MIC5 Week 6 – lab exercise

Introduction

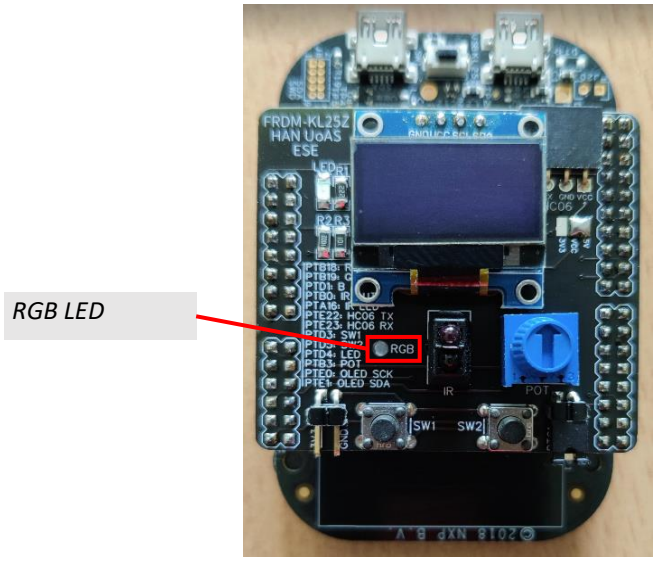
In this lab exercise we will compare task notifications to counting semaphores with respect to RAM requirements and execution time. We start with a copy of the *Week 04 – Example 01* project and replace the counting semaphore for a task notification.

Hardware

The hardware required for this project is outlined in the following table.

Description	MKL25Z128VLK4 pins	Notes
RGB LED	PTB18 (red) PTB19 (green) PTD1 (blue)	*This project uses the green LED only*

This hardware is available on the FRDM-KL25Z board and the oled shield.

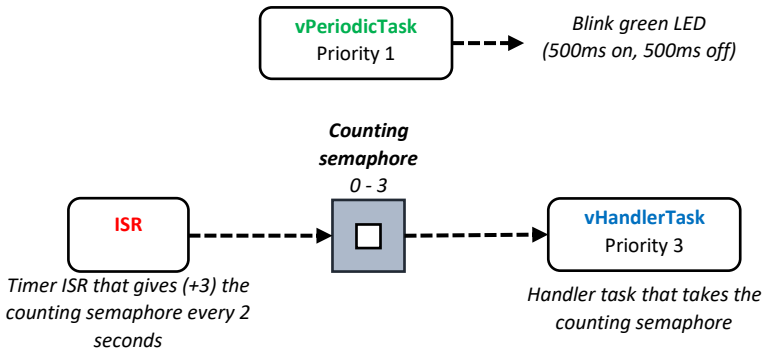


Software

A template project is provided:

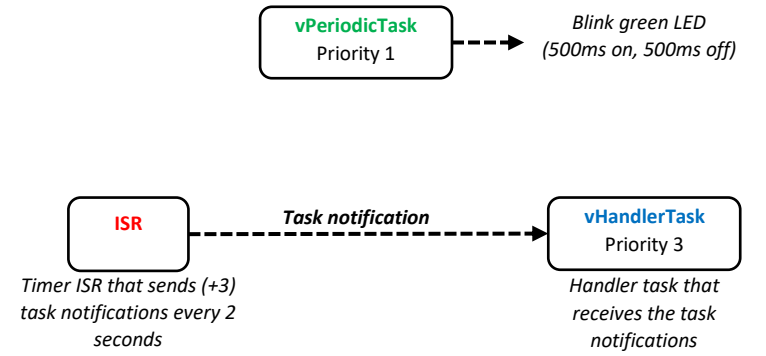
Week6 – Lab.zip

This template project is an exact copy from Week4 – Example01. The software design is depicted in the following image. For clarity, the queues and semaphore related to serial communication have been omitted. A description of the design elements follows next.



- The ISR is triggered every 2 seconds. It *gives* the counting semaphore three times. It prints a message to the serial monitor as soon as it is finished.
- The vHandlerTask is in Blocking state waiting for the counting semaphore. It will print a message as soon as it could *take* the semaphore.
- The vPeriodicTask blinks the green LED.

This software design must be updated as follows:



The counting semaphore has been replaced by a task notification. The functional behaviour of the application, as can be observed by the green LED and the output in the serial monitor, is exactly the same.

Disable the counting semaphores

Open the file FreeRTOSConfig.h. Disable the use of counting semaphores as follows:

```
#define configUSE_COUNTING_SEMAPHORES    0
```

Build the project. This will produce build errors. Remove all references to the counting semaphore from the project, so there will be no more build errors and warnings.

