Logic Design (Part 5)

Sequential Logic Devices & Sequential Circuits

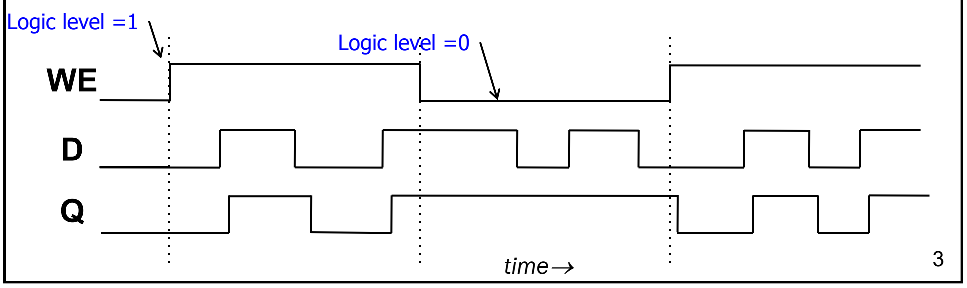
Based on slides © McGraw-Hill
Additional material © 2013 Farmer
Additional material © 2020 Narahari

Recap: Gates,Comb. Devices and Latches

- •Basic logic gates (AND, OR, NOT,….)
  - Built using transistors
- •Combinational (more complex) logic devices
  - Multiplexers, Decoders, Adders, Sub, Multipliers, Comparators (x=y?)
  - Behavior modeled using a Boolean function
  - Built using logic gates
- •Our first storage device….D-Latch and RS-Latch
- n-bit register using D-Latches
- model of Memory using Multiplexer, Decoder and Latches

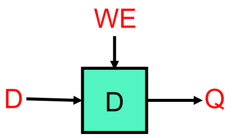
D-Latch Timing Diagram

- The diagram below is called a “Timing” Diagram
  - Our D-Latch is previous-state dependent
    - We can think of this as a time dependency
    - Moving to the right on diagram, represents forward moving time
  - The inputs & outputs to our D-Latch are on left
    - Inputs/Outputs can be either “HIGH” (logic 1) or “LOW” (logic)
  - Think of this as a time-dependent truth table

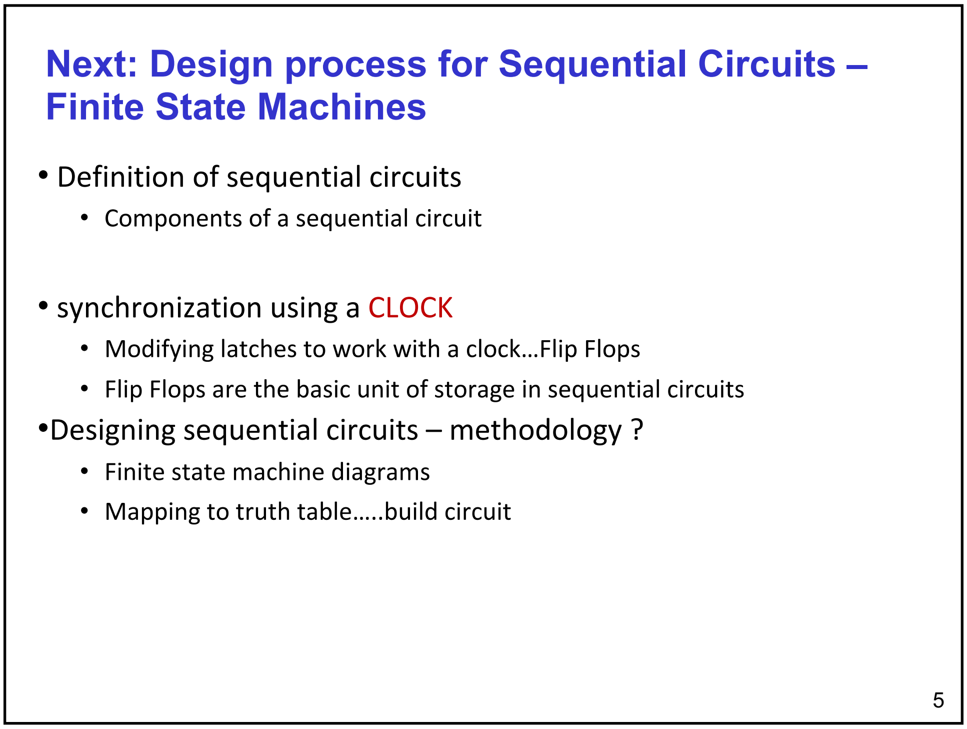


Storage Devices: Register and Memory

- Using a device (D-Latch) that can store a bit build more storage devices
  - Abstract the device: input D, WE; output/storage Q

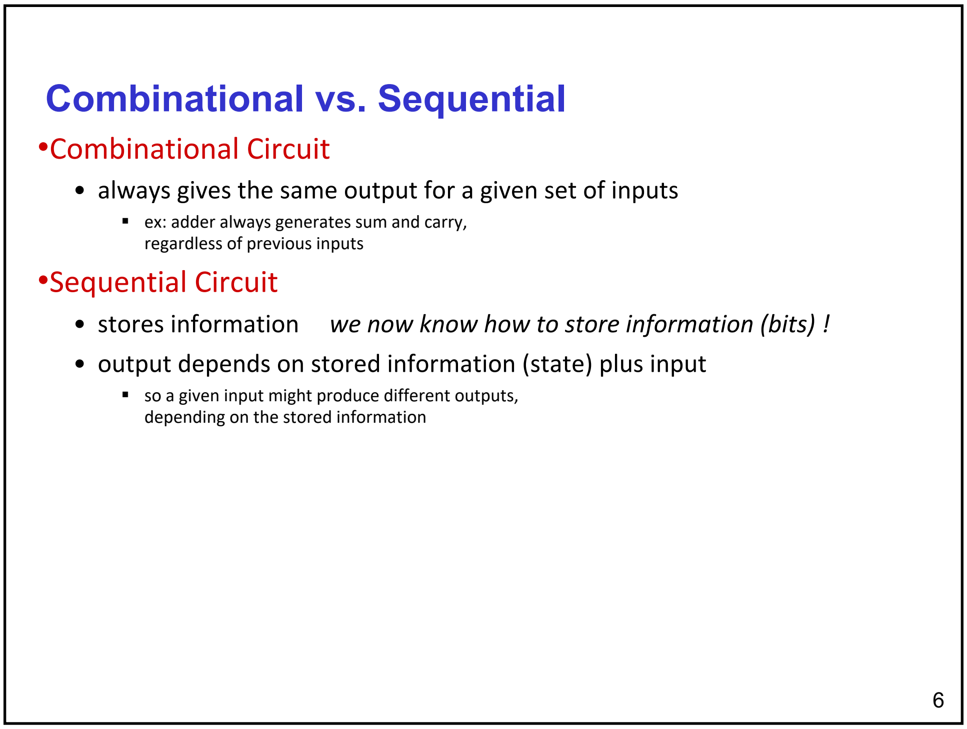


- Temporary storage in a computer…Register
  - This is where variables are stored before being sent to the arithmetic unit for operations on them
  - build an n-bit register using latches
- “main” memory



Next: Design process for Sequential Circuits – Finite State Machines

- Definition of sequential circuits
  - Components of a sequential circuit
- synchronization using a CLOCK
  - Modifying latches to work with a clock…Flip Flops
  - Flip Flops are the basic unit of storage in sequential circuits
- Designing sequential circuits – methodology ?
  - Finite state machine diagrams
  - Mapping to truth table…..build circuit

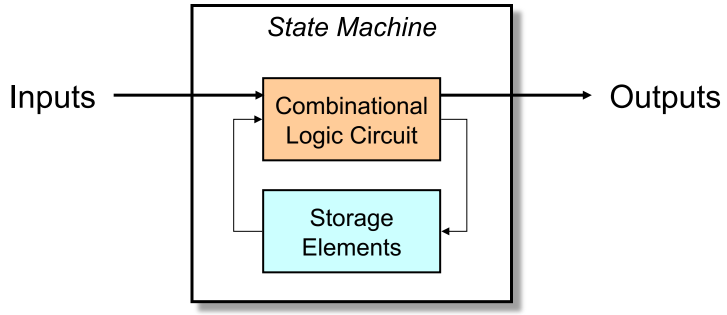


Combinational vs. Sequential

- Combinational Circuit
  - always gives the same output for a given set of inputs
    - ex: adder always generates sum and carry, regardless of previous inputs
- Sequential Circuit
  - stores information *we now know how to store information (bits) !*
  - output depends on stored information (state) plus input
    - so a given input might produce different outputs, depending on the stored information

Sequential Circuit Schematic & State Machine

- sequential circuit
  - Combines combinational logic with storage
  - "Remembers" state, and changes output (and state) based on inputs and current state



1.combinational circuit to compute output and next state

2.storage elements to store state

Sequential Circuits – The agenda

- Definition of sequential circuits
  - Components of a sequential circuit

- synchronization using a CLOCK

- Modifying latches to work with a clock…Flip Flops
  - These become the basic unit of storage in sequential circuits
- Designing sequential circuits – methodology
  - Finite state machine diagrams
  - Mapping to truth table…..build circuit

Are we ready to design sequential circuits and finite state machines ?

- When do states (content in memory) change in a machine ?
- Do we let it change at arbitrary times ?
  - Synchronous or asynchronous
- What do you think happens in a computer ?

- Example: Z = Y + X
  - Fetch Y from memory
  - Fetch X from memory
  - Send to Adder
  - Store result in Z

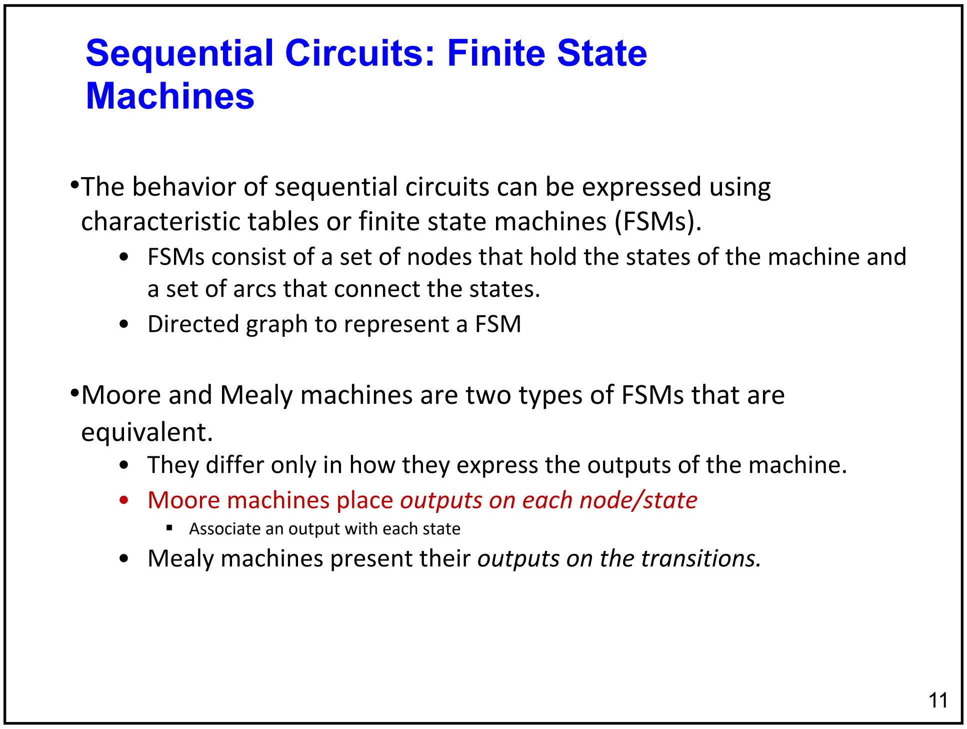
1. X,Y both need to be fetched before sending to adder

2. Output of adder should not be stored until addition is complete on X,Y

Synchronous Circuits…enter the Clock!

