Vivado AXI Timer and Interrupt

Robert Bara
Robert.Bara@temple.edu

Summary

In lab 4, “Vivado AXI Timer and Interrupt”, picks up where the last lab left off by introducing timers for the Vivado Zynq Processor System. The lab follows Chapter 2D of *The Zynq Book Tutorials* and extends upon tutorial code written in C.

Introduction

The purpose of this lab is to use both AXI Timers and Interrupts to manipulate LEDs. The cases for buttons generating an interrupt, switches enabling buttons or resetting conditions, and the LED output display is as follows:

SW2 On – This is a reset. If SW2 is on by itself, the timer resets back to 3 and LEDs will start over when SW2 gets turned off. If any other switches are high in conjunction to SW2, then a reset error occurs where the LEDs flash on and off for about 1 second each, until all switches are turned off and no buttons are pressed.

SW0 On – If SW0 is on, BTN0 can generate an interrupt where the LEDs will be set to display the current count of interrupts for 2 seconds, and the interrupt counter increments upon each interrupt, until it reaches a maximum count of 7.

SW1 On– If SW1 is on, BTN1 can generate an interrupt where the LEDs will be set to display the current count of interrupts for 2 seconds, and the interrupt counter decrements upon each interrupt, until it reaches a maximum count of 1.

Otherwise, the LEDs simply increment with a roll over and buttons will not generate an effective interrupt. BTN2 and BTN3 should always be ignored.

Discussion

Hardware Design

The Vivado hardware design continues where lab left off. First add the IP: ZYNQ7 Processing system(processor_system7_0) and run automation to generate/wire the processor(ps7_axi_periph) and reset block (rst_ps7_0_100M). Gpio_0 should then be added to wire push buttons as an interrupt input, be sure to enable the PL-PS interrupt port and select IRQ_F2P within the processing system block, in addition to enabling the interrupt. Gpio_1(axi_gpio_1) will wire LEDs as the output display. Gpi_2 should be generated to wire the Zybo’s switches as inputs. Add the axi_timer block and run automation. Finally, add a xlconcat block and wire the interrupt from Gpio_0 to the first input terminal, wire the interrupt from the axi_timer_0 to the second input terminal, and wire the output terminal to the IRQ_F2P terminal of the processing system block.

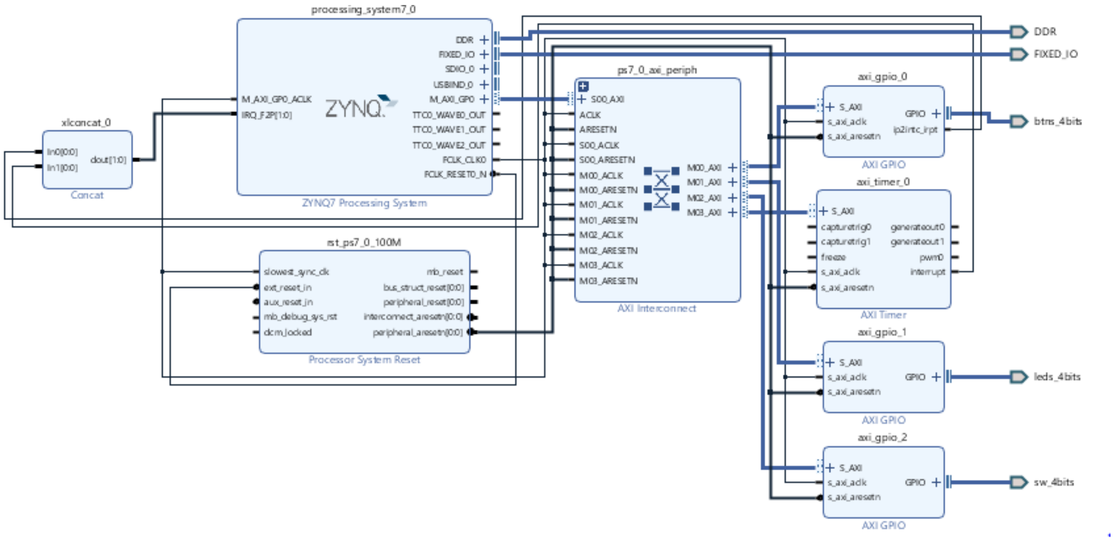


Figure 1. Final Block Diagram with timer and interrupt connections

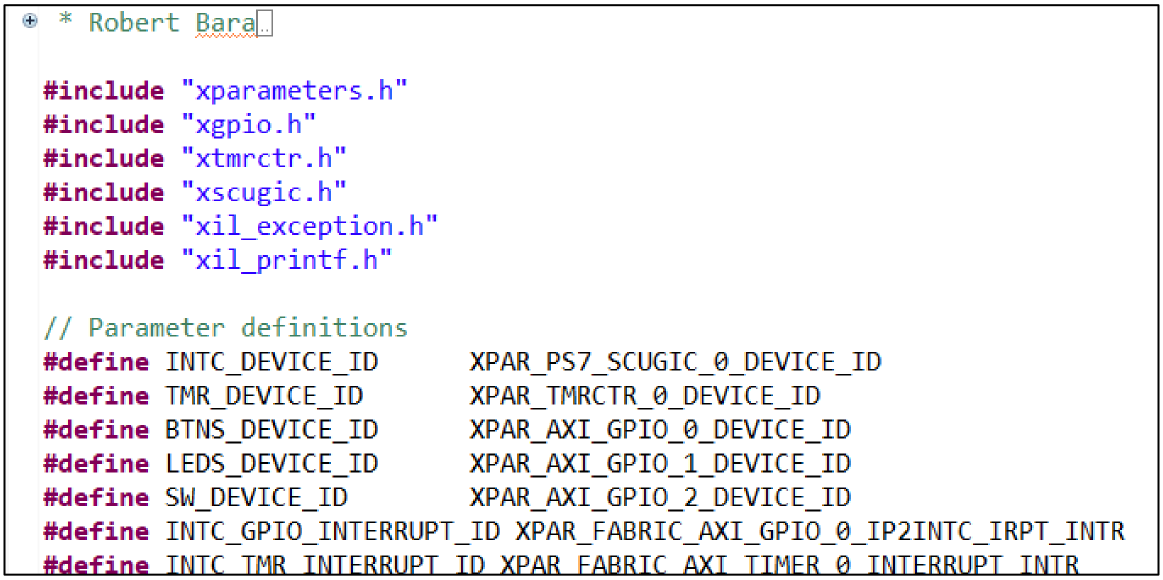
Save and validate the design, create the HDL wrapper, and generate the bitstream. Upon completion, the design should be exported to hardware, and SDK should be launched to begin programming.

C Program

The C programming from this lab, expands upon the *interrupt_controller_tut_2D.c* to tutorial program. Initialization and functions taken from the tutorial program will be explained in the following sections.

