CALIFORNIA STATE UNIVERSITY
LOS ANGELES

Department of Electrical and Computer Engineering

EE-2449 Digital Logic Lab

EXPERIMENT 11
SEQUENTIAL CIRCUITS

Text: Mano and Ciletti, *Digital Design, 5th Edition,* Chapter 5 (Sequential circuit design with D flip-flops).
Required chips: **7474:** dual D-Flip-flop, plus the following:
7400: NAND (for 11.2 only), **7410:** NAND (triple 3-input), **7402**: NOR (for 11.3 only).

11.1 As mentioned in Experiment 10, sequential logic circuits are a type of logic circuit where the output of the circuit depends not only on the input, as in combinational logic circuits, but also on the sequence of past inputs that are used to determine the state of the circuit.

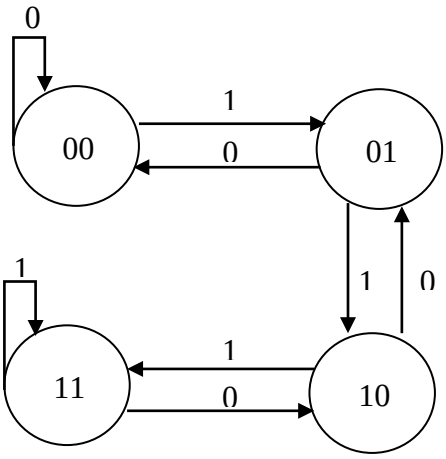
As we observed in Experiment 10, flip-flops can be used to hold (remember) the state of the system. Each flip-flop holds one bit of data and can be used to represent one state variable within a system. A system that has N states will require $\log_2 N$ state variables and hence $\log_2 N$ flip-flops. If N is not a power of two, then you have to round $\log_2 N$ up to the nearest integer.

For example, if you need 4 states, you will need two state variables. Let's assume that the names of the state variables are A and B. Then the four states are provided by the following values of A and B: AB = 00, 01, 10, and 11. If you need 8 states, you will need three state variables, for example, A, B, and C, where (ABC = 000, 001, 010, 011, 100, 101, 110, and 111). And if you need 6 states, you still need three state variables since $\log_2 6 = 2.58$ which will be 3 after rounding up (since you can't have a fractional number of state variables). In this case, you could use any 6 of the 8 states, leaving the other 2 unused.

For example, if you had an elevator controller for a building that had floors 1 through 6, the states would likely be: 001 (1), 010 (2), 011 (3), 100 (4), 101 (5), and 110 (6), with states 000 (0) and 111 (7) unused.

A state diagram is used to represent the behavior of a sequential system where the state is represented by a circle and the transitions between states is represented by an arrow (also called an edge or arc). The state diagram below represents a 2-bit saturating up/down counter.

There are two state variables, A and B, and one control input U. When U = 1, the counter will count up with each clock trigger until it reaches state AB = 11 where it will remain until U is changed to 0 (i.e. the counter is saturated). Similarly, when U = 0, the counter will count down until it reaches the lowest count AB = 00 where it will remain until U is changed to 1. If U is changed whenever the upper or lower limit state is reached, the system will oscillate up and down from 00 to 11 and back.

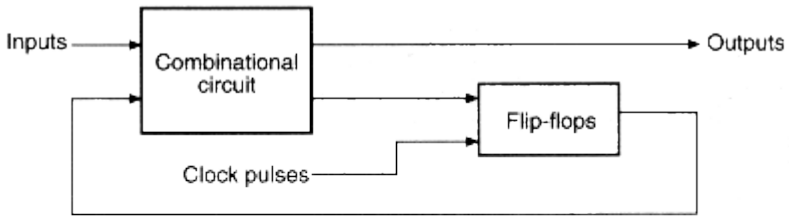


To understand how to read a state diagram, first note that the values of U appear alongside the arrows. Next, let's consider the possible transitions from state one (01). There are two possible next states: 00 and 10 (this can be determined by looking at the two arrows *leaving* state 01). On the triggering clock edge, state 01 will transition to state 00 if the input U = 0 (that is it will count down from state 01 to state 00) or transition to state 10 if the input U = 1 (that is it will count up from state 01 to state 10). Now suppose it goes to state 00. If U = 1, it will transition back to state 01. But if U = 0, it remains in state 00.

