This document will answer some of the common question encountered during grading of the 6th Lab.

Common Issue

Issue 1: Code Generation Failure

When using STM32CubeIDE to generate the code for the FreeRTOS threads or timers. The code in the thread/timer callback functions does not get executed.

Cause The CubeMX code generation tools contains bug. (See Figure 1, Discussion Online)

Solution Luckily, ST released a new version of STM32CubeIDE a week ago, you can update it by using Help > Check for Updates.

If you had created the project before updating, You must delete the project and start a new one.

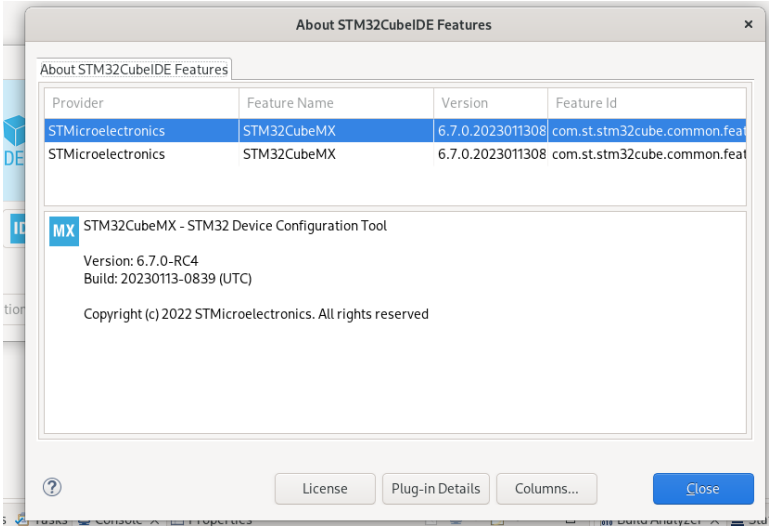

Figure 1: Help > About STM32CubeIDE > MX shows that the buggy < 6.7.x is installed

Issue 2: FreeRTOS Misconfigured

The timer code does not run

Possible Cause 1 You forgot to start the timer.

Solution Use osTimerStart to start the timer. See example.

Possible Cause 2 You forgot to enable timer in FreeRTOS settings

Solution The timer is enabled by default in CMSIS_V2 open .ioc file. Goto Middleware > FREERTOS > Interface then change it to CMSIS_V2. Ensure that Config Parameters > Software Timer Definition > USE_TIMERS is set to Enabled

Issue 3: Unstable Power

The Nucleo board exhibit strange behavior like freezing, serial monitor disconnecting, or otherwise unpredictable things

Possible Cause You forgot to connect the resistor in series to the LED causing brownout.

Solution GPIO can handle maximum output current of 20mA with total of 120mA (see figure 2). You must connect the resistor in series between the LEDs.

Output driving current

The GPIOs (general purpose input/outputs) can sink or source up to ±8 mA, and sink or source up to ±20 mA (with a relaxed V_{OL}/V_{OH}) except PC13, PC14 and PC15 which can sink or source up to ±3mA. When using the PC13 to PC15 GPIOs in output mode, the speed should not exceed 2 MHz with a maximum load of 30 pF.

In the user application, the number of I/O pins which can drive current must be limited to respect the absolute maximum rating specified in *Section 6.2*. In particular:

- The sum of the currents sourced by all the I/Os on $V_{DD,}$ plus the maximum Run consumption of the MCU sourced on $V_{DD,}$ cannot exceed the absolute maximum rating ΣI_{VDD} (see *Table 12*).
- The sum of the currents sunk by all the I/Os on V_{SS} plus the maximum Run consumption of the MCU sunk on V_{SS} cannot exceed the absolute maximum rating ΣI_{VSS} (see *Table 12*).

Figure 2: Current Limit for GPIO

Suplementary Lab Grading Criteria for Lab Item 4.

The Lab item 4 asked you to use Message Queue or Mailbox for inter-thread communication.

While the question may suggested that you can use Mail Queue, the CMSIS RTOS v2 API deprecated the Mail Queue in favor of the new message queue (see link). Therefore, it is *discouraged* that you use any of the following v1 functions: `osMail* osMessage*`

The following sections shows acceptable and unacceptable solutions:

Accepted Solution 1 : Fixed-size Message

// Defining queue with max capacity of 16 message storing message as struct my_message.

```
struct my_message {
	char data[32];
};
myQueue01Handle = osMessageQueueNew (
	16,
	sizeof(struct my_message),
	&myQueue01_attributes
);

...
```

Reason for being acceptable: This use the message queue properly.

Accepted solution 2 : Dynamic size message using Dynamic Allocation (please avoids malloc and use Memory Pool)

Reason for being acceptable: This also use the message queue properly.

Accepted solution 3: (but not recommended): Fixed-size Message with Char Array

```
 //Message Queue with 16 items each MAX_MSG_LEN bytes long.
myQueue01Handle = osMessageQueueNew (16, sizeof(char) * MAX_MSG_LEN,
&myQueue01_attributes);
```

Reason for not being recommended: This will caused the C code have interesting type behavior and be a bit confusing to read. (See stackoverflow)

Rejected Solution: Using Mutex or similar synchronizing mechanism.

```
osMutexAcquire(myMutex01Handle, osWaitForever);
osMessageQueuePut(...);
osMutexRelease(myMutex01Handle);
```

Reason for rejecton: This side-steps the benefit of the message queue. The message queue should allow us to completely eliminate locking mechanism. So using Mutex along-side with the message queue problably means that you are using it wrong.