Initialization

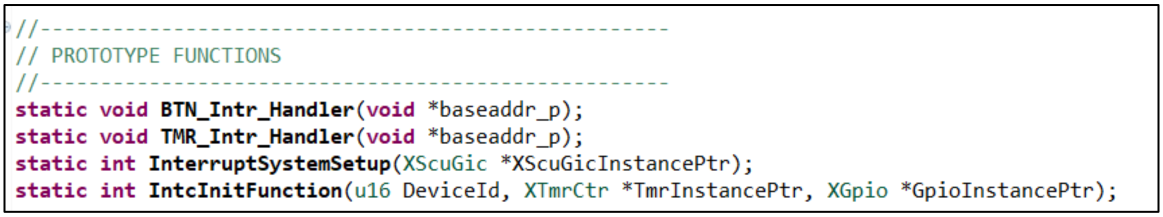
The program begins by initializing the necessary libraries and mapping Gpio's to their respective definitions:



```
 * Robert Bara

#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 SW_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
```

Figure 2. Headers and Parameter Definitions

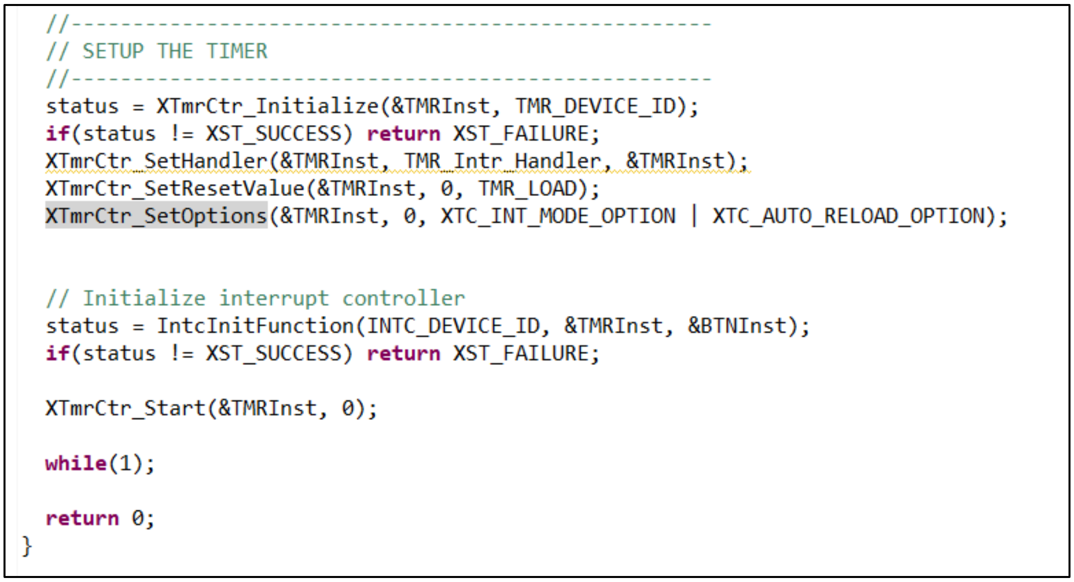
This lab introduces the timer controller header "xtmrctr.h", defines the device ID based upon the Vivado design, and maps the timer interrupt ID to its respective hardware using #define INTC_TMR_INTERRUPT_ID. Everything else remains from the previous labs to map IDs to hardware. Next, variables are defined that will be used throughout the lab for multiple functions such as the input buttons and switches, as well as an LED delay and volatile delay to count. Function prototypes are as follows, introducing the TMR_Intr_Handler which will control the timer interrupt and is passed in a pointer to a base address:



```
//----------------------------------------------------
// 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);
```

Figure 3. Function prototypes

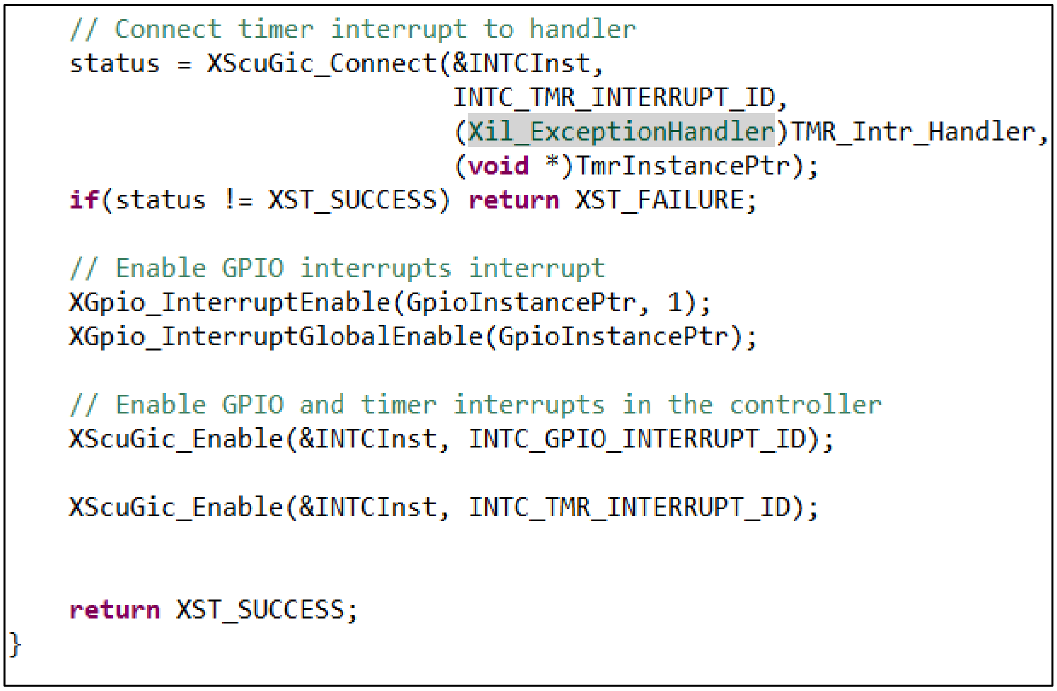
Within main, int status is used to check if the Gpio's are initialized correctly. If each Gpio is successful, then LEDs are set as outputs, while buttons and switches are set as inputs, this can be seen in the appendix. Status now uses the same method to check if the timer control initialization is set up correctly. If the initialization is successful, then the timer control handler sets the TMRInst pointer's address and Timer interrupt handler function. A Timer control reset value is also set using the definition TMR_Load which is set to 0xFA000000 which controls the rate at which the timer controller resets. I set the timer load to be approximately 1 second, so the LEDs increment in 1 second intervals when a valid button and switch interrupt is not generated. Timer controller options are then taken from the "xtmrctr_options.c" program from the timer header file. This maps the function to the timer and uses two macros to set options for the initialization and timer reload. Finally, the interrupt controller is initialized, and the timer controller starts:



```
  //----------------------------------------------------
  // 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;
}
```

Figure 4. Timer Initialization within Main

The interrupt system setup function remains the same as the previous lab, with its' purpose to set up buttons to enable interrupts when needed. The interrupt controller initialization function, however, differs slightly since now the timer is also passed in to generate an interrupt. The function now connects the timer interrupt to the handler using XScuGic_Connect which points to the address of the interrupt from the axi_timer_0 block and then examines the ID of the timer interrupt handler and creates the connection. Finally, the function uses XScuGic_Enable to call the interrupts when generated, based upon the conditions written in the timer and button handler functions:



