Introduction

This tutorial will guide you through the process of creating a Zynq design utilising interrupts. Using the Vivado™ Integraded Development Environment (IDE) and the IP Integrator environment, a simple Zynq™ processor design, to be implemented on the ZedBoard, will be generated. The Software Development Kit (SDK) will then be used to create a simple software application which will run on the Zynq's ARM Processing System (PS) to control the hardware that is implemented in the Programmable Logic (PL). This tutorial leads on from the previous one, expanding on the skills acquired in it.

The tutorial is split into four exercises, and is organised as follows:

Exercise 2A — This exercise provides a further guide to the process of launching Vivado IDE and creating a project using *New Project Wizard*

Exercise 2B — In this exercise, we will use the project that was created in Exercise 2A to build a Zynq embedded system utilising interrupts with IP Integrator and incorporating existing IP from the Vivado IP Catalog. This will expand on previous knowledge gained in creating and connecting a block based system in IP Integrator. The completed design will have an associated bitstream generated and will be exported to the Xilinx SDK for creating of a test application.

Exercise 2C — In the Xilinx SDK, a test software application for the generated hardware system will be created and explained. Running this application on the ZedBoard will demonstrate the function of interrupts and how the application is coded to utilise them.

Exercise 2D — Finally, we will return to the system from Exercise 2B and include an additional source of interrupt, making the necessary connections, and generating a bitstream and exporting to the Xilinx SDK. We will then modify our previous software application to inspect the operation of the altered system.

THE ZYNQ BOOK TUTORIALS

Louise H. Crockett
Ross A. Elliot
Martin A. Enderwitz
Robert W. Stewart
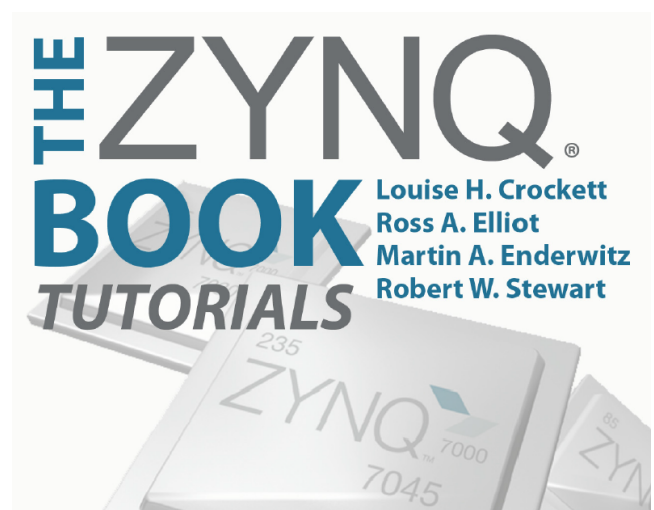
Exercise 2A Expanding the Basic IP Integrator Design

In this exercise we will expand upon the previous project in Vivado IDE by adding additional GPIO and configuring the system to utilise interrupts. For the sake of clarity and understanding, we will run through the building of a basic system once more. Start by launching the Vivado IDE.

(a) Launch Vivado by double-clicking on the Vivado desktop icon: , or by navigating to **Start > All Programs > Xilinx Design Tools > Vivado 2014.1 > Vivado 2014.1**

(b) When Vivado loads, you will be presented with the *Getting Started* screen as in Figure 2.1.

Figure 2.1: Vivado IDE Getting Started screen

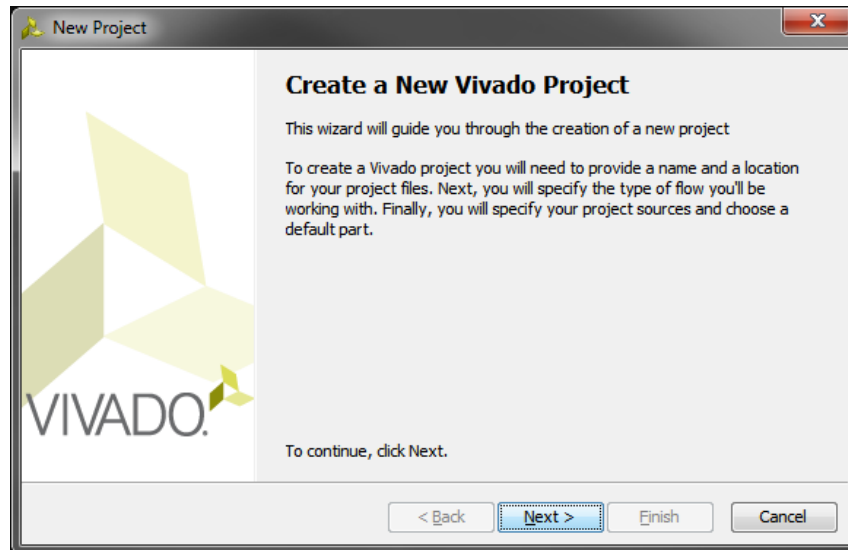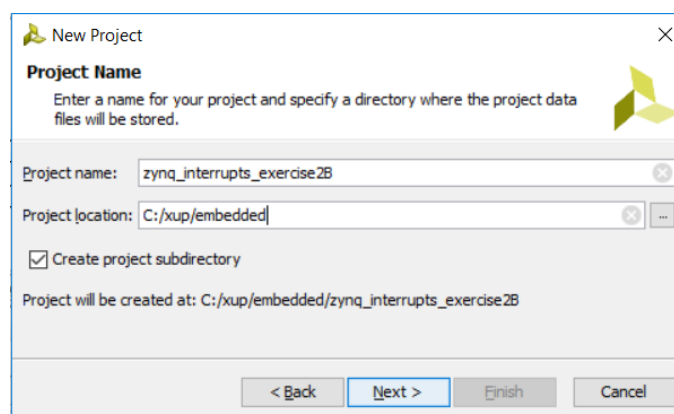(c) Select the option to **Create New Project** as in Figure 2.2


Figure 2.2: New Project dialogue

Click **Next**.

(d) At the Project Name dialogue, enter **zynq_interrupts_exercise2B** as the **Project name** and **C:/xup/embedded as Project location. Make sure that you select the option to Create project subdirectory. All options should be the same as shown below:**

Click **Next**. A directory will be created on your **C drive** if it did not already exist.

(e) At the *Project Type* dialogue, select **RTL Project** and ensure that the option **Do not specify ources a*t this time is not selected:**
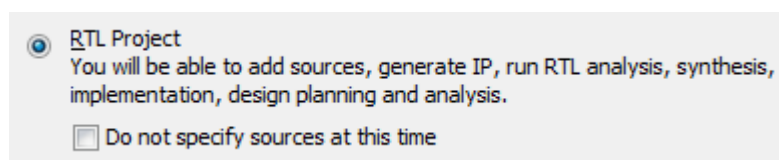
Click **Next**.

(f) Select **Verilog** as the **Target language** in the *Add Sources* dialogue.

If existing sources, in the form of HDL or netlist files, were to be added to the project they

could be imported at this stage.

As we do not have any sources to add to the project, click **Next**.

(g) The *Add Existing IP (optional)* dialogue will open.

If existing IP sources were to be included in the project, they could be added here.

As we do not have any existing IP to add, click **Next**.

(h) The *Add Constraints (optional)* dialogue will open.

This is the stage where any physical or timing constraints files could be added to the project.

As we do not have any constraints files to add, click **Next**.

