Analysis and Design of Sequential Systems-II
Using the D Flip Flop

Design of a Simple Up-Counter

Suppose we want to design a simple three-bit counter—a counter that just counts up:

000, 001, 010, 011, 100, 101, 110,111, 000, 001, 010, 011, 100, 101, 110, 111, 000, 001, …

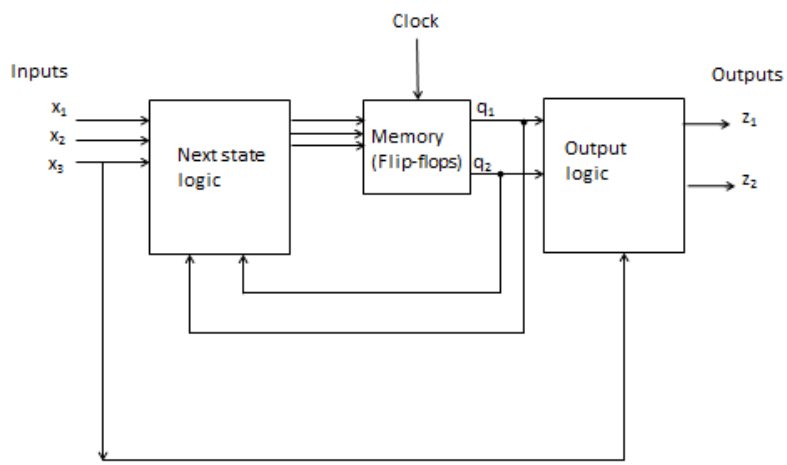
The idea is that the counter is controlled by a clock—every time a clock ticks, the counter advances by one automatically. Think: How many flip flops will we need?

Another way to look at it:

The number of flip-flops needed is

This sort of counter is the easiest place to begin our study. Let's see why.

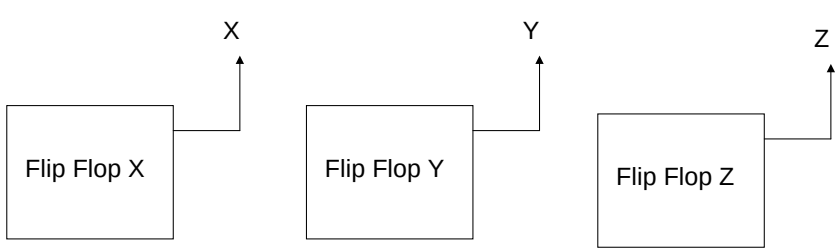
Recall from last lecture that a finite state machine (hereafter, ***FSM***) is implemented as something along the lines of:



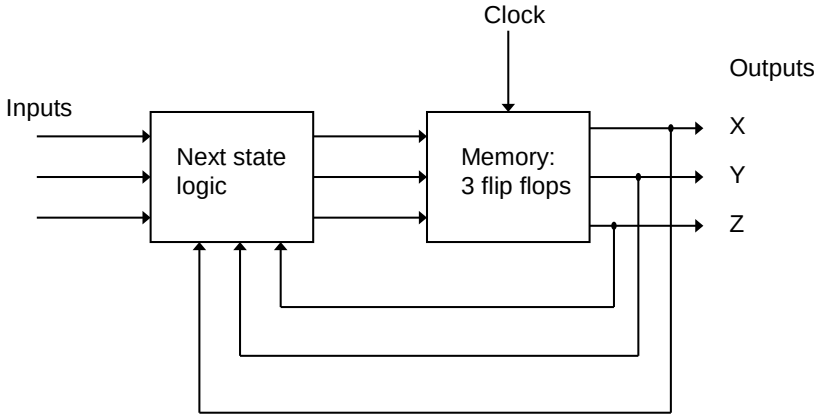
In general, a FSM may have any number of inputs, any number of outputs, any finite number of flip flops, and any number of inputs feeding to the output logic. Recall the functions of the Next State Logic and the Output Logic:

- **<u>The next state logic</u>**. Given the inputs and the current state, how do we determine the next state?
- **<u>The output logic</u>**. This function determines the output. The output function might depend only on the state, or might depend on the inputs as well as the state.

Now, for the counter, if we put some thought into this, **we should be able to read the output (the counter value) right off of the states of the internal flip flops**. Recalling our flip flops are names as *X*, *Y* and *Z*, letting Z represent the LSB, we should read the values off as:

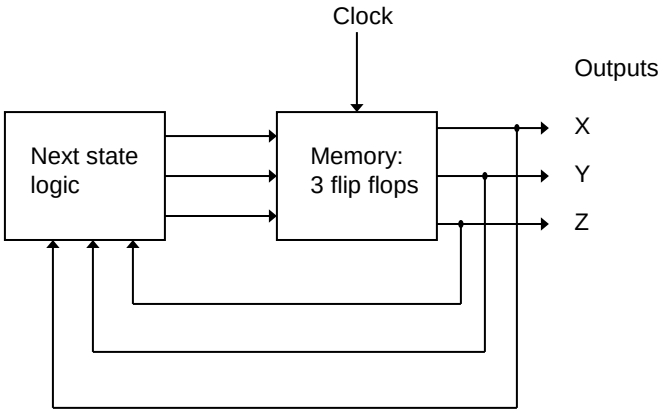


So, we should not need any Output Logic for this counter. Thus, our problem simplifies to the following (where we have changed the memory to note that we will need three flip flops):



But, this counter is even simpler! What are external inputs?

