CSCE 611
High-level VLSI Design

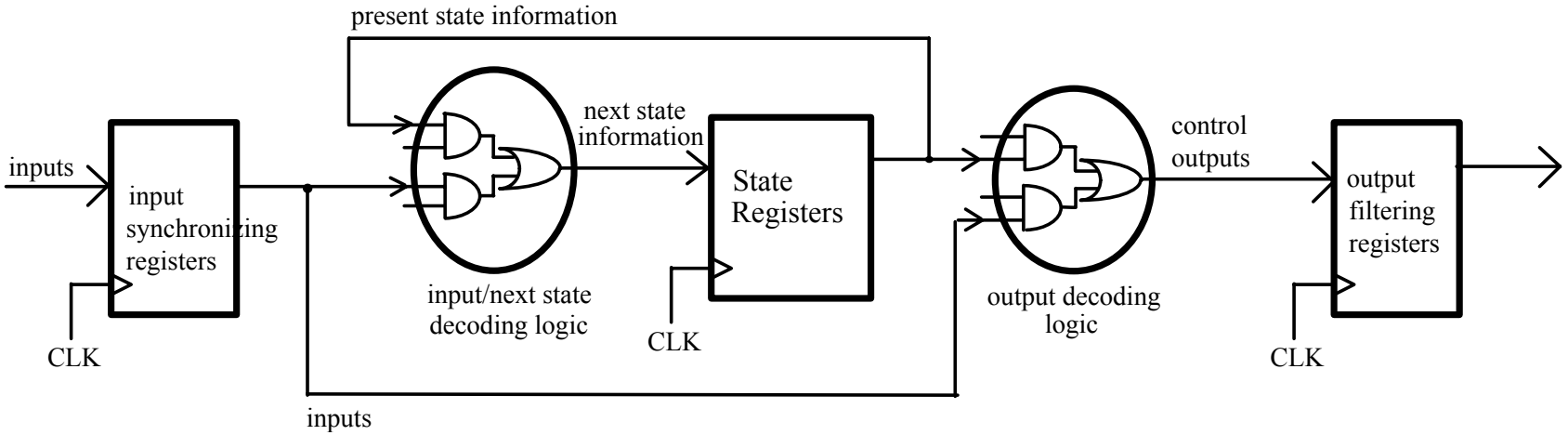
2002/9/8

Fall 2002 - Lecture 8
State Machine Design Methods

Outline

- Methods.
  - Structural – decomposition, refinement
  - Functional – input/output response specification.
  - Behavioral/ASM – sequencing and scheduling of operations in the data path.
  - Behavioral/Timing – input/output response over some time frame.
- Clocking specification.
  - Use waveform specification as a means for defining characteristic system behavior, based on arrival of input signals and the resultant response of the system, and its internal, intermediate actions.
- ASM/Flow diagram.
  - We can also start with a truth table to define the equations for the outputs based on value combinations of the inputs.
  - If we include all inputs—both data and control—then the truth tables can become quite large.
  - However, what we are looking to identify from the equations is the Sum of Products (SoP) form, that can be used to identify which operations are to be scheduled in which states of the state machine (should one be required).
  - Note: we use truth tables for combinational logic function specification, but sometimes these combinational logic functions are under the sequencing control of a state machine.

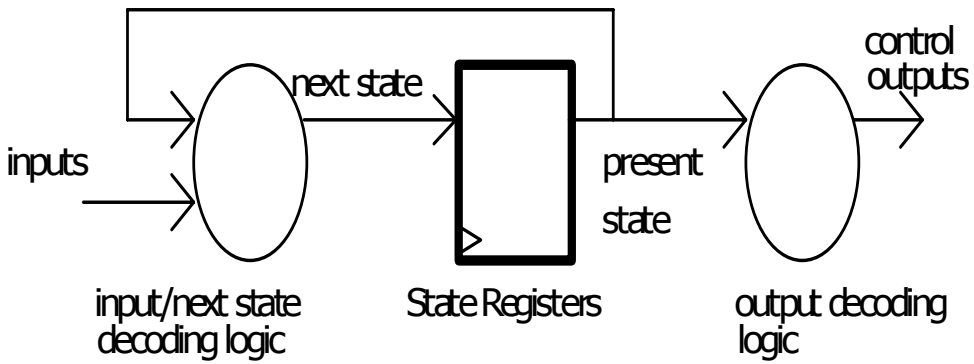
Finite State Machine Design - Introduction



Components of FSM Model

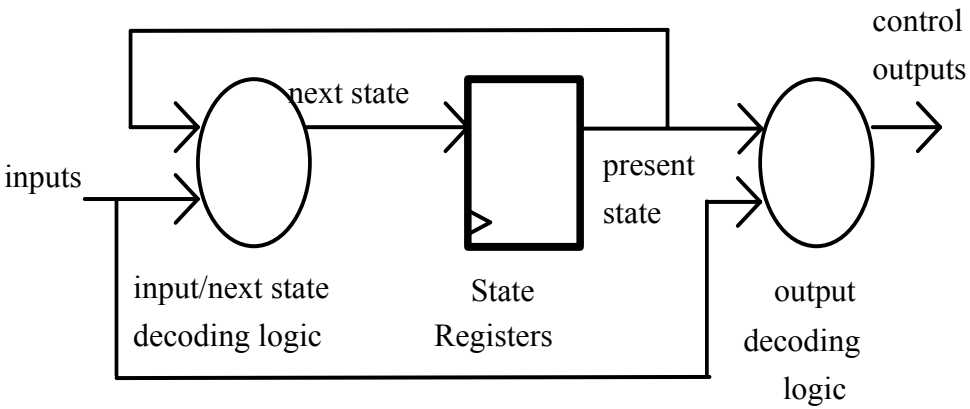
- State registers, input synchronization registers (optional) and output filter registers (optional).
- Next state decoding logic, and output decoding logic - combinational logic blocks.
- Input signals to the state machine, which are inputs to the next state and output decoding logic blocks (could be synchronized to clock with input registers).
- Next state information, which is generated as a result of input/next state decoding logic.
- Present state information, output from the state registers, which is fed back as an input to both next state and output decoding logic blocks.
- Outputs from the state machine - either generated synchronously from the output of the state registers (also used as present state information), or asynchronously as output of the output decoding logic block (which takes input and present state information to produce outputs). Could be filtered using output registers to eliminate possible signal transients.

