Spring 2021

California State University, Northridge

Department of Electrical & Computer Engineering

Lab 5

Interrupts

April 14, 2021

ECE 526L

Written By: Jose Luis Martinez

Introduction

For this lab we will be learning how to use interrupts with the Zynq SOC system. In a basic computer, the CPU has to wait for the I/O device when storing data from the I/O device to the memory. Interrupts are when an I/O device sends an interrupt signal to the processor letting it know it is ready to send data. The CPU can then service the interrupt and continue whatever it was doing before. This provides a solution to the problem mentioned beforehand as the CPU can keep performing other tasks while the I/O device readies its data.

Objective

After completing this lab, students will be able to:

- Create a new Zynq hardware project that includes interrupt components.
- Demonstrate the function of interrupts
- Develop an application code to utilize interrupts
- Add multiple sources of the interrupts to the system
- Use interrupt to add control functions

Methodology

The following are the steps I took to complete this lab:

Part I

1. Create a **Vivado** project selecting the **ZYBO** board files as the target device.
2. Create a block design using the **IP integrator** in the **Flow Navigator** window.
3. Add a **Zynq processor** to the block design and **Run Block Automation**.
4. Add 2 **AXI GPIO IP**s to the block design by right clicking in an empty area and selecting **Add IP**.
5. **Run Connection Automation** and for **axi_gpio_0** select **btns_4bits** for the interface.
6. For **axi_gpio_1** select **leds_4bits**.
7. To enable interrupts for the buttons, double click **axi_gpio_0** and go to the **IP Configuration** tab and check the box **Enable Interrupts**.
8. To configure the **Zynq PS** to accept interrupts requests we must configure it as follows.
    a. Double click the **Zynq PS** block and select **Interrupts** from the **Page Navigator.**
    b. Enable **Fabric Interrupts** and **IRQ_F2P**.
    c. Finally connect the interrupt signal between the **axi_block_0** and **Zynq PS**.
9. Save the design.
10. Validate the design by pressing F6.
11. In the **Sources** tab right click the top level design and select **Create HDL Wrapper**.
12. Click **Generate Bitstream** in the **Flow Navigator** window.
13. Export the hardware and include bitstream as well.
14. Open **Vitis** and create a new **application project**.
15. Create a new **Hardware Platform** using the xsa file created in step 13.
16. Click next then finish.
17. Import or copy the code **interrupt_counter_tut_2b.c** in the **Zynq Book Tutorial** from the sources folder.
18. Build the project.
19. Program and run on the **ZYBO.**

20. For the UART part of the project I simply added a xil_printf() statement in the BTN_Intr_Handler function.

Part II

1. Go back to the **Vivado** project that has the hardware configuration.
2. Go to **Flow Navigator** and **Open Block Design** under the IP integrator.
3. Right click in an empty space and add an **AXI Timer** to the block design.
4. Right click in an empty space and add an **Concat** IP to the design.
5. Delete the previous interrupt connection from **axi_gpio_0** to the **Zynq Ps.**
6. Connect the output of the **Concat** to the interrupt port on the **Zynq Ps.**
7. Connect **In0** to the interrupt signal coming from **axi_gpio_0.**
8. Connect **In1** to the interrupt signal coming from the **AXI Timer.**
9. Click **Generate Bitstream** in the **Flow Navigator** window.
10. Open **Implemented Design** and reload.
11. Export the hardware once again and make sure to include bitstream.
12. Open **Vitis** and create a new **application project**.
13. Create a new **Hardware Platform** using the xsa file created in step 11.
14. Click next then finish.
15. Import or copy the code **interrupt_counter_tut_2D.c** in the **Zynq Book Tutorial** from the sources folder.
16. Build the project.
17. Program and run on the **ZYBO.**
18. Modified the program provided to stop the timer when the left button is pressed, reset the timer when the right button is pressed, and reset the LED counter when the center button is pressed.

Part III

1. Go back to the **Vivado** project that has the hardware configuration.
2. Go to **Flow Navigator** and **Open Block Design** under the IP integrator.
3. Right click in an empty space and add another **GPIO** block to the design.
4. **Run Connection Automation** and connect the **GPIO** to the **sws_4bits**.
5. Double click **axi_gpio_2** and enable interrupts.
6. Double click **Concat** and add one more port.
7. Connect **In2** with the interrupt signal coming from **axi_gpio_2**.
8. Click **Generate Bitstream** in the **Flow Navigator** window.
9. Open **Implemented Design** and reload.
10. Export the hardware once again and make sure to include bitstream.
11. Open **Vitis** and create a new **application project**.
12. Create a new **Hardware Platform** using the xsa file created in step 10.
13. Click next then finish.
14. Copy the code from the **Part II** and modify it to include an interrupt source coming from the switches.
15. Build the project.
16. Program and run on the **ZYBO.**

