LAB 4 : Interrupts with the AXI Timer

Introduction

This lab introduces the interrupt structure of the ARM processor using the AXI timer. Interrupts allow a processor to respond to externally generated events, temporarily halting its current execution and jumping to subroutine which is written to service the interrupt. Upon completion of the interrupt service routine, program control resumes from where the processor was interrupted.

As briefly mentioned in Lab 3, the AXI timer can be configured (either in Generate mode or Capture mode) to generate a pulse for one clock cycle on its external Interrupt port. This Interrupt port of the AXI timer will be connected to the Interrupt Request pin of the ARM processor using XPS. The ARM processor contains hardware (General Interrupt Controller) which is designed to assist the processor in responding to interrupts. Software will be written to configure the interrupt controller and to bind a user-written C++ function to the Interrupt Handler which will be executed every time the interrupt occurs.

Setting up the user Linux environment:

1. Type the following on a terminal:

    source /CMC/tools/xilinx_14.7/14.7/ISE_DS/settings64_CMC_central_license.csh

This sets up the Xilinx environment

Launching PlanAhead:

2. Launch the planAhead tool from your COEN317_LABS/Lab4 directory:

    planAhead &

Create a new project from PlanAhead:

3.1. File -> New Project

3.2. A window pops up, select Next
Give your project a name and choose the desired project path.
Project name: lab4
Project location: .../COEN317_LABS/Lab4

Leave the “Create project subdirectory” checked

Select Next

3.3. RTL Project should be selected. Keep the selection.

Select Next

3.4 We do not need to specify any sources, select VHDL as the target language in the drop down box:

Target language: VHDL
Select Next

3.5. We do not need to specify any IPs. Select Next.

3.6. We do not need to specify any UCF files for this part. Select Next.

3.7. Select the ZYNQ-7 ZC702 Evaluation Board for the project:

Select Boards
Scroll down the list and choose ZYNQ-7 ZC702 Evaluation Board
Select Next

3.8. Review settings and Select Finish

Create an embedded processor project with the Add Source wizard:

4.1. On the left panel under Project Manager, select Add Sources.

4.2. Choose Add or Create Embedded Sources and select Next

4.3. Select Create Sub-design
A window will open prompting to enter a module name. Name it “system”
Module name: system
Select OK
Select Finish

Designing the system in XPS:

XPS will be used to design the system consisting of an AXI timer and its associated AXI interconnect. The Interrupt port of the timer will be connected to the processor’s interrupt request pin.

5.1. A window pops up and asks:
This project appears to be a blank zynq project.
Do you want to create a Base System using the BSB Wizard?

Select Yes

5.2. The BSB Wizard asks for additional settings
The AXI System should be selected by default with no other parameters.

Select OK

Verify the Zynq Processing System 7 is selected in “Select a System”
Select Next

Remove the GPIO_SW and LEDs_4Bits Peripherals because they are not needed.
Choose “Select All” , then “ < Remove”

Do **not** select Finish, yet.

5.3 Enable the axi_timer interrupt from the Peripheral Configuration window.

Select the axi_timer from the Internal Peripherals and select the Add> button to add an instance of an axi_timer to the processing system. Refer to Figure 1.