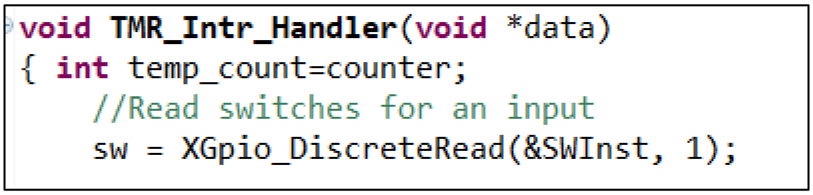
```
    // Connect timer interrupt to handler
    status = XScuGic_Connect(&INTCInst,
                             INTC_TMR_INTERRUPT_ID,
                             (Xil_ExceptionHandler)TMR_Intr_Handler,
                             (void *)TmrInstancePtr);
    if(status != XST_SUCCESS) return XST_FAILURE;

    // 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;
}
```

Figure 5. Connecting the timer interrupt to handler and enabling timer interrupts from the interrupt controller initialization function

The rest of the initialization may be found within the appendix. The Button interrupt and Timer interrupt handler functions will be explained below:

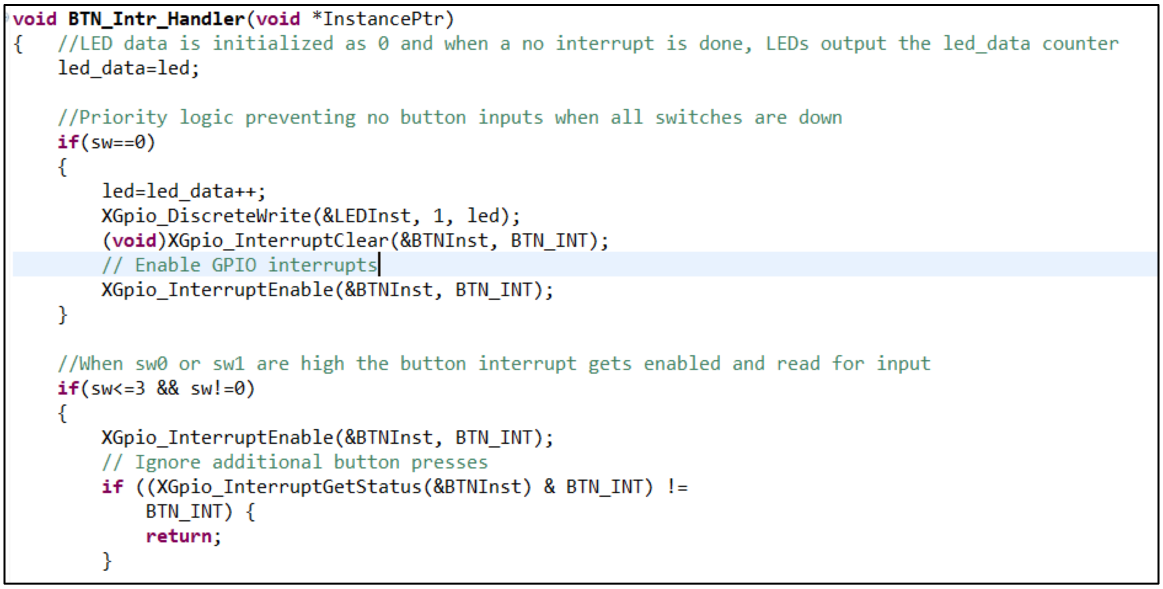
Subroutines

Since the button interrupt handler function works in conjunction with the timer interrupt handler, I think the best way to explain these two functions in terms of the tasks they accomplish. In TMR_Intr_Handler, switches are read, and a temporary count variable is defined to hold the current counter value when sw3 will be high, while in BTN_Intr_Handler button is read using a button debounce to ensure a stable input and the led_data is set to the output of LEDs for overlap reasons. Priority logic increments the LEDs and clears any button inputs when all switches are off. If sw0 or sw1 are high, then button interrupts are enabled and a button debounce system reads the button:



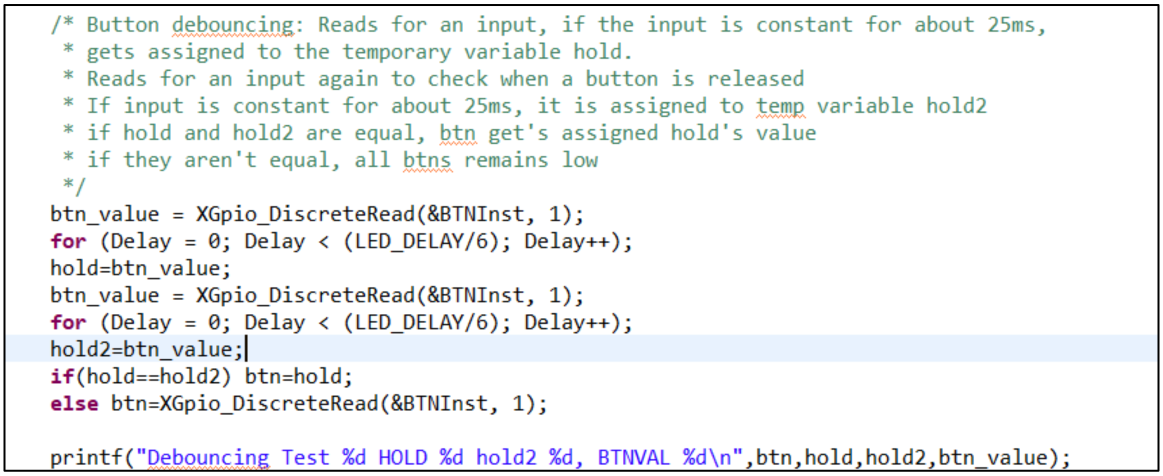
```
void TMR_Intr_Handler(void *data)
{ int temp_count=counter;
    //Read switches for an input
    sw = XGpio_DiscreteRead(&SWInst, 1);
```

Figure 6. Reading Switches which effect Button Interrupt Handler Function



```
void BTN_Intr_Handler(void *InstancePtr)
{   //LED data is initialized as 0 and when a no interrupt is done, LEDs output the led_data counter
    led_data=led;

    //Priority logic preventing no button inputs when all switches are down
    if(sw==0)
    {
        led=led_data++;
        XGpio_DiscreteWrite(&LEDInst, 1, led);
        (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
        // Enable GPIO interrupts
        XGpio_InterruptEnable(&BTNInst, BTN_INT);
    }

    //When sw0 or sw1 are high the button interrupt gets enabled and read for input
    if(sw<=3 && sw!=0)
    {
        XGpio_InterruptEnable(&BTNInst, BTN_INT);
        // Ignore additional button presses
        if ((XGpio_InterruptGetStatus(&BTNInst) & BTN_INT) !=
            BTN_INT) {
            return;
        }
```

