

# Modeling Embedded Systems

EE202A (Fall 2001): Lecture #2

**Mani Srivastava**  
**UCLA - EE Department**  
**Room: 7702-B Boelter Hall**  
**Email: mbs@ee.ucla.edu**  
**Tel: 310-267-2098**  
**WWW: <http://www.ee.ucla.edu/~mbs>**

## Reading List for This Lecture

### ■ MANDATORY READING

- ◆ E.A. Lee, "Embedded Software," UCB ERL Memorandum M01/26.  
<http://ptolemy.eecs.berkeley.edu/publications/papers/01/embsystems/>
- ◆ Harel, D.; Lachover, H.; Naamad, A.; Pnueli, A.; Politi, M.; Sherman, R.; Shtull-Trauring, A.; Trakhtenbrot, M.  
 STATEMATE: a working environment for the development of complex reactive systems. IEEE Transactions on Software Engineering, vol.16, (no.4), April 1990. p.403-14.  
<http://ielimg.ihs.com/iel1/32/1950/00054292.pdf>

### ■ OTHER READING

- ◆ None

3

Copyright 2001 © Mani Srivastava

## How to Design Embedded Systems? (Wescon 1975)

- "...deliberately avoid data processing aides such as assemblers, high-level languages, simulated systems, and control panels. These computer-aided design tools generally get in the way of cost-effective design and are more a result of the cultural influence of data processing, rather than a practical need."
- "It' s my observation that the bulk of real-world control problems require less than 2,000 instructions to implement. For this size program computer aided design does little to improve the design approach and does a lot to separate the design engineer from intimate knowledge of his hardware."

4

Copyright 2001 © Mani Srivastava

## Methodical Design of Embedded Systems

- Ad hoc approach to design does not work beyond a certain level of complexity, that is exceeded by vast majority of embedded systems
- Methodical, engineering-oriented, tool-based approach is essential
  - ◆ specification, synthesis, optimization, verification etc.
  - ◆ prevalent for hardware, still rare for software
- One key aspect is the creation of *models*
  - ◆ concrete representation of knowledge and ideas about a system being developed - specification
  - ◆ model deliberately modifies or omits details (abstraction) but concretely represents certain properties to be analyzed, understood and verified
  - ◆ one of the few tools for dealing with complexity

5

Copyright 2001 © Mani Srivastava

## Abstractions and Models

- Foundations of science and engineering
- Activities usually start with informal specification
- However, soon a need for Models and Abstractions is established
  - ◆ e.g.: Chess and Poker - No Rules, No Games
- Connections to Implementation (h/w, s/w) and Application
- Two types of modeling: system structure & system behavior
  - ◆ the behavior and interaction of atomic components
    - **coordinate computation of & communication between components**
- Models from classical CS
  - ◆ FSM, RAM (von Neumann)
  - ◆ CSP (Hoare), CCS (Milner)
  - ◆ Turing machine, Post Machine, Universal Register Machine

6

Copyright 2001 © Mani Srivastava

## Good Models

- Simple
  - ◆ Ptolemy vs. Galileo
- Amenable for development of theory to reason
  - ◆ should not be too general
- Has High Expressive Power
  - ◆ a game is interesting only if it has some level of difficulty!
- Provides Ability for Critical Reasoning
  - ◆ Science vs. Religion
- Practice is currently THE only serious test of model quality
- Executable (for Simulation)
- Synthesizable
- Unbiased towards any specific implementation (h/w or s/w)

7 Copyright 2001 © Mani Srivastava

## Separate Behavior from Architecture

◆ **System Behavior**

- ◆ Functional Specification of System.
- ◆ No notion of hardware or software!

◆ **Implementation Architecture**

- ◆ Hardware and Software
- ◆ Optimized Computer

8 Copyright 2001 © Mani Srivastava

## Elements of a Model of a Computation System: Language

- Set of symbols with superimposed syntax & semantics
  - ◆ textual (e.g. matlab), visual (e.g. labview) etc.
- Syntax: rules for combining symbols
  - ◆ well structured, intuitive
- Semantics: rules for assigning meaning to symbols and combinations of symbols
  - ◆ without rigorous semantics, precise model behavior over time is not well defined
  - ◆ full executability and automatic h/w or s/w synthesis is impossible
  - ◆ E.g. operational semantics (in terms of actions of an abstract machine), denotational semantics (in terms of relations)

9

Copyright 2001 © Mani Srivastava

## Simulation and Synthesis

- Two sides of the same coin
- **Simulation**: scheduling then execution on desktop computer(s)
- **Synthesis**: scheduling then code generation in C++, C, assembly, VHDL, etc.
- Validation by simulation important throughout design flow
- Models of computation enable
  - ◆ Global optimization of computation and communication
  - ◆ Scheduling and communication that is *correct by construction*

10

Copyright 2001 © Mani Srivastava

## Models Useful In Validating Designs

- **By construction**
  - ◆ property is inherent.
- **By verification**
  - ◆ property is provable.
- **By simulation**
  - ◆ check behavior for all inputs.
- **By intuition**
  - ◆ property is true. I just know it is.
- **By assertion**
  - ◆ property is true. Wanna make something of it?
- **By intimidation**
  - ◆ Don't even try to doubt whether it is true

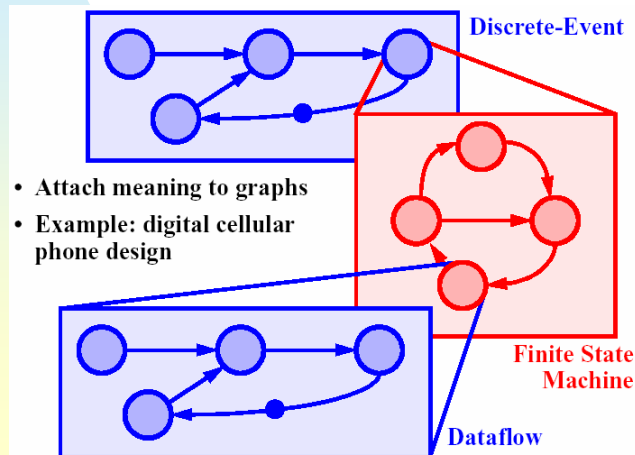
**It is generally better to be higher in this list**

11

Copyright 2001 © Mani Srivastava

## Heterogeneous Systems

- Hierarchical composition of models
- Need to understand how models relate when combined in a single system



- Attach meaning to graphs
- Example: digital cellular phone design

[Evans]

12

Copyright 2001 © Mani Srivastava

## Modeling Embedded Systems

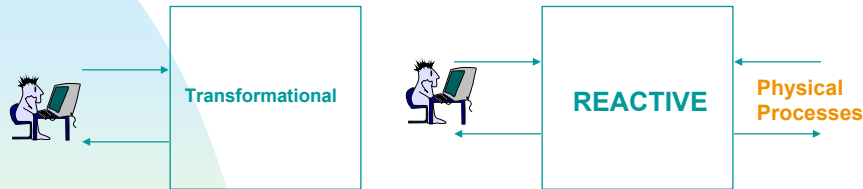
- Functional behavior: what does the system do
  - ◆ in non-embedded systems, this is sufficient
- Contract with the physical world
  - ◆ Time: meet temporal contract with the environment
    - temporal behavior important in real-time systems, as most embedded systems are
    - simple metric such as throughput, latency, jitter
    - more sophisticated quality-of-service metrics
  - ◆ Power: meet constraint on power consumption
    - peak power, average power, system lifetime
  - ◆ Others: size, weight, heat, temperature, reliability etc.

System model must support description of both  
functional behavior and physical interaction

13

Copyright 2001 © Mani Srivastava

## Importance of Time in Embedded Systems: Reactive Operation



Computation is in response to external events

- ◆ periodic events can be statically scheduled
- ◆ aperiodic events harder
  - worst case is an over-design
  - statistically predict and dynamically schedule
  - approximate computation algorithms
- As opposed to Transformation Operation in Interactive Systems

14

Copyright 2001 © Mani Srivastava

## Reactive Operation (contd.)

- Interaction with environment causes problems
  - ◆ indeterminacy in execution
    - e.g. waiting for events from multiple sources
  - ◆ physical environment is delay intolerant
    - can't put it on wait with an hour glass icon!
- Handling timing constraints are crucial to the design of embedded systems
  - ◆ interface synthesis, scheduling etc.
  - ◆ increasingly, also implies high performance

15

Copyright 2001 © Mani Srivastava

## Real Time Operation

- Correctness of result is a function of time it is delivered: the right results on time!
  - ◆ deadline to finish computation
  - ◆ doesn't necessarily mean fast: predictability is important
  - ◆ worst case performance is often the issue
    - but don't want to be too pessimistic (and costly)
- Accurate performance prediction needed

16

Copyright 2001 © Mani Srivastava

## Shades of Real-time

- Hard
  - ◆ the right results late are wrong!
    - On-time = nanoseconds, microseconds, days...
    - Worst case performance is important
  - ◆ catastrophic failure if deadline not met
  - ◆ safety-critical
- Soft
  - ◆ the right results late are of less value than right results on time
    - more they are late, less the value, but do not constitute system failure
    - usually average case performance is important
  - ◆ failure not catastrophic, but impacts service quality
    - e.g. connection timing out, display update in games
  - ◆ most systems are largely soft real-time with a few hard real-time constraints
- (End-to-end) quality of service (QoS)
  - ◆ notion from multimedia/OS/networking area
  - ◆ encompasses more than just timing constraints
  - ◆ classical real-time is a special case

17

Copyright 2001 © Mani Srivastava

## Example: Characterizing Real-time Performance of an OS

- Worst case interrupt disable time: The maximum duration that interrupts are disabled by the operating system, device driver or user software.
- Worst case interrupt dispatch times: The maximum time from the processor recognizing the interrupt to the first instruction in the interrupt handler. There may be multiple such measurements, for different types of handlers, each with more or less capability.
- Worst case kernel non-preemption time: The maximum duration that the kernel disables preemption of the running process to switch to a higher priority executable process.
- Worst case process response time: The maximum time from a process schedule event (whether generated via a hardware interrupt or a software event) in the kernel to execution of the first instruction in the process.

18

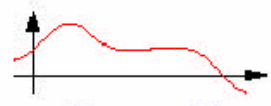
Copyright 2001 © Mani Srivastava

## Timing Constraints

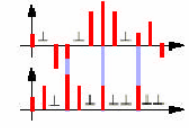
- Timing constraints are the key feature!
  - impose temporal restrictions on a system or its users
  - hard timing constraints, soft timing constraints, interactive
- Questions:
  - What kind of timing constraints are there?
  - How do we arrange computation to take place such that it satisfies the timing constraints?

