Introduction

For our fourth assignment, we will continue working on the Nucleo-64. This assignment will focus on the use of the STM32F091RC timing and PWM subsystem and the development of finite state machines.

As before, you will develop a proper GitHub project for this assignment, complete with code, documentation, and README. The assignment is due on Wednesday, October 16 at 9:00 am Mountain time, and should be submitted on GitHub with a "ready-for-grading" tag. For this assignment, this repo should have a README; an STM32CubeIDE project containing multiple .c and .h files; and a drawing of your state machine in a PDF file.

You may consult with other students, the SAs, and the instructor in reviewing concepts and crafting solutions to problems (and may wish to credit them in documentation). Also, you can choose to get your design/code reviewed with a partner after October 13. Add your code reviewer name in the Logistic Spreadsheet which has a column for code reviewer.

Assignment Details

You have been hired to develop a new train monitoring system for an International Airport which has a terminal/baggage claim(T), two concourse stops A and B, and train storage shed(S). An indicator light in the control room dashboard shows the current position of the train and its distance between the stops (more on it later). The indicator light will use an External LED(ELED) with **cathode connected to PA7 and anode connected to 3.3V on Nucleo-64 board**. This will allow us to use the TIMER 3 PWM functionality as given in Dean's chapter 7. **Please make these connections carefully as they will be used to test and grade the assignment.**

For the indicator light, let us define a few terms.

Blink – A blink is defined as the LED(ELED or ULED) on for 100 msec and off for 100 msec.

Pause – A pause is defined as E/U LED off for 400 msec.

Blink sequence – A blink sequence is defined as 3 blinks and a pause. That way, each blink sequence takes 1 second.

Blink sequence loop – This is a continuous loop of consecutive blink sequences. By counting the number of blink sequences, you can gauge the loop timing as each blink sequence takes 1 second.

LED brightness – It is set between 0(LED off) to 255 or 0xFF for maximum LED brightness.

Based on these definitions, the position of the train is indicated by ELED in the following manner.:

- When the train is at terminal **T**, the ELED is performing the blink sequence loop at 0xFF brightness.
- When the train is at concourse **A**, the ELED is performing the blink sequence loop at 0xAF brightness.
- When the train is at concourse **B**, the ELED is performing the blink sequence loop at 0x5F brightness.
- When the train is at shed **S**, the ELED is performing the blink sequence loop at 0x0F brightness.
- In normal operation, the train goes from Terminal T->Concourse A->Concourse B->Shed S->Terminal T in a circular fashion.
- When the train is between these stops, the ELED is not blinking but holds a steady brightness which gradually changes as described below.

To speed your development, a special DEBUG mode is used. While in DEBUG mode, timing is shorter (to save our QA engineers' time), and various diagnostic printouts may be delivered from the device as needed over the UART. Timing for the train stops is given in the following table:

	DEBUG	RELEASE
TERMINAL T	3 sec	5 sec
	4 sec transitions	
CONCOURSE A	2 sec	4 sec
	4 sec transitions	
CONCOURSE B	2 sec	4 sec
	4 sec transitions	
SHED S	2 sec	4 sec
	4 sec transitions back to T	

To gauge the location of train between stops during 4 second transitions, a 500 msec brightness transition scheme is used. To illustrate, when transitioning from Terminal T to Concourse A, the ELED blinks must transition from 0xFF to 0xAF brightness gradually over eight 500 msec periods. Therefore, the first change in brightness of the ELED at the end of 500 msec would be as illustrated in the table:

	Start Value		End Value				Current Value	
	Hex	Dec	Hex	Dec	Pct	Formula	Dec	Hex
ELED Brightness	FF	255	AF	175	12.5%	= (175-255) × 0.125 + 255	245	F5

To give further detail on the example above, the transition from T to A should look as follows:

msec into transition	0	500	1000	1500	2000	2500	3000	3500	4000
ELED Brightness	255	245	235	225	215	205	195	185	175
Hex ELED Brightness	FF	F5	EB	E1	D7	CD	C3	B9	AF

Remember that during transitions, the LED is not blinking but holds a steady brightness that adjusts at every 500 msec mark as given in this table.