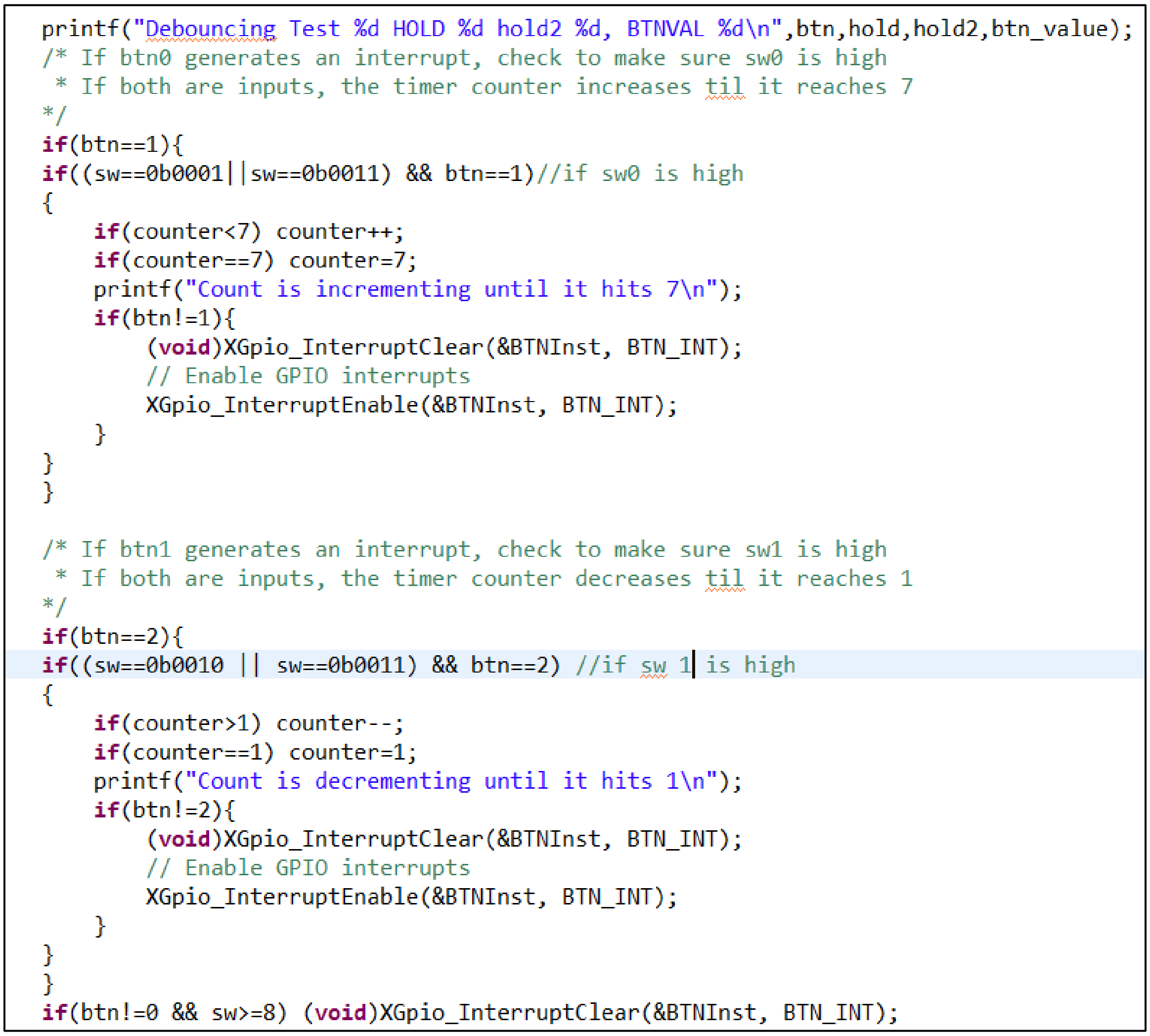
Figure 7. Initializing LEDs, when all sw are off condition. If sw0 or sw1 are high enable interrupts



```
    /* Button debouncing: Reads for an input, if the input is constant for about 25ms,
     * gets assigned to the temporary variable hold.
     * Reads for an input again to check when a button is released
     * If input is constant for about 25ms, it is assigned to temp variable hold2
     * if hold and hold2 are equal, btn get's assigned hold's value
     * if they aren't equal, all btns remains low
     */
    btn_value = XGpio_DiscreteRead(&BTNInst, 1);
    for (Delay = 0; Delay < (LED_DELAY/6); Delay++);
    hold=btn_value;
    btn_value = XGpio_DiscreteRead(&BTNInst, 1);
    for (Delay = 0; Delay < (LED_DELAY/6); Delay++);
    hold2=btn_value;
    if(hold==hold2) btn=hold;
    else btn=XGpio_DiscreteRead(&BTNInst, 1);

    printf("Debouncing Test %d HOLD %d hold2 %d, BTNVAL %d\n",btn,hold,hold2,btn_value);
```

Figure 8. If sw0 or 1 are high, read and perform button debounce

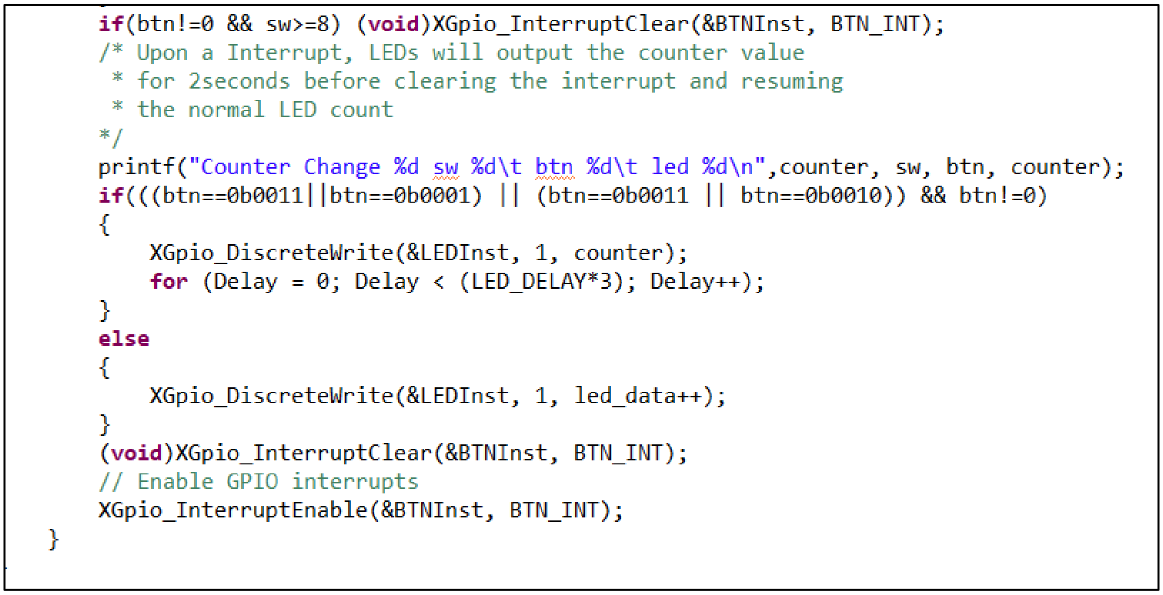
Buttons are then checked and compared to switches. If Sw0 is high and button 0 is pressed then counter will increment for each button interrupt, until it reaches a maximum count of 7. If button does not equal btn0 as an input, clear the interrupt. If Sw1 is high and button 1 is pressed then counter will decrement for each button interrupt, until it reaches a minimum count of 1. If button does not equal btn1 as an input, clear the interrupt. For security reasons If the interrupt is invalid, clear any button interrupts, this makes sure this code is only performed within the button interrupt handler function and ran when an interrupt is generated by the correct sw0 and btn0 or sw1 and btn1 combination:



```
printf("Debouncing Test %d HOLD %d hold2 %d, BTNVAL %d\n",btn,hold,hold2,btn_value);
/* If btn0 generates an interrupt, check to make sure sw0 is high
 * If both are inputs, the timer counter increases til it reaches 7
*/
if(btn==1){
if((sw==0b0001||sw==0b0011) && btn==1)//if sw0 is high
{
    if(counter<7) counter++;
    if(counter==7) counter=7;
    printf("Count is incrementing until it hits 7\n");
    if(btn!=1){
        (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
        // Enable GPIO interrupts
        XGpio_InterruptEnable(&BTNInst, BTN_INT);
    }
}
}

/* If btn1 generates an interrupt, check to make sure sw1 is high
 * If both are inputs, the timer counter decreases til it reaches 1
*/
if(btn==2){
if((sw==0b0010 || sw==0b0011) && btn==2) //if sw 1 is high
{
    if(counter>1) counter--;
    if(counter==1) counter=1;
    printf("Count is decrementing until it hits 1\n");
    if(btn!=2){
        (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
        // Enable GPIO interrupts
        XGpio_InterruptEnable(&BTNInst, BTN_INT);
    }
}
}
if(btn!=0 && sw>=8) (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
```

