FreeRTOS Software Timer

Robert Bara
Robert.Bara@temple.edu

Summary

Lab 9 returns to timers with the Vivado Zynq Processor System, however this lab utilizes a software timer instead of the AXI hardware timer. Software timers in FreeRTOS consist of two kinds of timers: one-shot timers and auto-reload timers. This lab will implement an auto-reload timer through the *prvTimerCallback()* API function.

Introduction

By utilizing a hardware design with buttons as an input and LEDs as an output, the software timer is implemented through the following program in which a single button with debouncing is fed as an input:

BTN0-The software timer is reset and the affect upon expiration timing for the LED pattern will be demonstrated.

BTN1-The software timer's period is decreased by 1 second (initially the period should be set to 10 seconds), until it reaches 5 seconds, which is the minimum period. When 5 seconds is reached, the LEDs should toggle between a pattern of 1100 and 0011.

BTN2-The software timer is stopped, and all LEDs are shut off.

BTN3-The software timer is restored to 10 seconds and resumes the timer with an LED pattern toggling between 1001 and 0110.

The software timer is an auto-reload and initially runs the LED pattern flipping between 1001 and 0110 upon expiration.

Discussion

Hardware Design

The hardware design consists of the Zynq processor block and one AXI GPIO that consists of 2 channels: buttons in channel 1 as an input and LEDs in channel 2 as an output. Upon the addition of these two blocks, automation connection should be ran and the block diagram should be verified, appearing as follows:

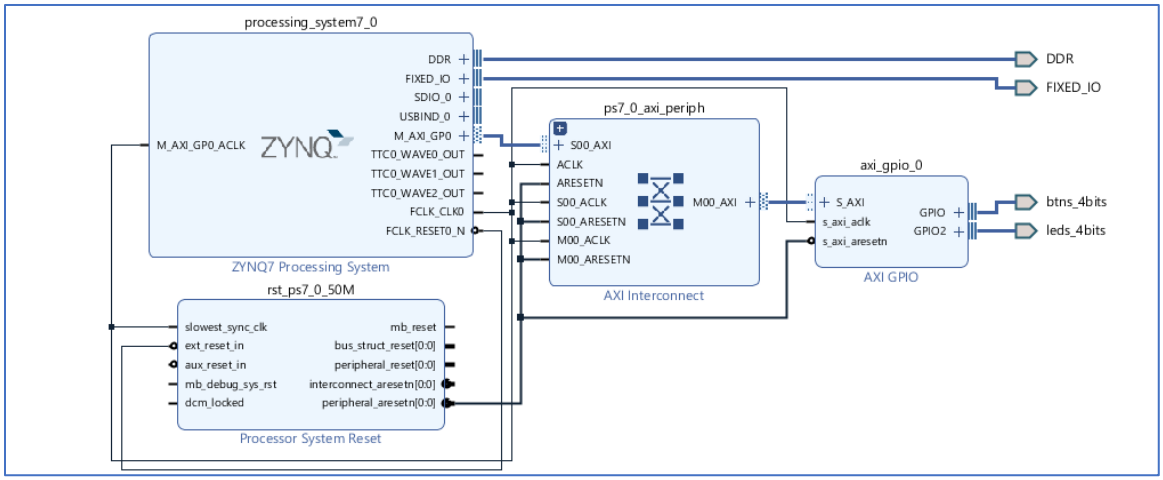

Figure 1. Block Diagram of Hardware Design

The HDL wrapper can then be created, and a bitstream should be generated. Upon completion, the design can be exported to hardware with the bitstream included, and SDK will be launched for the software design.

