the design.

Click the Run Block Automation option from the Designer Assistance message at the top of the Diagram window,

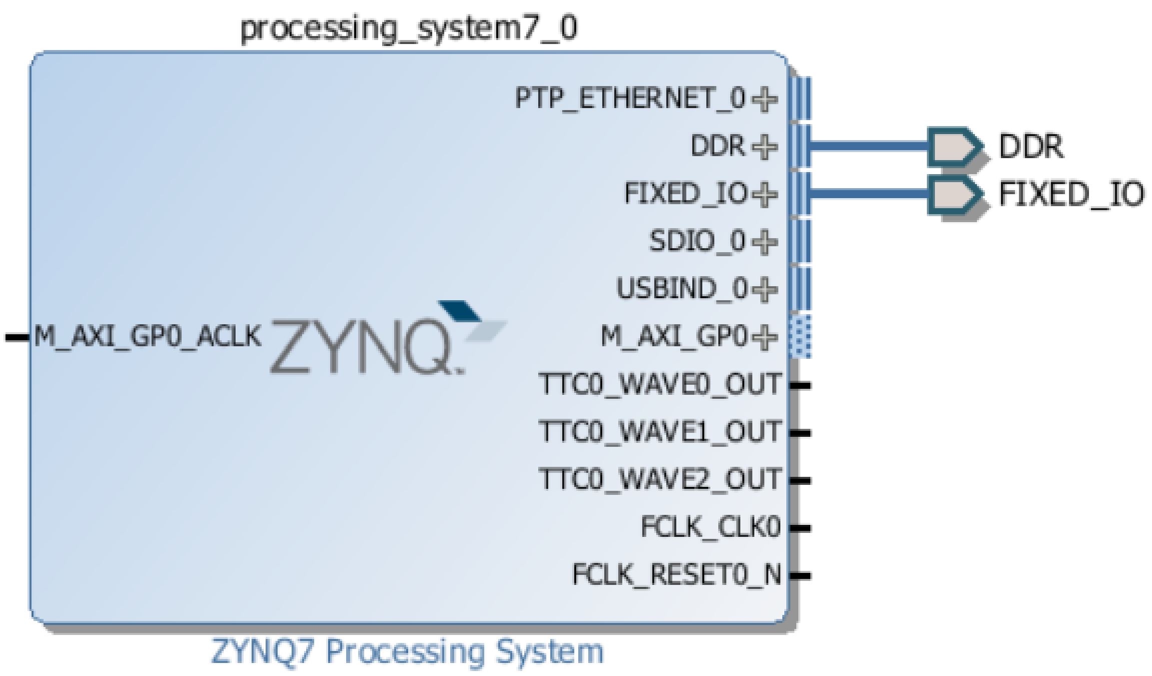


first_zynq_system

Designer Assistance available. Run Block Automation

ps: In
the Run Block Automation dialogue, ensure that the option to Apply Board Preset is selected and click OK. The external connections for both the DDR and FIXED_IO interfaces will now be generated.

Your block diagram should now resemble to the following figure:



- export the hardware and lunch the sdk
- Create a test software application to blink the mio 7 led.
- Configure the mio 50 as input, read the mio input and use it to toggle the led.

EX02 :

in this exercise we will use our global timer to blink our LED.

