CECS 346 Fall 2019 Project 3

Final Project – Drag Race + Car

General Comments:

- Demonstration will be due by the end of scheduled class time during finals week: Dec 13 at 12:15 pm. **Please come to class ready to demo!**
- Project report will be due at end of day of class during finals week: Dec 13 at 11:55 pm.
- It is allowable to demo before scheduled class time during finals week. Contact instructor or assistant to schedule a time.
- You must use a state machine to implement your car logic.
- You are able to get partial credit by demoing less than all 4 parts of the demo.
- You must demonstrate all functionality you will turn in within a 15 minute window. In other words, you must be able to demonstrate the car portion and the drag race starting line portion with maximum of <15 mins setup time in between. (The intent was that you will have 2 LaunchPads per pair: 1 for the drag race starting line and 1 for the car, but that is not strictly required, so some setup time is allowed.)
- If working with a partner, both team members must be present to demo.
- This final project will be worth double the amount of the previous projects. In other words, it will be worth half of your grade in the Projects category.

Preparation:

You will need:

Drag race starting line (same as Project 2):

- TM4C LaunchPad,
- two push buttons (not switches! ask the instructor for buttons if needed),
- two ~10kΩ resistors,
- eight color LEDs (2 red, 4 yellow, and 2 green),
- eight resistors for the LEDs (between 330Ω to 1kΩ each).

Car:

- TM4C LaunchPad,
- Car with two stepper motor driven wheels (with stepper motor drivers)
- Battery pack (not USB),
- Power supply circuit (voltage regulator with filter capacitors),
- IR avoidance sensor.

Book Reading: 4.3, 4.4, 6.5, 8.8

Starter Project: Lab 7, or pseudocode at end of this document

Purpose:

Project 3 requires you to build circuits on the breadboard and connect them to the LaunchPad. The purpose of this project is to combine most of the concepts learned throughout the semester, including:

- interfacing to switches and LEDs,
- interfacing an IR avoidance sensor,
- interfacing to stepper motors using wave driving or full
- using a Moore finite state machine (FSM),
- using interrupts (GPIO and SysTick).

You will perform explicit measurements on the circuits to verify they are operational and to improve your understanding of how they work.

System Requirements:

1. We will build a drag race starting line system based on Project 2, with one of the lane buttons being replaced by an IR avoidance sensor. Consider Figure 2. There are two lanes (Left and Right), with a "Christmas tree" showing LEDs for both lanes. **The system requirements for the "Christmas tree" are the same as Project 2, so see the Project 2 description for more details if necessary.**

2. We will also build a car that will stage itself (drive to starting line), race (drive straight down the drag strip), and stop before hitting any obstacles. The car will have 2 buttons to choose which program to run, either stage or race. See Figure 3.

   The car will have at least 4 states: Drive Forward/Backward (Phases 1,2,3,4). For stepper motor interfacing, you can use either wave driving or full stepping (full stepping recommended).

   There are 3 inputs to your car LaunchPad:

   - Stage button – Simulate lining up to race starting line.
   - Race button – Car drives forward specified distance.
   - IR obstacle avoidance sensor – mounted to vehicle facing forward. Causes vehicle to stop if obstacle is detected.

   There are 8 outputs on your car LaunchPad:

   - Stepper motor driver left – A,B,C,D
   - Stepper motor driver right – A,B,C,D

   The vehicle will be untethered. It will be powered by a battery pack and voltage regulator circuit.

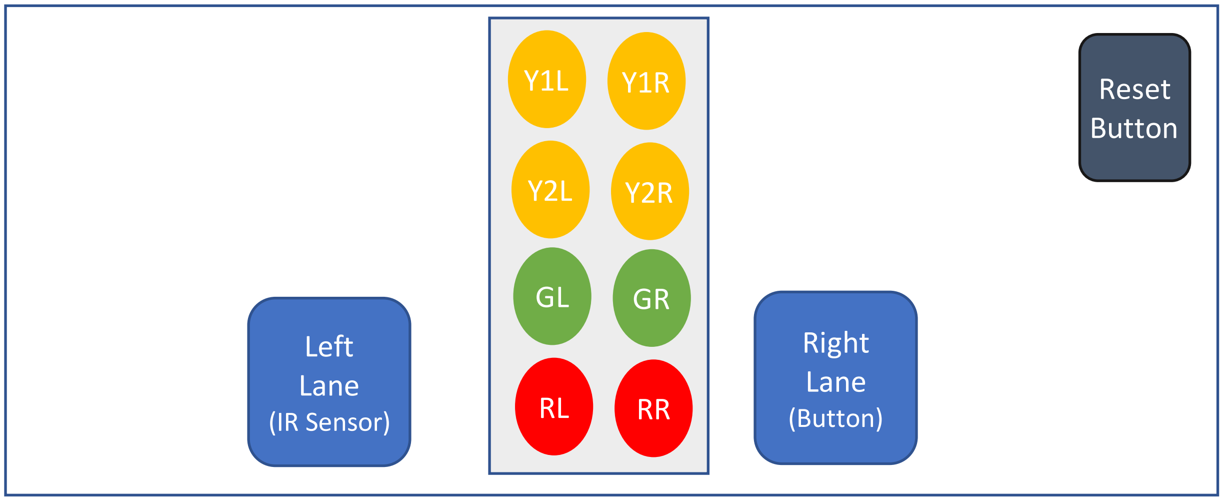

Figure 1 System diagram showing 'Christmas tree' and buttons. Note: Either the Left or Right lane buttons can be replaced by the IR sensor.

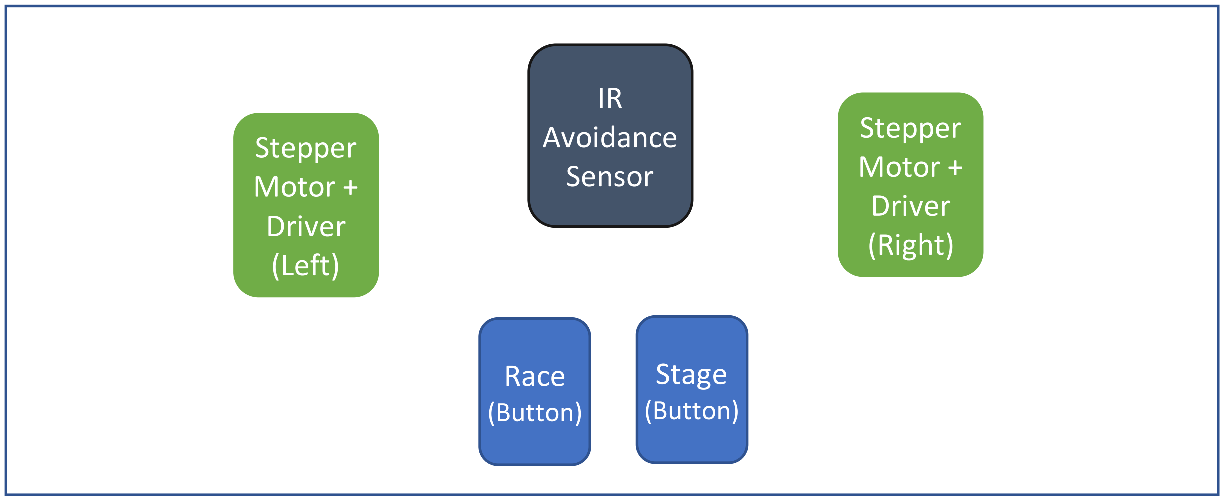

Figure 3 System diagram 2 showing car inputs and outputs.

Implementation Requirements:

In this project, you **must** build:

1. Drag race starting line / "Christmas tree" - **The intent is to reuse Project 2, modifying the implementation as minimally as possible with the below specific requirements:**
    - A two button **on breadboard** interface **using negative logic** for the left lane and right lanes. Use GPIO interrupts with priority 2 to detect both edges for both sensors.

    - **Replace one lane button with an IR avoidance sensor calibrated to detect a car within 2 to 3 inches - no code changes should be required compared to negative logic button!**
- An eight LED interface that implements either positive or negative logic (or combination of both) using on breadboard (not on LaunchPad) LEDs.
    - **This interface from Project 2 should be useable without changes for Project 3.**
- A one button interface on a separate port than the left and right lane sensor buttons to represent the reset button. Use GPIO interrupts with priority 1 to detect level sensitive (when pressed) for reset button.
    - **This interface from Project 2 should be useable without changes for Project 3.**

2. Car – **Based on Labs 6 and 7:**
    - A two button interface using either positive or negative logic (or combination of both) implemented either on board buttons or on breadboard buttons. These buttons will be used to select the 'mode' of the car: stage or race.
    - IR obstacle avoidance sensor mounted to vehicle facing forward, calibrated to detect objects 2-3 inches away from the front of the car.
    - In **race**, the car will drive forward 36-38 inches then stop within a maximum of 120 seconds. (These requirements may change as we get closer to the due date.)
    - In **stage**, the car will drive backwards 1-3 inches then stop within a maximum of 30 seconds. (These requirements may change as we get closer to the due date.)
    - If at any point an obstacle is detected in front of the vehicle (similar to our Reset button logic from Project 2), the vehicle immediately stops moving.
    - **Use a single port (eg Port B) to provide all 8 outputs to the stepper motor drivers.**

Implement a Moore finite state machine (FSM). There should be a 1-1 mapping between the FSM graph and data structure. For a Moore FSM, this means each state in the graph has a name, an output, a time to wait, and next state links. The state data structure has exactly these components: a name, an output, a time to wait, and 3 next state pointers (3 inputs, so $2^3 = 8$ combinations of inputs). There is no more and no less information in the data structure than the information in the state graph. **There should be only a single output variable; use bit masking and bit shifting if you need to separate values.**

For the state machine, assume the 3 inputs are:

- **Move** – if 0, no state transitions will happen. If Move is 1 and Forward or Reverse are 1 (but not both), then the car moves in the expected direction. Move is set to 1 if the requested number of steps is not reached, or 0 if it is reached.

- **Forward** – If Forward is 1 and Move is 1 and Reverse is 0, then car moves in the forward direction. Otherwise the car does not move.

- **Reverse** – If Reverse is 1 and Move is 1 and Forward is 0, then car moves in the reverse direction. Otherwise the car does not move.

(The above is similar to what I wrote on the board in lecture on 11/22)

You must use a SysTick interrupt with Priority 3 to implement state delays.

The state graph should define exactly what the system does in a clear and unambiguous fashion. In other words, do not embed functionality (e.g., flash 3 times) into the software that is not explicitly defined in the state graph.

Use good names and labels (easy to understand and easy to change). Examples of bad state names are **S0** and **S1**.

Procedure:

1. Decide which port and pins you will use for the inputs and outputs. Avoid the pins with hardware already connected.
2. Define a bit-specific addresses for inputs and outputs, one per direction (input/output) per port used.
3. Design a Moore finite state machine that implements the system described. Draw a graphical diagram of your FSM showing the various states, inputs, outputs, wait times and transitions, and a state table for the FSM.
4. Write and debug the C code that implements your system.
    a. Your GPIO interrupt handlers should update global variable(s) based on the GPIO.
    b. Your SysTick handler should:
        i. Determine the new current state based on the global variables set by the GPIO interrupt handlers and the current state,
        ii. Set system outputs (eg LEDs, stepper motor driver phases) based on the new current state,
        iii. Configure the SysTick delay to be the delay needed for the new current state.
5. Implement your system on the LaunchPad board. Do not place or remove wires on the protoboard while the power is on.

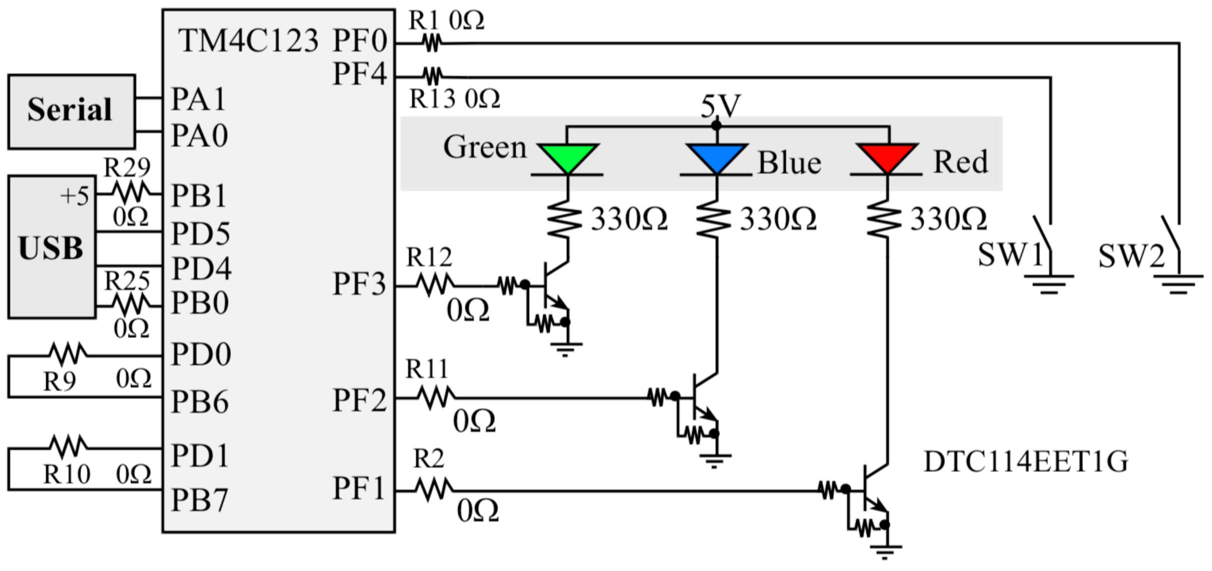

Figure 4 Switch and LED interfaces on the Stellaris® LaunchPad Evaluation Board.

