```c
/* interrupt_counter_tut_2D_new.c */
/* Created on:  Unknown
 *  Author:     Ross Elliot
 *  Version:        1.1  */
/***************************
* VERSION HISTORY
*   v2.0 - 01/06/2020 by Jianjian Song
*       Revised with a number of calls and remove all warnings.
*   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"

// Device ID and Interrupt ID 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_MASK            XGPIO_IR_CH1_MASK
//Timer load value:  Count up timer (0xFFFFFFFF - 0xF8000000)=0x7FFFFF=134217727
//134217727/100MHz = 134217727*10ns=1342177270ns = 1.34217727 (seconds)
#define TMR_LOAD            0xF8000000

//Four Device definitions
XGpio LEDInst, BTNInst;
```

```
45 XScuGic INTCInst;
46 XTmrCtr TMRInst;
47
48 //global variable
49 int led_data;
50
51 // Function Prototypes
52 static int SetupLEDs(XGpio *LEDdevice, int DeviceID);
53 static int SetupPushbuttons(XGpio *Pushbuttondevice, int DeviceID);
54 static void BTN_Intr_Handler(void *baseaddr_p);
55 static void TMR_Intr_Handler(void *baseaddr_p);
56 static int GICconfiguration(u16 DeviceId, XTmrCtr *TmrInstancePtr, XGpio *GpioInstancePtr);
57
58 void BTN_Intr_Handler(void *InstancePtr)
59 {
60     int btn_value, led_data;    //local and temporary variables
61     XGpio_InterruptDisable(&BTNInst, BTN_INT_MASK);
62     // Ignore additional button presses
63     if ((XGpio_InterruptGetStatus(&BTNInst) & BTN_INT_MASK) != BTN_INT_MASK) {  return;  }
64     btn_value = XGpio_DiscreteRead(&BTNInst, 1);
65     // Increment counter based on button value
66     //button change on both edges will cause interrupt but only button value = 1 will be active.
67     led_data = led_data + btn_value;
68     XGpio_DiscreteWrite(&LEDInst, 1, led_data);
69     XGpio_InterruptClear(&BTNInst, BTN_INT_MASK);
70     XGpio_InterruptEnable(&BTNInst, BTN_INT_MASK);
71 } //end BTN_Intr_Handler()
72
73 //this code will NOT work as it was intended because the interrupt flag will not be cleared
74 //unless tmr_count==3. Therefore, this handler will be called three times for each interrupt
75 void TMR_Intr_Handler(void *data)
76 {
77     static int tmr_count;   //local and perminent variable
78     if (XTmrCtr_IsExpired(&TMRInst,0)){
```

```
79// Once timer has expired 3 times, stop, increment counter. Reset timer and start running again
80        if(tmr_count == 3){
81            XTmrCtr_Stop(&TMRInst,0);
82            tmr_count = 0;
83            led_data++;
84            XGpio_DiscreteWrite(&LEDInst, 1, led_data);
85            XTmrCtr_Reset(&TMRInst,0);
86            XTmrCtr_Start(&TMRInst,0);
87        }
88        else tmr_count++;
89    }
90}   //end TMR_Intr_Handler
91
92int SetupLEDs(XGpio *LEDdevice, int DeviceID) {
93    int status;
94    // Initialize LEDs
95      status = XGpio_Initialize(LEDdevice, DeviceID);
96      if(status != XST_SUCCESS) return XST_FAILURE;
97      // Set LEDs direction to outputs
98        XGpio_SetDataDirection(&LEDInst, 1, 0x00);
99        return XST_SUCCESS;
100} //end SetupLEDs()
101
102int SetupPushbuttons(XGpio *Pushbuttondevice, int DeviceID) {
103    int status;
104    // Initialize Push Buttons
105      status = XGpio_Initialize(Pushbuttondevice, DeviceID);
106      if(status != XST_SUCCESS) return XST_FAILURE;
107
108      // Set all buttons direction to inputs
109      XGpio_SetDataDirection(Pushbuttondevice, 1, 0xFF);
110      // Level 3: Enable GPIO interrupts interrupt - JJS
111        XGpio_InterruptGlobalEnable(Pushbuttondevice);
112      //Level 3: channel 1 only
```

```
113         XGpio_InterruptEnable(Pushbuttondevice, BTN_INT_MASK);
114         return XST_SUCCESS;
115 } //end SetupPushbuttons()
116
117 int SetupTimer(XTmrCtr *Timerdevice, int DeviceID) {
118     int status;
119       status = XTmrCtr_Initialize(Timerdevice, DeviceID);
120       if(status != XST_SUCCESS) return XST_FAILURE;
121       //Level 3: timer. There is a warning on type mismatch for the handler but the statement works.