Results/Code

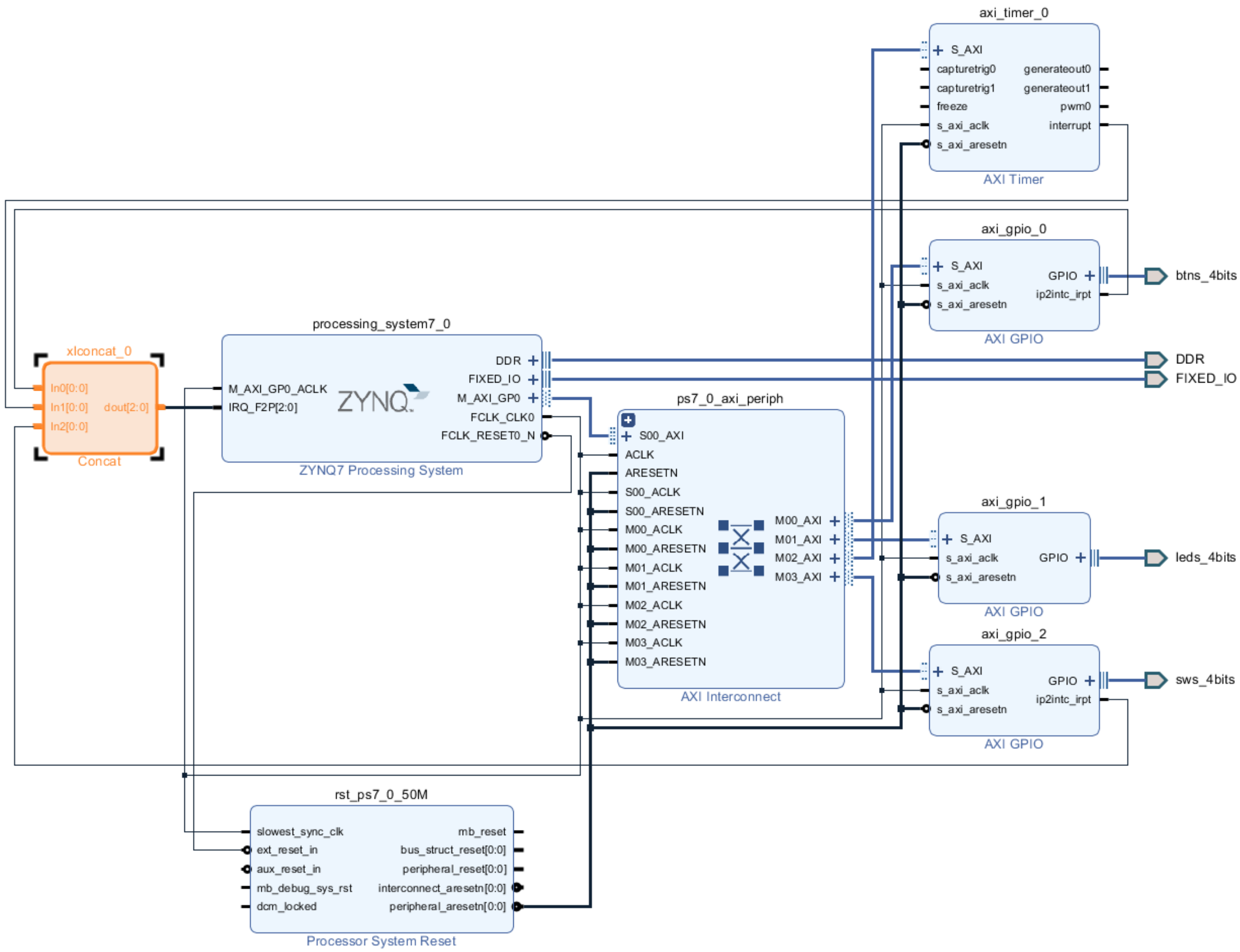


Fig. 1 IP Block Design

```c
/*
  Jose Luis Martinez
  Lab 5 - Interrupts
  ECE 520 - Professor Mirzaei
  4/12/2021
  Part I
 */

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
```

```c
#include "xparameters.h"
#include "xgpio.h"
#include "xscugic.h"
#include "xil_exception.h"

// Parameter definitions
#define INTC_DEVICE_ID          XPAR_PS7_SCUGIC_0_DEVICE_ID
#define BTNS_DEVICE_ID          XPAR_AXI_GPIO_0_DEVICE_ID
#define LEDS_DEVICE_ID          XPAR_AXI_GPIO_1_DEVICE_ID
#define INTC_GPIO_INTERRUPT_ID XPAR_FABRIC_AXI_GPIO_0_IP2INTC_IRPT_INTR

#define BTN_INT             XGPIO_IR_CH1_MASK

XGpio LEDInst, BTNInst;
XScuGic INTCInst;
static int led_data;
static int btn_value;

//----------------------------------------------------
// PROTOTYPE FUNCTIONS
//----------------------------------------------------
static void BTN_Intr_Handler(void *baseaddr_p);
static int InterruptSystemSetup(XScuGic *XScuGicInstancePtr);
static int IntcInitFunction(u16 DeviceId, XGpio *GpioInstancePtr);

//----------------------------------------------------
// INTERRUPT HANDLER FUNCTIONS
// - called by the timer, button interrupt, performs
// - LED flashing
//----------------------------------------------------


void BTN_Intr_Handler(void *InstancePtr)
{
	// Disable GPIO interrupts
	XGpio_InterruptDisable(&BTNInst, BTN_INT);
	// Ignore additional button presses
	if ((XGpio_InterruptGetStatus(&BTNInst) & BTN_INT) !=
			BTN_INT) {
			return;
		}
	btn_value = XGpio_DiscreteRead(&BTNInst, 1);
	// Increment counter based on button value
```

```c
      led_data = led_data + btn_value;
      xil_printf("Led value: %d, Button value: %d \n\r", led_data, btn_value);

    XGpio_DiscreteWrite(&LEDInst, 1, led_data);
    (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
    // Enable GPIO interrupts
    XGpio_InterruptEnable(&BTNInst, BTN_INT);
}

//----------------------------------------------------
// MAIN FUNCTION
//----------------------------------------------------
int main (void)
{
  init_platform();
  int status;
  xil_printf("Interupts lab. \n\r");
  //----------------------------------------------------
  // INITIALIZE THE PERIPHERALS & SET DIRECTIONS OF GPIO
  //----------------------------------------------------
  // Initialise LEDs
  status = XGpio_Initialize(&LEDInst, LEDS_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  // Initialise Push Buttons
  status = XGpio_Initialize(&BTNInst, BTNS_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  // Set LEDs direction to outputs
  XGpio_SetDataDirection(&LEDInst, 1, 0x00);
  // Set all buttons direction to inputs
  XGpio_SetDataDirection(&BTNInst, 1, 0xFF);

  // Initialize interrupt controller
  status = IntcInitFunction(INTC_DEVICE_ID, &BTNInst);
  if(status != XST_SUCCESS) return XST_FAILURE;




  while(1);
  cleanup_platform();
  return 0;
}
```

```
//----------------------------------------------------
// INITIAL SETUP FUNCTIONS
//----------------------------------------------------

int InterruptSystemSetup(XScuGic *XScuGicInstancePtr)
{
	// Enable interrupt
	XGpio_InterruptEnable(&BTNInst, BTN_INT);
	XGpio_InterruptGlobalEnable(&BTNInst);

	Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,

(Xil_ExceptionHandler)XScuGic_InterruptHandler,
	 	 	 	 	 	 	 	 	XScuGicInstancePtr);
	Xil_ExceptionEnable();


	return XST_SUCCESS;

}

int IntcInitFunction(u16 DeviceId, XGpio *GpioInstancePtr)
{
	XScuGic_Config *IntcConfig;
	int status;

	// Interrupt controller initialisation
	IntcConfig = XScuGic_LookupConfig(DeviceId);
	status = XScuGic_CfgInitialize(&INTCInst, IntcConfig,
IntcConfig->CpuBaseAddress);
	if(status != XST_SUCCESS) return XST_FAILURE;

	// Call to interrupt setup
	status = InterruptSystemSetup(&INTCInst);
	if(status != XST_SUCCESS) return XST_FAILURE;

	// Connect GPIO interrupt to handler
	status = XScuGic_Connect(&INTCInst,
								  INTC_GPIO_INTERRUPT_ID,
								  (Xil_ExceptionHandler)BTN_Intr_Handler,
								  (void *)GpioInstancePtr);
	if(status != XST_SUCCESS) return XST_FAILURE;
```

```
    // Enable GPIO interrupts interrupt
    XGpio_InterruptEnable(GpioInstancePtr, 1);
    XGpio_InterruptGlobalEnable(GpioInstancePtr);

    // Enable GPIO and timer interrupts in the controller
    XScuGic_Enable(&INTCInst, INTC_GPIO_INTERRUPT_ID);

    return XST_SUCCESS;
}
```

