



ECE 545—Digital System Design with VHDL Lecture 3

Sequential Logic Review and Algorithmic
State Machines

9/9/08

1

Outline

- Sequential Logic Building Blocks
 - Latches, Flip-Flops
- Sequential Logic Circuits
 - Registers, Shift Registers, Counters
 - Memory (RAM, ROM)
 - Simple Finite State Machines (Mealy, Moore)
- Complex Digital System Design with ASMs
 - Examples

2

Textbook References

- Sequential Logic Review
 - Stephen Brown and Zvonko Vranesic, *Fundamentals of Digital Logic with VHDL Design, 2nd or 3rd Edition*
 - Chapters 7 and 8
 - **OR** your undergraduate digital logic textbook (chapters on sequential logic and state machines)
- Algorithmic State Machines
 - Stephen Brown and Zvonko Vranesic, *Fundamentals of Digital Logic with VHDL Design, 2nd or 3rd Edition*
 - **Chapter 8.10 Algorithmic State Machine (ASM) Charts**
 - **Chapter 10.2.1 A Bit-Counting Circuit**
 - **Chapter 10.2.2 ASM Chart Implied Timing Information**
 - **Chapter 10.2.6 Sort Operation**
(handouts distributed in class)

3

Sequential Logic Building Blocks

some slides modified from:
Brown and Vranesic, "Fundamentals of Digital Logic with VHDL Design, 2nd Edition"
S. Dandamudi, "Fundamentals of Computer Organization and Design"

4

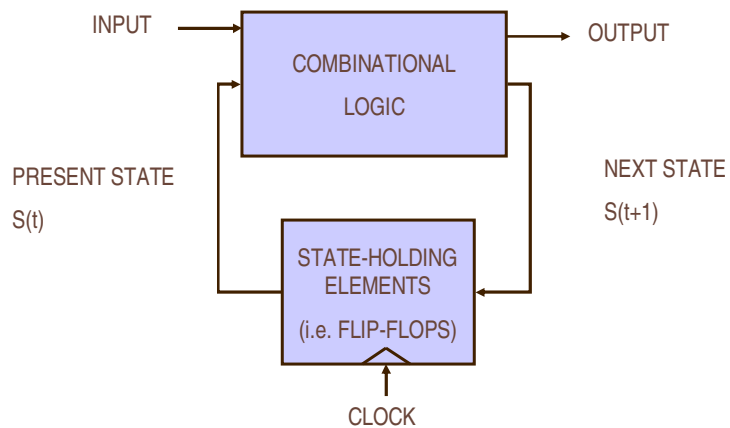
Introduction to Sequential Logic

- Output depends on current as well as past inputs
 - Depends on the history
 - Have “memory” property
- Sequential circuit consists of
 - Combinational circuit
 - Feedback circuit
- Past input is encoded into a set of state variables
 - Uses feedback (to feed the state variables)
 - Simple feedback
 - Uses flip flops

5

Introduction (cont'd)

Main components of a typical synchronous sequential circuit
(synchronous = uses a clock to keep circuits in lock step)



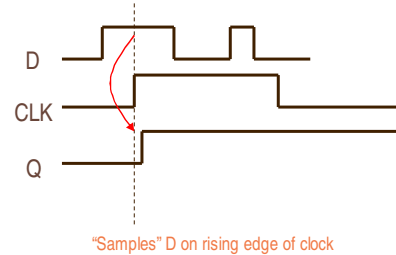
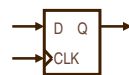
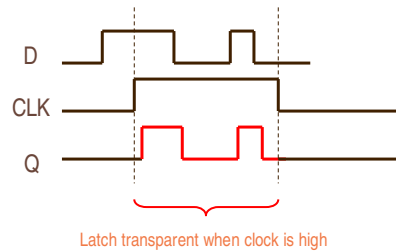
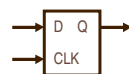
6

State-Holding Memory Elements

- Latch versus Flip Flop
 - Latches are level-sensitive: whenever clock is high, latch is transparent
 - Flip-flops are edge-sensitive: data passes through (i.e. data is sampled) only on a rising (or falling) edge of the clock
 - Latches cheaper to implement than flip-flops
 - Flip-flops are easier to design with than latches
- In this course, primarily use D flip-flops

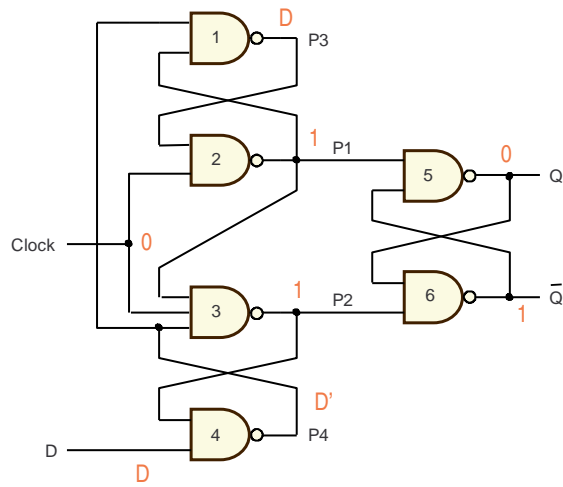
7

D Latch vs. D Flip-Flop

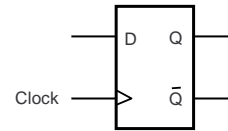


8

D Flip-Flop (positive edge triggered)



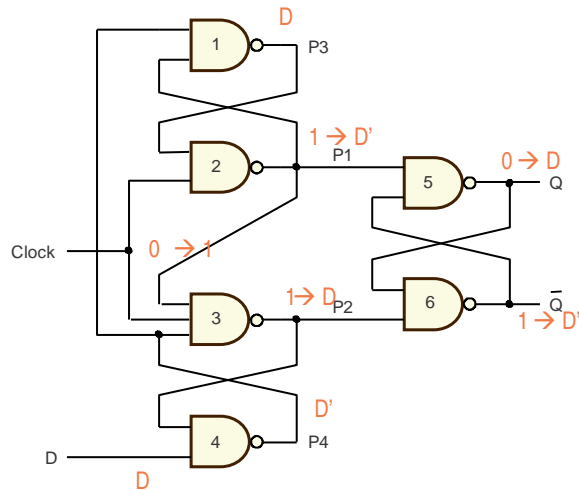
(a) Circuit



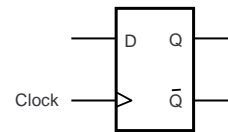
(b) Graphical symbol

NAND : one input 0, output always 1; all inputs 1 except A, output is A'

D Flip-Flop (positive edge triggered)



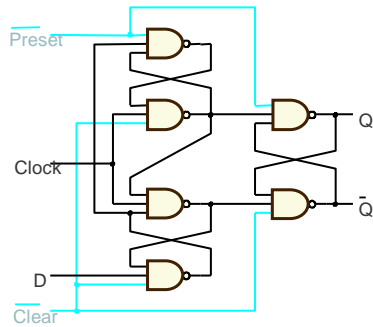
(a) Circuit



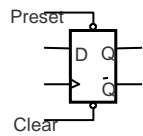
(b) Graphical symbol

NAND : one input 0, output always 1; all inputs 1 except A, output is A'

D Flip-Flop with Asynchronous Preset and Clear



(a) Circuit

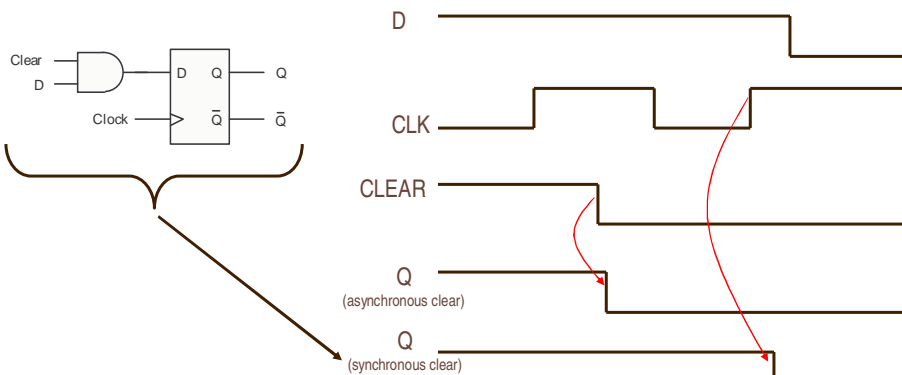


(b) Graphical symbol

- Bubble on the symbol means “active-low”
 - When preset = 0, preset Q to 1
 - When preset = 1, do nothing
 - When clear = 0, clear Q to 0
 - When clear = 1, do nothing
- “Preset” and “Clear” also known as “Set” and “Reset” respectively
- In this circuit, preset and clear are asynchronous
 - Q changes immediately when preset or clear are active, regardless of clock

11

D Flip-Flop with Synchronous Clear

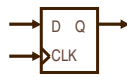


- Asynchronous active-low clear: Q immediately clears to 0
- Synchronous active-low clear: Q clears to 0 on rising-edge of clock

12

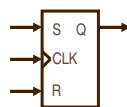
Other Types of Flip-Flops

D Flip-Flop



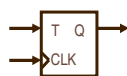
D	Q(t+1)
0	0
1	1

Set-Reset (SR) Flip-Flop



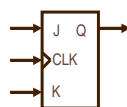
S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	not allowed