Finite State Machine Design –Model Types



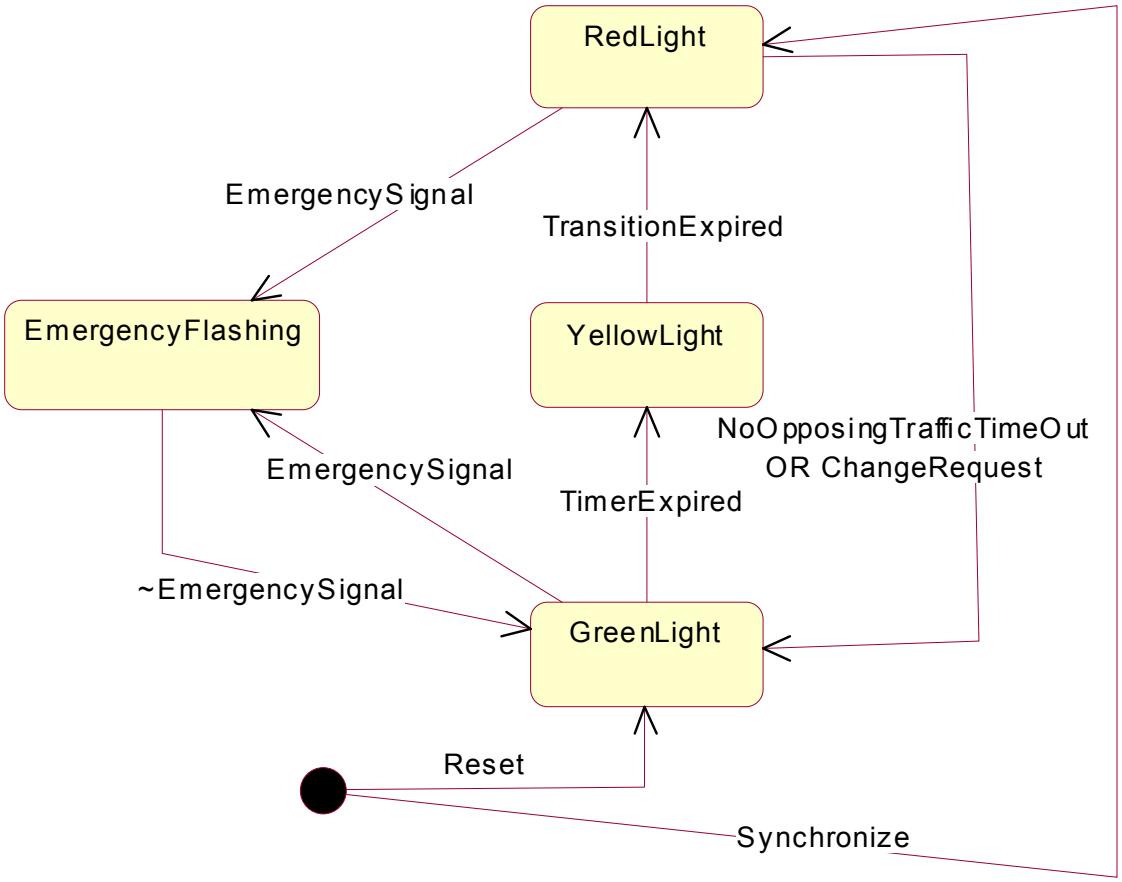
- **Moore machines:**
  - ✓ *Control outputs* generated by the state machine are dependent only on the *present state information*.
  - ✓ The *control outputs* are synchronized to the clock that controls state transitions.
  - ✓ Moore machines are used when it is important to synchronize all control actions with the change in state, and thus, by the clock.
  - ✓ Moore machines effectively filter out transients, and can be used to eliminate race conditions when inputs are unfiltered.

Finite State Machine Design – Model Types



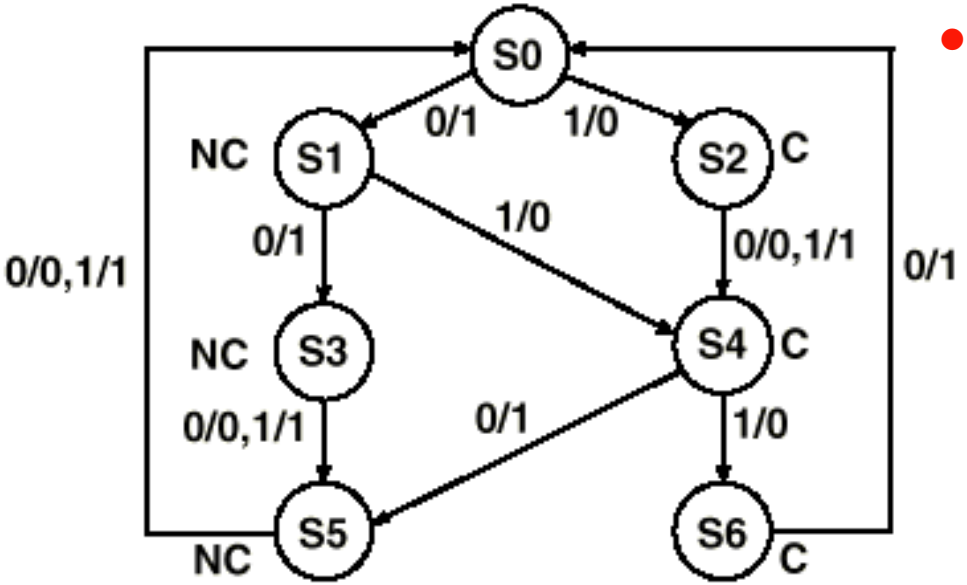
- **Mealy machines:**
  - ✓ The *control outputs* of the state machine are dependent on the *inputs* and *present state information*.
  - ✓ The *control outputs* can be asynchronous, in that outputs can change value as the *inputs* change value, provided the appropriate *present state information* is maintained.
  - ✓ The *control outputs* are gated by the *present state*.
  - ✓ Mealy machines are used to create control blocks that respond quickly to external signal changes.
  - ✓ Care must be taken to isolate the design from transients and race conditions.

Finite State Machine Design – States



- States in real-world domain applications
  - ✓ States of devices are generally determined from its specification.
  - ✓ Certain device applications are easy to discover the states. Other applications may require more work to uncover them.
  - ✓ Sometimes, the list of identified states must be modified to eliminate unused or unspecified states, redundant states, or missing and hidden state behaviors.
  - ✓ NOTE: this model was created using state chart notation in Rational Rose®.

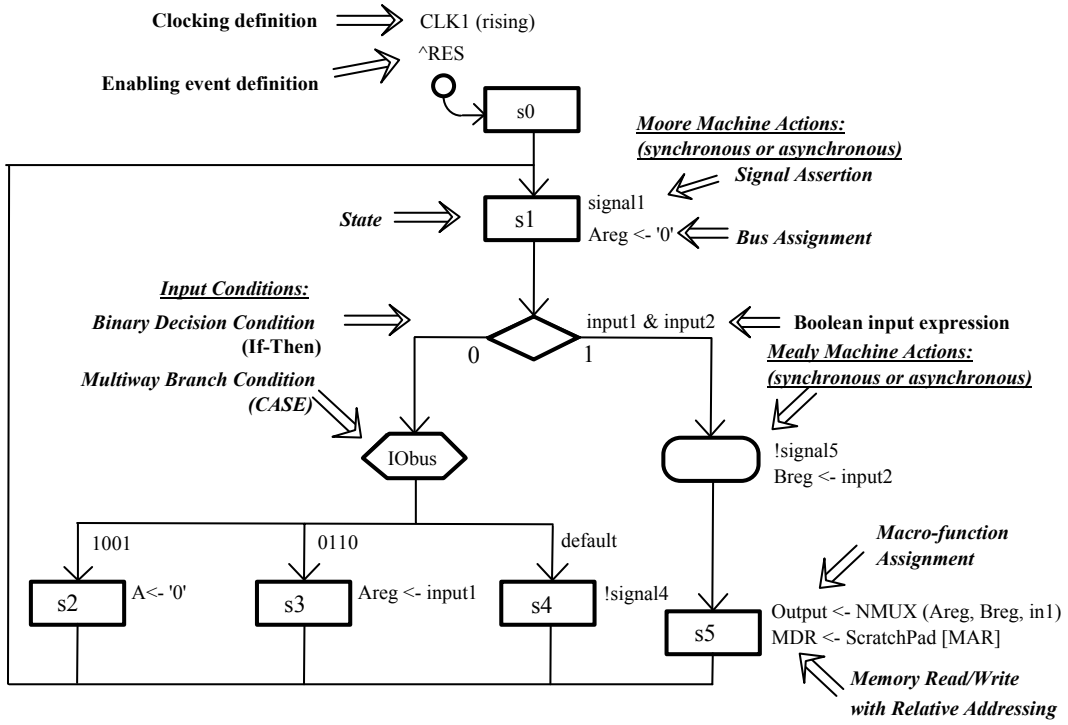
Finite State Machine Design – Notation



PS	NS X = 0	NS X = 1	Z X = 0	Z X = 1
S0	S1	S2	1	0
S1	S3	S4	1	0
S2	S4	S4	0	1
S3	S5	S5	0	1
S4	S5	S6	1	0
S5	S0	S0	0	1
S6	S0	–	1	–

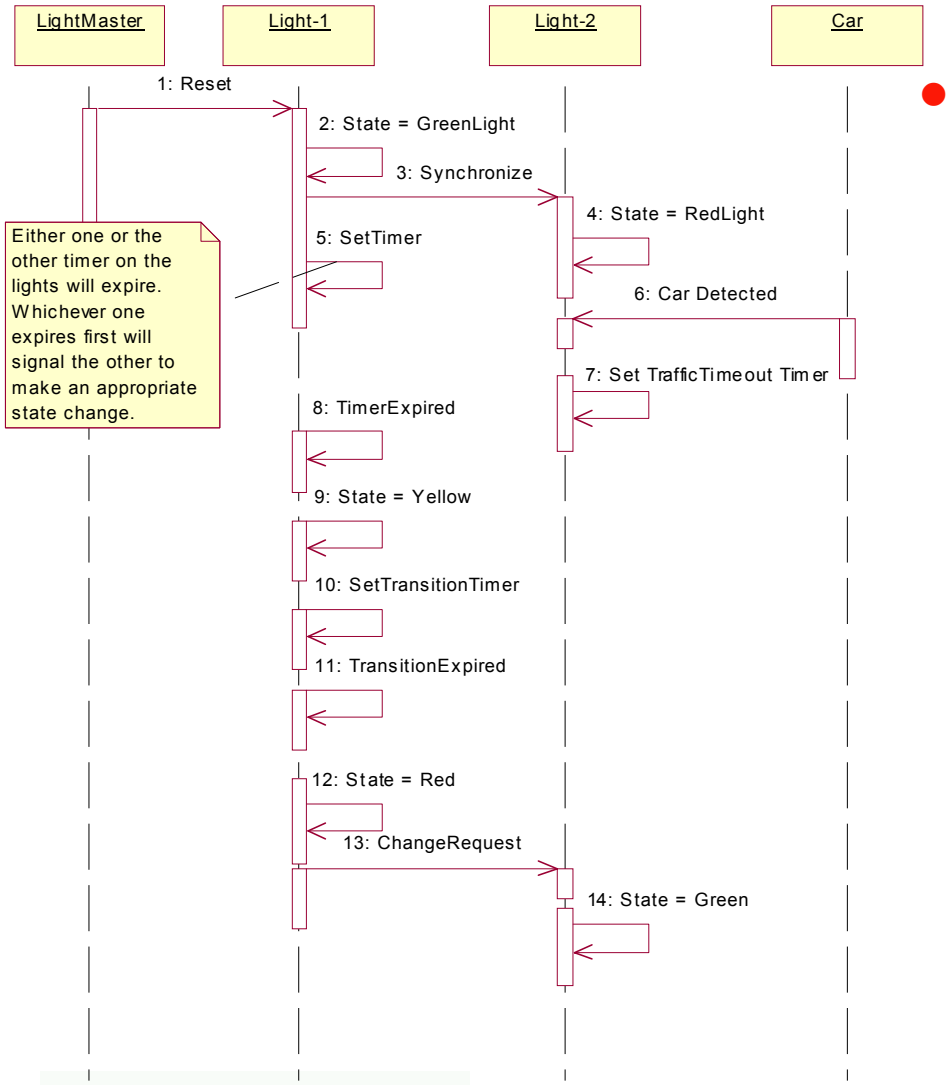
- FSM State Diagram
  - ✓ Allows designer to represent individual states, triggering events, state transitions, and outputs for a state machine.
  - ✓ Notation:
    - ✧ "bubble" is a state, with state name
    - ✧ "arc" is a transition from one state to another, or from a state to itself. It has the inputs indicated as a logical expression.
    - ✧ Moore-style FSM has outputs associated with state itself.
    - ✧ Mealy-style FSM has outputs associated with transition arc.

Finite State Machine Design – Notation



- Algorithmic State Machine (ASM) Chart
  - ✓ Uses notation of flow chart for modeling the sequencing and control in state machines.
  - ✓ Actions to be scheduled in each particular state are attached to the state, represented by the “state box”.
  - ✓ State transition decisions are modeled explicitly using VHDL control constructs, for If-Then and Case.
  - ✓ Actions attached to state box indicate Moore machine, but actions on “oval” after a state imply Mealy style of state machine.