Fig. 2 main.c **Part I**

```
/*
  Jose Luis Martinez
  Lab 5 - Interrupts
  ECE 520 - Professor Mirzaei
  4/12/2021
  Part II
 */

#include <stdio.h>
#include "xparameters.h"
#include "xgpio.h"
#include "platform.h"
#include "xtmrctr.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xil_printf.h"

// Parameter definitions
#define INTC_DEVICE_ID          XPAR_PS7_SCUGIC_0_DEVICE_ID
#define TMR_DEVICE_ID       XPAR_TMRCTR_0_DEVICE_ID
#define BTNS_DEVICE_ID          XPAR_AXI_GPIO_0_DEVICE_ID
#define LEDS_DEVICE_ID          XPAR_AXI_GPIO_1_DEVICE_ID
#define INTC_GPIO_INTERRUPT_ID XPAR_FABRIC_AXI_GPIO_0_IP2INTC_IRPT_INTR
#define INTC_TMR_INTERRUPT_ID XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR

#define BTN_INT             XGPIO_IR_CH1_MASK
#define TMR_LOAD            0xF8000000

XGpio LEDInst, BTNInst;
XScuGic INTCInst;
XTmrCtr TMRInst;
```

```
static int led_data;
static int btn_value;
static int tmr_count;

//----------------------------------------------------
// PROTOTYPE FUNCTIONS
//----------------------------------------------------
static void BTN_Intr_Handler(void *baseaddr_p);
static void TMR_Intr_Handler(void *baseaddr_p);
static int InterruptSystemSetup(XScuGic *XScuGicInstancePtr);
static int IntcInitFunction(u16 DeviceId, XTmrCtr *TmrInstancePtr, XGpio
*GpioInstancePtr);

//----------------------------------------------------
// INTERRUPT HANDLER FUNCTIONS
// - called by the timer, button interrupt, performs
// - LED flashing
//----------------------------------------------------


void BTN_Intr_Handler(void *InstancePtr)
{
	// Disable GPIO interrupts
	XGpio_InterruptDisable(&BTNInst, BTN_INT);
	// Ignore additional button presses
	if ((XGpio_InterruptGetStatus(&BTNInst) & BTN_INT) !=
			BTN_INT) {
			return;
		}

	btn_value = XGpio_DiscreteRead(&BTNInst, 1);
	// Increment counter based on button value
	// Reset if centre button pressed
	//led_data = led_data + btn_value;

	switch(btn_value){
		case 1:
			XTmrCtr_Reset(&TMRInst,0);
			break;
		case 2:
			led_data = 0;
			break;
		case 4:
```

```
                XTmrCtr_Stop(&TMRInst,0);
                break;
           case 8:
                XTmrCtr_Start(&TMRInst,0);
                break;
           default:
                break;
      }

    XGpio_DiscreteWrite(&LEDInst, 1, led_data);
    (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
    // Enable GPIO interrupts
    XGpio_InterruptEnable(&BTNInst, BTN_INT);
}

void TMR_Intr_Handler(void *data)
{
      if (XTmrCtr_IsExpired(&TMRInst,0)){
           // Once timer has expired 3 times, stop, increment counter
           // reset timer and start running again
           if(tmr_count == 3){
                XTmrCtr_Stop(&TMRInst,0);
                tmr_count = 0;
                led_data++;
                XGpio_DiscreteWrite(&LEDInst, 1, led_data);
                XTmrCtr_Reset(&TMRInst,0);
                XTmrCtr_Start(&TMRInst,0);

           }
           else tmr_count++;
      }
}



//----------------------------------------------------
// MAIN FUNCTION
//----------------------------------------------------
int main (void)
{
  int status;
  //----------------------------------------------------
  // INITIALIZE THE PERIPHERALS & SET DIRECTIONS OF GPIO
```

```
  //----------------------------------------------------
  // Initialise LEDs
  status = XGpio_Initialize(&LEDInst, LEDS_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  // Initialise Push Buttons
  status = XGpio_Initialize(&BTNInst, BTNS_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  // Set LEDs direction to outputs
  XGpio_SetDataDirection(&LEDInst, 1, 0x00);
  // Set all buttons direction to inputs
  XGpio_SetDataDirection(&BTNInst, 1, 0xFF);


  //----------------------------------------------------
  // SETUP THE TIMER
  //----------------------------------------------------
  status = XTmrCtr_Initialize(&TMRInst, TMR_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  XTmrCtr_SetHandler(&TMRInst, TMR_Intr_Handler, &TMRInst);
  XTmrCtr_SetResetValue(&TMRInst, 0, TMR_LOAD);
  XTmrCtr_SetOptions(&TMRInst, 0, XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);


  // Initialize interrupt controller
  status = IntcInitFunction(INTC_DEVICE_ID, &TMRInst, &BTNInst);
  if(status != XST_SUCCESS) return XST_FAILURE;

  XTmrCtr_Start(&TMRInst, 0);




  while(1);

  return 0;
}

//----------------------------------------------------
// INITIAL SETUP FUNCTIONS
//----------------------------------------------------

int InterruptSystemSetup(XScuGic *XScuGicInstancePtr)
{
```

```
	// Enable interrupt
	XGpio_InterruptEnable(&BTNInst, BTN_INT);
	XGpio_InterruptGlobalEnable(&BTNInst);

	Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,

(Xil_ExceptionHandler)XScuGic_InterruptHandler,
					XScuGicInstancePtr);
	Xil_ExceptionEnable();


	return XST_SUCCESS;

}

int IntcInitFunction(u16 DeviceId, XTmrCtr *TmrInstancePtr, XGpio
*GpioInstancePtr)
{
	XScuGic_Config *IntcConfig;
	int status;

	// Interrupt controller initialisation
	IntcConfig = XScuGic_LookupConfig(DeviceId);
	status = XScuGic_CfgInitialize(&INTCInst, IntcConfig,
IntcConfig->CpuBaseAddress);
	if(status != XST_SUCCESS) return XST_FAILURE;

	// Call to interrupt setup
	status = InterruptSystemSetup(&INTCInst);
	if(status != XST_SUCCESS) return XST_FAILURE;

	// Connect GPIO interrupt to handler
	status = XScuGic_Connect(&INTCInst,
					INTC_GPIO_INTERRUPT_ID,
					(Xil_ExceptionHandler)BTN_Intr_Handler,
					(void *)GpioInstancePtr);
	if(status != XST_SUCCESS) return XST_FAILURE;


	// Connect timer interrupt to handler
	status = XScuGic_Connect(&INTCInst,
					INTC_TMR_INTERRUPT_ID,
					(Xil_ExceptionHandler)TMR_Intr_Handler,
```

```
                                        (void *)TmrInstancePtr);
    if(status != XST_SUCCESS) return XST_FAILURE;

    // Enable GPIO interrupts interrupt
    XGpio_InterruptEnable(GpioInstancePtr, 1);
    XGpio_InterruptGlobalEnable(GpioInstancePtr);

    // Enable GPIO and timer interrupts in the controller
    XScuGic_Enable(&INTCInst, INTC_GPIO_INTERRUPT_ID);

    XScuGic_Enable(&INTCInst, INTC_TMR_INTERRUPT_ID);


    return XST_SUCCESS;
}
```

Fig. 3 main.c **Part II**

```
/*
  Jose Luis Martinez
  Lab 5 - Interrupts
  ECE 520 - Professor Mirzaei
  4/12/2021
  Part III
 */

#include <stdio.h>
#include "xparameters.h"
#include "xgpio.h"
#include "platform.h"
#include "xtmrctr.h"
#include "xscugic.h"
#include "xil_exception.h"
#include "xil_printf.h"

// Parameter definitions
#define INTC_DEVICE_ID              XPAR_PS7_SCUGIC_0_DEVICE_ID
#define TMR_DEVICE_ID           XPAR_TMRCTR_0_DEVICE_ID
#define BTNS_DEVICE_ID              XPAR_AXI_GPIO_0_DEVICE_ID
#define LEDS_DEVICE_ID              XPAR_AXI_GPIO_1_DEVICE_ID
#define SW_DEVICE_ID            XPAR_AXI_GPIO_2_DEVICE_ID
#define INTC_GPIO_INTERRUPT_ID_0 XPAR_FABRIC_AXI_GPIO_0_IP2INTC_IRPT_INTR
#define INTC_GPIO_INTERRUPT_ID_2 XPAR_FABRIC_AXI_GPIO_2_IP2INTC_IRPT_INTR
#define INTC_TMR_INTERRUPT_ID    XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR
```

```c
#define BTN_INT             XGPIO_IR_CH1_MASK
#define SW_INT                  XGPIO_IR_CH1_MASK

XGpio LEDInst, BTNInst, SWInst;
XScuGic INTCInst;
XTmrCtr TMRInst;
static int led_data;
static int btn_value;
static int tmr_count;
static int sw_value;
static int tmr_load = 0xF8000000;
static int countUp = 1;

//------------------------------------------------------
// PROTOTYPE FUNCTIONS
//------------------------------------------------------
static void BTN_Intr_Handler(void *baseaddr_p);
static void TMR_Intr_Handler(void *baseaddr_p);
static void SW_Intr_Handler(void *baseaddr_p);
static int InterruptSystemSetup(XScuGic *XScuGicInstancePtr);
static int IntcInitFunction(u16 DeviceId, XTmrCtr *TmrInstancePtr, XGpio
*GpioInstancePtr_BTN, XGpio *GpioInstancePtr_SW);

//------------------------------------------------------
// INTERRUPT HANDLER FUNCTIONS
// - called by the timer, button interrupt, performs
// - LED flashing
//------------------------------------------------------


void BTN_Intr_Handler(void *InstancePtr)
{
	// Disable GPIO interrupts
	XGpio_InterruptDisable(&BTNInst, BTN_INT);
	// Ignore additional button presses
	if ((XGpio_InterruptGetStatus(&BTNInst) & BTN_INT) !=
			BTN_INT) {
			return;
		}

	btn_value = XGpio_DiscreteRead(&BTNInst, 1);
	// Increment counter based on button value
```

```
        // Reset if centre button pressed
        led_data = led_data + btn_value;

    XGpio_DiscreteWrite(&LEDInst, 1, led_data);
    (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
    // Enable GPIO interrupts
    XGpio_InterruptEnable(&BTNInst, BTN_INT);
}

void TMR_Intr_Handler(void *data)
{
        if (XTmrCtr_IsExpired(&TMRInst,0)){
                // Once timer has expired 3 times, stop, increment counter
                // reset timer and start running again
                if(tmr_count == 3){
                        XTmrCtr_Stop(&TMRInst,0);
                        tmr_count = 0;
                        if (countUp == 1) led_data++;
                        else led_data--;
                        XGpio_DiscreteWrite(&LEDInst, 1, led_data);
                        XTmrCtr_Reset(&TMRInst,0);
                        XTmrCtr_Start(&TMRInst,0);

                }
                else tmr_count++;
        }
}

void SW_Intr_Handler(void *InstancePtr)
{
        // Disable GPIO interrupts
        XGpio_InterruptDisable(&SWInst, SW_INT);
        // Ignore additional button presses
        if ((XGpio_InterruptGetStatus(&SWInst) & SW_INT) !=
                        SW_INT) {
                        return;
                }

        sw_value = XGpio_DiscreteRead(&SWInst, 1);

        if (sw_value & 0x1) XTmrCtr_Start(&TMRInst,0);
        else XTmrCtr_Stop(&TMRInst,0);
```

```
      if (sw_value & 0x2) countUp = 1;
      else countUp = 0;

      if (sw_value & 0x4) {
           tmr_load = 0xFF000000;
           XTmrCtr_SetResetValue(&TMRInst, 0, tmr_load);
           XTmrCtr_Reset(&TMRInst,0);
      }
      else {
           tmr_load = 0xF8000000;
           XTmrCtr_SetResetValue(&TMRInst, 0, tmr_load);
           XTmrCtr_Reset(&TMRInst,0);
      }

      if (sw_value & 0x6) tmr_count = 3;
      else tmr_count = 0;

    (void)XGpio_InterruptClear(&SWInst, SW_INT);
    // Enable GPIO interrupts
    XGpio_InterruptEnable(&SWInst, SW_INT);
}

//-----------------------------------------------------