- current implication: states/storage/inputs/outputs can change any time….this is NOT how it works in a computer
- Reality: events take place in a synchronized manner – changes occur or are recorded at specific instances in time
- Different components all march to the same beat…

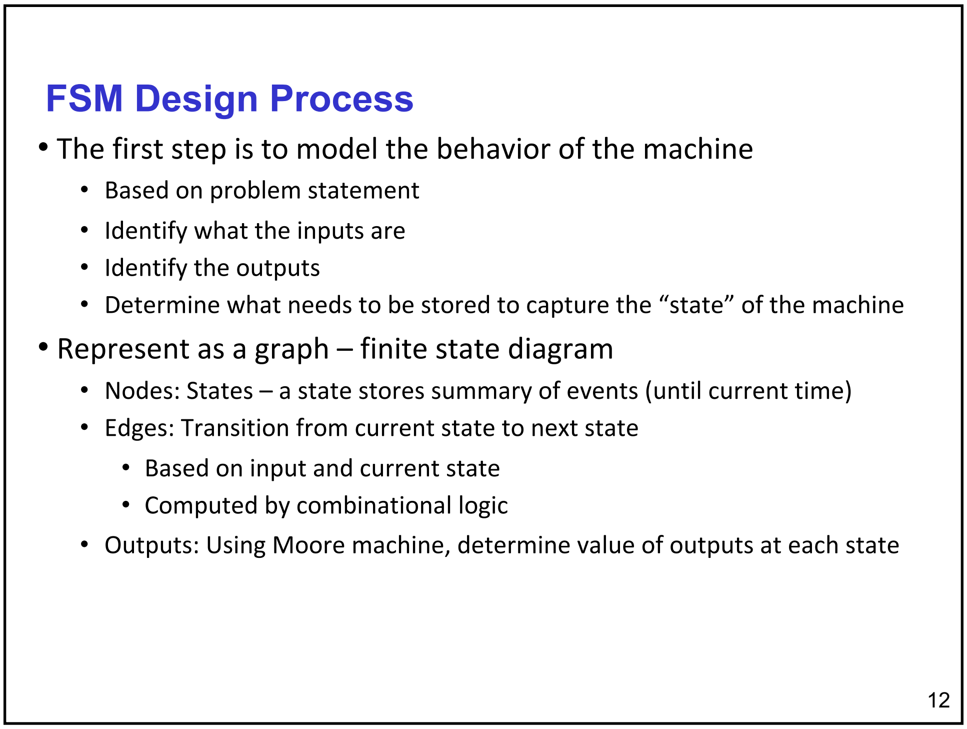
  the CLOCK



Sequential Circuits: Finite State Machines

- The behavior of sequential circuits can be expressed using characteristic tables or finite state machines (FSMs).
  - FSMs consist of a set of nodes that hold the states of the machine and a set of arcs that connect the states.
  - Directed graph to represent a FSM

- Moore and Mealy machines are two types of FSMs that are equivalent.
  - They differ only in how they express the outputs of the machine.
  - Moore machines place *outputs on each node/state*
    - Associate an output with each state
  - Mealy machines present their *outputs on the transitions.*

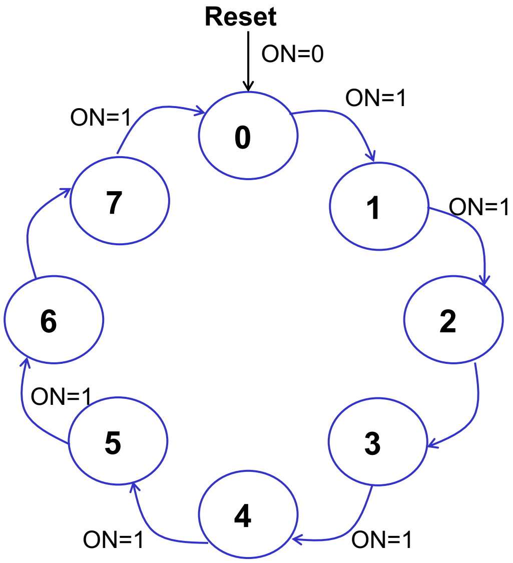


FSM Design Process

- The first step is to model the behavior of the machine
  - Based on problem statement
  - Identify what the inputs are
  - Identify the outputs
  - Determine what needs to be stored to capture the “state” of the machine
- Represent as a graph – finite state diagram
  - Nodes: States – a state stores summary of events (until current time)
  - Edges: Transition from current state to next state
    - Based on input and current state
    - Computed by combinational logic
  - Outputs: Using Moore machine, determine value of outputs at each state

Design a Counter: counts from 0 to 7

Finite State Machine Representation of Counter: Counter to count from 0 to 7 (while switch is ON)



Bubbles represent all possible "states" for the machine

Arrows show movement from one state to the next while switch ON=1.

If ON=0, FSM goes to state 0

Transitions occur at pulse of the clock

Example: A Vending Machine

- Accept user input (coins), when total is at least 50 cents dispense output (candy)
  - We will not model the change to be returned, and only care if the input is at least 50 cents
- Input valid coins:
  - Q (25cents) D (10) or N (5)
- What should it keep track of ?
  - current total
  - Is it 50 cents or more ?
- When it reaches 50 or more:
  - Generate output

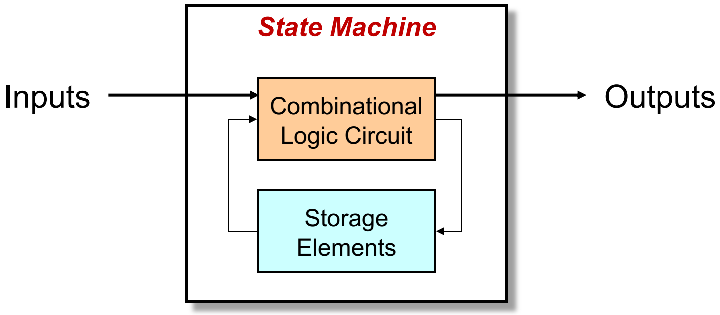
- States of the machine ?
  - What should each state capture ?
  - How many states ?

Finite State Machine => Sequential Circuit

- sequential circuit implements a Finite state machine

  1.combinational circuit to compute output and next state

  2.storage elements to store state



Sequential Logic

- Where do we start:
- Build a device, using combinational logic devices, to store a value….✓ *done!!*
  - D Latch (and RS Latch) – stores 1 bit
  - concept of memory

- Build other storage devices using the D Latch and logic devices we have at our disposal (i.e., in our *library*)… ✓ *done!!*

- What is the methodology behind design of sequential logic circuits
  - Finite State Machines to Truth Tables to Circuit
  - How do we synchronize/coordinate changes in state ? When do we allow states to change ? ….. CLOCK

- Combine sequential and combinational logic devices to “assemble” a simple processor!

Latches and Flip-Flops and Clock

- Latch: basic circuit for storage (a D-latch stores 1 bit)
  - Operate on changes in Level (i.e., 1 or 0)

- Flip-flop:
  - Sequential circuits take input from output of storage
  - Latches that work on change of level can lead to unstable sequential circuits
    - As level changes the outputs change --- inputs change!
  - Flip-Flop circuits designed to operate properly when they are part of a sequential circuit
    - Modify D Latch to get a D Flip Flop (DFF)
    - Flip Flop changes state at the ‘instant’ that the level changes

- Clock:
  - Need to coordinate and synchronize when states change…
  - Use Clock to enable or disable the devices in a timed manner

Are we ready to design sequential circuits and finite state machines ?

- Is something missing ?

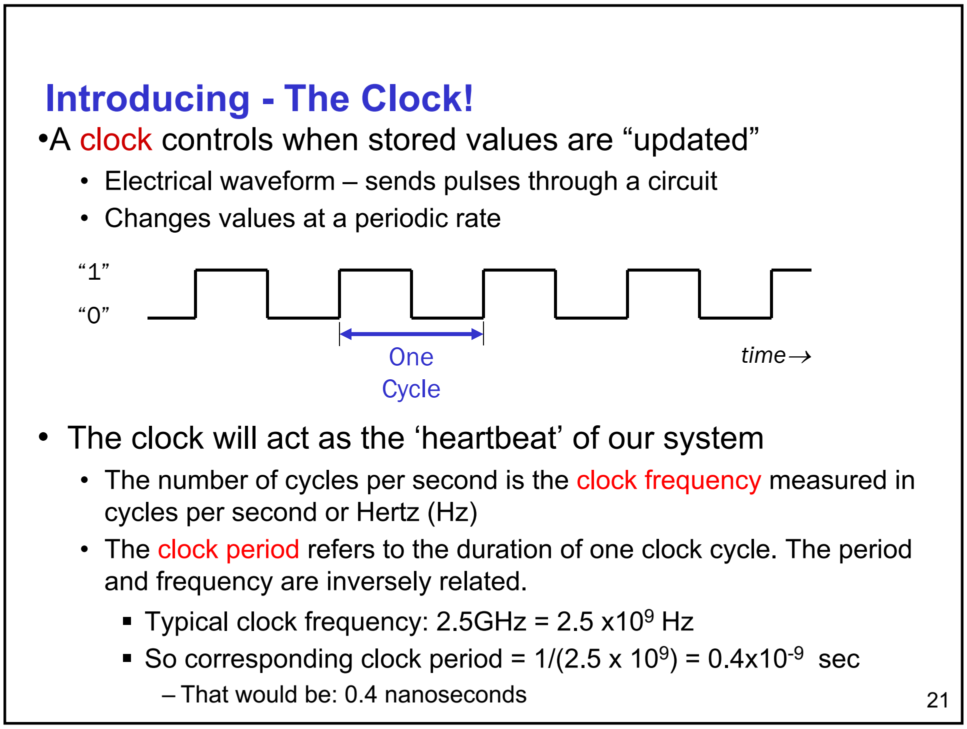
- When do states change in a machine ?

- Do we let states change at arbitrary times ?

- What do you think happens in a computer ?

Clocked Flip-Flops/Circuits

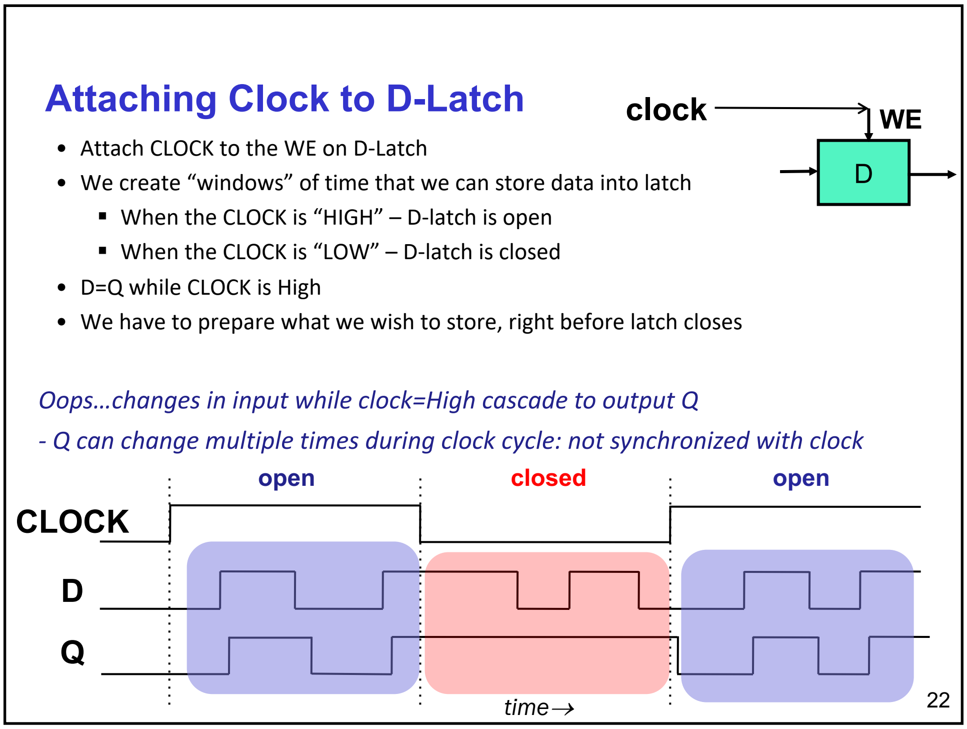
- Subsystem in a computer consists of a large number of combinational and sequential devices
  - Each sequential device is like latch which is in one of two states
  - As machine executes its cycle, the states of all sequential devices change with time
- To control large collection of devices in an orderly (synchronized) fashion, machine maintains a clock
  - Requires all devices to change their states **at the same time**
  - Clock generates sequence of pulses
  - Much easier to design, debug, implement, and test
- How do we change latches so that they allow change in state synchronized with the clock ?
- Sequential logic circuits require a means by which events can be sequenced…..clock!



Introducing - The Clock!