So we have:



And remember that the Next State Logic (and the Output Logic—which we don't have in this example) are entirely made up of combinational logic…the types of circuits that we are all experts on!

For starters, in order to gain confidence with designing circuits, and in order to minimize the number of new concepts thrown at you all at once, we will restrict our designs to those using *D* flip flops. *D* flip flops are the easiest to work with since the next state is equal to the value of the *D* input:

So…if we can derive an equation for the desired next state Q^*, we can then immediately implement our design with *D* flips flops by simply setting the desired next state value as the input (*D*) of a *D* flip flop.

So, now that we have given some thought to what the specific FSM (in this case, a simple 3-bit counter) should do, let's get down to the business of building these counters, selling them, and perhaps making a fortune!

So, think,… where should we start? We have some options:

- Should we first draw out the intended timing diagram?
- Should we immediately start sketching out the circuit diagram?
- Should we write out the state table?
- Should we start with the state diagram?
- Should we contemplate the mysteries of life?

Always start with the

Now, construct the state table.

Now we have to design the circuit for the next state logic. Let's look at each flip flop independently.

Think of what the next state logic does: it treats the current state as an input, and determines what value should be loaded into the flip flop at the next clock cycle (i.e., the next state).

Design the digital logic circuit to implement X^*.

Design the digital logic circuit to implement Y^*.

Design the digital logic circuit to implement Z^*.

Implement the counter using D flip flops! Use this picture as a starting point (ignoring the clock and ignoring the asynchronous inputs):

Answer:

Again, this is why we are starting to learn design by looking only at the *D* flip flop—the value of *D* is equal to the desired next state! So, once we derive the equations for the desired next state, we simply apply that to the *D* input of the *D* flip flop.

Example. A Different Counter

Design a counter that only counts up to 5, and then resets to zero and counts up again. In other word, the counter counts:

000, 001, 010, 011, 100, 101, 000, 001, …

Let's also give this counter one input, named X, that can be used to "freeze" the counter. Specifically, if $X = 1$, the counter increments normally on each clock cycle, while if $X = 0$, the counter freezes.

State diagram:

State Table:

Develop the circuit for A*.

Develop the circuit for B*.

Develop the circuit for C*.

Design the circuit:

Now, think…what happens if somehow or another the counter finds itself in state 110?

What happens if somehow or another the counter finds itself in state 111?

So, we can include these conditions in the state diagram:

This is not wonderful, although we can always ensure correct operation by

In retrospect, it would have been wiser to…

With these two examples under our belts, let's look at the general technique we have used.

<u>The General Guidelines for the Design of Sequential Systems</u>

1. From the initial description of the problem, determine the inputs, outputs, what needs to be stored in memory, and what states the system will move through
2. If necessary, code the inputs, outputs and states to binary
3. Derive the state diagram and the state table
4. Choose a flip flop type (for this lecture we are just using D flip flops; leter you will also use JK and T flip flops)
5. Derive the Next State Logic and Output Logic circuits

Example. Design a FSM that will accept a stream of bits, such as `10100110110110001`… and will output a 1 when the input stream seen thus far has an odd number of ones. Otherwise, the FSM will output a zero. This FSM is called an *odd-parity checker*.

First… could you have designed this circuit using combinational logic?

But didn't we design a parity-checker on the six-week exam that didn't have any sequential logic?

State diagram:

State table:

Revised state table (let even = 0, odd = 1)

How many flip flops are needed to represent two states?

Design the Next State Logic:

Design the Output Logic:

Build the circuit:

Example. You have been tasked with designing a sequential Moore circuit that accepts a string of bits and will detect if the string `110` occurs anywhere in the bit string. Specifically, if the most recent 3 bits are 110, the circuit will output a 1; otherwise the output will be 0. (Note that the output does not freeze at 1.)

Critique this solution: "*I'll build a circuit to check if the first three bits are 110. Then I'll use the same circuit to check if the second group of three bits (i.e., bits 4, 5 and 6) are 110. Then I'll use the same circuit to check if the third group of three bits (i.e., bits 7, 8 and 9) are 110.*"

Understanding the problem is critical! On an exam, a lack of understanding might make the problem much harder than intended, or might lead you to make a gross simplification that will garner no credit.

For this problem, let's go straight to the state table:

State Table:

Develop the circuit for A*.

Develop the circuit for B*.

Develop the circuit for C*.

Design the circuit: