```
/*
 * helloworld.c: simple test application
 *
 * This application configures UART 16550 to baud rate 9600.
 * PS7 UART (Zynq) is not initialized by this application, since
 * bootrom/bsp configures it to baud rate 115200
 *
 * ------------------------------------------------
 * | UART TYPE   BAUD RATE                        |
 * ------------------------------------------------
 *   uartns550   9600
 *   uartlite    Configurable only in HW design
 *   ps7_uart    115200 (configured by bootrom/bsp)
 */
/* Include Files */
#include <stdio.h>     /* Contains the standard input and output functions */
#include "platform.h"  /* Defines init and tidy up routines for MicroBlaze */
#include "xil_printf.h"
#include "xparameters.h" /* Contains address & configurations system needs */
#include "xtmrctr.h"
#include "xgpio.h"     /* General Purpose I/O */
#include "xstatus.h"   /* Contains the status definitions used */
#include "xintc.h"     /* Contains API for using the AI interrupt controller */
#include "xil_exception.h"   /* Contains the API for MicroBlaze exceptions */

/* Definitions */
#define GPIO_DEVICE0_ID XPAR_AXI_GPIO_0_DEVICE_ID /* GPIO dev for LEDs & btns */
```

```
#define GPIO_DEVICE1_ID XPAR_AXI_GPIO_1_DEVICE_ID /* GPIO dev for 7-seg disp */
#define LED 0x0000           /* Initial LED value */
#define CAT 0x00             /* Initial 7-segment display cathode value */
#define DELAY_7SEG 100       /* Delay between anodes of 7-segment display */
#define DELAY_DISP 1000      /* Delay between updates to displays */
#define BTN_CHANNEL 1        /* GPIO port for pushbuttons */
#define LED_CHANNEL 2        /* GPIO port for LEDs */
#define ANODE_CHANNEL 1      /* GPIO port for 7-segment display anodes */
#define CATHODE_CHANNEL 2    /* GPIO port for 7-segment display cathodes */
#define printf xil_printf    /* A smaller, optimized printf */
#define TMRCTR_DEVICE_ID XPAR_TMRCTR_0_DEVICE_ID
#define TMRCTR_INTERRUPT_ID XPAR_INTC_0_TMRCTR_0_VEC_ID
#define INTC_DEVICE_ID XPAR_INTC_0_DEVICE_ID
#define TIMER_CNTR_0 0
#define INTC XIntc
#define INTC_HANDLER XIntc_InterruptHandler
#define RESET_VALUE 0xFFFE795F

/* Global Variables */
XGpio Gpio0;                 /* GPIO Device driver instance 1 */
XGpio Gpio1;                 /* GPIO Device driver instance 2 */
INTC InterruptController;    /* The instance of the Interrupt Controller */
XTmrCtr TimerCounterInst;    /* The instance of the Timer Counter */
volatile int numMillisec = 0;
volatile int whichDigit = 0;
volatile int led = LED;      /* Hold current LED value. */
volatile int cath[4];        /* Hold current cathode value. */
volatile int Freeze = 0;

/* Function Declarations */
void lookup7Seg(void);
int Lab4Timer(INTC* IntcInstancePtr,XTmrCtr* TmrCtrInstancePtr,
           u16 DeviceId,u16 IntrId,u8 TmrCtrNumber);
void TimerCounterHandler(void *CallBackRef, u8 TmrCtrNumber);
static int TmrCtrSetupIntrSystem(INTC* IntcInstancePtr,
           XTmrCtr* TmrCtrInstancePtr,u16 DeviceId,u16 IntrId,u8 TmrCtrNumber);
void TmrCtrDisableIntr(INTC* IntcInstancePtr, u16 IntrId);


int main() {
    init_platform();
    int Status;
    Status = Lab4Timer(&InterruptController, &TimerCounterInst, TMRCTR_DEVICE_ID,
TMRCTR_INTERRUPT_ID, TIMER_CNTR_0);
    if (Status != XST_SUCCESS) {
      xil_printf("Timer Interrupt Counter Failed!\r\n");
      return XST_FAILURE;
    }
    return XST_SUCCESS;
}


void lookup7Seg(void) {
      int i, digit, num;
      num = led;
      for (i=0; i<4; i++) {
            digit = num % 10;
            if (digit==0)      { cath[i] =   63; }    /* 0x3F = 0b00111111 */
            else if (digit==1) { cath[i] =   6; }    /* 0x =0b00000000 */
```

```
            else if (digit==2) { cath[i] =   91; }    /* 0x =0b00000000 */
            else if (digit==3) { cath[i] =   79; }    /* 0x =0b00000000 */
            else if (digit==4) { cath[i] =   102; }   /* 0x =0b00000000 */
            else if (digit==5) { cath[i] =   109; }   /* 0x =0b00000000 */
            else if (digit==6) { cath[i] =   125; }   /* 0x =0b00000000 */
            else if (digit==7) { cath[i] =   7; }     /* 0x =0b00000000 */
            else if (digit==8) { cath[i] =   127; }   /* 0x =0b00000000 */
            else if (digit==9) { cath[i] =   111; }   /* 0x =0b00000000 */