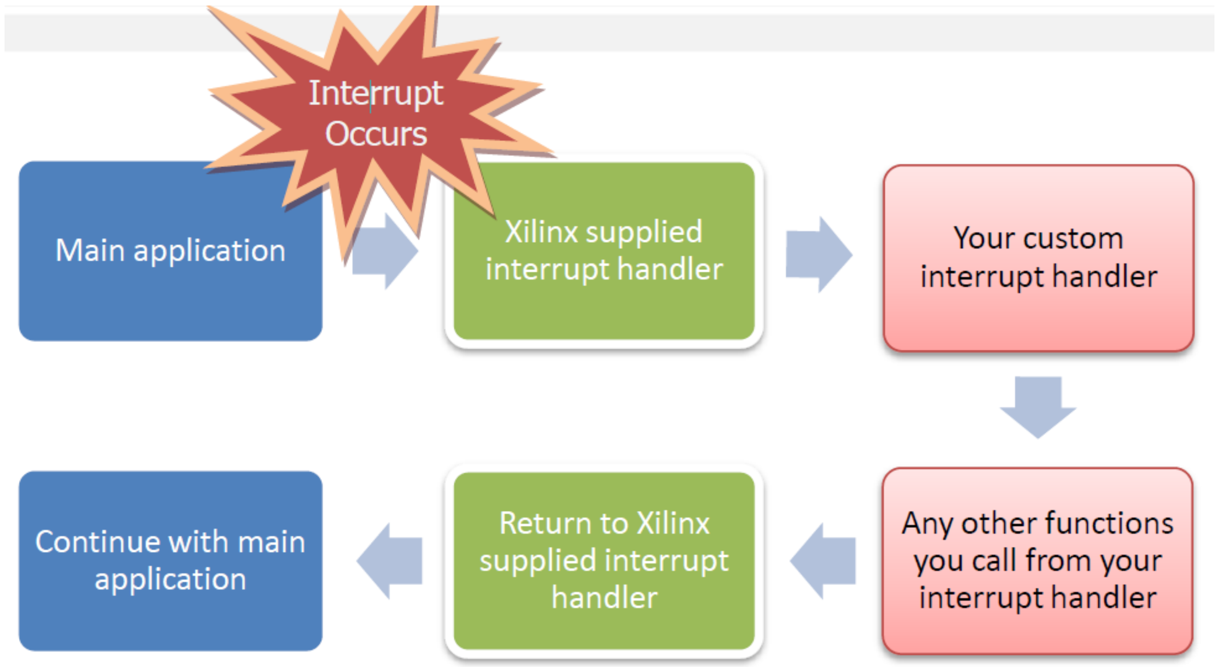
- include the global timer bare metal driver #include "xscutimer.h"
- Declare two structs for the timer instance and timer config

```
XScuTimer my_Timer;
XScuTimer_Config *Timer_Config;
```

- Use XscuTimer_LookupConfig as the one used for gpios to initialise the timer configuration instance.
- Use XscuTimer_CfgInitialize to initialize the timer instance.
- The global timer uses half of the CPU clock . Use the function XscuTimer_LoadTimer to load our counter with a value equivalent to 1 second.
- start the counter using the function XscuTimer_Start and use the function XscuTimer_GetCounterValue to check if our counter finished the counting to the desired value. When the count is finished toggle the led status and restart the counter using the function XscuTimer_RestartTimer.

EX03 :

In the ARM Cortex-A9 processor of the Zynq all interrupts are managed by and connected via the general interrupt controller.


Interrupt Occurs
Main application
Xilinx supplied interrupt handler
Your custom interrupt handler
Any other functions you call from your interrupt handler
Return to Xilinx supplied interrupt handler
Continue with main application

Interrupts can be generated by many different peripherals, but for this exercise we will be generating interrupts from the Snoop Control Unit Timer peripheral that we used in the previous exercise. In the last exercise we were operating the timer in polled mode, although the timer peripheral is in fact capable of producing interrupts.
The Zynq documentation states that an interrupt is generated by the timer whenever it reaches zero, so we shall use this feature to our advantage.In order to use interrupts, we need to use a few more driver functions to enable the interrupt functionality on the timer, to enable interrupts on the ARM processor, and also to set up the interrupt controller.
In this exercise we will use an interrupt that will indicate the end of our counting to drive the led blinking through this logic we will get rid of comparing our counting value to toggle the led output value.

- Create an instances of the general interrupt controller (GIC) and initialize it using the Lookup_Config and CfgInitialize functions.

- Follow the guide document to attach the timer generated interrupt to a handling function that will be executed by our CPU core by creating two functions the first to setup our interrupt and the second as a timer interrupt handler.

```
static void SetupInterruptSystem(XScuGic *GicInstancePtr,
          XScuTimer *TimerInstancePtr, u16 TimerIntrId);

static void TimerIntrHandler(void *CallBackRef)
```

- finally use the mio 50 and mio 51 buttons to increase and decrease the blinking frequency(blinking time interval should be for example at least 0.5s and the maximum is 5s).