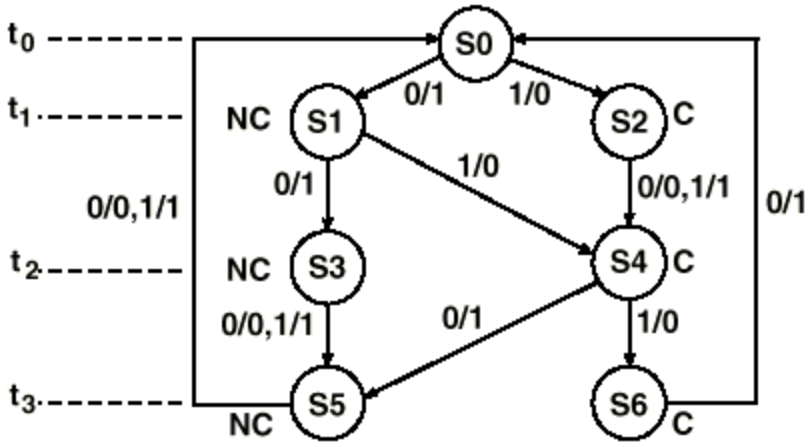
Finite State Machine Design – Notation



- Sequence Diagrams
  - ✓ Allow designer to model interactions between multiple design blocks containing state machines or data path units.
  - ✓ Represents the time sequenced actions and events between units.
  - ✓ Clearly shows the sequencing associated with handshaking or design protocol being implemented.
  - ✓ Should be verifiable using VHDL simulation.

UNIVERSITY OF SOUTH CAROLINA

Finite State Machine Design – Encoding



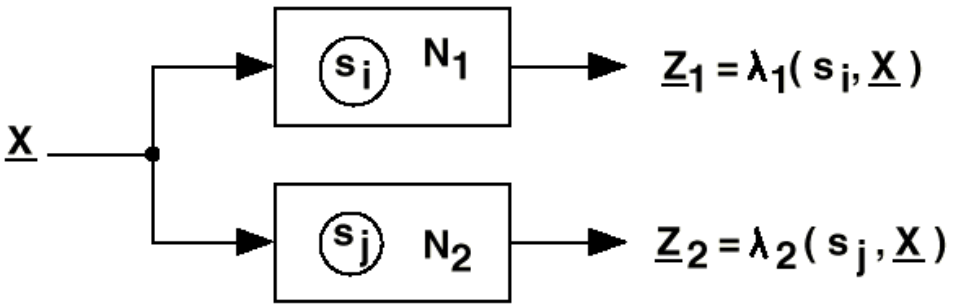
(a) Mealy state graph

	NS		Z	
PS	X = 0	X = 1	X = 0	X = 1
S0	S1	S2	1	0
S1	S3	S4	1	0
S2	S4	S4	0	1
S3	S5	S5	0	1
S4	S5	S6	1	0
S5	S0	S0	0	1
S6	S0	–	1	–

(b) State Table

- State Assignment
  - One technique for assigning states to flip flop encoding is to start with the State Table.
- State Encoding Schemes
  - *Binary*: the state assignment is made based on the order of the states, with the total number of flip flops used for state assignment being a power of two (so as to eliminate problems with unassigned states). See rules on Pages 20, 23 of Roth text.
  - *One-Hot*: For FPGA design, where the goal is to minimize interconnect across cells, this scheme is used to have one flip flop per state, instead of encoding as a power of 2.
  - *Grey Code*: Different ordering of the states according to adjacency and prioritization.

Finite State Machine Design – Reduction



$s_i \equiv s_j$ iff $\underline{Z}_1 = \underline{Z}_2$
for every input sequence $\underline{X}$

- State Equivalence
  - ✓ Two states in an FSM are "equivalent" if they have the same input and output sequence.
  - ✓ The output sequence (not just individual outputs) must be the same for all possible combinations of input sequences.
  - ✓ We can do this without having to look internally to the FSM if we can say that, for all input sequences, the outputs and next states are the same.
- Implication Reduction Technique
  - ✓ Starting with State Table, we create an implication Table and follow algorithm in text.

