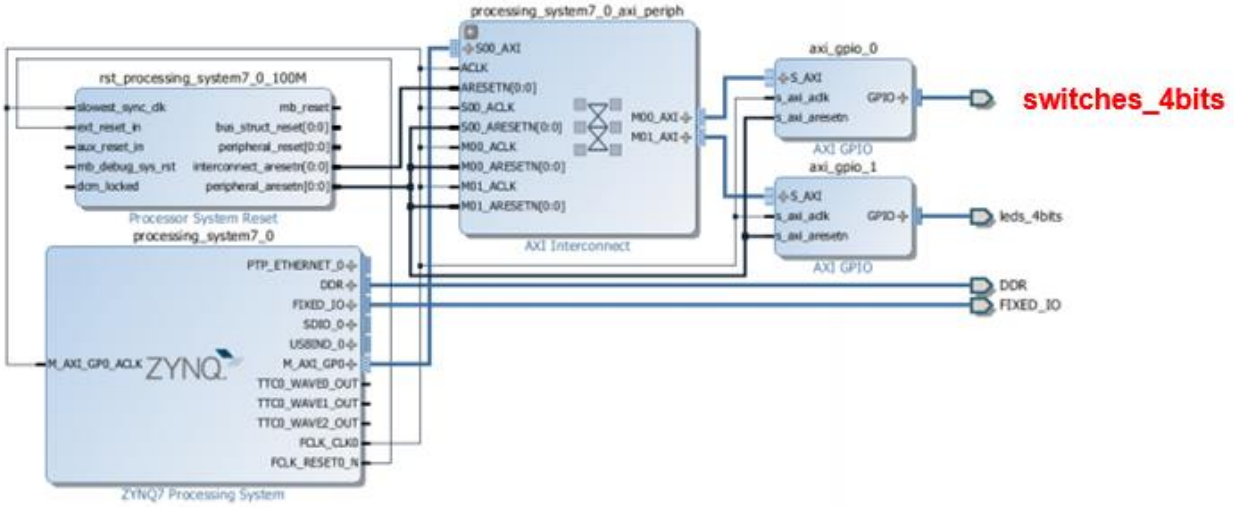
ECE3623 Embedded System Design Laboratory

Dennis Silage, PhD
silage@temple.edu

Task Management with FreeRTOS on Zybo

In this Laboratory you will investigate task management in the FreeRTOS environment on the Zybo board. In the previous Laboratory you created a Xilinx Vivado block design with two axi_gpio IP modules: axi_gpio_0 for the Zybo slide switches (SW) and axi_GPIO_1 for the Zybo LEDs as shown below. The operating system was *standalone* (single tasking) by default.

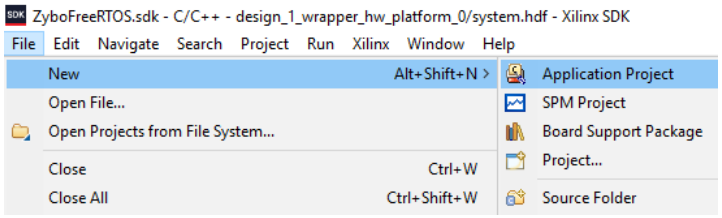


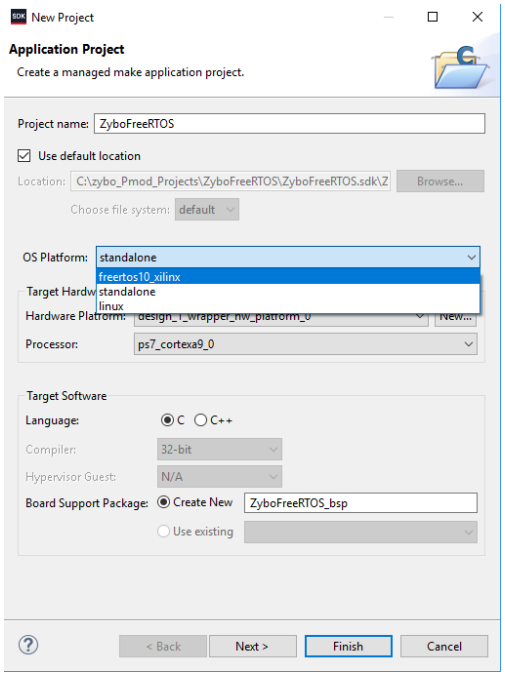
The SWs GPIO is at *XPAR_AXI_GPIO_0_DEVICE_ID* and the LEDs GPIO is at *XPAR_AXI_GPIO_1_DEVICE_ID*. SDK was imported with the hardware specifications from Vivado in the previous Laboratory. If required, complete the Vivaldi hardware designs and launch SDK.

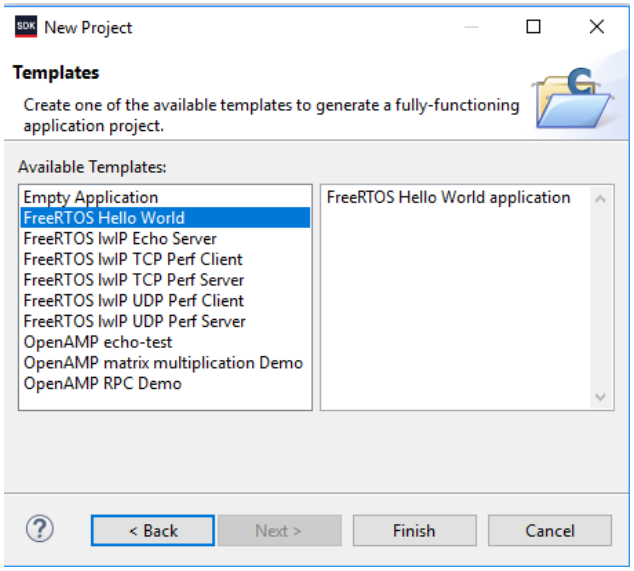
In SDK do *File…New…Application Project* then enter a new project name *initials_FreeRTOStm.prj* and select *freertos10_xilinx* as the OS Platform. Not *Finish* but *Next* to continue. In the *New Project…Templates* select *FreeRTOS Hello World* and *Finish* to continue. The *Project Explorer* shows the *freertos_hello_world.c* as the *main()* with the tasks required for the FreeRTOS OS and will be used as a template for this Laboratory.

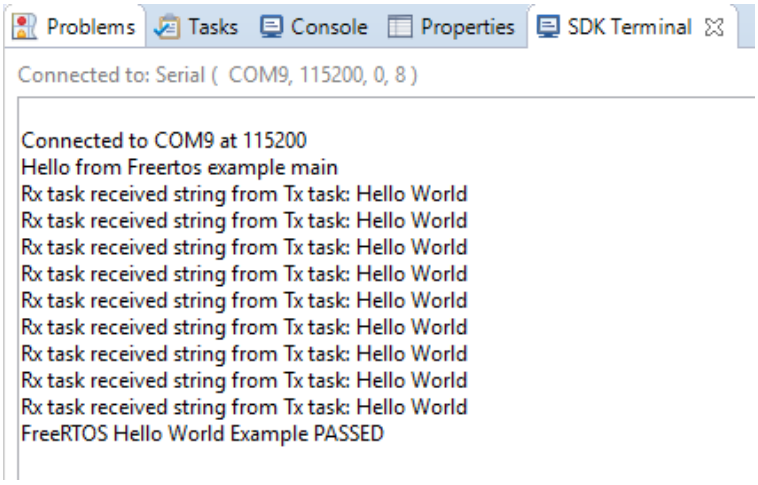
To test the SDK application, program the FPGA then start the SDK Terminal. Next, do *Run As…4 Launch on Hardware (GDB)*. The output in the SDK Terminal is shown below and is at the bootm of the standard SDK IDE. You will have to use the COM port for this

serial data (COM 9 here) and a bit rate of 115 200 b/sec with 8 bits and no parity as shown below.









The *freertos_hello_world.c* is available on Canvas and is also shown below. There are four FreeRTOS include files and two Xilinx include files. Refer to the *Introduction to*

FreeRTOS PowerPoint and the *Mastering the FreeRTOS Real Time Kernel* pdf on Canvas for a discussion of the FreeRTOS functions used in this example program.

Using the *freertos_hello_world.c* file as a template the Laboratory task is to have a new *main*() application (using another name in the *.c* file) that performs the following FreeRTOS task management operations.

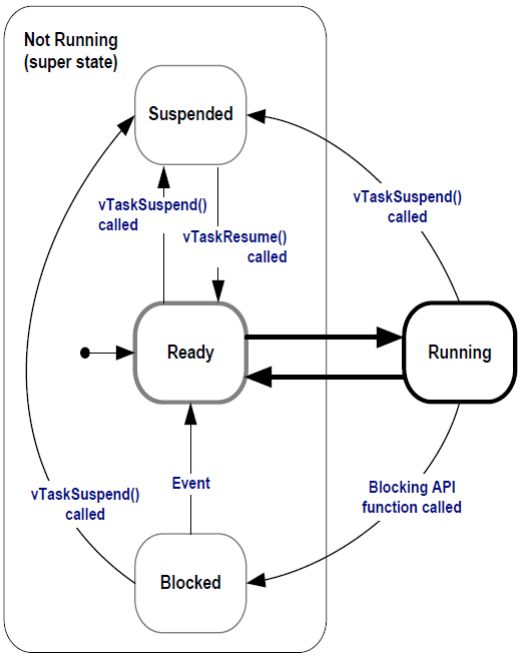
1. Create a FreeRTOS task *TaskLED* that has the LED count and display start at 0 (0000) and increase as a somewhat random sequence as listed then repeats the sequence starting at 0 (with a delay of approximately 1 second and at the Idle task priority +1 (*tskIDLE_PRIORITY+1*). This was one of the tasks in Lab 1.

0000, 1001, 0110, 1010, 0001, 0101, 1000, 0100, 1100, 1110, 0010, 1011, 1101, 0011, 0111, 1111

2. Create a FreeRTOS task *TaskSW* that reads the SWs at the Idle task priority+ 1. Within the FreeRTOS task *TaskSW* if SW0 is ON and no other SWs are ON then *TaskLED* is suspended (*vTaskSuspend()*). If SW1 is ON and no other SWs are ON then *TaskLED* is resumed (*vTaskResume()*).

- *The FreeRTOS timer (vTimerCallback) and queue (xQueue…) functions are not used in this Lab 2 and should be removed.*

You should discuss and demonstrate *all possible conditions* of the task management interactions between the SWs and the resulting LEDs.



The FreeRTOS the task management functions *vTaskSuspend()* and *vTaskResume()* utilizes the *handles* of the created tasks and should use the *configMINIMAL_STACK_SIZE* which is the default.

The completed Laboratories should be archived on your laptop and will form the basis of SNAP Quizzes and Exams.

You are to use the *Project Report Format* posted on *Canvas*. You are to upload your *Report* to Canvas for time and date stamping to avoid a late penalty. This Laboratory is for the week of January 27th and due no later than 11:59 PM Sunday February 2nd.

Freertos_hello_world.c

```
/* 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(          /* 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++;
		}
}

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 );
}
```