Lab 3 Traffic Light Controller

This laboratory assignment accompanies the book, Embedded Microcomputer Systems: Real Time Interfacing, by Jonathan W. Valvano, published by Brooks-Cole, copyright © 2000.

Goals
- The parallel I/O functions of the 6812,
- The usage of linked list data structures,
- The design of a traffic light controller.

Review
- Chapters 4, 5, 10, 13 of the M68HC812A4 Technical Summary, in particular look up PUCR, RDRIV, PORTH, DDRH, PORTJ, DDRJ, PORTT, DDRT,
- Valvano Section 1.6 about open collector logic,
- Valvano Section 1.7 about initializing and accessing I/O ports,
- Valvano Section 2.4 about abstraction, linked lists and FSM's,
- Valvano Section 3.4.2 about accurate time delays.

Starter files
- FSM12.C, MOORE12.*, MEALY12.*, MOORE2.C

Background

In this lab, the input-output parallel ports of the 6812 will be used in designing a traffic light controller. You can use any of the 6812 parallel input/output ports on the H1 connector to interface the IC to the computer, but we suggest ports H, J or T because each bit has a direction register. Finally, this lab is a good example of where a linked list data structure can be used to simplify programming. A linked list solution may not run the fastest, or occupy the fewest memory bytes, but it is a structured technique that is easy to understand, easy to implement, easy to debug, and easy to upgrade. *You are free to implement this lab with any software programming technique you wish, but you must consider a linked list implementation.*

Consider a typical 4-corner intersection as shown in Figure 3.1. The streets are labeled North/South and East/West. There are three inputs. Two are car sensors, and one is a pedestrian walk button. Once again, you are free to modify the number of inputs and outputs, as long as your system is at least as complex as this one.

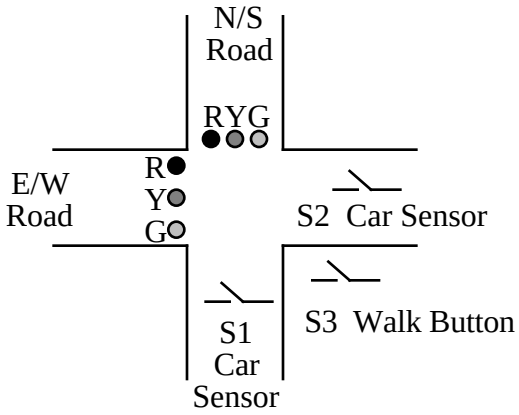

Figure 3.1. Traffic Light Intersection.

There are at least three inputs (you can add more if you like):

Switch 1	ON	means a car is waiting on the North/South road.
	OFF	means no car is on the N/S road
Switch 2	ON	means a car is on the East/West road.
	OFF	means no car is on the E/W road.
Switch 3	ON	means a pedestrian would like to cross.
	OFF	means there is no pedestrian.

There are at least six LED outputs (you can add more if you like):

LED1,LED2,LED3 North/South traffic light green, yellow, red
LED4,LED5,LED6 East/West traffic light green, yellow, red

You may implement the old Boston *walk* signal by lighting up yellow and red in all directions. I was in Pittsburgh 20 years ago, and an old traffic light there went green, green+yellow, yellow, then red. Traffic should not be allowed to crash. In other words, there should not be a green or yellow on N/S at the same time there is a green or yellow on E/W. The student should exercise common sense when assigning the length of time that the traffic light will spend in each state. A walk signal should be implemented in order to guide pedestrians through the intersection controlled by your traffic light. You can add a "walk" LED or implement the Boston code.

Creative Option

You have the option to solve a different but similar problem, as long as your system has at least three binary inputs and 4 binary outputs. There will be no grade penalty or bonus for choosing this option. Instead of the traffic light conceive of a different machine to implement (e.g., elevator, stepper motor, electronic ignition, automatic braking, etc.) In addition to the regular parts of this lab (finite state machine, hardware interface, software design, TA demonstration), please include a description of the problem you are solving in enough detail that the TA can evaluate whether or not your solution works. If you choose this option your preparation must include this description.

Preparation (do this before your lab period)

1) Design the FSM state graph for your traffic light controller or other appropriate abstraction. The graph (or other description) is required at the beginning of lab.

2) Prepare a wire list with pin numbers for the switches (sensors) and the LEDs (lights). See Fig 3.2 below. Show all connections from the 6812 socket to the breadboard area. The 1µF capacitor will debounce the switch and the 22Ω resistor prevents sparks across the switch. Include a parts list with chip numbers. The 230Ω resistor value was calculated using the following equation

$$R = \frac{+5 - V_d - V_{ol}}{I_d} = \frac{+5 - 2.2 - 0.5}{10mA} = 230\Omega$$

where I_d is the desired diode current (brightness varies as 5 mA ≤ I_d ≤ 20mA), V_d is the voltage drop across the LED (a constant 2.2 volts almost independent of the current), and V_{ol} is the voltage output low voltage of the 7405.

