Real-time Embedded systems

Lab2 Task Management – Part 2

Example 4 Using the Blocked state to create a delay

All the tasks created so far have been ‘periodic’ – they have delayed for a period and print out their string, before delaying once more, and so on. So the task effectively polled an incrementing loop counter using a null loop until it reached a fixed value. While executing the null loop, the task remained in the Ready state, starving the other tasks with lower priority.

This example, we will replace the polling null loop with a call to the *vTaskDelay(portTickTYpe xTicksToDelay)* library function. It places the calling task into the Blocked state for a fixed number of tick interrupts. While in the Blocked state, the task does not use any processing time. Thus, the processing time is consumed only when there is work to be done.

The input parameter *xTicksToDelay* is the number of tick interrupts that the calling task should remain in the blocked state before being transitioned back into the Ready state. For example, if a task called *vTaskDelay(100)* while the tick count was **10,000**, then it would immediately enter the blocked state and remain there until the tick count reached **10, 100**.

1. Open the FreeRTOSConfig.h header file, set the macro definition of *INCLUDE\_vTaskDelay* to **1**. This enables compiling the vTaskDelay() library function.
2. Open the main.c file, edit the infinite *for (;;)* loop body within the task function *vTaskFunction()*.

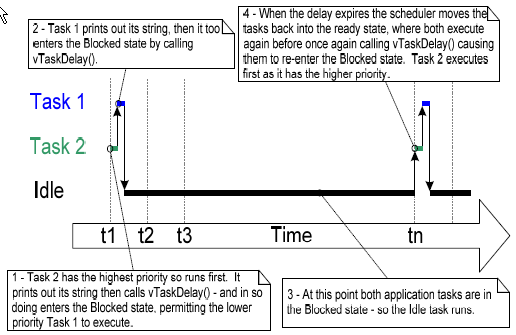
* Remove the null for loop for generating delay and all related variables:

*for( ul = 0; ul < mainDELAY\_LOOP\_COUNT; ul++ )*

* Next, call the *vTaskDelay( )* library function. The input parameter is given as “*250/portTICK\_RATE\_MS*”. The constant *portTICK\_RATE\_MS* can be used to convert milliseconds into ticks.

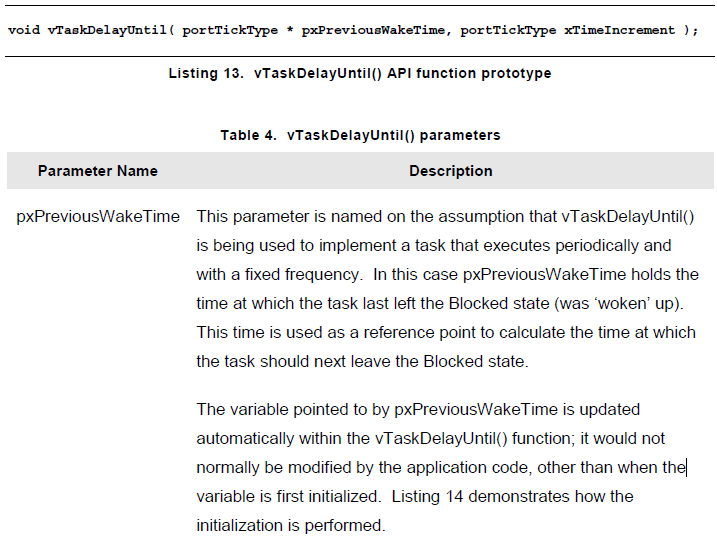
1. Now, build and run the project. You could observe that both tasks run even though they are created at different priorities.

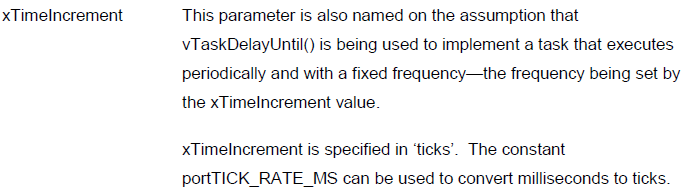
The expected execution pattern is shown below.



Example 5. Converting the example tasks to use *vTaskDelayUntil*()

The two tasks created in Example 4 are periodic tasks, but using *vTaskDelay*() does not guarantee that the frequency at which they run is fixed, as the time at which the tasks leave the Blocked state is relative to when they call *vTaskDelay*(). We solve this potential problem by using the *vTaskDelayUntil()* library function.





1. Open the FreeRTOSConfig.h header file, and set the macro definition of *INCLUDE\_vTaskDelayUntil* to **1**. This enables compiling the vTaskDelayUntil() library function.
2. Open the main.c file and edit the task function *vTaskFunction()*.

* Declare a new local variable *xLastWakeTime* with the type *portTickType*;
* Initialize the variable *xLastWakeTime* with the current tick count. Note that this is the only time we access this variable. This tick count could be obtained by calling the library function *xTaskGetTickCount*().

From this point on, xLastWakeTime is managed automatically by the *vTaskDelayUntil()* function.

* Inside the infinite *for( ;; )* loop, call *vTaskDelayUntil()* library function instead of *vTaskDelay()* to generate delay.

*vTaskDelayUntil()* requires two parameters. You need to pass the pointer of the variable xLastWakeTime (i.e., *&xLastWakeTime*) as the first parameter, then the second parameter *250/portTICK\_RATE\_MS*.

1. Build and debug example 5.

The output produced is same as the output from Example 4.

Example 6. Combining block and non-blocking tasks

Previous examples have examined the behavior of both polling and blocking tasks in isolation. This example aims at demonstrating an execution sequence when two schemes are combined as follows.

1. Two tasks are created in priority 1. These do nothing other than continuously print out a string.

These tasks never make any library function calls that could cause them to enter the Blocked state, so are always in either the Ready or the Running state. Tasks of this nature are called *continuous processing* tasks as they always have work to do.

1. A third task is created at priority 2. It just prints out a string periodically, so uses the *vTaskDelayUntil()* library function to place itself into the Blocked state between each print iteration.

You need to modify the main.c() file as the following steps.

1. Create one continuous processing task function named as *vContinuousProcessingTask()*. It includes a null loop delay inside the infinite *for( ; ; )* loop as the vTask1 (or vTask2) in Example 1.
2. Create another task function for the third periodic task named as *vPeriodicTask()*. Within its infinite *for( ; ; )* loop, call the *vTaskDelayUntil()* library function to create the fixed-time delay as in the *vTaskFunction()* function in Example 5.
3. In the main() function, create two tasks instances of *vContinousProcessingTask()* at priority 1 and one task instance of *vPeriodicTask()* at priority 2.
4. Build and debug the project.

The execution pattern is expected to be as shown before.