// MAIN FUNCTION
//-----------------------------------------------------
int main (void)
{
  int status;
  //-----------------------------------------------------
  // INITIALIZE THE PERIPHERALS & SET DIRECTIONS OF GPIO
  //-----------------------------------------------------
  // Initialise LEDs
  status = XGpio_Initialize(&LEDInst, LEDS_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  // Initialise Push Buttons
  status = XGpio_Initialize(&BTNInst, BTNS_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  // Initialise Switch Buttons
  status = XGpio_Initialize(&SWInst, SW_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  // Set LEDs direction to outputs
  XGpio_SetDataDirection(&LEDInst, 1, 0x00);
  // Set all buttons direction to inputs
```

```
  XGpio_SetDataDirection(&BTNInst, 1, 0xFF);
  // Set all switches direction to inputs
  XGpio_SetDataDirection(&SWInst, 1, 0xFF);


  //------------------------------------------------------
  // SETUP THE TIMER
  //------------------------------------------------------
  status = XTmrCtr_Initialize(&TMRInst, TMR_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  XTmrCtr_SetHandler(&TMRInst, TMR_Intr_Handler, &TMRInst);
  XTmrCtr_SetResetValue(&TMRInst, 0, tmr_load);
  XTmrCtr_SetOptions(&TMRInst, 0, XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);


  // Initialize interrupt controller
  status = IntcInitFunction(INTC_DEVICE_ID, &TMRInst, &BTNInst, &SWInst);
  if(status != XST_SUCCESS) return XST_FAILURE;

  XTmrCtr_Start(&TMRInst, 0);




  while(1);

  return 0;
}

//------------------------------------------------------
// INITIAL SETUP FUNCTIONS
//------------------------------------------------------

int InterruptSystemSetup(XScuGic *XScuGicInstancePtr)
{
      // Enable interrupt
      XGpio_InterruptEnable(&BTNInst, BTN_INT);
      XGpio_InterruptGlobalEnable(&BTNInst);

      // Enable interrupt
      XGpio_InterruptEnable(&SWInst, SW_INT);
      XGpio_InterruptGlobalEnable(&SWInst);
```

```
	Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,

(Xil_ExceptionHandler)XScuGic_InterruptHandler,
	                                                XScuGicInstancePtr);
	Xil_ExceptionEnable();


	return XST_SUCCESS;

}

int IntcInitFunction(u16 DeviceId, XTmrCtr *TmrInstancePtr, XGpio
*GpioInstancePtr_BTN, XGpio *GpioInstancePtr_SW)
{
	XScuGic_Config *IntcConfig;
	int status;

	// Interrupt controller initialisation
	IntcConfig = XScuGic_LookupConfig(DeviceId);
	status = XScuGic_CfgInitialize(&INTCInst, IntcConfig,
IntcConfig->CpuBaseAddress);
	if(status != XST_SUCCESS) return XST_FAILURE;

	// Call to interrupt setup
	status = InterruptSystemSetup(&INTCInst);
	if(status != XST_SUCCESS) return XST_FAILURE;

	// Connect BTN GPIO interrupt to handler
	status = XScuGic_Connect(&INTCInst,
	                          INTC_GPIO_INTERRUPT_ID_0,
	                          (Xil_ExceptionHandler)BTN_Intr_Handler,
	                          (void *)GpioInstancePtr_BTN);
	if(status != XST_SUCCESS) return XST_FAILURE;

	// Connect SW GPIO interrupt to handler
	status = XScuGic_Connect(&INTCInst,
	                          INTC_GPIO_INTERRUPT_ID_2,
	                          (Xil_ExceptionHandler)SW_Intr_Handler,
	                          (void *)GpioInstancePtr_SW);
	if(status != XST_SUCCESS) return XST_FAILURE;

	// Connect timer interrupt to handler
	status = XScuGic_Connect(&INTCInst,
```

```
                                  INTC_TMR_INTERRUPT_ID,
                                  (Xil_ExceptionHandler)TMR_Intr_Handler,
                                  (void *)TmrInstancePtr);
    if(status != XST_SUCCESS) return XST_FAILURE;

    // Enable BTN GPIO interrupts interrupt
    XGpio_InterruptEnable(GpioInstancePtr_BTN, 1);
    XGpio_InterruptGlobalEnable(GpioInstancePtr_BTN);

    // Enable BTN GPIO interrupts interrupt
    XGpio_InterruptEnable(GpioInstancePtr_SW, 1);
    XGpio_InterruptGlobalEnable(GpioInstancePtr_SW);

    // Enable GPIO and timer interrupts in the controller
    XScuGic_Enable(&INTCInst, INTC_GPIO_INTERRUPT_ID_0);
    XScuGic_Enable(&INTCInst, INTC_GPIO_INTERRUPT_ID_2);
    XScuGic_Enable(&INTCInst, INTC_TMR_INTERRUPT_ID);


    return XST_SUCCESS;
}
```

Analysis

For the three parts of this lab, a separate Vitis application was made for each one and the hardware platform in Vivado was modified accordingly for each part.

The first part of the lab introduces only one interrupt source to the Zynq processor. The code in **Fig. 2** shows that buttons are the source of interrupts. When a button is pressed an interrupt signal will be sent to the processor and the interrupt handler will run. In part I the handler will read the value of the button presses and add them to the current LED value and write to the LED gpio.

The second part of the lab takes the code prom the previous part and adds another interrupt source. The new interrupt source being an AXI Timer. The code for this part will be acquired from the Zynq Tutorial book in the sources folder. The code is shown in **Fig. 3** and it works the same way as above but this time we will also have the AXI Timer as an interrupt source. The timer will send an interrupt signal every cycle specified and every time an interrupt is requested the processor will increment LED value by one and print to the LEDs.

The third part of the lab adds another interrupt source to the processor. The switches are the new interrupt source and they will be incharge of changing the settings of the AXI timer. The code for this part can be seen in **Fig. 4** and will work the same way as part II but with switches as an extra interrupt source. The switches will control various settings of the timer, switch 1 will pause/resume timer operation, switch 2 will control LED downcount/upcount, switch 3 will switch between two tmr_load values, and switch 4 will switch between two tmr_count values.

The results for all three parts will be shown in the verification video provided to canvas.

Conclusion

In conclusion, I learned how to use interrupts with the Zynq SOC. For the first part buttons were used as an interrupt source and their button value was added to the current LED value and writing it to the LEDs. The second part of the lab includes an AXI Timer as an interrupt source and will increment every time the timer sends an interrupt signal. For the third part switches were also added as an interrupt source and these controlled the settings of the AXI timers. I was able to complete all the requirements for this lab as shown in the video submitted in canvas.

Questions

Part I

2. Step 2C-h: Review the source file (interrupt_counter_tut_2B.c) and answer the following questions:

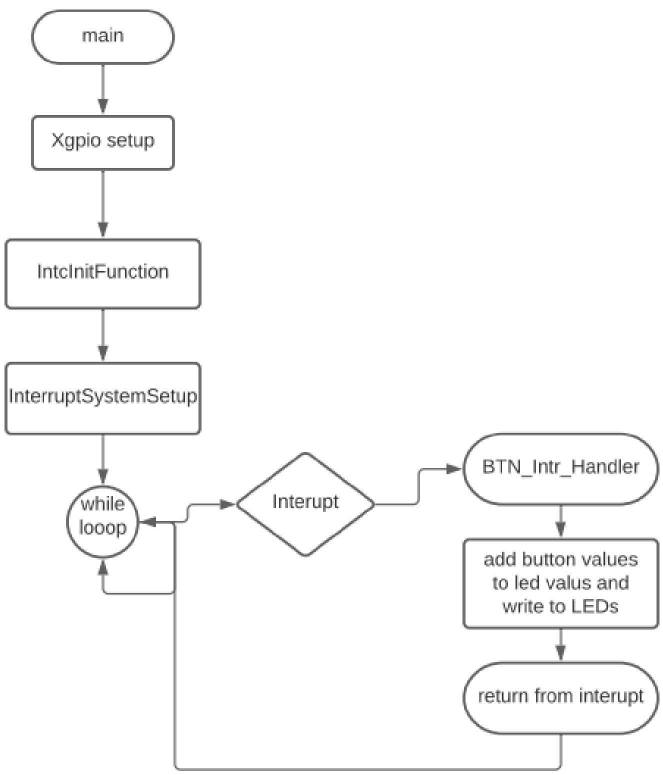
    a. Write a brief description of the following functions used by this software application.
        - BTN_Intr_Handler
        - InterruptSystemSetup
        - IntcInitFunction

    b. Draw a flowchart that represents the function of interrupt service routine BTN_Intr_Handler.

    c. Explain why interrupt is enabled and disabled in this function (BTN_Intr_Handler) in a specific order.

    d. Review the main function and explain what while loop does in this function.

A. The BTN_int_handler function is what happens when the button GPIO sends an interrupt signal. The InterruptSystemSetup function enables the Gpio interrupt for the button. The IntcInitFunction function set ups the interrupt controller and connects the interrupt to the handler.



B. **Fig. 5** interrupt flow chart

C. The interrupt is disabled in the beginning because we don’t want to trigger another interruption while we are still handling the current interrupt. Then at the end the interrupt is enabled to handle other incoming interrupts.

D. The while loop prevents the program from exiting and the interrupts should override anything that happens within the loop.

3. What are the base addresses and range for GPIO0 and GPIO1?

Address Map for processor ps7_cortexa9[0-1]

Filter: Search: 29 Loaded - 29 Shown - 0 Selected -

Cell	Base Address	High Address	Slave Interface	Addr Range Type
axi_gpio_0	0x41200000	0x4120ffff	S_AXI	register
ps7_scuwdt_0	0xf8f00620	0xf8f006ff	-	register
ps7_l2cachec_0	0xf8f02000	0xf8f02fff	-	register
ps7_scuc_0	0xf8f00000	0xf8f000fc	-	register
axi_gpio_1	0x41210000	0x4121ffff	S_AXI	register

Fig. 6 axi_gpio_0 & axi_gpio_1 addresses

From **Fig. 6** we can see that GPIO0 has a base address of 0x4120000 up to the high address of 0x4120ffff and GPIO1 has a base address of 0x41210000 up to the high address of 0x4121ffff.

4. Run the software application on hardware. Each press on one of the buttons adds a value to the counter value shown by LEDs. What is the value assigned to each of the five buttons?

For the ZYBO we only have four buttons. From right to left the button values are 1, 2, 4, and 8.

5. Explain why counter value cannot reach 255. Modify the software application so that you can generate the final count of 255. Include the change in source code in your answer. In order to be able to do this, you should add the math library by following the steps below.

For the ZYBO I did not have to modify anything in order to reach 15 or 255.

6. You connected GPIO0 interrupt pin to IRQ_F2P pin of the ZYNQ processor. What interrupt pin on Zynq is that? Provide status bit and ID number for this interrupt signal. Provide the proof for your claim.

Interrupt Port	ID	Description
Fabric Interrupts		Enable PL Interrupts to PS and vice versa
PL-PS Interrupt Ports		
IRQ_F2P[15:0]	[91:84], [68:61]	Enables 16-bit shared interrupt port from the PL. MSB is assigned the hi

```
/* Definitions for Fabric interrupts connected to ps7_scugic_0 */
#define XPAR_FABRIC_AXI_GPIO_0_IP2INTC_IRPT_INTR 61U

/******************************************************************/
```

Fig. 7 IRQ_F2P pin

As we can see in **Fig. 7** the IRQ_F2P pin the gpio interrupt signal uses is 61.

9. Write your conclusion from this part of the lab and the main key points you have learned in this part of the experiment.

See conclusion.

Part II

2. Review the source file (interrupt_counter_tut_2D.c). What is the purpose of this program and what does it do?

This program has two interrupt sources, being the buttons and an AXI Timer. This program will keep track of a value called LED_value that is written to the LEDs and is modified by the interrupt handlers of the buttons and AXI Timer. The button handler will add to the LED_value based on the value that is read from the buttons. The AXI Timer handler will increment by one to the LEF_value after a certain amount of time.

3. What are the main differences between this part of the lab and part I?

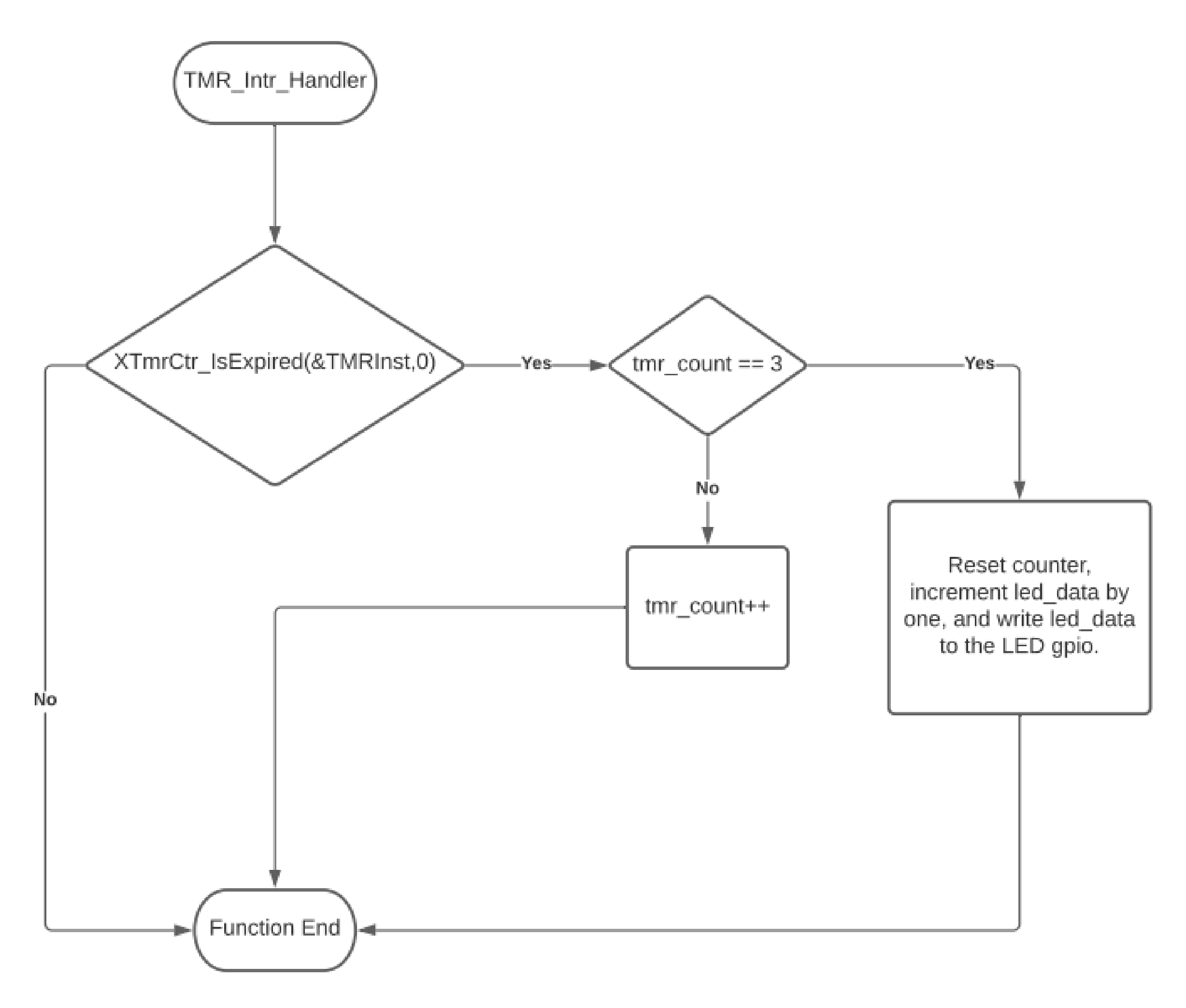
The main difference is that this part of the lab uses two sources of interrupts. And another being that this time the LEDS also increment by one periodically.