(i) From the *Default Part* dialogue, select **Boards** from the *Specify* box and choose **Zybo Zynq Evaluation and Development Kit**, Board Version **c** from the list of boards, as shown in Figure 2.3.
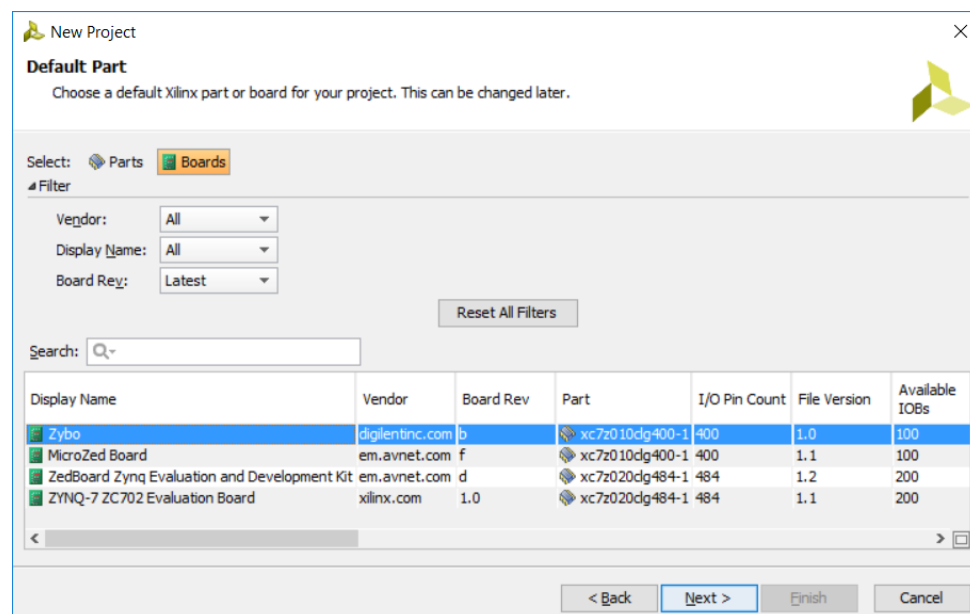
Figure 2.3: Default part dialogue options

Click **Next**.

(j) In the *New Project Summary* dialogue, review the specified options, and click **Finish** to create the project.

As in the previous tutorial we will now create the basic Zynq embedded system design before adding and configuring additional IP to utilise hardware interrupts.

Exercise 2B Creating a Zynq System with Interrupts in Vivado

In this exercise we will create a simple Zynq embedded system which implements two General Purpose Input/Output (GPIO) controllers in the PL of the Zynq device on the ZedBoard, one of which uses the push buttons to generate interrupts. The other GPIO controller will connect to the LEDs. Both will also be connected to the Zynq processor via an AXI bus connection, allowing the LEDs to be controlled by a software application which we will create in Exercise 2C.
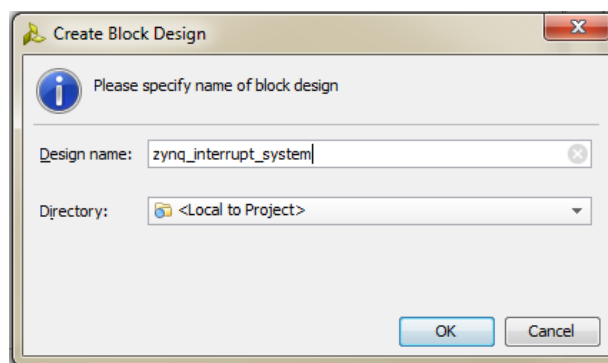
(a) In the *Flow Navigator* window, select **Create Block Design** from the *IP Integrator* section, as in Figure 2.4:
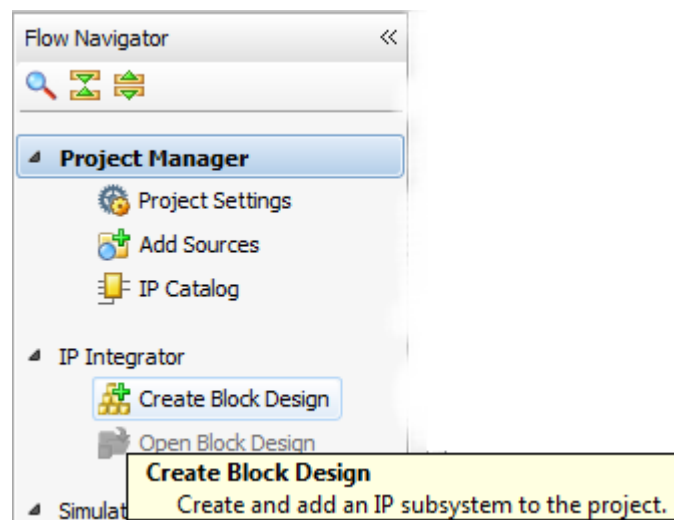
Figure 2.4: Creating a new Block Design in Flow Navigator

The *Create Block Design* dialogue will open.

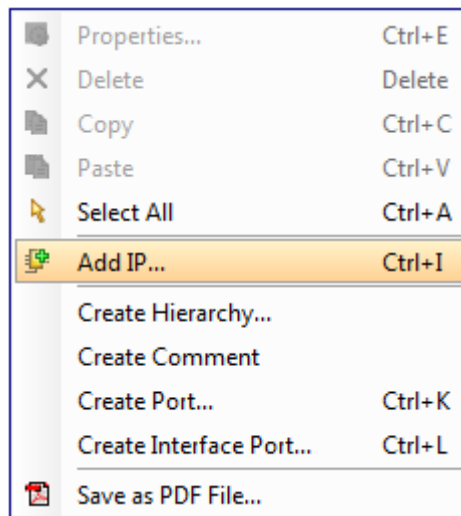(b) Enter ***zynq_interrupt_system*** in the Design name box, as in Figure 2.5:

Figure 2.5: Create Block Design dialogue

Click **OK**. The *Vivado IP Integrator Diagram* canvas will open in the *Workspace*.

The first block that we will add to our design will be a Zynq Processing System.

(c) In the *Vivado IP Integrator Diagram* canvas, right-click anywhere and select **Add IP**, as in Figure 2.6.

Figure 2.6: Add IP option

Alternatively, select the **Add IP** option from the information message at the top of the canvas, shown in Figure 2.7.
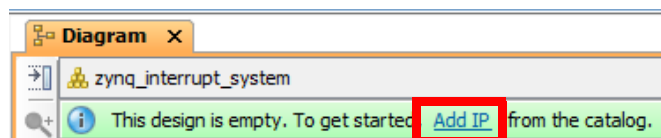
Figure 2.7: Add IP option in IP Integrator canvas information message

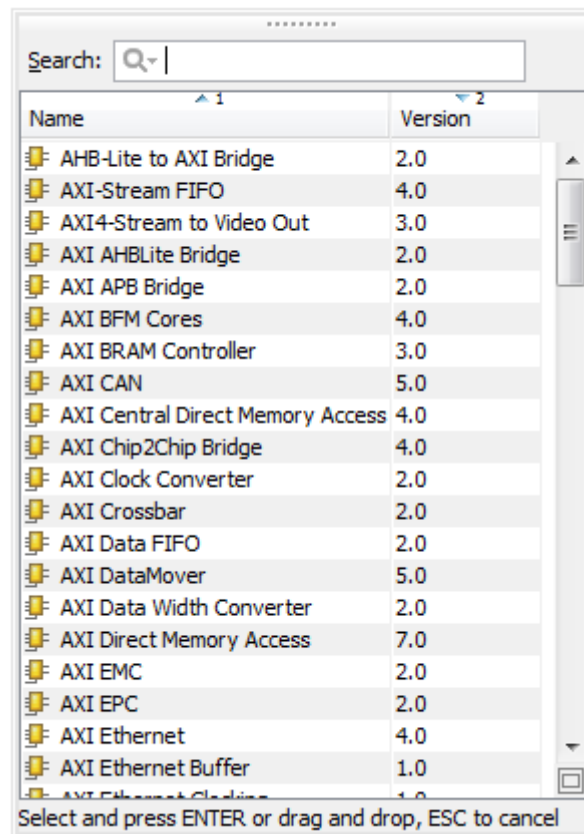The pop-up IP Catalog window will open, as in Figure 2.8.

