Lab 1

Vivado SDK Basic IP Integrator

Christian Chiarulli
Tug19562@temple.edu

Summary

This laboratory will introduce embedded development on the Vivado Zynq Processor System (PS) with a basic IP integrator on the Zybo Board. We will be generating the hardware model and modifying a C template for the software that is meant to be run on the board. This lab will also be utilizing two GPIO interfaces as oppose to the template program only using one.

Introduction

The problems we will be facing in this lab are properly generating the hardware for use with two GPIO interfaces, modifying the template code to achieve the desired outcome on the hardware, and doing this without the interrupts provided in example 2B from the eText.

Button presses will be the most difficult part because the hardware may detect presses that were unintentional by the operator. Interrupts solved this for the example but a creative solution must be put in place to solve this issue without access to interrupts.

The template code used will be from the first example which only implements one GPIO. This must be expanded to two interfaces, one for the buttons on for the LED's. The buttons also must have specific functionality. If button 3 is pressed singly count operations are suspended and 8 is added to the count. When the button is released the new count will be incremented by 8 and the LED display will output the new count. The same operations are true for buttons 2 and 1 using values 4 and 2 respectively. Button 0 will act as a reset for the counter, if it is pressed the count will be set to zero and the count may resume when released.

Discussion

The first step to complete this project was to generate the hardware needed interface with the buttons and LED's. This was done by following the basic examples outlined in the eText. This includes importing the zybo board, creating a block design, generating the HDL wrapper, generating the bit stream and exporting the hardware. Then you can launch the SDK to write the software for the project.

Creating the block design was done by adding the Zynq processing system and two GPIO interfaces. GPIO 0 is used to implement the buttons and GPIO 1 is used to implement the LED's. After this the block design is saved and then validated. The diagram for the hardware can be found in figure 1.

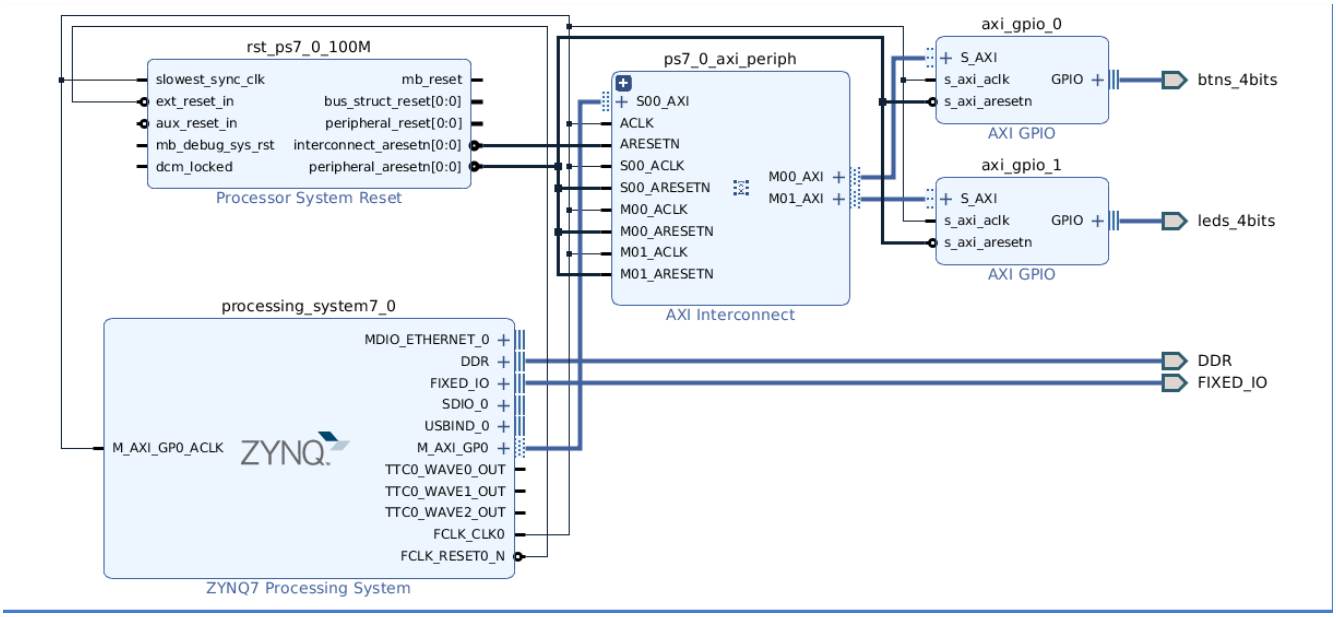

Figure 1: VHDL Hardware Design

After finishing the block diagram it is time to write the software in the SDK. We first create a new empty application and then import the template C program used in the first exercise in the Zynq tutorials. The file is also renamed “LED_BUTTON.c”.

Some changes had to be made to the initial template code. A second GPIO had to be added for the buttons. The buttons had to be initialized in addition to the LED’s. Much of the initial function was cleared out but the polling loop remained. The buttons were then polled to check if one was pressed. The value for the buttons press was stored in btn_value, which was set equal to `XGpio_DiscreteRead(&BTNInst, CHANNEL);`. The function comes from the xgpio.h library and reads in a Instance pointer and the specified channel. The following if statements check if the button is equal to 0x1, 0x2, 0x4, 0x8, these are the values that are internally assigned to each button. The program also uses the discrete read function to test if the button has been let go, if it has the value can then be updated that has been assigned to a temporary variable named count. The value for led_data is read into the discrete write function which takes an instance pointer for the XGPIO, the specified channel, and the value that will be used to update the physical LED’s.

The original code and modified version can be found in the appendix. The original and modified code will also be attached with this report.

Conclusions

In conclusion this project as a success. A problem with the overall implementation is that the program is essentially frozen while waiting for the button to be released. This means the hardware can only serve one purpose at a time which is highly inefficient.

Appendix

MODIFIED CODE

```
/* Include Files */
#include "xparameters.h" // variables such as XPAR_AXI_GPIO_1_DEVICE_ID can be found here
#include "xgpio.h"              // adds GPIO funtions
#include "xstatus.h"    // adds status functionality
#include "xil_printf.h"         // just adds a better print


/* Definitions */
#define CHANNEL 1                                                       /* GPIO port for
LEDs and buttons */
#define printf xil_printf                                               /* smaller, optimized printf
*/

#define BTNS_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID   // GPIO device buttons are
connected to
#define LEDS_DEVICE_ID XPAR_AXI_GPIO_1_DEVICE_ID   // GPIO device LED's are connected
to

static int led_data;                                                    // Hold current LED value.
Initialize to led_data definition
static int btn_value;                                                   // Hold current value for
buttons

XGpio LEDInst, BTNInst;                                                 // GPIO Device
driver instances

int LEDOutputExample(void)
{
        int status;                                                             // should
equal XST_SUCCESS if initialization went smoothly
        int count = 0x0;                                                        // holds
temporary LED values
        int b0 = 0x1;                                                   // button0
        int     b1 = 0x2;                                                       // button1
        int     b2 = 0x4;                                                       // button2
        int b3 = 0x8;                                                   // button3

        //---------------------------------------------------
                // 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;
		// Set LEDs direction to outputs
		XGpio_SetDataDirection(&LEDInst, CHANNEL, 0x00);
		// Set all buttons direction to inputs
		XGpio_SetDataDirection(&BTNInst, CHANNEL, 0xFF);


	while(1){
			/* Read input from the buttons. */
			btn_value = XGpio_DiscreteRead(&BTNInst, CHANNEL);

			// check what button is pressed and assign the appropriate value to temp
			if (b0 == btn_value){
				count = 0x0;
			}

			if (b1 == btn_value){
				count = led_data + 0x2;
			}

			if (b2 == btn_value){
				count = led_data + 0x4;
			}

			if (b3 == btn_value){
				count = led_data + 0x8;
			}
			// update only when button is let go
			if((btn_value != 0x8) & (btn_value != 0x4) & (btn_value != 0x2) & (btn_value !
= 0x1)){

			led_data = count; // load the value to the LED's
			/* Write output to the LEDs. */
			XGpio_DiscreteWrite(&LEDInst, CHANNEL, led_data);
			}

			else{
				while(1){
					// stay in the loop unless the button is let go, this stops the
program from picking up continuous presses
					if (XGpio_DiscreteRead(&BTNInst, CHANNEL) == 0x0){
						break;
					}
				}
			}
```

```
	}
		return XST_SUCCESS; /* Should be unreachable */
}

/* Main function. */
int main(void){


	int Status;

	/* Execute the LED output. */
	Status = LEDOutputExample();
	if (Status != XST_SUCCESS) {
		xil_printf("GPIO output to the LEDs failed!\r\n");
	}

	return 0;
}
```

ORIGINAL CODE

```
/* Include Files */
#include "xparameters.h"
#include "xgpio.h"
#include "xstatus.h"
#include "xil_printf.h"

/* Definitions */
#define GPIO_DEVICE_ID  XPAR_AXI_GPIO_0_DEVICE_ID   /* GPIO device that LEDs are
connected to */
#define LED 0x9                                                    /* Initial
LED value - X00X */
#define LED_DELAY 10000000                                         /* Software delay
length */
#define LED_CHANNEL 1                                              /* GPIO port for
LEDs */
#define printf xil_printf                                        /* smaller, optimised printf
*/

XGpio Gpio;                                                        /* GPIO
Device driver instance */

int LEDOutputExample(void)
{

	volatile int Delay;
	int Status;
	int led = LED; /* Hold current LED value. Initialise to LED definition */
```

```
		/* GPIO driver initialisation */
		Status = XGpio_Initialize(&Gpio, GPIO_DEVICE_ID);
		if (Status != XST_SUCCESS) {
			return XST_FAILURE;
		}

		/*Set the direction for the LEDs to output. */
		XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x0);

		/* Loop forever blinking the LED. */
			while (1) {
				/* Write output to the LEDs. */
				XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, led);

				/* Flip LEDs. */
				led = ~led;

				/* Wait a small amount of time so that the LED blinking is visible. */
				for (Delay = 0; Delay < LED_DELAY; Delay++);
			}

		return XST_SUCCESS; /* Should be unreachable */
}

/* Main function. */
int main(void){

	int Status;

	/* Execute the LED output. */
	Status = LEDOutputExample();
	if (Status != XST_SUCCESS) {
		xil_printf("GPIO output to the LEDs failed!\r\n");
	}

	return 0;
}
```