- A clock controls when stored values are "updated"
  - Electrical waveform – sends pulses through a circuit
  - Changes values at a periodic rate

- The clock will act as the 'heartbeat' of our system
  - The number of cycles per second is the clock frequency measured in cycles per second or Hertz (Hz)
  - The clock period refers to the duration of one clock cycle. The period and frequency are inversely related.
    - Typical clock frequency: 2.5GHz = 2.5 x10^9 Hz
    - So corresponding clock period = 1/(2.5 x 10^9) = 0.4x10^{-9} sec
      - That would be: 0.4 nanoseconds



Attaching Clock to D-Latch

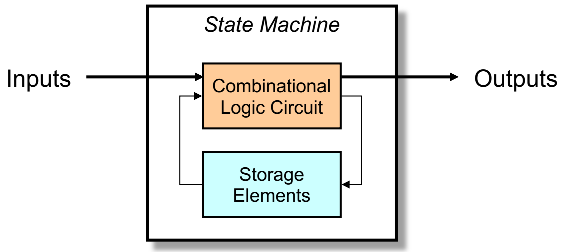
- Attach CLOCK to the WE on D-Latch
- We create "windows" of time that we can store data into latch
  - When the CLOCK is "HIGH" – D-latch is open
  - When the CLOCK is "LOW" – D-latch is closed
- D=Q while CLOCK is High
- We have to prepare what we wish to store, right before latch closes

Oops…changes in input while clock=High cascade to output Q

- Q can change multiple times during clock cycle: not synchronized with clock

Why Clocked D-latches may not work

- Sequential circuit: the next state (and output) depend on values of current state and input
- If input to D-latch changes while clock is High, then next state (and output) can change during this single clock cycle
  - state we want to store as next state could be overwritten..*Race Condition*
- we want to force changes (in state and input) to be synchronized to the clock – at a precise instant in time
  - allow input/states to be read and output/next state to change ONCE per clock cycle

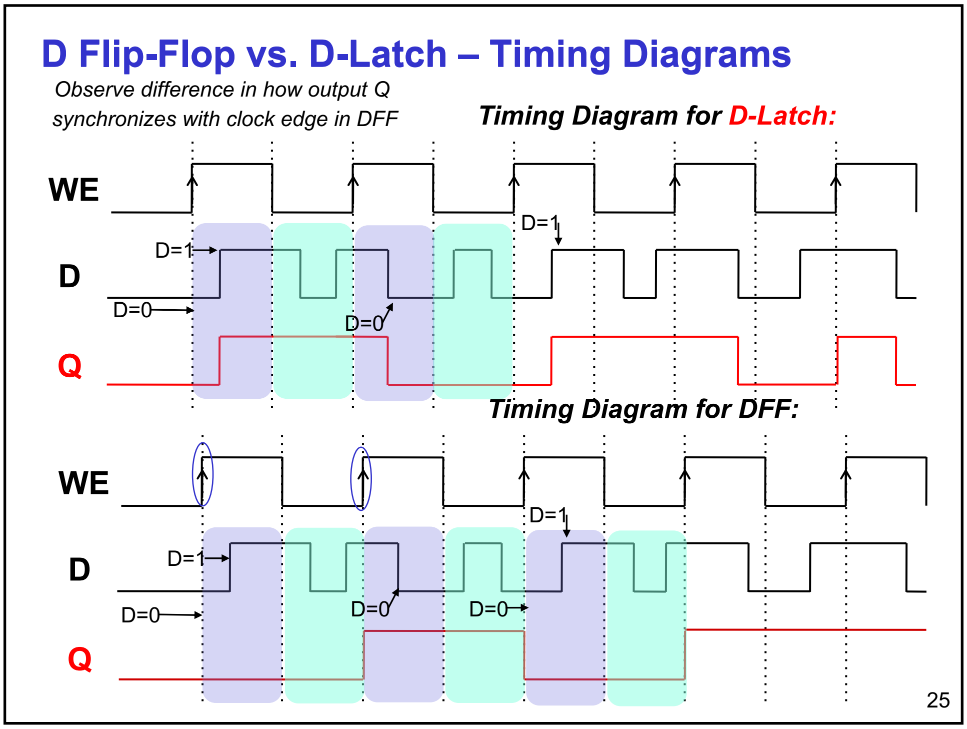


Enter the Flip Flop….

- Flip flops are edge triggered devices
  - They capture the input and change state when the clock changes level
    - Positive edge triggered: when clock goes from 0 to 1
    - Negative edge triggered: when clock goes from 1 to 0

- We refer to the D flip flop as an edge-triggered device.
  - D=Q ONLY when WE changes from 0 to 1

- This differs from D latch, which is: level-triggered
  - D=Q anytime WE equals 1

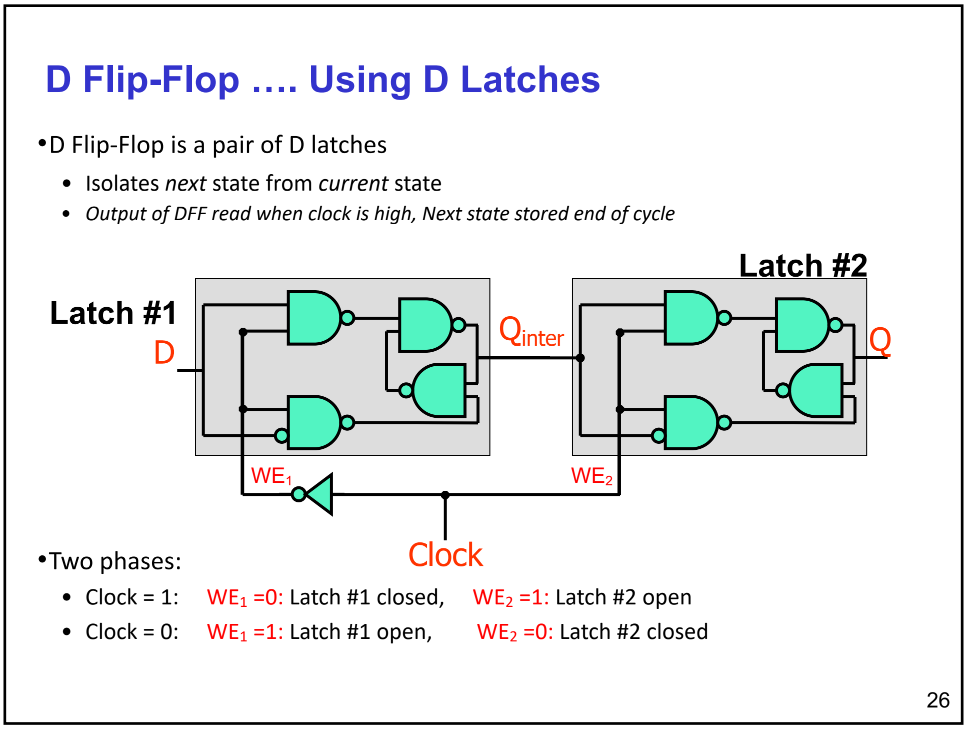


D Flip-Flop vs. D-Latch – Timing Diagrams

Observe difference in how output Q synchronizes with clock edge in DFF

Timing Diagram for D-Latch:

Timing Diagram for DFF:

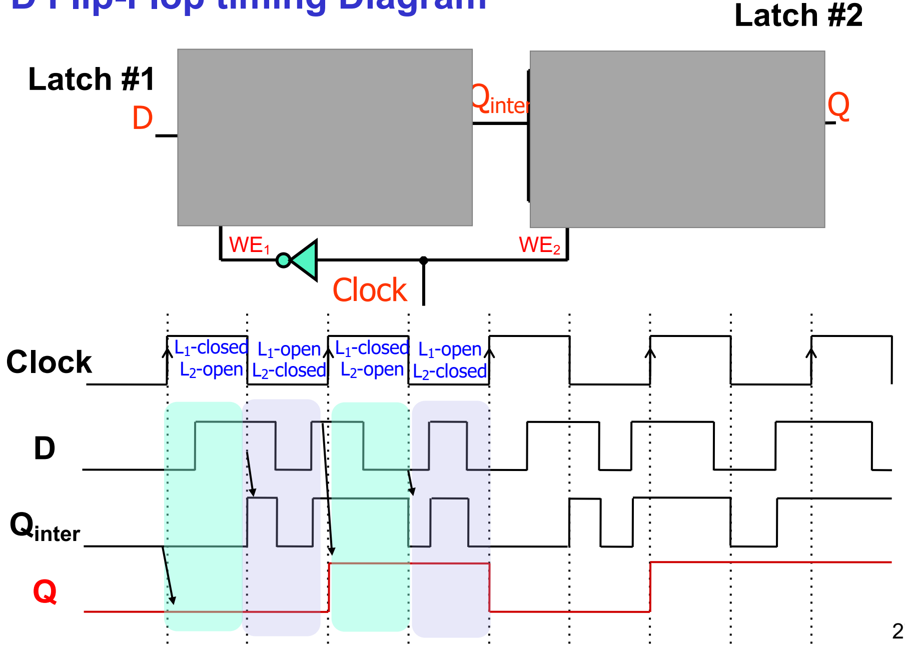


D Flip-Flop …. Using D Latches

- D Flip-Flop is a pair of D latches
  - Isolates *next* state from *current* state
  - *Output of DFF read when clock is high, Next state stored end of cycle*

- Two phases:
  - Clock = 1: WE_1 =0: Latch #1 closed, WE_2 =1: Latch #2 open
  - Clock = 0: WE_1 =1: Latch #1 open, WE_2 =0: Latch #2 closed

D Flip-Flop timing Diagram

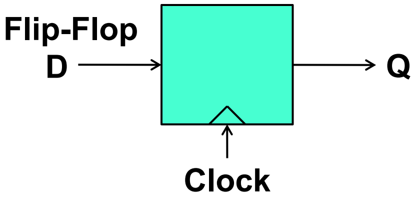


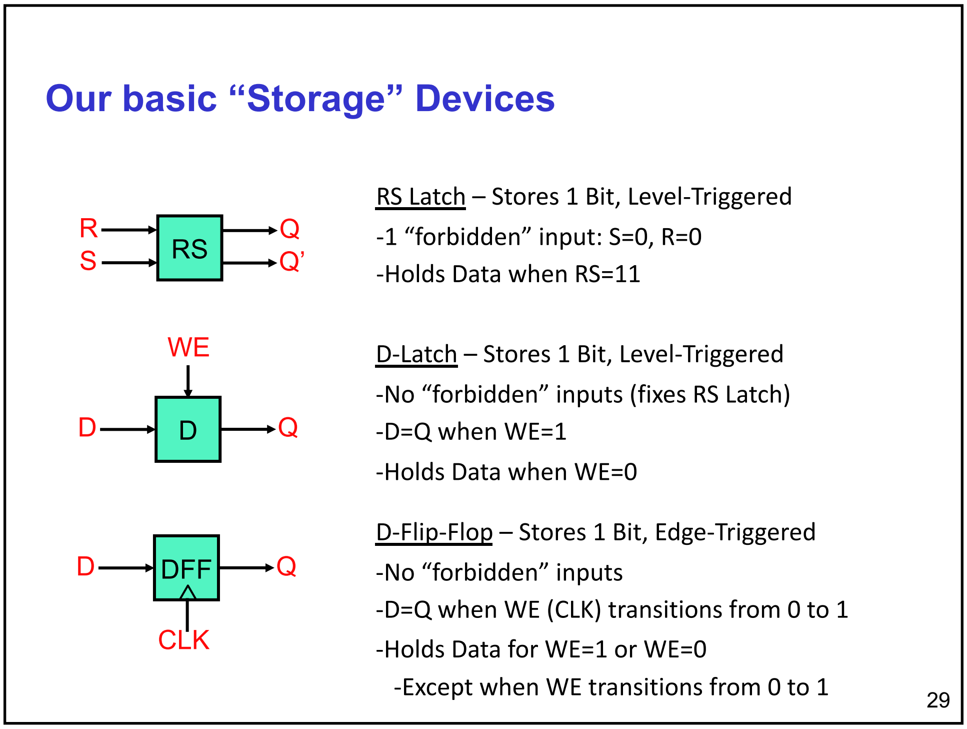
D Flip-Flop

- We can think of the D Flip-Flop as a 1 bit storage container with an input, D, and an output, Q and connected to a clock input
- A set of D flip-flops can be grouped together with common Clock and WE inputs to form a register -- replace D-latch with D flip flop

Truth table for DFF:
X = don't care (i.e, 0 or 1)

D	Clk	Q
0	0 ↑ 1	0
1	0 ↑ 1	1
X	0	Last Q
X	1	Last Q





Our basic "Storage" Devices

RS Latch – Stores 1 Bit, Level-Triggered

-1 "forbidden" input: S=0, R=0

-Holds Data when RS=11

D-Latch – Stores 1 Bit, Level-Triggered

-No "forbidden" inputs (fixes RS Latch)

-D=Q when WE=1

-Holds Data when WE=0

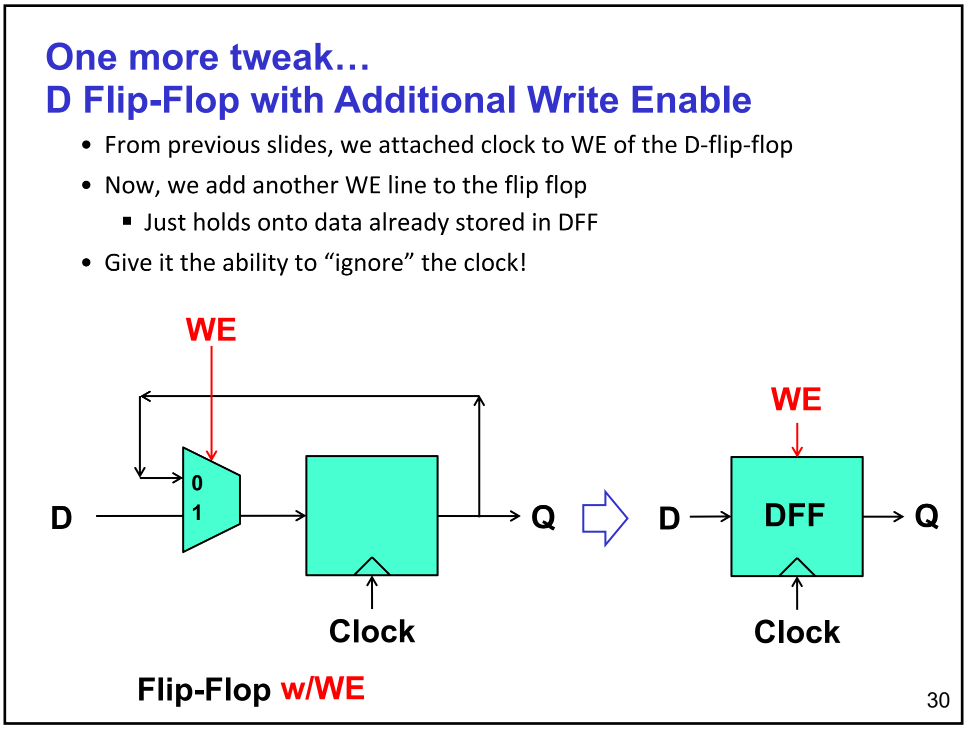
D-Flip-Flop – Stores 1 Bit, Edge-Triggered

