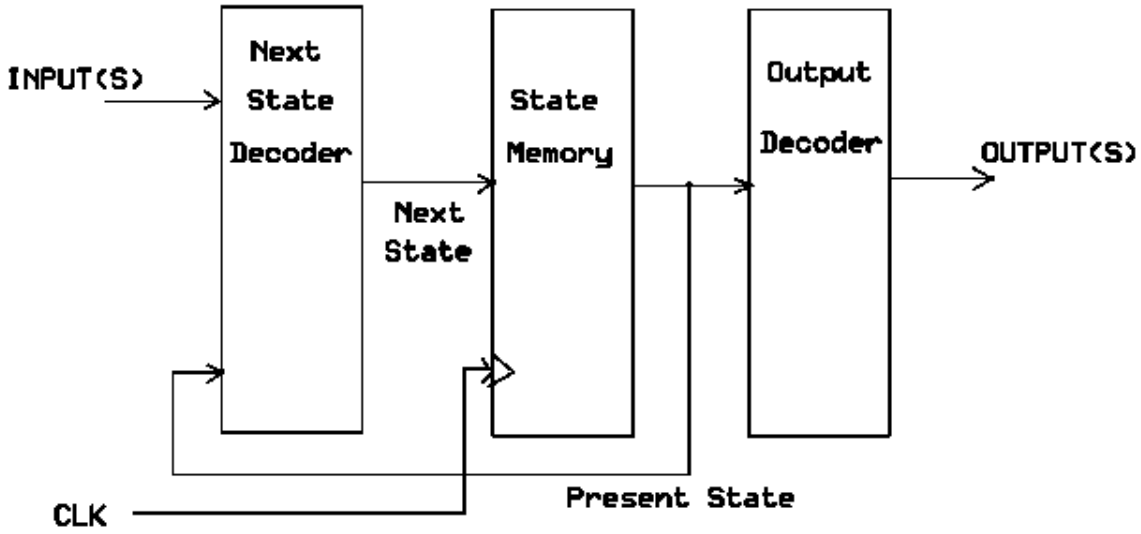
ECE 2372 Modern Digital Design July 22

Reading assignment for today: Ch 4: 4-4, 4-5

Last time we made use of one or more memory elements, (flip-flops or latches) that can each store the state of one bit of information. Using those to form "state memory", we were able to design a simple Finite State Machine (FSM) which would repeatedly go thru a specific sequence it was programmed to do. In that case, it sequenced thru the states 00, 01, 10, 11, and back to 00 and repeated forever. Left as an exercise for the student was designing a 3-bit Gray Code sequence generator. The simple FSM we developed last time is just a sequence generator, and it always generates the exact same sequence. This is quite useful in some types of control systems and even in the control unit of a computer or microprocessor or microcontroller. The control unit that generates the sequence fetch-decode-execute for instructions in a computer is a simple FSM that repeats that sequence.

Below is a general form for a type of FSM called a Moore Machine:



A Moore FSM

Let's note some things about the Moore FSM. Notice at the core we have "state memory". The size of this memory (number of bits, number of possible states) is not specified, it is merely given as a basic core element of the Moore FSM. If we needed 4 states for example, then a 2-bit memory would be sufficient since that would give us states: 00, 01, 10, and 11. If we needed 8 states, then a 3-bit memory would be adequate, giving us: 000, 001, 010, 011, 100, 101, 110, and 111. What if we just needed 5 states? Then we use a 3-bit state memory, that provides 8 states in total, and we only use the 5 that we need, and design the system to not use the other states.

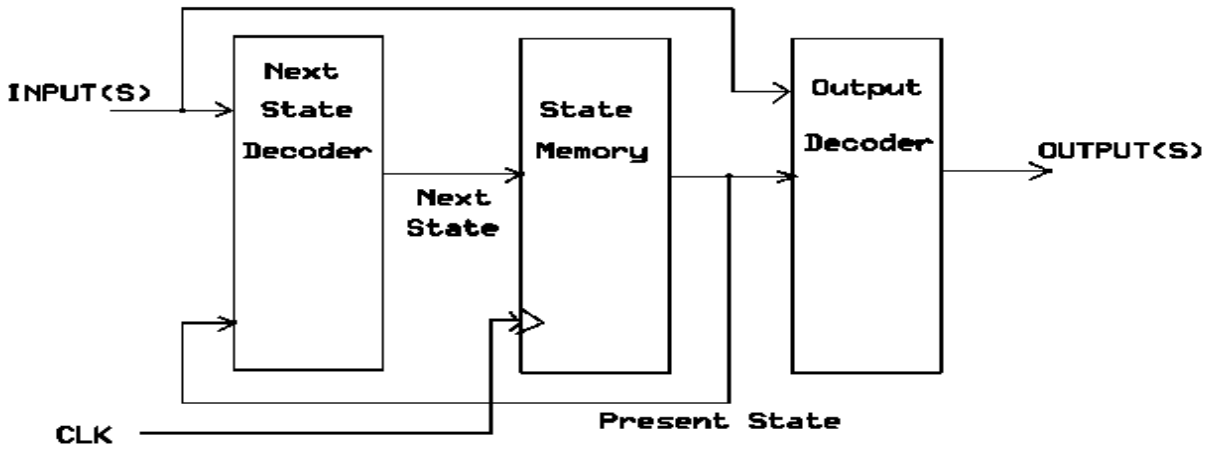
Note on the output side of the state memory, we have a set of signals that goes two places. First, the Present State is fed into the Output Decoder which then drives one or more outputs based on the Present State. Notice also that the Present State is fed back to the input of the Next State Decoder. This

feedback mechanism provides the true power of the FSM. It can make decisions about which Next State to go to based on its Present State. Notice also that the Next State Decoder takes inputs from the outside world. Coming out of the Next State Decoder is the Next State information, which is fed to the State Memory and causes an update to the State Memory with the next clock event.

Notice the power of what is happening here. The Next State is a function of the Present State and the input(s) from the outside world. This is a system that can make conditional branches based on past history and current input information. That provides tremendous power.

There are some minor simplifications one can make to the basic Moore FSM that still leaves it as a Moore Machine but slightly simplified. For example, if the output is simply the state, then the Output Decoder won't be necessary. An example where an Output State Decoder would be needed is for example a decimal display counting system. We might want the display to display 0,1,2,3,4,5,6,7,8, and 9, but the state memory might go in the sequence: 0000, 0001, 0010, 0011, etc. We would not necessarily want to display the binary codes (or BCD codes), but rather want a 7-segment LED display. In that case, we might well use a 74LS47 BCD to 7-segment decoder driver chip to take the 4 bits of state information to drive a 7-segment LED display.

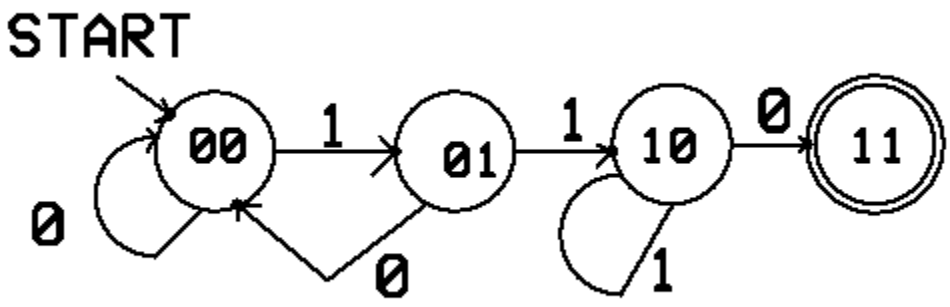
Another simplification might be to remove the input(s). Suppose we just need a simple machine that will sequence thru a series of states, repeatedly, always the same sequence, for example: 00, 01, 10 then repeat. The fetch-decode-execute sequence is an example. In this case, we always do the same sequence, in the same order. There is no need for an input to change the sequence. In that case, we might not need the Output Decoder nor the input signals. We would still need the feedback of the Present State to the input of the Next State Decoder. And, we still need State Memory. At a minimum, we need State Memory, Next State Decoder, feedback from the Present State to the Next State Decoder, and the Clock system to step the FSM thru its states. Notice that for the Moore FSM, the output of the system is always a function of the Present State only. Notice it is a "clocked" system. I.E. the output only changes when a new clock event occurs.



A Mealy FSM

Notice that although the Moore and the Mealy FSM's are similar, one very fundamental difference is that the Input Signals are also routed to the Output Decoder, and the state of the Input signals can directly affect the Output Signals. Note that since the Input signals are not directly "clocked", an Input signal can change at any time and the Output signal can also change at any time. It is not synchronized to a clock event. Generally, having output signals that are not synchronized to a clock event is a bad idea. Not always, but in many cases we much prefer to have signals synchronized to clock events. For this reason, most modern practical FSM's are of the Moore type.

One common usage for FSM's is as a sequence recognizer. Suppose we want to recognize the input sequence: 1,1,0. We would need the following FSM:



