ECE 152A LABORATORY 5

Objective:

In this experiment you will build a general-purpose programmable state machine. Your hardware will provide for the different functional components. A memory (RAM) will be programmed with the state table and used as a lookup table for the state update. Two multiplexers are used in the circuit. One multiplexer switches between “programming" and “execution" mode, and the other multiplexer chooses between outside input and internal data for the state update.

You will write software to program the RAM, control the circuit and operate two sample state machines: a 1-digit BCD counter and a serial adder. The specifications for the state machines are given in the experiment description; you have to design and implement them for the programmable state machine.

Deliverables:

- **There is no Pre-lab assignment for this lab.**
- Parts 1 and 2 are due in your lab section one week after the laboratory starts.
- **Lab # 5 checkout:** is due two weeks after the lab starts.
  This comprises of Parts 3 and 4.

(See the course web page and syllabus for exact due dates.)

Grading:

Grading will be done as follows for the steps completed on time:

- Part 1: 20 %
- Part 2: 20 %
- Part 3: 30 %
- Part 4: 30 %

Design Strategy:

- **Part 1:**
  - Clearly define the inputs, outputs, and state variables that you will use for the 1-digit BCD counter, and the serial adder defined below.
  - Draw the state diagram for both machines.
  - Write the state table for both machines.
- **Part 2:**
  - Implement the state machines **AS STATE TABLES** in Verilog. Use the template *template.v* on the website as a guide.
  - Simulate the state machines to ensure their correctness. Be sure to print out your simulation waveforms.
- **Part 3:**
  - Draw a chip level schematic that outlines all the connections that will be made on your breadboard. It should include power and ground connections to chips as well as the pull-up resistors with their values.
  - Breadboard your programmable state machine.
  - Implement a C program to download and run your state tables.

- Download and demonstrate your machine running the 1-digit BCD counter.
- Download and demonstrate your machine running the Serial Adder.

- **Part 4:**

  Write up a lab report that describes your design and implementation. You should include print outs or photocopies of all software code, simulation tests, and diagrams. You should write a short paragraph that describes each figure you include. Also, analyze the possible trade offs between using a programmable state machine versus a directly implemented state machine. When you would you want to use one rather than the other? Can you think of any real-world examples of a programmable state machine? Also, discuss any problems you may have had during the design and implementation of your circuit and how they might be avoided in the future.

Introduction:

A variety of hardware approaches are available for the implementation of synchronous sequential machines besides the traditional methods of realization employing small-scale integrated circuit flip-flops and gates and PLDs.

One alternative approach which provides a high degree of flexibility is to implement either Mealy or Moore machines in a “table driven fashion" by storing the state tables in a random access memory (e.g. RAM). The only additional circuitry required are a multiplexer, a data register and some small scale integrated circuits to control the initial programming and sequencing of the RAM. The major innovative feature of this method is that it provides a programmable RAM-based sequential machine whose behavior can be changed by altering the memory contents.

This lab is concerned with the realization of RAM-based state machines using the above approach. You will use a 16 by 4 RAM chip (7489), i.e. 16 words and each word is 4 bits wide. Therefore your state machines can have up to 16 states.

This experiment consists of three parts:

1. In the first stage you will design two finite state machines -- a 1-digit BCD counter and a serial adder.

2. In the second stage, a generic “programmable” finite state machine will be breadboarded and interfaced with the PC digital I/O port.

3. You will write software to initialize and operate the hardware. The state machines you designed in the first part also get a software front end: you will write routines to exercise the two state machines and provide their inputs etc.

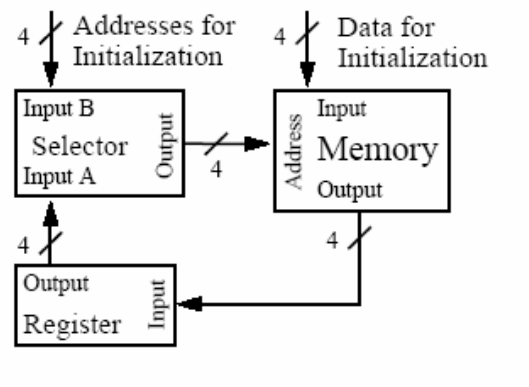

Figure 1

State Machine Design:

The first part of this experiment evolves around state machine design. You can do this part at home before you come to the lab. The sections below list the specifications that your machines must adhere to.

Generally, your state machines will be of the Moore type. However, the state machine for the serial adder can also be viewed as a Mealy type machine.

Design the state machines in four steps:

1. Define the states and the transitions between them and draw a state graph. Remember to show the transition conditions and the output(s) for all cases.

2. Write down the state table: current state, next state and next output (both possibly depending on input). These two parts should be abstract from the actual binary realization on your hardware circuit.

3. Decide how you will encode the state data in the given memory space. Your memory is a 16 by 4 RAM chip, i.e. it has 16 storage locations of 4 bits each. What is the assignment of address bits for present state and input? What are the state assignments and the outputs? *Note that these design decisions will affect your breadboard connections.*

4. Use the state tables obtained above to construct the binary “memory tables" for the RAM.

1-Digit BCD Counter:

A simple binary counter that counts modulo 10, i.e. it counts 0-9 and then loops back to 0. If the machine starts off in an invalid state, it should reset to 0 on the first clock.

Serial Adder:

Binary addition of two numbers can be performed bitwise from LSB to MSB with a simple state machine -- much like the way you manually perform a binary addition. Two bit streams are created from the operands and fed into the machine sequentially from MSB to LSB. The inputs to the machine are the bit values of the two operands at the current bit position. Output of the machine is the result bit at the current bit position -- do not forget to consider a possible carry from the previous bit! Of course, the carry is updated with each addition and is thus part of the machine state.

You can either view this machine as a Moore machine with two outputs (sum, carry) or as a Mealy machine with 2 states (carry or no carry), each of which has two transitions (sum bit 0 or 1).

Hardware Description:

Figure 2 shows a more detailed block diagram of the RAM-based sequential machine. The central component is a 64-bit RAM chip (7489) organized as 16 words by 4 bits. It stores the binary state tables that you developed from the previous section.

The quad multiplexer (74157) is used to select the address input to the RAM. When loading the memory, the computer will supply the address. Once initialization is complete, the multiplexer switches to the address stored in the register (74374).

The register (74374) stores the current RAM address. Two of its input bits always come straight from the RAM chip. The other two bits are selected by the dual multiplexer (1/2 x 74157) and can either be a two bit input from the computer or the remaining two RAM data bits.

The closed loop consisting of the dual multiplexer, memory, and data register form the sequential machine. The data path in the circuit closely reflects a state machine's canonical representation. Which components (give chip number) compute the next state? How is the state update performed?

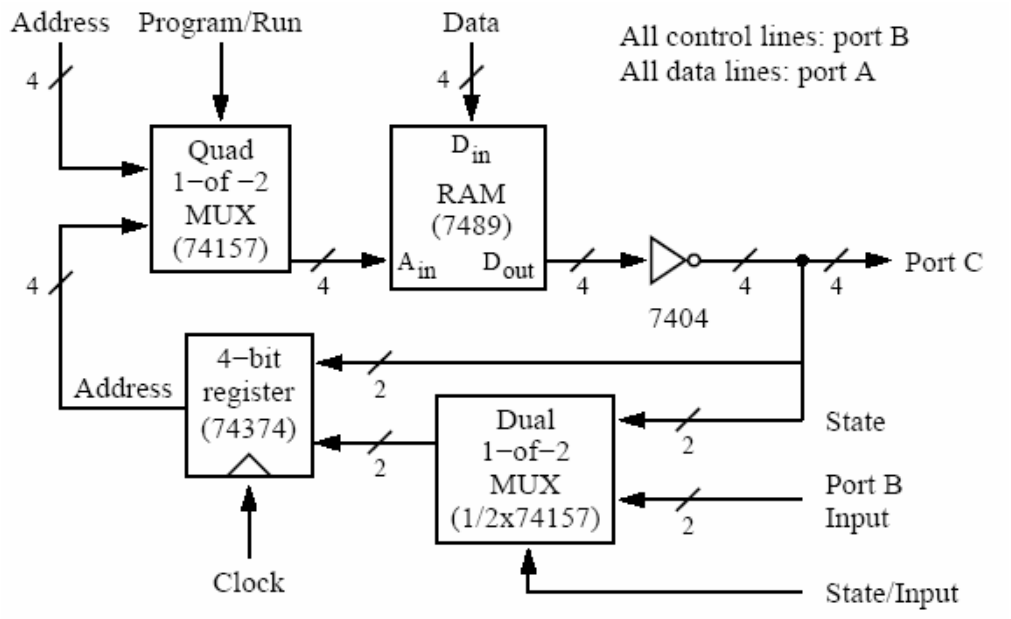

Figure 2

Breadboarding:

Using the high-level block diagram shown in Figure 1, breadboard the circuit. **Use color coding for your wires!** Construct the circuit modularly and debug as you build.

