Temple University College of Engineering
Department of Electrical and Computer Engineering

Vivado AXI Timer and Interrupts

Course Number : 3613

Course Section : 1

Experiment # : 2

Student Name (print) : Christian Chiarulli
Tuid# : 915306875
Date : 2/9/2018

Grade : /100

TA Name : Sayemul Islam

Summary

This laboratory will be based on The second set of exercises from the Zynq tutorial book. After going through the tutorials we will add extra functionality introducing the switches but without giving them interrupt capabilities. The lab will also test on timers and interrupts in general.

Introduction

Before beginning the lab the code from exercise 2D must be understood thoroughly. After this we must then add switches as a functionality. This can be done by adding a second channel on the input GPIO for the buttons or by adding an entirely new GPIO interface. We are then to disable certain buttons or change their functionality depending on which switch is toggled on. The final part of the lab has to do with timer and finding the range in seconds and implementing a simple way to increase its expiration time.

Discussion

This section will begin by describing some of the functions and from exercise 2D. The definitions are found from the system.mss file. The important functions to scrutinize are the functions and parameters related to the interrupts and timers.

The program will begin in the main function and first initialize the GPIO for inputs and outputs by setting data directions. A third GPIO was added later for the switches. We then initialize the timer, set the timer, set the reset value which is loaded with TMR_LOAD and set the timer options to interrupt mode.

The function IntcInitFunction() is also called in main to enable the interrupts. The interrupt controller is first initialized by looking up the device ID and passing that configuration into the General interrupt initializer. The InterruptSystemSetup() function is then called which Enables interrupts for the buttons and uses the exception handler to catch any issues.

The IntcInitFunction() continues by connecting the GPIO interrupt to the handler and the timer to the interrupt handler. The Xgpio library then calls two functions to enable interrupts and global interrupts. Finally the General interrupt controller enables the GPIO interrupts for the buttons and timer. If all of this has been successful the timer is then started using XtmrCtr_Start().

With the interrupts enabled the buttons can now be pressed without disabling the timer and vice versa.

In exercise 2D the timer will increment and add one to the LED display after the timer has interrupted three times. While this happens pressing any of the buttons will increment the timer by 1, 2, 4, 8 by pressing buttons 0, 1, 2, 3 respectively.

The next part o this lab is to add the switches on a seperate GPIO or channel. In this case another GPIO was added. The rules of operation with the switches integrated is as follows:

If the SW0 is on BTN1 and BTN2 are disabled.

If SW1 is on then BTN1 will increase the number of interrupts needed to increases the LED count up to 7.

If SW2 is on then BTN2 will decrease the number of interrupts needed to decrease the LED count down to 1.

If SW3 is on the following two operations are switched, that is BTN1 will decrease the count and BTN2 will increase th count.

The program will also not behave as expected if anymore than one switch is enabled at one time.

The original timer counts to 1 second at 0xF8000000, it is set by the TMR_LOAD parameter which is the value that the counts starts at. The timer is a 32 bit timer so it can be set to 0xFFFFFFFF which will effectively set the expiration time to be infinitely small, in this case all the LED's will remain on.

To increase the expiration time you could increase the amount of counts needed to reset the timer. Which will add time to the timer. You could also lower the frequency of the clock or possibly add a second timer. In my code I increased the value for dynamic_count to be very high which increased the max time. Also for the original max time TMR_LOAD must be set to 0x00000000 which is the minimum value to count up from

The original and updated code can be found in the appendix.

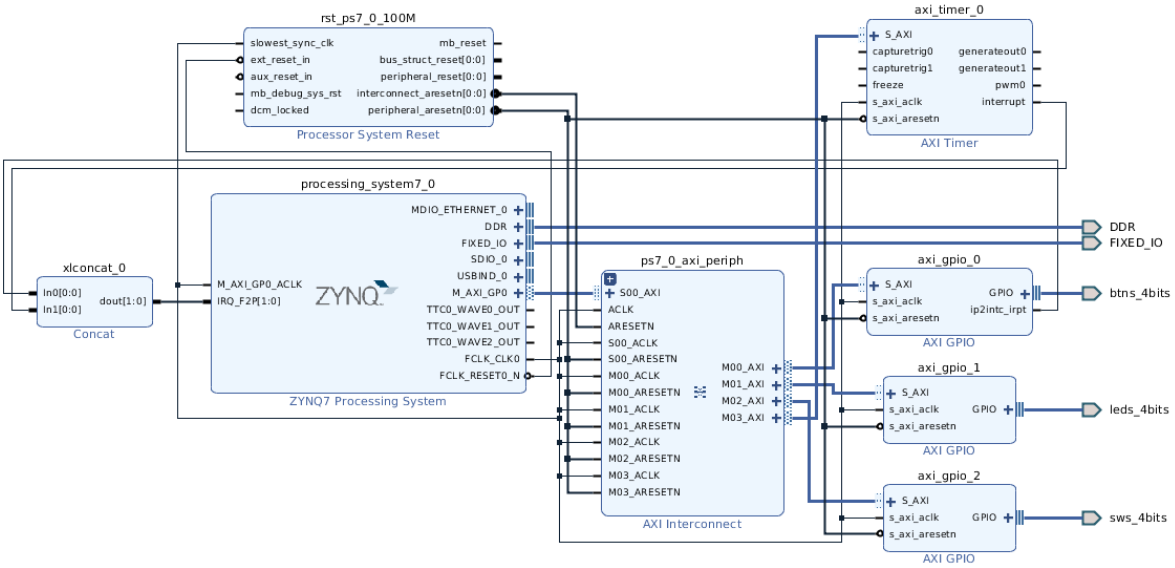

Figure 1

Conclusions

In conclusion this project as a success. This lab illustrates the different use cases for interrupts pertaining to the timer and buttons. This lab also shows the difference between how the switches behave when not implenting interrupts and the buttons which do use interrupts.

Appendix

```
// if multiple switches are on at once the program may not behave as expected

#include "xparameters.h"
#include "xgpio.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 SWS_DEVICE_ID         XPAR_AXI_GPIO_2_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
//#define TMR_LOAD                  0x1f0000000
//#define TMR_LOAD                  0x10000000
// 28 seconds at 0x60000000
// 12 seconds at 0xC0000000
// 4  seconds at 0x1f0000000
XGpio LEDInst, BTNInst, SWSInst;
XScuGic INTCInst;
XTmrCtr TMRInst;
static int led_data;
static int btn_value;
static int sws_value;
static int tmr_count;
static volatile int dynamic_count = 3000000;

//----------------------------------------------------
// 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);
	sws_value = XGpio_DiscreteRead(&SWSInst, 1);

	// changed this to interact appropriately with switches
	if ((sws_value != 0x1 || btn_value == 0x1 || btn_value == 0x8) && (btn_value
!= 0x2 || btn_value != 0x4)){
		if (sws_value != 0x2 && sws_value != 0x4 && sws_value != 0x8){

			// Increment counter based on button value
			// Reset if center button pressed
			led_data = led_data + btn_value;
		}
	}

	if (sws_value == 0x2 && btn_value == 0x2){
		if(dynamic_count < 11000000){
			dynamic_count += 2000000;
		}
		else{
			dynamic_count = dynamic_count;
		}
	}

	if (sws_value == 0x4 && btn_value == 0x4){
		if (dynamic_count > 1000000){
			dynamic_count -= 2000000;
			tmr_count = 0;
		}
	}

	if ((sws_value == 0x8 && btn_value == 0x2) || (sws_value == 0x8 && btn_value
== 0x4)){
		if(btn_value == 0x2){
			if(dynamic_count > 1000000){
				dynamic_count -= 2000000;
				tmr_count = 0;
			}
		}
		if(btn_value == 0x4){
			if(dynamic_count < 11000000){
				dynamic_count += 2000000;
			}
		}
	}
    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 == dynamic_count/){ // added dynamic count
                  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
  //----------------------------------------------------
  // Initialize LEDs
  status = XGpio_Initialize(&LEDInst, LEDS_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  // Initialize Push Buttons
  status = XGpio_Initialize(&BTNInst, BTNS_DEVICE_ID);
  if(status != XST_SUCCESS) return XST_FAILURE;
  // Initialize Switches
  status = XGpio_Initialize(&SWSInst, SWS_DEVICE_ID); //added switches
  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(&SWSInst, 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;
}

UNMODIFIED CODE


/*
 * interrupt_counter_tut_2B.c
 *
 *  Created on:   Unknown
 *      Author:   Ross Elliot
 *     Version:         1.1
 */

/
*****************************************************************************
*********

* VERSION HISTORY
*****************************************************************************
*********
*     v1.1 - 01/05/2015
*           Updated for Zybo ~ DN
*
*     v1.0 - Unknown
*           First version created.
*****************************************************************************
********/

#include "xparameters.h"
#include "xgpio.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;

    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;
```