Figure 2.8: Pop-up IP Catalog window

(d) Enter **zynq** in the search field and select the **ZYNQ7 Processing System**, ensuring that you select the option for *Version 5.4*, as shown in Figure 2.9, and press the **Enter** key on your keyboard.
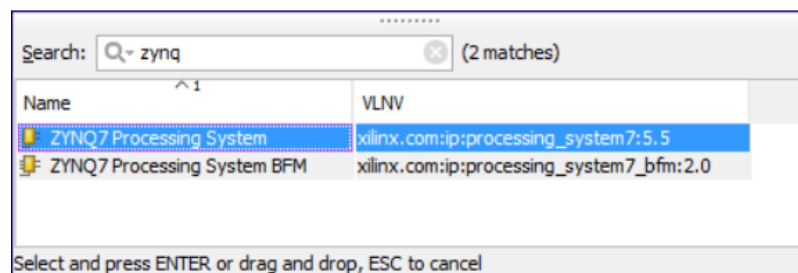

Figure 2.9: Adding ZYNQ7 Processing System from IP Catalog

As in the previous tutorial, 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.

(e) Select the **Run Block Automation** option from the *Designer Assistance* message at the top of the Diagram window. Select **OK**, ensuring that the option to **Apply Board Preset** is selected, to generate the external connections for both the **DDR** and **FIXED_IO** interfaces, and apply the relevant board presets.

Your block diagram should now resemble Figure 2.10.
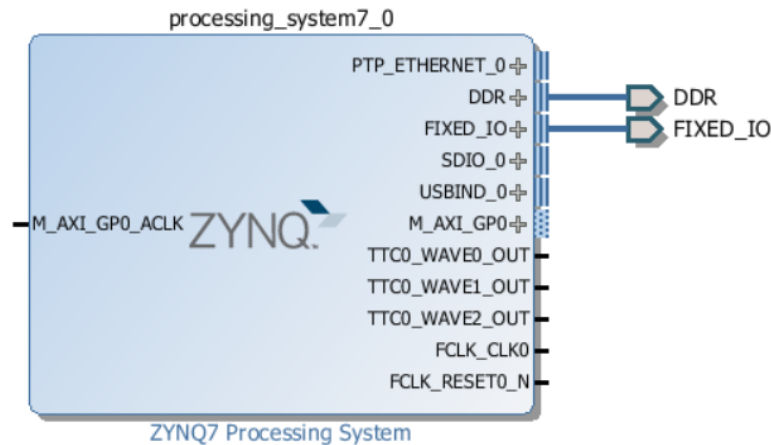
Figure 2.10: ZYNQ7 Processing System external connections

Now that the main Zynq PS has been added to our design and configured, we can now add further blocks which will be placed in the PL to add functionality to the system. In this case we require an **AXI GPIO** block for the **LEDs** and another for the **push buttons**.

(f)   Right-click in an empty area of the *Diagram* window and select **Add IP**. Enter **GPIO** in the search field and add an instance of the **AXI GPIO** IP. Repeat this procedure to add a second **AXI GPIO** block to the design.

We will now use the *IP Integrator Designer Assistance* tool to automate the connection of the **AXI GPIO** blocks to the **ZYNQ7 Processing System**.

(g) Click **Run Connection Automation** from the *Designer Assistance* message at the top of the *Diagram* window and select **S_AXI in Run Connection Automation**, as shown Figure 2.11.
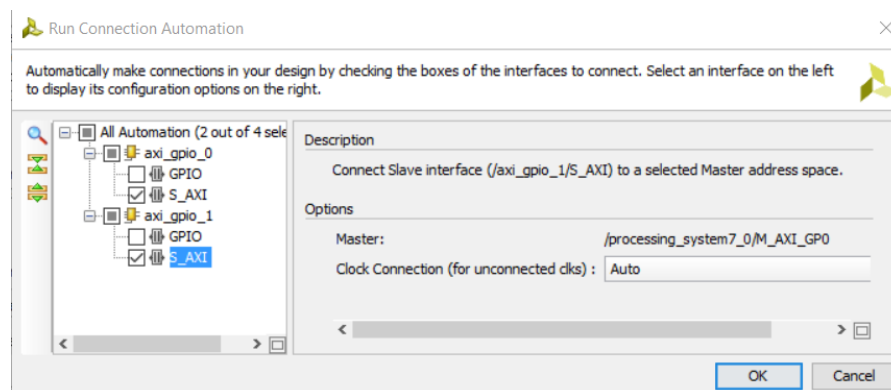
Figure 2.11: Run Block Automation - GPIOinstance 1

Click **OK** to ensure automatic clock connection, which adds the **Processor System Reset Module** and the **AXI Interconnect** blocks.

(h) Click **Run Connection Automation** from the *Designer Automation* message at the top of the *Diagram* window and select **/axi_gpio_0/GPIO**.

The Run Connection Automation dialogue will open, as in Figure 2.12. Select **btns_5bits** from the drop-down menu, and click **OK**.
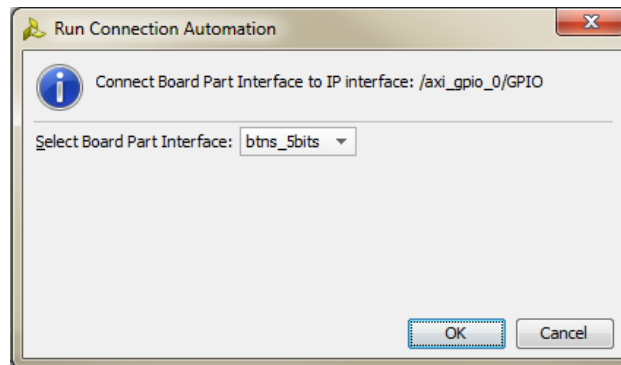

Figure 2.12: Run Connection Automation dialogue — GPIO

(i) Repeat steps **(g)** and **(h)** for the second **GPIO** block, this time selecting **leds_4bits** for **/ axi_gpio_1/GPIO**.
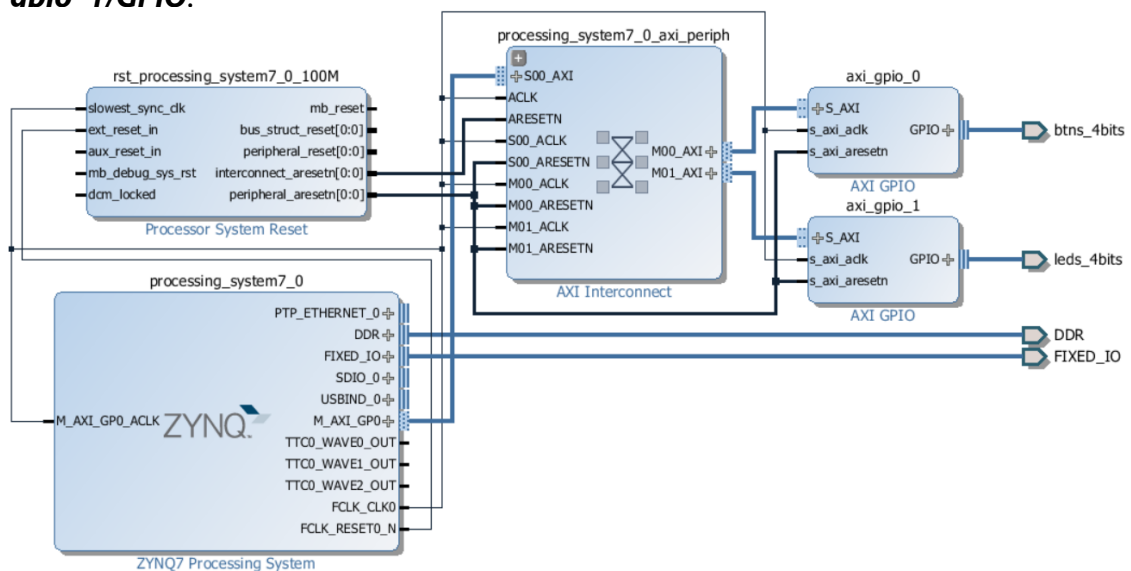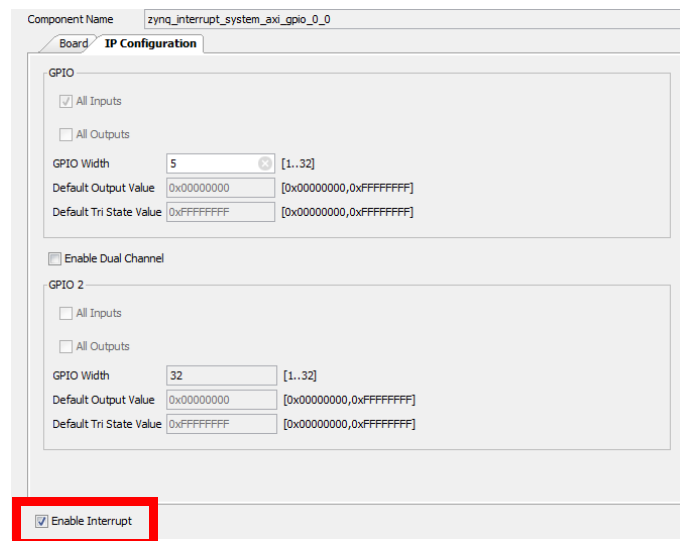

Figure 2.13: Zynq processor system

You will now have a system that is similar to Figure 2.13. We now need to configure the system to utilise hardware interrupts from the **push buttons** to trigger functions in the **Zynq PS**.

