Introduction

For our fourth assignment, we will continue working on the FRDM-KL25Z. This assignment will focus on the use of the KL25Z's timing and PWM subsystem and the development of finite state machines. We will also experiment with peer code review to see if that can improve the quality of our code.

As before, you will develop a proper GitHub project for this assignment, complete with code, documentation, and README. Details on submissions are a bit different than we've done to date, and are given below.

Assignment Details

You have been hired to develop a new traffic light for export to the little-known tropical island of Buffahiti. Due to a genetic mutation, the residents of this island exhibit a certain color-blindness, and as a result their traffic lights have the following unusual behavior:

- When the traffic light is in the **STOP** state, Buffahiti residents use the color #611E3C[1].
- When the traffic light is in the **GO** state, Buffahiti residents use the color #229622
- When the traffic light is in the **WARNING** state, Buffahiti residents use the color #FFB200
- When someone presses the button to cross the street (**CROSSWALK** state), the traffic light is blinking with the color #001030.

In the CROSSWALK state, the light blinks 250 msec off, 750 msec on. No other mode has the light blinking.

Except for the blinking in CROSSWALK mode, all other changes in traffic light color must transition gradually and linearly over a one-second period, or otherwise the residents of Buffahiti complain of headaches. To illustrate, when transitioning from STOP to GO, the color is initially #611E3C. 125 msec into the one-second transition, the color is computed as follows:

	Start Value		End Value				Current Value	
	Hex	Dec	Hex	Dec	Pct	Formula	Dec	Hex
R	61	97	22	34	12.5%	= (34 - 97) × 0.125 + 97	89	59
G	1E	30	96	150	12.5%	= (150 - 30) × 0.125 + 30	45	2D
B	3C	60	22	34	12.5%	= (34 - 60) × 0.125 + 60	57	39

[1] Colors in this document follow the 24-bit hex triplet format common in "web colors", as described here. For example, the color #611E3C represents red=0x61=decimal 97; green=0x1E=decimal 30; and blue=0x3c=decimal 60. Each color may vary within the range 0-255 decimal.

To give further detail on the example above, the transition from STOP to GO should look as follows. (Note this table is abbreviated and does not show all intermediate values):

msec into transition	0	63	125	188	250	…	750	…	1000
Color	#611E3C	#5D263A	#592D39	#553537	#513C36	…	#327829	…	#229622

In order to speed development, a special DEBUG mode is used. While in DEBUG mode, light 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 traffic light is given in the following table:

	DEBUG	PRODUCTION
STOP	5 sec	20 sec
	One sec transition to GO state	
GO	5 sec	20 sec
	One sec transition to WARNING state	
WARNING	3 sec	5 sec
	One sec transition to STOP state	

At any point in the above cycle, if a resident presses the button to cross the street (represented on our FRDM-KL25Z by a touch anywhere on the capacitive slider), the traffic light should do the following:

- Transition, over a one-second period, from whatever the current color is to the CROSSWALK color (#001030). This one-second transition should follow the approach given above.
- Stay in the CROSSWALK mode for 10 seconds, blinking 250 msec off, 750 msec on. (This duration is the same in both DEBUG and PRODUCTION modes.)
- Transition, over a one-second period, from the CROSSWALK state to the GO state.

Button presses are ignored if they arrive when the light is either transitioning into or is already in the CROSSWALK state. Otherwise, the transition into the CROSSWALK state 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
- Button press detected, with the system time (msec since startup)

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[2] and include it as a PDF in your repo, titled for instance "State machine.pdf".

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

Implementation Notes

As in our Blinkenlights assignment, this project is intended to be a bare metal implementation, so you may not use advanced SDK or RTOS routines. You may use include files generated from MCUXpresso tools such as pin_mux.h. You may also use "board.h" and "MKL25Z4.h". You may not use the "fsl_" include files that provide higher-level SDK-based functions for LED or slider control, with the exception that you may use "fsl_debug_console.h".

