Lab 7 Task Management with FreeRTOS on Zybo

Christopher Antochy

[*tuh03116@temple.edu*](mailto:tuh03116@temple.edu)

**Summary**

This lab explores the task management system of FreeRTOS on the Zybo board. Using a previously built hardware design on the Zybo using Vivado with 2 AXI GPIOs for LED outputs and switch and button inputs and a 7 Zynq processor, a template for running FreeRTOS hello world is modified to run a series of tasks with using the AXI GPIO inputs and outputs that include management functions from FreeRTOS for further control. This is all coded and run on hardware through Xilinx SDK. The results are successful and shown in a video.

**Introduction**

FreeRTOS is an open source real-time operating system that can be used on embedded systems. In this lab, the operating system will be used to apply control functionality to a series of tasks involving I/O. Like in most operating systems, processes or tasks follow simple finite state machines that control the flow of jobs that run on the CPU. In FreeRTOS, the task state diagram shown in Figure 1 highlights the state transitions of tasks.



Figure Task State Diagram

When tasks are created, they approach the ready state, a priority queue the scheduler manages to decide what tasks the CPU will take next. Tasks in the ready state may also be suspended by a function call. To the right is the running state, a state for when tasks are being run by the CPU. Tasks that are running can either be blocked, made ready, or suspended to remove them from the CPU and set them to the appropriate state. Tasks can suspend through function calls like vTaskSuspend(). They can be sent back to ready when scheduler interactions happen like a higher priority tasks has become ready. A block can occur when a delay has been called in the running task and idles the CPU. Tasks that are blocked can either be suspended by function call or made ready by an event handler through the OS API like completion of a delay. Suspended tasks may only become ready by calling the vTaskResume() function. Through these controls, tasks follow this state flow so that the CPU is shared appropriately between the tasks.

The hardware design for this lab is shown in Figure 2.

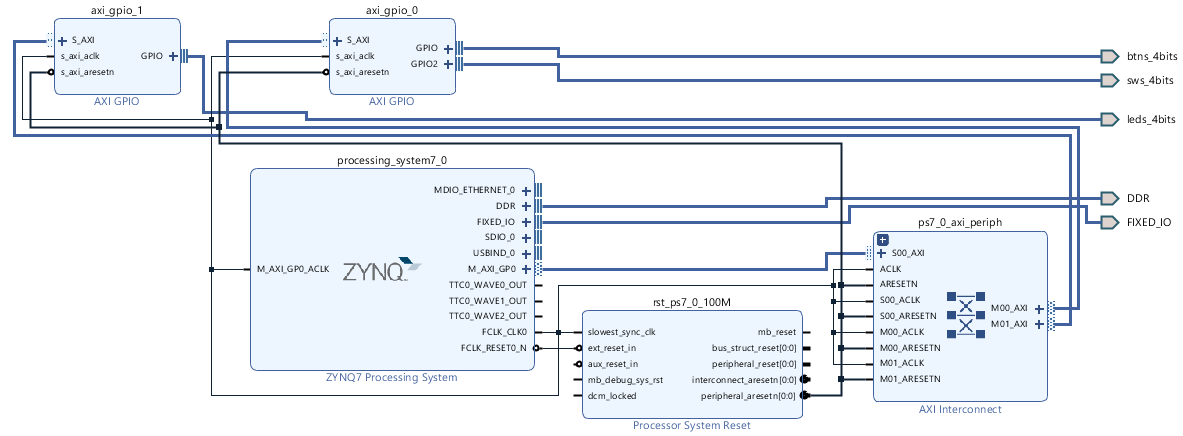


Figure Hardware Design

The most important elements of the design are the Zynq processor and the AXI peripheral interface with GPIO0 and GPIO1. GPIO0 uses channel 0 for 4 button inputs and channel 1 for 4 switch inputs. GPIO1 uses channel 0 for 4 LED outputs. This design in exported into Xilinx SDK where a project is made for FreeRTOS hello world which has a code template shown in Appendix 1. This code provides basic syntax for appropriately making tasks in FreeRTOS. Next are the tasks that are required for the lab.

Task1:

Create a FreeRTOS task *TaskLED* that increments a 4-bit counter and displays the results in the LEDs with an appropriate delay at the Idle task priority +1 (*tskIDLE\_PRIORITY+1*)

Task2:

Create a FreeRTOS task *TaskBTN* that reads the BTNs at the Idle task priority+ 1. Within the FreeRTOS task *TaskBTN* if BTN0 is depressed *TaskLED* is suspended (*vTaskSuspend()*). If BTN1 is depressed *TaskLED* is resumed (*vTaskResume()*).