19 Copyright 2001 © Mani Srivastava

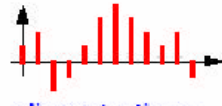
## Many Notions of Time



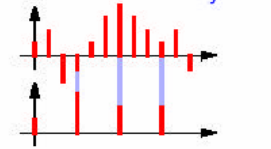
continuous time



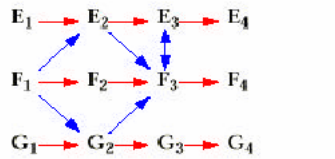
synchronous/reactive



discrete time



multirate discrete time




partially-ordered discrete events

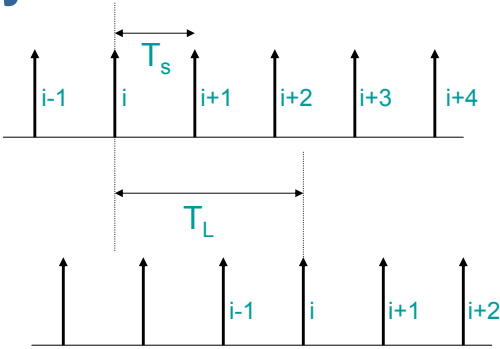
20 Copyright 2001 © Mani Srivastava

## Timing Constraints in a DSP Systems

Input samples



Output samples



- Independent timing constraints:  $T_S$  &  $T_L$



## More General Timing Constraints

- Two categories of timing constraints
  - ◆ **Performance constraints:** set limits on response time of the system
  - ◆ **Behavioral constraints:** make demand on the rate at which users supply stimuli to the system
- Further classification: three types of temporal restrictions (not mutually exclusive)
  - ◆ **Maximum:** no more than  $t$  amount of time may elapse between the occurrence of one event and the occurrence of another
  - ◆ **Minimum:** No less than  $t$  amount of time may elapse between two events
  - ◆ **Durational:** an event must occur for  $t$  amount of time

*Note: "Event" is either a stimulus to the system from its environment, or is an externally observable response that the system makes to its environment*

## Maximum Timing Constraints

- A. S-S combination:** a max time is allowed between the occurrences of two stimuli
  - e.g. 2nd digit must be dialed no later than 20s after the 1st digit
- B. S-R combination:** a max time is allowed between the arrival of a stimulus and the system's response
  - e.g. the caller shall receive a dial tone no later than 2s after lifting the phone receiver
- C. R-S combination:** a max time is allowed between a system's response and the next stimulus from the environment
  - e.g. after receiving the dial tone, the caller shall dial the first digit within 30s
- D. R-R combination:** a max time is allowed between two system's responses
  - e.g. after a connection is made, the caller will receive a ringback tone no more than 0.5s after the callee has received a ring tone

## Observation

- A and C are behavioral requirements
  - constraints on users
  - if the user fails to generate the required stimulus within the prescribed time, the system will take a specific course of action
  - e.g. start a “timer” at the occurrence of the first stimulus (case A) or the system response (case C)
    - “timer alarm” used as an artificial stimulus
- B and D are system performance requirements

## Minimum Timing Constraints

- A. S-S combination:** a min time is required between the occurrences of two stimuli
  - e.g. a min of 0.5s must elapse between the dialing of one digit and dialing of the next
- B. S-R combination:** a min time is required between the arrival of a stimulus and the system’s response
  - e.g. after the caller has dialed 0, the system shall wait 15s before dialing (so that user may complete operator-assisted call himself)
- C. R-S combination:** a min time is required between a system’s response and the next stimulus from the environment
  - e.g. where system can be busy processing requests from several ports, and can’t accept new inputs at this port for a certain time
- D. R-R combination:** a min time must pass between two system’s responses
  - e.g. user may need a certain time to act upon the first response

27

Copyright 2001 © Mani Srivastava

## More Complex Timing Constraints

- Performance constraints on a sequence of responses
  - e.g. caller should dial 7 digits in 30s or less after lifting the receiver
  - express using a timer
- Durational: express duration of a stimulus or response
  - e.g. to get back to the operator, press the button for at least 15s (but not more than 30s); to get the dial tone press the button for more than 30s.
- Two responses  $r_1$  and  $r_2$  should be heard within 60s after a stimulus  $s_1$ , and  $r_2$  should be delayed at least by 15s after  $r_1$  and should occur within 30s after  $r_1$ . Also,  $r_1$  &  $r_2$  should last for 3s and 4s respectively.

28

Copyright 2001 © Mani Srivastava

## Popular Computation Models for Embedded Systems

- Finite State Machines
- Communicating Finite State Machines
- Discrete Event
- Synchronous / Reactive
- Dataflow
- Process Networks
- Rendezvous-based Models (CSP)
- Petri Nets
- Tagged-Signal Model

## How do the models differ?

- State: finite vs. infinite
- Time: untimed, continuous time, partial order, global order
- Concurrency: sequential, concurrent
- Determinacy: determinate vs. indeterminate
- Data value: continuous, sample stream, event
- Communication mechanisms
- Others: composability, availability of tools etc.

## Example: Modeling DSP Systems

<i>Subsystem</i>	<i>Model of Computation</i>
speech/audio processing	1-D dataflow
image processing	1-D/2-D dataflo
image/video resampling	m-D multirate dataflow
user interface	synchronous/reactive
communication protocols	finite state machine
digital control	dataflow
scalable descriptions	process networks

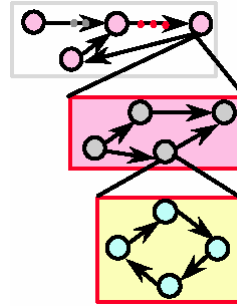
[Evans]

31

Copyright 2001 © Mani Srivastava

## Digression: Graphs as an Abstract Syntax for Models

- Graph = <Nodes, Arcs>
- Many computation models are obtained by ascribing specific interpretations (semantics) to the nodes and arcs, and sometimes by constraining the structure of the graph
  - ◆ e.g.: Petri nets, Data flow, FSM
- Hierarchical graphs offer a useful visual representation

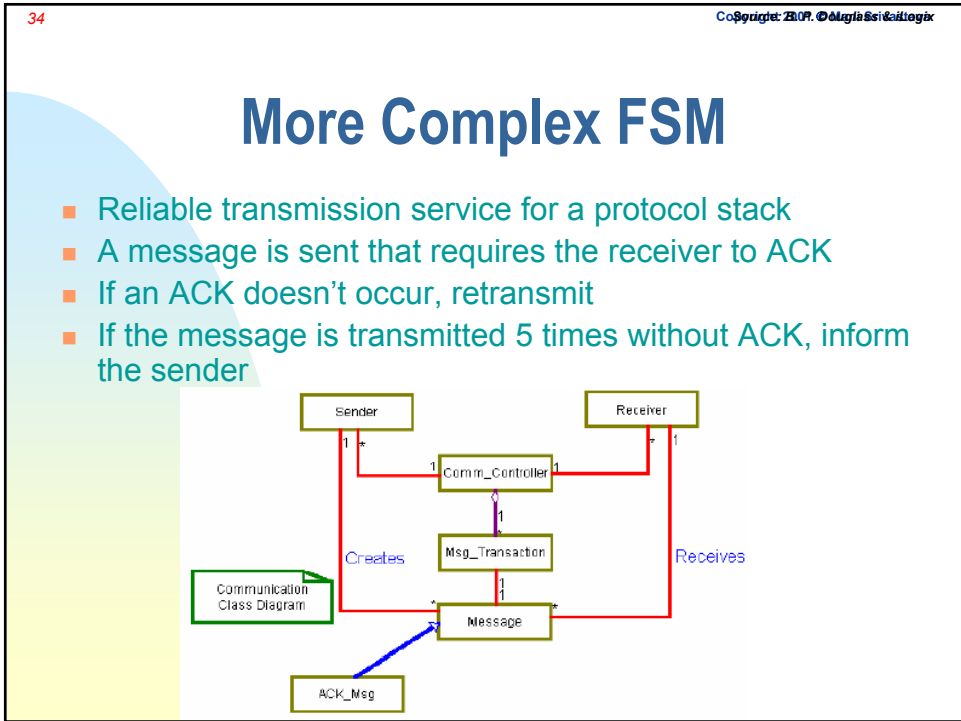
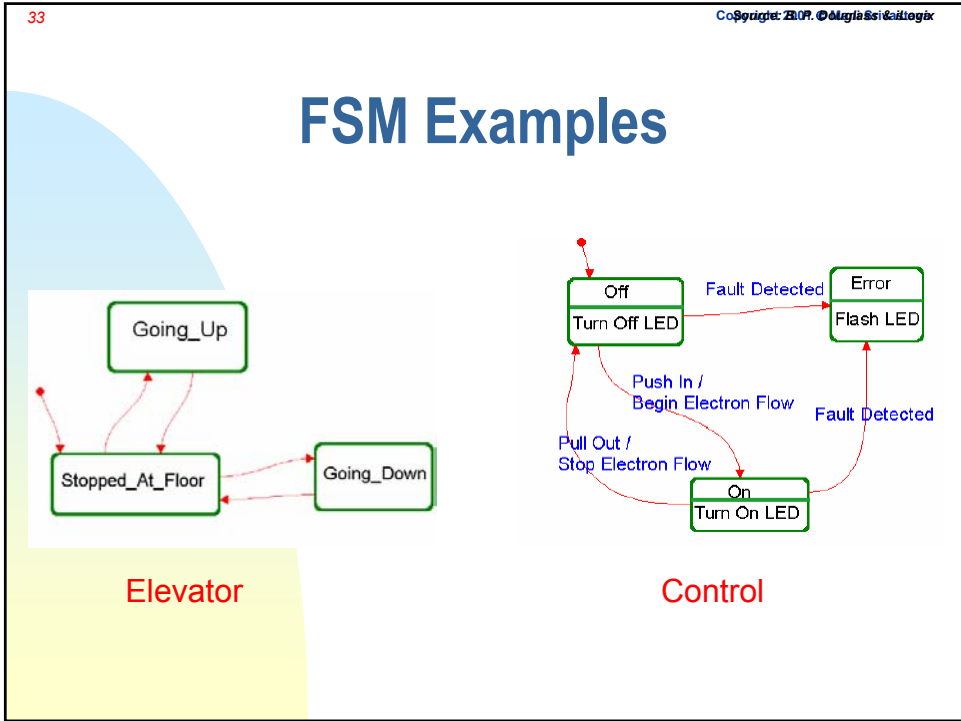