(j)   Double--click on the **GPIO** block connected to the **push buttons**, *axi_gpio_0*, to open the *Re-customize IP* window,.

Figure 2.14: Enabling GPIO interrupts

Click the **IP Configuration** tab and enable interrupts from the **push buttons** by clicking in the box highlighted in Figure 2.14 and click *OK.* This will add an additional output port for the interrupt request to the **GPIO** block as in Figure 2.15.
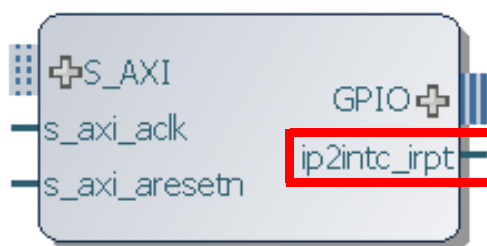
Figure 2.15: GPIO block with interrupt port

Now we must configure the **Zynq PS** to accept interrupt requests.

(k)   Double-click on the **Zynq PS** block, ***processing_system7_0,*** to open the *Re-Customize IP* window.

(l)   Select *Interrupts* from the *Page Navigator* on the left-hand side and expand the menu on the right as in Figure 2.16. Since we want to allow interrupts from the **programmable logic** to the **processing system,** tick the box to enable **Fabric Interrupts**, then click to enable the shared interrupt port as in Figure 2.16. This means interrupts from the **PL** can be connected to the interrupt controller within the **Zynq PS**. Click *OK.*
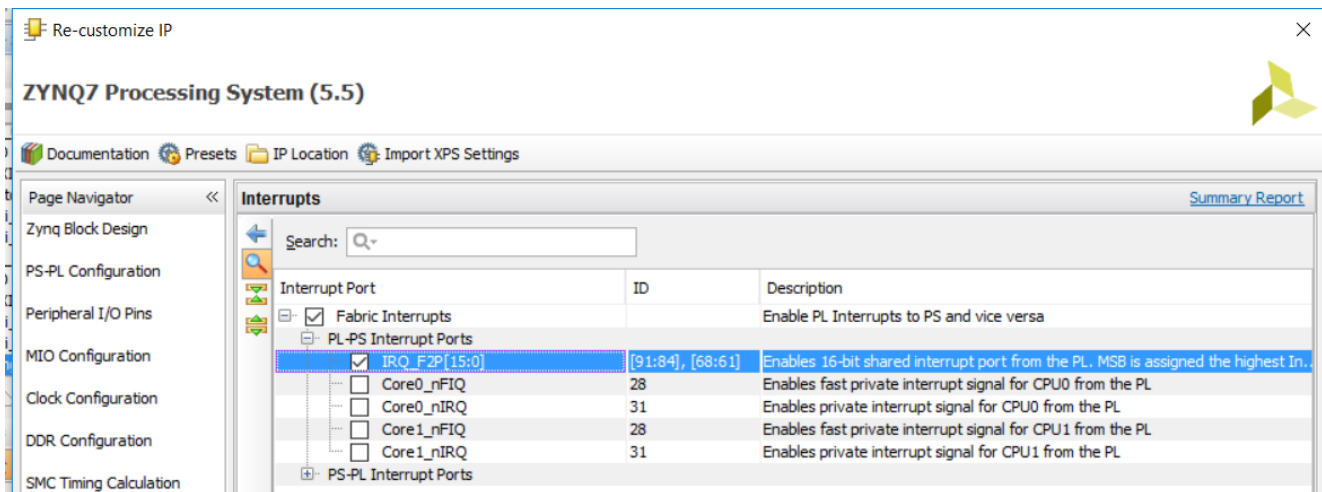
Figure 2.16: Configuring Zynq PS to utilise interrupts

(m) Make a connection between the interrupt request of the **GPIO** block and the newly created interrupt port of the **Zynq PS**, highlighted in Figure 2.17.
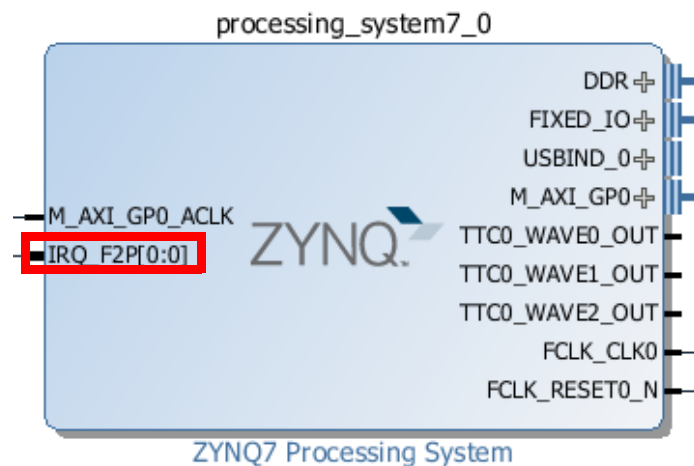
Figure 2.17: Zynq PS with interrupt port

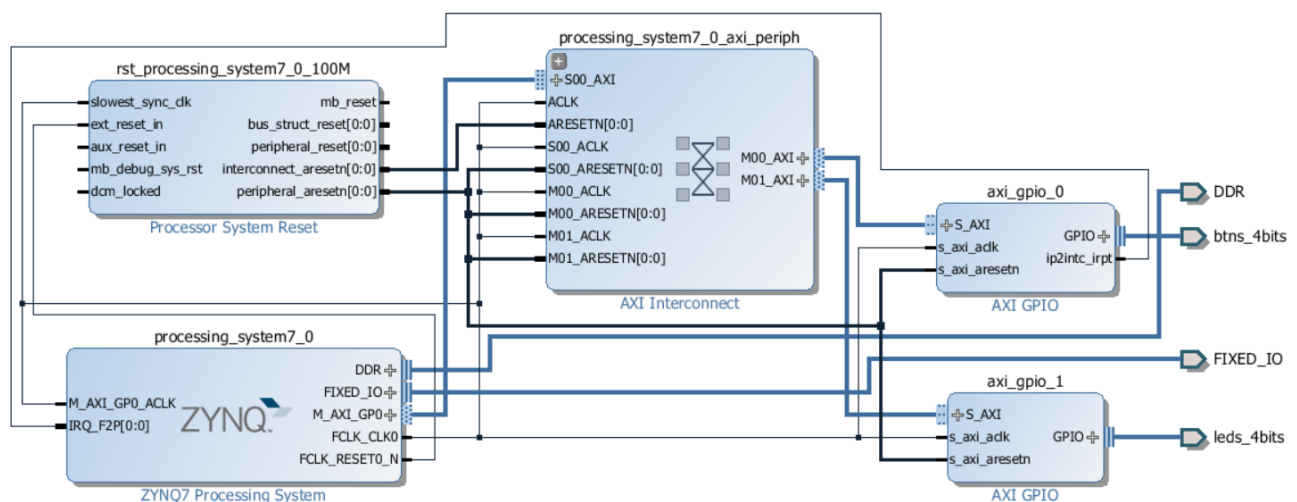Your final design should resemble Figure 2.18, although the positioning of your blocks may be different. You may want to remove unused peripherals such as SPI, USB, Ethernet, SD, TTC0, etc.

Figure 2.18: Zynq processor system with interrupts

(n) Save your design by selecting *File > Save Block Design* from the *Menu Bar*.