-No "forbidden" inputs

-D=Q when WE (CLK) transitions from 0 to 1

-Holds Data for WE=1 or WE=0

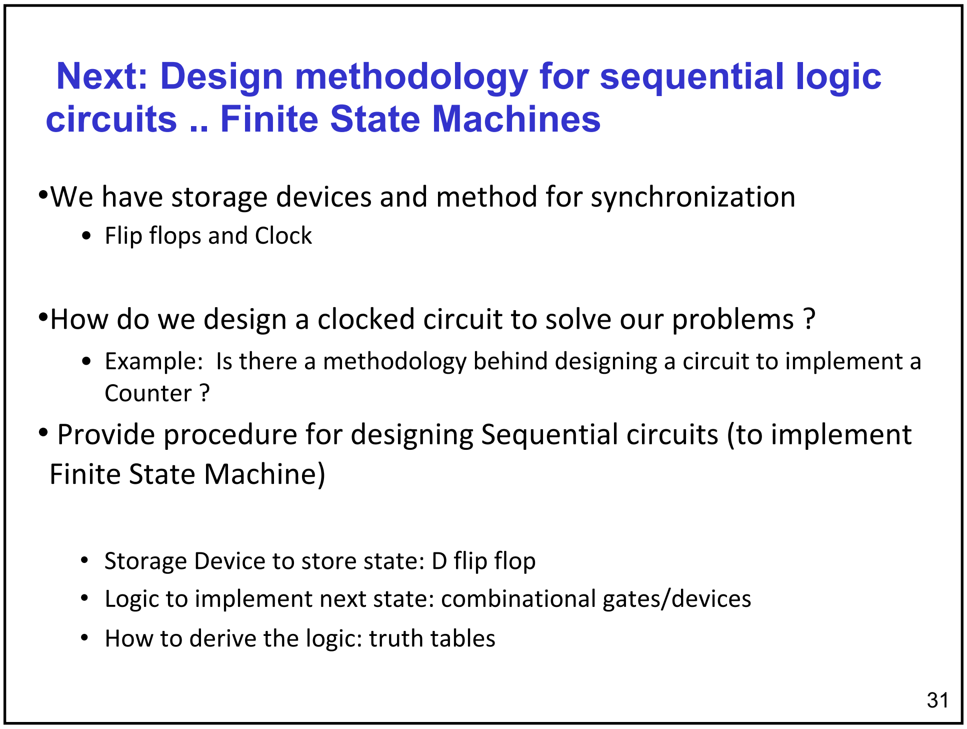
-Except when WE transitions from 0 to 1



One more tweak…
D Flip-Flop with Additional Write Enable

- From previous slides, we attached clock to WE of the D-flip-flop
- Now, we add another WE line to the flip flop
  - Just holds onto data already stored in DFF
- Give it the ability to "ignore" the clock!

Flip-Flop w/WE

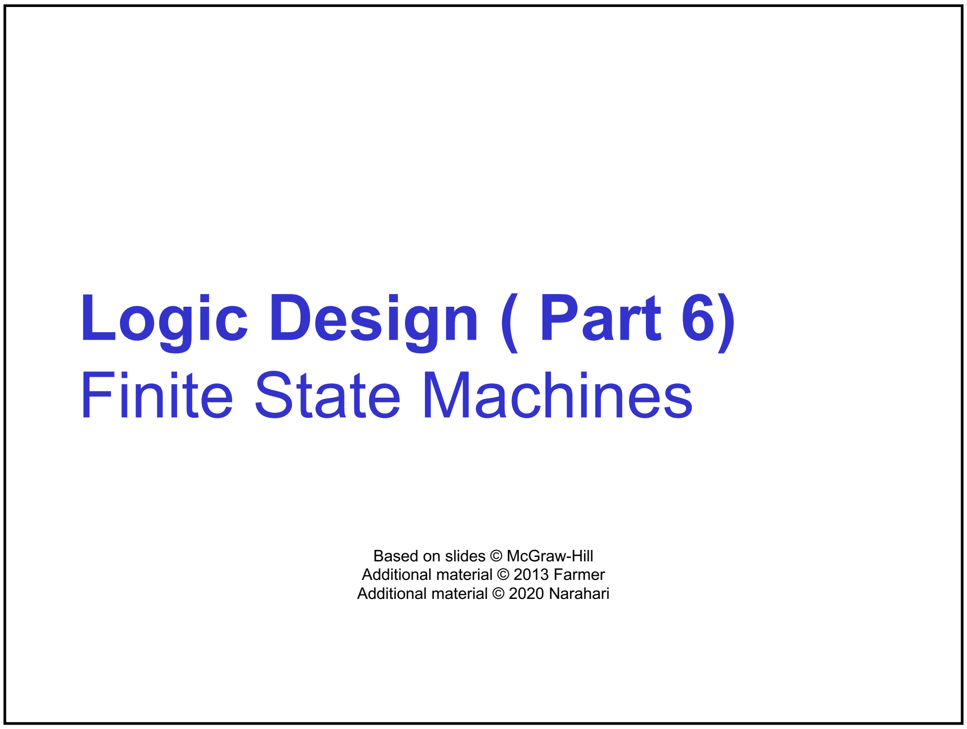


Next: Design methodology for sequential logic circuits .. Finite State Machines

- We have storage devices and method for synchronization
  - Flip flops and Clock
- How do we design a clocked circuit to solve our problems ?
  - Example: Is there a methodology behind designing a circuit to implement a Counter ?
- Provide procedure for designing Sequential circuits (to implement Finite State Machine)
  - Storage Device to store state: D flip flop
  - Logic to implement next state: combinational gates/devices
  - How to derive the logic: truth tables

31

31

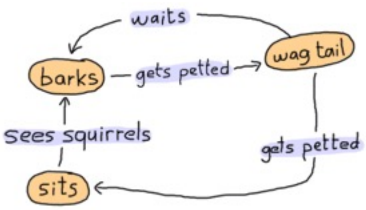


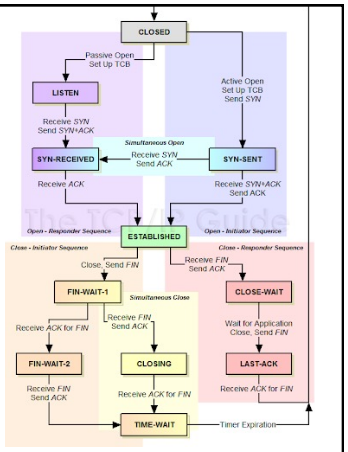
Logic Design (Part 6)

Finite State Machines

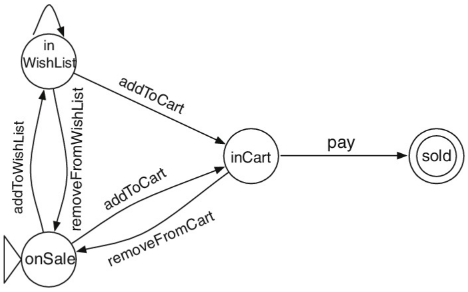
Based on slides © McGraw-Hill
Additional material © 2013 Farmer
Additional material © 2020 Narahari

32





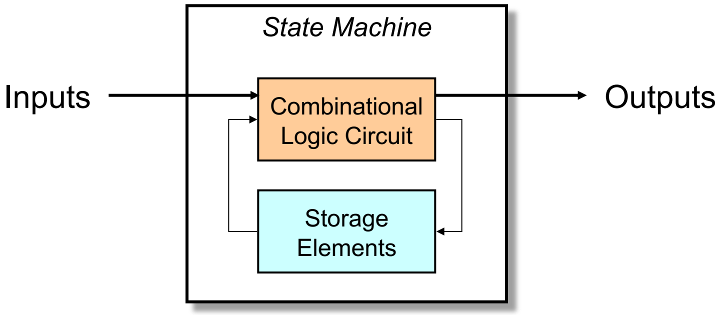
Finite State Machines



Sequential Circuit Schematic & State Machine

•sequential circuit

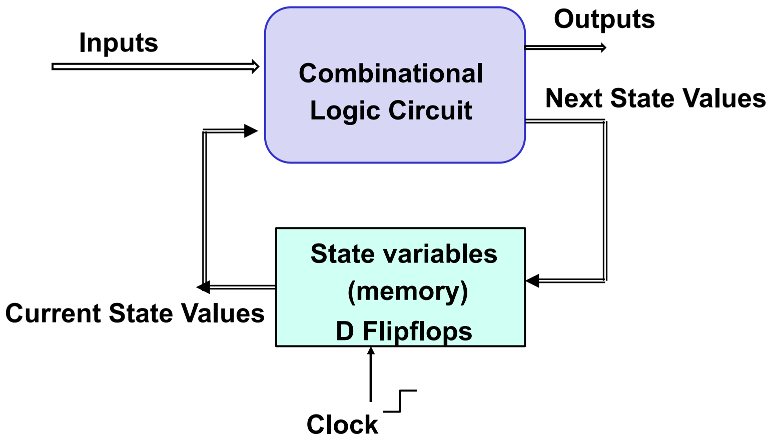
- Output depends on input and stored information (state)
- Combines combinational logic with storage
- “Remembers” state, and changes output (and state) based on inputs and current state



