FreeRTOS Software Timer

Leomar Durán
https://github.com/lduran2

Summary

This lab more formally introduces the software timer in FreeRTOS. It builds on both Tutorial 2 and Lab 2, and has a standard hardware design of the Zybo board with 2 GPIOs each with up to 2 channels. So it is fairly straightforward.

The software timer is implemented as a task, so it is very similar to develop as the tasks in Lab 2 FreeRTOS Task Management, so it is very familiar. However, this means it has similar pitfalls, such as confusing the task handle with the callback function as a parameter.

The software timer is used in lieu of the hardware timer and interrupts from Tutorial 2. The timer is controlled using similar life cycle mutating function as the tasks.

This experiment demonstrates yet another way of scheduling with the Zybo board. This method is important because although it is limited to FreeRTOS, it offers some flexibility over the hardware timer.

Introduction

This project exercises the Software timer included in FreeRTOS. The software timer was introduced in the FreeRTOS Hello World template. However, it was not put to use since then. This project replaces the Hardware timer in the standalone system on the Zybo board with the Software timer to perform similar tasks.

After this project, the student will be able to

- Start a software timer.
- Stop a software timer.
- Reset a software timer.
- Change the expiry time on a software timer.

The direct objective of this project is to build a design in FreeRTOS, which uses the switches and buttons to control a software timer.

The analysis will simply cycle through the different functions to show that they all work as specified.

The instructions are very clear, except for one point which is a bit more open to interpretation. That point is "`SW0` has priority of operation over `SW1` and this should be demonstrated." This is assumed to mean that `SW0` must be switched before `SW1`, and that if `SW1` is switched first, it will have no effect.

Discussion

Rather than use the hardware timer on the Zybo board and other interrupts, this project uses the FreeRTOS software timer.

The software timer is part of the FreeRTOS API and is used through the functions `xTimerCreate`, `xTimerStart`, `xTimerStop`, `xTimerReset`, and `xTimerChangePeriod` to manipulate its life cycle.

The software design builds on `rtos_task_management.c` from Lab 2, which in turn builds on `FreeRTOS_Hello_World.c`.

The analysis of the design is to simply cycle through the different functions with the following in mind

- The initial timer period is $5\,\mathrm{s}$.
- Changes made by the switches only occur once, rather than continuously, so that buttons will have effect.
- For stopping the timer, the sequence is `SW0` on, then `SW1` off.
- For starting the timer, the sequence is `SW0` off, then `SW1` on.
- `BTN0` resets the timer. If the timer is off, after the expiration, it will start again.
- `BTN1` sets the timer timer period to $10\,\mathrm{s}$.
- `BTN2` turns off the LEDs and stops the timer.
- `BTN3` reinitializes the timer and the LED to their initial state, including the timer period to $5\,\mathrm{s}$.

Hardware design

The hardware design is the Zybo board with two GPIOs. The first GPIO has the LEDs at channel 1 and the buttons at channel 2. The second GPIO has the switches on channel 1. Figure 1 illustrates this.

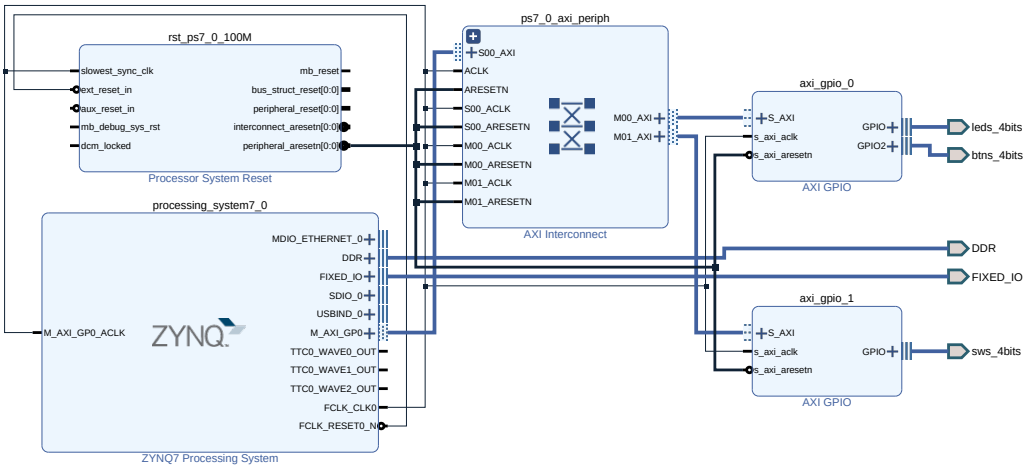

Figure 1: The hardware design.

Main function

The main function sets up the two tasks `BTNtask` and `SWtask`, the timer task `TIMERtask`. It also sets up the GPIOs and their channel's directions.

TIMERtask

Like non-timer tasks, the software timer task is implemented as a callback function with a handle, and it is created with a similar function.

This task checks if it is being called by the correct timer, then it writes the LED data to the LED channel, and inverts the LED data in preparation for next time.

The timer is then reset.

SWtask

The switch task keeps track of the current and previous switch value, because a lot of the functionality depends on which switch was currently flipped. It also keeps track of a state, which may be either `STANDBY`, `STOPPABLE` or `STARTABLE`.

Once the current and previous values are different, the task checks if the new value is `SW0` on from off, in which case, the state becomes `STOPPABLE`. In the reverse case, the state becomes `STARTABLE`. If any other switch is flipped, the state becomes `STANDBY`.

However, before this, if the state is `STOPPABLE`, pulling `SW1` stops the timer. Or if the state is `STARTABLE`, throwing `SW1` starts the timer.

BTNtask

The button is first debounced, and after that, it is very straightforward. As mentioned earlier

- `BTN0` resets the timer. If the timer is off, after the expiration, it will start again.
- `BTN1` sets the timer timer period to 10 s.

- `BTN2` turns off the LEDs and stops the timer.
- `BTN3` reinitializes the timer and the LED to their initial state, including the timer period to $5\,\mathrm{s}$.

The debounce is performed by skipping the current iteration of the infinite loop if the button has not changed. Then skipping if it is still changing. If neither skip happens, the task continues to inspecting the button.

Changing the LED with `BTN2` and `BTN3` consists of an extra step.

Reinitializes the timer with `BTN3` is more involved, requiring changing the period before starting again.

Manipulating the software timer here was a little troublesome only because I confused the handle `xTIMERtask` with the callback function `vTIMERtask`, and Vivado did not have an error since both are compatible pointers. I fixed this by renaming the callback function to `vTIMERtaskCallback`.

The results

The results are as expected, and can be viewed at https://youtu.be/x5ZMw6b6jy4.

Conclusions

The project met the objectives presented in the Introduction, as I was able to control the life cycle of the Software timer, from start to stop and reset, to being able to control the period of the timer.

Appendices

The software design is below in verbatim. It can also be found at `https://github.com/lduran2/ece3623-lab8-rtos_software_timer/blob/master/lab08_freertos_software_timer.sdk/rtos_software_timer/src/rtos_software_timer.c`

```
/*
    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!
*/
/*
 * rtos_software_timer.c
 *
 * Created on:      10 November 2020 (based on rtos_task_management.c)
 *     Author:      Leomar Duran
 *    Version:      2.3
 */

/*
 * rtos_task_management.c
 *
 * Created on:      16 September 2020 (based on FreeRTOS_Hello_World.c)
 *     Author:      Leomar Duran
 *    Version:      1.5
 */

/*************************************************************************************
********
* VERSION HISTORY
*************************************************************************************
*******
*      v2.3 - 10 November 2020
*             Implemented BTNtask.
*             Renamed vTIMERtask -> vTIMERtaskCallback to avoid confusion.
*
*      v2.2 - 10 November 2020
*             Implemented SWtask.
```

```
*
*	v2.1 - 10 November 2020
*		Re-implemented LED blinker with the software timer, 500 ms -> 5 s.
*
*	v2.0 - 10 November 2020
*		Set up GPIOs and implemented LED blinker with `vTaskDelay`.
*
*	v1.5 - 16 September 2020
*		Added TaskBTN feature that controls TaskSW.
*
*	v1.4 - 16 September 2020
*		Added TaskSW  feature that controls TaskLED and TaskBTN.
*
*	v1.3 - 16 September 2020
*		Stubbed TaskSW.  Optimized TaskBTN.
*
*	v1.2 - 16 September 2020
*		Added TaskBTN feature that controls TaskLED.
*
*	v1.1 - 15 September 2020
*		Set up LED counter.
*
*	v1.0 - 2017
*		Started with FreeRTOS_Hello_World.c
*
*******************************************************************************
******/

/*******************************************************************************
********
* TASK DESCRIPTION
*******************************************************************************
*******
* TIMERtask := a blinker between 0b1100 and 0b0011, displayed in the LEDs.
*
* BTNtask := reads the buttons to control the other tasks
*
* SWtask  := reads the switches to control the other tasks
*
*******************************************************************************
******/

/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
/* Xilinx includes. */
#include "xil_printf.h"
#define	printf xil_printf
#include "xparameters.h"
#include "xgpio.h"
#include "xstatus.h"

/* task definitions */
```

```
#define	DO_TIMER_TASK1                                                  /*
whether to do TIMERtask */
#define	DO_BTN_TASK		1
	/* whether to do BTNtask */
#define	DO_SW_TASK		1
	/* whether to do SWtask */

/* GPIO definitions */
#define	LD_BTN_DEVICE_ID	XPAR_AXI_GPIO_0_DEVICE_ID /* GPIO device for LEDs,
Buttons */
#define	SW_DEVICE_ID		XPAR_AXI_GPIO_1_DEVICE_ID /* GPIO device for
switches */
#define BTN_DELAY   250UL                                       /* button delay
length for debounce */
#define LED_DEV_CH  &LdBtnInst, 1                       /* GPIO device and port
for LEDs */
#define BTN_DEV_CH  &LdBtnInst, 2                       /* GPIO device and port
for buttons */
#define  SW_DEV_CH    &SwInst, 1                        /* GPIO device and port
for switches */

#define	TIMER_TASK_ID			1
#define	TIMER_TASK_CHECK_THRESHOLD9
#define	TIMER_DELAY_INIT	5000UL                          /* initial LED
delay length (in ms) */
#define	TIMER_DELAY_BTN1	10000UL                             /* LED delay
length on BTN1 (in ms) */

/* GPIO instances */
XGpio LdBtnInst;                        /* GPIO Device driver instance for
LEDs, Buttons */
XGpio SwInst;                           /* GPIO Device driver instance for
switches */

/* leds masks */
#define	LED_INIT	0b1100                                  /* initial value of
LED */

/* switch masks */
#define	SW0		0b0001
#define	SW1		0b0010
#define	SWOFF		0b0000

/* button masks */