Deliverable:

1) Demonstrate your lab on board. **Both team members must be present to demo!**
   a. Drag race starting line / "Christmas tree" - Demonstrate that the lane IR sensor and lane button trigger the LEDs as expected. Demonstrate that the reset button works as expected from any state.
   b. Demonstrate that pressing the "race" button on the car causes the car to travel forward the specified distance within the specified amount of time.
   c. Demonstrate that pressing the "stage" button on the car causes the car to travel backward the specified distance within the specified amount of time.
   d. Demonstrate the car stops when an obstacle is detected without hitting the obstacle. (This is easiest to demonstrate in combination with (b) or (c) above.)
2) Submit a project report (eg Word Document) to the Beachboard Dropbox containing:
   a. Class name, lab number and name, your name and your partner's name (if applicable)
   b. Table showing list of GPIO ports + pins used (similar in format to "Recommended GPIO Pins" in Project 1 description)
   c. Description of how each bit in the input maps to which GPIO ports + pins.
   d. Description of how each bit in the output maps to which GPIO ports + pins.
   e. State table for your Moore FSM (for the car only)
   f. State diagram for your Moore FSM (for the car only)
   g. Labeled photograph of your hardware system with all LEDs and buttons labeled.
   h. Diagram of your hardware system (see Figure 4 for an example)
   i. Software source code: The .c file or files.

Code Setup:

Lab 7 code is a good starting point. If you need help setting up the state machine, see below pseudocode as a starting point.

```
#include <stdint.h> // C99 data types
#include "tm4c123gh6pm.h"

// Function Prototypes (from startup.s)
void DisableInterrupts(void); // Disable interrupts
void EnableInterrupts(void);  // Enable interrupts
void WaitForInterrupt(void);  // Go to low power mode while waiting
for the next interrupt

// function declarations
...
void GPIOPortF_Handler(void);
void SysTick_Handler(void);

#define BUTTONS (*((volatile uint32_t *) ...))
#define STEPPERS (*((volatile uint32_t *) ...))

// global variables
uint32_t currentState;
uint32_t stepsRemaining;
uint32_t Input;

struct State {
     uint32_t Out;
     uint32_t Delay;
     const uint32_t Next[8];
};

typedef const struct State STyp;

#define S0 0
#define S1 1
#define S3 2
#define S4 3

STyp FSM[4] = {
     {...,...,{S0, S0, S0, S0, S0, S4, S1, S0}}, // S0
     {...,...,{S1, S1, S1, S1, S1, S0, S3, S1}}, // S1
     {...,...,{S3, S3, S3, S3, S3, S1, S4, S3}}, // S3
     {...,...,{S4, S4, S4, S4, S4, S3, S0, S4}}  // S4
};

int main(void) {
     // Initialize Inputs, Outputs, SysTick, and Interrupts
     ...

  Input = 0;
     currentState = S0;

  while(1){
          WaitForInterrupt();
  }
}

// Handle GPIO Port F interrupts. When Port F interrupt triggers, do
what's necessary and update Input bits 0 and 1
void GPIOPortF_Handler(void) {
     GPIO_PORTF_ICR_R = 0X11;            // clear interrupt

     if (PF0 pressed) {
          stepsRemaining = 16; // 180 degree turn
          Input &= ~0x01;
          Input |= 0x02;
     } else if (PF4 pressed) {
          stepsRemaining = 8;  // 90 degree turn
          Input &= ~0x02;
          Input |= 0x01;
     } else {
          stepsRemaining = 0;
          Input &= ~0x03;
     }
}

// Handle SysTick generated interrupts. When timer interrupt triggers,
do what's necessary, update state and outputs
void SysTick_Handler() {
     if (stepsRemaining > 0) {
          stepsRemaining--;
     }

     // Update Input bit 3
     if (stepsRemaining > 0) {
          Input |= 0x04;
     } else {
          Input &= ~0x04;
     }

     currentState = FSM[currentState].Next[Input];
     STEPPERS = FSM[currentState].Out; // set stpper motor output

     // Reload SysTick timer with new value
     ...
}
```