**Task Management with FreeRTOS on Zybo**

Benjamin A. Blouin

*benjamin.blouin@temple.edu*

**Summary**

This lab extends the previous lab using FreeRTOS to create our own tasks. All tasks have the same priority, with one task counting up on the Zybo board LEDs, a second task using buttons to suspend or resume the counting task, and a third task that uses switches to suspend the button functionality. The lab uses the same hardware setup as the previous lab, so no new block design is necessary. The lab also doesn’t use a timer or queue, so control and verification are done explicitly through tasks and user interaction.

**Introduction**

FreeRTOS tasks can suspend or resume other tasks, as well as suspend themselves. The ability to control task functioning allows for an interrupt style functionality without the confusing interrupt APIs. The buttons and switches are connected to the same GPIO block, and the LEDs are on their own. The LEDs are given a task where the increment a variable, output it to the LEDs, and then delay itself for one second. A delay can be thought of as a time-bounded self-suspend. The least significant button suspends the LED count task, and the next least significant button resumes the task. The least significant switch suspends the button task when on and resumes when off. The prioritization is given on task creation, and these values are not modified. Task control is asserted through suspends and resumes.

**Discussion**

The included libraries can be slimmed down from the original to just include the “FreeRTOS.h,” for using the general FreeRTOS libraries, “task.h,” for task-specific libraries, and “xgpio.h,” to allow usage of the GPIO blocks. A TickType\_t data type is made to hold the value for the one second delay. All task creation structures are the same, except for the name of the handler and object name. The LED task writes a value to the LEDs, suspends itself for one second, then increment the value variable. The button task checks for the value of button zero and button one, stopping and starting the LED task respectively. The switch task does the same, except with one switch, and suspends or resumes the button task. A GPIO initialization function is needed to setup the GPIO block channels and their respective data direction registers.

**Conclusions**

This lab shows how to use tasks to control other tasks, instead of creating different priorities. This is useful so priority changes don’t get lost through bad documentation. It also shows how to hierarchically control multiple levels of tasks through the passage of control methods. This task was a success and very simple.

**Appendices**

**Video**

[**Taskmanagement in FreeRTOS**](https://youtu.be/fY-qrGRVCso)

**Github**

<https://github.com/3keepmovingforward3/Embedded-System-Design-Sp19/blob/master/introToRTOS/introToRTOS.sdk/task_management/src/freertos_task_management.c>

**Code**

/\*

Benjamin Blouin

\*/

/\* FreeRTOS includes. \*/

#include "FreeRTOS.h"

#include "task.h"

/\* Xilinx includes. \*/

#include "xgpio.h"

#define DELAY\_1\_SECOND      1000UL

/\*-----------------------------------------------------------\*/

static void gpioInit();

static void prvTaskBTN( void \*pvParameters );

static void prvTaskSW( void \*pvParameters );

static void prvTaskLED( void \*pvParameters );

/\*-----------------------------------------------------------\*/

#define TIMER\_ID    1

static TaskHandle\_t xTaskBTN;

static TaskHandle\_t xTaskSW;

static TaskHandle\_t xTaskLED;

static XGpio swibtwInst, ledInst;

static BaseType\_t xLEDCount;

const TickType\_t x1second = pdMS\_TO\_TICKS( DELAY\_1\_SECOND );

int main( void )

{

    gpioInit();

    XGpio\_DiscreteWrite(&ledInst, 1, xLEDCount); //clear leds

    xTaskCreate(    prvTaskBTN,

                    ( const char \* ) "TaskBTN",

                    configMINIMAL\_STACK\_SIZE,

                    NULL,

                    tskIDLE\_PRIORITY + 1,

                    &xTaskBTN );

    xTaskCreate(    prvTaskLED,

                    ( const char \* ) "TaskLED",

                    configMINIMAL\_STACK\_SIZE,

                    NULL,

                    tskIDLE\_PRIORITY + 1,

                    &xTaskLED );

    xTaskCreate( prvTaskSW,

                     ( const char \* ) "TaskSW",

                     configMINIMAL\_STACK\_SIZE,

                     NULL,

                     tskIDLE\_PRIORITY + 1,

                     &xTaskSW );

    /\* Start the tasks and timer running. \*/

    vTaskStartScheduler();

    while(1);

}

/\*-----------------------------------------------------------\*/

static void prvTaskLED( void \*pvParameters )

{

    while(1)

    {

        XGpio\_DiscreteWrite(&ledInst,1,xLEDCount);

        vTaskDelay(x1second);

        xLEDCount++;

    }

}

/\*-----------------------------------------------------------\*/

static void prvTaskSW( void \*pvParameters )

{

    while(1)

    {

        if(XGpio\_DiscreteRead(&swibtwInst,2)==0)

            {vTaskResume(xTaskBTN);}

        else if (XGpio\_DiscreteRead(&swibtwInst,2)==1)

            {vTaskSuspend(xTaskBTN);}

    }

}

/\*-----------------------------------------------------------\*/

static void prvTaskBTN( void \*pvParameters )

{

    while(1)

    {

        if(XGpio\_DiscreteRead(&swibtwInst,1)==2)

            {vTaskResume(xTaskLED);}

        else if (XGpio\_DiscreteRead(&swibtwInst,1)==1)

            {vTaskSuspend(xTaskLED);}

    }

}

/\*-----------------------------------------------------------\*/

static void gpioInit(){

    XGpio\_Initialize(&swibtwInst,XPAR\_AXI\_GPIO\_0\_DEVICE\_ID);

    XGpio\_Initialize(&ledInst,XPAR\_AXI\_GPIO\_1\_DEVICE\_ID);

    XGpio\_SetDataDirection(&swibtwInst,1,0xf);

    XGpio\_SetDataDirection(&swibtwInst,2,0xf);

    XGpio\_SetDataDirection(&ledInst,1,0x0);

}

A screenshot of a cell phone

Description automatically generated

Figure 1 Block Diagram