Software Design

Initialization

The program initializes by defining all of the proper headers, macros, GPIO, and input/output channels connected to the GPIO:

```
*/
//Robert Bara Lab9
/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"
#include "stdio.h"
/* Xilinx includes. */
#include "xil_printf.h"
#include "xgpio.h"

//definitions
#define TIMER_ID    1
#define DELAY_10_SECONDS    10000UL
#define DELAY_1_SECOND      1000UL
#define DELAY_5_SEC     5000UL
#define TIMER_CHECK_THRESHOLD   9
#define printf xil_printf

//GPIO definitions
#define INP_DEVICE_ID XPAR_PS7_GPIO_0_DEVICE_ID //GPIO device connected to BTNs and LEDs
XGpio InpInst;

//Defining each channel
#define BTN_CHANNEL 1
#define LED_CHANNEL 2
```

Figure 2. Headers and definitions

The tasks are initialized, and Main begins with the typical initialization of the GPIO Driver, buttons, and LEDs:

```
/*-----------------------------------------------------------*/
/* The tasks as described at the top of this file. */
static void BTNTask( void *pvParameters );
static void vTimer1Callback( TimerHandle_t pxTimer );
/*-----------------------------------------------------------*/
static TimerHandle_t xTimer = NULL;
static int tmr_period=( DELAY_10_SECONDS );
static int led=0b1001;

int main( void )
{
    //GPIO Driver Initialization
    int status;
    status = XGpio_Initialize(&InpInst, INP_DEVICE_ID);             // LED initialization
    if (status != XST_SUCCESS) {printf("LED Initialization failed"); return XST_FAILURE;}
    XGpio_SetDataDirection(&InpInst, LED_CHANNEL, 0x00);            //Setting LED to Output
    //Input Initialization
    status = XGpio_Initialize(&InpInst, INP_DEVICE_ID);
    if (status != XST_SUCCESS) {printf("Input Initialization failed"); return XST_FAILURE;}
    //Setting Btn as Input
    XGpio_SetDataDirection(&InpInst, BTN_CHANNEL, 0xFF);
    //initializing LED as 0b1001
    XGpio_DiscreteWrite(&InpInst,LED_CHANNEL,0b1001);
    printf("Initialization Complete\n");
```

Figure 3. Task prototypes and GPIO Driver Initialization

The rest of main defines the button input task and creates an auto-reload timer with respective expiration set to 10 seconds. The timer is configured and the scheduler starts running:

```
/* Create the input Button task*/
xTaskCreate(    BTNTask,                    /* The function that implements the task. */
                ( const char * ) "BTN",         /* Text name for the task, provided to assist debugging only. */
                configMINIMAL_STACK_SIZE,   /* The stack allocated to the task. */
                NULL,                       /* The task parameter is not used, so set to NULL. */
                tskIDLE_PRIORITY,           /* The task runs at the idle priority. */
                &BTNTask );
/* Create a timer with an initial timer expiry of 10 seconds.
 * The timer set to auto reload. */
xTimer = xTimerCreate( (const char *) "Timer",
                       pdMS_TO_TICKS(tmr_period),
                       pdTRUE,
                       (void *) TIMER_ID,
                       vTimer1Callback);

/* Check the timer was created. */
configASSERT( xTimer );

/* start the timer with a block time of 0 ticks. This means as soon
   as the schedule starts the timer will start running and will expire after
   10 seconds */
xTimerStart( xTimer, 0 );

/* Start the tasks and timer running. */
vTaskStartScheduler();

/* If all is well, the scheduler will now be running, and the following line
will never be reached.  If the following line does execute, then there was
insufficient FreeRTOS heap memory available for the idle and/or timer tasks
to be created.  See the memory management section on the FreeRTOS web site
for more details. */
for( ;; );
```