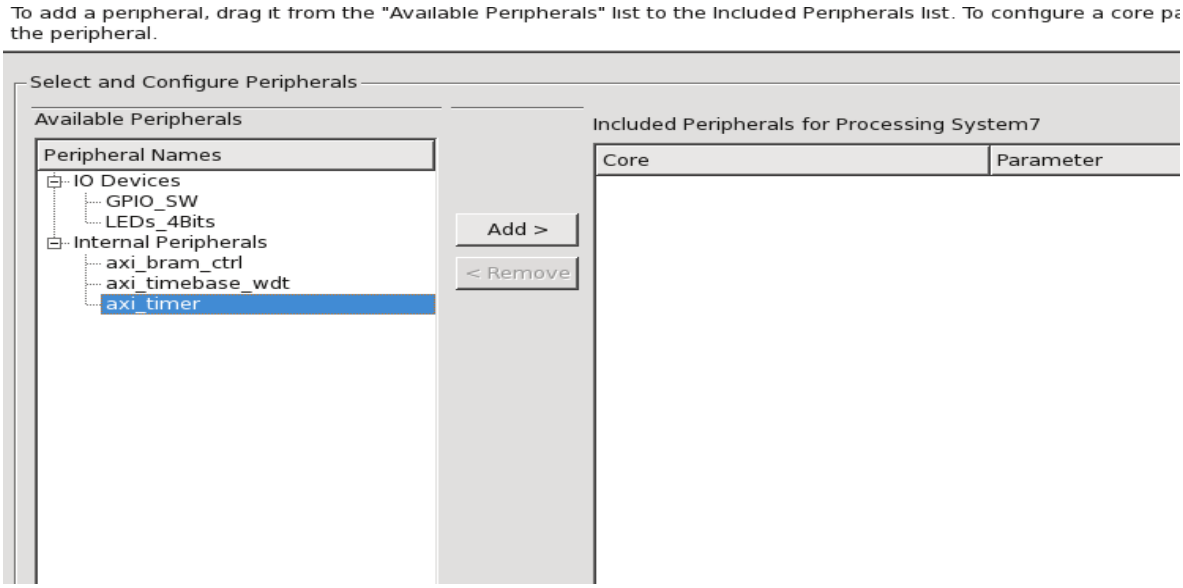

Figure 1: Including an axi_timer peripheral.

Once the timer has been added to the Included Peripherals, select the Use Interrupt checkbox to enable the timer interrupt. Refer to Figure 2.

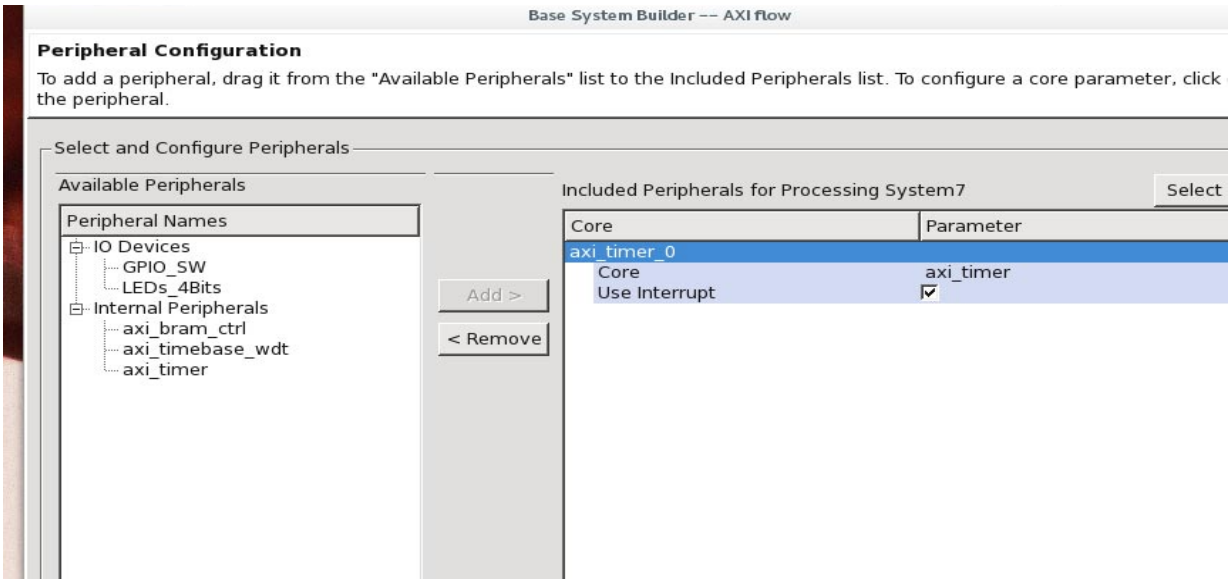

Figure 2: Enabling the axi_timer interrupt.