Because of BTN *bounce* you should make sure the BTN is no longer depressed before executing the FreeRTOS task management function.

Task3:

Create a FreeRTOS task *TaskSW* that reads the SWs at the Idle task priority+ 1. Within the FreeRTOS task *TaskSW* if SW0 is ON *TaskBTN* is suspended (*vTaskSuspend()*). If SW0 if OFF *TaskBTN* is resumed (*vTaskResume()*).

Additionally, the functions for task suspend and resume will use the TaskHandle\_t variables made like the ones used in Appendix 1. Also the tasks should be created with size configMINIMAL\_STACK\_SIZE directive. The video in Appendix 3 will demonstrate all the possibilities of the tasks.

**Discussion**

The code is written in Appendix 2. First thing that needed to be added was the xgpio.h library to use the XGPIO’s for input and output. Next the XGPIOs needed to defined with as well as instance declarations. The task function name and TaskHandle\_t were also made. Then, in main the XGPIOs needed to be instantiated, data directions defined for XPGIO channels, tasks needed to be created with appropriate names, configMINIMAL\_STACK\_SIZE for stack size, and tskIDLE\_PRIORITY+1 task priority, and the scheduler needed to be started. From here we’ll talk about each individual task.

For task 1, the assignment wanted an LED counter with an appropriate wait time. To do this, I defined a TickType\_t for 1 second delay using 1000UL from pdMS\_TO\_TICKS function in FreeRTOS. Next I set the for loop that never ends for the task. Inside is a discrete write to the LED channel of XPGIO1 channel 0 with counter and an increment. Then, vTaskDelay for 1 second is called which blocks the task until the delay is complete. By using the delay to block the task, this will allow the other tasks to round robin since they all share the same priority until the delay is finished and this task is made ready again to the be run. The result of this can be found in Appendix 3.

For task 2, the assignment wanted button 0 when depressed to suspend the LED task and resume the LED task when button 1 was depressed. In order to handle button value interpretation, I started the task in a for loop and in each iteration, the XGPIO0 channel 0 is read for the button value. Next in the loop, some conditional logic looks for state change in a pressed flag for buttons 0 and 1. If state change occurs for button 0 where it was high but becomes low, a depress has occurred and vTaskSuspend is executed on LED task. This will definitely suspend LED task until the depression of button 1. If state change occurs for button 1 where it was high but becomes low, a depress has a occurred and vTaskResume is executed on LED task. Appendix 3 shows the action of press of button 0 and button 1 not affecting the state of LED task but when they are depressed they do make the changes to LED task. This action sequence relies on the repetition of the for loop and the scheduler providing the task a time share by switching it between running and ready through the Priority queue. The task is always given an appropriate amount of time to operate and look for button presses because it is of equal priority to the other tasks and gets equal time share that is quick enough for the action of pressing and depressing the buttons to not be affected.

For task 3, the assignment wanted switch 0 when depressed to suspend BTN task and resume the BTN task when switch 1 was depressed. In order to handle switch value interpretation, I started the task in a for loop and in each iteration, the XGPIO0 channel 1 is read for the switch value. Next in the loop, some conditional logic looks for state change in a pressed flag for switches 0 and 1. If state change occurs for switch 0 where is was high but becomes low, a depress has occurred and vTaskSuspend is executed on BTN task. This will definitely suspend BTN task until the depression of switch 1. If state change occurs for switch 1 where is was high but becomes low, a depress has occurred and vTaskResume is executed on BTN task. Appendix 3 shows the action of press on switch 0 and switch 1 not affecting the state of BTN task but when they are depressed they do make changes to BTN task. Because this task can suspend BTN task and BTN task can suspend LED task, situations can occur in which only this task runs. It’s also possible for only this task and LED task to run. Also it is possible for only this task and BTN task to run. This action sequence relies on the repetition of the for loop and the scheduler providing the task a time share by switching it between running and ready through the Priority queue. The task is always given an appropriate amount of time to operate and look for switch presses because it of equal priority to the other tasks and gets equal time share that is quick enough for the action of pressing and depressing the switches to not be affected.