1. combinational circuit computes output and next state

2. storage elements store current state

Schematic of a Sequential Circuit



Inputs

Combinational Logic Circuit

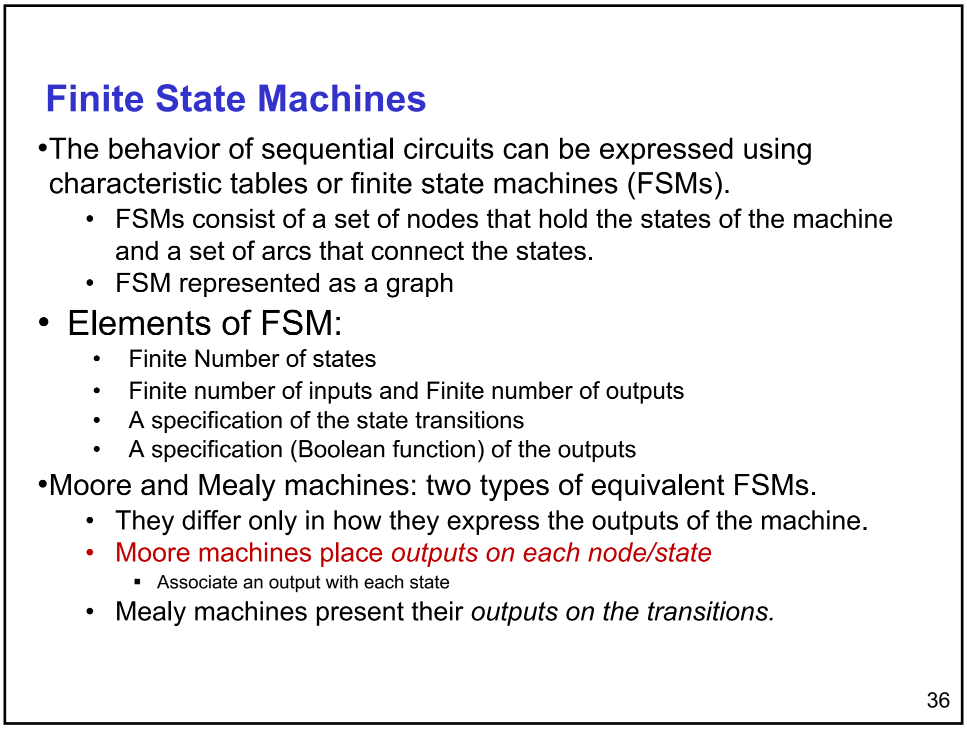
Outputs

Next State Values

State variables (memory) D Flipflops

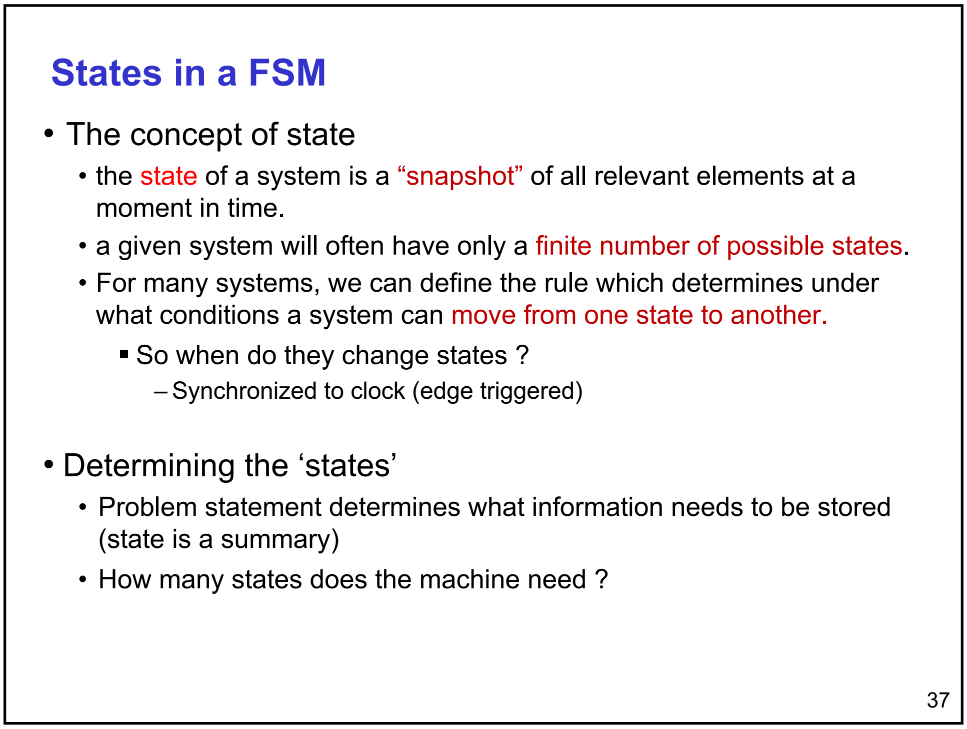
Current State Values

Clock



Finite State Machines

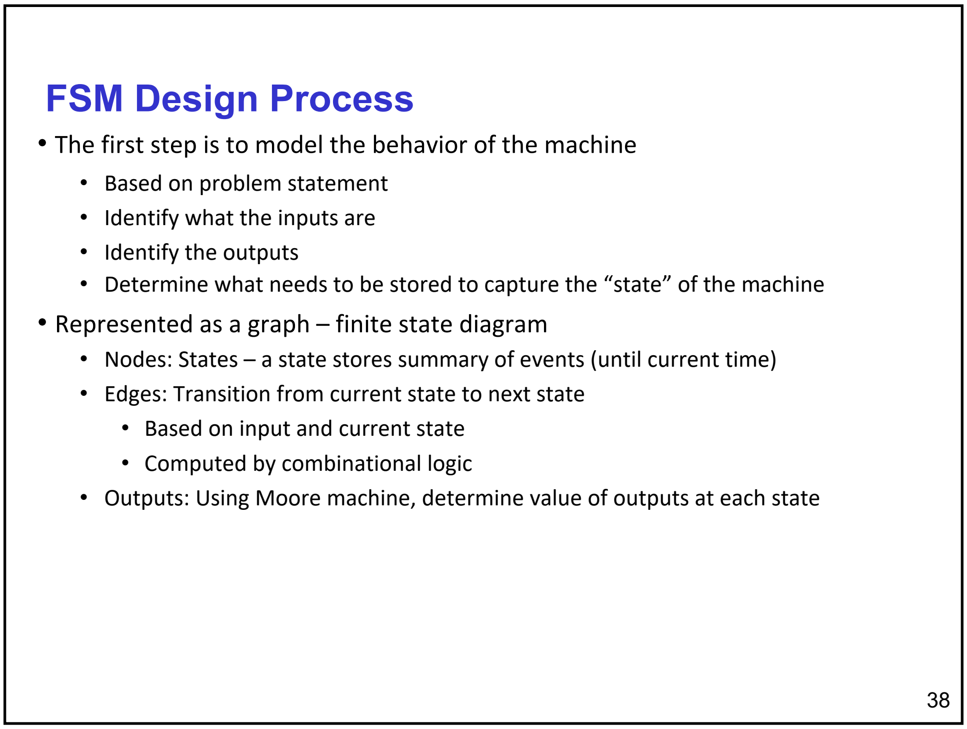
- The behavior of sequential circuits can be expressed using characteristic tables or finite state machines (FSMs).
  - FSMs consist of a set of nodes that hold the states of the machine and a set of arcs that connect the states.
  - FSM represented as a graph
- Elements of FSM:
  - Finite Number of states
  - Finite number of inputs and Finite number of outputs
  - A specification of the state transitions
  - A specification (Boolean function) of the outputs
- Moore and Mealy machines: two types of equivalent FSMs.
  - They differ only in how they express the outputs of the machine.
  - Moore machines place *outputs on each node/state*
    - Associate an output with each state
  - Mealy machines present their *outputs on the transitions.*



States in a FSM

- The concept of state
  - the state of a system is a "snapshot" of all relevant elements at a moment in time.
  - a given system will often have only a finite number of possible states.
  - For many systems, we can define the rule which determines under what conditions a system can move from one state to another.
    - So when do they change states ?
      - Synchronized to clock (edge triggered)
- Determining the 'states'
  - Problem statement determines what information needs to be stored (state is a summary)
  - How many states does the machine need ?

37



FSM Design Process

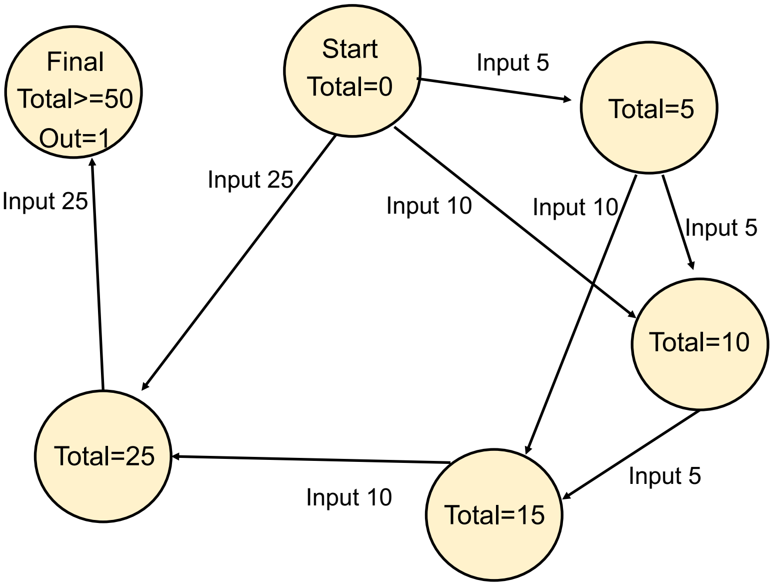
- The first step is to model the behavior of the machine
  - Based on problem statement
  - Identify what the inputs are
  - Identify the outputs
  - Determine what needs to be stored to capture the "state" of the machine
- Represented as a graph – finite state diagram
  - Nodes: States – a state stores summary of events (until current time)
  - Edges: Transition from current state to next state
    - Based on input and current state
    - Computed by combinational logic
  - Outputs: Using Moore machine, determine value of outputs at each state

38

Example: A Vending Machine

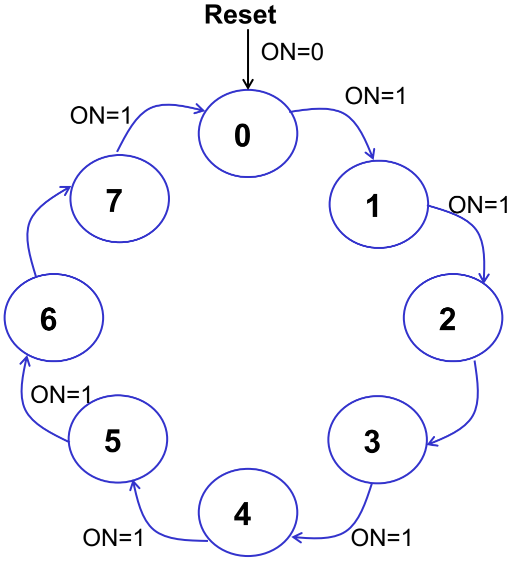
- Accept user input (coins), when total is at least 50 cents dispense output
  - For this example, we will not model the change to be returned, and only care if the input is at least 50 cents
- Input valid coins:
  - Q (25cents) D (10) or N (5)
- What should it keep track of ?
  - current total
  - Is it 50 cents or more ?
- When it reaches 50 or more:
  - Generate output – dispense the candy/snacks
- States of the machine ?
  - What should each state capture ?
  - How many states ?

Finite State Machine for a Vending Machine



Design a Counter: counts from 0 to 7

Finite State Machine Representation of Counter: Counter to count from 0 to 7 (while switch is ON)



Bubbles represent all possible “states” for the machine

Arrows show movement from one state to the next while switch ON=1.

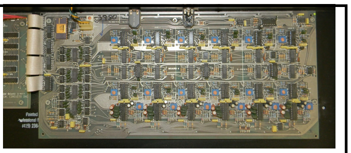
If ON=0, FSM goes to state 0

Transitions occur at pulse of the clock

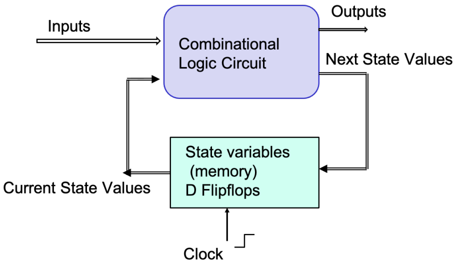
Designing and implementing a FSM

1. Understand the problem statement and determine inputs/outputs
2. Identify states and draw the state diagram
3. Next, derive the truth table (from state diagram)
4. From truth table, implement combinational circuit (boolean function) for each of the next state values & outputs
   - States stored in your N storage elements – Flip Flops

FSM Design & Implementation: Sequential Circuit Design



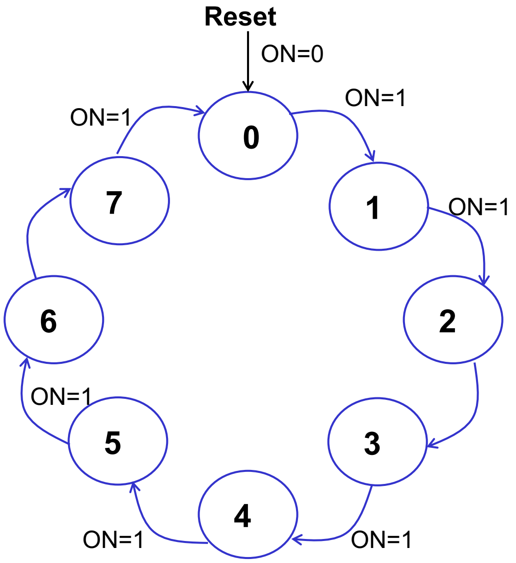
- The state diagram describes the behavior of the machine
  - Can determine number of storage elements (Flip Flops) that we need.
  - States are numbered in binary
- To implement the circuit: derive truth table from state diagram and then design the circuit (combinational logic)


Inputs
Outputs
Combinational Logic Circuit
Next State Values
State variables (memory) D Flipflops
Current State Values
Clock

Designing and implementing a FSM

1. Understand the problem statement and determine inputs/outputs
2. Identify states and draw the state diagram
   - Encode each state in binary using N bits
   - State diagram will show transitions from state to state based on value of inputs
3. Next, derive the truth table (from state diagram)
   - "inputs" in truth table are N current state variables and the inputs
   - These N state variables will need to be stored in N flips flops,
   - Label the N state variables $S_{N-1}\,S_{N-2}\ldots S_1S_0$
   - "outputs" are the values of the state variables in the next state and the output at each state -- common notation is S' but confusion with complement operator, so let's use S*
4. From truth table, implement combinational circuit (boolean function) for each of the next state values & outputs
   - State variables are stored in your N storage elements

Finite State Machine Representation of Counter: Counter to count from 0 to 7 (while switch is ON)

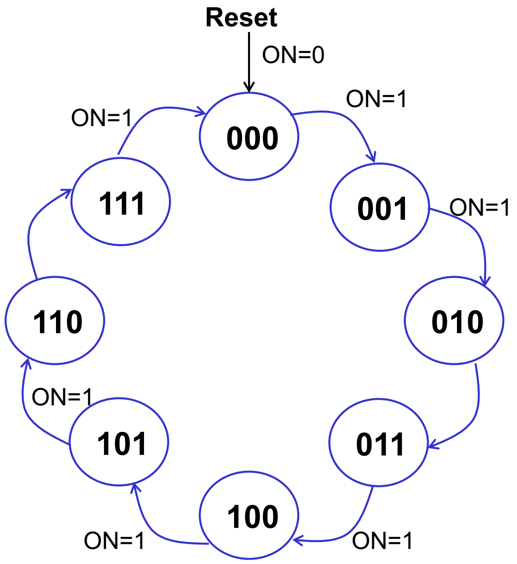


Bubbles represent all possible "states" for the machine

Arrows show movement from one state to the next while switch ON=1.
If ON=0, FSM goes to state 0