Toggle (T) Flip-Flop



T	Q(t+1)
0	Q(t)
1	$\overline{Q(t)}$

JK Flip-Flop



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

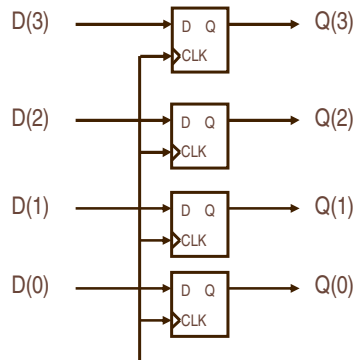
13

Sequential Logic Circuits

some slides modified from:
 Brown and Vranesic, "Fundamentals of Digital Logic with VHDL Design, 2nd Edition"
 S. Dandamudi, "Fundamentals of Computer Organization and Design"

14

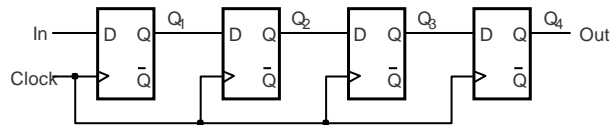
Register



- In typical nomenclature, a register is a name for a collection of flip-flops used to hold a bus (i.e. `std_logic_vector`)

15

Shift Register



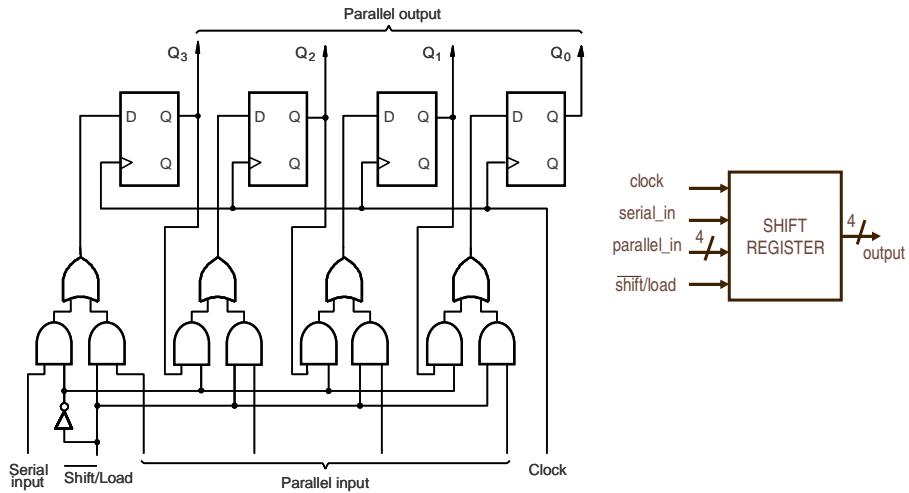
(a) Circuit

	In	Q ₁	Q ₂	Q ₃	Q ₄ = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

(b) A sample sequence

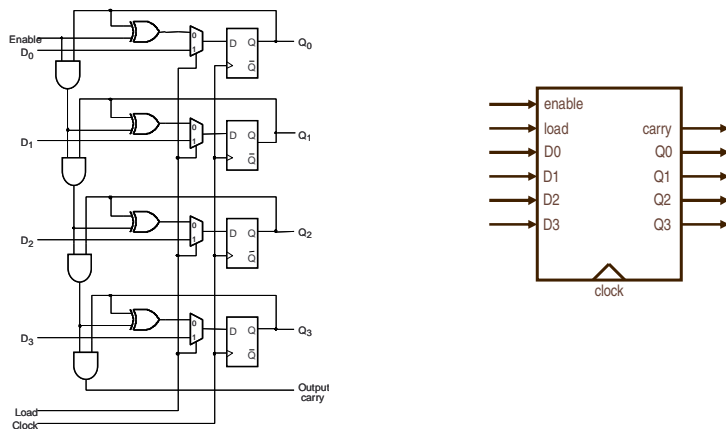
16

Parallel Access Shift Register



17

Synchronous Up Counter



- Enable (synchronous): when high enables the counter, when low counter holds its value
- Load (synchronous) : when load = 1, load the desired value into the counter
- Output carry: indicates when the counter "rolls over"
- D₃ down to D₀, Q₃ down to Q₀ is how to interpret MSB to LSB

18

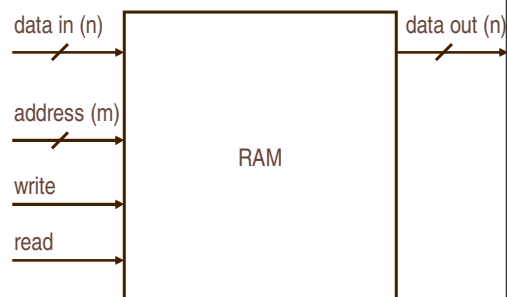
Memories

some slides modified from:
Brown and Vranesic, "Fundamentals of Digital Logic with VHDL Design, 2nd Edition"
S. Dandamudi, "Fundamentals of Computer Organization and Design"

19

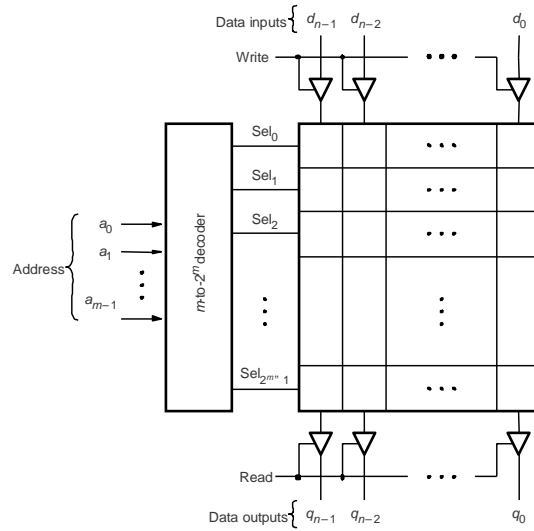
Random Access Memory (RAM)

- More efficient than registers for storing large amounts of data
- Can read and write to RAM
- Addressable memory
- Can be synchronous (with clock) or asynchronous (no clock)
- SRAM dimensions are:
 - (number of words) x (bits per word) SRAM
- Address is m bits, data is n bits
 - 2^m x n-bit RAM
- Example: address is 5 bits, data is 8 bits
 - 32 x 8-bit RAM
- Write
 - Data_in and address are stable
 - Assert write signal (then de-assert)
- Read
 - Address is stable
 - Assert read signal
 - Data_out is valid



20

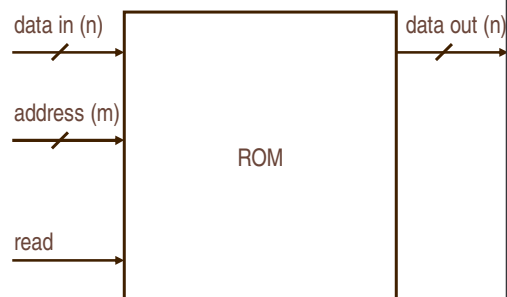
Random Access Memory (RAM)



21

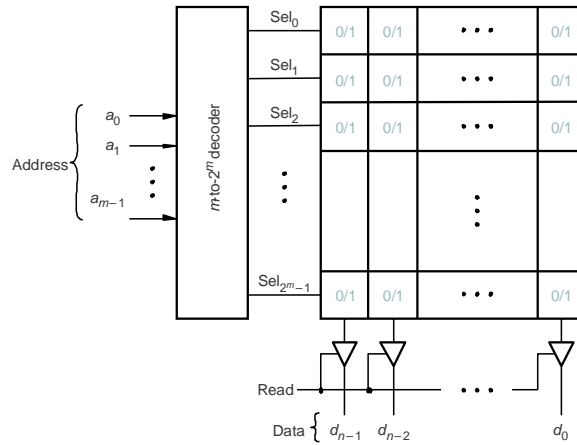
Read Only Memory (ROM)

- Similar to RAM except read only
- Addressable memory
- Can be synchronous (with clock) or asynchronous (no clock)



22

Read-Only Memory (ROM)



23

Simple Finite State Machines

24

Finite State Machines (FSMs)

- Any Circuit with Memory Is a Finite State Machine
 - Even computers can be viewed as huge FSMs
- Design of FSMs Involves
 - Defining states
 - Defining transitions between states
 - Optimization / minimization
- Above Approach Is Practical for Simple FSMs Only

25

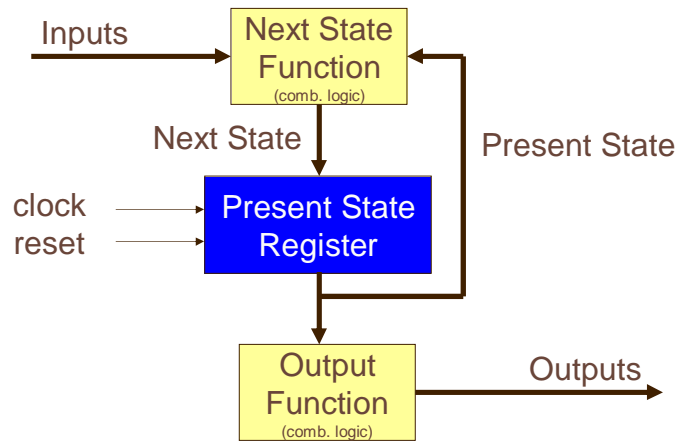
Mealy vs. Moore State Machines

- Finite State Machines (FSM) are of two types:
- Moore Machines
 - Next State = Function(Input, Present State)
 - Output = Function(Present State)
- Mealy Machines
 - Next State = Function(Input, Present State)
 - Output = Function(Input, Present State)

26

Moore FSM

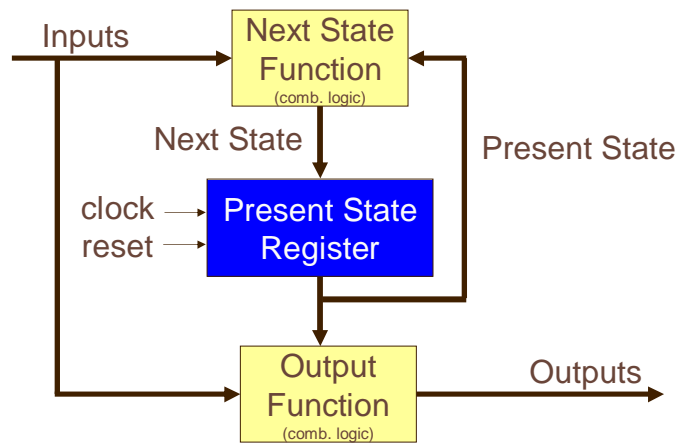
- Output Is a Function of a Present State Only



27

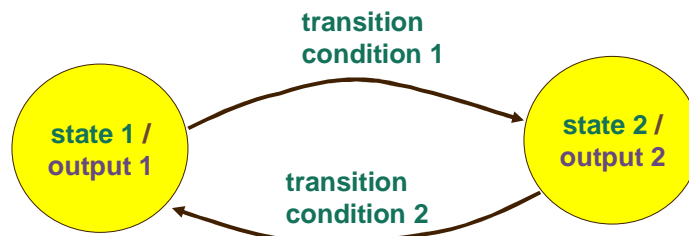
Mealy FSM

- Output Is a Function of a Present State and Inputs



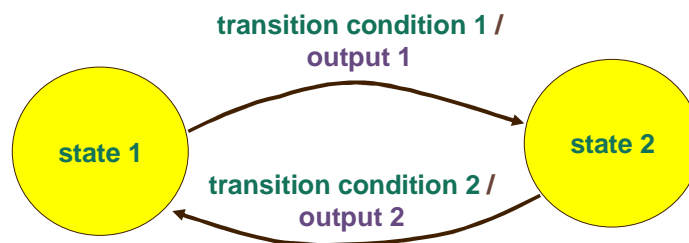
28

Moore Machine



29

Mealy Machine



30

Moore vs. Mealy FSM (1)

- Moore and Mealy FSMs can be functionally equivalent
 - Equivalent Mealy FSM can be derived from Moore FSM and vice versa
- Mealy FSM Has Richer Description and Usually Requires Smaller Number of States
 - Smaller circuit area

31

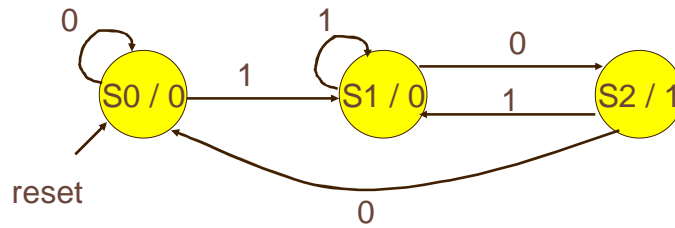
Moore vs. Mealy FSM (2)

- Mealy FSM computes outputs as soon as inputs change
 - Mealy FSM responds one clock cycle sooner than equivalent Moore FSM
- Moore FSM Has no combinational path between inputs and outputs
 - Moore FSM is more likely to have a shorter critical path
 - Moore outputs synchronized with clock; Mealy outputs may not be (may have race conditions, timing issues, etc.)

32

Moore FSM - Example 1

- Moore FSM that Recognizes Sequence "10"



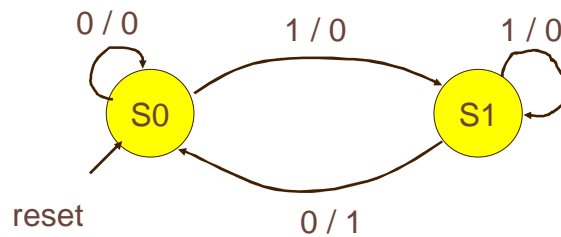
Meaning of states:

S0: No elements of the sequence observed	S1: "1" observed	S2: "10" observed
--	------------------	-------------------

33

Mealy FSM - Example 1

- Mealy FSM that Recognizes Sequence "10"

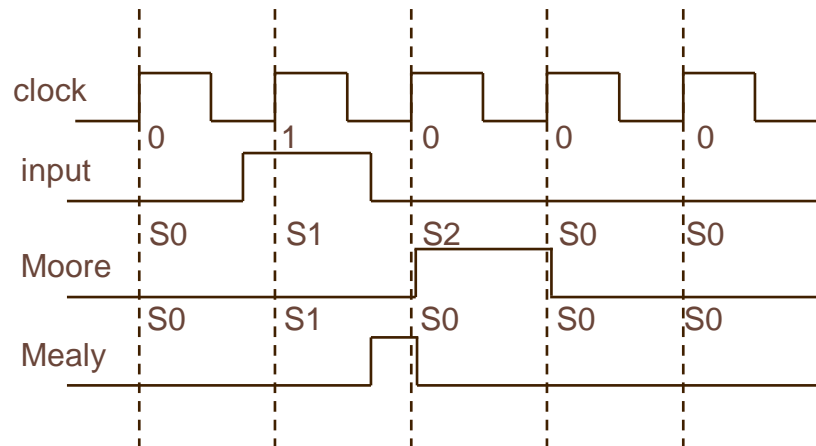


Meaning of states:

S0: No elements of the sequence observed	S1: "1" observed
--	------------------

34

Moore & Mealy FSMs – Example 1



35

FSM Limitations

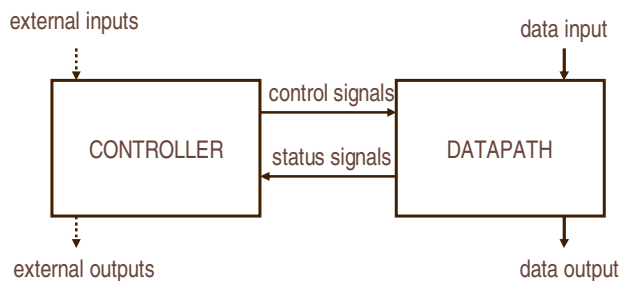
- Simple finite state machines (those expressed using state diagrams and state tables) good only for simple designs
 - Many inputs and many outputs make it awkward to draw state machines
 - Often only one input affects the next change of state
 - Most outputs remain the same from state to state
- Instead use algorithmic state machines (ASM)

36

Complex Digital System Design with Algorithmic State Machines

37

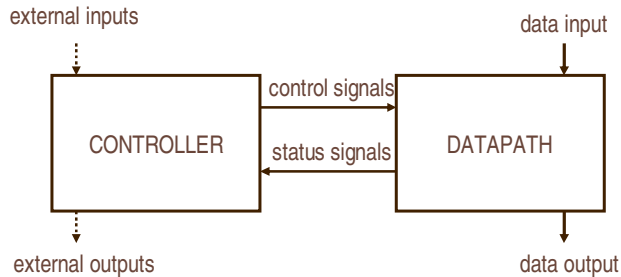
Complex Digital System Design



- Complex digital systems can be decomposed into a datapath and controller
 - Some refer to this as the register-transfer level (RTL) design method
- Datapath manipulates and processes data
 - To perform arithmetic, logic, shifting, and other data-processing tasks
 - These operations are implemented with ALUs, registers, multiplexers, adders, comparators, etc.
- Controller determines the **enabling** and **sequencing** of datapath operations
 - Controller provides signal to activate various processing in the datapath
 - Example: enable signals for registers
 - Example: control signals for muxes
 - Controller also determines the sequence the operations are performed

38

Inputs and Outputs



- Datapath inputs and outputs
 - External data input
 - Example: data to be processed
 - External data output
 - Example: result data
 - Status signals to the controller (to indicate what is occurring in the datapath)
 - Example: when an adder has overflowed
- Controller inputs and outputs
 - External inputs
 - Example: reset signal, mode select
 - External outputs
 - Example: done flag to outside world
 - Control signals to the datapath
 - Example: mux select, enable signals

39

Controller

- Controller can be programmable or non-programmable
- Programmable
 - Has a program counter which point to next instruction
 - Instructions are held in a RAM or ROM externally
 - Microprocessor is an example of programmable controller
- Non-Programmable
 - Once designed, implements the same functionality
 - Another term is a “hardwired state machine” or “hardwired instructions”
 - **We will be focusing primarily on the non-programmable type in this course**

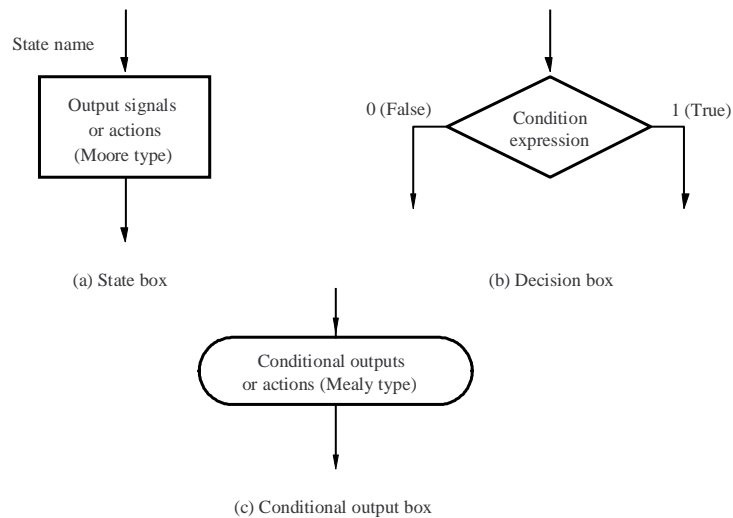
40

Algorithmic State Machine (ASM)