(o) Validate the design by selecting *Tools > Validate Design* from the *Menu Bar*. This will run a Design-Rule-Check (DRC).

Alternatively, select the Validate Design button, 🗹 , from the Main Toolbar.

(p) A *Validate Design* dialogue should appear to confirm that validation of the design was successful. Click *OK,* to dismiss the message.

With the design successfully validated, we can now move on to generating the HDL design files for the system. The procedure here is identical to the previous tutorial, **First Designs on Zynq**.

(q)   In the **Sources** window of the *Data Windows* pane, select the **Sources** tab.

(r)   Right-click on the top-level system design, which in this case is **zynq_interrupt_system**, and select **Create HDL Wrapper**.

The *Create HDL Wrapper* dialogue window will open. Accept the default option specifying that VIvado should manage the wrapper and click **OK**.

With all HDL design files generated, the next step in Vivado is to implement our design and generate a bitstream file.

(s)   In *Flow Navigator*, click **Generate Bitstream** from the *Program and Debug*   section.

If a dialogue window appears prompting you to save your design, click **Save**.

The combination of running the synthesis, implementation and bitstream generation processes back-to-back may take a few minutes, depending on the power of your computer system.

(t)   Once the bitstream generation is complete a dialogue window will open to inform you that the process has been completed successfully, as in Figure 2.19.
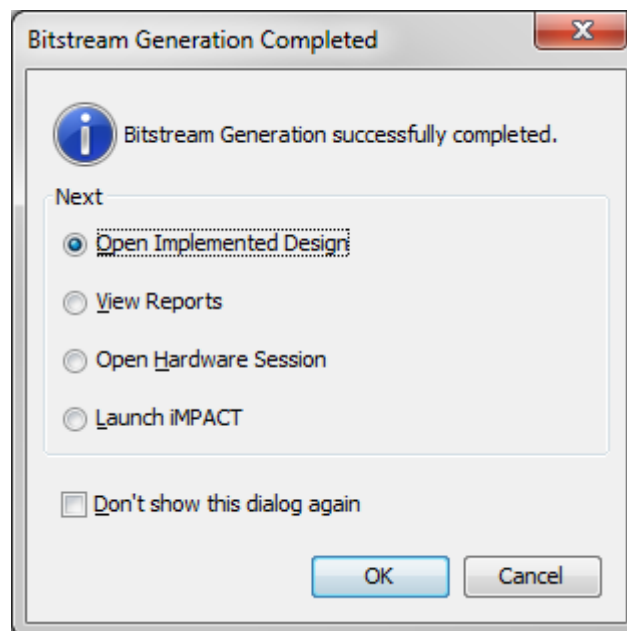

**Figure 2.19:** Bitstream Generation completion dialogue window

Select **Open Implemented Design**, and click **OK**.

At this point you will be presented with the *Device* view, where you can see the PL resources which are utilised by the design.

With the bitstream generation complete, the final step in Vivado is to export the design to the SDK, where we will create the software application that will allow the Zynq PS to control the LEDs on the Zybo board.

(u) Select **File > Export > Export Hardware for SDK...** from the *Menu Bar*.

(v) The *Export Hardware for SDK* dialogue window will open. Ensure that the options to **Include bitstream** and **Launch SDK** are selected, and Click **OK**.

This concludes the steps that are required in Vivado IDE. All hardware components of the system have been configured and generated. In the next exercise we will create the software application that utilises this hardware system.
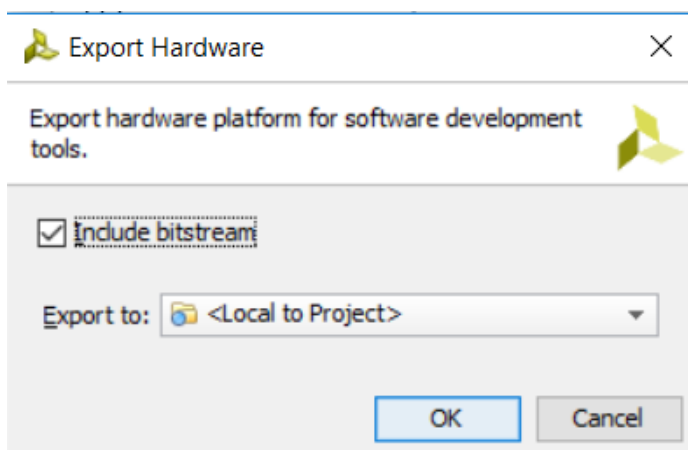
Creating a Software Application in the SDK

In this exercise a software application will be created that utilises hardware interrupts on the Zedboard. The push buttons will be used to increment a counter by different values, and the count will be continuously displayed on the LEDs in binary form, where LED0 corresponds to the least significant bit (LSB) and LED7 the most significant bit (MSB). This application will run on the Zynq processing systems, communicating with the AXI GPIO blocks implemented in the PL.

The SDK should have opened after the conclusion of Exercise 2B. If it did not open, you can open the SDK by navigating to **Start > All Programs > Xilinx Design Tools >Vivado 2014.4>SDK> Xilinx SDK 2014.4** and specifying the workspace as in Exercise 2A.

(a)   Select **File > New > Application Project** from the *Menu bar*.

(b)   The *New Project* dialogue window will open. Enter ***interrupt_counter*** in the *Project name* field, as shown in Figure 2.20, keeping all other options with the default settings. Click ***Next***.
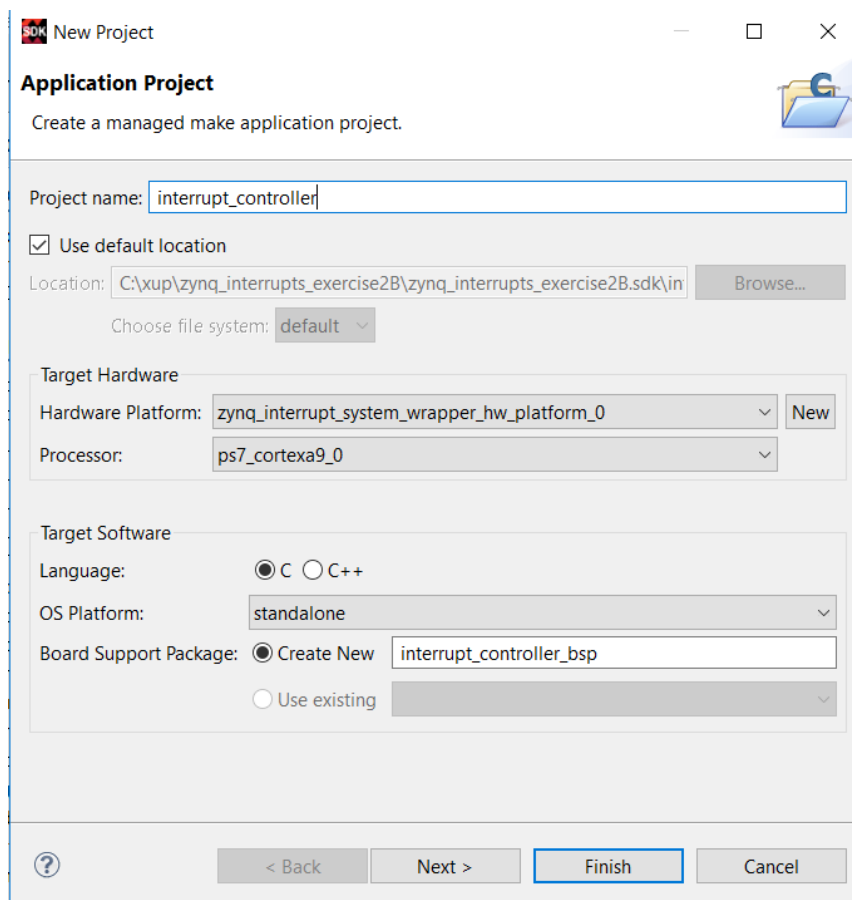
Figure 2.20: New Application Project dialogue