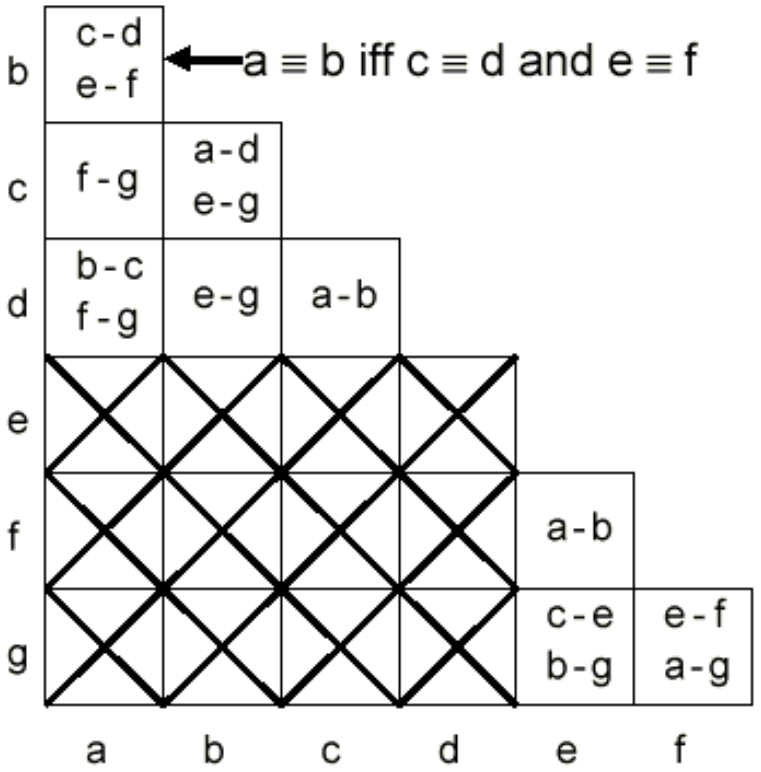
UNIVERSITY OF SOUTH CAROLINA

Finite State Machine Design – Reduction

State Table

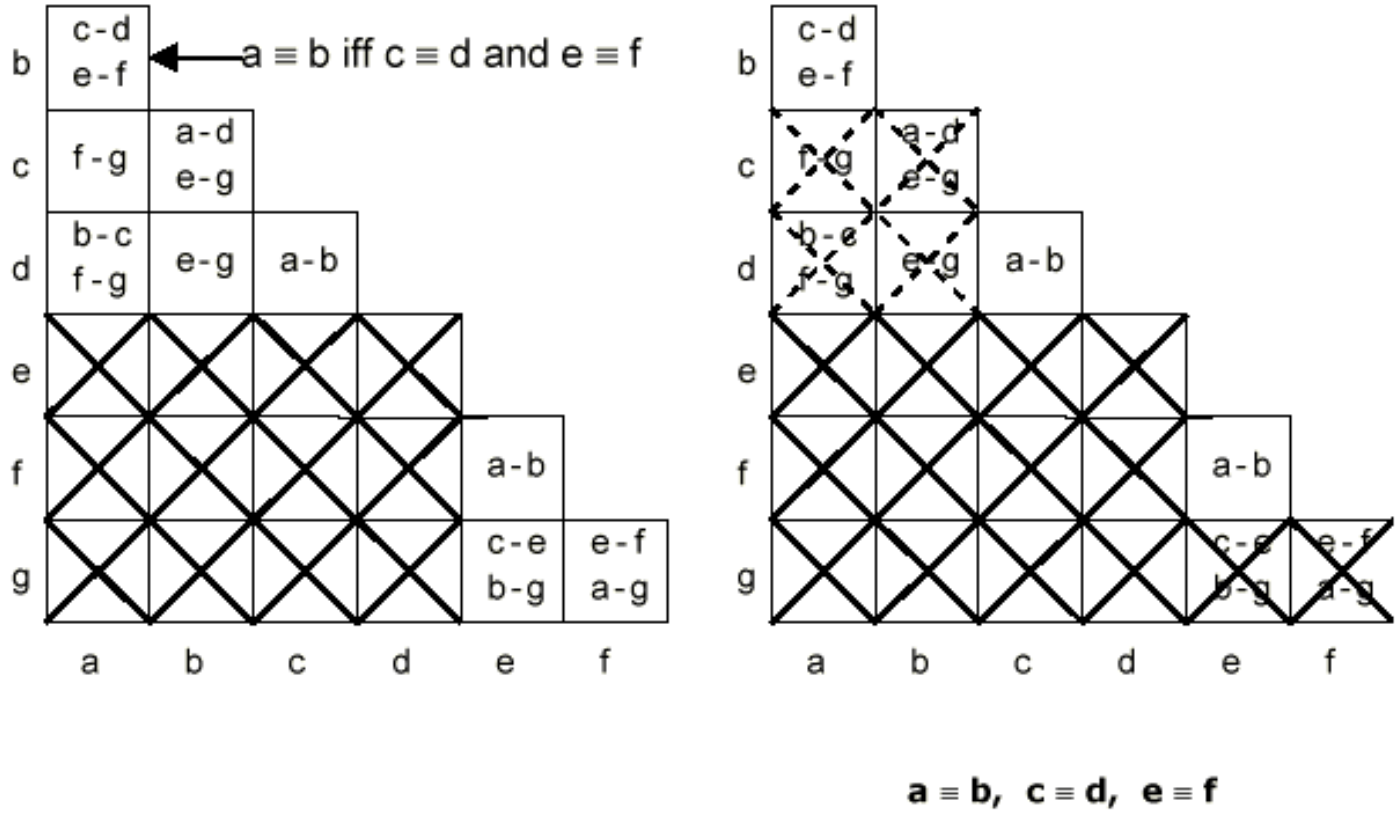
Present State	Next State X = 0	Next State X = 1	Present Output X = 0	Present Output X = 1
a	c	f	0	0
b	d	e	0	0
c	~~h~~ a	g	0	0
d	b	g	0	0
e	e	b	0	1
f	f	a	0	1
g	c	g	0	1
~~h~~	~~c~~	~~f~~	~~0~~	~~0~~

Constructed Implication Chart



Finite State Machine Design – Reduction

Subsequent passes minimizing the Implication Chart



Finite State Machine Design – Reduction

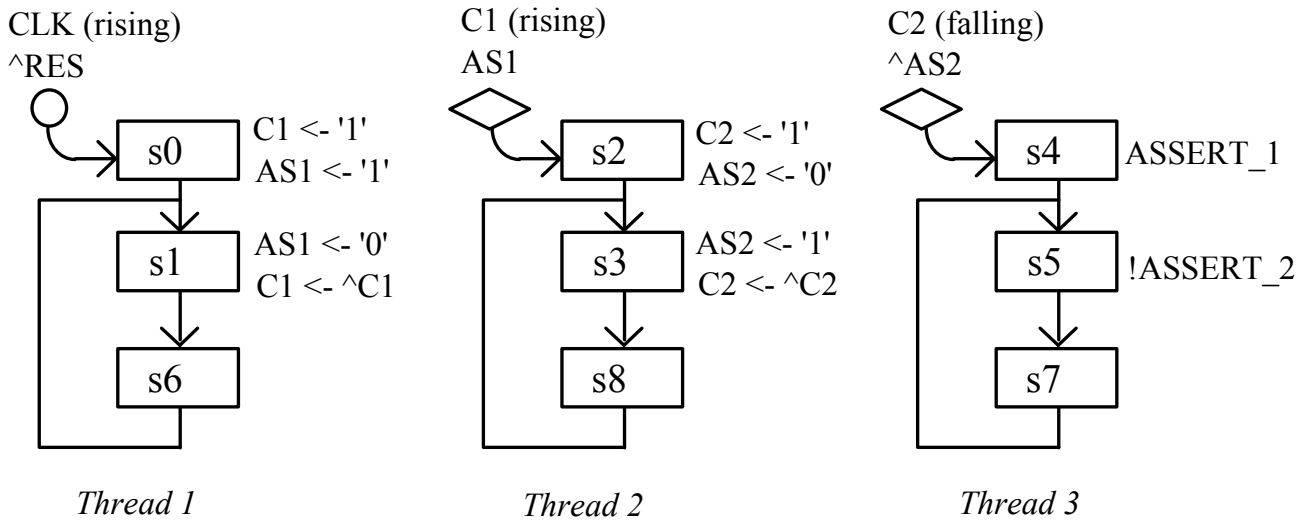
Present State	Next State X = 0	1	Present Output X = 0	1
a	c	f	0	0
b	d	e	0	0
c	~~h~~ a	g	0	0
d	b	g	0	0
e	e	b	0	1
f	f	a	0	1
g	c	g	0	1
~~h~~	~~c~~	~~f~~	~~0~~	~~0~~