Transitions occur at pulse of the clock

Labelling States in Binary



We have 8 states, therefore we need $\log_2 8 = 3$ bits to encode states in binary:

from 000 to 111

Each bit is called a state variable

& Stored in one D flip flop

we need 3 Flip flops to store 3 bits $S_2 S_1 S_0$

Storage

- Each D flip flop stores one state bit.
- The number of storage elements (flip-flops) needed is determined by the number of states
(and the representation of each state).
  - Each bit can be 0 or 1 = 2 states
  - N bits can represent 2^N states

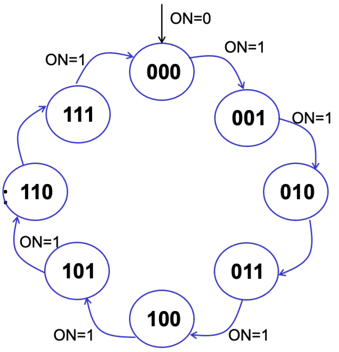
- Example: If FSM has 12 states, then circuit needs $\log_2 12 = 4$ storage elements (i.e., flip flops).
  - Fewer the states, less hardware needed
    - *Concept of Minimization of States for a given FSM…*
      - *…in Foundations (where else!!)*

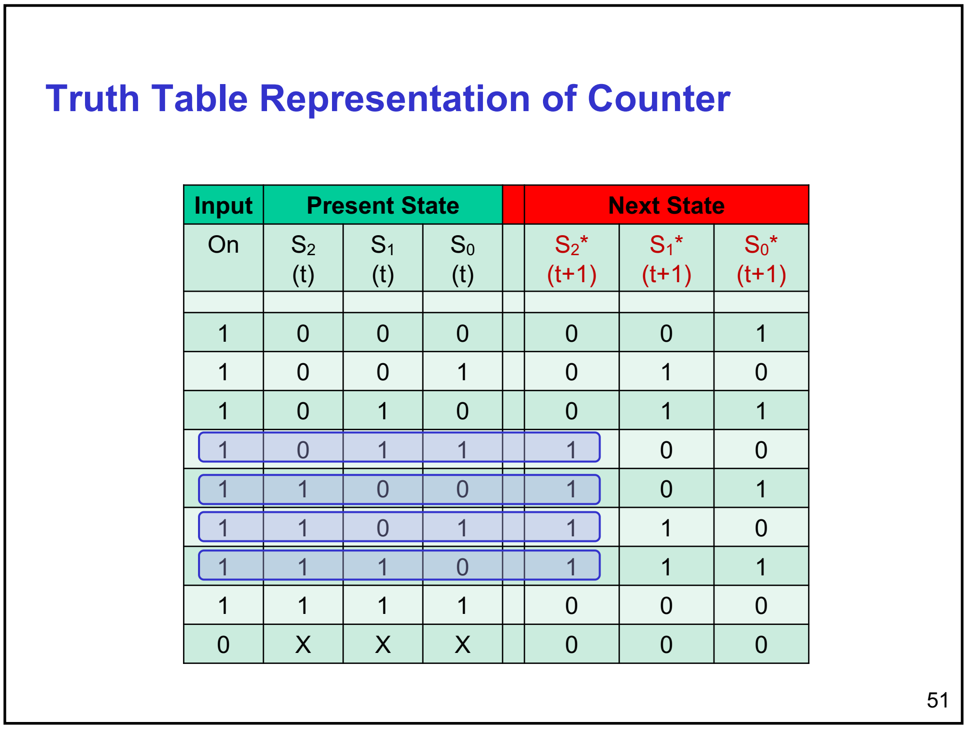
Designing and implementing a FSM

1. Understand the problem statement and determine inputs/outputs
2. Identify states and draw the state diagram
   - Encode each state in binary using N bits
   - State diagram will show transitions from state to state based on value of inputs
3. Next, derive the truth table (from state diagram)
   - “inputs” in truth table are N current state variables and the inputs
   - These N state variables will need to be stored in N flips flops,
   - Label the N state variables $S_{N-1}\,S_{N-2}\ldots S_1S_0$
   - “outputs” are the values of the state variables in the next state and the output at each state -- common notation is S’ but confusion with complement operator, so let’s use S*
4. From truth table, implement combinational circuit (boolean function) for each of the next state values & outputs
   - State variables are stored in your N storage elements

Truth Table…and next state values (S*)

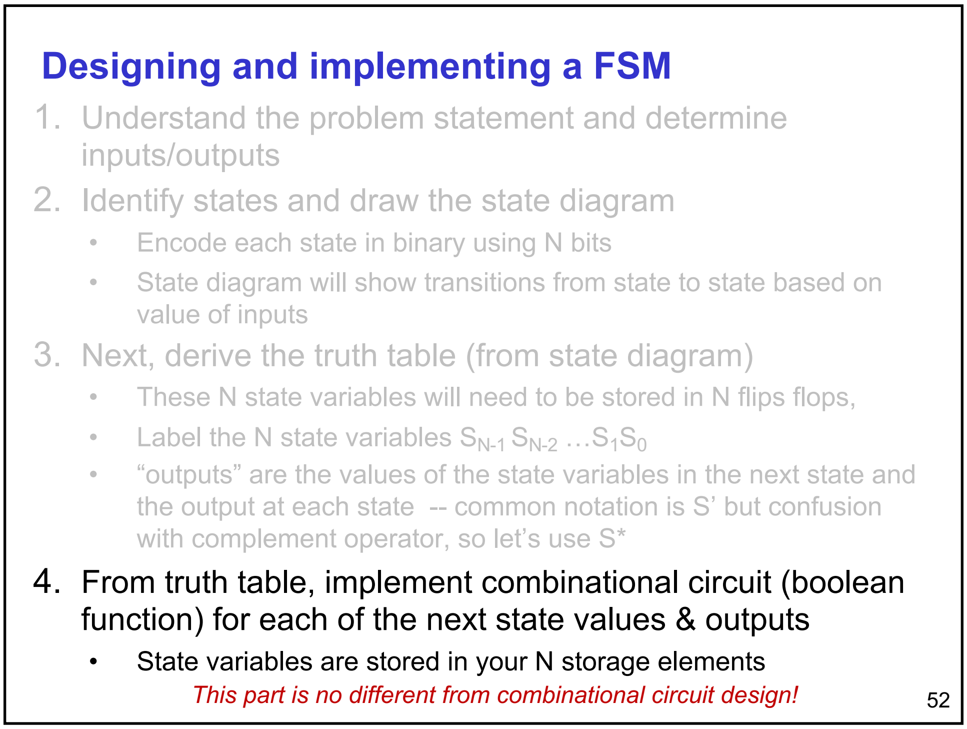
- Three bits to store state, state variables $S_2\,S_1\,S_0$
- If FSM is currently in state 011, and ON=1, what is the next state ?
- next state= 100
- In terms of the state variables/flip flops:
- If *ON=1*, $S_2=1\ S_1=0\ S_0=0$ then
- $S_2^*=1\ S_1^*=0\ S_0^*=0$
- Note: In this example the output (of counter) is simply the values of the state
  - In general you can have an output associate with a state





Truth Table Representation of Counter

Input	Present State				Next State		
On	S_2 (t)	S_1 (t)	S_0 (t)		S_2* (t+1)	S_1* (t+1)	S_0* (t+1)
1	0	0	0		0	0	1
1	0	0	1		0	1	0
1	0	1	0		0	1	1
1	0	1	1		1	0	0
1	1	0	0		1	0	1
1	1	0	1		1	1	0
1	1	1	0		1	1	1
1	1	1	1		0	0	0
0	X	X	X		0	0	0



Designing and implementing a FSM

1. Understand the problem statement and determine inputs/outputs
2. Identify states and draw the state diagram
   - Encode each state in binary using N bits
   - State diagram will show transitions from state to state based on value of inputs
3. Next, derive the truth table (from state diagram)
   - These N state variables will need to be stored in N flips flops,
   - Label the N state variables S_{N-1} S_{N-2} …S_1S_0
   - “outputs” are the values of the state variables in the next state and the output at each state -- common notation is S’ but confusion with complement operator, so let’s use S*
4. From truth table, implement combinational circuit (boolean function) for each of the next state values & outputs
   - State variables are stored in your N storage elements

This part is no different from combinational circuit design!

Designing and implementing a FSM

1. Understand the problem statement and determine inputs/outputs
2. Identify states and draw the state diagram
   - Encode each state in binary using N bits
   - State diagram will show transitions from state to state based on value of inputs
3. Next, derive the truth table (from state diagram)
   - These N state variables will need to be stored in N flips flops,
   - Label the N state variables $S_{N-1} S_{N-2} \ldots S_1 S_0$
   - "outputs" are the values of the state variables in the next state and the output at each state -- common notation is S' but confusion with complement operator, so let's use S*
4. From truth table, implement combinational circuit (boolean function) for each of the next state values & outputs
   - State variables are stored in your N storage elements

This part is no different from combinational circuit design!

53

53

Boolean functions for values of next state

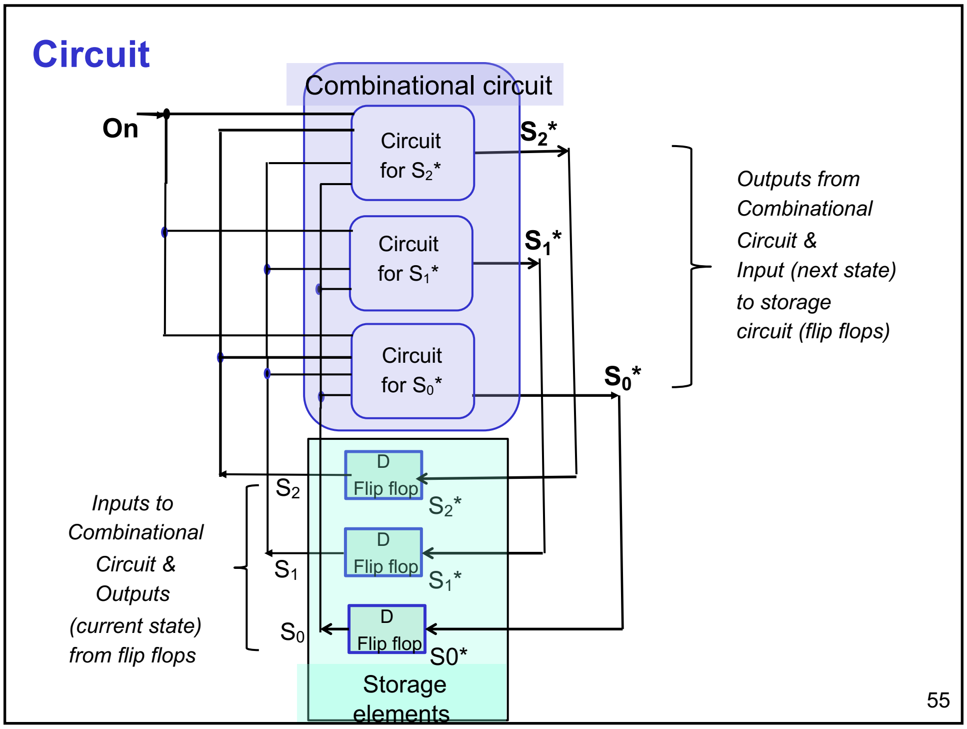
- For each state variable, derive function that determines next state for that variable:
- Note: next state value for each state variable denoted as S*
- Current states: S_2 S_1 S_0
- Derivation here is only for case when On=1

- $S_2^* = On.(S_2' S_1S_0 + S_2S_1'S_0' + S_2S_1'S_0 + S_2S_1S_0')$
  - Optimized: $S_2^* = On.(S_2S_1' + S_2S_1S_0' + S_2'S_1S_0)$
- $S_1^* = On.(S_2' S_1'S_0 + S_2'S_1S_0' + S_2S_1'S_0 + S_2S_1S_0')$
  - Optimized: $S_1^* = On. (S_1S_0' + S_1'S_0)$

- $S_0^* = On.(S_2' S_1'S_0' + S_2'S_1S_0' + S_2S_1'S_0' + S_2S_1S_0')$
  - Optimized: $S_0^* = On.(S_1S_0' + S_2S_1'S_0 + S_2'S_1'S_0')$