(c) At the *New Project Templates* screen, select ***Empty Application***, as in Figure 2.21, and click ***Finish*** to create the project.
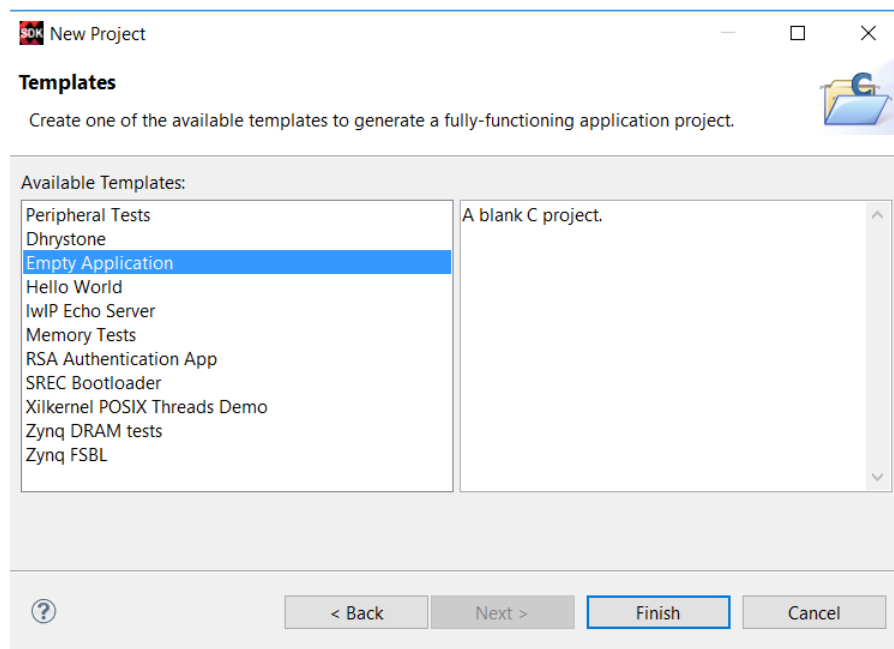
Figure 2.21: New Project Template dialogue

NOTE: the new project should open automatically. If it doesn't, you may need to close the Welcome tab in order to view the project.

With the new Application Project created, we can now import some pre-prepared source code for the application. You can open and copy source code interrupt_counter_2B.c from any directories you know.

If you have created a source directory as follows. you can follow the instructions to copy interrupt_counter_2B.c to your project.

(d) In the *Project Explorer* panel, expand ***interrupt_counter*** and highlight the *src* directory. Right-click and select ***Import...***, choosing ***General > File System*** as an import source.

(e) In the *Import File System* window, click the ***Browse...*** button.

(f) Navigate to the directory: **C:\Zynq_Book\sources\zynq_interrupts** and click ***OK.***

(g) Select the file **interrupt_counter_tut_2B.c**, as shown in Figure 2.22, and click **Finish**.
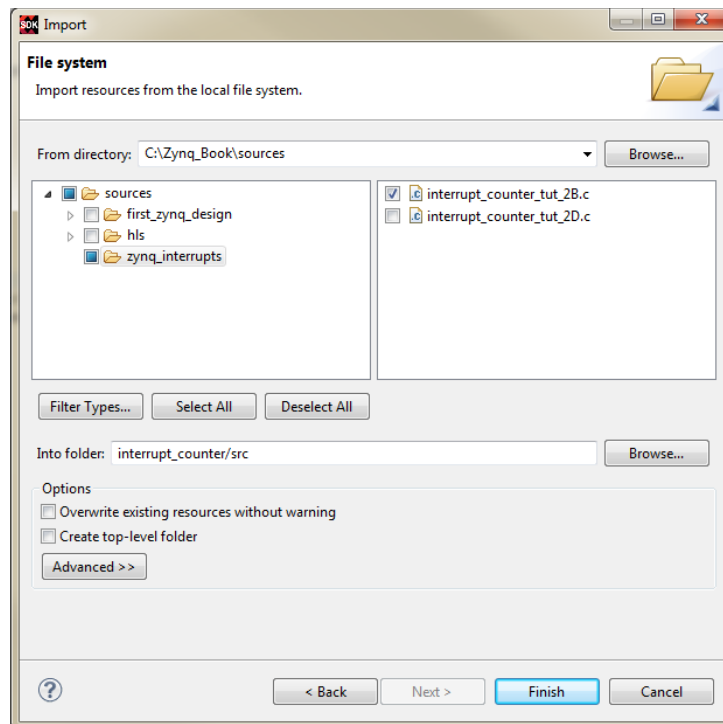
Figure 2.22: Import C source file

This file contains C Code that has been written to perform the interrupt triggered counter operation on the Zybo board.

(h) Open the imported source file by expanding the *src* folder and double-clicking on **interrupt_counter_tut_2B.c**, and explore the code.

The code has been fully commented, but will be briefly discussed here for clarity. Note that this file contains several portions of code which have been commented out; these will be utilised and discussed further in the next exercise and can be ignored for now.

By now, you should be familiar with the use of drivers and parameters in configuring and operating the GPIO. Remember, detailed information of these drivers can be found in the **system.mss** file, explaining the purpose of each function and the parameters passed to it. Predesignated parameters can also be found in `xparameters.h`.

The Zynq PS features a built in interrupt controller, initialised here as `XScuGic INTCInst`. This handles all incoming interrupt requests passed to the PS and performs the function associated with each interrupt source. It is also capable of prioritising multiple interrupt sources to the requirements of the application.

Of particular note is the inclusion of the function `BTN_Intr_Handler(void *InstancePtr);`. This is the interrupt handler function for the push buttons and is called every time an interrupt request from the push buttons in the PL is received in the PS. This performs a counter increment on each call and displays the value of the counter on the LEDs in binary.

An initial setup function can be found below the main function. This is `InterruptSystemSetup(XScuGic *XScuGicInstancePtr);`. The function initialises and configures the interrupt controller in the Zynq PS, connecting the interrupt handler to the interrupt source. It also makes a call to the latter function which enables the interrupt sources and registers exceptions.

The next step is to program the Zynq PL with the bitstream file that we generated in Exercise 2B.

Ensure that the ZedBoard is powered on and that the JTAG port is connected to the PC via the provided USB-A to USB-B cable.

(i)   Download the bitstream to the Zynq PL by selecting **Xilinx Tools > Program FPGA** from the *Menu bar*. The *Program FPGA* window will appear. The Bitstream field should already be populated with the correct bitstream file, as in Figure 2.23.
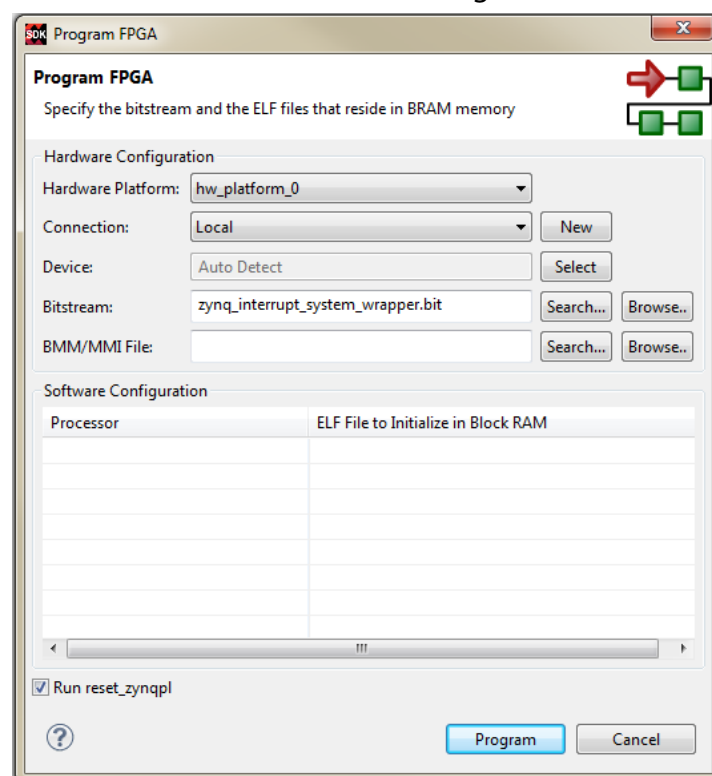
Figure 2.23: Program FPGA dialogue window

If it is not, enter:

**zynq_interrupt_system_wrapper.bit**

and click ***Program***.