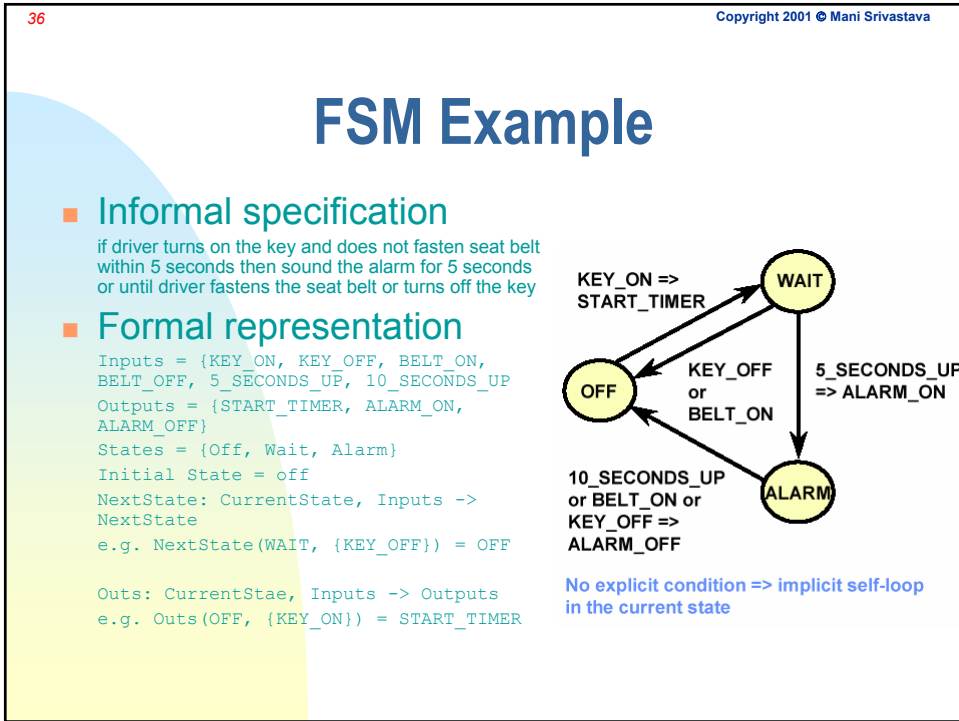
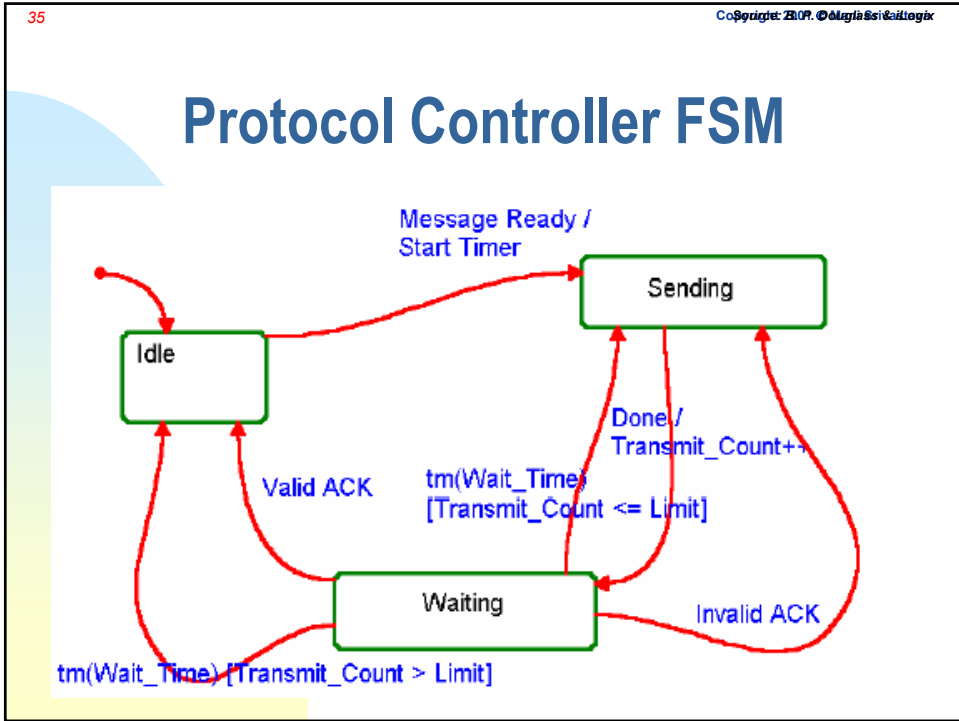
32

Copyright 2001 © Mani Srivastava

## Finite State Machines

- FSM is a mathematical model of a system that assumes:
  - system can be in a finite # of conditions called *states*
  - system behavior within a given state is essentially identical
  - system resides in states for significant periods of time
  - system may change states only in a finite # of well-defined ways, called *transitions*
  - transitions are the response of the system to external or internal *events*
  - Functions or operations called *actions* may be executed when the transition is taken, a state is entered, or a state is exited
    - implemented by an internal or external datapath or object's operations
  - transitions and actions take (approximately) zero time, i.e. instantaneous
    - "synchronous"
  - events not permitted in a state are ignored or result in error or queued
- FSM = (Inputs, Outputs, States, InitialState, NextState, Outs)
- Often suitable for controllers, protocols etc.
- Not Turing Complete, but more amenable to analysis
- Rarely suitable for Memory and Datapaths
- Easy to use with powerful algorithms for synthesis and verification





37

Copyright 2001 © Mani Srivastava

## Non-Deterministic FSM

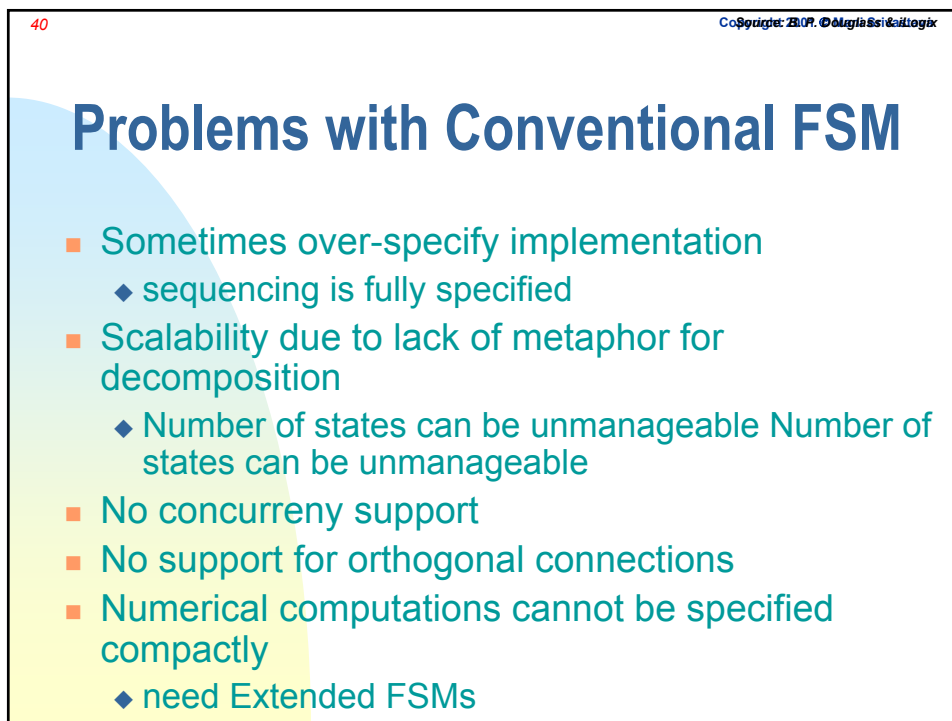
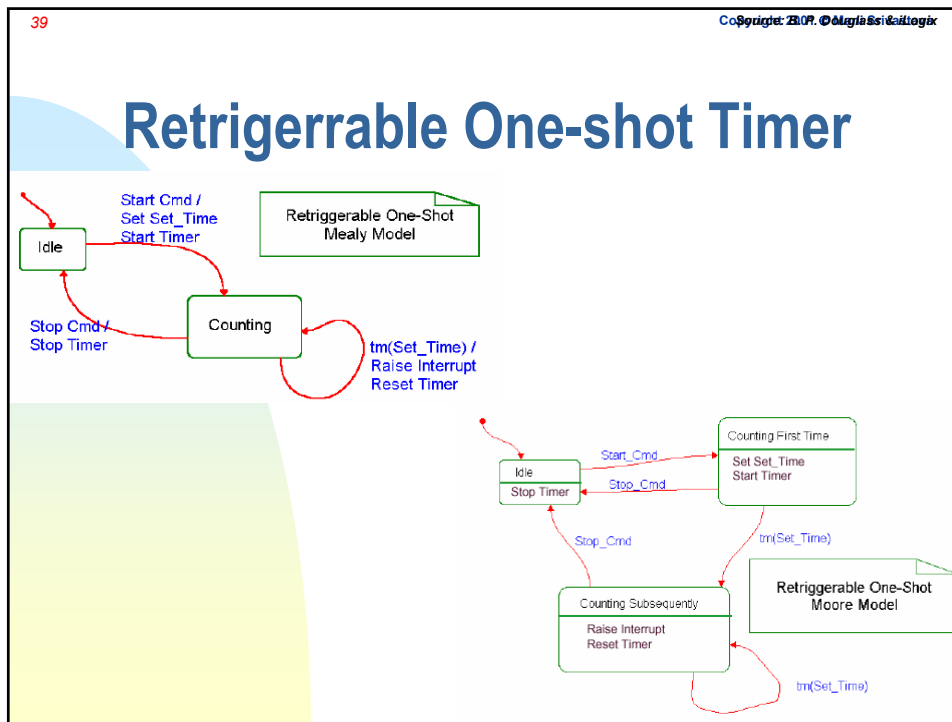
- A FSM is said to be non-deterministic when the NextState and Outs functions may be RELATIONS (instead of functions)
- Non-determinism can be used to model
  - ◆ unspecified behavior
    - incomplete specification
  - ◆ unknown behavior
    - e.g., the environment model
  - ◆ abstraction
    - (the abstraction may result in insufficient detail to identify previously distinguishable situations)

38

Copyright 2001 © Mani Srivastava

## Mealy-Moore FSMs

- The set of states define the state space
- State space are flat
  - ◆ all states are at the same level of abstraction
  - ◆ all state names are unique
- State models are single threaded
  - ◆ only a single state can be valid at any time
- Moore state models: all actions are upon state entry
  - ◆ Non-reactive (response delayed by a cycle)
  - ◆ Easy to compose (always well-defined)
  - ◆ Good for implementation
- Mealy state models: all actions are in transitions
  - ◆ Reactive (0 response time)
  - ◆ Hard to compose: problem with combinational cycles
  - ◆ s/w or h/w must be fast enough: synchronous hypothesis

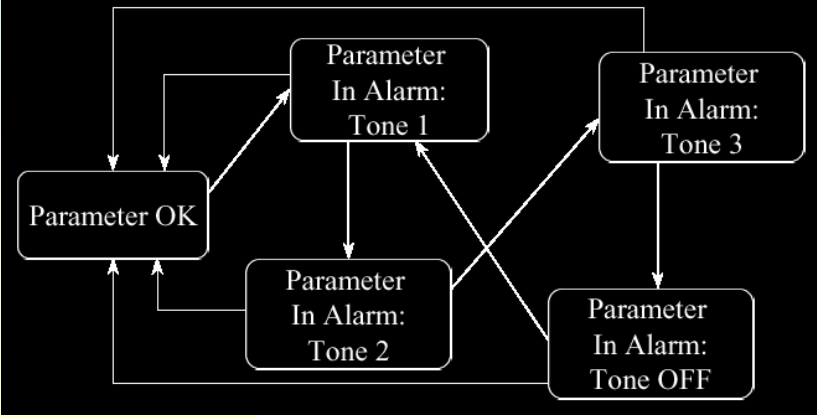


41

Copyright © 2001 by Motorola, Inc.