The following block diagram shows the overall design of a sequential circuit. The flip-flops are used to hold the state of the system. The Q outputs of the flip-flops represent the current state of the system or the *present state.* In the saturating counter example, the present state represents the current counter value (for example: state AB = 01 so count is 1). The state will change or transition when triggered by the clock (in Experiment 10 we learned that there are positive-edge triggered flip-flops and negative-edge triggered flip-flops).

The *present state* of the circuit and any external *inputs* to the circuit are used to determine the *flip-flop inputs* which will be used to determine the *next state* of the system. In this experiment we are using D flip-flops. Since the next state of a D flip-flop follows the D input, the flip-flop inputs actually will be the same as the next state (this is not true for the other types of flip-flops, SR, JK, and T). In the saturating counter example, if the present state is AB = 01 and the input is U = 0, the next state will be AB = 00. If we are using D flip-flops to design the saturating counter, then the D flip-flop inputs will be the same as the next state, or 00 ($D_A = 0$ and $D_B = 0$). Note, you can use any type of flip-flop in a sequential logic design but the flipflop inputs will be different depending on the type of flip-flop used.

The saturating counter example does not have an output (other than the state of the counter). Referring to the elevator example in Experiment 10, the state of the system would be the current floor that the elevator is on, the input would be the buttons pushed to call the elevator, and the output would be used to control the motors to make the elevator go up and down. As we will see in this experiment, sometimes the output depends on both the present state and the inputs (for a Mealy machine) and sometimes the output just depends on the present state (for a Moore machine).



(a) Block diagram

(b) Timing diagram of clock pulses

A state table is used to design the combinational circuit within a sequential circuit. A state table differs from a truth table in that in addition to inputs and outputs, it also represents both the present state of the system (as an input) and the next state of the system (as an output). Below is the state table for the saturating up/down counter:

Input	**Present State (p.s.)**		**Next State (n.s.)**		**Flip-Flop Inputs**	
U	**A**	**B**	**A**	**B**	**D_A**	**D_B**
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	1	0	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	1	0	1	0
1	1	0	1	1	1	1
1	1	1	1	1	1	1

Notice in the state table that the State Variables are shown twice, both as the present state value (before the clock trigger) and as the next state value (after the clock trigger). To design the combinational logic circuit, we need to follow these steps.

1. Derive the state table based on the desired behavior of the sequential circuit (represented by the state diagram.
2. Given the present and next states, based on the type of flip-flops used determine the flip-flop inputs (since we are using D flip-flops, notice how the flip-flop inputs are the same as the next state values for states A and B).
3. Use K-maps to determine the combinational logic equations for the flip-flop inputs as a function of the sequential circuit input(s) and present state. (Note, the K-maps will give us the minimum sum-of-product combinational logic equation).
4. If the sequential circuit has explicit output(s), use K-maps to determine the combinational logic equations for the outputs as a function of the sequential circuit input(s) and present state (for Mealy machines) or only as a function of the present state (for Moore machines).
5. Draw the logic diagram with one flip-flop per state variable, the combinational logic circuits for the flip-flop inputs and if used, the combinational logic circuits for the sequential circuit outputs (we don't have outputs in our saturating counter example).

