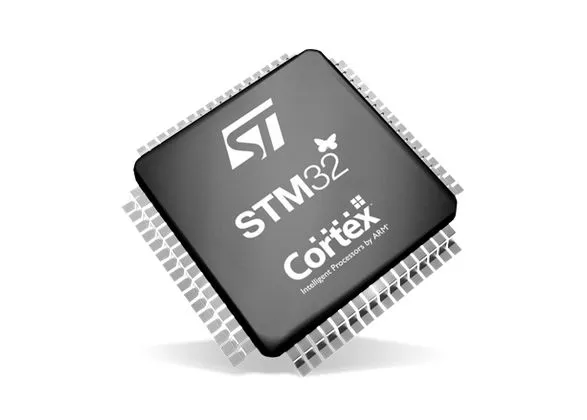
FreeRTOS



Semester 3 Embedded Systems

FreeRTOS assignment

Oct 16, 2022.

Student: Andre Sanao

Course: Technology

# Introduction

The purpose of this assignment is to know more about FreeRTOS in STM32. FreeRTOS is a free and open-source real-time operating system (RTOS) that runs on many popular microcontrollers, including STM32. A general-purpose OS is normally designed with a focus on user experience. For now, the main goal is to learn about threads, mutex and queues.

# Research

In this section is about how FreeRTOS is implemented in STM32. FreeRTOS is extremely useful when trying to run concurrent tasks (also known as thread) within the same program and offering a level of code portability. Threads make sure the codes run simultaneously without interrupting each other. This is done by simulating 3 LEDs in 2 different threads running at the same time.

Then there is Mutex. Mutex prevents several threads from accessing a resource mutually. It ensures that at one time, only one task have access to the resource. For example, when multiple threads use the same LED, the one that has a higher priority or the one that is pressed first will be active and the other thread will wait for it to be finished and then activate the LED again.

To make it more secure that only 1 task is executed at a time, Queue is used. A Queue is created to allow for threads to communicate. For example thread 1 sends a task and put it in the Queue and thread 2 gets the task from the Queue. This way it ensures that only 1 task is executed at a time.

# Testing

In this section will be the testing of threads, mutex and queues that was implemented in the code. For demonstration, scenes containing sprinklers (LED) were used.

## Threads

The following were implemented in threads part:

* 2 threads are created. (Terminal and Scene)
* 3 LEDS called Sprinklers were added to the scene.
* scenes can be toggled simultaneously.
* Scenes are stored in an array of strings.

For the first step of threads, 2 sprinklers per scene was used. The operation of the scene is initialized using enum and stored in an array. Each scene has an array where the Sprinklers turn on and off. A for loop is used to iterate through the operations stored in the array. All 3 scene operation are stored in 1 thread called the execScene. Another thread is made for the input commands called Terminal. These scenes are initiated through entering a character in Putty. Whenever a character is received a new thread is created for that 1 scene.

There were problems and code inefficiency. In my first try I made a thread for each scene but it made the code longer so I found out a website that explains all the functions that FreeRTOS has and learned that you can use osThreadNew() to create new threads.

## Mutex

For the demonstration of the Mutex, the following were implemented:

* Mutex per Sprinkler.
* Acquire its Mutex before switching on the Sprinkler.
* Release its Mutex after switching off the Sprinkler.
* Same for the scene. Acquire and release.

For mutex, I enclosed each Sprinkler and Scenes using osMutexAcquire() and osMutexRelease(). This makes that the resource is unable to make a sprinkler go off simultaneously in different scenes. So for example Scene 1 was executed first and then Scene 2. Both scenes use the same Sprinkler so Scene 1 will acquire the Sprinkler first then after release, then Scene 2 is allowed to acquire the same Sprinkler. Mutex is also implemented also in each Scenes.

For testing, Scenes and Sprinklers can be reused after they are released. The problems that were encountered was when I put the osMutexRelease() before a Sprinkler was turned off which made me look for the bug for hours.

## Queues

For the demonstration of the Queue, the following were implemented:

* 2 Queues are created. (Queues for Scene and for Sprinklers)

For the first step, I created a Queue store Scenes and inside the Scene has another Queue which stores the operations of the Scene. With osMessageQueuePut(), I iterate using for loop and each operation is stored inside the Queue. The operations of the Queue will be received using osMessageQueueGet(). The get functions is used inside another function called Tasks() where the operations are defined. After the Queues is not needed anymore, the character ‘4’ can be pressed to execute the function osMessageQueueDelete(). Queues makes the Threads execute in a safe way to avoid quick redefinition of a Scene.

# Conclusion

In conclusion, this assignment was to learn how to use Threading, Mutex and Queues. It was a bit confusing at first but after getting to know the functions and how they can work together and it was a bit frustrating how the heap memory gets full and needs to be reset. But all in all it was a good assignment to learn about FreeRTOS.

# Bibliography

(no date) CMSIS-RTOS API V2. Available at: https://www.keil.com/pack/doc/cmsis/RTOS2/html/group\_\_CMSIS\_\_RTOS.html (Accessed: October 22, 2022).