# Scalability

THIS....

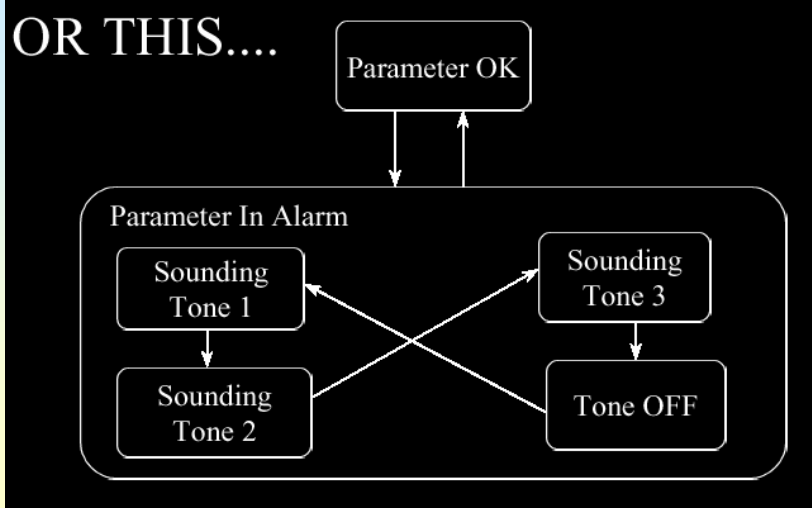


42

Copyright © 2001 by Motorola, Inc.

# Scalability

OR THIS....

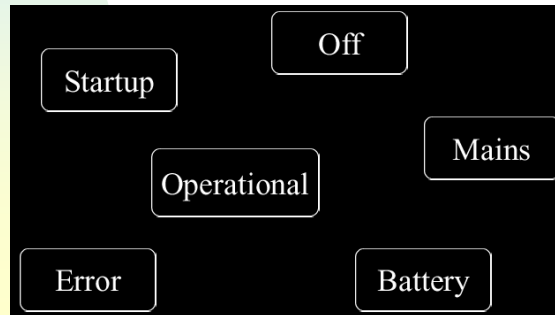


43

Copyright: Zilber, © Morgan Stanley &amp; Co.

# Concurrency

- Problem:
  - ◆ a device can be in states
    - Off, Starting-up, Operational, Error
  - ◆ And, can be running from
    - mains, battery
- How to arrange these states?



44

Copyright: Zilber, © Morgan Stanley &amp; Co.

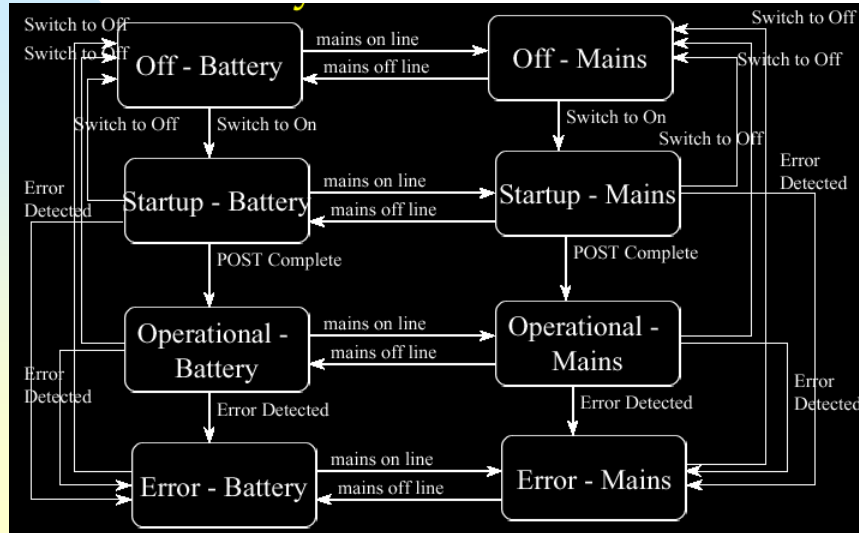
# Concurrency

- Following are different states in Mealy/Moore view:
  - ◆ Operation with battery
  - ◆ Operation with mains
- Leads to state explosion
- Solution?
  - ◆ Allow states to operate concurrently

45

Copyright: ZILIF. © MegisSi&illogix

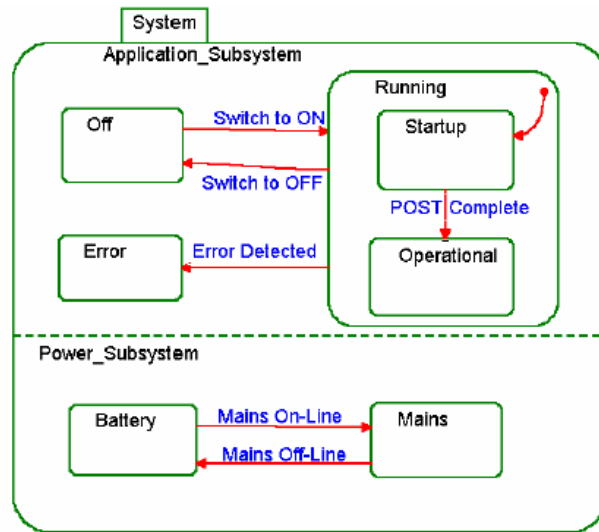
# Mealy-Moore Solution



46

Copyright: ZILIF. © MegisSi&illogix

# Concurrent State Model Solution

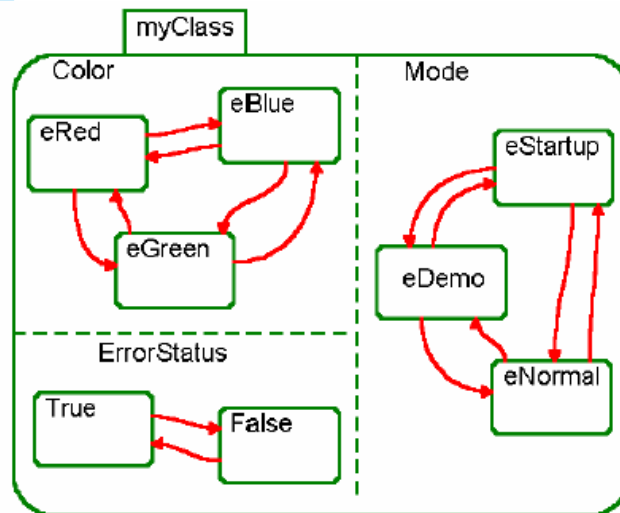




49

Copyright 2001 © Mani Srivastava

## Approach 2



50

Copyright 2001 © Mani Srivastava

## Harel's StateCharts: Extension of Conventional FSMs

- Conventional FSMs are inappropriate for the behavioral description of complex control
  - flat and unstructures
  - inherently sequential in nature
  - give rise to an exponential blow-up in # of states
    - small system extensions cause unacceptable growth in the number of states to be considered
- StateCharts support:
  - *repeated decomposition* of states into AND/OR sub-states
    - nested states, concurrency, orthogonal components
  - *actions* (may have parameters)
  - *activities* (functions executed as long as state is active)
  - *guards*
  - *history*
  - a *synchronous* (instantaneous broadcast) comm. mechanism

51

Copyright 2001 © Mani Srivastava

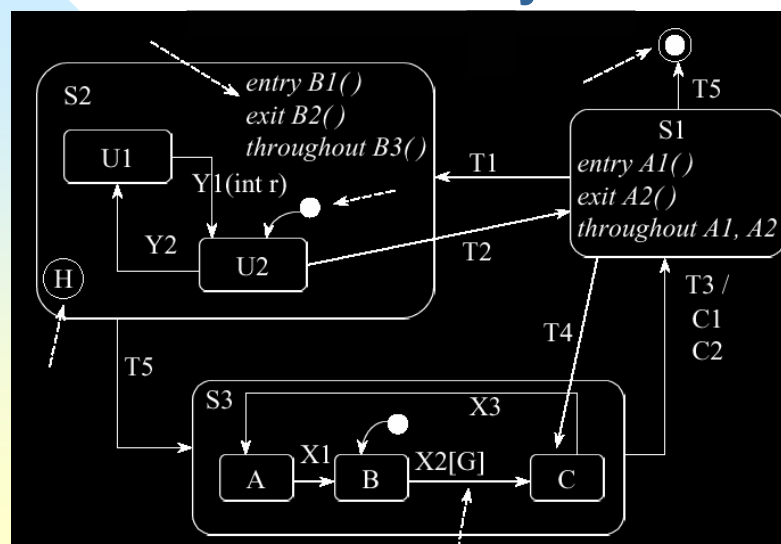
## Features of StateCharts

- Nested states and hierarchy
  - ◆ Improves scalability and understandability
  - ◆ helps describing preemption
- Concurrency - two or more states can be viewed as simultaneously active
- Nondeterminism - there are properties which are irrelevant

52

Copyright 2001 © Mani Srivastava

## Basic Harel Syntax



53

Copyright 2001 © Mani Srivastava

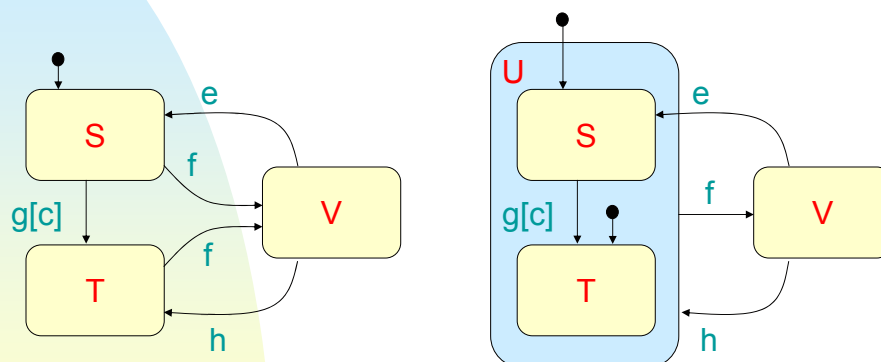
## State Decomposition

- **OR-states** have sub-states that are related to each other by *exclusive-or*
- **AND-states** have *orthogonal state components* (synchronous FSM composition)
  - AND-decomposition can be carried out on any level of states
    - more convenient than allowing only one level of communicating FSMs
- **Basic States:** no sub-states (bottom of hierarchy)
- **Root State:** no parent states (top of hierarchy)

54

Copyright 2001 © Mani Srivastava

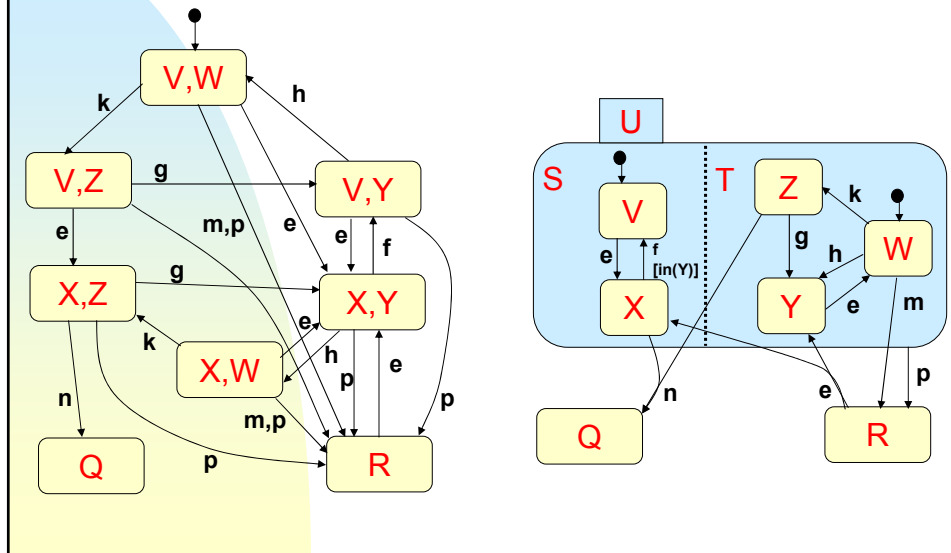
## StateChart OR-decomposition



55

Copyright 2001 © Mani Srivastava

## StateChart AND-decomposition



56

Copyright 2001 © Mani Srivastava

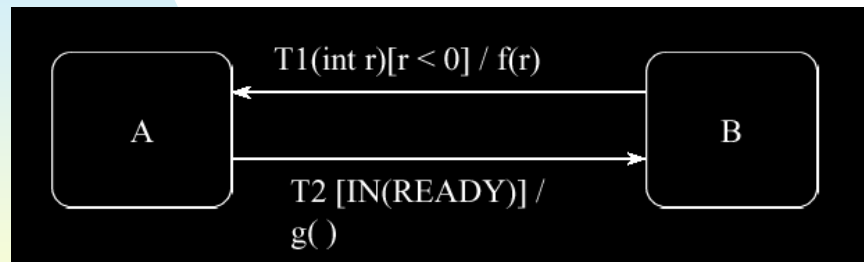
## StateCharts Syntax

- The general syntax of an expression labeling a transition in a StateChart is  $n[c]/a$ , where
  - $n$  is the **event** that triggers the transition
  - $c$  is the **condition** that guards the transition (cannot be taken unless  $c$  is true when  $e$  occurs)
  - $a$  is the **action** that is carried out if and when the transition is taken
- Alternative:  $name(params)[guards]^event\_list/action\_list$ 
  - Event list, aka propagated transitions, is a list of transitions that occur in other concurrent state machines because of this transitions
- For each transition label, event condition and action are optional
  - an event can be the changing of a value
  - standard comparisons are allowed as conditions and assignment statements as actions

57

Copyright 2001 © Mani Srivastava

## Transitions



58

Copyright 2001 © Mani Srivastava

## StateCharts Actions and Events

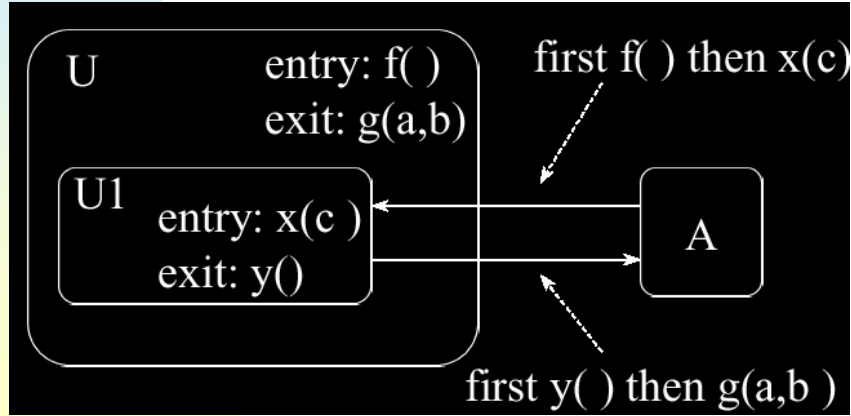
- An action *a* on the edge leaving a state may also appear as an event triggering a transition going into an orthogonal state
  - executing the first transition will immediately cause the second transition to be taken simultaneously
- Actions and events may be associated to the execution of orthogonal components:
  - action *start(A)* causes activity A to start
  - event *stopped(B)* occurs when activity B stops
  - *entered(S), exited(S), in(S)* etc.

59

Copyright: ZILP, © Magnus & Logix

## Order of Nested Actions

- Executed from outermost – in on entry
- Executed from innermost – out on exit

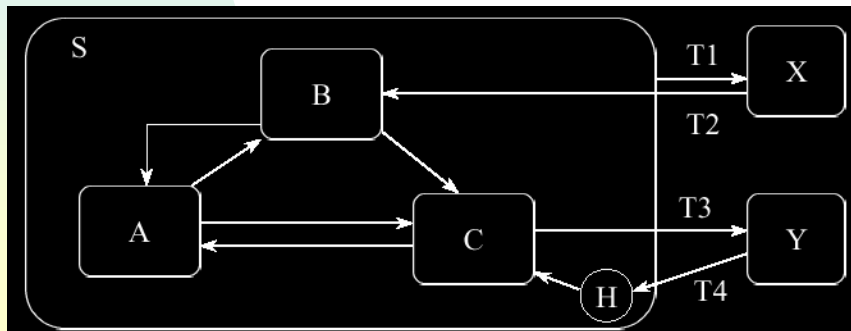


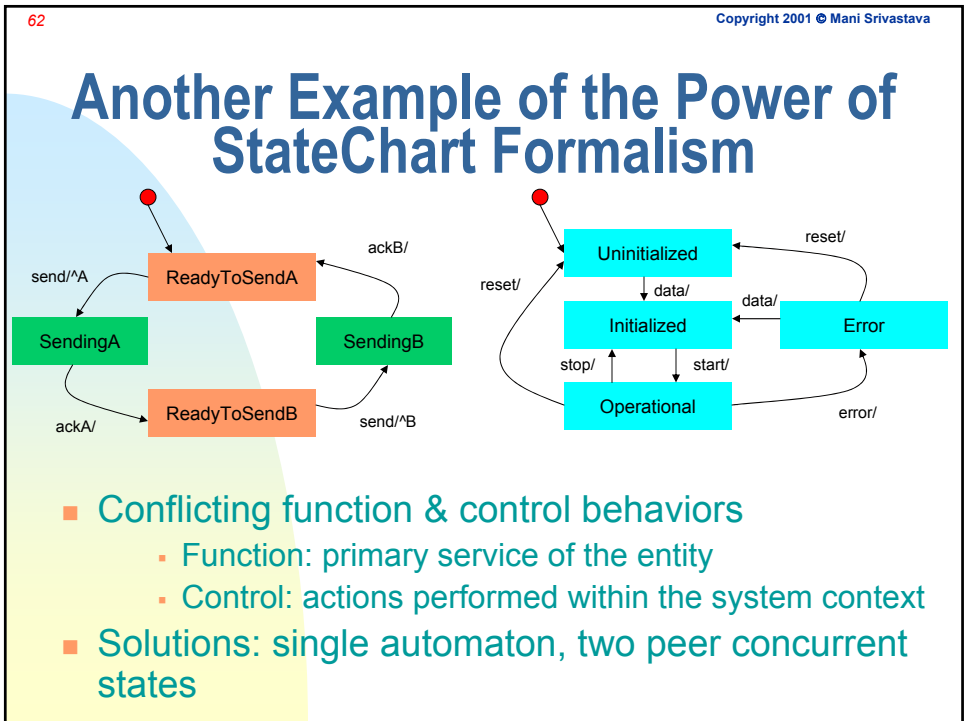
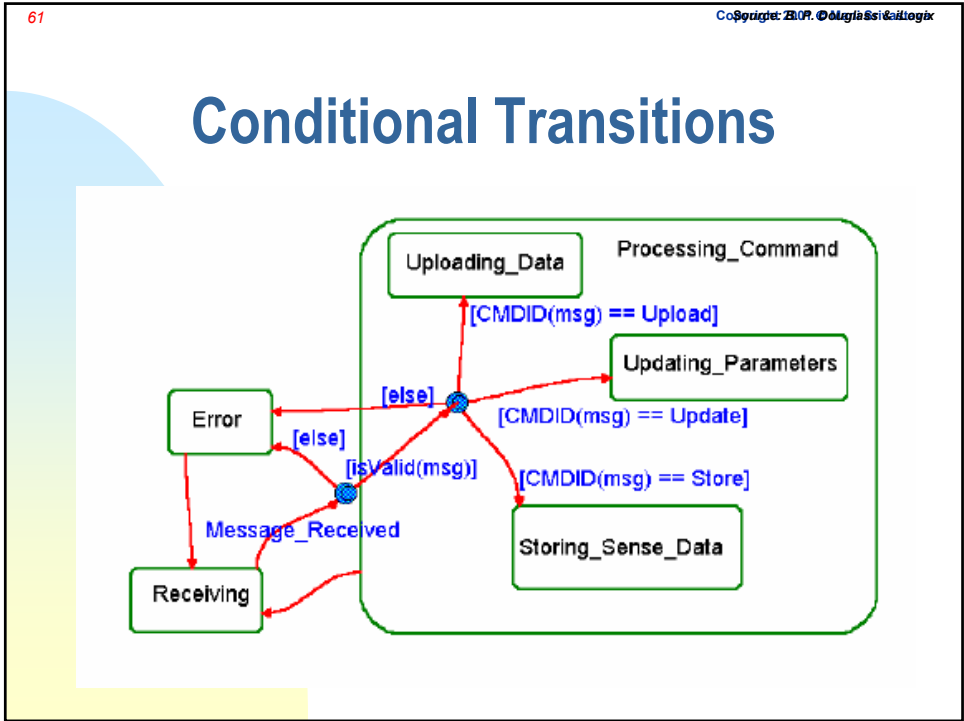
60