At any point in the above cycle, if an emergency button (User Switch on the Nucleo board is pressed but not released:

- The train comes to an abrupt stop if moving and the User LED on the board (ULED) starts blink sequence loop as defined above. The ELED also starts or continues the blink sequence loop at the same brightness level that was before the switch pressed. Therefore, if the ELED was going through the blink sequence loop when the train was at the stop, it will continue to do so. If the ELED was holding a steady brightness during transition when the User Switch was pressed, it will stay in that brightness but starts the blink sequence loop. In addition, the ULED starts the continuous blink sequence loop.
- The train thereafter stays in this emergency mode till the emergency is cleared when the User Switch is released.
- When the emergency is cleared, the train resumes the state before emergency and completes that state with whatever time was remaining for that state along with the normal ELED behavior for that state. For example, if the emergency button was pressed when the train was at Concourse A with 1 second remaining in Debug mode, and the emergency is cleared, the ELED

will continue the blink sequence loop for concourse A and spend the remaining 1 second, before starting the move to Concourse B.

- Similarly, when the train is between stops (i.e. in transition), and the emergency button is pressed(but not released), the ELED starts the blink sequence loop with the current transition color and ULED starts the continuous blink sequence loop. Once the emergency is cleared by releasing the User Switch, the ELED goes back to being steady color as it was during the transition and finishes the remaining time(1 second in this example) completing the transition. During this remaining transition, the brightness of the ELED adjusts on the 500 msec marks as given in the table above.

The emergency set and clear actions must start within 100 msec of a button press occurring.

When in DEBUG mode, your code should at minimum print the following to the diagnostic output:

- Main loop is starting.
- All state transitions, with the system time of the event (msec since startup), and the names of the state being transitioned from and to.
- It is expected that the timings may get a little off in DEBUG mode due to all printing. Use the systick timers to trigger the signals at the given time markers in this document. Assignment timing will be graded in RELEASE mode with -O0 level of optimization.

State Machine

At the core of this assignment is a finite state machine. You will need to think carefully about how to construct yours—it is not completely spelled out for you above, although all the information you need is present. You should draw your FSM[1] and include it as a PDF in your repo, titled for instance “State machine.pdf”.

You may write your state machine using either a switch/case approach or a table-driven approach.

Implementation Notes

As in our DancingLights assignment, this project is intended to be a bare metal implementation, so you may not use advanced SDK or RTOS routines.

You should use the STM’s timing circuitry for all timing. This will involve configuring the SysTick device and setting the SysTick_Handler interrupt. Details on this are given in Dean ch7 and sec B3.3 of the “ARM®v6-M Architecture Reference Manual”. You will need to `#include "core_cm0plus.h"` to

[1] You may use whatever tool you like to draw your FSMs. A possibly useful option might be http://madebyevan.com/fsm/ or draw.io

access the SysTick device. I recommend implementing a timing subsystem using something similar to the following API[2]:

```
typedef uint32_t ticktime_t;  // time since boot, in sixteenths of a second

void init_systick();   // initialize the timing system

ticktime_t now(); // returns time since startup, in sixteenths of a second

void reset_timer(); // resets timer to 0; doesn't affect now() values

ticktime_t get_timer();  // returns ticks since the last call to reset_timer()
```

You should use the STM's PWM functionality to control the LED colors. Details are given in Dean chapter 7, and in chapter 31 of the STM Reference Manual.

For serial output in DEBUG mode, you should use the UART output and display the results in a terminal window (which can be done either within or external to STM32CubeIDE).

Your code should follow the ESE C Style Guide (posted on Canvas) as closely as possible.

As with all code you ever write, I recommend that you simplify the program to its most basic possible behavior, and then slowly accrete additional functionality.

Grading

Points will be awarded as follows:

Points	Item
20	Overall elegance: Is your code professional, well documented, easy to follow, efficient, and refined?
10	Accuracy of your state machine design: Does it consider all the appropriate states and events?
20	Main sequence (T->A->B->S->T): Does this sequence meet all the requirements outlined above? Do the four-second transitions happen correctly?
15	Emergency press: Does this functionality meet all the requirements outlined above?
15	Emergency clear: Does this functionality meeting all the requirements outlined above?

[2] I chose to track time in my implementation using "ticktime" units, with a resolution of 62.5 msec (1/16th of a second) per ticktime tick. You may use this resolution, or something else such as 500 msec.

10	Logging: Is your logging clear and appropriate? Are all messages printed as needed?
10	DEBUG vs RELEASE: Does your code have the required functionality in both states?

Good luck and happy coding!