NOTE: There is no direct option in the BSB to resize the counter bit width. However, once it has been added as a peripheral from the BSB, once can double-cliock in XPS on the AXI timer IP and specify the desired counter bit width.

5.4 Verify the interrupt connection in the Graphical Design View.

To verify that the timer's interrupt output line has been connected to the processor's IRQ_F2P input, select the Grphical Design View in the XPS window. Refer to Figure 3.

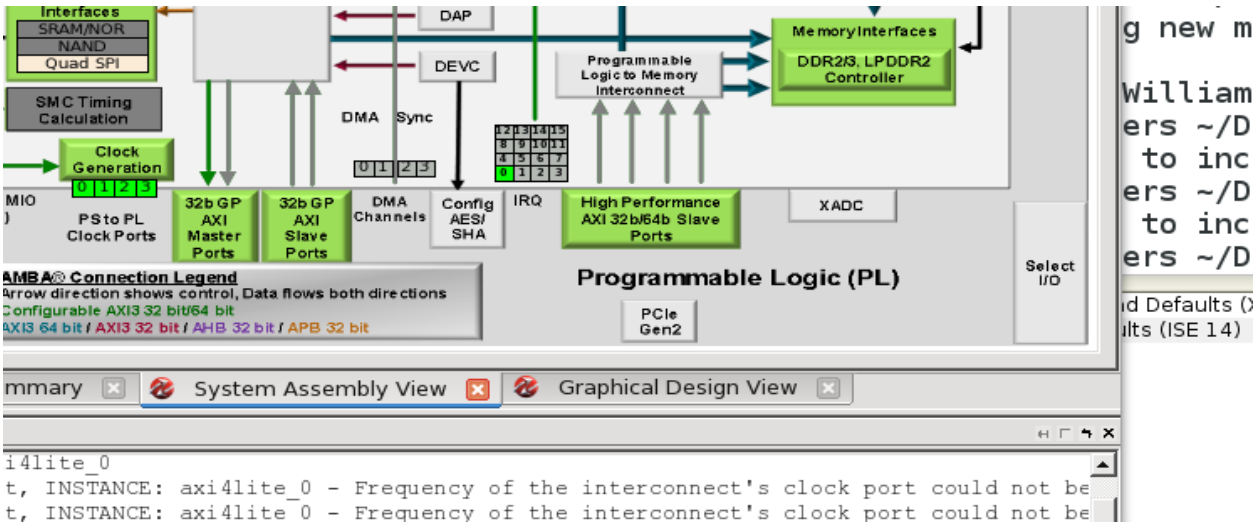

Figure 3: Selecting the Graphical Design View.

The view will change to a graphical representation of the system components. Note that the default settings do **not** show any interrupt ports. Select the 'Interrupt Port" checkbox in the "Show Ports" of the 'Standard Filters" . The interrupt connection from the AXI timer to the IRQ_F2P will be now shown with a wire connection. Refer to Figure 4.