Copyright: ZILP, © Magnus & Logix

## History

- The history annotation (H) means that the state “remembers” the substate and returns to it as the default
- Can also work with an initial state indicator

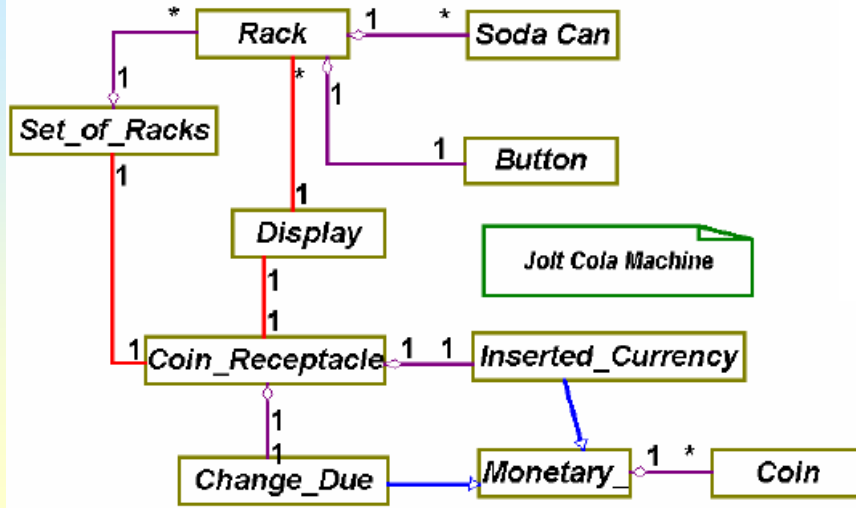




63

Copyright 2001 © Mani Srivastava

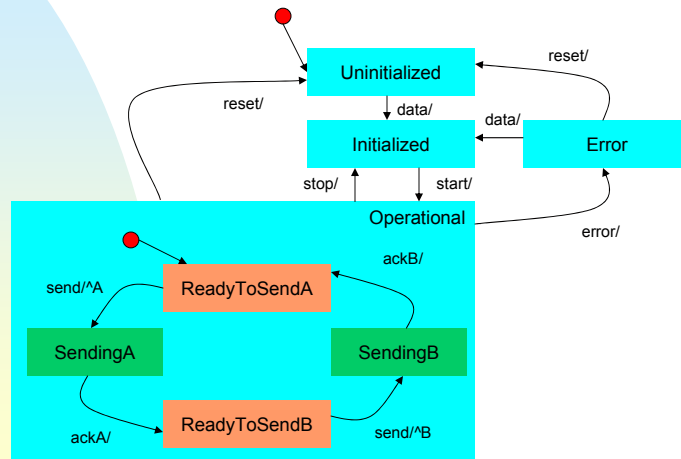
## Example: Jolt Cola Machine



64

Copyright 2001 © Mani Srivastava

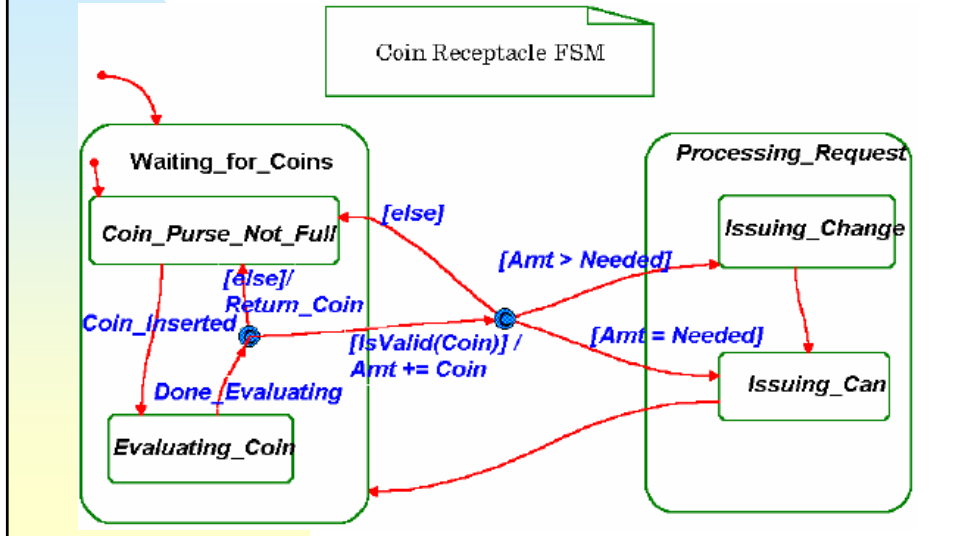
## The Combined State Machine in StateChart Formalism



65

Copyright: ZILIF. © Morgan Stanley & Co. Inc.

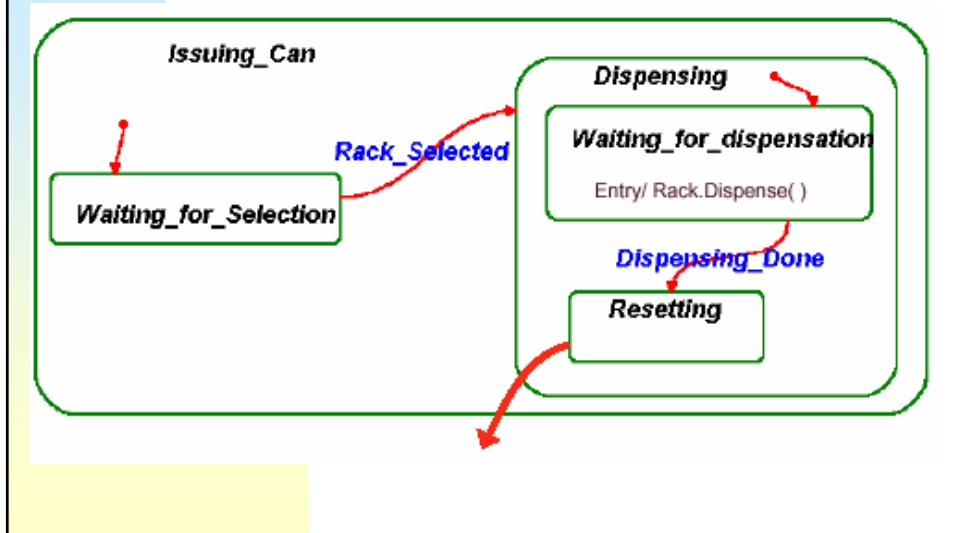
# Example: Coin Receptacle FSM

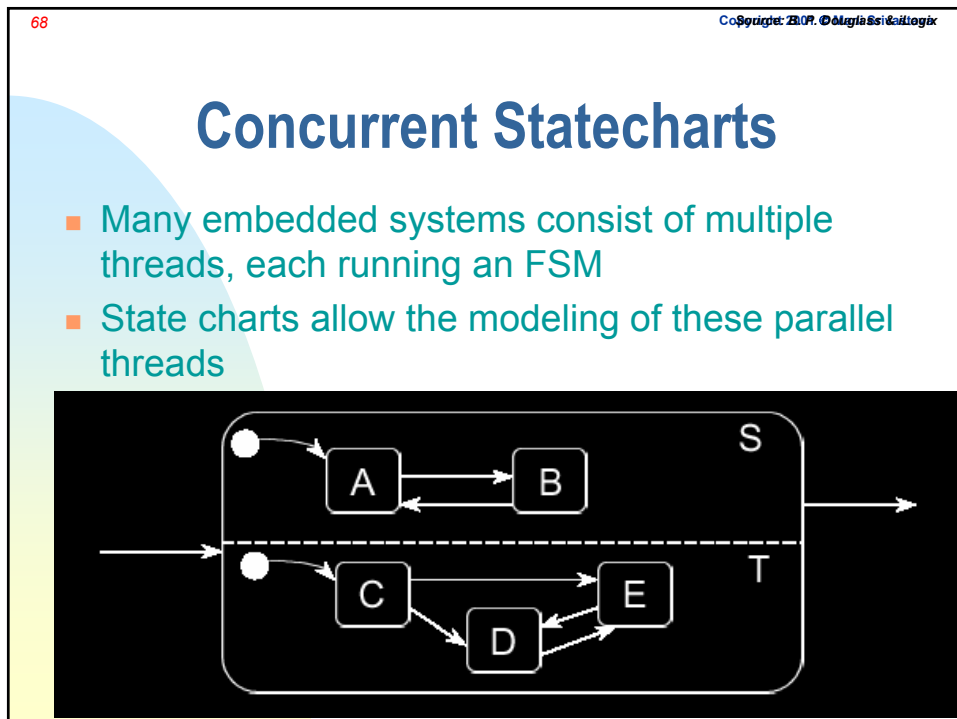
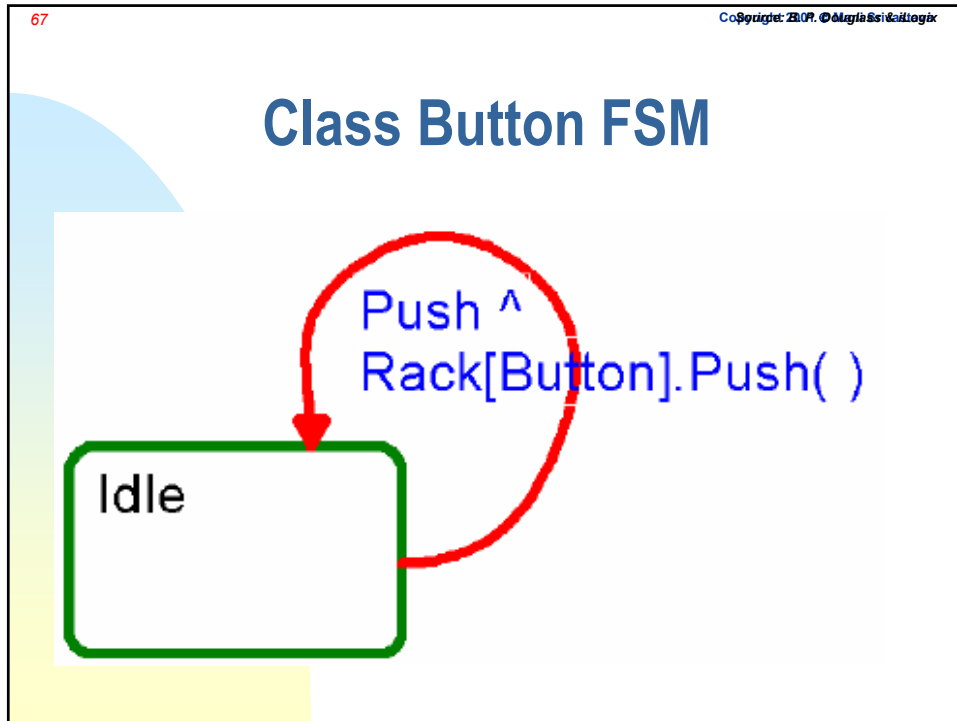


66

Copyright: ZILIF. © Morgan Stanley & Co. Inc.

# Substate: Issuing Can



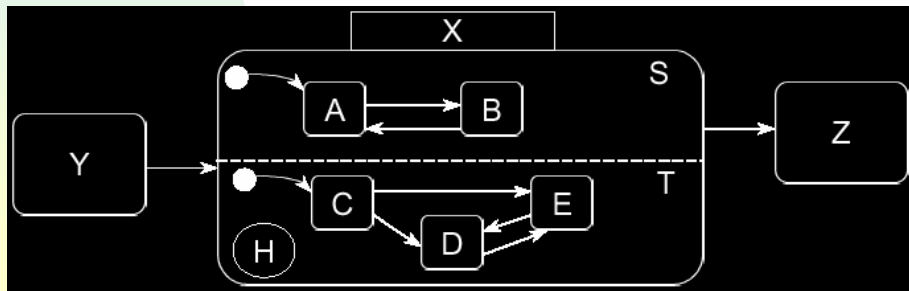


69

Copyright: ZILP, © Morgan Stanley &amp; Co.

## Concurrent Statecharts

- States S and T are active at the same time as long as X is active
  - ◆ Either S.A or S.B must be active when S is active
  - ◆ Either T.C, T.D or T.E must be active when T is active

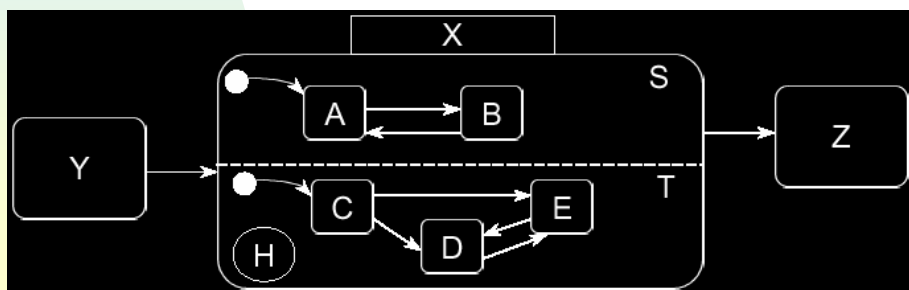


70

Copyright: ZILP, © Morgan Stanley &amp; Co.

## Concurrent Statecharts

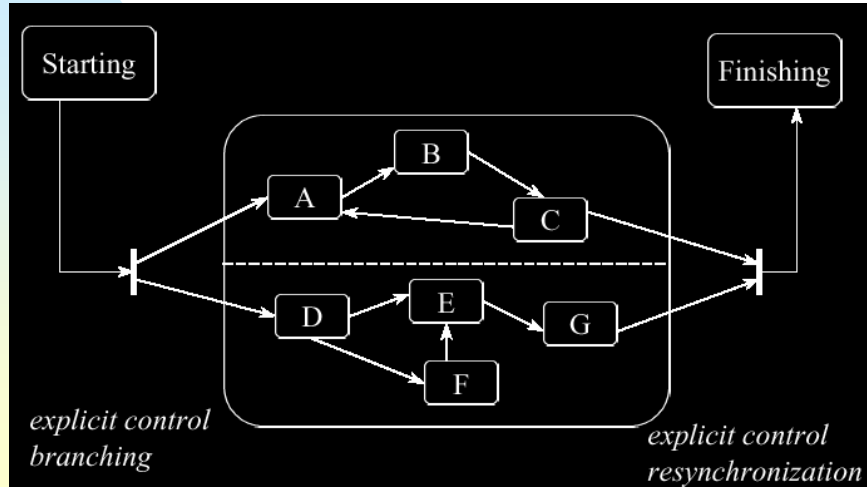
- When X exits, both S and T exit
  - ◆ If S exits first, the FSM containing X must wait until T exits
  - ◆ If the two FSMs are always independent, then they must be enclosed at the highest scope



71

Copyright © Morgan Stanley & Co.

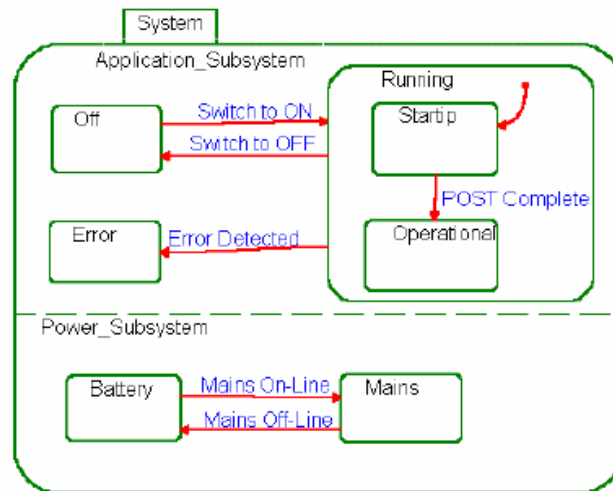
# Explicit Synchronization



72

Copyright © Morgan Stanley & Co.

# Example Concurrent FSM



73

Copyright 2001 © Mani Srivastava

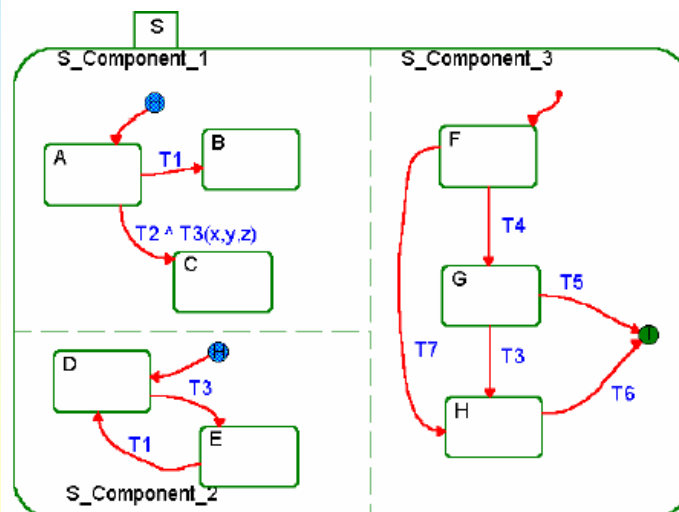
## Communication in Concurrent FSMs

- Broadcast events
  - ◆ Events are received by more than one concurrent FSM
  - ◆ Results in transitions of the same name in different FSM
- Propagated transitions
  - ◆ Transitions which are generated as a result of transitions in other FSMs

74

Copyright 2001 © Mani Srivastava

## Propagations and Broadcasts



75

Copyright 2001 © Mani Srivastava

## Graphical Hierarchical FSM Languages

- Several commercial & non-commercial variants
  - StateCharts, VisualHDL, SpecCharts, SpeedChart, StateVision, etc.
- Easy to use in control dominated systems
- Simulation, S/W and H/W synthesis
- Extended with arithmetic

76

Copyright 2001 © Mani Srivastava

## Rhapsody, StateMate etc. by i-Logix

- Set of tools for development of complex reactive systems
  - heavy graphical orientation
  - captures system behavior using StateCharts
  - rigorous executions and simulations
  - create run-time code for rapid-prototyping
- Key features:
  - executable and analyzable specifications
  - visual formalisms

77

Copyright 2001 © Mani Srivastava

## Co-design Finite State Machines (CFSMs)

- Underlying model of computation for Berkeley's POLIS system for H/W-S/W synthesis
- Combines aspects of several other models
  - FSM extended with support for data handling and asynchronous communications
    - FSM part: I/O, states, transition & output relation
    - Data computation part: external, instantaneous functions without side effects
- Mixes
  - synchronicity: zero & infinite delay
  - asynchronicity: non-zero, finite, & bounded delay
- Embedded systems have both aspects

78

Copyright 2001 © Mani Srivastava

## Network of CFSMs

- Interacting CFSMs that communicate through a very low-level primitive: *events*
- Broadcast communication model
  - ◆ a CFSM, or its environment, emits events
  - ◆ one or more CFSMs or the environment can later detect it
- GALS: globally asynchronous, locally synchronous
  - ◆ contrast: in concurrent FSMs, all FSMs change states exactly at the same time
    - does not work in S/W with interleaved FSMs

79

Copyright 2001 © Mani Srivastava

## Communication Primitives

- Events implement a communication protocol that does not require acknowledgment
  - ◆ receiver waits for sender to emit event
  - ◆ sender can proceed immediately after emission
  - ◆ implicit one-place buffer between the sender and receiver saves the event until it is detected or overwritten
    - efficient H/W implementation with synchronous circuits, and S/W implementation using polling or interrupts
- Handshake, if needed, modeled using events

80

Copyright 2001 © Mani Srivastava

## More on Events

- Sender does not remove the event immediately after sending it, only when emitting the next one
- Each CFMSM can detect an event at most once any time after the event's emission, until another event of the same type overwrites it
  - ◆ event can be correctly received even with the unpredictable reaction time associated with S/W implementations
  - ◆ correct reception is ensured as long as
    - sender data rate < receiver processing ability
    - or, explicit synchronization is done

81

Copyright 2001 © Mani Srivastava

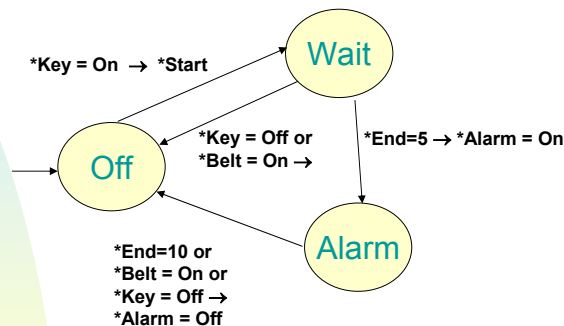
## Generalization to Signals

- Signals carry information in the form of events and/or values
  - Event signals: present/absent
  - Data signals: arbitrary values
    - event, data may be paired
- 1 input buffer / signal / receiver
- Signal is emitted by sender (by setting buffer)
- Consumed by receiver (by resetting buffer)
- “Present” if emitted but not consumed

