Finite State Machine Design

COMP370

Intro to Computer Architecture

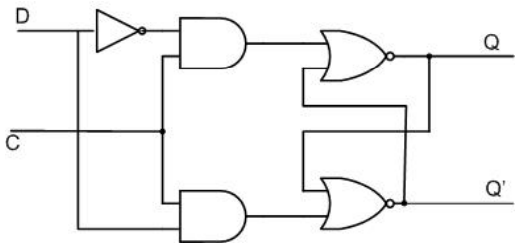
Flip-Flops and Finite State Automata

- Finite State Automata (FSA) have states.
- The system has to remember the current state.
- Flip-Flops are one bit memories.
- Flip-Flops can be used to remember the current state of an FSA.
- A sequential logic circuit using Flip-Flops can implement an FSA.

D Flip-Flop

- A D flip-flop has only one data input, D, plus enable, C

C	D	Q_{next}	Comment
0	X	Q_{prev}	No change
1	0	0	Reset
1	1	1	Set



Clock

- Many sequential logic circuits use a clock input.
- A flip-flop can change state on a clock rising edge.
- The new state will be available on the falling edge.

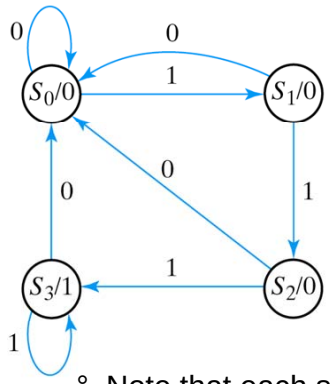
Overview of Design Process

1. Specifications start with a word description
2. Create a Finite State Automata to model the system
3. Create a state table to indicate next states
4. Convert next states and outputs to output and flip flop input equations
5. Simplify logic expressions
6. Draw resulting circuits

Example

- Create a sequential circuit that detects when three or more consecutive 1 inputs have been received.
- Inputs are checked each clock pulse.
- The output will be true only when three or more 1 inputs have been detected.
- The output will be false when zeroes and less than three inputs of 1 have been received.

FSA for 3 Consecutive 1 inputs



- State S_0 : zero 1s detected
- State S_1 : one 1 detected
- State S_2 : two 1s detected
- State S_3 : three 1s detected

° Note that each state has 2 output arrows

Moore Machine

- With a Moore Finite State Automata, the output is associated with the state of the FSA.
- Often the new state will be available during the second half of a clock pulse.
- The output is related to the state and not how the FSA got to the state.
- In our example, state 3 (three consecutive 1's) has an output of 1. All other are zero.

Flip Flops and States

- The output of a D flip-flop is Q. The current state of the flip-flop is Q.
- The input to a D flip-flop is labeled “D”. Setting D to a value (when the clock is true) changes the state of the flip-flop.

Flip-Flops to Remember States

- There are four states. It takes $\log_2(4) = 2$ bits to remember a number from 0 to 3.
- Two Flip-Flops can remember the two bit state number, $\mathbf{Q_1}$ and $\mathbf{Q_0}$.
- The current state and the input determine the next state and the output.
- In our example we will use two D flip-flops.
- The inputs to the flip-flops will be $\mathbf{D_A}$ and $\mathbf{D_B}$

Flip-Flops to State

- The two flip-flops each represent one bit.
- Two bits can hold four different numbers.
- Each of the four states is represented by a number.

State	Q_1	Q_0
0	0	0
1	0	1
2	1	0
3	1	1

Truth Table for FSA

Old Q_1	Old Q_0	input	New D_A	New D_B	output
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	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Simplify Each Result

Old Q_1	Old Q_0	input	New D_A
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

	Q'_0 in'	Q'_0 in	Q_0 in	Q_0 in'
Q'_1	0	0	1	0
Q_1	0	1	1	0

new D_A = in * Q_1 + Q_0 * in

Simplify new D_B

Old Q_1	Old Q_0	input	New D_B
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

	Q'_0 in'	Q'_0 in	Q_0 in	Q_0 in'
Q'_1	0	1	0	0
Q_1	0	1	1	0

new D_B = in * Q_1 + Q'_0 * in

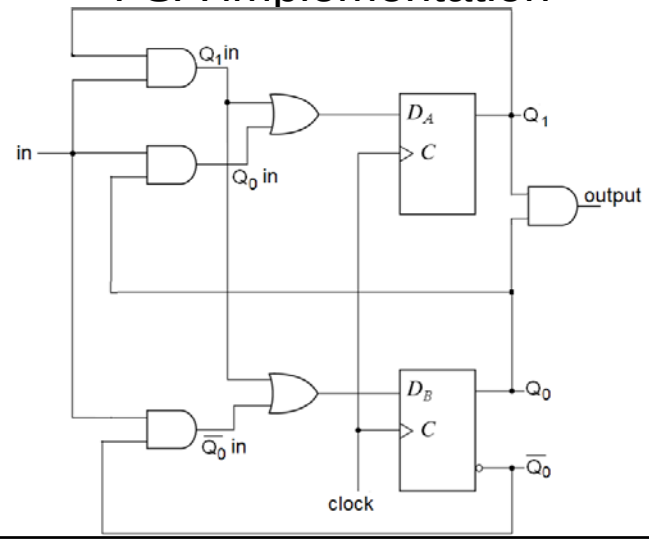
Simplify Output

Old Q_1	Old Q_0	input	output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

	Q'_0 in'	Q'_0 in	Q_0 in	Q_0 in'
Q'_1	0	0	0	0
Q_1	0	0	1	1

output = Q_1 * Q_0

FSA implementation



Flip-Flops Required

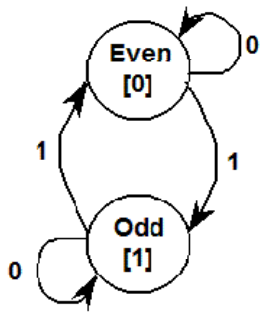
- If there are **n** states in the FSA, it will require $\log_2(\mathbf{n})$ flip-flops.
- If **n** is not an even power of two, you have to round up.
- When n is not an even power of two, the truth table may have ***"don't care"*** entries.

Example: Parity Checker

- Even parity is when the count of the "1" bits is an even number.
- We want the output to be true when the parity is even.

Parity FSA

- The parity changes when a 1 is input
- The parity stays the same on 0 input.

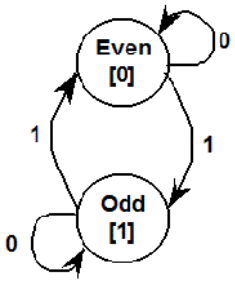


How many flip-flops will be required to implement the parity?

1. 1
2. 2
3. 3
4. none

FSA Truth Table

Q	In	D	out
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

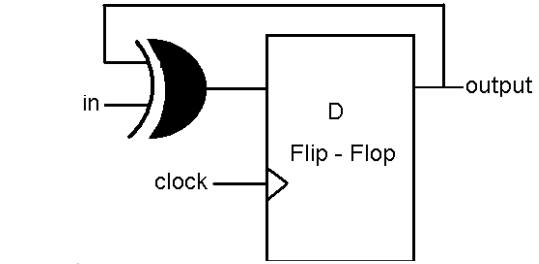


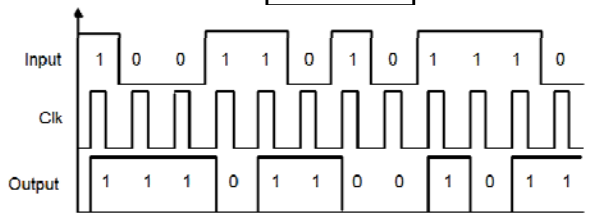
output = Q

new D = Q' * in + Q * in'