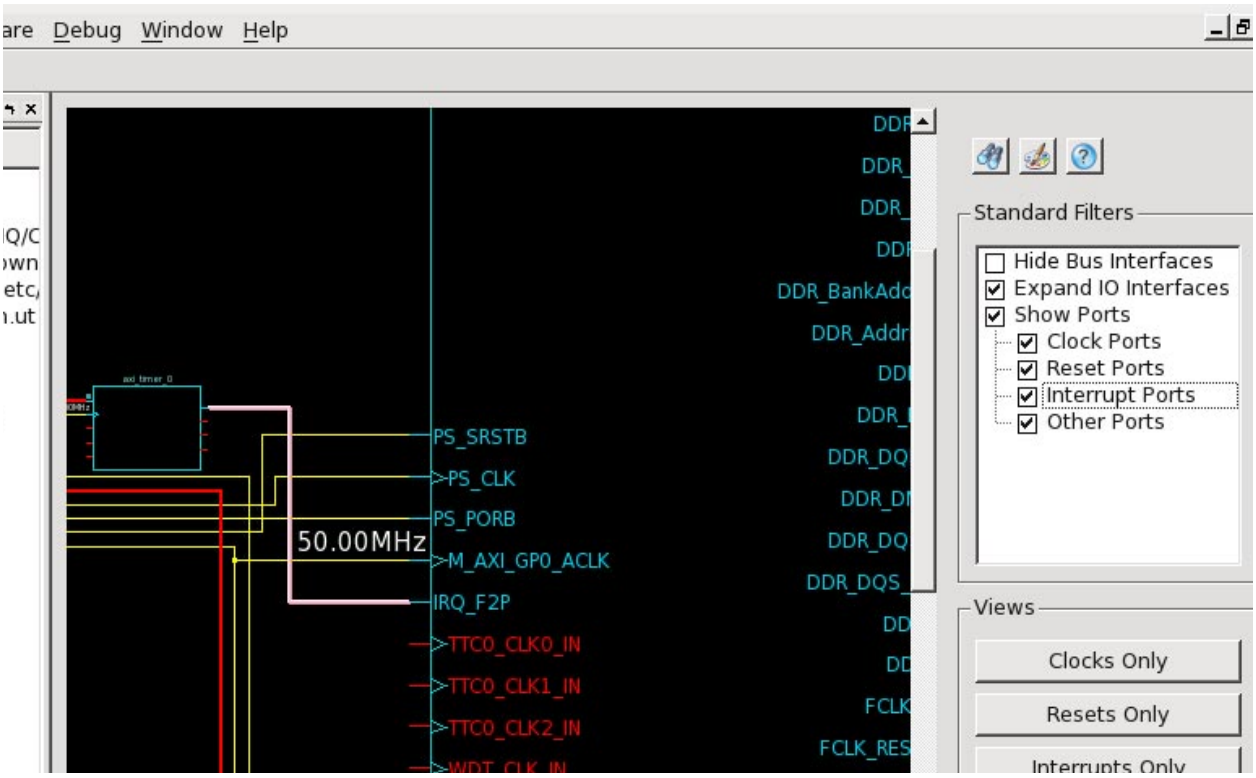

Figure 4: Graphical Design view indicating the interrupt connection.

5.5. Verifying the interrupt connection through the Connected Ports.

One can also verify that the interrupt connection has been properly made through the Port view of the Zynq system assembly view. Switch from the Graphical Design view back to the System Assembly View (the view with the green components) and select the Ports tab at the top portion and expand the "processing_system7_0". Verify that in the Connected Ports column the IRQ_F2P input of the prcoessor is indicated as being connected to "L to H: axi_timer' as shown in Figure 5.

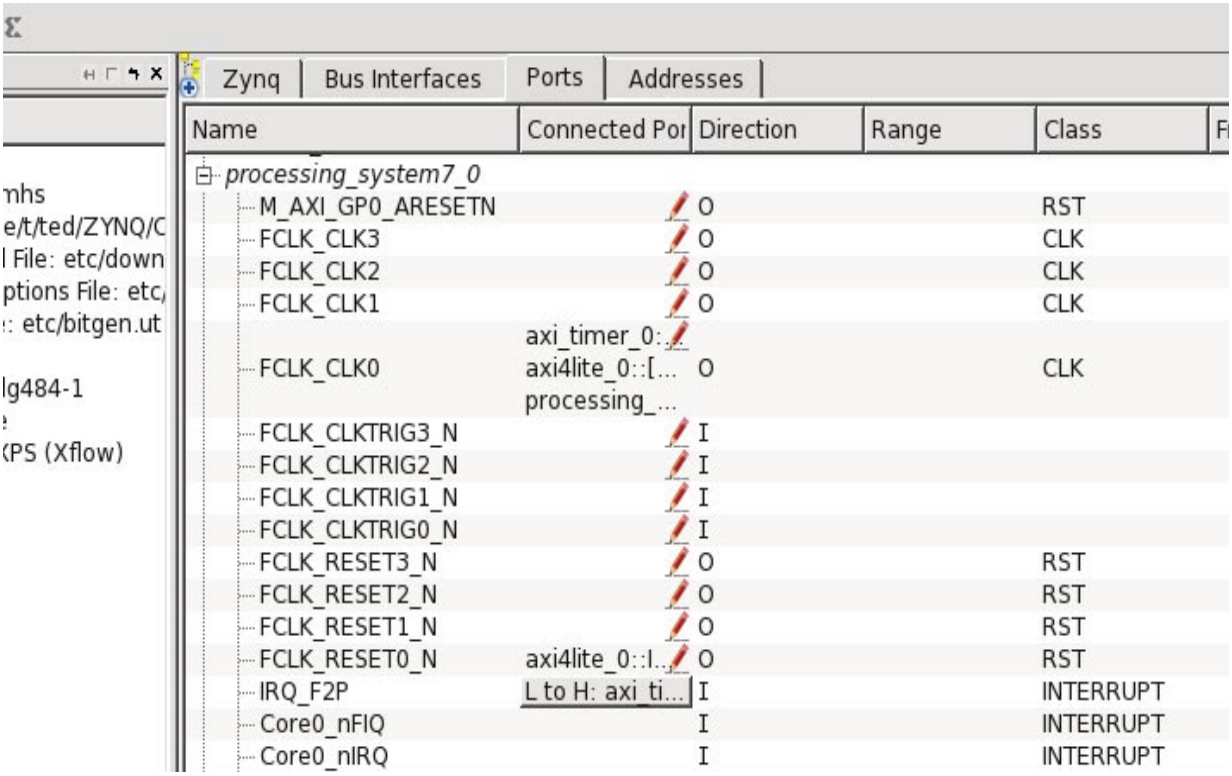

Figure 5: IRQ_F2P port connection.

Once you have verified the timer interrupt connexction, collapse the processing_system7_0 instance.

5.6. Close the XPS tool.
File -> Exit
Closing the XPS moves PlanAhead back in the foreground.

Exporting the Hardware to SDK:

6.1. Right-click on “system (system.xmp)” and choose Create Top HDL
We do not need to modify the stub,.

6.2. Synthesize the VHDL code
Click on Run Synthesis on the left Panel
Once Synthesis has finished, click Open Synthesized Design
There may be 3 constraint warnings, they do not affect our system and you can click OK

6.3. Implement the Design
Click Run Implementation.

6.4. Generate the bitstream.

6.5 Click on Bitstream Settings (On the bottom LHS)
A window should have appeared

In the More Options, type in:
-g UnconstrainedPins:Allow

If t he PlanAhead GUI does not detect any change (Apply button is greyed out) check and uncheck -d option for PlanAhead to detect a change

Click Apply
Click OK

6.6. Select Generate Bitstream
Wait for the bitstream generation to finish

6.7 Download the bitstream to the Zynq Board

Attach the power cable, the Platform Cable USB II, and the serial UART cable. Apply power the board verify the Platform Cable USB II status LED is green. Ensure the board DIP switch settings are correct if the LED is not green. Choose Launch iMPACT and select OK.
A window may pop up asking you to save the file, click Cancel.
In iMPACT, right-click the Target and choose Program
Make sure Device 2 (FPGA xc7z020) is highlighted and click OK

When you see Program Succeeded, close iMPACT
File -> Exit
You do not need save the iMPACT project
Click No

6.8 Export the hardware:

Go back to the PlanAhead tool and select:

File -> Export -> Export Hardware for SDK
Check “Launch SDK” and click OK on the pop up screen
SDK will open.

Using SDK to create a new application project:

7.1 From the main SDK window select:

File -> New -> Application Project
Name the project: lab4
Select Continue

7.2 Copy the file /home/t/ted/PUBLIC/COEN317/Lab4/main.cc into your .../COEN317_LABS/Lab4/lab4/lab4/lab4.sdk/SDK/SDK_Export/lab4/src directory.