Create a handle to the vHandlerTask

In order to use task notifications, we need to reference the vHandlerTask by its handle. This handle must be known in both main.c and in timer.c. Declare an external TaskHandle_t variable called xHandlerTask in timer.h and declare the variable in timer.c.

Finally, make sure to add the handle when creating vHandlerTask in main().

Answer

Update the timer ISR

Instead of using three calls to the counting semaphore function xSemaphoreGiveFromISR(), use three calls to an appropriate task notification function. The ISR is located in the file timer/timer.c.

Answer

Update the task vHandlerTask

Instead of using the counting semaphore function xSemaphoreTake(), use a call to an appropriate task notification function.

Answer

Run the updated project

Build and run the project. The output in the serial monitor must be equal to the output from *Week4 – Example01*.

Comparing RAM usage and execution time

The FreeRTOS book states that using task notifications produces “lighter weight” and “faster code execution”. Let’s examine how much that is.

Ram usage

Queues are kernel objects and will therefor be stored in the FreeRTOS heap. Compare heap usage *RTOS > Heap usage* for both projects by starting the debugger and filling in the following table:

Project	**Heap Usage (%)**	**Heap Used**
Week4 – Example01		
Week6 – Lab		

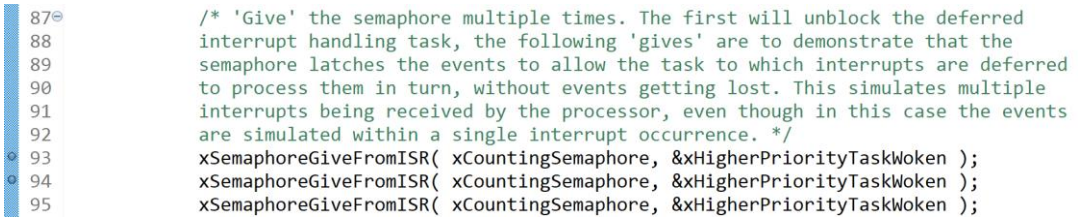
What is your conclusion?

Answer

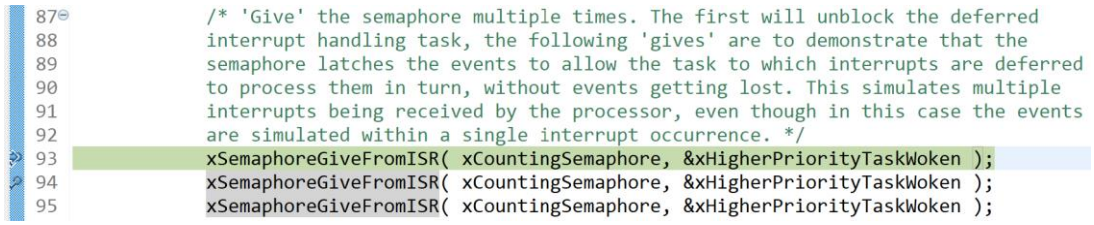
Execution time

We will measure the execution time by using one of the hardware timers. The difference in timer value before and after a function has been executed, will give us a rough estimate of its execution time. As the System Tick timer is already in use in every FreeRTOS project, we will use that timer. Proceed as follows for project *Week4 – Example01*:

1. Set breakpoints on the first two calls to the function xSemaphoreGiveFromISR() in timer.c. A breakpoint is set by double clicking the line number (93 and 94 in the following example) and visualized by a blue dot.



2. Start the debugger. Run the project and wait until code execution halts at the first breakpoint:



3. Open the Peripherals+ view: *Window > Show View > Peripherals+*.
4. In the Peripherals+ view, open *SysTick*. Here you can see the four registers associated with the System Tick timer. We are interested in the Current Value Register (CVR). Write down the CVR value in the table below.
5. Resume code execution: *Run > Resume*
6. When code execution stops at the second breakpoint, write down the CVR value in the table below.
7. Stop the debugger.

Repeat all the steps above for the *Week6 – Lab* project.

Project	**SysTick CVR value <u>before</u> executing the function** *(at the first breakpoint)*		**SysTick CVR value <u>after</u> executing the function** *(at the second breakpoint)*		**Difference in CVR values** (before-after)
	Hexadecimal	**Decimal**	**Hexadecimal**	**Decimal**	**Decimal**
Week4 – Example01					
Week6 – Lab					

What is your conclusion?

Answer