Figure 9. incrementing or decrementing the counter based on inputs and interrupts

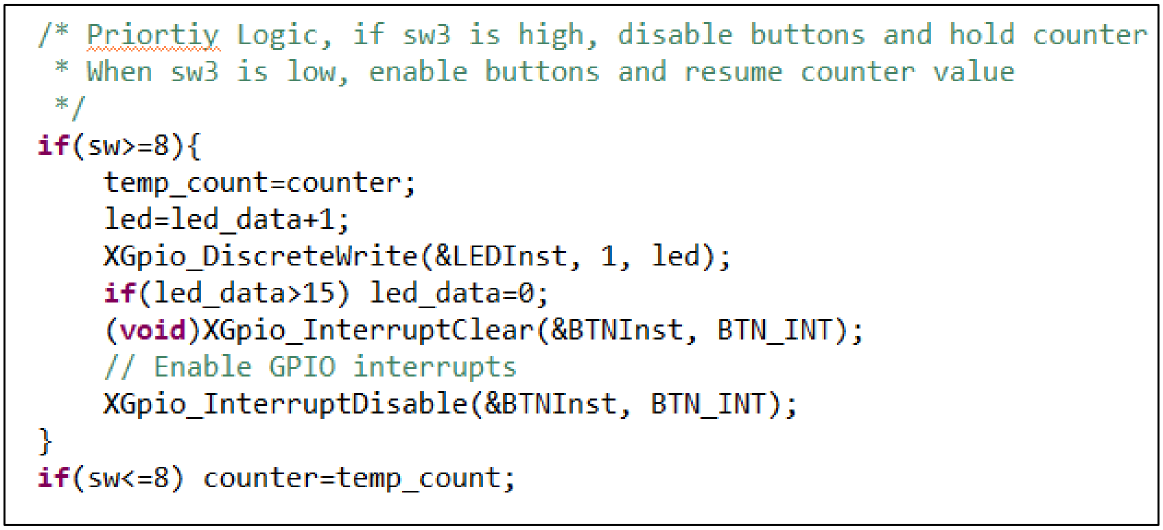
Finally, when a button interrupt increments or decrements the counter, it will be displayed across the LED display for approximately 2 seconds, since the LED_DELAY macro is set to about 1 second, otherwise it will default to display the LED data and increment by 1 as usual:



```
        if(btn!=0 && sw>=8) (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
        /* Upon a Interrupt, LEDs will output the counter value
         * for 2seconds before clearing the interrupt and resuming
         * the normal LED count
        */
        printf("Counter Change %d sw %d\t btn %d\t led %d\n",counter, sw, btn, counter);
        if(((btn==0b0011||btn==0b0001) || (btn==0b0011 || btn==0b0010)) && btn!=0)
        {
            XGpio_DiscreteWrite(&LEDInst, 1, counter);
            for (Delay = 0; Delay < (LED_DELAY*3); Delay++);
        }
        else
        {
            XGpio_DiscreteWrite(&LEDInst, 1, led_data++);
        }
        (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
        // Enable GPIO interrupts
        XGpio_InterruptEnable(&BTNInst, BTN_INT);
    }
```

Figure 10. Outputting to LEDs

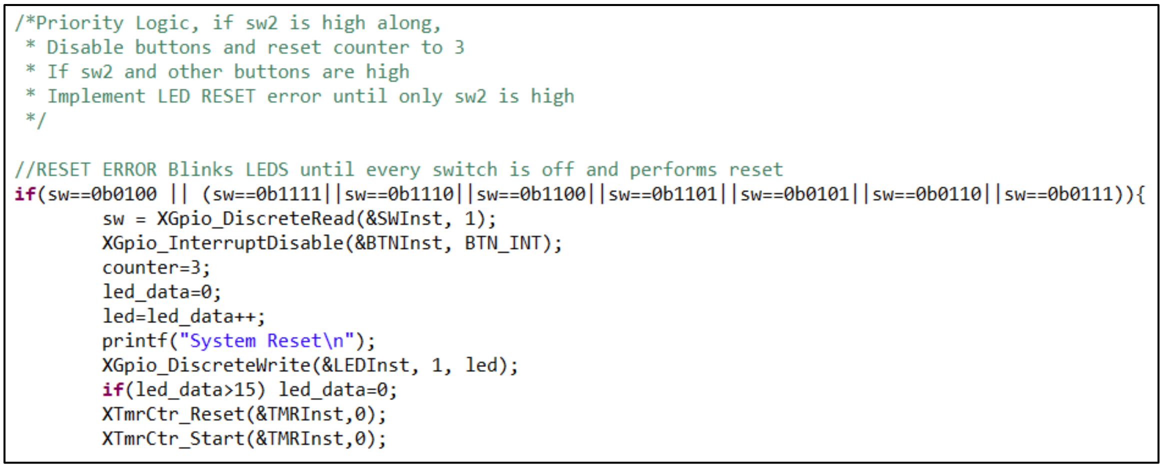
In the Timer interrupt handler, priority logic is done. If sw3 is high and ignoring other switches, then the counter stays the same, buttons are disabled, and LEDs continue to increment, until sw3 goes low:



```
/* Priortiy Logic, if sw3 is high, disable buttons and hold counter
 * When sw3 is low, enable buttons and resume counter value
 */
if(sw>=8){
    temp_count=counter;
    led=led_data+1;
    XGpio_DiscreteWrite(&LEDInst, 1, led);
    if(led_data>15) led_data=0;
    (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
    // Enable GPIO interrupts
    XGpio_InterruptDisable(&BTNInst, BTN_INT);
}
if(sw<=8) counter=temp_count;
```

