# Lab 03

**Due** No Due Date **Points** 10

# Instructions

Complete the following procedures. When finished make sure your code is checked into gitlab (**gitlab.cset.oit.edu (https://gitlab.cset.oit.edu/)**).

# Procedure

In the last lab we explored more of the TASK API of FreeRTOS by creating, destroying, delaying, suspending, and resuming tasks. We used the task handles set by the xTaskCreate() function to manipulate the tasks. We also used the vTaskDelay() function to cause the tasks to sleep for a specified time.

In this lab, we will create three main control tasks. Only one of these will be active at a time (the other two will be in a suspended state). This task will poll the onboard button and the two external buttons we wired up last week. It will then take the following actions. If the **EXT_SW1** button is pressed, we will decrease the amount of delay for the currently selected **LED**. If **EXT_SW2** is pressed, we will increase the amount of delay for the current **LED**. **SW0** will cause us to move to the next **LED**. There will also be three independent LED tasks that will be blinking LED's (**EXT_LED1**, **EXT_LED2**, and **EXT_LED3**). You will also create one additional task to send status messages through the **UART** and of course the heartbeat task we implemented last week as well. You will also create 4 Queues to talk between these tasks. Each Control Task will communicate to one LED task through one Queue. The last Queue will be for the UART communication.

Even if there are many ways to model this function, you are required to follow the program architecture specified here. You are only to use the FreeRTOS functions provided as part of this Lab 3 Description.

# Application Function

## Queue Creation

You will create 4 queues for this application. The queues should have a depth of 5. For the **LED** queue's you should create a custom **enum** message. The UART queue should be set to a size of char [50]. You will figure out a method to get the Queue handles to the appropriate tasks.

## MainControl Tasks

The MainControl task will poll the status of all three switches (just as we did in lab 2). The switches should be debounced with a three stage strategy: Read the switch, Delay 10ms using a vTaskDelay, then read switch again to verify the change. In addition you should implement a lockout feature in your driver only allowing a switch to perform a single action on a press. There will be three of these tasks (only 1 active at a time). Each of these task will send messages via a queue to its corresponding LED task. The switches will perform the

following function.

### EXT_SW1

This Switch will decrease the delay time of the corresponding LED task. It will accomplish this by sending a message to this task via the queue setup for this pair using the API function vQueueSendToBack() function. The message will tell the LED task to decrease its delay time.

### EXT_SW2

This Switch will increase the delay time of the corresponding LED task. It will accomplish this by sending a message to this task via the queue setup for this pair. It will accomplish this by calling vQueueSendToBack() function.

### SW0

This Switch will cause the next LED to be selected. It will accomplish this by resuming the next MainControl (vTaskResume) Task and then suspending (vTaskSuspend) itself. So if **LED1** is the current MainControl Task then after the SUSPEND/RESUME **LED2** will be the active MainControl Task. During this operation a message will also be sent to the **UART** task to identify the new active **LED**. This message will be in the form of a char * and contain a message stating “LED N IS NOW ACTIVE”. Make sure that it includes the appropriate Line feeds to move cursor to the next line.

The polling of the switches will happen in sequence and actions will be taken accordingly. After all three switches have been polled, the task will perform a vTaskDelay() for 100ms. Error checking must be implemented to assure that the Message was sent to the appropriate queue with the vQueueSendtoBack().

**Suggestion:** create an enum for the message to say either increment or decrement the delay time.

## LED Tasks

The LED tasks will blink the corresponding **LED** on and off. It will do this at an initial rate of 500ms. It will also check its Queue to see if it has a message waiting (uxQueueMessagesWaiting). If there is a message waiting it should read the message (xQueueReceive) and perform the necessary action. The wait time to the xQueueReceive() should be set to 0. The action will be either to increase or decrease the delay time in increments of 50ms. This should be bounded to a MAX delay of 1000ms and a MIN delay of 200ms. There will be three of these tasks as well. The task should use your LED Driver from Lab 2.

## Heartbeat Task

The heartbeat task will be responsible for toggling the onboard LED every second. This will give you a visual clue that the FreeRTOS system is still running.

