

Lab 3

Lab Objective: To apply knowledge learned in lab 2 to real-world application. This project is designed to challenge students to implement a state machine using buttons and LEDs. The system must be able to automatically cycle through the traffic light sequence and allow for manual intervention if necessary. Students must demonstrate their understanding of state machines, as well as their ability to design and implement a system using input and output devices.

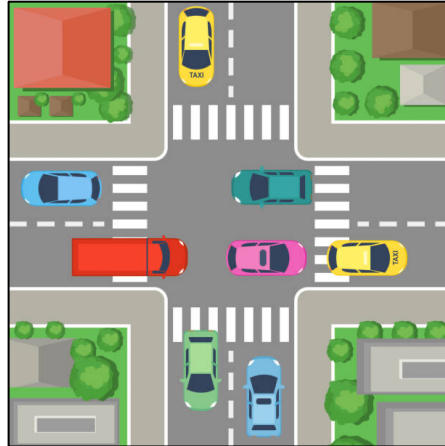
Title: Design a Traffic Light Control System

Problem: You have been assigned to California's Division of Traffic Operations. A traffic intersection has a problem with traffic congestion during peak hours. The traffic lights are poorly timed, causing long wait times and traffic jams. Your task is to implement a traffic light control system that will reduce the wait time for drivers and improve traffic flow. The traffic light system consists of two intersections, North-South and East-West.

The traffic light control system will operate in the following way:

- Both intersections will initially be red and the pedestrian light is red.
- After 10 seconds, the North-South light turns green, and the East-West light is red.
- After 20 seconds, the North-South light turns yellow for 3 seconds before turning red.
- After 5 seconds of red light at the North-South light, the East-West light turns green, and the North-South light remains red.
- After 20 seconds, the East-West light turns yellow for 3 seconds before turning red.
- After 5 seconds of red light, the system returns to the initial state of both lights being red.

The system should be designed to allow for manual intervention if necessary. A push button at each intersection can be used to trigger a pedestrian crossing mode. When the button is pressed, the current green light should change to flashing yellow for 10 seconds, then switch to red to allow pedestrians to cross for 15 secs. At this state, the other light should continue to remain red, and the pedestrian light turns green. Once the pedestrian crossing mode is complete, the system should return to the previous state.



State Machine: The state machine for this project will have six states:

- Both intersections red
- North-South green, East-West red
- North-South yellow
- East-West green, North-South red
- East-West yellow
- Pedestrian crossing mode

The state machine will transition between states based on the elapsed time and button input. The push buttons will trigger the system to enter pedestrian crossing mode and interrupt the current state. Once the pedestrian crossing mode is complete, the system should return to the previous state.

Required Equipment and Parts:

- Breadboard
- 4 Pushbuttons (Pedestrian crossing at each intersection)
- 8 LEDs (2 Red + 2 Green + 2 Yellow for Traffic Lights and 1 Red + 1 Green for Pedestrians)
- 12 Resistors ($300\Omega < R < 1000\Omega$)
- Jumper wires
- Rev1B Board

Pin Selection:

Students must carefully pick pins with which to work. Below is a E310 Pinout table that specifies which GPIO offset corresponds to which pin number, also IO functions and LEDs are specified. Students must pick a pin that does not have any IOF₀ or LED associated with it. Good choices would be Pin 2, 7, 8, 9, 17, 18, and 19.

HiFive1 Pin Number	GPIO Offset	IOF0	IOF1	LED
0	16	UART0:RX		
1	17	UART0:TX		
2	18			
3	19		PWM1_1	GREEN
4	20		PWM1_0	
5	21		PWM1_2	BLUE
6	22		PWM1_3	RED
7	23			
8	0		PWM0_0	
9	1		PWM0_1	
10	2	SPI1:SS0	PWM0_2	
11	3	PI1:SD0/MOSI	PWM0_3	
12	4	SPI1:SD1/MISO		
13	5	SPI1:SCK		
14		<i>Not Connected</i>		
15	9	SPI1:SS2		
16	10	SPI1:SS3	PWM2_0	
17	11		PWM2_1	
18	12		PWM2_2	
19	13		PWM2_3	

E310 Pinout
(from "HiFive1 getting started v1.0.2")

Bitfield (GPIO Offset)

Bitfield or GPIO Offset is a bit number in 32-bit sequence that corresponds to a pin. For example, student wants to enable input for pin 2. First, the student needs to figure out the GPIO offset of pin 2 which is 18 in this case. The binary 32-bit number with 18th offset bit enabled is below. 0000 0000 0000 0100 0000 0000 0000 0000 which equals to 0x00040000 in hexadecimal(0x4000).

The hexadecimal number is called a "mask" for pin 2. Now when mask is acquired it can be stored to corresponding memory location which activates the input. In E310's case its base GPIO address of 0x10012000 plus *input_en* offset of 0x04 which gives memory location of 0x10012004. The mask for pin 2 can be stored into *input_en* offset by the following assembly commands:

```

Li t0, 0x10012000    -- load GPIO base address into t0
Li t1, 0x04          -- load input_en offset to t1
Li t2, 0x4000        -- load pin 2 mask to t2
Sw t2, t1(t0)        -- store t2 into the address provided by t0 plus the offset of t1

```

Offset	Name	Description
0x00	input_val	Pin value
0x04	input_en	Pin input enable*
0x08	output_en	Pin output enable*
0x0C	output_val	Output value
0x10	pue	Internal pull-up enable*
0x14	ds	Pin drive strength
0x18	rise_ie	Rise interrupt enable
0x1C	rise_ip	Rise interrupt pending
0x20	fall_ie	Fall interrupt enable
0x24	fall_ip	Fall interrupt pending
0x28	high_ie	High interrupt enable
0x2C	high_ip	High interrupt pending
0x30	low_ie	Low interrupt enable
0x34	low_ip	Low interrupt pending
0x40	out_xor	Output XOR (invert)

E310 GPIO Memory Map
(from SiFive FE310-G002 Manual)

Getting input from pin 2 is a similar procedure. First, you load the value stored at *input_val* offset into a register. To determine if pin 2 is in the high state student must AND the mask and newly stored *input_val*, when the result of the AND operation is 0x0 the pin is low, otherwise it's high.

Often it is desirable to enable input or output to many pins at the same time. In that case, simple additions of all the masks can be done, since no 1s overlap the result will be a bitmap that covers all the desired pins.

Procedure:

1. Use the same file setup as Lab 2.
2. Select the GPIO's to be used for the inputs and outputs. Use recommended pins. Modify *setupGPIO.S* to enable selected pins for input or output.
3. On the solderless breadboard, wire the outputs to the anodes of the red, yellow, and green LEDs. Tie the cathodes of the LED to the ground through resistors.
4. Modify code of *checkBot.S* to accept mask for a pin (ex. int checkBot(int PIN)). The parameter will be passed onto the register a0. Before proceeding any further, test

this new functionality.

5. In *setLED.S*, change the first line after `ledOn:` to **or t1, t1, a0**.
6. Add code to *main.c* to handle the states of the state machine. When returning a value from `checkBot(int PIN)` do not forget to invert it, since the pull-up function is on, pins are in an active low state which is not intuitive.
7. Test and debug the code.

Submission Worksheet:

1. Make a state diagram showing the signals or time needed to move from one state to another.
2. Submit the complete code package with comments explaining which input or output pin corresponds to what item of the problem statement (Ex: `PIN_2` is for the red light for the North-South intersection).
3. Make a demonstration video introducing the team members, showing and explaining your circuit, and demonstrating the functionality at each state.
4. Report any corner cases that are not covered in the above problem statement (Bonus Points)