$a \equiv b,\ c \equiv d,\ e \equiv f$

Present State	X = 0	1	X = 0	1
a	c	e	0	0
c	a	g	0	0
e	e	a	0	1
g	c	g	0	1

Final Reduced Table

Finite State Machine Design – Concurrency



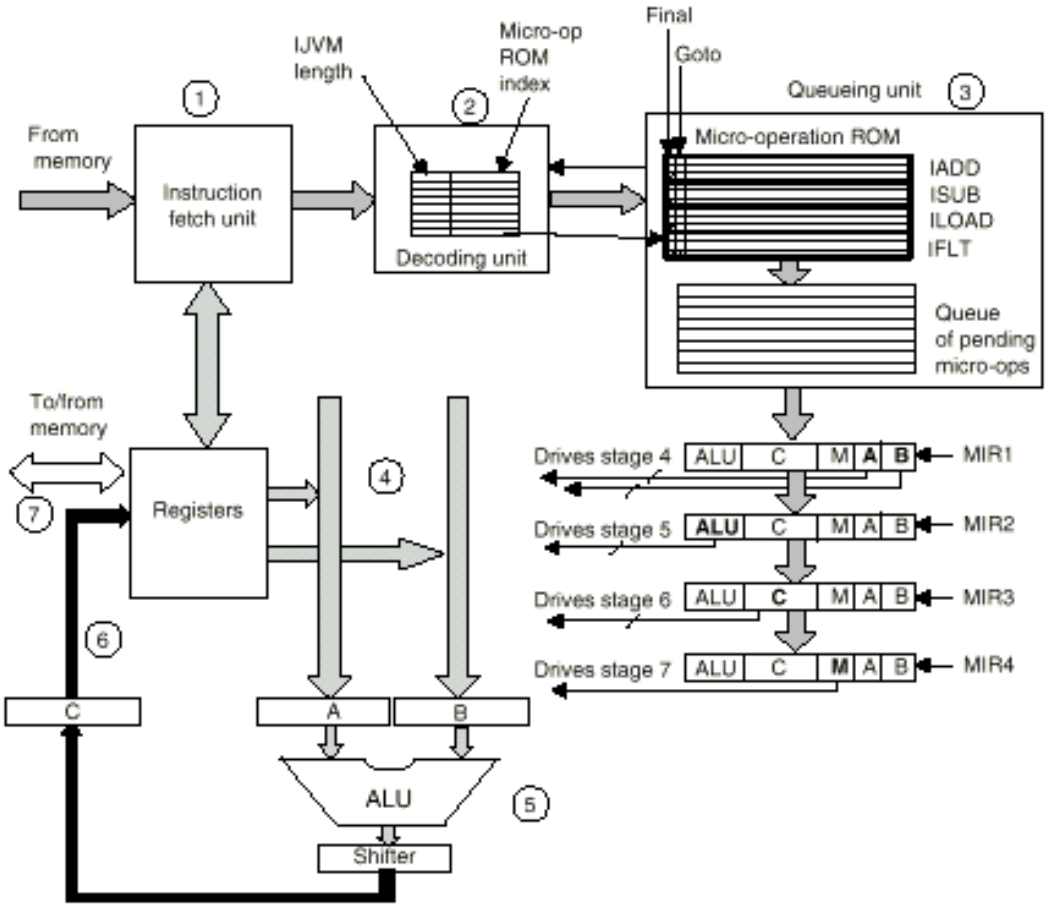
☐**Modeling Concurrency:**

- ☐ Multiple model FSM "threads" having shared buses.
- ☐ Independent clocking schemes and enabling events (e.g., ^RES).
- ☐ Types of concurrent interaction:

Synchronization
- coordinated activities (e.g., handshaking, pipelining)
- implicit references to shared buses

Competition
- shared resources (for example, bus arbitration)
- explicit use of other concurrent processes, components, or entities to model the arbitration protocol.

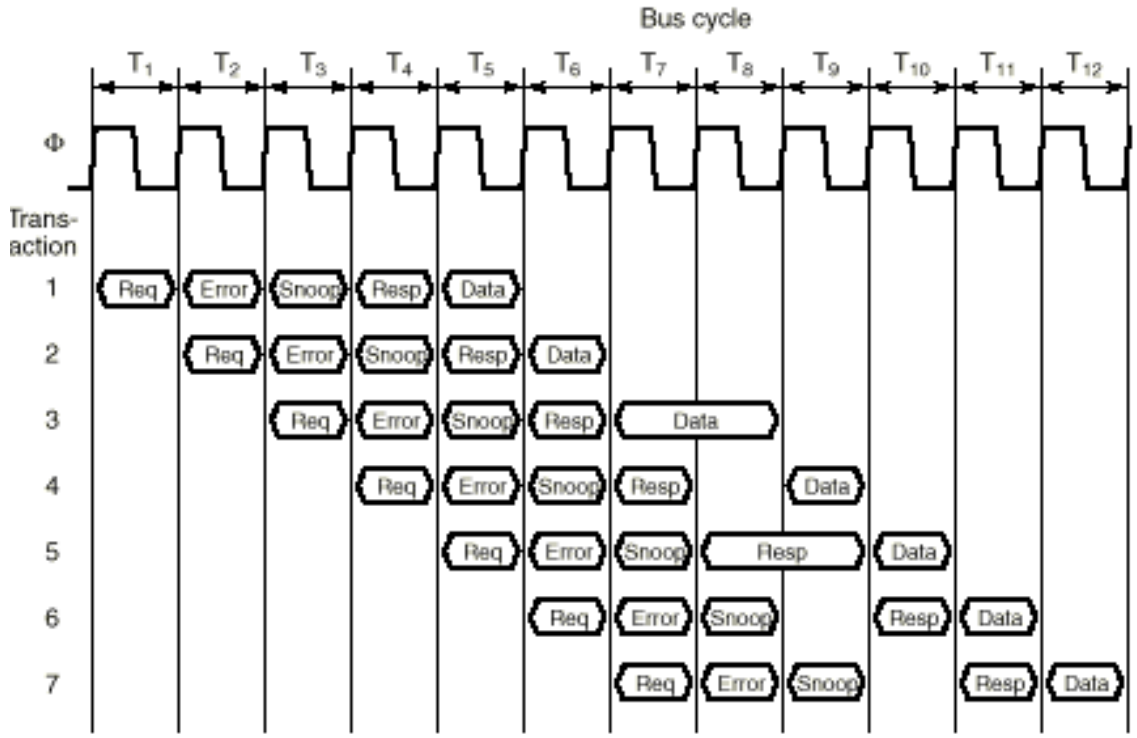
Finite State Machine Design – Patterns



- Sequencing
  - ✓ This pattern has the sequencing of data path operations by one or more state machines
  - ✓ The example shown is the data path for a small CPU, where micro-operations based on program instructions are decoded and staged to execute multi-cycle instructions out of memory.
  - ✓ This example also used pipelining (discussed later).