Figure 4. Task Creation

Subroutines

In the button task a button debounce is set for 0.4 sec debouncing, and after a valid button press is held, it will enter the conditionals. The first condition is when button0 is pressed, the timer will be reset. This button does not reset the timer's period:

```
static void BTNTask( void *pvParameters )
{
int x10second = ( DELAY_10_SECONDS );
int x1second = ( DELAY_1_SECOND );
int x5second = (DELAY_5_SEC);
int btn=0, debounce=0;
const TickType_t x0_4second=pdMS_TO_TICKS(400uL ); //initialize 0.4 second delay for debouncing
    while(1)
    {

        //Reading Inputs from Board
        btn=XGpio_DiscreteRead(&InpInst,BTN_CHANNEL);

        //store value debouncing
        debounce=btn&0b1111;
        /* Delay for 0.4second. */
        vTaskDelay(x0_4second);
        //read button again
        btn=XGpio_DiscreteRead(&InpInst,BTN_CHANNEL);
        //printf("btn %d led %d tmr_per %d\n",btn,led,tmr_period);
        if(btn=(debounce&0b1111))
        {
            if(btn==0b0001)
            {
                //resetting timer
                printf("BTN 0:reset timer, timer period %d\n", tmr_period);
                xTimerReset(xTimer,0);
            }
```

Figure 5. Button Debounce and Button0

The rest of the button conditions is as follows. Notice that throughout changing the timer period I have set all of xsecond variables as ints instead of type tick and then I use the pdMS_TO_TICKS on the timer period variable, this is to prevent any issues when subtracting since the type tick variable type was giving me some issues. Furthermore, when I change the period, I set 0 as a block time equal to pdPASS, this is based on the documentation on the FreeRTOS change period function from FreeRTOS's website, though this seems like this could be optional, I would like to keep it in to follow the documentation. I could have modelled my timer period decrementation off of their template as well but I decided to keep using the x1second idea, since it is easier to understand. I will attach the link for this documentation in the appendix.

Button 1 decrements the timer period by 1 second and upon reaching a minimum timer period of 5 seconds, a new LED pattern is toggled. Button 2 stops the software timer and shuts off the LEDs utilizing xTimerStop. Button 4 undo's what button 3 does by using xTimerStart to start the timer, but this function also resets the LEDs back to their original pattern as well as changing the period back to 10 seconds. An else is placed if the button input is invalid.

```
        if(btn==0b0010)
        {
            //change timer period to decrease by 1 seconds until 5sec
            //is reached.
            tmr_period=tmr_period-x1second;
            xTimerChangePeriod(xTimer,pdMS_TO_TICKS(tmr_period),0==pdPASS);
            printf("BTN 1: Decrease Timer Period, timer period %d\n", tmr_period);
            //when 5 seconds is reached
            if(tmr_period<=x5second)
            {   tmr_period=x5second;    //Timer period can not decrease any further
                xTimerChangePeriod(xTimer,pdMS_TO_TICKS(tmr_period),0==pdPASS);
                printf("BTN 1: Timer Period is %d or min of 5 sec\n", tmr_period);
                led=0b1100;     //toggle new LED flip pattern
            }
        }
        if(btn==0b0100)
        {   printf("BTN 2: stop timer and turn off leds\n");
            //Stop software timer and shut off LEDS
            xTimerStop(xTimer,0);
            XGpio_DiscreteWrite(&InpInst,LED_CHANNEL,0);
        }
        if(btn==0b1000)
        {
            //Start software timer and LEDS Pattern
            tmr_period=x10second;   //resets the period to 10seconds
            xTimerChangePeriod(xTimer,pdMS_TO_TICKS(tmr_period),0==pdPASS);
            xTimerStart(xTimer,0);  //starts the timer
            led=0b1001;             //initializes LEDs
            XGpio_DiscreteWrite(&InpInst,LED_CHANNEL,led);
            printf("BTN 3: start timer and leds, Timer Period %d\n", tmr_period);
        }
        else    btn=XGpio_DiscreteRead(&InpInst,BTN_CHANNEL); //Check for a valid button input
    }
```

Figure 6. The rest of the button conditionals

The timer task simply flips the LEDs upon an expiration as long as the LEDs are on.

```
/*-----------------------------------------------------------*/
static void vTimer1Callback( TimerHandle_t pxTimer )
{   //Just a safety/priority logic here incase the program fails
    if (led==0)led=0;
    else
    {   //Flip the LEDs upon timer expiration
        led=~led;
        printf("timer expires, flip LEDs %d Timer Period %d \n", led, tmr_period);
    }
    XGpio_DiscreteWrite(&InpInst,LED_CHANNEL,led);
}
```

