LAB #2—Intro to FreeRTOS

November 16, 2022

1 EXERCISE 1

In this exercise, you will design a program that changes the flashing rate of LEDs depending on the push of a button. The program should work as follows:

- The program start blinking every second
- If switch SW_1 is pressed, the blinking rate is multiplied by 2
- If switch SW_2 is pressed, the blinking rate is divided by 2
- The minimum flashing rate is 0.1Hz and the maximum flashing rate is 2Hz.

2 EXERCISE 2

In this exercise, you will use FreeRTOS and control LEDs with 3 tasks. Your program will implement what is commonly known as a running LED light[1] and will work as outline in the flow chart outlined in Figure 1 Your program should contains three tasks as follows:

- You will need to use 8 running LEDs.
- **vTask1**—check if value from queue is legal. If not, a reset signal is needed to be sent. When implementing your solution, you may consider to use **xSemaphoreGive()**[2] to release a signal and **xQueuePeek()**[3] to read item but not pop from a queue.
- **vTask2**—shift the value of LEDs (led_val) and queue it, and to reset both led_val and queue when illegal led_val is detected. During your implementation, you may consider to use the following three functions xQueueSend()[4], xSemaphoreTake()[5], and xQueueReset()[6].
- **vTask3**—It used to retrieve new value from the queue and assign the value to led_val.

---

[1] https://www.youtube.com/watch?v=bqml8xa4b4g
[2] https://www.freertos.org/a00123.html
[3] https://www.freertos.org/xQueuePeek.html
[4] https://www.freertos.org/a00117.html
[5] https://www.freertos.org/a00122.html
[6] https://www.freertos.org/a00018.html#xQueueReset

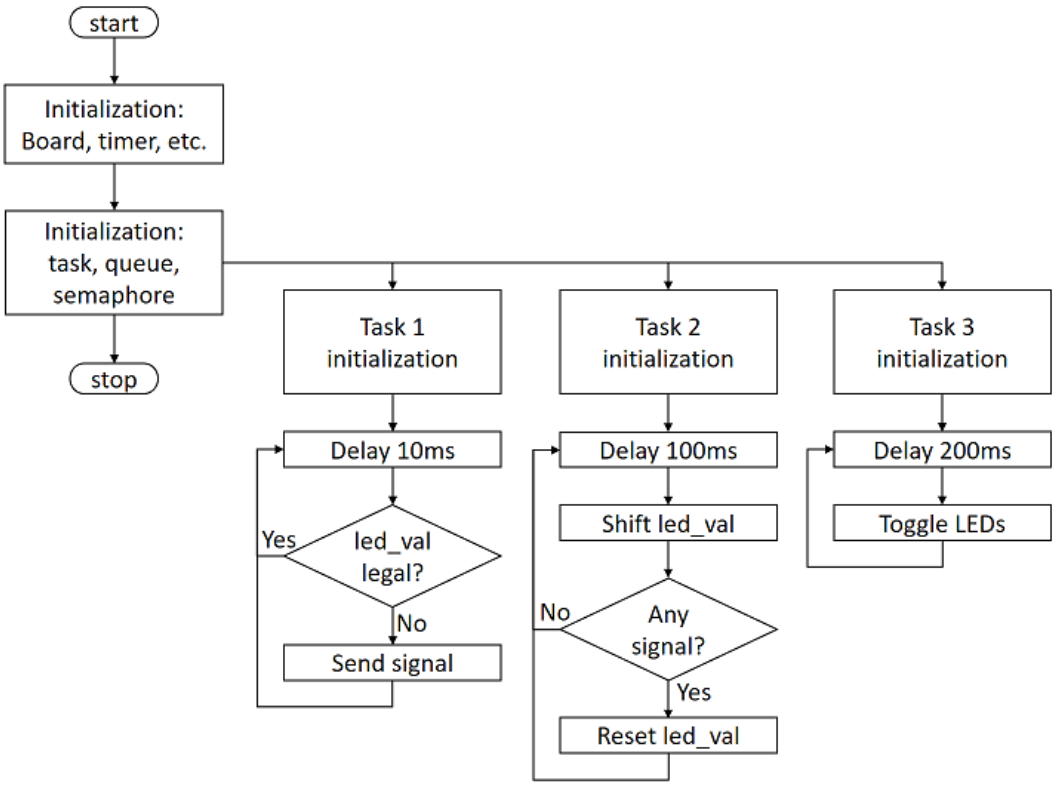

FIGURE 1. Flowchart of a program to control LEDs with 3 tasks

3 EXERCISE 3

3.1 Introduction

Software timers are important parts of any real-time multitasking operating system. The timers are used in tasks to schedule the execution of a function at a time in the future, or periodically with a fixed frequency. Software timers under FreeRTOS do not require any hardware and are not related to hardware timers as they are implemented in software. When a timer expires, the program can be configured to call a function names as the timer's callback function.

Software timers are optional in FreeRTOS and the application programs must be built with the source file *timers.h* included as part of the program. Parameter *configUSE_TIMERS* must be set to 1 in file *FreeRTOSConfig.h* before the software timers can be used.

