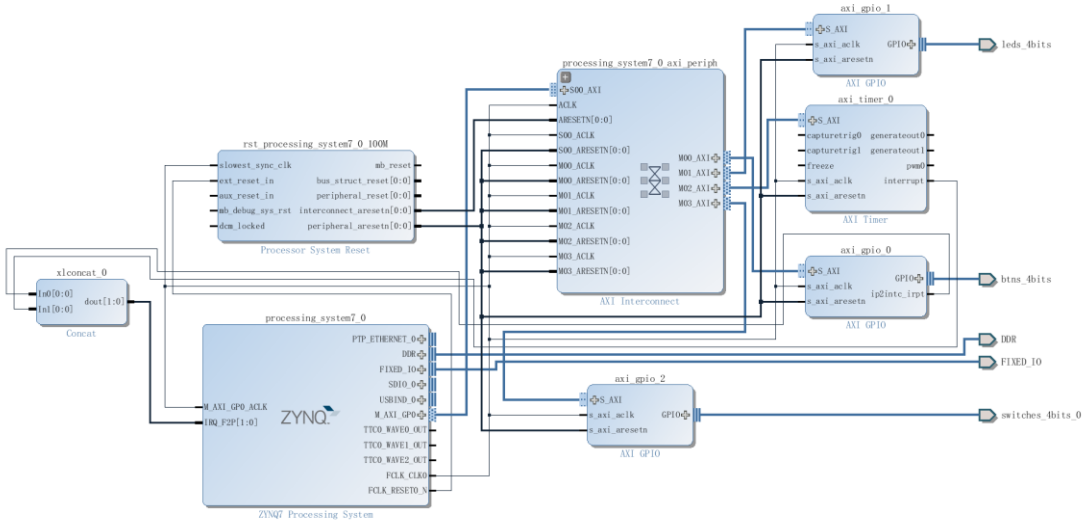
Lab8 Interrupt

Name : Yi Zihong ID : U201613634

Part One

In this lab, I uses the FPGA to add numbers with timer and button interrupt.