Verification

Video Verification Link

https://youtu.be/J0If1QRfz9k

I should mention I did forget to show that the timer stop function truly works in the video by cross comparing it with other buttons (only button 3 should work/start the timer), so rather than redo the video here are 4 screenshots showing this verification:

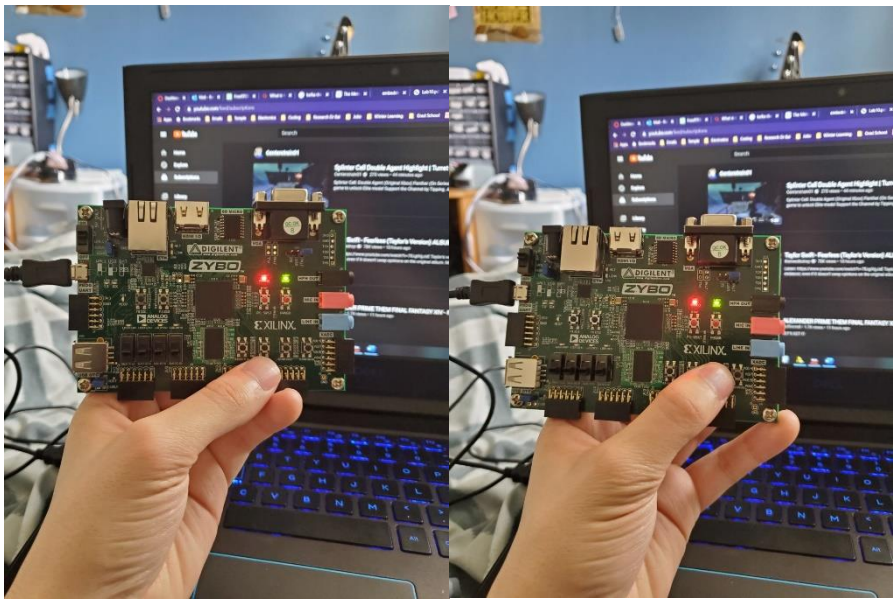

Figure 7 A and B. Shutting off timer, testing button 2 does not work

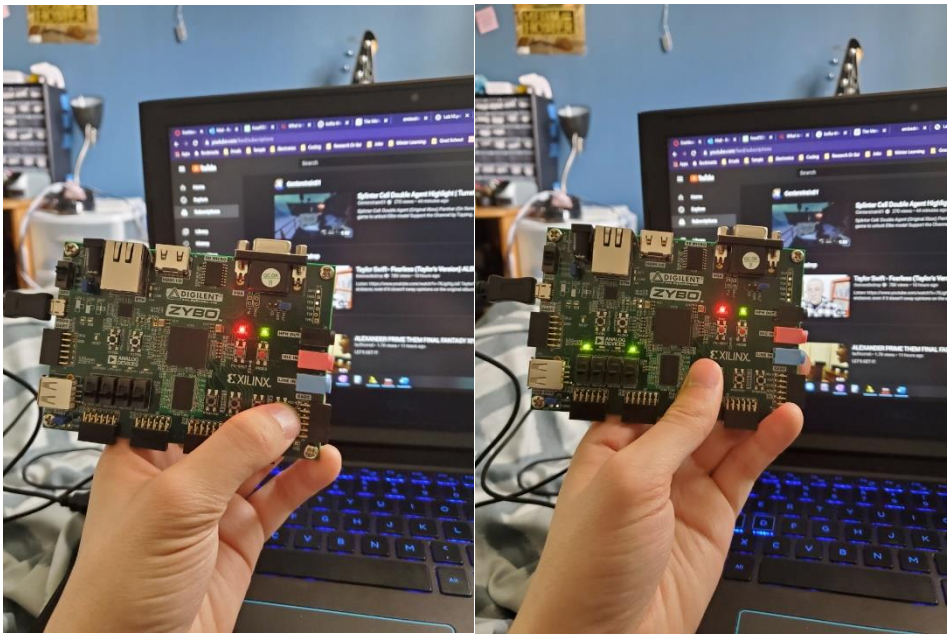

Figure 8 A and B. Testing button 0 does not work, testing only button 3 works by starting the timer and LEDs