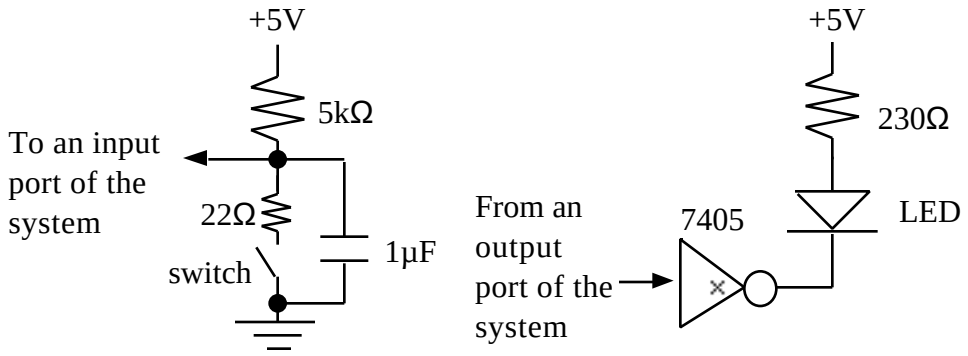

Figure 3.2. Interface of Lights and Switches to the Microcomputer.

3) Design the software for the traffic light controller. You may use either assembly code or C. Include COMMENTS that document the features of the system. There must be a 1-1 correspondence between the FSM machine graph designed in part 1) and the statically-allocated linked list data structure. If you don't use a linked list, develop other means to couple your abstraction (part 1) with the implementation (this part). Do not use for loop delays, rather use the TCNT timer to implement time delays in your FSM. A hardcopy software listing of your program is required at the beginning of lab.

Procedure (do during or after the lab period)

- Connect the circuit you have designed as part of the prep.
- Debug your system on the Adapt812 project board.

  It is important to break the system in simple well-defined modules.

  Test each module separately. Typical modules might include:

  The "real time" 1 second WAIT subroutine,
  The subroutine that reads the switches,
  The subroutine that sets the LED's,
  The linked list interpreter.

• Connect the circuit and verify proper operation.

Checkout

All features of the traffic system must be demonstrated to receive full credit. In particular, the correct operation of each of the six LED's under various switch settings must be demonstrated. Be prepared to discuss the various implementation alternatives for systems like this.

Hints

1. There is no single, "best" way to implement your traffic light. However, your scheme must be reasonable and if you don't use a linked-list data structure, you must be prepared to discuss the advantages and disadvantages. A "good" linked list solution has about 5 to 10 states in the finite state machine.
2. Your software will be graded on the efficiency of traffic flow. For example, if there are no cars currently on the roads and a new car approaches a red light, then the lights should change quickly to allow this car to proceed. On the other hand, if there are many cars going North/South and one car approaches East/West, it may not be efficient to quickly change the lights.
3. If your C program is too large (more than 4096 bytes) to fit into the single chip EEPROM, you may have to redesign your data structure, or implement this lab in assembly language.
4. Consider the following situation. Let's say there are many cars going from East to West, so the light is green in the E/W direction. Now a car comes from the South, makes a full stop, then turns right and goes East. Should the light rotate to green in the N/S direction?
5. Choose good labels for state names and software variables. Use `#define` or assembly `=` pseudo-ops to clarify your software. It is much better to design well-structured software using good labels that requires almost no comments to understand, than to write bad software then add many comments in a failed attempt to explain.
6. The following code implements a very accurate delay if the wait is less than 32767 cycles

```
void delay(void){ int Endt;   // it is important for Endt to be signed 16 bits
   Endt=TCNT+10000;           // calculate the TCNT time at the end of the delay
   while(Endt-(int)TCNT>0);}  // wait  10000 cycles
```

If you need to wait more than 32767, then you can put this code in a for loop

```
void delay(unsigned int wait){ unsigned int time;
   int Endt;   // it is important for Endt to be signed 16 bits
   Endt=TCNT+10000;           // calculate the TCNT time at the end of first delay
   for(time=0;time<wait;time++){
       while(Endt-(int)TCNT>0); // wait  10000 cycles
       Endt=Endt+10000;}}       // calculate the TCNT time at the end of the next delay
```

We will learn later in this class to implement an even more robust delay using output compare.
7. Use you real time debugging tools you learned in the last lab to test the accuracy of your delay.
8. One "creative option" idea is to interface a stepper motor, and use the 3 or more switches to control the speed, position, and direction of the motor.
9. You will be graded on how well your software solution abstracts the problem whether you do the traffic light or a "creative option". In other words, your software should be easy to debug, understand, and modify.
10. Even though the TExaS application does not simulate the traffic intersection (yet), you can connect the input/outputs to switches and LEDs and test the software functions. In particular, the MEALY12.UC, MEALY12.IO, MEALY12.SCP files can be run on the TExaS simulator. The files MOORE12.UC, MOORE12.IO, and MOORE12.SCP files can also be simulated.