Part Two

In this lab, I finished the Ping-Pong game with timer and button interrupt. Also it can change the speed with reading the values of switches. These are my code.

```
1.  /*
2.   * interrupt_counter_tut_2B.c
3.   *
4.   *  Created on:     Unknown
5.   *      Author:     Ross Elliot
6.   *     Version:     1.1
7.   */
8.
9.  /************************************************************************
    ****************
10.
11. * VERSION HISTORY
12. ************************************************************************
    ****************
13. *   v1.1 - 01/05/2015
14. *       Updated for Zybo ~ DN
15. *
16. *   v1.0 - Unknown
17. *       First version created.
```

```
18. ******************************************************************************
    ***************/
19.
20. #include "xparameters.h"
21. #include "xgpio.h"
22. #include "xtmrctr.h"
23. #include "xscugic.h"
24. #include "xil_exception.h"
25. #include "xil_printf.h"
26. #include "XScuTimer.h"
27.
28. // Parameter definitions
29. #define INTC_DEVICE_ID      XPAR_PS7_SCUGIC_0_DEVICE_ID
30. #define TMR_DEVICE_ID       XPAR_TMRCTR_0_DEVICE_ID
31. #define BTNS_DEVICE_ID      XPAR_AXI_GPIO_0_DEVICE_ID
32. #define LEDS_DEVICE_ID      XPAR_AXI_GPIO_1_DEVICE_ID
33. #define SWITCES_DEVICE_ID   XPAR_AXI_GPIO_2_DEVICE_ID
34. #define INTC_GPIO_INTERRUPT_ID XPAR_FABRIC_AXI_GPIO_0_IP2INTC_IRPT_INTR
35. #define INTC_TMR_INTERRUPT_ID XPAR_FABRIC_AXI_TIMER_0_INTERRUPT_INTR
36.
37. #define BTN_INT             XGPIO_IR_CH1_MASK
38. #define TMR_LOAD            0xF8000000
39.
40. #define RESETBUTTON 0b0100
41. #define STARTBUTTON 0b0010
42. #define LEFTPADDLE 0b1000
43. #define RIGHTPADDLE 0b0001
44.
45. #define LED_PATTERNS_ORDER_LEFT_OUT 0
46. #define LED_PATTERNS_ORDER_RIGHT_OUT 5
47.
48. #define Paddle_Left 1
49. #define Paddle_Right 4
50.
51. #define START 1
52. #define STOP 0
53. #define LEFT 0
54. #define RIGHT 1
55.
56. #define ONE_TENTH 32500000 // half of the CPU clock speed/10
57.
58. int LED_PATTERNS_ORDER[6] = { 0b0000, 0b1000, 0b0100, 0b0010, 0b0001, 0b0000
     };
59.
```

```
XGpio LEDInst, BTNInst;
XScuGic INTCInst;
XTmrCtr TMRInst;
static int btn_value;
static int tmr_count;

static int scoreright;
static int scoreleft;
//----------------------------------------------------
// 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 *Gp
ioInstancePtr);

//----------------------------------------------------
// INTERRUPT HANDLER FUNCTIONS
// - called by the timer, button interrupt, performs
// - LED flashing
//----------------------------------------------------
char GameOver, StartDirection;
int led_order;
XGpio dip;
XScuTimer Timer; /* Cortex A9 SCU Private Timer Instance */
XScuTimer *TimerInstancePtr = &Timer;
int dip_check, dip_check_prev;

int number = 3;

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);

     switch(btn_value){
         case RESETBUTTON:
```

```
103.             xil_printf("\n\rNew Game - Scores Reset\r\n");
104.             scoreright = 0;
105.             scoreleft = 0;
106.             GameOver = STOP;
107.             xil_printf("Score Left = %d Score Right = %d\r\n", scoreright,s
    coreleft);
108.             break;
109.
110.         case STARTBUTTON:
111.
112.             GameOver = START; //start game
113.
114.             break;
115.
116.         case LEFTPADDLE:
117.             if (led_order != Paddle_Left) {
118.
119.                 xil_printf("Press too early !\r\n");
120.                 StartDirection = RIGHT;
121.                 GameOver = STOP;
122.                 led_order = LED_PATTERNS_ORDER_LEFT_OUT;
123.                 XGpio_DiscreteWrite(&LEDInst, 1,  LED_PATTERNS_ORDER[led_or
    der]);
124.                 scoreright += 1;
125.                 xil_printf("Score  = Left = %d Score Right %d\r\n", scorele
    ft, scoreright);
126.
127.             } else {
128.                 StartDirection = RIGHT;
129.                 scoreright += 1;
130.                 XGpio_DiscreteWrite(&LEDInst, 1,  LED_PATTERNS_ORDER[led_or
    der]);
131.                 xil_printf("Score  = Left = %d Score Right %d\r\n", scorele
    ft, scoreright);
132.             }
133.
134.             break;
135.
136.         case RIGHTPADDLE:
137.             if (led_order != Paddle_Right) {
138.                 StartDirection = LEFT;
139.                 GameOver = STOP;
140.
141.                 xil_printf("Press too early !\r\n");
```

```
142.                 led_order = LED_PATTERNS_ORDER_RIGHT_OUT;
143.                 XGpio_DiscreteWrite(&LEDInst, 1,  LED_PATTERNS_ORDER[led_or
    der]);
144.                 scoreleft += 1;
145.                 xil_printf("Score  = Left = %d Score Right %d\r\n", scorele
    ft, scoreright);
146.             } else {
147.                 StartDirection = LEFT;
148.                 scoreright += 1;
149.                 XGpio_DiscreteWrite(&LEDInst, 1,  LED_PATTERNS_ORDER[led_or
    der]);
150.                 xil_printf("Score  = Left = %d Score Right %d\r\n", scorele
    ft, scoreright);
151.             }
152.             break;
153.
154.         default:break;
155.     }
156.
157.     (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
158.     // Enable GPIO interrupts