new D = Q XOR in

Parity Implementation

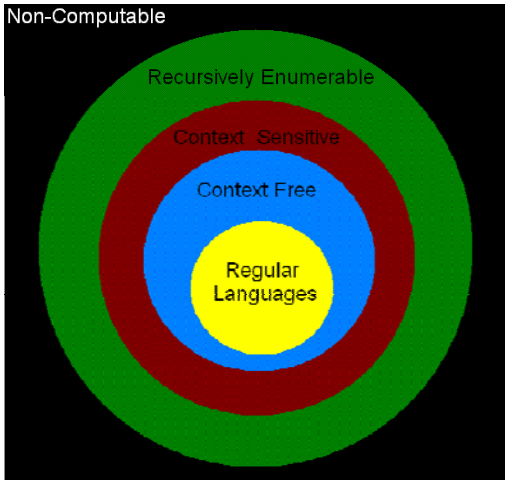




Computer Theory

- In the 1930's people started to develop the theory of computing.
- Noam Chomsky, a linguist, defined a series of machine that could recognize input of different complexity.
- There is an equivalence between computing solutions and recognizing an input language.

Chomsky Hierarchy



Regular Languages

- A regular language can be recognized by a Finite State Automata.
- Compilers use FSA to remove comments, separate strings and numbers and divide the input into a series of “tokens”.

Context Free Languages

- Most computer languages, like Java, C++, Pearl and PHP, are context free languages.
- Context Free languages can be recognized by a FSA with a stack memory.

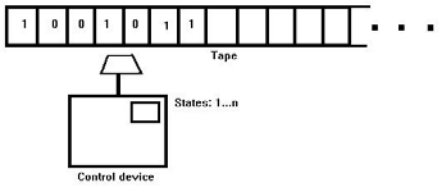
Context Sensitive and R.E.

- Context Sensitive languages and Recursively Enumerable languages can be recognized by a Turing Machine.
- A Turing Machine can compute anything that a modern “real” computer can.
- The Turing-Church thesis states:

  “Every function which would naturally be regarded as computable, can be computed by a Turing machine”

Turing Machine

- A Turing Machine has a FSA and a tape for memory.
- Based on the state of the FSA and the current symbol on the tape, the Turing Machine can write a new symbol, move the tape left or right and change the state of the FSA.



Non-Computable

- There are problems that cannot be computed.
- There are several well defined problems for which there cannot be a computer solution.

Halting Problem

- You cannot write a program that will read in the source code of any program and tell if that program will get caught in an infinite loop.
- You can write a program that can detect infinite loops in many programs, but not all programs.

Post Correspondence Problem

- Image you have a piles of dominos that each have a binary number on the top and bottom.
- Can you put the dominos in a series so the numbers on the top and bottom are the same? You can use each domino as often as you like.

100	0	1
1	100	00

Solution to Simple Problem

Dominos

100	0	1
1	100	00

Solution

100	1	100	100	1	0	0
1	00	1	1	00	100	100

1001100100100

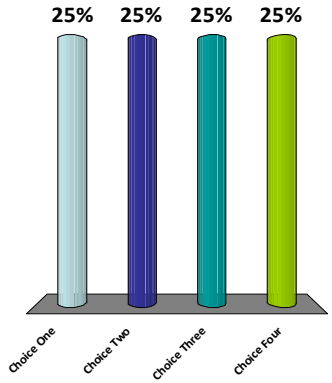
From Simple to Impossible

- The simple example had only three dominos with at most three binary bits.
- Slightly harder problems have much longer solutions.
- The solution for these dominos requires a string of 216 dominos.

101	010	1
1	1101	01

The C++ compiler may give an error that a program will create an infinite loop. How can this be?

1. The halting problem only applies in theory
2. Some infinite loops can be detected
3. C++ is context free
4. All of the above



FSA for 2 Consecutive 1 inputs

- Design a flip-flop circuit that will recognize input when there are 2 consecutive 1's.
- Write the FSA.
- Create the Truth Tables
- Define equations for the states and output.
- Draw the logic circuit.