Conclusion

This lab took a step back from the DSP topics we have been exploring throughout the semester, to return to FreeRTOS with a lab similar to lab 4's timer interrupts. The biggest difference between this lab and lab 4 is the fact that this lab utilizes the FreeRTOS software timer compared to the AXI Hardware timer. Upon completion of this lab, the lab solidified the courses' discussion of timers and taught us how to use auto-reload timers for future FreeRTOS projects.

Appendix

FreeRTOS online reference documentation which helped with changing the timer period:

https://www.freertos.org/FreeRTOS-timers-xTimerChangePeriod.html

FreeRTOS Code

```
/*
    Copyright (C) 2017 Amazon.com, Inc. or its affiliates.  All Rights Reserved.
    Copyright (C) 2012 - 2018 Xilinx, Inc. All Rights Reserved.

    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
    the Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software. If you wish to use our Amazon
    FreeRTOS name, please do so in a fair use way that does not cause confusion.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

```
    http://www.FreeRTOS.org
    http://aws.amazon.com/freertos


    1 tab == 4 spaces!
*/
//Robert Bara Lab9
/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"
#include "stdio.h"
/* Xilinx includes. */
#include "xil_printf.h"
#include "xgpio.h"

//definitions
#define TIMER_ID    1
#define DELAY_10_SECONDS      10000UL
#define DELAY_1_SECOND          1000UL
#define DELAY_5_SEC           5000UL
#define TIMER_CHECK_THRESHOLD    9
#define printf xil_printf

//GPIO definitions
#define INP_DEVICE_ID XPAR_PS7_GPIO_0_DEVICE_ID //GPIO device connected to BTNs and LEDs
XGpio InpInst;
```

```
//Defining each channel
#define BTN_CHANNEL 1
#define LED_CHANNEL 2

/*-----------------------------------------------------------*/
/* The tasks as described at the top of this file. */
static void BTNTask( void *pvParameters );
static void vTimer1Callback( TimerHandle_t pxTimer );
/*-----------------------------------------------------------*/
static TimerHandle_t xTimer = NULL;
static int tmr_period=( DELAY_10_SECONDS );
static int led=0b1001;

int main( void )
{
	//GPIO Driver Initialization
	int status;
	status = XGpio_Initialize(&InpInst, INP_DEVICE_ID);			// LED initialization
	if (status != XST_SUCCESS) {printf("LED Initialization failed"); return XST_FAILURE;}
	XGpio_SetDataDirection(&InpInst, LED_CHANNEL, 0x00);
	//Setting LED to Output
	//Input Initialization
	status = XGpio_Initialize(&InpInst, INP_DEVICE_ID);
	if (status != XST_SUCCESS) {printf("Input Initialization failed"); return XST_FAILURE;}
	//Setting Btn as Input
	XGpio_SetDataDirection(&InpInst, BTN_CHANNEL, 0xFF);
```

```
	//initializing LED as 0b1001
	XGpio_DiscreteWrite(&InpInst,LED_CHANNEL,0b1001);
	printf("Initialization Complete\n");


	/* Create the input Button task*/
	xTaskCreate(   BTNTask,                                  /* The function that implements the task. */
                                ( const char * ) "BTN",                  /* Text name for the task, provided to assist debugging only. */
                                configMINIMAL_STACK_SIZE,     /* The stack allocated to the task. */
                                NULL,
	/* The task parameter is not used, so set to NULL. */
                                tskIDLE_PRIORITY,                /* The task runs at the idle priority. */
                                &BTNTask );
	/* Create a timer with an initial timer expiry of 10 seconds.
	 * The timer set to auto reload. */
	xTimer = xTimerCreate( (const char *) "Timer",
                                                pdMS_TO_TICKS(tmr_period),
                                                pdTRUE,
                                                (void *) TIMER_ID,
                                                vTimer1Callback);