As in the previous tutorial, once the device has successfully been programmed, the *DONE LED* on the ZedBoard will turn blue.

With the Zynq PL successfully configured with the bitstream file, we can now launch our software application on the Zynq PS.

(j) Select *interrupt_counter* in *Project Explorer*. Right-click and select **Run As > Launch on Hardware**.

The counter increments by different values based on the push button which is pressed. The counter operates as demonstrated in Figure 2.24.

LSB
MSB

LED0
LED7

00000000 = 0

00000001 = 1

00000010 = 2

01111111 = 254

11111111 = 255
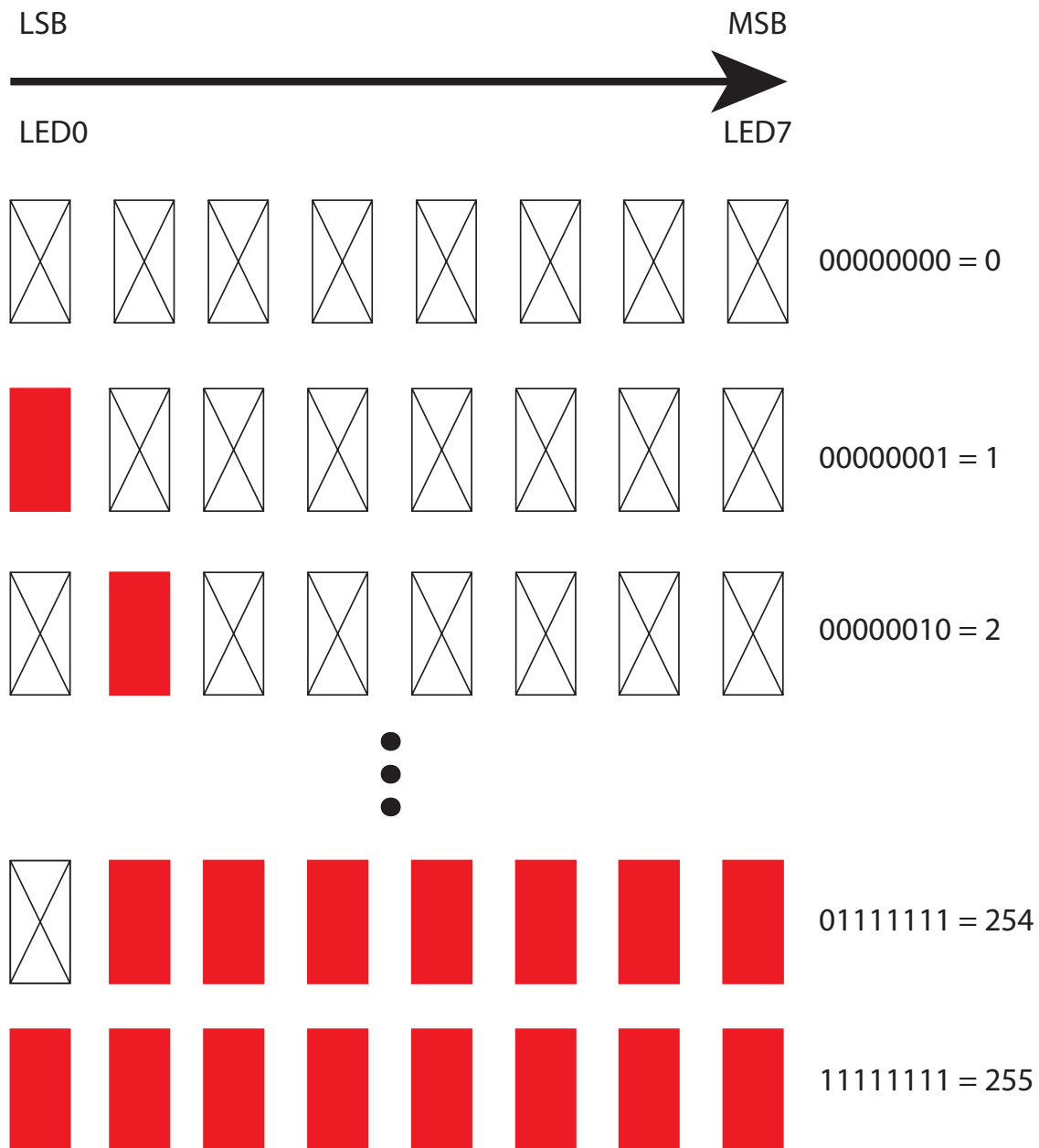
Figure 2.24: LED flashing states

(k) Try pressing different push buttons and observing how the counter increments (or does it increment at all?) Based on your findings, can you determine the value assigned to each of the push buttons (BTNU, BTND, BTNL, BTNR and BTNC as noted on the ZedBoard)?

You have successfully created and executed a software application utilising interrupts on the Zynq PS. The next step is to go back and add an additional interrupt source with higher priority to alter the functionality of the system.

In this exercise we will add an additional source of interrupt to the project created in Exercise 2B in the form of an **AXI Timer**. Save the last project as a new project for this part.

(a) Launch Vivado by double-clicking on the Vivado desktop icon: , or by navigating to **Start > All Programs > Xilinx Design Tools > Vivado 2014.1 > Vivado 2014.1**

(b) When the program launches, Create a new project called zynq_interrupts_exercise2D by saving project zynq_interrupts_exercise2B as zynq_interrupts_exercise2D.

(c) Open the block design from the sources panel by expanding the sources and double clicking on the block design as highlighted in Figure 2.25.
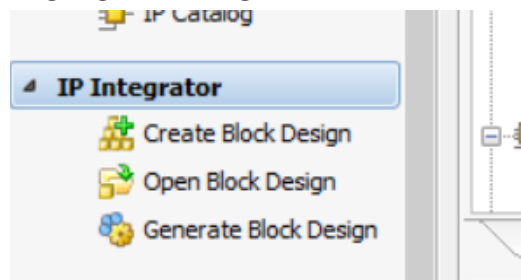

Figure 2.25: Opening an existing block diagram

(d) With the block diagram now open we will add an **AXI Timer** to the design. In the *Vivado IP Integrator Diagram* canvas, right-click anywhere and select ***Add IP***. Enter ***timer*** in the search field and add the IP ***AXI TIMER*** to the design by either dragging it onto the canvas or selecting it and pressing ENTER.
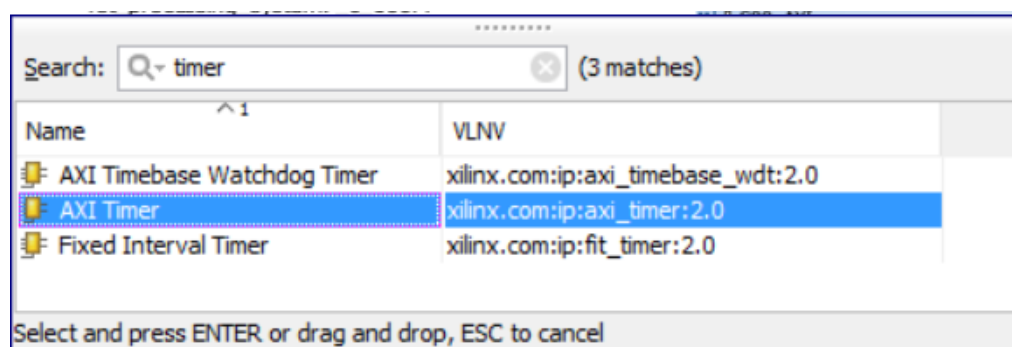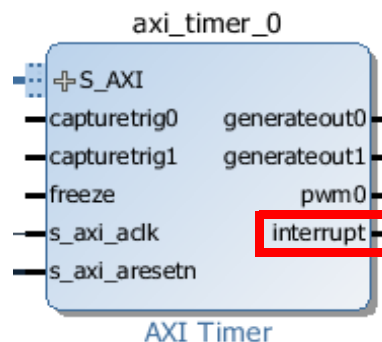

Figure 2.26: AXI Timer in the IP Catalog

(e) Select **Run Connection Automation** option from the *Designer Assistance* message at the top of the Diagram window and select**/axi_timer_0/S_AXI** and click **OK** to connect the timer to the AXI Interconnect.


