## Another state transition diagram example (Chapter 3)

Design a state diagram which has one input “x” and generates a “z” if (and only if) the last four values of x were “1010”.

Let’s consider another problem from the book. Say we have a door with color-coded locks.

* Unlock door (u=1) only when buttons pressed in sequence:
  + start, then red, blue, green, red
* Input from each button: *s, r, g, b*
  + Also, output *a* indicates that some colored button pressed
* FSM
  + Wait for start (s=1) in “Wait”
  + Once started (“Start”)
    - If see red, go to “Red1”
    - Then, if see blue, go to “Blue”
    - Then, if see green, go to “Green”
    - Then, if see red, go to “Red2”
      * In that state, open the door (u=1)
    - Wrong button at any step, return to “Wait”, without opening door

Before we pick apart issues with this design, let’s notice some stuff.

1. *From every state we always know exactly where to go.*
   1. Put differently, for every input in every state we have ***exactly*** one state to go to.
   2. Notice that not all inputs need be used in each state…
2. We’ve made the initial state clear (how?)
3. Each state has all of the outputs defined (okay, only one in this case).

The above three things are all important properties of a state diagram!

#### Question

Why would this diagram not actually work in the real world? (Think about what would happen if you held the button for more than one clock tick (which you would)). How would you fix that?

The moral of the story here is that for embedded systems (things used by people) you need to keep in mind that we *do* have a clock even though it doesn’t show up in our state transition diagrams!

# Back to gates for a bit (3.2)

We’ve seen that we’d like to do things on the rising edge of the clock. How do we manage that?

Answer: two back-to-back latches and an inverter.



Clk

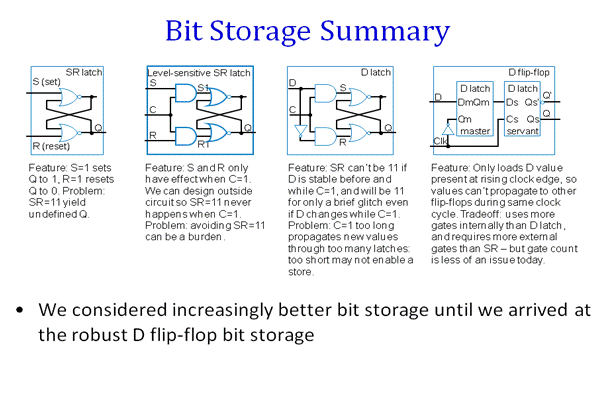
D/Dm

Cm

Qm/Ds

Cs

Qs

****

D

C

Q

D

C

Q

D latch with enable:

D Flip-flop:

Finite State Machines (sections 3.3 & 3.4)  
We’ve touched on designing state transition diagrams. Let’s now focus on the mechanical process of going from a state-transition diagram to a gate-level implementation of the state machine (and then go the other way!)

A “simple” example:

!a

a

!a

a

Fill in the table, and then draw the devices!

**State/Output Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current**  **State** | **Input** | | **Output**  X |
| a=0 | a=1 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**State assignment:**

|  |  |  |
| --- | --- | --- |
| **State** | **Q1** | **Q0** |
|  | 0 | 0 |
|  | 0 | 1 |
|  | 1 | 0 |
|  | 1 | 1 |

**State/Output Table  
(Different format)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Inputs** | | | **Outputs** | | |
| **Q1** | **Q0** | **a** | **D1** | **D0** | **X** |
| 0 | 0 | 0 |  |  |  |
| 0 | 0 | 1 |  |  |  |
| 0 | 1 | 0 |  |  |  |
| 0 | 1 | 1 |  |  |  |
| 1 | 0 | 0 |  |  |  |
| 1 | 0 | 1 |  |  |  |
| 1 | 1 | 0 |  |  |  |
| 1 | 1 | 1 |  |  |  |

## FSM Design – One more

That problem is a bit more complex than works well for a simple example, so let’s go with:

|  |
| --- |
| Design a state transition diagram (the lines and circles) which solves the following problem.  There is one input, X, and one output, A.  A is high if the last 3 inputs on X were 101.  X: 11101011010001111010  A: 00000101001000000001  As before, the output is going high \_after\_ the input pattern occurs due to the delay inherent in a Moore machine. |

**State Transition Diagram:**

(How many flip-flops are needed?)

**State/Output Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current**  **State** | **Input** | | **Output**  A |
| X=0 | X=1 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**State assignment:**

|  |  |  |
| --- | --- | --- |
| **State** | **Q1** | **Q0** |
|  | 0 | 0 |
|  | 0 | 1 |
|  | 1 | 0 |
|  | 1 | 1 |

**State table in a different format**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q1** | **Q0** | **X** | **D1** | **D0** |
| 0 | 0 | 0 |  |  |
| 0 | 0 | 1 |  |  |
| 0 | 1 | 0 |  |  |
| 0 | 1 | 1 |  |  |
| 1 | 0 | 0 |  |  |
| 1 | 0 | 1 |  |  |
| 1 | 1 | 0 |  |  |
| 1 | 1 | 1 |  |  |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Q1** | **Q0** | **X** | **D1** | **D0** |
| 0 | 0 | 0 |  |  |
| 0 | 0 | 1 |  |  |
| 0 | 1 | 0 |  |  |
| 0 | 1 | 1 |  |  |
| 1 | 0 | 0 |  |  |
| 1 | 0 | 1 |  |  |
| 1 | 1 | 0 |  |  |
| 1 | 1 | 1 |  |  |

D1=\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

D0=\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A=\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Where do each of those things go in the above architecture?

Draw the circuit: