Task Management in FreeRTOS on Zybo

Robert Bara
Robert.Bara@temple.edu

Summary

This laboratory serves as an introduction to Task Management in FreeRTOS by using Vivado and SDK to modify the *FreeRTOS_Hello_World.c* and perform various tasks discussed in the manual, on the Zybo Board. A hardware design using Vivado is modified to have two GPIOs similar to Lab 1, but GPIO0 contains input channels for buttons and switches, while GPIO1 contains an output channel for LEDs.

Introduction

To create the hardware design, this lab stems from following the first tutorial in the *Zynq Book Tutorials* as a foundation, just as done in Lab 1. Two GPIOs are added to the hardware design to yield buttons and switches on the Zybo Board as inputs, and LEDs will act as an output based on the following tasks, all of which, are the same priority, so they receive an equal amount of CPU, or executes in parallel:

TaskLED is set at priority 1, which clears a 4-bit counter and displays results across the LEDs on the Zyboboard. Switch 0 (SW0) increments the LEDs from 0 (0b0000) to 15 (0b1111) with an appropriate delay, when SW0 is low. When SW0 is high, LEDs are decremented. This task ignores any possible input from SW1, SW2, and/or SW3.

TaskBTN is set at priority 1 as well. This task reads an input button (btn) and determines the following cases when pressed and released, as long as no other buttons are being held.
Btn0: Suspends *TaskLED*

Btn1: Resumes *TaskLED*

Btn2: Suspends *TaskSW*

Btn3: Resumes *TaskSW*

Additionally, a button debouncing system should be implemented to ensure only 1 button is being read at a time, and to ignore multiple buttons being held at once.

TaskSW is set at priority 1 and utilizes SW1, SW2, and SW3 for the following cases:

SW1 High, SW2 & SW3 Low, ignores SW0: *TaskBTN* is suspended

SW2 High, SW1 & SW3 Low, ignores SW0: *TaskBTN* is resumed

SW3 High, SW1 & SW2 Low, ignores SW0: *TaskLED* is suspended

SW1 & SW2 High, SW3 Low, ignores SW0: *TaskLED* is resumed

Discussion

Hardware Design

The hardware design within this lab begins with adding the ZYNQ 7 Processing System which will wire up all additional components by bringing in the reset block (rst_ps7_0_100M) and the processor (ps7_axi_periph). Next, two GPIOs should be added with GPIO_0 wiring to the ZYBO's push buttons and switches as input channels. GPIO_1 is wired to the LEDs to display an output. Figure 1 demonstrates the completed block diagram below:

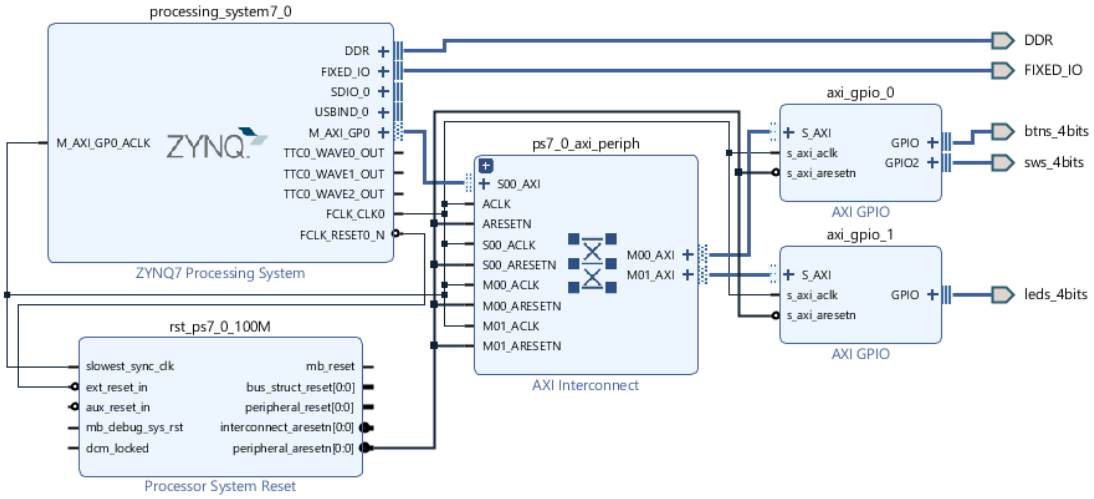

Figure 1. Vivado Block Diagram for Lab2, with SW and BTN inputs

Upon completing the design, the design must be validated, then a wrapper should be created in order to generate a bitstream. Finally, when the bitstream is written, the hardware is exported, and SDK can be launched to program the board.

C Program

The C program used within this lab, stems as a modified code of *FreeRTOS_Hello_World.c* and runs as follows.

Initialization

The top of the C program consists of defining the FreeRTOS and hardware header files to be included.

```
/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
/* Xilinx includes. */
#include "xil_printf.h"
#include "xparameters.h"
#include "xgpio.h"
```

Figure 2. All headers to be included

Definitions are then added to define printf as well as a delay of 1 second that will be used.

```
//definitions
#define DELAY_1_SECOND      1000UL
#define printf xil_printf
```

Figure 3. Definitions

Tasks are then defined for the LED, BTN, and SW tasks, as well as the task handles used by the queue for suspending and resuming each task:

```
/*-----------------------------------------------------------*/
/* The tasks as described at the top of this file. */
static void taskLED( void *pvParameters );
static void taskBTN( void *pvParameters );
static void taskSW( void *pvParameters );
/*-----------------------------------------------------------*/

/* The queue used by the taskLED, taskBTN, taskSW tasks, as described at the top of this
file. */
static TaskHandle_t xTaskLED;
static TaskHandle_t xTaskBTN;
static TaskHandle_t xTaskSW;
```

Figure 4. Task definitions and task handle declarations

The hardware requirements are then fulfilled by mapping each GPIO to their respective input and output channels. Additionally, static ints are declared to be used throughout the program, and may be seen in the appendix:

```
//GPIO definitions
#define LED_DEVICE_ID XPAR_AXI_GPIO_1_DEVICE_ID //GPIO device connected to LED
#define INP_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID //GPIO device connected to BTNs and SW
XGpio LEDInst, InpInst;
//defining each channel
#define LED_CHANNEL 1
#define BTN_CHANNEL 1
#define SW_CHANNEL 2
```

Figure 5. Hardware definitions of inputs and outputs

The main function initially starts by validating each GPIO driver initialization, and setting BTNs & SW to inputs, while LEDs are outputs:

```
//Main Function
int main( void ){
    //GPIO Driver Initialization
    int status;
    status = XGpio_Initialize(&LEDInst, LED_DEVICE_ID);             // LED initialization
    if (status != XST_SUCCESS) {printf("LED Initialization failed"); return XST_FAILURE;}
    XGpio_SetDataDirection(&LEDInst, 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 and SW as Inputs
    XGpio_SetDataDirection(&InpInst, BTN_CHANNEL, 0xFF);
    XGpio_SetDataDirection(&InpInst, SW_CHANNEL, 0xFF);
```

Figure 6. Main Function-GPIO initialization

Next the program follows FreeRTOS's structure to create the three tasks and set each task to a priority of 1. The remainder of the main program then calls *vTaskStartScheduler();* to start the tasks, and the for loop to control insufficient memory may be referred to in the appendix.

```
  /* Creating the three tasks.*/
  xTaskCreate(    taskLED,                    /* The function that implements the task. */
                  ( const char * ) "L",       /* 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 + 1,       /* The task runs at the idle priority. */
                  &xTaskLED );

  xTaskCreate( taskBTN,
               ( const char * ) "B",
               configMINIMAL_STACK_SIZE,
               NULL,
               tskIDLE_PRIORITY + 1,
               &xTaskBTN );

  xTaskCreate( taskSW,
               ( const char * ) "SW",
               configMINIMAL_STACK_SIZE,
               NULL,
               tskIDLE_PRIORITY + 1,
               &xTaskSW );
```

Figure 7. Task creation, all tasks are set to 1

Task Subroutines

TaskLED runs as follows, led is initialized at 0 and SW0 is read and checked to be even or odd. If it is odd, that means the sum of all switches are odd, meaning sw0 is high and the rest of the switches are ignored. When sw0 is high, the LEDs will increment until reaching 15 (0b1111), then a roll over occurs, and the LEDs count up starting at 0.

```
/*-----------------------------------------------------------*/
static void taskLED( void *pvParameters )
{
const TickType_t x1second = pdMS_TO_TICKS( DELAY_1_SECOND ); //initialize 1 second task delay
led=0; //initialize LEDs
inp=sw;//switch input
    while(1)
    {
        inp=XGpio_DiscreteRead(&InpInst, SW_CHANNEL);   //reading an input from the switches
        //If Sw 0 is high, the sum of sw will be odd, this case which decrements the LEDs, no matter SW 1-3
        if(sw%2==1){
            XGpio_DiscreteWrite(&LEDInst, LED_CHANNEL, led);
            led--;
            if(led<=0) led=15;
```

Figure 8. Checking if odd (SW0 is high), decrementing LEDs

If sw0 is even, then that means the sum of the sw input does not have sw0 engaged, so sw0 is low and ignores the remaining switches. Similarly, to the first conditional, the LEDs take the current led value and decrement until the roll over is reached.

```
//If Sw 0 is low, the sum of SW will be even, this case which increments the LEDs, ignoring SW 1-3
else if(sw%2==0){
    XGpio_DiscreteWrite(&LEDInst, LED_CHANNEL, led);
    led++;
    if(led==15) led=0;
}
```

Figure 9. Checking if Even (Sw0 is low), incrementing LEDs

The remaining two cases of *TaskLED* will be discussed in the verification section of the report. A delay is then called, and the while loop repeats unless the task is suspended.

TaskBTN runs as follows. To address a button-bounce issue, a button is taken as an input and a delay of two seconds is ran. If the button value is constant for at least two seconds, then the button value is recognized as stable and the input debounce takes the value and runs the corresponding conditional, for the cases of when btn and debounce are the same value.

```
/*-----------------------------------------------------------*/
static void taskBTN( void *pvParameters )
{const TickType_t x1second = pdMS_TO_TICKS( DELAY_1_SECOND );//delay of 1 second
    btn=0; //initialize buttons
    debounce=0; //initialize buttons

    while(1)
    {   //Read button value
        btn=XGpio_DiscreteRead(&InpInst,BTN_CHANNEL);
        vTaskDelay(x1second*2); /*Button debouncing for 2seconds,
                        if the input is stable for 2 seconds then debounce takes the input*/
        debounce=btn;
```

Figure 10. Button debouncing

The following conditionals will then be executed and either suspend or resume tasks as described in the procedure. Alternatively, a switch statement could have been used to describe each case. If no button is pressed or multiple buttons are pressed, no operation is held.

```
            if(debounce==0b0001 && btn==0b0001) //if btn0 is pressed the LEDs are suspended, ignoring other btn inputs
            {
                vTaskSuspend(xTaskLED);
                printf("\n TASK LED IS SUSPENDED\n");
            }
            else if (debounce==0b0010 && btn==0b0010) //if btn 1 is pressed the LEDs will resume, ignoring other btn inputs
            {
                vTaskResume(xTaskLED);
                printf("\n TASK LED IS RESUMED\n");
            }
            else if(debounce==0b0100 && btn==0b0100) //if btn 2 is pressed the SW task is suspended, ignoring other btn inputs
            {
                vTaskSuspend(xTaskSW);
                printf("\n TASK SW IS SUSPENDED\n");
            }

            else if(debounce==0b1000 && btn==0b1000) //if btn 3 is pressed the SW task will resume,ignoring other btn inputs
            {
                vTaskResume(xTaskSW);
                printf("\n TASK SW IS RESUMED\n");

            }
            else{ //if multiple buttons are pressed, nothing happens
                asm("NOP");
            }

        /*print the received btn data, because of btn debouncing, btn only outputs ever 2 seconds*/
        printf("\t\tbtn=%d", btn);

    }
```

Figure 11. Conditions for respective button cases

TaskSW operates almost exactly the same way as *TaskBTN* operates. This task however ignores sw0 as stated in the lab manual, which is why each conditional uses the or operands "| |". A "NOP" is placed for no operation amongst any combination of switching that was not specified in the lab manual, and a task delay is generated amongst every iteration of the while loop.

```
/*-----------------------------------------------------------*/
static void taskSW( void *pvParameters )
{const TickType_t x1second = pdMS_TO_TICKS( DELAY_1_SECOND );//delay of 1 second

       while(1)
       {   sw=XGpio_DiscreteRead(&InpInst, SW_CHANNEL);
           if(sw==0b0010 || sw==0b0011){ //if sw 1 is on and sw 2, 3, are off, sw 0 is ignored
               vTaskSuspend(xTaskBTN);
               printf("\n TASK BTN IS SUSPENDED\n");
           }
           else if(sw==0b0100 || sw==0b0101){ //if sw2 is on and sw 1 and sw 3 are off, sw 0 is ignored
               vTaskResume(xTaskBTN);
               printf("\n TASK BTN IS RESUMED\n");
           }
           else if(sw==0b1000 || sw==0b1001){ //if sw3 is on and sw 1 and sw 2 are off, sw 0 is ignored
               vTaskSuspend(xTaskLED);
               printf("\n TASK LED IS SUSPENDED\n");
           }
           else if(sw==0b0110 || sw==0b0111){ //if sw3 is off and sw 1 and sw 2 are on, sw 0 is ignored
               vTaskResume(xTaskLED);
               printf("\n TASK LED IS RESUMED\n");
           }
           else{ //if multiple switches are pressed, or sw 1 is flipped nothing happens
               asm("NOP");
           }
           /*print the received SW data and wait 1 second before repeating loop*/
           printf("\n sw= %d", sw);
           vTaskDelay(x1second);
   }
}
```

Figure 12. TaskSW, suspending or resuming based upon switch inputs

Verification

The hardware demo may be examined using the following link:

https://youtu.be/9_gL5EQhvb4

In addition to the hardware demo above, each task was executed correctly and verified through the SDK terminal using printf statements. The lab does include one case of overlap, which is when btn3 is high and *TaskSW* is suspended. While switches 1-3 are suspended, sw0 is ignored in this task and still controls the incrementing or decrementing of the LEDs. To combat this, I added two additional else if statements within *TaskLED* and created a temporary variable inp. Both btn, debounce, inp, and sw are declared as static integers to be used for when debounce and btn are equal to 0b01000, which activates the *TaskSW* suspension, then the temporary value inp is now set to the current led and continues to increment or decrement depending on the last known value of sw0. When debounce and btn=0b1000, *TaskSW* is resumed, and inp is once again set equal to sw, which uses sw0 as an input to determine the LED output pattern.

```
        else if(debounce==0b0100 && btn==0b0100){ //If TaskSW is suspended do not recognize Sw0 as an input
            inp=led;                              //keep incrementing or decrementing
        }
        else if(debounce==0b1000 && btn==0b1000){ //If TaskSW is resumed recognize Sw0 as an input
            inp=sw;                               //switch incrementing or decrementing depending on Sw
        }
        /*print the received LED data and wait 1 second before repeating loop*/
        printf("\n LEDs: %d", led);
        vTaskDelay(x1second);
    }
}
```

Figure 13. Suspending SW0 when TaskSW is suspended

I think the hardware demo best describes how each task is functioning properly, however I will show a few screenshots below to prove the LED suspension. The first figure shows when button 1 is enabled and all switches except sw0 are low, led sits at 3:

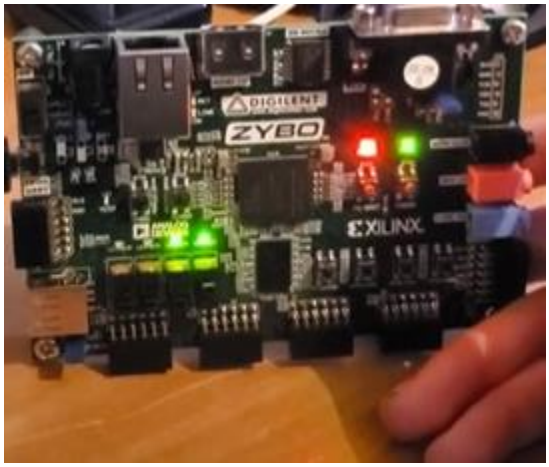

Figure 14. LED suspension from button 0

Upon holding button 2, the led resumes and decrements to 2.

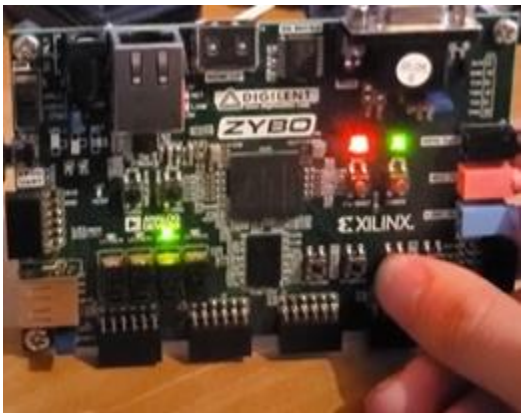

Figure 15. LED resumes from button 1 and sits at 2

The LEDs can also be suspended by using the switches. When SW3 is high the LEDs get suspended:

Figure 16. LED suspension from Switches, before and after switching sw3

Of course, resuming the LEDs can be accomplished by switching both SW1 and SW2 at the same time:

Figure 17. LEDs are resumed from sw1 and sw2 and continue to decrement

Conclusion

This lab serves as an introduction to FreeRTOS, which will be used throughout the course as a real time operating system, using task management to set priorities and split equal amounts of CPU to each task, given their priority. The lab modified the Vivado hardware design used in Lab 1 to include switches as an additional output and was programmed using SDK/eclipse as an IDE. Upon modifying the *FreeRTOS_Hello_World.c* sample program, I was able to successfully create three tasks, each set to priority 1, and display the correct LED outputs, given the button and switch inputs.

Appendix

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 Lab 2 ECE3623 Spring 2021
/* FreeRTOS includes. */
#include "FreeRTOS.h"
#include "task.h"
/* Xilinx includes. */
#include "xil_printf.h"
#include "xparameters.h"
#include "xgpio.h"
//definitions
#define DELAY_1_SECOND          1000UL
#define printf xil_printf
/*-----------------------------------------------------------*/
```

```
/* The tasks as described at the top of this file. */
static void taskLED( void *pvParameters );
static void taskBTN( void *pvParameters );
static void taskSW( void *pvParameters );
/*-----------------------------------------------------------*/

/* The queue used by the taskLED, taskBTN, taskSW tasks, as described at the
top of this
file. */
static TaskHandle_t xTaskLED;
static TaskHandle_t xTaskBTN;
static TaskHandle_t xTaskSW;

//static int declarations to be used throughout the tasks
static int led;
static int sw;
static int inp;
static int btn;
static int debounce;
//GPIO definitions
#define LED_DEVICE_ID XPAR_AXI_GPIO_1_DEVICE_ID //GPIO device connected to LED
#define INP_DEVICE_ID XPAR_AXI_GPIO_0_DEVICE_ID //GPIO device connected to
BTNs and SW
XGpio LEDInst, InpInst;
//defining each channel
#define LED_CHANNEL 1
#define BTN_CHANNEL 1
#define SW_CHANNEL 2

//Main Function
int main( void ){
	//GPIO Driver Initialization
	int status;
	status = XGpio_Initialize(&LEDInst, LED_DEVICE_ID);		// LED
initialization
	if (status != XST_SUCCESS) {printf("LED Initialization failed"); return
XST_FAILURE;}
	XGpio_SetDataDirection(&LEDInst, 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 and SW as Inputs
	XGpio_SetDataDirection(&InpInst, BTN_CHANNEL, 0xFF);
	XGpio_SetDataDirection(&InpInst, SW_CHANNEL, 0xFF);


	/* Creating the three tasks.*/
	xTaskCreate( taskLED,					/* The function
that implements the task. */
				( const char * ) "L",		/* 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 + 1,           /* The task
runs at the idle priority. */
                                &xTaskLED );

      xTaskCreate( taskBTN,
                           ( const char * ) "B",
                           configMINIMAL_STACK_SIZE,
                           NULL,
                           tskIDLE_PRIORITY + 1,
                           &xTaskBTN );

      xTaskCreate( taskSW,
                           ( const char * ) "SW",
                           configMINIMAL_STACK_SIZE,
                           NULL,
                           tskIDLE_PRIORITY + 1,
                           &xTaskSW );

      /* Start tasks */
      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 taskLED( void *pvParameters )
{
const TickType_t x1second = pdMS_TO_TICKS( DELAY_1_SECOND ); //initialize 1
second task delay
led=0; //initialize LEDs
inp=sw;//switch input
      while(1)
      {
            inp=XGpio_DiscreteRead(&InpInst, SW_CHANNEL); //reading an input
from the switches
            //If Sw 0 is high, the sum of sw will be odd, this case which
decrements the LEDs, no matter SW 1-3
            if(sw%2==1){
                  XGpio_DiscreteWrite(&LEDInst, LED_CHANNEL, led);
                  led--;
                  if(led<=0) led=15;
```

```
			}
			//If Sw 0 is low, the sum of SW will be even, this case which
increments the LEDs, ignoring SW 1-3
			else if(sw%2==0){
				XGpio_DiscreteWrite(&LEDInst, LED_CHANNEL, led);
				led++;
				if(led==15) led=0;
			}
			else if(debounce==0b0100 && btn==0b0100){ //If TaskSW is
suspended do not recognize Sw0 as an input
				inp=led;
//keep incrementing or decrementing
			}
			else if(debounce==0b1000 && btn==0b1000){ //If TaskSW is resumed
recognize Sw0 as an input
				inp=sw;
//switch incrementing or decrementing depending on Sw
			}
			/*print the received LED data and wait 1 second before repeating
loop*/
			printf("\n LEDs: %d", led);
			vTaskDelay(x1second);
		}
}

/*-----------------------------------------------------------*/
static void taskBTN( void *pvParameters )
{const TickType_t x1second = pdMS_TO_TICKS( DELAY_1_SECOND );//delay of 1
second
	btn=0; //initialize buttons
	debounce=0; //initialize buttons

	while(1)
	{	//Read button value
		btn=XGpio_DiscreteRead(&InpInst,BTN_CHANNEL);
		vTaskDelay(x1second*2); /*Button debouncing for 2seconds,
					if the input is stable for 2 seconds
then debounce takes the input*/
		debounce=btn;
				if(debounce==0b0001 && btn==0b0001)	//if btn0 is
pressed the LEDs are suspended, ignoring other btn inputs
				{
					vTaskSuspend(xTaskLED);
					printf("\n TASK LED IS SUSPENDED\n");
				}
				else if (debounce==0b0010 && btn==0b0010) //if btn 1
is pressed the LEDs will resume, ignoring other btn inputs
				{
					vTaskResume(xTaskLED);
					printf("\n TASK LED IS RESUMED\n");
				}
				else if(debounce==0b0100 && btn==0b0100) //if btn 2
is pressed the SW task is suspended, ignoring other btn inputs
				{
```

```
                                vTaskSuspend(xTaskSW);
                                printf("\n TASK SW IS SUSPENDED\n");
                        }

                        else if(debounce==0b1000 && btn==0b1000) //if btn 3
is pressed the SW task will resume,ignoring other btn inputs
                        {
                                vTaskResume(xTaskSW);
                                printf("\n TASK SW IS RESUMED\n");

                        }
                        else{ //if multiple buttons are pressed, nothing
happens
                                asm("NOP");
                        }

                /*print the received btn data, because of btn debouncing, btn
only outputs ever 2 seconds*/
                printf("\t\tbtn=%d", btn);

        }
}
/*-----------------------------------------------------------*/
static void taskSW( void *pvParameters )
{const TickType_t x1second = pdMS_TO_TICKS( DELAY_1_SECOND );//delay of 1
second

                while(1)
                {       sw=XGpio_DiscreteRead(&InpInst, SW_CHANNEL);
                        if(sw==0b0010 || sw==0b0011){ //if sw 1 is on and sw 2, 3,
are off, sw 0 is ignored
                                vTaskSuspend(xTaskBTN);
                                printf("\n TASK BTN IS SUSPENDED\n");
                        }
                        else if(sw==0b0100 || sw==0b0101){ //if sw2 is on and sw 1
and sw 3 are off, sw 0 is ignored
                                vTaskResume(xTaskBTN);
                                printf("\n TASK BTN IS RESUMED\n");
                        }
                        else if(sw==0b1000 || sw==0b1001){ //if sw3 is on and sw 1
and sw 2 are off, sw 0 is ignored
                                vTaskSuspend(xTaskLED);
                                printf("\n TASK LED IS SUSPENDED\n");
                        }
                        else if(sw==0b0110 || sw==0b0111){ //if sw3 is off and sw
1 and sw 2 are on, sw 0 is ignored
                                vTaskResume(xTaskLED);
                                printf("\n TASK LED IS RESUMED\n");
                        }
                        else{ //if multiple switches are pressed, or sw 1 is
flipped nothing happens
                                asm("NOP");
                        }
                        /*print the received SW data and wait 1 second before
repeating loop*/
```

```
                    printf("\n sw= %d", sw);
                    vTaskDelay(x1second);
        }
}
/*-----------------------------------------------------------*/
```