- Complex digital systems can be represented by algorithmic state machines
- Simple finite state machines (expressed using state diagrams and state tables) good only for simple designs
- Algorithmic State Machines (ASM) are
 - flow-chart type diagrams to represent finite state machines
 - suitable for a larger number of inputs and outputs compared to simple FSMs

41

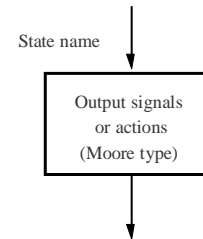
Elements used in ASM charts



42

State Box

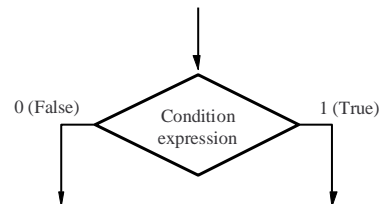
- **State box** – represents a state.
- Equivalent to a node in a state diagram or a row in a state table.
- Contains register transfer actions or output signals
- **Moore-type outputs are listed inside of the box.**
- It is customary to write only the name of the signal that has to be asserted in the given state, e.g., z instead of z=1.
- Also, it might be useful to write an action to be taken, e.g., count = count + 1, and only later translate it to asserting a control signal that causes a given action to take place.



43

Decision Box

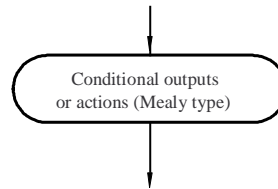
- **Decision box** – indicates that a given condition is to be tested and the exit path is to be chosen accordingly
- The condition expression consists of one or more inputs to the FSM.



44

Conditional Output Box

- **Conditional output box**
- Denotes output signals that are of the Mealy type.
- The condition that determines whether such outputs are generated is specified in the decision box.



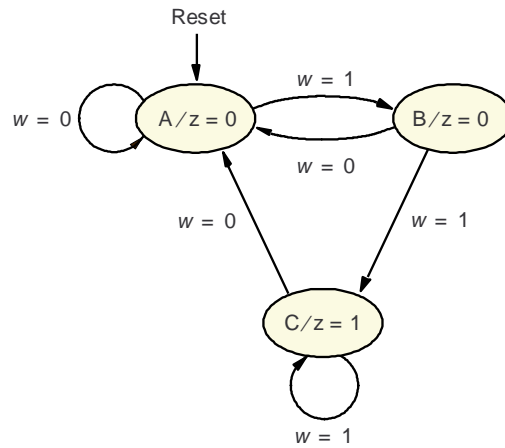
45

ASMs representing simple FSMs

- Algorithmic state machines can model both Mealy and Moore simple finite state machines
- Three examples follow