D_A

	B			
U\AB	**00**	**01**	**11**	**10**
0	0	0	1	0
1	0	1	1	1

U (rows) / **A** (columns)

$$D_A = UA + AB + UB$$

D_B

	B			
U\AB	**00**	**01**	**11**	**10**
0	0	0	0	1
1	1	0	1	1

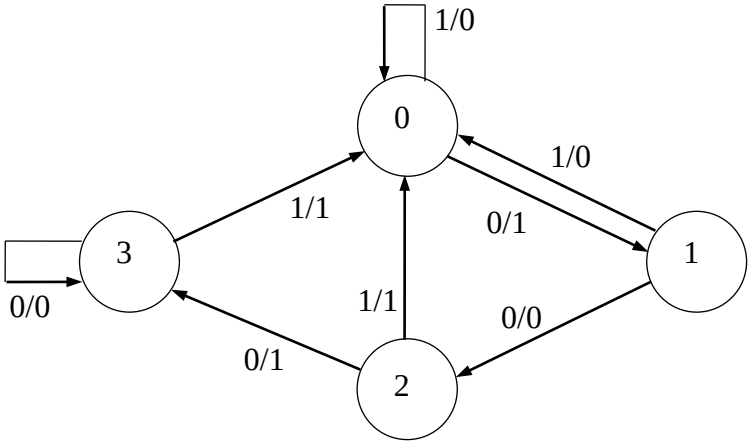
U (rows) / **A** (columns)

$$D_B = UA' + UB + A'B$$

In your lab notebook, draw the state diagram and the state table. Then verify the above K-maps and the D flip-flop input equations for the saturating up/down counter. Next, draw the circuit diagram including two D flip-flops (7474) and the logic circuits for D_A and D_B. Do not forget to include the clock input.

11.2 In this section, you will design (but not build) a sequential circuit, or state-machine, using two D-flip-flops (7474) and some gates. This will be a "**Mealy**" machine; i.e. a synchronous circuit whose output is a function of *both* present state and inputs. The circuit will consist of external input X, an output Y, and two state-variable D-flip-flops A and B. The state diagram is shown below with X/Y values along the arrows. Because this is a Mealy machine, the output Y is determined by input X as well as by the present state AB.

For example: alongside the arrow from state-0 to state-1, we see X/Y = 0/1. This means that with X = 0, there will be a transition to state-1 when the clock pulse comes. It also means that *until* the clock pulse comes and while the circuit is *still* in state-0, the output Y will be 1 as long as X = 0.



State Diagram for Mealy Machine

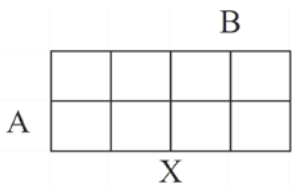
p.s.		input	output	n.s.	
A	B	X	Y	A	B
0	0	0	1	0	1
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

Partially Completed State Table

From the above state diagram, finish the state table (on your own paper). Notice in the table that each pair of rows represents the same state of AB (i.e. the same circle in the diagram).

Next, design (but don't build) the circuit. Do this with just ***three*** chips: one 7474 and 2 NAND chips. (If you need to complement A and/or B, just use inverted flip-flop outputs--there's no need to use gates as inverters. But you will need a gate to produce X' .)

For this experiment (as well as **11.3**)



1. From the state table, derive maps for DA, DB and Y (labeled as shown).
2. From the maps, derive their equations.
3. Draw the circuits using NANDs only (7400's and/or 7410's, but no 7408's or 7432's). Use DeMorgan symbols for NANDs *but only where doing so preserves the original AND/OR structure*. A diagram with only normal or with only DM symbols **is not acceptable**. (Remember: connecting wires must either have bubbles at both ends, or no bubbles at all.) Also, use the text function to label circuit inputs A or A', B or B', X or X' and outputs DA, DB, Y.

11.3* This section is similar to 11.2 except that the sequential circuit here is a "**Moore**" machine--a state machine whose output depends *only* on present state, not on inputs. The circuit has two D-flip-flops, A, B, an input X (from a switch) and an output Y (connected to an LED). Its state diagram is on the next page.