RAM chip (7489)

- The memory enable line ME' should be tied to ground.
- Use the write enable input WE' as a control signal. WE' is an active low signal, i.e. WE'=0 enables writing and WE'=1 puts the chip in read mode.

- ***The data outputs are open-collector outputs and should be individually tied high with an appropriate pull-up resistor.
- The data outputs are complemented and must be inverted by the 7404.

Multiplexers (74157)

- The select A'/B inputs are controlled through the PC (port B). Note when A'/B is logic 0, the outputs are connected to the A inputs; otherwise (logic 1) the B inputs are selected.
- The strobe input G' should be tied to ground.

Register (74374)

- The clock signal should be provided by the PC (port B).
- The output control OC' input should be tied to ground.

Digital I/O ports

- All control signals (register clock, multiplexer selects, memory write enable) are connected to port B.
- Port A is used as an output port for programming the RAM (address & data).
- Configure port C (CL or CH) for input to read back the output of your state machine.

Software:

The software should program and operate your general purpose state machine circuit. Write routines to program the RAM with the various state machine tables and drive the control signals like the multiplexer selects. The clock to the register should be provided from the PC as needed.

Provide a user interface to select and operate the desired state machine. However, the program must not contain any functionality implemented by the state machines; it should only serve as input/output terminal. Make sure that after programming the state machine does not start off in a random, but in a well defined state. How do you do that?

Programming the RAM:

Write a routine to program the RAM with your state tables. Store your binary encoded state tables in an array. Other parts of your software can then call this function to program the memory with their desired state machine. This way you avoid code replication and it is easier to change your machines in case you encounter a problem.

Timing is important for the write access to execute correctly without overwriting other memory locations etc. Generally, address and data must be valid for some time **BEFORE** WE' is pulsed low and should be held stable until **AFTER** the pulse.

Your write timing should be similar to the following sequence:

1. Setup address (that you want to write to) and data (to be written).
2. Assert the write enable for a short duration.
3. Disable writing.
4. After a short delay, you can continue writing to the memory by setting up the next address and data.

You can implement the write process as a simple loop cycling through all addresses. Be sure to insert small delays (milliseconds will be sufficient) between successive I/O board writes, so that you will meet setup and hold time requirements on the RAM. Refer to the sample code provided for Lab #4 for an example of how to provide a variable delay.

Software Interface: 1-digit BCD Counter:

The software should be clocking the counter at a regular pace. It should read back the count value from port C and display it on the screen.

Software Interface: Serial Adder:

Write a routine to input two numbers in decimal and display the result of their addition on the screen. The routine internally converts these numbers into a binary representation and then computes the sum by feeding the two operands bitwise from LSB to MSB to the state machine. Display the machine's state & output (carry and sum bit) after each binary addition. Display the final result in decimal.

Helpful Hints:

- The 7489, 74157 and 74374 IC datasheets are available on the course web page under the **datasheets** link.
- Use the file *template.v* available from the course web page.
- Use the sample C program from lab 4 as a guide in designing your interface code. It will require significant modifications to make it work. Use the Borland C universal library tutorial available on the course web page under the **datasheets** link.

Lab Report:

The report should have the following sections:

1. Abstract: describes concisely but in concrete terms what was accomplished in this lab. Example: "A combinational logic circuit was designed that displays the letters A through J on a 7-segment display, one letter per input combination." For Lab5, you will need probably 4-5 sentences.

2. Design:
   This section describes the main design decisions made by the team. This section should not replicate the lab handout but rather complement it. Describe these design decisions in a top-down manner, starting with the most abstract ones (which you probably dealt with first), going down to details. Keep the descriptions clearly organized into subsections. Provide any schematics that you used as intermediate design steps (the schematic should go into the relevant subsection). Provide any important details such as pin numbers, or any other ambiguities that you resolved during the design.

3. Conclusions:
   Give any conclusions that you derived from the lab. For example, you may discuss the advantage of storing the state table in the RAM. Under what conditions is this useful? Why would one prefer a design like this over the standard Mealy/Moore implementations that we discussed in class? Discuss issues of hardwiring versus programmability. What are the advantages and disadvantages? Try to generalize what you learned in this lab to other possible design problems. Did the lab give you new ideas to do anything differently?