4. Review the source file (interrupt_counter_tut_2D.c) and answer the following questions:

    a. What are the sources of interrupts in this design?

    b. Draw a flowchart that represents the function of interrupt service routine TMR_Intr_Handler.

A. axi_gpio_0 and axi_timer_0 are the sources of interrupts.



B. Fig. 8

5. What are the address spaces assigned to the peripherals in this part? Where can you find this information?

Address Map for processor ps7_cortexa9[0-1]

Filter: Search: 30 Loaded - 30 Shown - 1 Selected -

Cell	Base Address	High Address	Slave Interface	Addr Range Type
axi_gpio_0	0x41200000	0x4120ffff	S_AXI	register
ps7_scuwdt_0	0xf8f00620	0xf8f006ff	-	register
ps7_l2cachec_0	0xf8f02000	0xf8f02fff	-	register
ps7_scuc_0	0xf8f00000	0xf8f000fc	-	register
axi_gpio_1	0x41210000	0x4121ffff	S_AXI	register
axi_timer_0	0x42800000	0x4280ffff	S_AXI	register

Fig. 9 axi_gpio_0, axi_gpio_1, and axi_timer_0 addresses.

From **Fig. 9** we can see that for axi_gpio_0 the base address is 0x41200000 to 0x4120ffff for the high address, axi_gpio_1 the base address is 0x41210000 to 0x4121ffff for the high address, and axi_timer_0 the base address is 0x42800000 to 0x4280ffff for the high address.

6. Run the software application on hardware. Answer the following questions:

   a. What is the function of this program? Explain what the operation of this program is.

   b. What is the difference between this software application in part I and part II?

   c. Do the push buttons work in the same way as part I?

A. This program has two interrupt sources, being the buttons and an AXI Timer. This program will keep track of a value called LED_value that is written to the LEDs and is modified by the interrupt handlers of the buttons and AXI Timer. The button handler will add to the LED_value based on the value that is read from the buttons. The AXI Timer handler will increment by one to the LEF_value after a certain amount of time.
B. The main difference is that this part of the lab uses two sources of interrupts. And another being that this time the LEDS also increment by one periodically.
C. The push buttons work exactly the same as in part 1 before we modified the button functionality.

7. You connected GPIO0 interrupt and AXI timer interrupt pins to IRQ_F2P pins of the ZYNQ processor. What interrupt pins on Zynq are those? Provide status bit and ID number for both interrupts. Provide the proof for your claim.

Interrupt Port	ID	Description
☑ Fabric Interrupts		Enable PL Interrupts to PS and vice versa
PL-PS Interrupt Ports		
☑ IRQ_F2P[15:0]	[91:84], [68:61]	Enables 16-bit shared interrupt port from the PL. MSB is assigned the hi

```
/* Definitions for Fabric interrupts connected to ps7_scugic_0 */
#define XPAR_FABRIC_AXI_GPIO_0_IP2INTC_IRPT_INTR 61U
#define XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR 62U

/******************************************************************/
```

Fig. 10 IRQ_F2P axi_gpio_0 and axi_timer_0 pins

From **Fig. 10** we can see that the pins for axi_gpio_0 is 61 and axi_timer_0 is 62.

8. The LED counter increments on the interrupt issued by the AXI timer. The counter rate can be controlled using two different variables: TMR_LOAD and tmr_count. Explain how each of these variables affect the rate of the counter.

   a. Does altering TMR_LOAD change the blinking rate of the LED counter? If yes, how? If no, is there a bug in the code? How do you fix this? Explain the correlation of the blinking rate and TMR_LOAD value.

   b. Does altering tmr_cnt change the blinking rate of the LED counter? If yes, how? If no, is there a **bug** in the code? How do you fix this? Explain the correlation of the blinking rate and tmr_cnt value.

A. Changing the TMR_LOAD value does affect the blinking rate of the LEDS. The higher value of TMR_LOAD the faster the LED blinking rate will be.
B. tmr_cnt can modify the blinking as after the timer expires 3 times the led will increment. So if you increase the number of counts to expire the circuit will take longer to blink the LEDs.