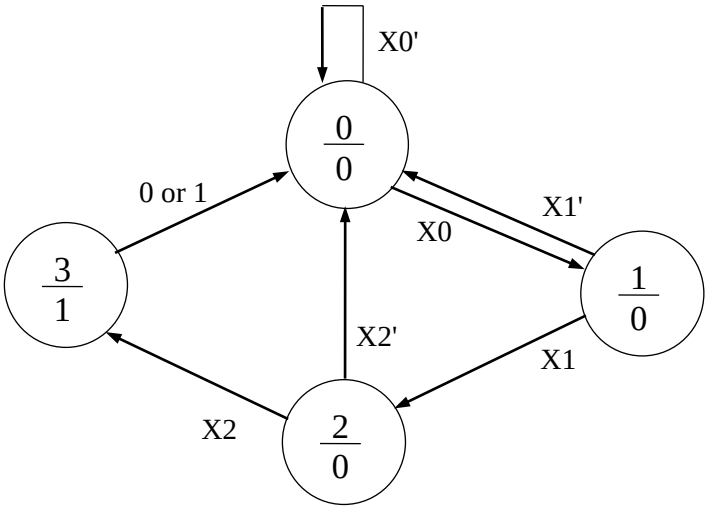
You can think of this circuit as a primitive entry-code detector. You enter a sequence of values for input X and move the circuit from state to state. (This would be like pressing a sequence of keys in a special order to gain entry to a room, a car, or whatever.)

Here, each step involves setting input X to 0 or 1 (using a switch) and then pressing a pulser. You do this at least 3 times. If you set X to the right value each time, the state machine reaches state 3 (AB =11). At this time, Y goes high and turns on its LED, indicating that you've gained entry. If at any point, you clock in a wrong value for X, the state machine returns to 0 where you have to try a new sequence for X.

Of course, with a real entry-code detector, if you didn't know the code there would be too many possible sequences to try--here there are only 8. Also, unlike here, just pressing a key enters it; you don't have to clock it in with a pulser.

In the state diagram, you see a sequence of values for X alongside the arrows (i.e. X1, X2, X3). You will have to choose these values. This will be the sequence required to get you to state 3. If the complement of any required X is entered, the circuit returns to AB = 00 and waits for a new sequence to begin. Notice that once in state 3, the next state is 00, regardless of X3's value.

In a Moore machine, output Y is not dependent on X as it was in the Mealy machine; it depends *only* on the present state of the circuit (AB). This is why it is placed under the state number inside the state circle, not alongside the arrows. So, in states 0, 1, and 2, Y = 0, indicating that the required sequence of X's is not yet complete. When it is, Y = 1, which turns on an LED to signal that the key code was successfully entered.



State Diagram for Moore Machine

Design steps:

1. Choose a sequence of values for X. Make X2 the same as X0 (0,0 or 1,1) and make X1 the opposite. Example: X0 = 0, X2 = 0, so X1 = 1 or the opposite. (This may simplify the circuit you will build.)
2. Derive the state table from the above diagram. The table has the same format as that in **11.2**.
3. From the table, draw K-maps for DA, DB, and Y, and from the maps, derive equations in their *simplest* form.
4. Draw the circuit. Only three chips are needed: a **7474** (dual D-flip-flops), a **7410,** and a **7402**. Do *not* use a 7400. Remember: a 7410 NAND gate can also be drawn as an invert-OR and a 7402 NOR gate can also be drawn as an invert-AND. **Use DeMorgan symbols where appropriate**. Inverters are not needed for A' and B' since they are available as flip-flop outputs.
5. Build the circuit. Bring state variables A and B as well as Y to LEDs so you can monitor the sequence of states and the output.

Start by momentarily grounding the clear inputs of both flip-flops to put the circuit into state 0. Then,

- Set X to X0' and clock the circuit with a pulser. The state should remain at 00.
- Repeat with X = X0. The state should move to 01.

Repeat this test method, moving to the next state by entering the correct subsequence to get there. Then enter X' for that state and return to 00. Continue until you've checked all the states. (Of course, from state 3 you can only go back to 00--there is no correct or incorrect value for X.)

Have your instructor check your results.