54

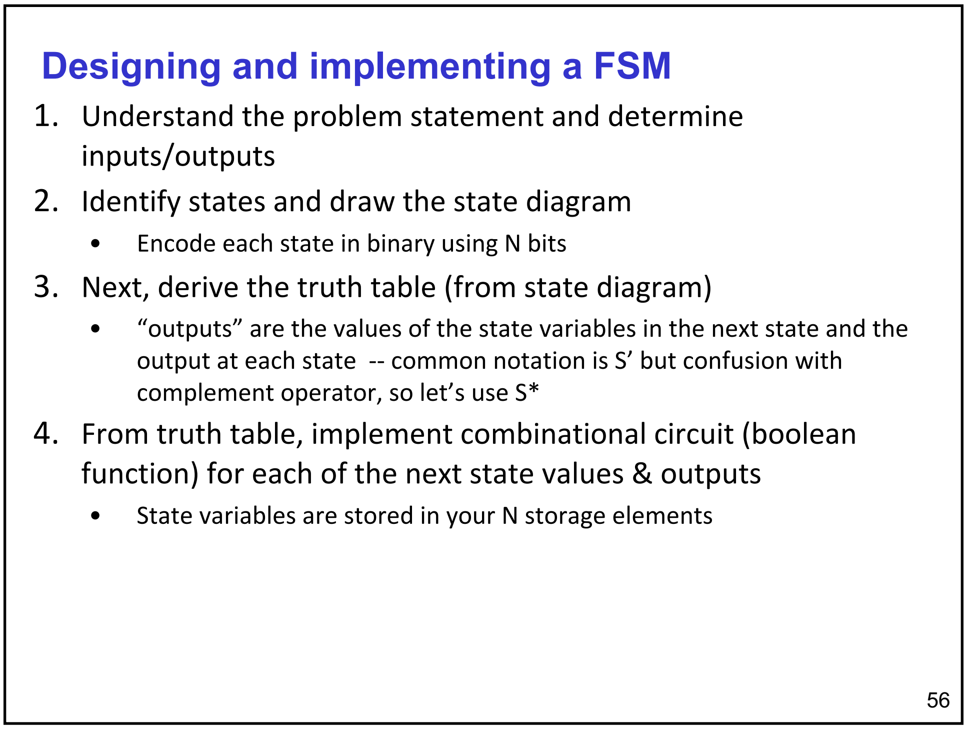
54



Circuit

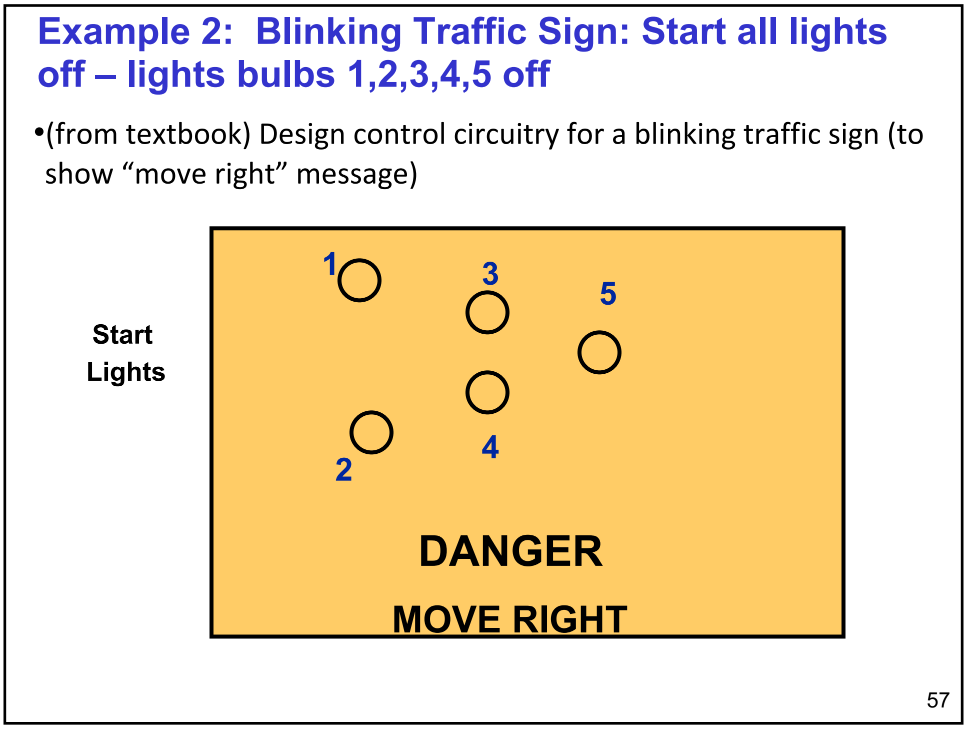
Outputs from Combinational Circuit & Input (next state) to storage circuit (flip flops)

Inputs to Combinational Circuit & Outputs (current state) from flip flops



Designing and implementing a FSM

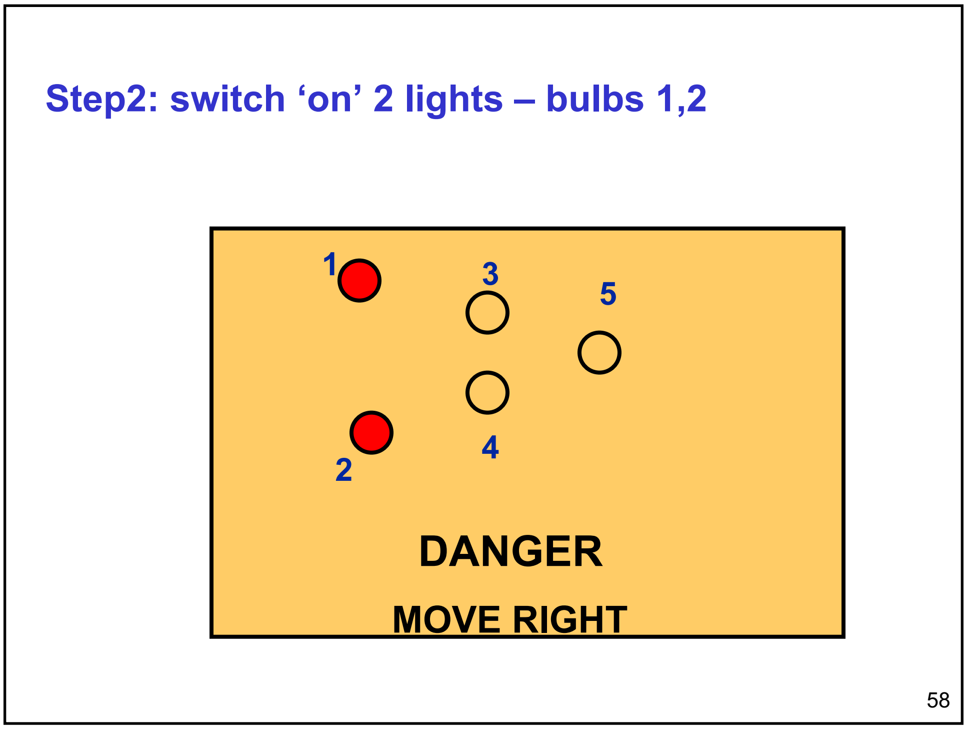
1. Understand the problem statement and determine inputs/outputs
2. Identify states and draw the state diagram
   - Encode each state in binary using N bits
3. Next, derive the truth table (from state diagram)
   - “outputs” are the values of the state variables in the next state and the output at each state -- common notation is S’ but confusion with complement operator, so let’s use S*
4. From truth table, implement combinational circuit (boolean function) for each of the next state values & outputs
   - State variables are stored in your N storage elements



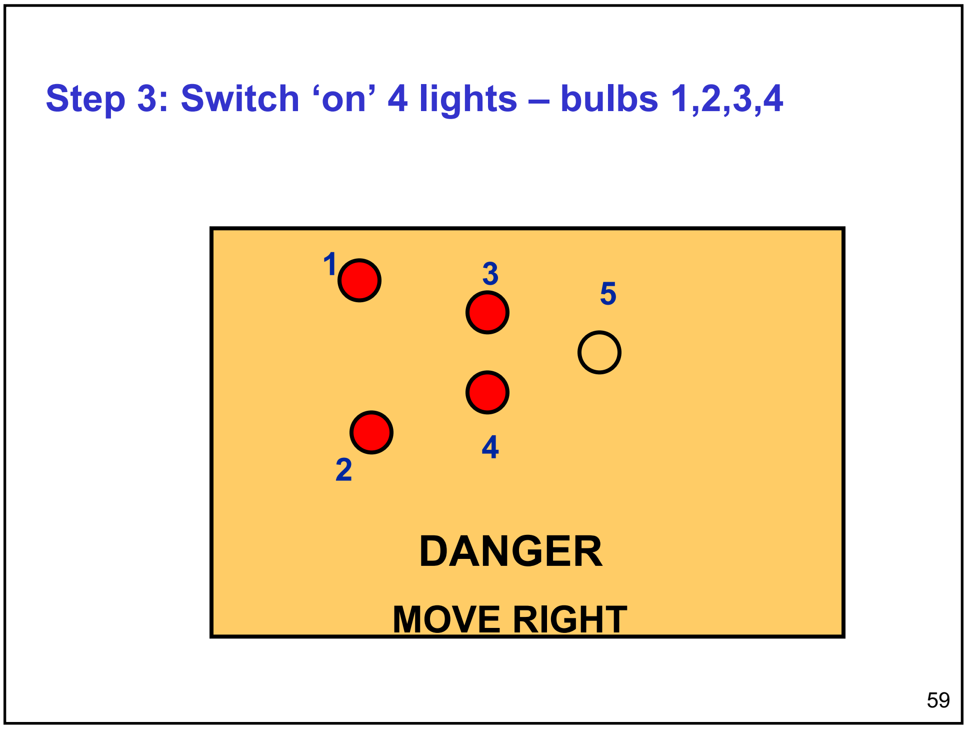
Example 2: Blinking Traffic Sign: Start all lights off – lights bulbs 1,2,3,4,5 off

- (from textbook) Design control circuitry for a blinking traffic sign (to show “move right” message)

Start Lights



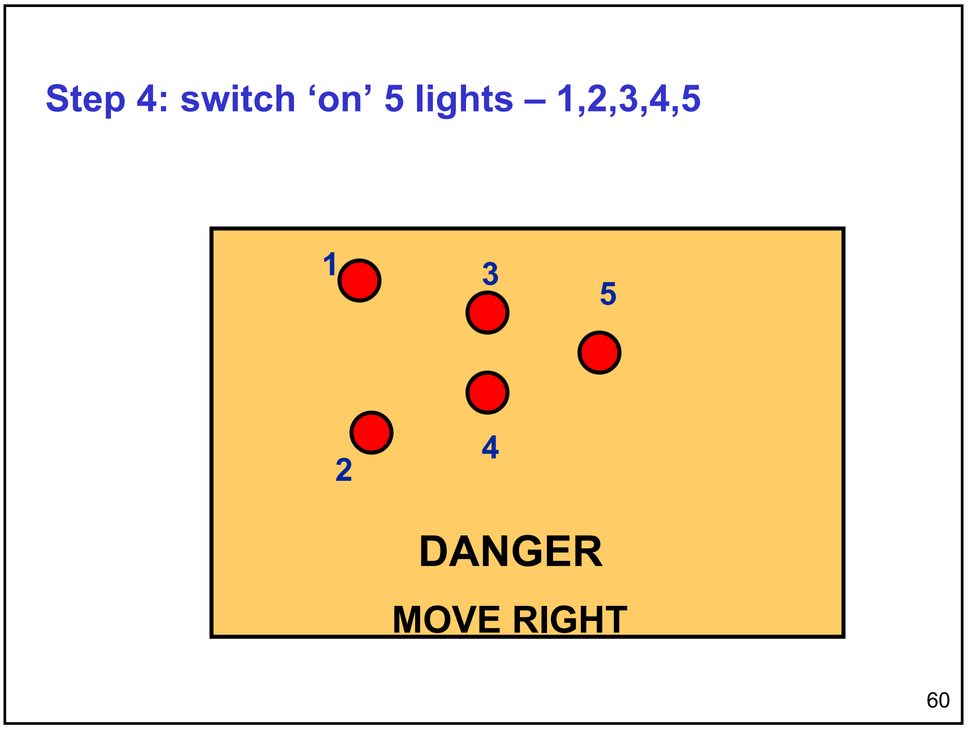
Step2: switch ‘on’ 2 lights – bulbs 1,2



Step 3: Switch ‘on’ 4 lights – bulbs 1,2,3,4

59

59

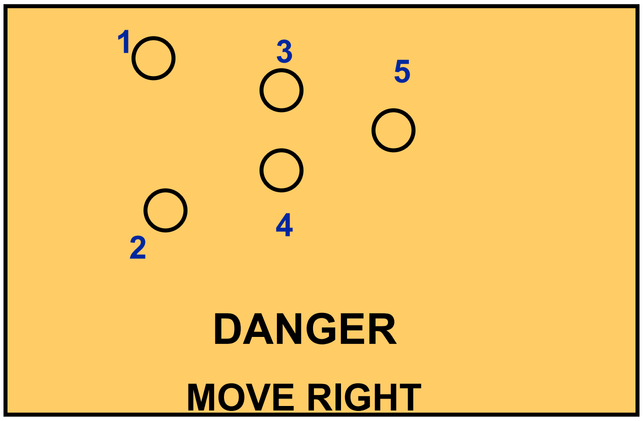


Step 4: switch ‘on’ 5 lights – 1,2,3,4,5

60

60

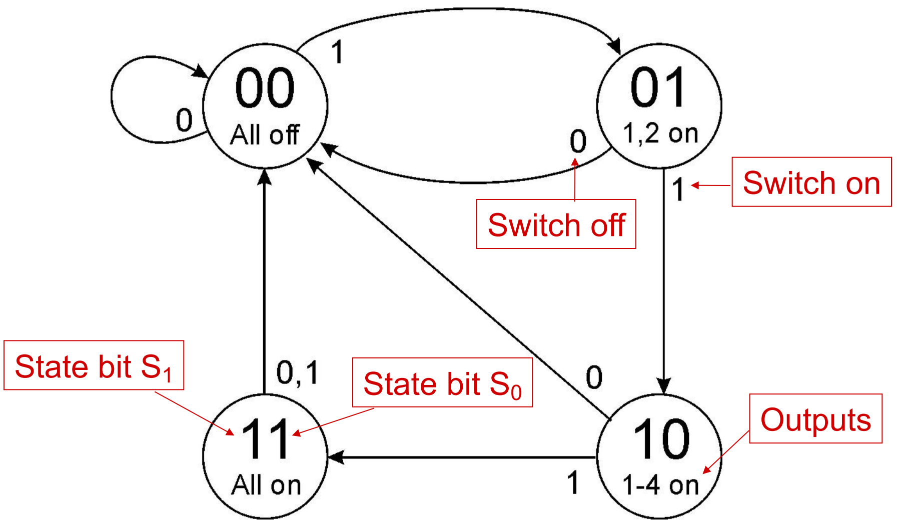
Step 5- switch all ‘off’ – lights bulbs 1,2,3,4,5 off And repeat the cycle



Example: Traffic Sign

- A blinking traffic sign: How many lights/lightbulbs = 5
- How many states ?
- 4 states
  - No lights on
  - 1 & 2 on
  - 1, 2, 3, & 4 on
  - 1, 2, 3, 4, & 5 on
  - (repeat as long as switch is turned on)
- How many bits to represent the 4 states
- S_1S_0
  - With S_1S_0 values: 00, 01, 10, 11
- How many ‘outputs’ (to control the 5 lights) = 5 ?
- If the sign is switched off then all lights turn off

Traffic Sign State Diagram



Switch on

Switch off

State bit S_1

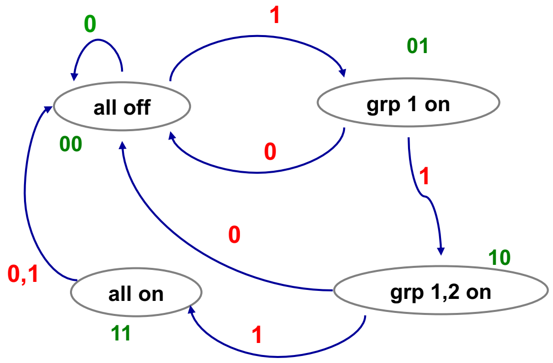
State bit S_0

Outputs

Transition on each clock cycle.

Outputs

- Note we really have 3 groups of lights to be controlled = 3 control lines X,Y,Z
  - Group 1: Lights 1 and 2; controlled by Z
    - If Z=1 then Group 1 lights (1 and 2) are switched on
  - Group 2: lights 3 & 4; controlled by Y
  - Group 3: Light 5; controlled by X
- In this example, we associate each state with an output
  - Depending on the current state, we switch on specific groups of lights



- When is group 1 on?
  - in states 01, 10 and 11 - but only when the switch IN is on!
- Logic expressions for X,Y,Z
  - Depends on S_0 and S_1 and Input is on
    - If Input is off then X,Y,Z are al 0
- can you come up with a logic expression for next state values of S_0 and S_1?
  - Depends on current values of S_0 and S_1 and Input is on
    - Input off then both bits are set to 0 since next state is 00
- When do we switch to the next state?
  - the two bits of S[1:0] are updated at every clock cycle

Traffic Sign Truth Tables

Outputs
(depend only on state: S_1S_0)

Lights 1 and 2

Lights 3 and 4

Light 5

S_1	S_0	Z	Y	X
0	0	0	0	0
0	1	1	0	0
1	0	1	1	0
1	1	1	1	1

Next State: $S_1'S_0'$
(depend on state and input)

Switch

In	S_1	S_0	S_1^*	S_0^*
0	X	X	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Whenever In=0, next state is 00.

Boolean functions for light control bits

- from truth table, consider all rows where outputs =1

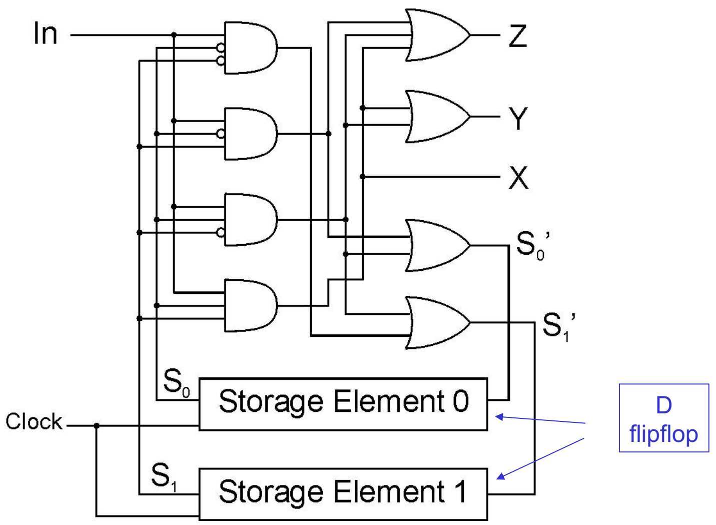
- $Z = ((\text{NOT } S_1)S_0 + S_1 (\text{NOT } S_0) + S_1S_0).\text{In}$

- $Y = (S_1S_0 + S_1 (\text{NOT } S_0)).\text{In}$

- $X = (S_1S_0).\text{In}$

- $S_1^* = (\text{In}.\ S_1.S_0') + (\text{In}.\ S_1'.\ S_0)$
- $S_0^* = (\text{In}.\ S_1'.S_0') + (\text{In}.\ S_1.S_0')$

Traffic Sign Logic



D flipflop

Summary of Digital Logic

- Combinational logic
  - Basic gates
  - Combinational devices: adders, decoders, multiplexers,…
- Sequential logic
  - Storage element…Flip flop
  - Theory behind design of finite state machines
    - They act like controllers of a circuit
- Sequential logic devices
  - Memory, ROM, RAM, Registers

So, what is the purpose of all this ?.........

Next…Our Agenda: Putting it all together

- The goal: Turn a theoretical device - Turing's Universal Computational Machine - into an actual computer ...
  - … interacting with data and instructions from the outside world, and producing output data.
- Smart building blocks:
  - We have at our disposal a powerful collection of combinational and sequential logic devices.
- Now we need a master plan ...
  - We need to start with defining the model of a computer architecture
    - **Von Neuman model**
  - And we use our building blocks to build a circuit that can perform a sequence of arithmetic operations, i.e., build a simple CPU.

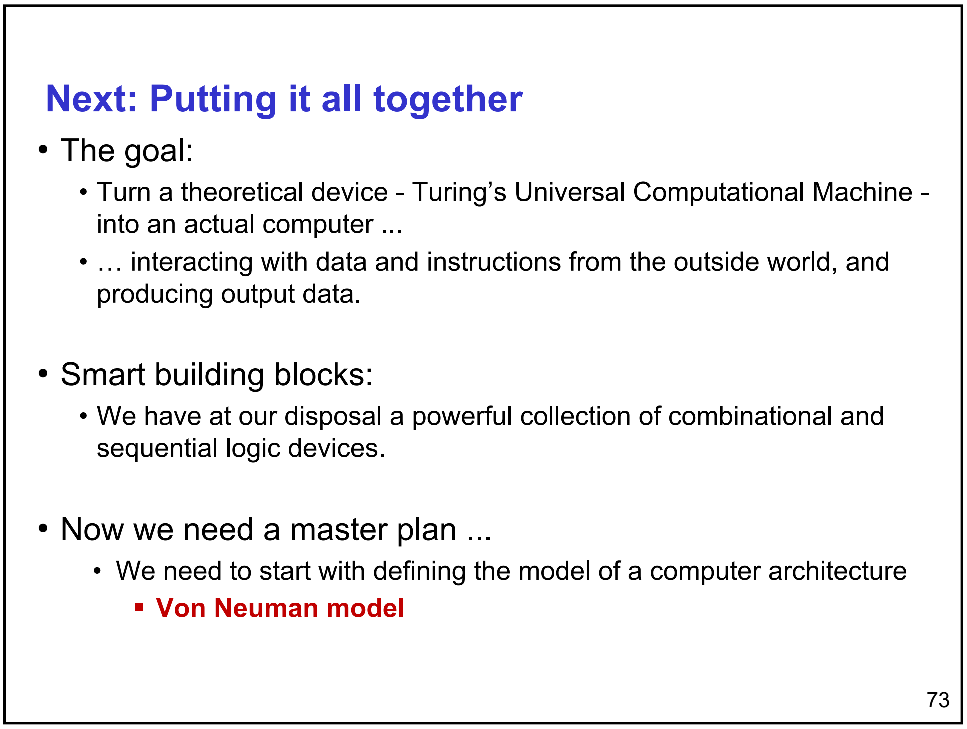
From Logic to Processor Design

•Combinational Logic

- Decoders -- convert instructions into control signals
- Multiplexers -- select inputs and outputs
- ALU (Arithmetic and Logic Unit) -- operations on data

•Sequential Logic

- State machine -- coordinate control signals and data movement
- Registers and latches -- storage elements

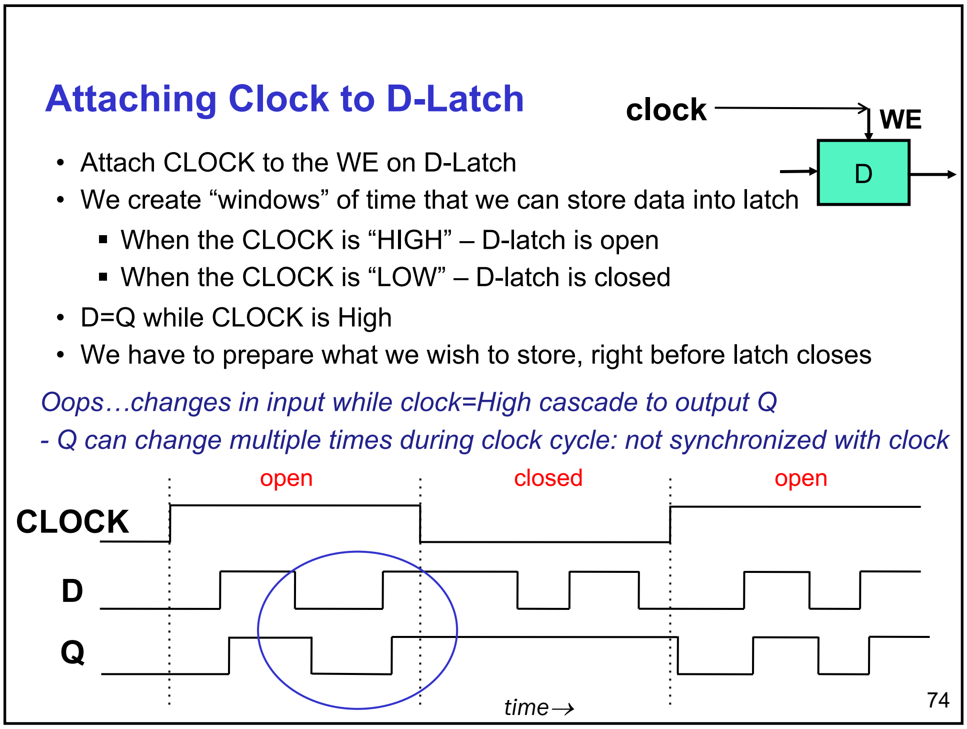


Next: Putting it all together

- The goal:
  - Turn a theoretical device - Turing’s Universal Computational Machine - into an actual computer ...
  - … interacting with data and instructions from the outside world, and producing output data.
- Smart building blocks:
  - We have at our disposal a powerful collection of combinational and sequential logic devices.
- Now we need a master plan ...
  - We need to start with defining the model of a computer architecture
    - **Von Neuman model**

73

73



Attaching Clock to D-Latch

- Attach CLOCK to the WE on D-Latch
- We create “windows” of time that we can store data into latch
  - When the CLOCK is “HIGH” – D-latch is open
  - When the CLOCK is “LOW” – D-latch is closed
- D=Q while CLOCK is High
- We have to prepare what we wish to store, right before latch closes

Oops…changes in input while clock=High cascade to output Q

- Q can change multiple times during clock cycle: not synchronized with clock

74

74