Software Timer in FreeRTOS

Benjamin A. Blouin

benjamin.blouin@temple.edu

Summary

This lab is an exploration of the FreeRTOS software timer. It uses two GPIO blocks; the buttons and switches as independent channels on a block, and the LEDs on their own block. The button choice lights up the corresponding LED for four seconds. The switches augment the behavior by inverting the button switch combination to display the other LEDs instead of the original button choice-led combination. The time in FreeRTOS has a workflow similar to tasks, but the behavior is different. In the end, the lab was success.

Introduction

The FreeRTOS software time has a similar workflow and implementation to tasks. There are two types of timer behavior settings: auto-reload starts the timer over after each timeout, while the one-shot timer runs once, and then it becomes dormant. Timers are useful in that they run without using computational clock cycles like creating a manual timer counter would.

Discussion

The block design is like the previous labs, as well as using the SDK. The previous “Hello, World!” lab included a timer, so this can be used to help as a template. The timers and GPIO libraries must be included. A Ticktype_t variable must be made to ensure a four second period for the one-shot timer. A xTimerHandle object must be created, and it is used when creating the task through the xTimerCreate macro. Setting the third argument to false makes the timer a one-shot, and the last argument is the timer callback function, which is invoked on timeout. The logic happens in the task, LED output is created, and the timer is used to wait four seconds, and then clear the LEDs. This is the entirety of the algorithm.

Conclusions

This lab is a simple exploration of the FreeRTOS one-shot timer. Starting from scratch, this lab is tough, but utilizing things learned from previous labs helped make this lab easy. Learning to use the timers for embedded applications is essential to real-world success.

Appendices

GitHub

https://github.com/3keepmovingforward3/Embedded-System-Design-Sp19/blob/master/software_timer_in_free_rtos/software_timer_in_free_rtos.sdk/lab9/src/main.c

Pictures

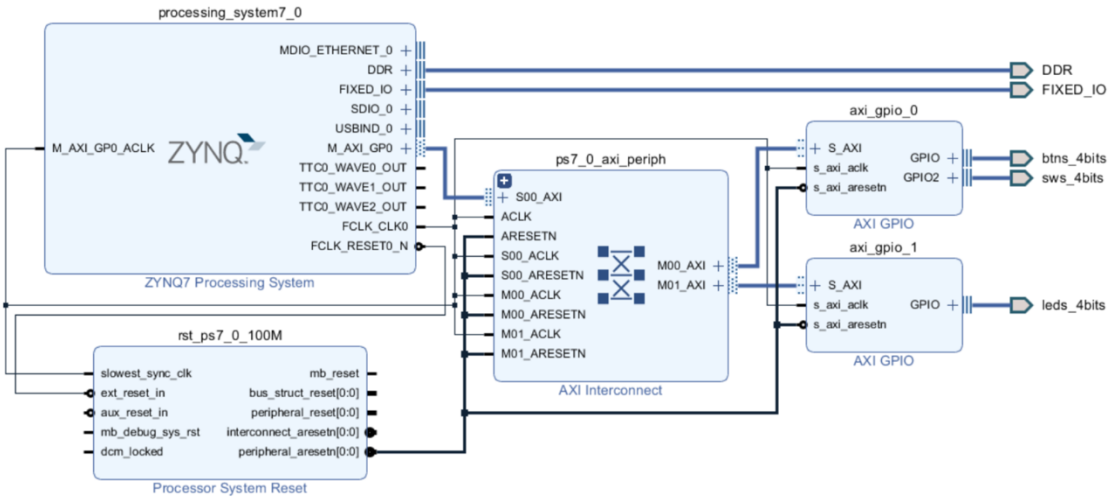

Figure 1 Block Diagram

Code

```
/*
 * main.c
 *
 *  Created on: Apr 17, 2019
 *      Author: bblouin
 */


// FreeRTOS includes
#include "FreeRTOS.h"
#include <task.h>
#include <timers.h>
#include <stdio.h>


// Xilinx includes
#include <xgpio.h>
#include <xil_printf.h>


// Preprocessor Definitions
#define TIMER_ID 1
#define BUTTONS 1
```

```
#define SWITCHES 2
#define FOURSEC 4000UL

static TickType_t four = pdMS_TO_TICKS(FOURSEC); // 4 seconds
static TaskHandle_t Task_BTN_SW = NULL; // Taskhandle object
static XGpio xBTNSWI, xLEDS; // gpio objects
static xTimerHandle xtimer1; // timer handle

// Function Prototypes
static void prvTask_BTN_SW(void *pvParameters); // Task
static void vTimer1Callback(TimerHandle_t pxTimer); // Timer
static void gpioInit(); //gpio initialization

int main (void){

        gpioInit();
        xTaskCreate(prvTask_BTN_SW,
                        (const char *) "TASK_BTN_SW",
                        configMINIMAL_STACK_SIZE,
                        NULL,
                        tskIDLE_PRIORITY + 4,
                        &Task_BTN_SW);

        xtimer1 = xTimerCreate((const char *)"Timer1",
                        four,
                        pdFALSE,
                        (void *) TIMER_ID,
                        vTimer1Callback);

        if(xtimer1!=NULL){
                        xTimerStart(xtimer1,0);
                        vTaskStartScheduler();
        }
```

```
		while(1){};
}

// Task
static void prvTask_BTN_SW(void *pvParameters){
		while(1){

		if(XGpio_DiscreteRead(&xBTNSWI,BUTTONS)==1&&XGpio_DiscreteRead(&xBTNSWI,SWITCHES)!=1){
					xTimerStart(xtimer1,0UL);
					XGpio_DiscreteWrite(&xLEDS,1,1);
			}
			else
if(XGpio_DiscreteRead(&xBTNSWI,BUTTONS)==2&&XGpio_DiscreteRead(&xBTNSWI,SWITCHES)!=2){
					xTimerStart(xtimer1,0UL);
					XGpio_DiscreteWrite(&xLEDS,1,2);
			}
			else
if(XGpio_DiscreteRead(&xBTNSWI,BUTTONS)==4&&XGpio_DiscreteRead(&xBTNSWI,SWITCHES)!=4){
					xTimerStart(xtimer1,0UL);
					XGpio_DiscreteWrite(&xLEDS,1,4);
			}
			else
if(XGpio_DiscreteRead(&xBTNSWI,BUTTONS)==8&&XGpio_DiscreteRead(&xBTNSWI,SWITCHES)!=8){
					xTimerStart(xtimer1,0UL);
					XGpio_DiscreteWrite(&xLEDS,1,8);
			}
			else
if(XGpio_DiscreteRead(&xBTNSWI,BUTTONS)==1&&XGpio_DiscreteRead(&xBTNSWI,SWITCHES)==1){
					xTimerStart(xtimer1,0UL);
					XGpio_DiscreteWrite(&xLEDS,1,14);
			}
			else
if(XGpio_DiscreteRead(&xBTNSWI,BUTTONS)==2&&XGpio_DiscreteRead(&xBTNSWI,SWITCHES)==2){
					xTimerStart(xtimer1,0UL);
```

```
                        XGpio_DiscreteWrite(&xLEDS,1,13);
                    }
                    else
if(XGpio_DiscreteRead(&xBTNSWI,BUTTONS)==4&&XGpio_DiscreteRead(&xBTNSWI,SWITCHES)==4){
                        xTimerStart(xtimer1,0UL);
                        XGpio_DiscreteWrite(&xLEDS,1,11);
                    }
                    else
if(XGpio_DiscreteRead(&xBTNSWI,BUTTONS)==8&&XGpio_DiscreteRead(&xBTNSWI,SWITCHES)==8){
                        xTimerStart(xtimer1,0UL);
                        XGpio_DiscreteWrite(&xLEDS,1,7);
                    }
            }
}

// Timer
static void vTimer1Callback(xTimerHandle pxTimer){
            XGpio_DiscreteWrite(&xLEDS,1,0);
}

// Initialize GPIO
static void gpioInit(){
            XGpio_Initialize(&xBTNSWI,XPAR_AXI_GPIO_0_DEVICE_ID);
   XGpio_Initialize(&xLEDS,XPAR_AXI_GPIO_1_DEVICE_ID);
   XGpio_SetDataDirection(&xBTNSWI,1,0xf);
            XGpio_SetDataDirection(&xBTNSWI,2,0xf);
            XGpio_SetDataDirection(&xLEDS,1,0x0);
}
```