Two types of software timers are supported by FreeRTOS:

- One-shot timers—These timers are started manually and do not re-start when they complete. The callback function is executed only one when the timer expires.
- Auto-reload timers—These timers re-start each time they expire, thus resulting in

repetitive execution of the callback function attached to the timer.

A software timer can be in one of two states: Dormant, and Running. A Dormat timer exists but it is not active. A Running timer is active and it will call its callback function when its period expires.

All software timer callback functions execute in the context of the same RTOS daemon (or "timer service") task. The daemon task is a standard FreeRTOS task that is created automatically when the scheduler is started. The priority and stack size are set at compile time by the two parameters in file FreeRTOSConfig.h: *configTIMER_TASK_PRIORITY* and *configTIMER_TASK_STACK_DEPTH*. Callback functions must not call to functions that may cause the enter the Blocked state.

A timer must be created before it can be used. Creating a timer does not start it. Timers must be started, stopped, or reset manually by the user programs. Software timer API functions send commands from the calling task to the daemon task on a queue called the "timer command queue". The timer command queue is a standard FreeRTOS queue that is created automatically when the scheduler is started. The length of the timer command queue is set by the configTIMER_QUEUE_LENGTH compile time configuration constant in FreeRTOSConfig.h.

The daemon task is scheduled like any other FreeRTOS task and it will process commands, or execute timer callback functions, when it is the highest priority task that is able to run. Parameter configTIMER_TASK_PRIORITY controls the timer task priority and is recommended to be set to higher than other tasks to allow the timers to work smoothly.

3.2 Description

In this exercise, you will create a reaction timer. The exercise uses an LED and a push-button switch. The user is expected to press the push-button switch as soon as the LED is turned ON. The time between the LED being turned ON and the user pressing the button is measured and printed to the serial port in milliseconds. The LED is turned ON again after a random delay, ready for the next measurement.

3.3 Implementation details

- The program will consist of only one task
- You can generate the random numbers using the *rand()* as shown in Listing 1

```
#include <stdint.h>
#include <stdlib.h>
#include <time.h>
uint8_t get_random_number(uint8_t min, uint8_t max){
  srand(time(0));
  double scaled = (double)random()/RAND_MAX;
```

```
  return (max - min +1)*scaled + min;
}
```

Listing 1: Function to generate a random number between two integers

- The random number generate from the pseudo-code in Listing 1 is used to create random delay in the program so that the user does not know when the LED will lit again.
- As soon as the LED is turned on, the current tick count is stored in variable. The program then waits until the button is pressed by the user, and then calculates the elapsed time by getting the new tick count and subtracting the old tick count from it. This value is the reaction time of the user in milliseconds, which is then displayed on the serial port.

Algorithm 1: Pseudo-code to control the reaction timer device

Input:
- RESET_SWITCH —when the reset switch is pressed, the device turn the LED and clears the serial port
- REACTION_TIMER_SWITCH —when pressed, the user's reaction time is displayed on the serial port

Initialization
- All pins connected to the switches are initialized as inputs
- All pins connected to the LED are initialized as outputs
- Turn off the LED

end

Output: A message on the serial port that shows how many milliseconds a user took to press an the REACTION_TIMER_SWITCH

Loop
- **if** *RESET_SWITCH==PRESSED* **then**
  - Turn OFF the LED
  - Clear the serial port display
- **if** *REACTION_TIMER_SWITCH==PRESSED* **then**
  - Wait 5 seconds
  - Wait random time between 1 and 10 seconds
  - Turn the LED ON
  - Save the current time (in terms of tick counts)
  - Wait until push-button switch is pressed
  - Save the new tick count
  - Calculate the elapsed time
  - Turn LED OFF
  - Display elapsed time in milliseconds on LCD

EndLoop

4 EXERCISE 4

4.1 Introduction

In this exercise, you will use FreeRTOS software timer to measure the number a given event occurred. Event counting is used for example in measuring the speed of car. In this particular case, a magnet and a hall sensors are combined to measure how often a wheel rotates. A Hall sensor detects the presence of a magnet based on its polarity. If a magnet is fixed on a wheel and and place the hall sensor near it in such a way that every time the wheel rotates the hall sensor detects it it is possible to count the number of wheel rotations; thus, the overall speed of the vehicle.

4.1.1 Implementation details

- Your program shall work as shown in Algorithm 2
- Full wheel rotation will be simulated using a square wave generated by a function generator, i.e., when the square wave completes its period, you can assume that the wheel has completed its circle.
- The program will consist of two tasks:
  - The ideal task [7], which is used to print the vehicle's velocity on the serial port
  - The vFrequencyCounter, which compute the number of time the wheel completed a full circle
  - Assume the vehicle has a wheel of 35 cm. For reference, please read this document to learn how to compute the final speed of the vehicle.

Algorithm 2: Pseudo-code to compute the speed of a vehicle

Input: SQUARE_WAVE_SIGNAL —similate the a wheel's full circle that would be detected by a hall sensor. The square wave will be generated by an external signal generator

Output: The speed of the vehicle (in Km/h)

Loop

Wait until a rising signal edge of the square wave is detected
Start timer with period of 1 second
Count the number of rising signal edges
When timer is expired, compute the speed of the of vehicle
Display the speed of the vehicle on the serial port

EndLoop

[7] https://www.freertos.org/RTOS-idle-task.html