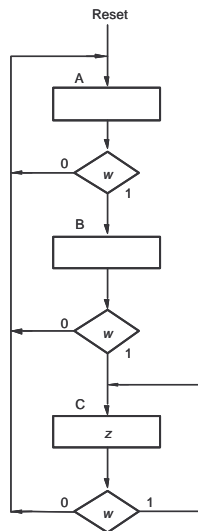
46

Moore FSM – Example 1: State diagram



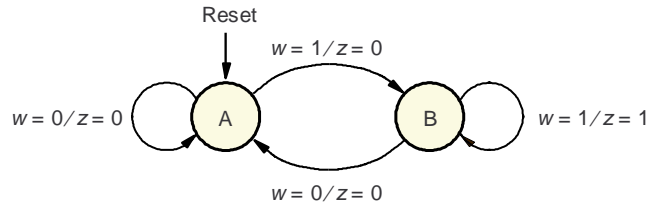
47

ASM Chart for Moore FSM – Example 1



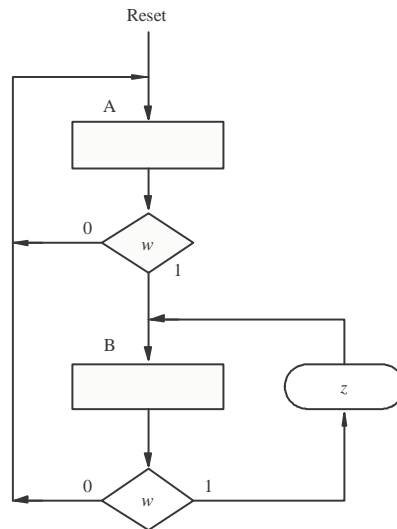
48

Mealy FSM – Example 2: State diagram



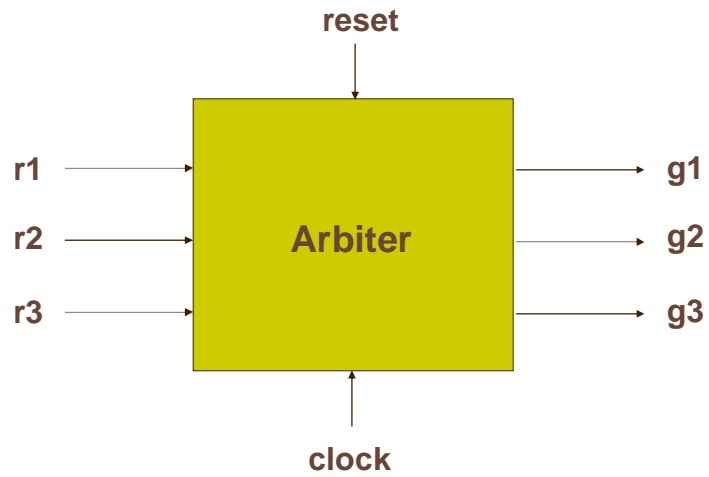
49

ASM Chart for Mealy FSM – Example 2



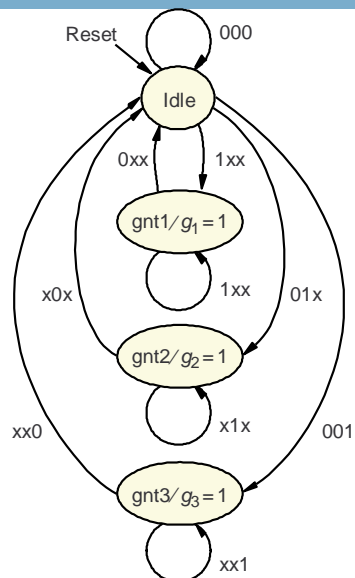
50

Control Unit Example: Arbiter (1)



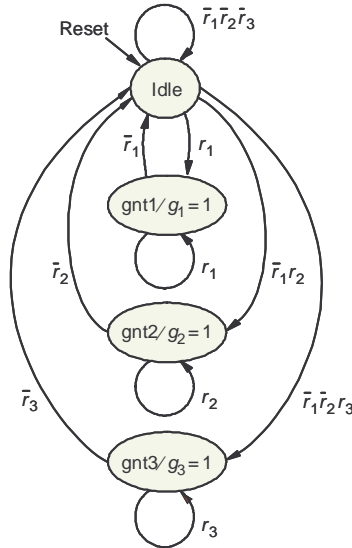
51

Control Unit Example: Arbiter (2)



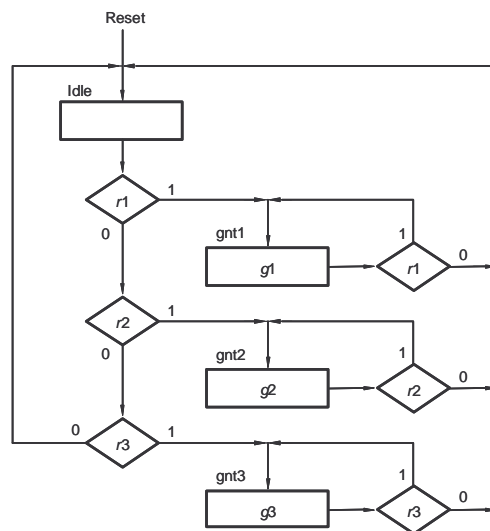
52

Control Unit Example: Arbiter (3)



53

ASM Chart for Control Unit - Example 3



54

Complex Digital System Design: ASM Design Steps

55

Complex Digital Design: ASM Design Steps

- Given a specification, to design a complex digital system using ASMs, the following steps are involved:
 1. Translate specification into pseudocode.
 2. Translate pseudocode into a **high-level ASM**. Also called pseudocode ASM, since it uses pseudocode instead of actual signal names.
 3. Design a **datapath block diagram** based on high-level ASM. Also called an execution unit. (Some references decompose block diagram into a datapath block diagram and controller block diagram.)
 4. Draw **top-level interface diagram**. This diagram connects the datapath with the controller to show all inputs, outputs, and internal connections of the entire digital system.
 5. Design **detailed controller ASM** based on high-level ASM. Detailed means using the exact signal names, not pseudocode representations.
- After this process you have three results:
 - Datapath: represented by a **datapath block diagram**
 - Controller: represented by a **detailed controller ASM**
 - Top-Level: represented by **top-level interface diagram**
- From this it is “easy” to translate into VHDL

56

Examples

- We will do examples in class:
 - Bit-counting circuit
 - Sorting circuit