159.     XGpio_InterruptEnable(&BTNInst, BTN_INT);
160. }
161.
162. void TMR_Intr_Handler(void *data)
163. {
164.     if (XTmrCtr_IsExpired(&TMRInst,0)){
165.         // Once timer has expired 3 times, stop, increment counter
166.         // reset timer and start running again
167.         if(tmr_count == number*10){
168.
169.             //judge different directions
170.             if (GameOver != STOP){
171.                 switch(StartDirection){
172.
173.                 case LEFT:
174.                 XTmrCtr_Stop(&TMRInst,0);
175.                 tmr_count = 0;
176.
177.                 led_order--;
178.                 XGpio_DiscreteWrite(&LEDInst, 1,  LED_PATTERNS_ORDER[led_or
    der]);
179.                 if(led_order == LED_PATTERNS_ORDER_LEFT_OUT){
180.                     scoreright += 1;
```

```
                    xil_printf("Score  = Left = %d Score Right %d\r\n", sco
releft, scoreright);
                    GameOver = STOP;
                    StartDirection = RIGHT;
                }
                XTmrCtr_Reset(&TMRInst,0);
                XTmrCtr_Start(&TMRInst,0);
                break;

            case RIGHT:
                XTmrCtr_Stop(&TMRInst,0);
                tmr_count = 0;

                led_order++;
                XGpio_DiscreteWrite(&LEDInst, 1,  LED_PATTERNS_ORDER[led_or
der]);

                if(led_order == LED_PATTERNS_ORDER_RIGHT_OUT){
                    scoreleft += 1;
                    xil_printf("Score  = Left = %d Score Right %d\r\n", sco
releft, scoreright);

                    GameOver = STOP;
                    StartDirection = LEFT;
                }

                XTmrCtr_Reset(&TMRInst,0);
                XTmrCtr_Start(&TMRInst,0);
                break;

            default:break;
                }
            }
          } else {
                tmr_count++;
                xil_printf("tmr_count %d\r\n", tmr_count);
        }
      }
}


//----------------------------------------------------
// MAIN FUNCTION
```

```
222. //----------------------------------------------------
223. int main (void)
224. {
225.   int status;
226.   //initialize variables, timers, ports
227.   XGpio_Initialize(&dip, SWITCES_DEVICE_ID);
228.   XGpio_SetDataDirection(&dip, 1, 0xffffffff);
229.
230.   //----------------------------------------------------
231.   // INITIALIZE THE PERIPHERALS & SET DIRECTIONS OF GPIO
232.   //----------------------------------------------------
233.   // Initialise LEDs
234.   status = XGpio_Initialize(&LEDInst, LEDS_DEVICE_ID);
235.   if(status != XST_SUCCESS) return XST_FAILURE;
236.   // Initialise Push Buttons
237.   status = XGpio_Initialize(&BTNInst, BTNS_DEVICE_ID);
238.   if(status != XST_SUCCESS) return XST_FAILURE;
239.   // Set LEDs direction to outputs
240.   StartDirection = RIGHT;
241.   led_order = 1;
242.   XGpio_DiscreteWrite(&LEDInst, 1,  LED_PATTERNS_ORDER[led_order]);
243.
244.   //----------------------------------------------------
245.   // SETUP THE TIMER
246.   //----------------------------------------------------
247.   status = XTmrCtr_Initialize(&TMRInst, TMR_DEVICE_ID);
248.   if(status != XST_SUCCESS) return XST_FAILURE;
249.   XTmrCtr_SetHandler(&TMRInst, TMR_Intr_Handler, &TMRInst);
250.   XTmrCtr_SetResetValue(&TMRInst, 0, TMR_LOAD);
251.   XTmrCtr_SetOptions(&TMRInst, 0, XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPT
    ION);
252.
253.   // Initialize interrupt controller
254.   status = IntcInitFunction(INTC_DEVICE_ID, &TMRInst, &BTNInst);
255.   if(status != XST_SUCCESS) return XST_FAILURE;
256.
257.   XTmrCtr_Start(&TMRInst, 0);
258.   xil_printf("\n\rInit all finished\r\n");
259.
260.
261.   while(1){
262.         dip_check = XGpio_DiscreteRead(&dip, 1);
263.         if (dip_check != dip_check_prev) {
264.             xil_printf("DIP Switch Status %x, %x\r\n", dip_check_prev,
```

```
                    dip_check);
            dip_check_prev = dip_check;
            number = dip_check;
        }
  }

  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);
```

```
306.     if(status != XST_SUCCESS) return XST_FAILURE;
307.
308.     // Connect GPIO interrupt to handler
309.     status = XScuGic_Connect(&INTCInst,
310.                              INTC_GPIO_INTERRUPT_ID,
311.                              (Xil_ExceptionHandler)BTN_Intr_Handler,
312.                              (void *)GpioInstancePtr);
313.     if(status != XST_SUCCESS) return XST_FAILURE;
314.
315.
316.     // Connect timer interrupt to handler
317.     status = XScuGic_Connect(&INTCInst,
318.                              INTC_TMR_INTERRUPT_ID,
319.                              (Xil_ExceptionHandler)TMR_Intr_Handler,
320.                              (void *)TmrInstancePtr);
321.     if(status != XST_SUCCESS) return XST_FAILURE;
322.
323.     // Enable GPIO interrupts interrupt
324.     XGpio_InterruptEnable(GpioInstancePtr, 1);
325.     XGpio_InterruptGlobalEnable(GpioInstancePtr);
326.
327.     // Enable GPIO and timer interrupts in the controller
328.     XScuGic_Enable(&INTCInst, INTC_GPIO_INTERRUPT_ID);
329.
330.     XScuGic_Enable(&INTCInst, INTC_TMR_INTERRUPT_ID);
331.
332.     return XST_SUCCESS;
333. }
```