**Conclusion**

This lab is an introduction to the task management system of FreeRTOS on the Zybo board. A hardware design for the Zybo was designed in Vivado and demonstrated in Xilinx SDK on the actual board. It uses GPIO0 and 1 for I/O operations that can then be managed with the tools provided by the FreeRTOS operating system. The tasks in this lab involves using buttons, switches and LEDs that are worked through in code and shown with a video.

**Appendix**

1. Template Code

/\* FreeRTOS includes. \*/

**#include** "FreeRTOS.h"

**#include** "task.h"

**#include** "queue.h"

**#include** "timers.h"

/\* Xilinx includes. \*/

**#include** "xil\_printf.h"

**#include** "xparameters.h"

**#define** TIMER\_ID 1

**#define** DELAY\_10\_SECONDS 10000UL

**#define** DELAY\_1\_SECOND 1000UL

**#define** TIMER\_CHECK\_THRESHOLD 9

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

/\* The Tx and Rx tasks as described at the top of this file. \*/

**static** **void** **prvTxTask**( **void** \*pvParameters );

**static** **void** **prvRxTask**( **void** \*pvParameters );

**static** **void** **vTimerCallback**( TimerHandle\_t pxTimer );

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

/\* The queue used by the Tx and Rx tasks, as described at the top of this

file. \*/

**static** TaskHandle\_t xTxTask;

**static** TaskHandle\_t xRxTask;

**static** QueueHandle\_t xQueue = NULL;

**static** TimerHandle\_t xTimer = NULL;

**char** HWstring[15] = "Hello World";

**long** RxtaskCntr = 0;

**int** **main**( **void** )

{

**const** TickType\_t x10seconds = pdMS\_TO\_TICKS( DELAY\_10\_SECONDS );

xil\_printf( "Hello from Freertos example main\r\n" );

/\* Create the two tasks. The Tx task is given a lower priority than the

Rx task, so the Rx task will leave the Blocked state and pre-empt the Tx

task as soon as the Tx task places an item in the queue. \*/

xTaskCreate( prvTxTask, /\* The function that implements the task. \*/

( **const** **char** \* ) "Tx", /\* 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. \*/

&xTxTask );

xTaskCreate( prvRxTask,

( **const** **char** \* ) "GB",

configMINIMAL\_STACK\_SIZE,

NULL,

tskIDLE\_PRIORITY + 1,

&xRxTask );

/\* Create the queue used by the tasks. The Rx task has a higher priority

than the Tx task, so will preempt the Tx task and remove values from the

queue as soon as the Tx task writes to the queue - therefore the queue can

never have more than one item in it. \*/

xQueue = xQueueCreate( 1, /\* There is only one space in the queue. \*/

**sizeof**( HWstring ) ); /\* Each space in the queue is large enough to hold a uint32\_t. \*/

/\* Check the queue was created. \*/

configASSERT( xQueue );

/\* 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. \*/

xTimer = xTimerCreate( (**const** **char** \*) "Timer",

x10seconds,

pdFALSE,

(**void** \*) TIMER\_ID,

vTimerCallback);

/\* 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** **prvTxTask**( **void** \*pvParameters )

{

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

**for**( ;; )

{

/\* Delay for 1 second. \*/

vTaskDelay( x1second );

/\* Send the next value on the queue. The queue should always be

empty at this point so a block time of 0 is used. \*/

xQueueSend( xQueue, /\* The queue being written to. \*/

HWstring, /\* The address of the data being sent. \*/

0UL ); /\* The block time. \*/

}

}

**static** **void** **prvRxTask**( **void** \*pvParameters )

{

**char** Recdstring[15] = "";

**for**( ;; )

{

/\* Block to wait for data arriving on the queue. \*/

xQueueReceive( xQueue, /\* The queue being read. \*/

Recdstring, /\* Data is read into this address. \*/

portMAX\_DELAY ); /\* Wait without a timeout for data. \*/

/\* Print the received data. \*/

xil\_printf( "Rx task received string from Tx task: %s\r\n", Recdstring );

RxtaskCntr++;

}

}

* 1. Source Code TimerCallback Function

**static** **void** **vTimerCallback**( TimerHandle\_t pxTimer )

{

**long** lTimerId;

configASSERT( pxTimer );

lTimerId = ( **long** ) pvTimerGetTimerID( pxTimer );

**if** (lTimerId != TIMER\_ID) {

xil\_printf("FreeRTOS Hello World Example FAILED");

}

/\* If the RxtaskCntr is updated every time the Rx task is called. The

Rx task is called every time the Tx task sends a message. The Tx task

sends a message every 1 second.

The timer expires after 10 seconds. We expect the RxtaskCntr to at least

have a value of 9 (TIMER\_CHECK\_THRESHOLD) when the timer expires. \*/

**if** (RxtaskCntr >= TIMER\_CHECK\_THRESHOLD) {

xil\_printf("FreeRTOS Hello World Example PASSED");

} **else** {

xil\_printf("FreeRTOS Hello World Example FAILED");

}

vTaskDelete( xRxTask );

vTaskDelete( xTxTask );

}

1. Modified Code

/\* FreeRTOS includes. \*/

**#include** "FreeRTOS.h"

**#include** "task.h"

**#include** "queue.h"

**#include** "timers.h"

/\* Xilinx includes. \*/

**#include** "xil\_printf.h"

**#include** "xparameters.h"

**#include** "xgpio.h"

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

**static** **void** **TaskBTN**( **void** \*pvParameters );

**static** **void** **TaskLED**( **void** \*pvParameters );

**static** **void** **TaskSW**( **void** \*pvParameters );

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

**#define** IN\_DEVICE\_ID XPAR\_AXI\_GPIO\_0\_DEVICE\_ID

**#define** OUT\_DEVICE\_ID XPAR\_AXI\_GPIO\_1\_DEVICE\_ID

XGpio OUTInst, INInst;

**static** TaskHandle\_t xTaskBTN;

**static** TaskHandle\_t xTaskLED;

**static** TaskHandle\_t xTaskSW;

**int** **main**( **void** )

{

**int** status;

status = XGpio\_Initialize(&OUTInst, OUT\_DEVICE\_ID);

**if**(status != XST\_SUCCESS) **return** XST\_FAILURE;

status = XGpio\_Initialize(&INInst, IN\_DEVICE\_ID);

**if**(status != XST\_SUCCESS) **return** XST\_FAILURE;

// Set OUT direction to outputs for channel 1

XGpio\_SetDataDirection(&OUTInst, 1, 0x00);

// Set all IN direction to inputs for channel 1

XGpio\_SetDataDirection(&INInst, 1, 0xFF);

// set all IN direction to inputs for channel 2

XGpio\_SetDataDirection(&INInst, 2, 0xFF);

xTaskCreate( TaskBTN, /\* 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+1, /\* The task runs at the idle priority. \*/

&xTaskBTN );

xTaskCreate( TaskLED,

( **const** **char** \* ) "LED",

configMINIMAL\_STACK\_SIZE,

NULL,

tskIDLE\_PRIORITY + 1,

&xTaskLED );

xTaskCreate( TaskSW,

( **const** **char** \* ) "SW",

configMINIMAL\_STACK\_SIZE,

NULL,

tskIDLE\_PRIORITY + 1,

&xTaskSW );

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( 1000UL );

**int** counter = 0;

**for**( ;; )

{

XGpio\_DiscreteWrite(&OUTInst, 1, counter++);

/\* Delay for 1 second. \*/

vTaskDelay( x1second );

}

}

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

**static** **void** **TaskBTN**( **void** \*pvParameters )

{

**int** btn\_value = 0;

**int** btn\_0\_press = 0;

**int** btn\_1\_press = 0;

**for**( ;; )

{

btn\_value = XGpio\_DiscreteRead(&INInst, 1);

**if**((btn\_value&0x1) == 0){

**if**(btn\_0\_press == 1) vTaskSuspend(xTaskLED);

btn\_0\_press=0;

}

**else** btn\_0\_press=1;

**if**((btn\_value&0x2) == 0){

**if**(btn\_1\_press == 1) vTaskResume(xTaskLED);

btn\_1\_press=0;

}

**else** btn\_1\_press=1;

}

}

**static** **void** **TaskSW**( **void** \*pvParameters )

{

**int** sw\_value = 0;

**int** sw\_0\_press = 0;

**int** sw\_1\_press = 0;

**for**( ;; )

{

sw\_value = XGpio\_DiscreteRead(&INInst, 2);

**if**((sw\_value&0x1) == 0){

**if**(sw\_0\_press == 1) vTaskSuspend(xTaskBTN);

sw\_0\_press=0;

}

**else** sw\_0\_press=1;

**if**((sw\_value&0x2) == 0){

**if**(sw\_1\_press == 1) vTaskResume(xTaskBTN);

sw\_1\_press=0;

}

**else** sw\_1\_press=1;

}

}

1. Video

<https://www.youtube.com/watch?v=agGY6v3ao5Y>