Lab 3 – Dealing with shared resources

The purpose of this lab is to demonstrate access contention problems using shared resources in a multitasking system

PART I

Tasks may unwittingly interfere with each other by making simultaneous access to the shared items. The purpose of this exercise is to demonstrate that this is a real issue. It is designed to show that interference will occur unless protection measures are used.

In this exercise the shared data item is intended to simulate the action of a read-write data store. The read-write processes are simulated using a simple software delay. We then will implement an “access indicator” to detect if a simultaneous access has occurred. The access indicator will be a binary flag “Up” and “Down” (initialized to Up). Up indicates that the resource is not in use while Down indicates the resource is in use. So when a task makes an access to the critical area, it may find:

1. Start flag is Up; all is well
2. Start flag is Down; the other task has been executing the shared code.

Implementation: Create two tasks the Green Led and the Red Led task

Task 1: Flash the Green Led

Loop:

1. Turn the Green Led on.
2. Access the shared data
3. Turn the Green Led off.
4. Delay for 0.5 seconds

End loop.

Task 2: Flash the Red Led.

Loop:

1. Turn the Red Led on.
2. Access the shared data
3. Turn the Red Led Off
4. Delay for 0.1 seconds

End loop.

Access shared data function.

1. Check if the Start flag is *Up*.

If Up then set the Start flag to *Down*

Else turn the Blue Led on.

1. Simulate read/write operations for 500 milliseconds.
2. Set Start flag to up.

Compile, download and run with both tasks active it shouldn’t be long before the Blue Led will turn on, showing that task interference has taken place.

PART II

The purpose of this exercise is to demonstrate how to eliminate resource contention using a semaphore to protect the critical code section. In order to use a semaphore in an RTOS you typically perform the following:

1. Define a reference to the semaphore
2. Define the semaphore
3. Create the semaphore

All the code needed to do this can be generated by the CubeMX. Just load the Cube project, go to FreeRTOS Configuration section and select ‘Timers and Semaphores’. Add a binary semaphore and give it a name (‘CriticalResourceSemaphore’). Generate and examine the source code and you should see the new semaphore implementation code.

For the Green and Red Led flashing tasks the following template can be used

*for (;;) {*

*HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_xx, GPIO\_PIN\_SET);*

*AccessSharedData( );*

*osDelay(DelayTimeMsec);*

*HAL\_GPIO\_WritePin(GPIOD, GPIO\_PIN\_xx, GPIO\_PIN\_RESET);*

*osDelay(DelayTimeMsec);*

*}*

Access shared data function

1. Check if the Start flag is Up.

If Up then set the Start Flag to Down

Else turn the Blue Led On.

1. Simulate the read/write operations for 1 second.
2. Turn the Blue Led off.
3. Set the Start flag to Up.

Add the code *osSemaphoreWait (CriticalResourceSemaphoreHandle, WaitTimeMilliseconds)* before the *AccessSharedData* function. Add the code *osSemaphoreRelease (CriticalResourceSemaphoreHandle)* right after the *AccessSharedData* function. Build, download and run the software. Observe the Led behavior and compare that with your work in the previous excerise.