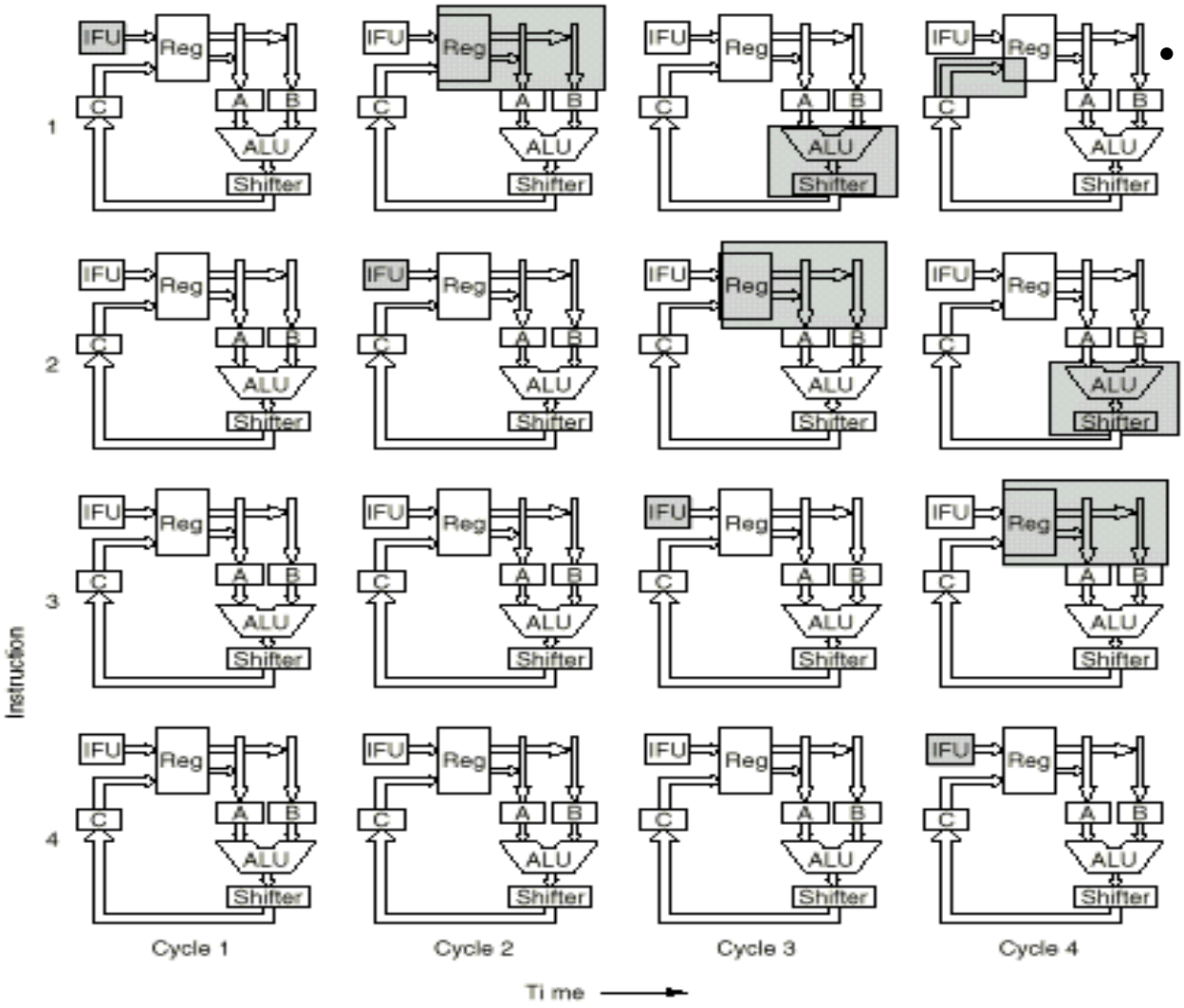
UNIVERSITY OF SOUTH CAROLINA

Finite State Machine Design – Patterns



- Pipelining - 1
  - ✓ There are two kinds of pipelining: data path pipelining and control pipelining.
  - ✓ An example of control pipelining is the Instruction Fetch, Decode, Execute cycle used in all CPU architectures.
  - ✓ Another example is Bus Reads and Writes, which are generally pipelined so as to interleave the control cycles, thus saving clock cycles (shown in the figure).