## UART Task

The UART task will block until a message is ready for it. To accomplish this blocking set the last parameter of the xQueueReceive() to portMAX_DELAY. It will then get the message and write it out on the **UART** using your UARTPutStr(). You will need to write a very rudimentary UART driver for your program. The filenames

for the driver will be uartdrv.c and uartdrv.h. This should include three functions. The functions should be initUART(), UARTPutC() and UARTPutStr(). The initUART() function will initialize the **UART**. The UARTPutC() will print a single byte (char) to the **UART**. The UARTPutStr() function will use the UARTPutC() to write a complete string to the **UART**.

*Uartdrv.h*

You will need to include a number of things in this include file. The following defines need to be made.

```
#define EDBG_UART                       UART0
#define UART_SERIAL_BAUDRATE            115200ul
#define UART_SERIAL_CHAR_LENGTH         US_MR_CHRL_8_BIT
#define UART_SERIAL_PARITY              US_MR_PAR_NO
#define UART_SERIAL_STOP_BIT            false
#define UART_MCK_DIV                    16
#define UART_MCK_DIV_MIN_FACTOR         1
#define UART_MCK_DIV_MAX_FACTOR         65535
```

Along with the function prototypes for the driver API.

```
uint8_t initUART(Uart * pUart);
void UARTPutC(Uart * pUart, char data);
void UARTPutStr(Uart * pUart, char * data, uint8_t len);
```

## initUART()

### Function Signature

```
uint8_t initUART(Uart * pUart);
```

### Function Contents

```
uint8_t initUart(Uart * p_Uart)
{
	uint32_t cd = 0;
	uint8_t retVal = 0;

	// configure UART pins
	ioport_set_port_mode(IOPORT_PIOA, PIO_PA9A_URXD0 | PIO_PA10A_UTXD0, IOPORT_MODE_MUX_A);
	ioport_disable_port(IOPORT_PIOA, PIO_PA9A_URXD0 | PIO_PA10A_UTXD0);
	sysclk_enable_peripheral_clock(ID_UART0);

	// Configure UART Control Registers
	// Reset and Disable RX and TX
	p_Uart->UART_CR = UART_CR_RSTRX | UART_CR_RSTTX | UART_CR_RXDIS | UART_CR_TXDIS;

	// Check and configure baudrate
	// Asynchronous, no oversampling
	cd = (sysclk_get_peripheral_hz() / UART_SERIAL_BAUDRATE) / UART_MCK_DIV;
	if(cd < UART_MCK_DIV_MIN_FACTOR || cd > UART_MCK_DIV_MAX_FACTOR)
	{
```

```
		retVal = 1;
	}

	if(retVal != 1)
	{
		// Set The Actual BAUD to Control Register
		p_Uart->UART_BRGR = cd;

		// Configure Mode
		p_Uart->UART_MR = UART_SERIAL_PARITY;

		// Disable PDC Channel
		p_Uart->UART_PTCR = UART_PTCR_RXTDIS | UART_PTCR_TXTDIS;

		// Enable RX and TX
		p_Uart->UART_CR = UART_CR_RXEN | UART_CR_TXEN;
	}

	return retVal;
}
```

**Example Usage**

```
initUART(EDBG_UART);
```

## UARTPutC()

### Function Signature

```
void UARTPutC(Uart * pUart, char data);
```

### Calls Needed

```
	// Wait for Transmitter to be Ready
	while((p_Uart->UART_SR & UART_SR_TXRDY) == 0)
	{ }
```

This while loop will make sure that the transmitter is ready before you send your character to it.

```
	pUart->UART_THR
```

This is the register you send the data to.

## UARTPutStr()

### Function Signature

```
	void UARTPutStr(Uart * pUart, char * data, uint8_t len);
```

Implement a put string function by calling your vUartPutC().

# FreeRTOS API

The API documentation can be found on the web at the freeRTOS.org site at the following location:

**http://www.freertos.org/a00106.html (http://www.freertos.org/a00106.html)**

These are the Calls you are allowed to use on this lab:

- xTaskCreate()
- vTaskSuspend()
- vTaskResume()
- vTaskDelay()
- xQueueCreate()
- xQueueSendToBack()
  - Note: xQueueSend() is now deprecated
- uxQueueMessagesWaiting()
- xQueueReceive()