Figure 1. sw3 priority logic

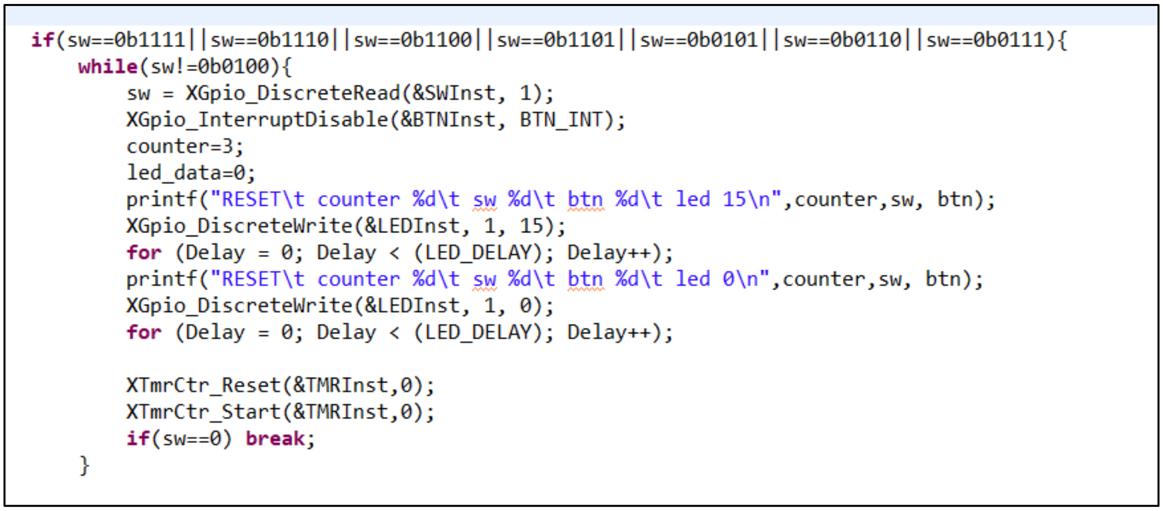
If sw2 is high, then reset the counter to its default value and restart the system:



```c
/*Priority Logic, if sw2 is high along,
 * Disable buttons and reset counter to 3
 * If sw2 and other buttons are high
 * Implement LED RESET error until only sw2 is high
 */

//RESET ERROR Blinks LEDS until every switch is off and performs reset
if(sw==0b0100 || (sw==0b1111||sw==0b1110||sw==0b1100||sw==0b1101||sw==0b0101||sw==0b0110||sw==0b0111)){
        sw = XGpio_DiscreteRead(&SWInst, 1);
        XGpio_InterruptDisable(&BTNInst, BTN_INT);
        counter=3;
        led_data=0;
        led=led_data++;
        printf("System Reset\n");
        XGpio_DiscreteWrite(&LEDInst, 1, led);
        if(led_data>15) led_data=0;
        XTmrCtr_Reset(&TMRInst,0);
        XTmrCtr_Start(&TMRInst,0);
```

Figure 11. System Reset from sw2

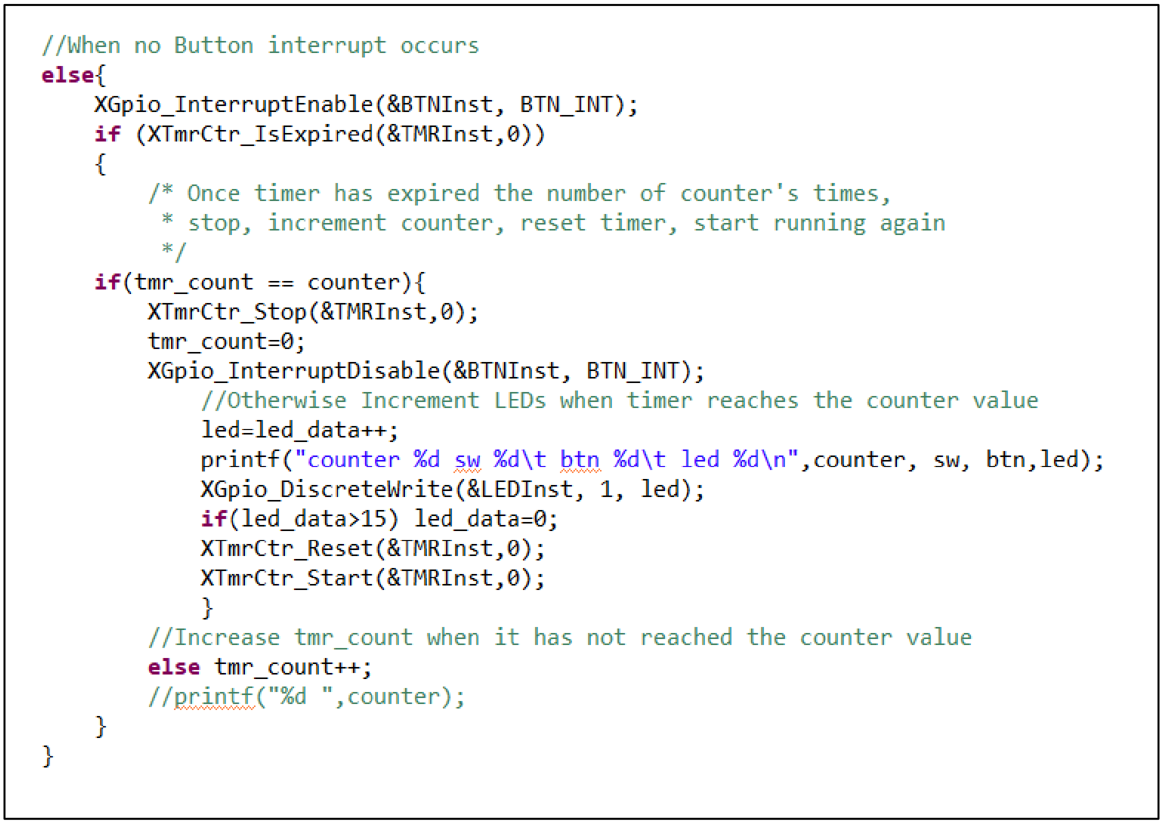
However, if any other switch is on with switch 2, then a reset error occurs, flashing LEDs until all switches are turned off, even if switch 2 is turned off. Once all switches are off, a system reset occurs by breaking back into the previous if statement:



```c
if(sw==0b1111||sw==0b1110||sw==0b1100||sw==0b1101||sw==0b0101||sw==0b0110||sw==0b0111){
    while(sw!=0b0100){
        sw = XGpio_DiscreteRead(&SWInst, 1);
        XGpio_InterruptDisable(&BTNInst, BTN_INT);
        counter=3;
        led_data=0;
        printf("RESET\t counter %d\t sw %d\t btn %d\t led 15\n",counter,sw, btn);
        XGpio_DiscreteWrite(&LEDInst, 1, 15);
        for (Delay = 0; Delay < (LED_DELAY); Delay++);
        printf("RESET\t counter %d\t sw %d\t btn %d\t led 0\n",counter,sw, btn);
        XGpio_DiscreteWrite(&LEDInst, 1, 0);
        for (Delay = 0; Delay < (LED_DELAY); Delay++);

        XTmrCtr_Reset(&TMRInst,0);
        XTmrCtr_Start(&TMRInst,0);
        if(sw==0) break;
    }
```

Figure 12. Reset error if other switches are enable with 2 enabled

Finally, when no button interrupt occurs the LEDs will increment by 1 upon every counter expiration. Otherwise, the counter will increment until the counter value is reached for expiration:



```
//When no Button interrupt occurs
else{
    XGpio_InterruptEnable(&BTNInst, BTN_INT);
    if (XTmrCtr_IsExpired(&TMRInst,0))
    {
        /* Once timer has expired the number of counter's times,
         * stop, increment counter, reset timer, start running again
         */
    if(tmr_count == counter){
        XTmrCtr_Stop(&TMRInst,0);
        tmr_count=0;
        XGpio_InterruptDisable(&BTNInst, BTN_INT);
            //Otherwise Increment LEDs when timer reaches the counter value
            led=led_data++;
            printf("counter %d sw %d\t btn %d\t led %d\n",counter, sw, btn,led);
            XGpio_DiscreteWrite(&LEDInst, 1, led);
            if(led_data>15) led_data=0;
            XTmrCtr_Reset(&TMRInst,0);
            XTmrCtr_Start(&TMRInst,0);
            }
        //Increase tmr_count when it has not reached the counter value
        else tmr_count++;
        //printf("%d ",counter);
    }
}
```