You should use the KL25Z'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 access the SysTick device. I recommend implementing a timing subsystem using something similar to the following API:

```
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 KL25Z's PWM functionality to control the LED colors. Details are given in Dean chapter 7, and in chapter 31 of the KL25Z Reference Manual. You will probably also need to reference the KL25Z data sheet to understand how PWM outputs map to the three R, G, B colors.

You may poll the capacitive slider.

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 MCUXpresso).

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

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

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.

Peer Code Review, and Submission Schedule

Your primary submission is due on Monday, October 12, at 9 am. At that time, you should submit the following:

- A private GitHub repo URL which will be accessed by SAs to review your code and documentation. For this assignment, this repo should have an MCUXpresso project containing multiple .c and .h files. Your repo should also have a drawing of your state machine in a PDF file.

  New for this assignment: You must tag your repo at this time with the tag "submitted-to-code-review"

- A PDF containing all C code. As before, the PDF is used specifically for plagiarism checks.

We will randomly assign you to a partner in the class, and we will announce assignments prior to 9 am on October 12. Between 9 am on Monday and the start of class on Tuesday, you will need to spend time to familiarize yourself with your partner's code. Class time on Tuesday will be spent performing peer code reviews; during this time, you will each be given about 20 minutes to review the other's code. Your participation grade for this class (5 participation points) will be based on how prepared, insightful, and tactful you are in critiquing your partner's code.

To be clear, your code should meet all the project requirements, including extra credit if you choose to do it, by the Monday 9 am deadline. The goal of the code review is to help you find stray bugs and improve the elegance and readability of your code.

After the code review, you should plan to make additional check-ins to your GitHub repo to address the feedback received during code review. These modifications are due by 9 am on Thursday, October 14.

Note that the peer review process adds a hard deadline to the initial due date of Monday at 9 am. If your code is not functional at this point, you are preventing your peer from reviewing your code, which will hurt their grade as well as yours. It is therefore especially important that your code is completed on time.

Grading

Points will be awarded as follows:

Points	Item	When
30	Overall elegance: Is your code professional, well documented, easy to follow, efficient, and elegant? (Would James Bond be proud, or is this a Bruce Willis dirty t-shirt job?)	Assessed based on your final (post-code-review) state
10	Accuracy of your state machine design: Does it consider all the appropriate states and events?	Based on Monday check-in
20	Main sequence (STOP→GO→WARNING): Does this sequence meet all the requirements outlined above? Do the one-second transitions happen correctly, or do the Buffahiti residents complain of headaches?	Based on Monday check-in
20	Crosswalk sequence: Does this functionality meet all the requirements outlined above?	Based on Monday check-in
10	Logging: Is your logging clear and appropriate? Are all messages printed as needed?	Based on final (post-code-review) state
10	DEBUG vs RUN: Does your code have the required functionality in both states?	Based on final (post-code-review) state
10	Extra credit as described below	Based on Monday check-in

As mentioned above, in addition to the assignment grading, you will earn participation points on October 13 based on your preparation, insight, and tactfulness in code review.

Good luck and happy coding!

Extra Credit

If you have the ability to connect a momentary switch to your FRDM-KL25Z board, you may choose to do so. The best way to do this is to acquire the following headers and fully populate your board:

- 1x ‘Arduino’ header 2×10, 2.54 mm: SAMTEC – SSW-110-01-G-D or similar
- 2x ‘Arduino’ header 2×8, 2.54 mm: SAMTEC – SSW-108-02-G-D or similar
- 1x ‘Arduino’ header 2×6, 2.54 mm: SAMTEC – SSW-106-01-G-D or similar

(I ordered mine from Mouser.) However, there are many headers that will work, and of course for this portion of the assignment, you only need access to two pins. You could even solder jumper wires directly onto those two pins, if you have access to a soldering iron but cannot get headers.

Once you have gotten your hardware set up, configure the appropriate GPIO pin to be an input, setting the pull-up resistor functionality if necessary, and set an interrupt on (do not poll) the switch. Use this interrupt rather than the touch slider to drive the CROSSWALK behavior in the assignment.