            else               { cath[i] =   0; }     /* 0x00=0b00000000 */
            num /= 10;
      }
      return;
}

int Lab4Timer(INTC* IntcInstancePtr,XTmrCtr* TmrCtrInstancePtr,u16 DeviceId,u16
IntrId,u8 TmrCtrNumber) {
      int Status, Mode = 0, Pressed = 0;
      uint8_t btn, last_btn=0x00; /* Hold current & last pushbutton values */
      Status = XTmrCtr_Initialize(TmrCtrInstancePtr, DeviceId);
      if (Status != XST_SUCCESS) { return XST_FAILURE; }
      Status = XTmrCtr_SelfTest(TmrCtrInstancePtr, TmrCtrNumber);
      if (Status != XST_SUCCESS) { return XST_FAILURE; }
      Status = TmrCtrSetupIntrSystem(IntcInstancePtr,TmrCtrInstancePtr,DeviceId,
IntrId,TmrCtrNumber);
      if (Status != XST_SUCCESS) { return XST_FAILURE; }
      XTmrCtr_SetHandler(TmrCtrInstancePtr,TimerCounterHandler,
TmrCtrInstancePtr);
      XTmrCtr_SetOptions(TmrCtrInstancePtr,TmrCtrNumber,
XTC_INT_MODE_OPTION | XTC_AUTO_RELOAD_OPTION);
      XTmrCtr_SetResetValue(TmrCtrInstancePtr, TmrCtrNumber, RESET_VALUE);
      XTmrCtr_Start(TmrCtrInstancePtr, TmrCtrNumber);
      /* GPIO driver initialization */
      Status = XGpio_Initialize(&Gpio0, GPIO_DEVICE0_ID);
      Status = XGpio_Initialize(&Gpio1, GPIO_DEVICE1_ID);
      if (Status != XST_SUCCESS) { return XST_FAILURE; }
      /* Set the direction for the LEDs to output */
      XGpio_SetDataDirection(&Gpio0, BTN_CHANNEL, 0xFF);
      XGpio_SetDataDirection(&Gpio0, LED_CHANNEL, 0x00);
      XGpio_SetDataDirection(&Gpio1, ANODE_CHANNEL, 0x00);
      XGpio_SetDataDirection(&Gpio1, CATHODE_CHANNEL, 0x00);
      xil_printf("Entering while(1).\r\n");
      while (1) {
            btn = XGpio_DiscreteRead(&Gpio0, BTN_CHANNEL);
            if (btn == 4 && Mode == 1 && Pressed == 0){
                  Mode = 0;
                  Pressed = 1;
                  XTmrCtr_Start(TmrCtrInstancePtr, TmrCtrNumber);
            }
            else if (btn==4 && Mode == 0 && Pressed == 0){
                  Mode = 1;
                  Pressed = 1;
                  XTmrCtr_Stop(TmrCtrInstancePtr, TmrCtrNumber);
            }
            if (btn != 4){
                  Pressed = 0;
            }
            xil_printf("Button: %2d Mode = %2d Pressed = %2d
\r\n",btn,Mode,Pressed);
      }
```

```
	xil_printf("Exiting while(1).\r\n");
	TmrCtrDisableIntr(IntcInstancePtr, DeviceId);
	return XST_SUCCESS;
}

void TimerCounterHandler(void *CallBackRef, u8 TmrCtrNumber) {
	XTmrCtr *InstancePtr = (XTmrCtr *)CallBackRef;
	if (XTmrCtr_IsExpired(InstancePtr, TmrCtrNumber)) {
		if (numMillisec == 0) {
			XGpio_DiscreteWrite(&Gpio0, LED_CHANNEL, led);
			lookup7Seg();
			led++;
		}
		numMillisec = (numMillisec+1) % 100;
		if (whichDigit==0) {
			XGpio_DiscreteWrite(&Gpio1,ANODE_CHANNEL,~(0x1));
			XGpio_DiscreteWrite(&Gpio1,CATHODE_CHANNEL,~(cath[0]));
		} else if (whichDigit==1) {
			XGpio_DiscreteWrite(&Gpio1,ANODE_CHANNEL,~(0x2));
			XGpio_DiscreteWrite(&Gpio1,CATHODE_CHANNEL,~(cath[1]|0x80));
		} else if (whichDigit==2) {
			XGpio_DiscreteWrite(&Gpio1,ANODE_CHANNEL,~(0x4));
			XGpio_DiscreteWrite(&Gpio1,CATHODE_CHANNEL,~(cath[2]));
		} else if (whichDigit==3) {
			XGpio_DiscreteWrite(&Gpio1,ANODE_CHANNEL,~(0x8));
			XGpio_DiscreteWrite(&Gpio1,CATHODE_CHANNEL,~(cath[3]));
		}
		whichDigit = (whichDigit+1) % 4;
	}
}

static int TmrCtrSetupIntrSystem(INTC* IntcInstancePtr,
XTmrCtr* TmrCtrInstancePtr,u16 DeviceId,u16 IntrId,u8 TmrCtrNumber) {
	int Status;
	Status = XIntc_Initialize(IntcInstancePtr, INTC_DEVICE_ID);
	if (Status != XST_SUCCESS) { return XST_FAILURE; }
	Status = XIntc_Connect(IntcInstancePtr, IntrId,
		(XInterruptHandler)XTmrCtr_InterruptHandler,
(void *)TmrCtrInstancePtr);
	if (Status != XST_SUCCESS) { return XST_FAILURE; }
	Status = XIntc_Start(IntcInstancePtr, XIN_REAL_MODE);
	if (Status != XST_SUCCESS) { return XST_FAILURE; }
	XIntc_Enable(IntcInstancePtr, IntrId); /* Enable interrupt for timer ctr */
	Xil_ExceptionInit();		/* Initialize the exception table. */
	Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
		(Xil_ExceptionHandler) INTC_HANDLER,IntcInstancePtr);
	Xil_ExceptionEnable();		/* Enable non-critical exceptions. */
	return XST_SUCCESS;
}
void TmrCtrDisableIntr(INTC* IntcInstancePtr, u16 IntrId) {
	/* Disable the interrupt for the timer counter */
	XIntc_Disable(IntcInstancePtr, IntrId);
	return;
}
```