The program performs the following:

• it declares as global variable an instance of an Interrupt Controller (the variable called `InterruptController` which is of data type `XScuGic`.

• it declares the AXI timer as variable `TimerInstancePtr` of data type `XTmrCtr`;

• it defines the interupt handler as the following function:

```
void Timer_InterruptHandler(void)
{
        cout << "I am the interrupt handler" << endl;
}
```

This is the function which will be invoked every time the timer generates an interrupt.

• It defines two functions related to the processor's interrupt controller:

```
int SetUpInterruptSystem(XScuGic *XScuGicInstancePtr)

int ScuGicInterrupt_Init(u16 DeviceId,XTmrCtr *TimerInstancePtr)
```

Within function `main()`, the following is performed:

• In Step 1, The Xilinx provided function `XTmrCtr_Initialize(&TimerInstancePtr, XPAR_AXI_TIMER_0_DEVICE_ID)` is used to initialize the instance associated with the AXI Timer. This is required since this function initializes some values in aninternal table which are subsequetnly referred to by function `XTmrCtr_SetHandler()` in Step 2.

• Step 2 performs the binding of our user defined function `Timer_InterruptHandler()` as the interrupt service routine associated with the AXI timer generating an interrupt.

• In Step 3, we use the pointer deference method to load a reset value of `0xffffff00` into the timer's load register. Note we we have first declared and initialized a pointer to an `unsigned int` with the base address of the AXI timer (with a symbol defined in the xparameters.h file). Every time the counter rolls over from all '1' to all '0's it will generate a pulse on its Interrupt port for one clock cycle.

• Step 4 configures the timer to function in Generate mode, as an Up counter, with interrupt enabled and autoreload of load register. Refer to the AXI Timer datasheet to understand why the hexadecimal value of 0x0f4 was used to write into the timer status register.

• Step 5 performs the required initialization of the SCUGIC interrupt controller by invoking the function `ScuGicInterrupt_Init()` which as part of it's execution invokes functions `SetUpInterruptSystem()` and `XScuGic_Connect()`.

• After these initialization steps, in order to allow the user to control when the timer starts, the program prompt the user to enter any key to continue and once a value has been entered, the timer is started and the program enters an infinite while loop in which it does nothing but wait for the timer to generate the periodic interrupts.

7.3 Connect the Terminal (and set the serial settings as required).

7.4. Download and run the program on the board. Observe the program output in the terminal window. Investigate the behaviour of using different values for the counter load register reset value on how frequently the interrupt routine is invoked.

Questions

1. The AXI timer contains two timer/counters but only has a single interrupt signal. The timer datasheet states "The interrupt signal is the OR of the interrupts from the two counters. The interrupt service routine must poll the TCSRs (timer/counter status registers) to determine the source or sources of the interrupt. The interrupt status bit (TINT in the TCSR) can only be cleared by writing a '1' to it. Writing a '0' to it has no effect on the bit. Because the interrupt condition is an edge (the counter rollover or the capture event), it can be cleared at any time and does not indicate an interrupt condition until the next interrupt event." [1] Create a new project which enables both of the timers to generate periodic interrupts. Modify the `Timer_InterruptHandler()` function so that it:

• first stops both timers

• reads the values of both status registers to determine which of the two timers generated the interrupt and displays a message on the terminal indicating which timer generated the interrupt.

• it clears the TINT bit of the timer which caused the interrupt by writing a '1' to bit 8 of the appropriate status register.

• prompts the user to enter any value and upon the user entering a value both timers are started.

References

1. LogiCORE IP AXI Timer Product Specification, DS764, Xilinx, July 25, 2012, p5.

T. Obuchowicz/R. Lee
July 2014

Revised Oct. 20, 2016: interupt connection made with BSB for v14.7. Revised for new Lab 4.