	/* Check the timer was created. */
	configASSERT( xTimer );

	/* start the timer with a block time of 0 ticks. This means as soon
	   as the schedule starts the timer will start running and will expire after
	   10 seconds */
```

```
	xTimerStart( xTimer, 0 );

	/* Start the tasks and timer running. */
	vTaskStartScheduler();

	/* If all is well, the scheduler will now be running, and the following line
	will never be reached.  If the following line does execute, then there was
	insufficient FreeRTOS heap memory available for the idle and/or timer tasks
	to be created.  See the memory management section on the FreeRTOS web site
	for more details. */
	for( ;; );
}


/*-----------------------------------------------------------*/
static void BTNTask( void *pvParameters )
{
int x10second = ( DELAY_10_SECONDS );
int x1second = ( DELAY_1_SECOND );
int x5second = (DELAY_5_SEC);
int btn=0, debounce=0;
const TickType_t x0_4second=pdMS_TO_TICKS(400uL ); //initialize 0.4 second delay for debouncing
	while(1)
	{

		//Reading Inputs from Board
		btn=XGpio_DiscreteRead(&InpInst,BTN_CHANNEL);
```

```
//store value debouncing
debounce=btn&0b1111;
/* Delay for 0.4second. */
vTaskDelay(x0_4second);
//read button again
btn=XGpio_DiscreteRead(&InpInst,BTN_CHANNEL);
//printf("btn %d led %d tmr_per %d\n",btn,led,tmr_period);
if(btn=(debounce&0b1111))
{
	if(btn==0b0001)
	{
		//resetting timer
		printf("BTN 0:reset timer, timer period %d\n", tmr_period);
		xTimerReset(xTimer,0);
	}
	if(btn==0b0010)
	{
		//change timer period to decrease by 1 seconds until 5sec
		//is reached.
		tmr_period=tmr_period-x1second;

	xTimerChangePeriod(xTimer,pdMS_TO_TICKS(tmr_period),0==pdPASS);
		printf("BTN 1: Decrease Timer Period, timer period %d\n",
tmr_period);
		//when 5 seconds is reached
		if(tmr_period<=x5second)
		{	tmr_period=x5second;	//Timer period can not
decrease any further

	xTimerChangePeriod(xTimer,pdMS_TO_TICKS(tmr_period),0==pdPASS);
```

```
                                printf("BTN 1: Timer Period is %d or min of 5
sec\n", tmr_period);
                                led=0b1100;             //toggle new LED flip pattern
                        }
                }
                if(btn==0b0100)
                {       printf("BTN 2: stop timer and turn off leds\n");
                        //Stop software timer and shut off LEDS
                        xTimerStop(xTimer,0);
                        XGpio_DiscreteWrite(&InpInst,LED_CHANNEL,0);
                }
                if(btn==0b1000)
                {
                        //Start software timer and LEDS Pattern
                        tmr_period=x10second;       //resets the period to
10seconds

        xTimerChangePeriod(xTimer,pdMS_TO_TICKS(tmr_period),0==pdPASS);
                        xTimerStart(xTimer,0);      //starts the timer
                        led=0b1001;                         //initializes LEDs
                        XGpio_DiscreteWrite(&InpInst,LED_CHANNEL,led);
                        printf("BTN 3: start timer and leds, Timer Period %d\n",
tmr_period);
                }
                else    btn=XGpio_DiscreteRead(&InpInst,BTN_CHANNEL);
//Check for a valid button input
                }

}
}
```

```
/*-----------------------------------------------------------*/
static void vTimer1Callback( TimerHandle_t pxTimer )
{	//Just a safety/priority logic here incase the program fails
	if (led==0)led=0;
	else
	{	//Flip the LEDs upon timer expiration
		led=~led;
		printf("timer expires, flip LEDs %d Timer Period %d \n", led,
tmr_period);
	}
	XGpio_DiscreteWrite(&InpInst,LED_CHANNEL,led);
}
```