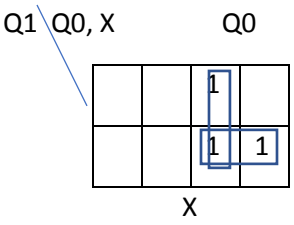
Notice some things about the notation for this FSM state diagram. We have a start state, represented by the binary bit pattern 00 here. The state of an input signal at the time of the next clock event, along with the Present State contribute to determining the Next State. Notice when we are in the Start state 00, if we get a '1' input, we move to state 01. If we get a '0' input while we are in state 00, we simply remain in state 00. Suppose we do have a '1' input in state 00, the next clock event occurs and the FSM transitions to the next state, 01. Here, to move forward in the sequence (or string) detection, we need to see another '1'. If we do see another '1', then we will move to state 10 upon the next clock event. However, if we do not have another '1' at that point, then having an input of '0' causes use to transition back to state 00. Suppose we have gotten a '1' and moved from state 00 to state 01, and then we get another '1' on the input and move to state 10. At that point we are waiting on a '0' to complete the sequence 1, 1, 0. If we get the '0' input, we do transition to state 11, which is known as an "accepting" state. Notice it has two circles. Suppose we had gotten a '1' followed by another '1' and then at that point instead of getting the '0' we are looking for, we get another '1'. Based on the machine defined above, we would simply stay in state 10 waiting on a '0'. Notice that being in state 10 says that we have already received a '1' followed by another '1'. Depending on exactly how we want the sequence (or string) recognizer to work, we might make the state diagram differently. However, the above state diagram does solve the problem of a sequence or string recognizer to recognize the input string: 1, 1, 0. Let us see if we can complete the design for this particular Moore FSM.

First, a truth table to define the operation;

Present State		Input	Next State		Output
Q1	Q0	X	D1	D0	Z
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	1	0	1	1	1
1	1	1	1	1	1

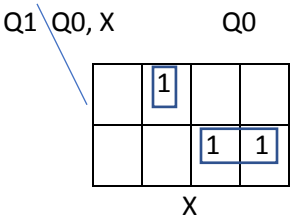
We develop K-maps for the Next State Decoders (for D1 and D0) and also for the output function Z. The Next State Decoder K-maps are each 3-variable since they involve Q1, Q0, and the input X. The output function Z only involves two variables, Q1, and Q0.

Solve for D1 first



D1 = F(Q1, Q0, X) = Q1Q0 + Q0X

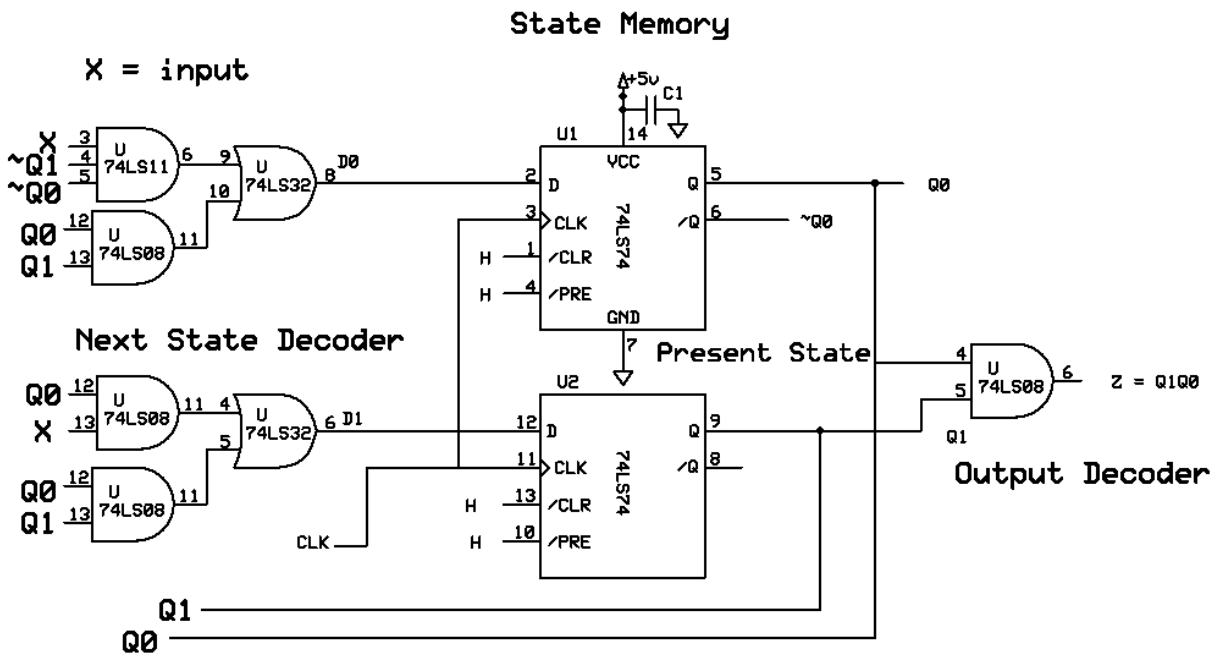
Now solve for D0



D0 = F(Q1, Q0, X) = Q1Q0 + ~Q1~Q0X

Now solve for the output function Z, which is just a function of Q1 and Q0. By inspection of the truth table, we see that Z = F(Q1, Q0) = Q1Q0. So, that is a simple AND gate.

The final design of the Moore FSM to detect the input sequence ‘110’ is given below:



As an exercise for the student, redesign the above system so it detects the input sequence '101'. That will be good practice since you will be needing to design such systems for the exam later this week.

We will delay the exam until Friday to allow more time for additional homework problems and another quiz. So, we will have a quiz on Thursday, and the next exam on Friday.