Figure 13. Incrementing LEDs based upon timer counter

Verification

Video Link:

https://www.youtube.com/watch?v=H_dnLdqiZPM&ab_channel=RobertBara

While the video goes through all possible combinations I can think of, here are a few screenshots for verification of the reset error. I am not sure how I could fully screenshot all the button interrupts:

System Reset, when only sw2 is high then the system resets once sw2 goes low:

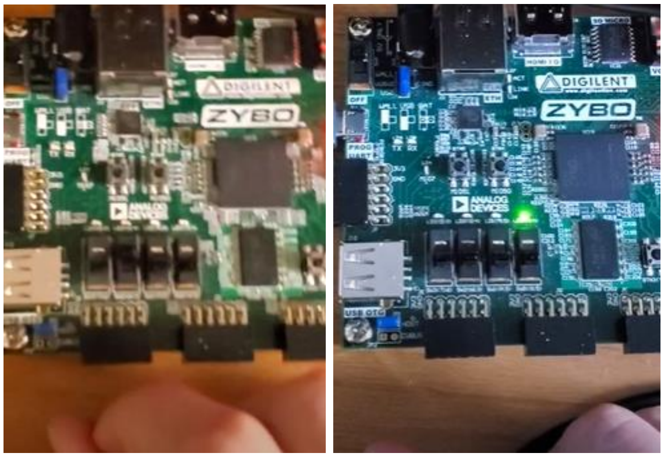


Figure 14a and 14b.System reset before and after switching sw2

Here is the reset error functioning until all switches are off, even sw2 is off and random switches are on. Finally, when all switches are off, a system reset occurs:

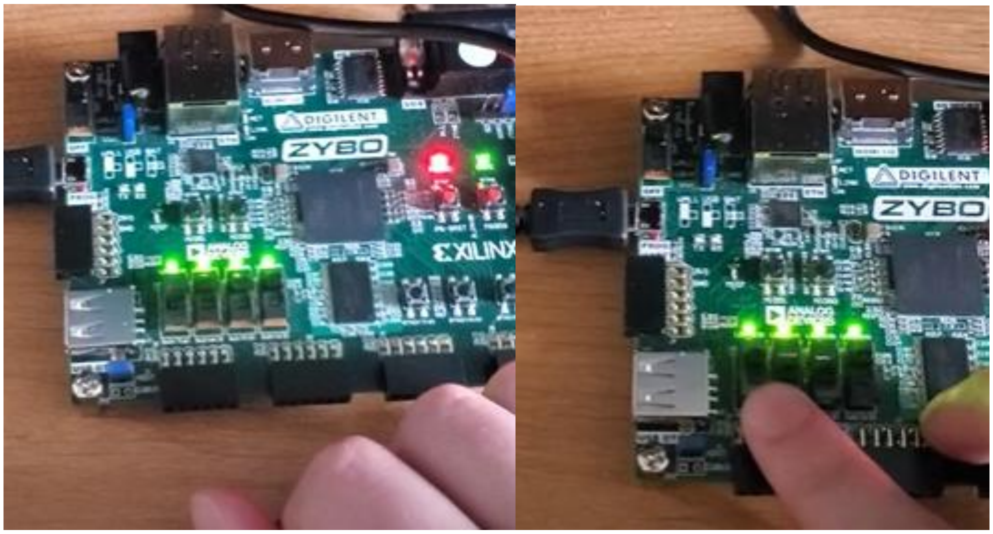


Figure 15a and 15b. Reset errors occur

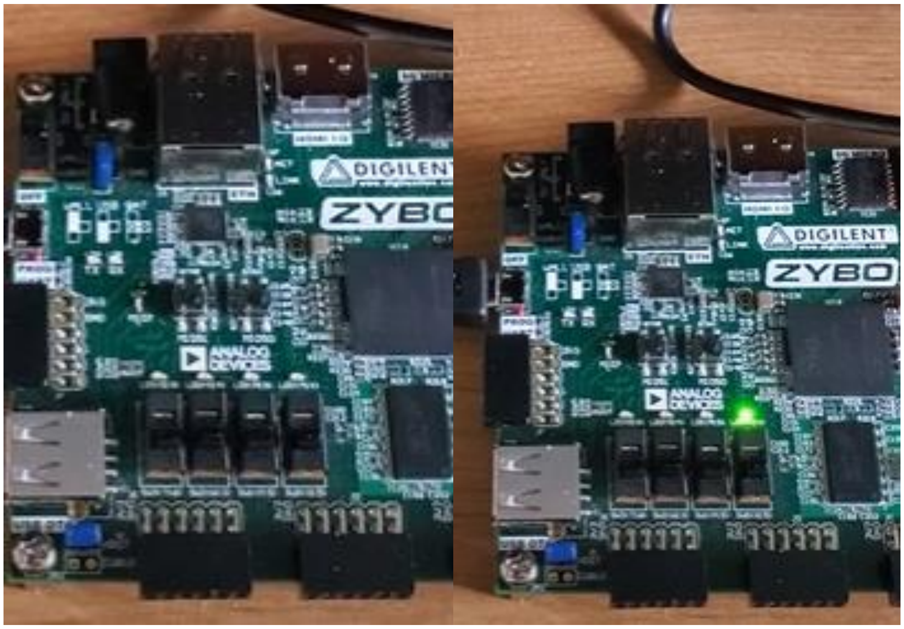


Figure 15c and 15d. System reset after the reset error is shut off

Conclusion

Overall, I found this lab to be a little more challenging than the previous labs. I spent a lot of time understanding how both interrupt handlers interact with each other and while I was able to get the conditions correct for sw0 and sw1 with their respective buttons within the first few days of working on the lab, it took me a few days to understand how the priority logic of sw2 and sw3 interact with each other. I spent a lot of time reading within the header file for the timer interrupt trying to make sense of everything, and eventually managed to perform every task and validate the lab, given I have submitted this 2 days later than the typical due date. I think this lab gave me a decent understanding of how timer interrupts operate, and over the course of the next week I plan on revising this lab to study for the exam, since while I was able to complete all of the tasks given, I do think there are some neater ways to approach the lab than the solution I proposed, and I think I could figure it out given more time.