122       XTmrCtr_SetHandler(Timerdevice, (XTmrCtr_Handler) TMR_Intr_Handler, Timerdevice);
123       XTmrCtr_SetResetValue(Timerdevice, 0, TMR_LOAD);
124       XTmrCtr_SetOptions(Timerdevice, 0, XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);
125       //Start timer
126       XTmrCtr_Start(Timerdevice, 0);
127       return XST_SUCCESS;
128 } //end SetupTimer()
129
130 //GIC Level 2 and Cortex A9 CPU Level 1configuration
131 int GICconfiguration(u16 DeviceId, XTmrCtr *TmrInstancePtr, XGpio *GpioInstancePtr)
132 {
133     XScuGic_Config *IntcConfig;
134     int status;
135 // Level 2: Generic Interrupt controller (GIC) initialization
136     IntcConfig = XScuGic_LookupConfig(DeviceId);
137     status = XScuGic_CfgInitialize(&INTCInst, IntcConfig, IntcConfig->CpuBaseAddress);
138     if(status != XST_SUCCESS) return XST_FAILURE;
139 // Level 2: Connect GPIO interrupt to handler
140     status = XScuGic_Connect(&INTCInst,
141         INTC_GPIO_INTERRUPT_ID, (Xil_ExceptionHandler)BTN_Intr_Handler, (void *)GpioInstancePtr);
142     if(status != XST_SUCCESS) return XST_FAILURE;
143 // Level 2: Connect timer interrupt to handler
144     status = XScuGic_Connect(&INTCInst,
145          INTC_TMR_INTERRUPT_ID,
146      (Xil_ExceptionHandler)TMR_Intr_Handler,(void *)TmrInstancePtr);
```

```
147     if(status != XST_SUCCESS) return XST_FAILURE;
148     // Level 2: Enable GPIO and timer interrupts in the controller - JJS
149     XScuGic_Enable(&INTCInst, INTC_GPIO_INTERRUPT_ID);
150     XScuGic_Enable(&INTCInst, INTC_TMR_INTERRUPT_ID);
151 //Level 1: to assign GIC handler to IRQ vector of Cortex A9 CPU
152         Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT, (Xil_ExceptionHandler)XScuGic_InterruptHandler,
    &INTCInst);
153         Xil_ExceptionEnable();  //this will enable IRQ interrupt
154     return XST_SUCCESS;
155 } //end GICconfiguration()
156
157 int main (void)
158 {
159   int status;
160   // Initialize LEDs
161   status = SetupLEDs(&LEDInst, LEDS_DEVICE_ID);
162   if(status != XST_SUCCESS) return XST_FAILURE;
163
164   // Initialize Push Buttons
165   status = SetupPushbuttons(&BTNInst, BTNS_DEVICE_ID);
166   if(status != XST_SUCCESS) return XST_FAILURE;
167
168   // initialize timer
169   status = SetupTimer(&TMRInst, TMR_DEVICE_ID);
170   if(status != XST_SUCCESS) return XST_FAILURE;
171
172   // Level 1 and 2: Initialize generic interrupt controller GIC and Cortex A9
173   status = GICconfiguration(INTC_DEVICE_ID, &TMRInst, &BTNInst);
174   if(status != XST_SUCCESS) return XST_FAILURE;
175   while(1); //idle forever
176   return 0;
177 } //end main()
178
```