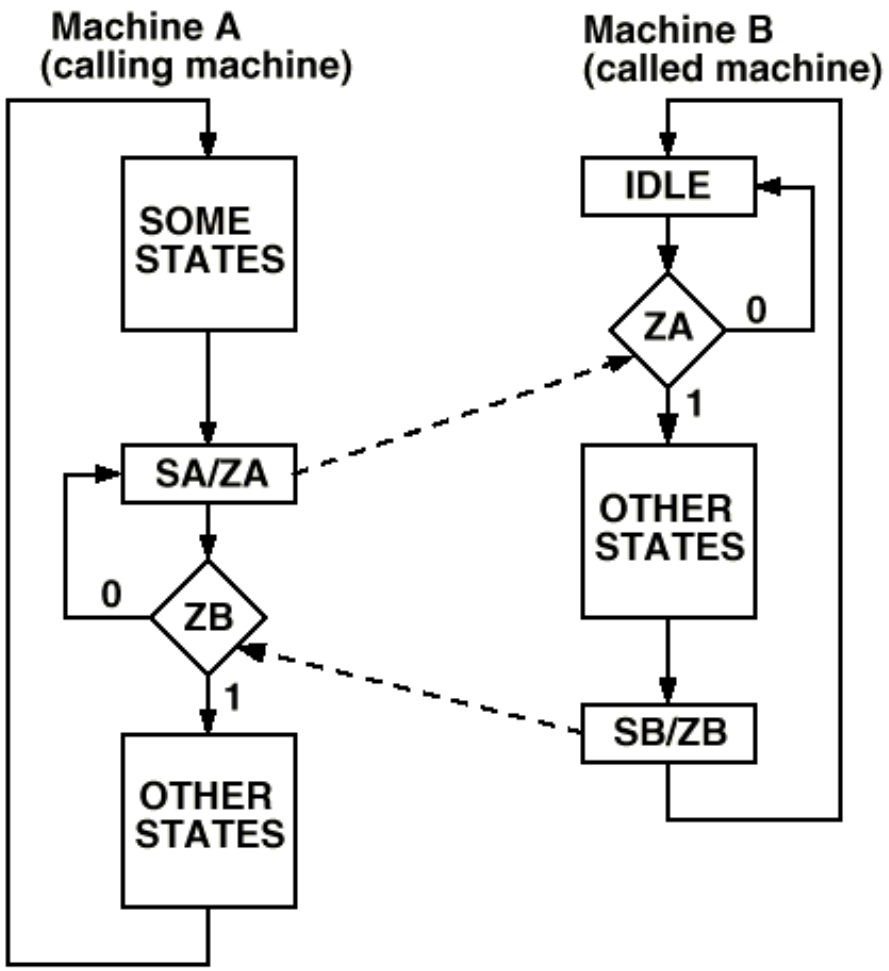
Finite State Machine Design – Patterns



Pipelining - 2

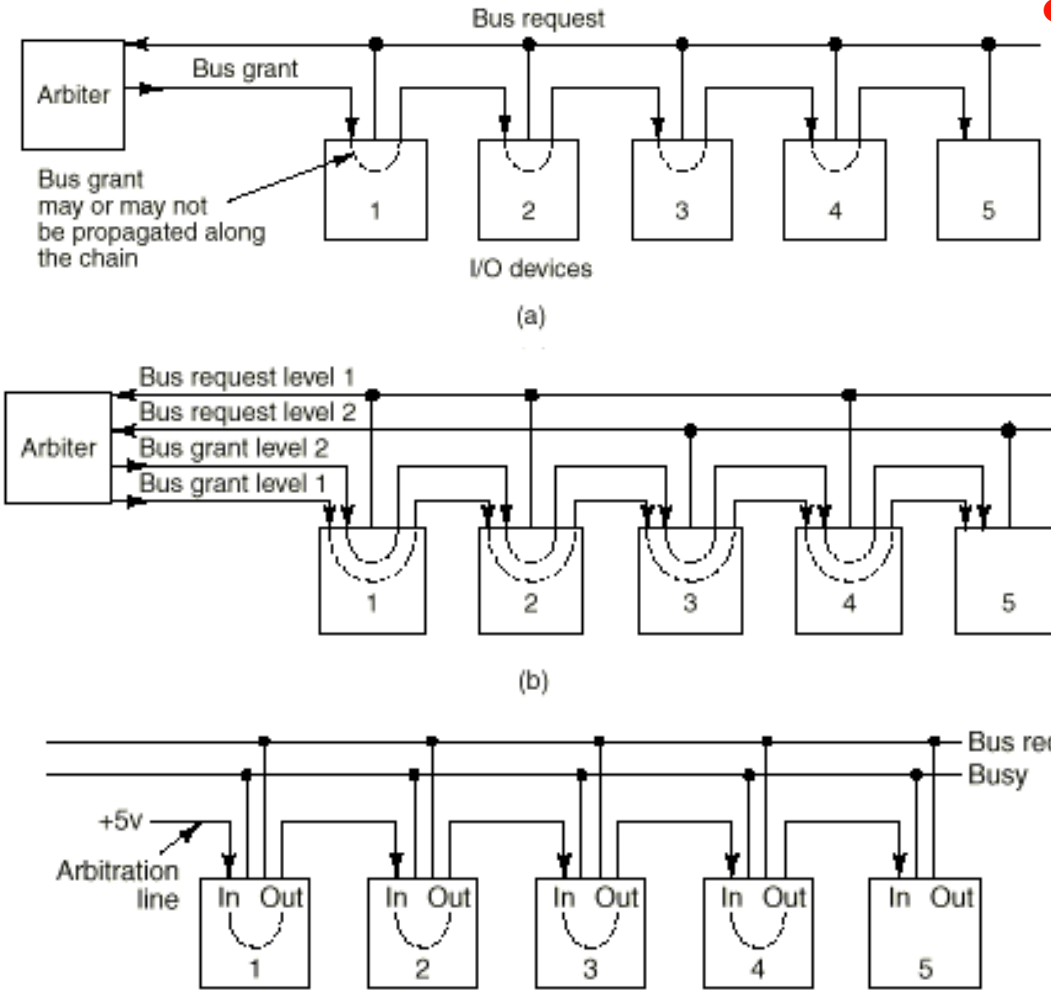
- The sequence of figures show how pipelining works in the control path.
- The control pipelining is the Instruction Fetch, Decode, Execute cycle used in all CPU architectures.
- Each stage of the control pipeline is buffered by registers that provide setup of data.
- The different stages of the pipeline also use handshaking.

Finite State Machine Design – Patterns



- Handshaking
  - ✓ Polled handshaking:
    - ✧ FSM A thread waits in a polling loop, testing for signal ZB to be asserted by FSM B.
    - ✧ FSM B thread waits in IDLE loop for signal ZA to be asserted by FSM A.
  - ✓ Asynchronous handshaking:
    - ✧ FSM threads use an asynchronous interrupt mechanism to alert it to when the event has occurred.
    - ✧ However, to minimize the effects of timing skew, it is most likely gated to a clock signal.

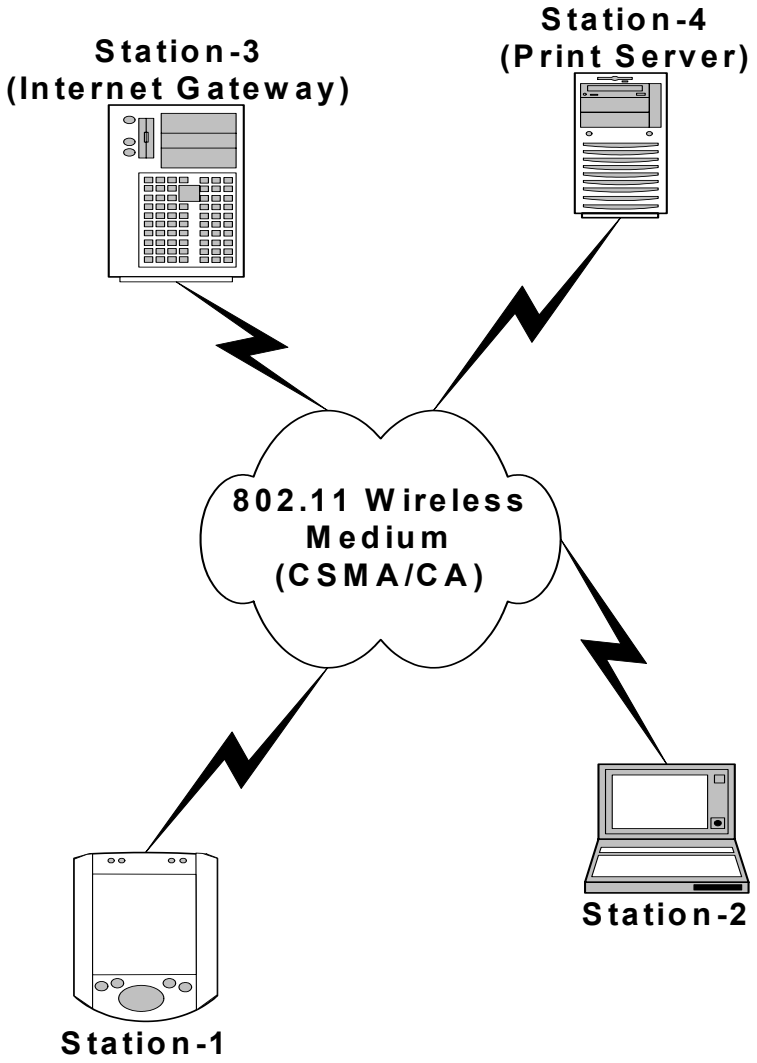
Finite State Machine Design – Patterns



- Arbitration-1
  - ✓ The pattern works in situations where multiple service "requesters" want access to a scarce resource (such as a Bus).
  - ✓ There are different arbitration schemes for requesting and granting control of the resource by one requester by the "arbiter" module.
  - ✓ Some use daisy chaining, or prioritization schemes to grant access.
  - ✓ Arbitration can be centralized, using an "arbiter" module, or it can be decentralized.

UNIVERSITY OF SOUTH CAROLINA

Finite State Machine Design – Patterns



- Arbitration-2
  - ✓ The one scheme involves use of a separate “arbiter” module, as is the case with most bus schemes.
  - ✓ Another scheme involves no centralized arbiter:
    - ✧ CSMA/CD: Carrier Sense Multiple Access/Collision Detect. Sense for a distributed “carrier” signal, and detect for collisions as a means to gain access to the shared resource (wired network medium).
    - ✧ CSMA/CA: Carrier Sense Multiple Access/Collision Avoidance. Sense for “carrier” signal, but don’t rely on it solely as the means for gaining access. Use an additional timing mechanism passed among the data frames (needed because of the “hidden node” problem).