Appendix

C Code

```
/*
 * Robert Bara
 * Lab4.c
 *
 */

#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 SW_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                 0xFA000000   //delay of 1 second for timer
#define LED_DELAY                50000000     //delay of about 1 second for
interrupts
#define printf                         xil_printf

XGpio LEDInst, BTNInst, SWInst;
XScuGic INTCInst;
XTmrCtr TMRInst;
static int led_data;      //LED count for no interrupt
static int led;                  //What will output across the LEDs
static int btn_value;     //Reads a button
static int hold;          //Temporary Variable for Debounce Press
static int hold2;         //Temporary Variable for Debounce Release
static int btn;                  //Button's input after debouncing
static int tmr_count;     //Timer counter
static int sw;                   //Input Switches
static int counter=3;     //When timer reaches this variable, it expires
volatile int Delay;       //Delay for Button Debouncing

//----------------------------------------------------
// 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)
{      //LED data is initialized as 0 and when a no interrupt is done, LEDs
output the led_data counter
       led_data=led;

       //Priority logic preventing no button inputs when all switches are down
       if(sw==0)
       {
              led=led_data++;
              XGpio_DiscreteWrite(&LEDInst, 1, led);
              (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
              // Enable GPIO interrupts
              XGpio_InterruptEnable(&BTNInst, BTN_INT);
       }

       //When sw0 or sw1 are high the button interrupt gets enabled and read
for input
       if(sw<=3 && sw!=0)
       {
              XGpio_InterruptEnable(&BTNInst, BTN_INT);
              // Ignore additional button presses
              if ((XGpio_InterruptGetStatus(&BTNInst) & BTN_INT) !=
                     BTN_INT) {
                     return;
              }

              /* Button debouncing: Reads for an input, if the input is
constant for about 25ms,
               * gets assigned to the temporary variable hold.
               * Reads for an input again to check when a button is released
               * If input is constant for about 25ms, it is assigned to temp
variable hold2
               * if hold and hold2 are equal, btn get's assigned hold's value
               * if they aren't equal, all btns remains low
               */
              btn_value = XGpio_DiscreteRead(&BTNInst, 1);
              for (Delay = 0; Delay < (LED_DELAY/6); Delay++);
              hold=btn_value;
              btn_value = XGpio_DiscreteRead(&BTNInst, 1);
              for (Delay = 0; Delay < (LED_DELAY/6); Delay++);
              hold2=btn_value;
              if(hold==hold2) btn=hold;
              else btn=XGpio_DiscreteRead(&BTNInst, 1);

              printf("Debouncing Test %d HOLD %d hold2 %d, BTNVAL
%d\n",btn,hold,hold2,btn_value);
              /* If btn0 generates an interrupt, check to make sure sw0 is high
               * If both are inputs, the timer counter increases til it reaches
7
              */
```

```
             if(btn==1){
             if((sw==0b0001||sw==0b0011) && btn==1)//if sw0 is high
             {
                    if(counter<7) counter++;
                    if(counter==7) counter=7;
                    printf("Count is incrementing until it hits 7\n");
                    if(btn!=1){
                           (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
                           // Enable GPIO interrupts
                           XGpio_InterruptEnable(&BTNInst, BTN_INT);
                    }
             }
             }

             /* If btn1 generates an interrupt, check to make sure sw1 is high
              * If both are inputs, the timer counter decreases til it reaches
1
             */
             if(btn==2){
             if((sw==0b0010 || sw==0b0011) && btn==2) //if sw 1 is high
             {
                    if(counter>1) counter--;
                    if(counter==1) counter=1;
                    printf("Count is decrementing until it hits 1\n");
                    if(btn!=2){
                           (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
                           // Enable GPIO interrupts
                           XGpio_InterruptEnable(&BTNInst, BTN_INT);
                    }
             }
             }
             if(btn!=0 && sw>=8) (void)XGpio_InterruptClear(&BTNInst,
BTN_INT);
             /* Upon a Interrupt, LEDs will output the counter value
              * for 2seconds before clearing the interrupt and resuming
              * the normal LED count
             */
             printf("Counter Change %d sw %d\t btn %d\t led %d\n",counter, sw,
btn, counter);
             if(((btn==0b0011||btn==0b0001) || (btn==0b0011 || btn==0b0010))
&& btn!=0)
             {
                    XGpio_DiscreteWrite(&LEDInst, 1, counter);
                    for (Delay = 0; Delay < (LED_DELAY*3); Delay++);
             }
             else
             {
                    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)
{ int temp_count=counter;
    //Read switches for an input
    sw = XGpio_DiscreteRead(&SWInst, 1);

    /* Priortiy Logic, if sw3 is high, disable buttons and hold counter
     * When sw3 is low, enable buttons and resume counter value
     */
    if(sw>=8){
        temp_count=counter;
        led=led_data+1;
        XGpio_DiscreteWrite(&LEDInst, 1, led);
        if(led_data>15) led_data=0;
        (void)XGpio_InterruptClear(&BTNInst, BTN_INT);
        // Enable GPIO interrupts
        XGpio_InterruptDisable(&BTNInst, BTN_INT);
    }
    if(sw<=8) counter=temp_count;


    /*Priority Logic, if sw2 is high along,
     * Disable buttons and reset counter to 3
     * If sw2 and other buttons are high
     * Implement LED RESET error until only sw2 is high
     */

    //RESET ERROR Blinks LEDS until every switch is off and performs reset
    if(sw==0b0100 ||
(sw==0b1111||sw==0b1110||sw==0b1100||sw==0b1101||sw==0b0101||sw==0b0110||sw==0
b0111)){
            sw = XGpio_DiscreteRead(&SWInst, 1);
            XGpio_InterruptDisable(&BTNInst, BTN_INT);
            counter=3;
            led_data=0;
            led=led_data++;
            printf("System Reset\n");
            XGpio_DiscreteWrite(&LEDInst, 1, led);
            if(led_data>15) led_data=0;
            XTmrCtr_Reset(&TMRInst,0);
            XTmrCtr_Start(&TMRInst,0);


    if(sw==0b1111||sw==0b1110||sw==0b1100||sw==0b1101||sw==0b0101||sw==0b011
0||sw==0b0111){
            while(sw!=0b0100){
                sw = XGpio_DiscreteRead(&SWInst, 1);
                XGpio_InterruptDisable(&BTNInst, BTN_INT);
                counter=3;
                led_data=0;
                printf("RESET\t counter %d\t sw %d\t btn %d\t led
15\n",counter,sw, btn);
                XGpio_DiscreteWrite(&LEDInst, 1, 15);
                for (Delay = 0; Delay < (LED_DELAY); Delay++);
                printf("RESET\t counter %d\t sw %d\t btn %d\t led
0\n",counter,sw, btn);
```

```
                        XGpio_DiscreteWrite(&LEDInst, 1, 0);
                        for (Delay = 0; Delay < (LED_DELAY); Delay++);

                        XTmrCtr_Reset(&TMRInst,0);
                        XTmrCtr_Start(&TMRInst,0);
                        if(sw==0) break;
                  }
      }
      }

      //When no Button interrupt occurs
      else{
            XGpio_InterruptEnable(&BTNInst, BTN_INT);
            if (XTmrCtr_IsExpired(&TMRInst,0))
            {
                  /* Once timer has expired the number of counter's times,
                   * stop, increment counter, reset timer, start running
again
                   */
            if(tmr_count == counter){
                  XTmrCtr_Stop(&TMRInst,0);
                  tmr_count=0;
                  XGpio_InterruptDisable(&BTNInst, BTN_INT);
                        //Otherwise Increment LEDs when timer reaches the
counter value
                        led=led_data++;
                        printf("counter %d sw %d\t btn %d\t led
%d\n",counter, sw, btn,led);
                        XGpio_DiscreteWrite(&LEDInst, 1, led);
                        if(led_data>15) led_data=0;
                        XTmrCtr_Reset(&TMRInst,0);
                        XTmrCtr_Start(&TMRInst,0);
                        }
                  //Increase tmr_count when it has not reached the counter
value
                  else tmr_count++;
                  //printf("%d ",counter);
            }
      }
}

//------------------------------------------------------
// 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 Switches
  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);
  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;
}
```