#define	BTN0		0b0001
#define	BTN1		0b0010
#define	BTN2		0b0100
#define	BTN3		0b1000
/*-----------------------------------------------------------*/

/* The tasks as described at the top of this file. */
static void prvBTNtask( void *pvParameters );
static void prvSWtask ( void *pvParameters );
static void vTIMERtaskCallback( TimerHandle_t pxTimer );
/*-----------------------------------------------------------*/
```

```
/* The task handles to control other tasks. */
static TaskHandle_t xBTNtask;
static TaskHandle_t xSWtask;
static TimerHandle_t xTIMERtask = NULL;
long RxtaskCntr = 0;
/* The LED blinker. */
int ledBlnkr = LED_INIT;

int main( void )
{
	int Status;
	const TickType_t xTIMERticks = pdMS_TO_TICKS( TIMER_DELAY_INIT );

	if (DO_BTN_TASK) {
		printf( "Starting BTNtask. . .\r\n" );
		/* Create BTNtask with priority 1. */
		xTaskCreate(
				prvBTNtask,					/* The
function implementing the task. */
			( const char * ) "BTNtask",			/* Text name
provided for debugging. */
				configMINIMAL_STACK_SIZE,	/* Not much need
for a stack. */
				NULL,					/* The
task parameter, not in use. */
				( UBaseType_t ) 1,			/* The next
to lowest priority. */
				&xBTNtask );
		printf( "\tSuccessful\r\n" );
	}

	if (DO_SW_TASK) {
		printf( "Starting SWtask . . .\r\n" );
		/* Create SWtask with priority 1. */
		xTaskCreate(
				prvSWtask,					/* The
function implementing the task. */
			( const char * ) "SWtask",		/* Text name
provided for debugging. */
				configMINIMAL_STACK_SIZE,	/* Not much need
for a stack. */
				NULL,					/* The
task parameter, not in use. */
				( UBaseType_t ) 1,			/* The next
to lowest priority. */
				&xSWtask );
		printf( "\tSuccessful\r\n" );
	}

	if (DO_TIMER_TASK) {
		printf( "Starting TIMERtask. . .\r\n" );
		/* Create a timer with a timer expiry of 10 seconds. The timer would
expire
```

```
               after 10 seconds and the timer call back would get called. In the timer
call back
               checks are done to ensure that the tasks have been running properly
till then.
               The tasks are deleted in the timer call back and a message is printed
to convey that
               the example has run successfully.
               The timer expiry is set to 10 seconds and the timer set to not auto
reload. */
              xTIMERtask = xTimerCreate( (const char *) "TIMERtask",
                                                    xTIMERticks,
                                                    pdTRUE,
       /* this is a multiple shot timer */
                                                    (void *) TIMER_TASK_ID,
                                                    vTIMERtaskCallback);
              /* Check the timer was created. */
              configASSERT( xTIMERtask );

              /* 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( xTIMERtask, 0 );
              printf( "\tSuccessful\r\n" );
       }

       /* initialize the GPIO driver for the LEDs */
       Status = XGpio_Initialize(&LdBtnInst, LD_BTN_DEVICE_ID);
       if (Status != XST_SUCCESS) {
              return XST_FAILURE;
       }
       /* set LEDs direction to output */
       XGpio_SetDataDirection(LED_DEV_CH, 0x00);
       /* set buttons direction to input */
       XGpio_SetDataDirection(BTN_DEV_CH, 0xFF);

       /* initialize the GPIO driver for the buttons */
       Status = XGpio_Initialize(&SwInst, SW_DEVICE_ID);
       if (Status != XST_SUCCESS) {
              return XST_FAILURE;
       }
       /* set switches to input direction to input */
       XGpio_SetDataDirection(SW_DEV_CH, 0xFF);

       /* 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( ;; );
}
```

```
/*-----------------------------------------------------------*/
/* prints the integer i as a 4-bit boolean */
void printb(char* format, int i) {
	printf(format,
				((i >> 3) & 1),
				((i >> 2) & 1),
				((i >> 1) & 1),
				((i >> 0) & 1)
			);
}

/*-----------------------------------------------------------*/
static void vTIMERtaskCallback( TimerHandle_t pxTimer )
{
	static long lTimerId;
	configASSERT( pxTimer );

	// get the ID of the timer
	lTimerId = ( long ) pvTimerGetTimerID( pxTimer );

	if (lTimerId != TIMER_TASK_ID) {
		xil_printf("TIMERtask FAILED: Unexpected timer.");
		return;
	}

	/* display the blinker */
	XGpio_DiscreteWrite(LED_DEV_CH, ledBlnkr);
	printb("TIMERtask: blink := 0b%d%d%d%d.\r\n", ledBlnkr);

	/* update the blinker */
	ledBlnkr = ~ledBlnkr;

	/* reset the timer */
	xTimerReset( xTIMERtask, 0 );
}


/*-----------------------------------------------------------*/
static void prvBTNtask( void *pvParameters )
{
	const TickType_t BTNseconds = pdMS_TO_TICKS( BTN_DELAY );
	const TickType_t xTIMERticksInit = pdMS_TO_TICKS( TIMER_DELAY_INIT );
	const TickType_t xTIMERticksBtn1 = pdMS_TO_TICKS( TIMER_DELAY_BTN1 );

	int btn[2];  /* Hold the button values, 0 : current, 1 : previous. */

	for( ;; )
	{
		/* Read input from the buttons. */
		btn[0] = XGpio_DiscreteRead(BTN_DEV_CH);

		/* Debounce: */
		/* skip this iteration if the button value has not changed */
		if ( btn[0] == btn[1] ) {
```

```
			continue;
		}
		btn[1] = btn[0];	/* push current switch value to previous value */
		vTaskDelay( BTNseconds ); /* delay until the end of the bounce */

		btn[0] = XGpio_DiscreteRead( BTN_DEV_CH );	/* read again */

		/* skip if the button value is still changing */
		if ( btn[0] != btn[1] ) {
			continue;
		}
		printb("BTNtask: Button changed to 0b%d%d%d%d.\r\n", btn[0]);

		/* BTN0 resets the TIMERtask */
		if ((btn[0] & BTN0) == BTN0) {
			printf("BTNtask : TIMERtask is reset.\r\n");
			xTimerReset( xTIMERtask, 0 );
		}
		/* BTN1 sets the TIMERtask to 10 seconds */
		if ((btn[0] & BTN1) == BTN1) {
			printf("BTNtask : TIMERtask <- 10 seconds\r\n");
			xTimerChangePeriod( xTIMERtask,
					xTIMERticksBtn1,
					0
				);
		}
		/* BTN2 stops the TIMERtask, and resets the LEDs */
		if ((btn[0] & BTN2) == BTN2) {
			printf("BTNtask : TIMERtask is stopped, LEDs is off.\r\n");
			xTimerStop( xTIMERtask, 0 );
			XGpio_DiscreteWrite( LED_DEV_CH, 0b0000 );
		}
		/* BTN3 starts the TIMERtask, and sets the reinitializes the LEDs */
		if ((btn[0] & BTN3) == BTN3) {
			printf("BTNtask : TIMERtask and LEDs are reinitialized.\r\n");
			xTimerChangePeriod( xTIMERtask,
					xTIMERticksInit,
					0
				);
			XGpio_DiscreteWrite( LED_DEV_CH, LED_INIT );
			xTimerStart( xTIMERtask, 0 );
		}

	} /* end for( ;; ) */
}


/*-----------------------------------------------------------*/
static void prvSWtask( void *pvParameters )
{
	char sw[2];  /* Hold the switch values, 0 : current, 1 : previous. */
	enum { STANDBY, STOPPABLE, STARTABLE } state = STANDBY;

	for( ;; )
	{
```

```
		/* Read input from the switches. */
		sw[1] = sw[0];		/* push current switch value to previous value */
		sw[0] = XGpio_DiscreteRead(SW_DEV_CH);

		/* skip this iteration if the switch value has not changed */
		if (sw[0] == sw[1]) {
			continue;
		}

		/* prioritize SW0 over SW1 */

		/* depending on state, inspect SW1 */
		/* If stoppable, SW1 is OFF, then stop  the timer */
		if ((state == STOPPABLE) && ((sw[0] & SW1) == SWOFF)) {
			printf("SWtask : TIMERtask is stopped.\r\n");
			xTimerStop (xTIMERtask, 0);
		}
		/* If startable, SW1 is ON , then start the timer */
		else if ((state == STARTABLE) && ((sw[0] & SW1) == SW1  )) {
			printf("SWtask : TIMERtask is started.\r\n");
			xTimerStart(xTIMERtask, 0);
		}

		/* set the state according to new switch value */
		/* stoppable state if SW0 switched ON  from previous value */
		if (((sw[0] & SW0) == SW0) && ((sw[1] & SW0) == SWOFF)) {
			state = STOPPABLE;
		}
		/* startable state if SW0 is OFF from previous value */
		else if (((sw[0] & SW0) == SWOFF) && ((sw[1] & SW0) == SW0)) {
			state = STARTABLE;
		}
		/* all other cases, the state is STANDBY */
		else {
			state = STANDBY;
		}
	}
}
```