Figure 2.27: AXI Timer in the block design

(f) Note that in Figure 2.27 the AXI Timer features an interrupt request, which requires connection to the Zynq PS. However, we already have an interrupt connected to the input of the PS. This input is a shared interrupt port, and so accepts multiple interrupts via one signal. We therefore require an additional IP block to concatenate these two interrupt requests into one signal. In the canvas, right-click anywhere and select **Add IP**. Enter **concat** in the search field and add the IP **Concat** to the design.
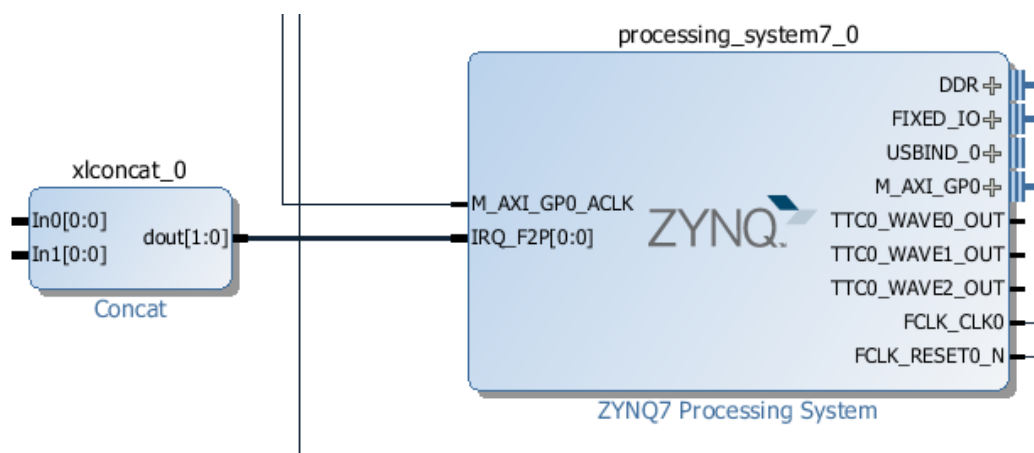
Figure 2.28: Concat in the block design

(g) Remove the connection to **IRQ_F2P[0:0]** on the Zynq PS by clicking it and pressing DELETE. Connect the output from the **Concat** block, **xlconcat_0** to this instead. Then, connect the interrupt request from the pushbutton GPIO to **In0[0:0]** and the interrupt from the timer to **In1[0:0]**, creating a shared interrupt signal that is passed to the PS. Your block diagram should be similar to Figure 2.29.
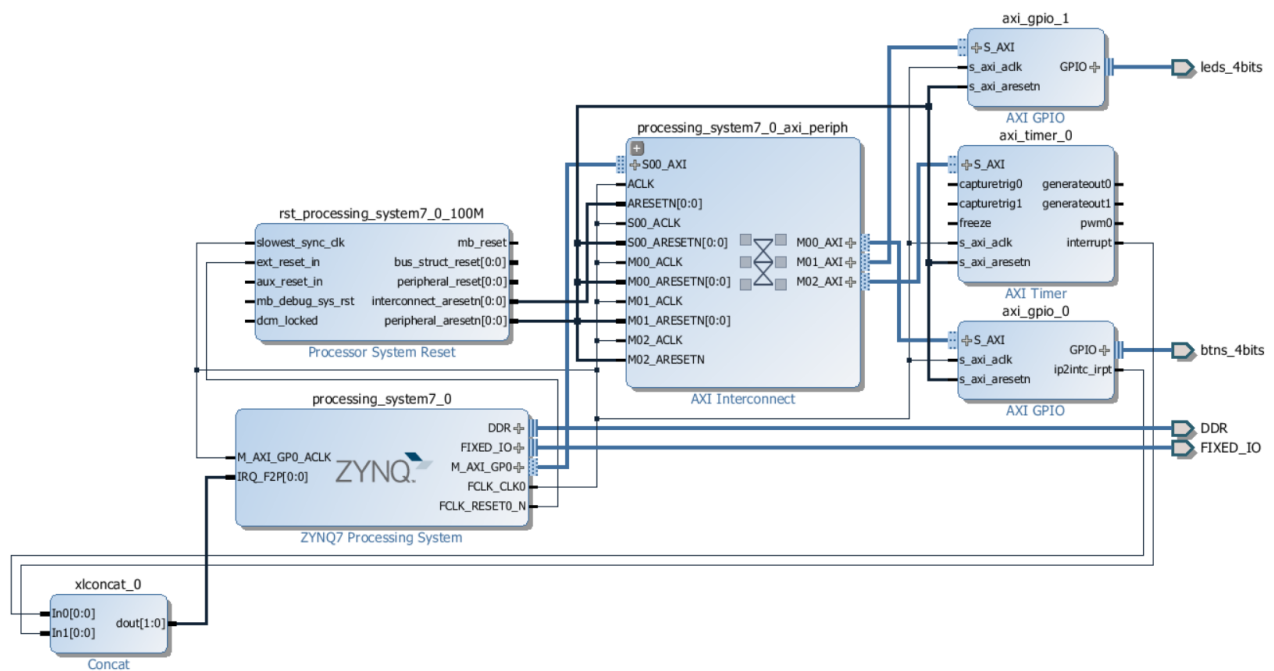
Figure 2.29: Complete system with multiple interrupt sources

We now need to regenerate the output products, update the HDL wrapped and generate a new bitstream for our altered design.

(h) Right-click on the top-level system design and select **Create HDL Wrapper...** selecting the default option as previous. Click **OK**.

(i) In *Flow Navigator*, click **Generate Bitstream** from the *Program and Debug* section.

If a dialogue window appears prompting you to save your design, click **Save**.

(j) A dialogue window will open requesting that you launch synthesis and implementation before starting the *Generate Bitstream* process. Click **Yes** to accept.

Again these back-to-back processes may take a few minutes, depending on the power of your computer system.

(k) When this process is completed click **OK**.

(l) Select **File > Export > Export Hardware for SDK...** from the *Menu Bar*.

(m) The *Export Hardware for SDK* dialogue window will open. Ensure that the options to **Include bitstream** and **Launch SDK** are selected, and Click **OK**. A dialog will be presented asking if you wish to overwrite an exported file, which is the initial system featuring a single interrupt. Select **Yes** for this and any further prompts.

(n) Once the SDK opens and builds the project, we will alter our application to make use of the new interrupt source. Right-click on the project **interrupt_counter** in the *Project Explorer* and select **Delete**.

Repeat for the BSP, **interrupt_controller_bsp**.

Repeat the steps outlined in Exercise 2B (a) to (h) for creating a new application project, BSP and importing a source file, this time selecting **interrupt_counter_2D.c**.

Notice the inclusion of a second interrupt handler, `TMR_Intr_Handler(void *data);` which will increment the value of the counter after the timer has expired three times, writing the new value to the LEDs.

Additional code has been included in the main to configure and start the timer, and full details of these functions can be found in the **system.mms**. The function `IntcInitFunction(u16 DeviceId, XTmrCtr *TmrInstancePtr,XGpio *GpioInstancePtr);` also contains additional code to connect the timer interrupt to the handler and enable it.

In brief, the timer is loaded with a value `TMR_LOAD` and configured to automatically reload on each expiration. The interrupt handler keeps track of the number of expirations and after three expirations performs the required steps, otherwise it simply increments the variable storing the number of expirations.

Save the file.

(o) Download the bitstream to the Zynq PL by selecting **Xilinx Tools > Program FPGA** from the *Menu bar*.

(p) Once the blue LED signalling successful programming lights, select ***zynq_interrupts*** in *Project Explorer*. Right-click and select **Run As > Launch on Hardware**.

Note that the counter will increment by 1 every time the timer expires three times. The buttons still operate as in the previous exercise.

This completes this tutorial and systems utilising both a single and multiple interrupt sources have been created and tested.