82

Copyright 2001 © Mani Srivastava

## Example CFMSM Specification



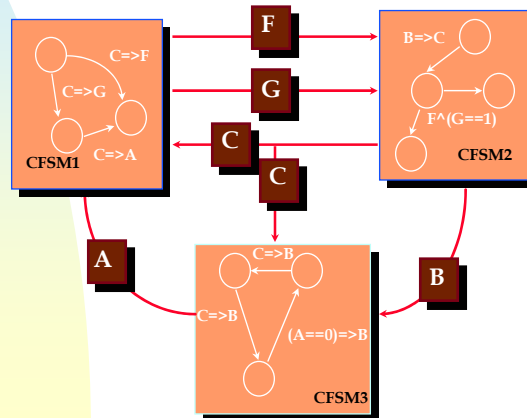
- State of a CFMSM includes those events that are at the same time input and output for it
  - non-zero reaction time implies storage capability

83

Copyright 2001 © Mani Srivastava

## Network of CFMSs: Depth-1 Buffers

- Globally Asynchronous, Locally Synchronous (GALS) model



84

Copyright 2001 © Mani Srivastava

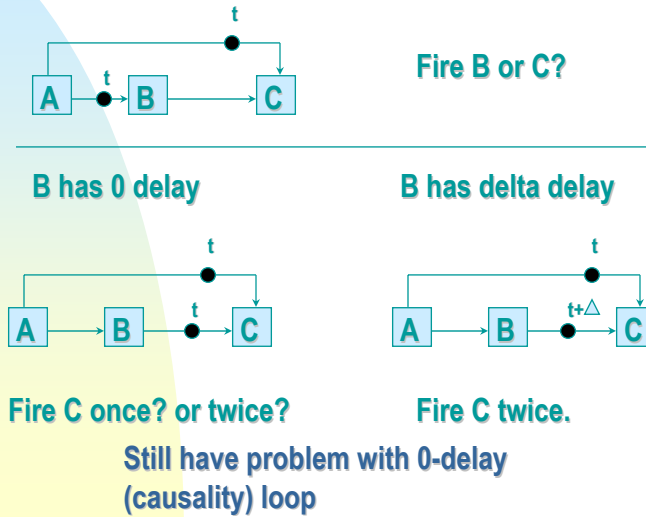
## Discrete Events

- Notion of time is fundamental: global order
  - ◆ events are objects which carry ordered time info
  - ◆ there is a casual relationship between events
- DE simulator maintains global event queue
  - ◆ Verilog and VHDL
- Expensive - ordering time stamps can be time consuming
- Large state & Low Activity => Effective simulation
- Simultaneous events lead to non-determinacy
  - ◆ require complex conflict resolution schemes
  - ◆ e.g. delta delays

85

Copyright 2001 © Mani Srivastava

## Simultaneous Events in the Discrete Event Model



86

Copyright 2001 © Mani Srivastava

## Reactive Synchronous Models

- Assumptions
  - ◆ system reacts to internal and external events by emitting other events
  - ◆ events can only occur at discrete time steps
  - ◆ reactions are assumed to be instantaneous
    - communication by shared variables that are read & written in zero time
    - Communication/computation happen instantaneously
      - negligible or relatively small time to process event
      - if processing is significant, start & end events associated with tasks
- Synchronous languages
  - ◆ Imperative: Estrel, Statechart
  - ◆ Dataflow: Lustre, Signal, Argos
- Simple, clean semantics (FSM based), deterministic behavior, and lots of tools

87

Copyright 2001 © Mani Srivastava

## Esterel

- Describes collection of FSMs
- Imperative syntax
  - S1;S2 executes in sequence
  - S1 || S2 in parallel
  - [S1||S2] in parallel until both terminate
- Succinct specification of interrupts
  - do <body> watching <event>
    - the <body> is executed until either it terminates, or the <event> occurs
- As the number of signals to watch increases, the size of the Esterel program grows linearly, while the FSM complexity grows exponentially

88

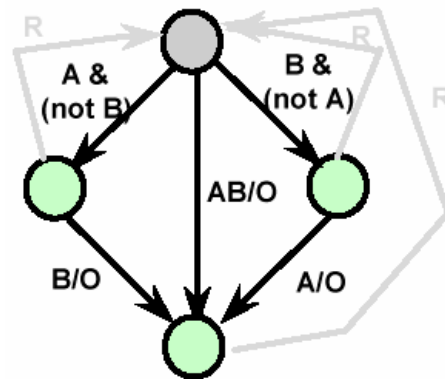
Copyright 2001 © Mani Srivastava

## Esterel Example

```

module EsterelFSM
  input A, B, R;
  output O;
  loop
    do
      [await A ||
       await B];
      emit O;
      halt;
    watching R
  end loop
end module;

```



89

Copyright 2001 © Mani Srivastava

## Data Flow Based Models

- Powerful formalism for data-dominated system specification
- Partially-ordered model (no over-specification)
- Deterministic execution independent of scheduling
- Used for simulation, scheduling, memory allocation, code generation for DSP (h/w & s/w)
- Syntax & semantics: actors, tokens, firings
- Variants: static dataflow, boolean dataflow, dynamic dataflow etc.
- Several recent implementations
  - ◆ Graphical: Ptolemy, Khoros, SPW, Labview
  - ◆ Textual: Silage

90

Copyright 2001 © Mani Srivastava

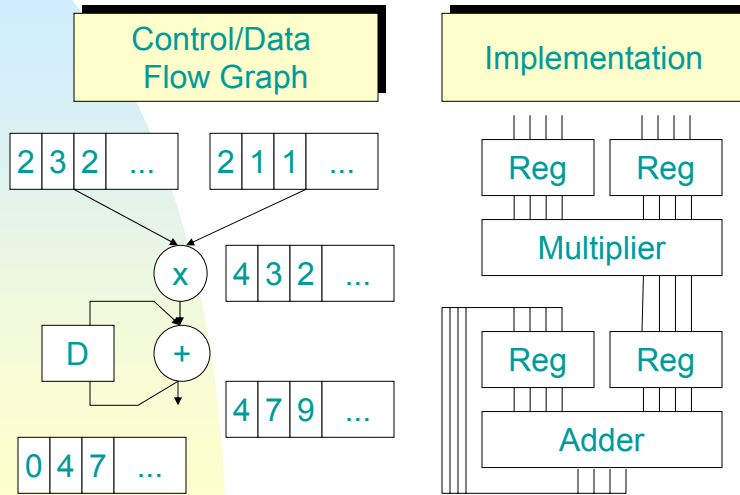
## Data Flow Network

- A Data-flow network is a collection of functional nodes which are connected and communicate over unbounded FIFO queues
  - ◆ Nodes are commonly called actors
  - ◆ The bits of information that are communicated over the queues are commonly called tokens
- (Often stateless) actors perform computation
- Unbounded FIFOs perform communication via *sequences of tokens* carrying values
  - ◆ int, float, fixed point, matrix of int, float, fixed point, image of pixels
- State implemented as self-loop
- Determinacy:
  - ◆ unique output sequences given unique input sequences
  - ◆ Sufficient condition: *blocking read* (cannot test input queues for emptiness)

91

Copyright 2001 © Mani Srivastava

## Flow Graph Models



92

Copyright 2001 © Mani Srivastava

## Semantics

- At each time, one actor is fired
- When firing, actors consume input tokens and produce output tokens
- Actors can be fired only if there are enough tokens in the input queues

93 Copyright 2001 © Mani Srivastava

## Synchronous Data Flow

The diagram shows two representations of a synchronous data flow graph. On the left, a vertical sequence of operations: addition (+), subtraction (-), addition (+), subtraction (-), addition (+), subtraction (-), addition (+), subtraction (-). Each operation is followed by a multiplication by X. On the right, a block diagram showing two delay elements (D) and two multiplication elements (X) connected to the operations.

94 Copyright 2001 © Mani Srivastava

## What Latency & Sample Period can a SDFG achieve?

The diagram shows a SDFG with input  $x$  and output  $y$ . It contains two delay elements ( $D1$ ,  $D2$ ) and several operations. Red arrows highlight the longest paths for  $T_L$  and  $T_S$ .

Longest Path for  $T_S$   
 $T_S = m + 2$

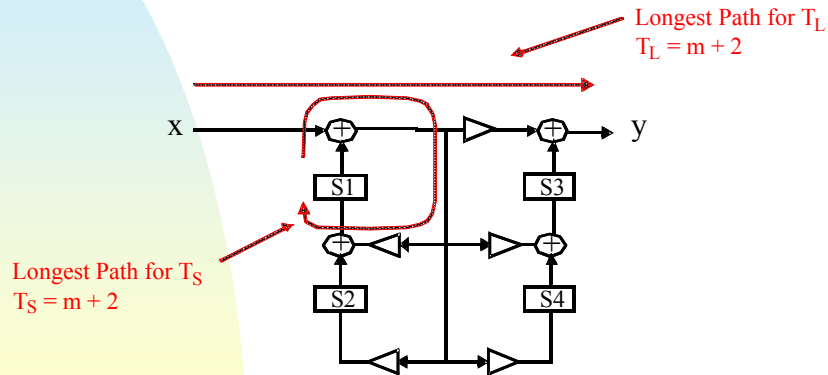
Longest Path for  $T_L$   
 $T_L = 2m + 3$

- $T_L = \max(\text{I-O path, D-O path})$
- $T_S = \max(\text{D-D path, I-D path})$

95

Copyright 2001 © Mani Srivastava

## SDFG can be Transformed to Affect $T_L$ & $T_S$

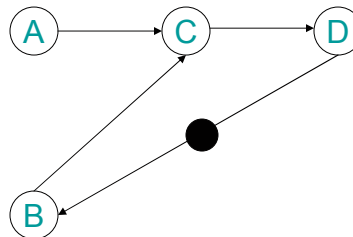


96

Copyright 2001 © Mani Srivastava

## Dataflow Process Networks

- Graph: Nodes (actors) are computations, edges (arcs) are ordered sequences of events (tokens)
- Kahn process network is generalization where unbounded FIFO buffering is available
- Firing - quant of computation
- Karp-Miller computation graphs, Lee-Messerschmidt's synchronous SDF, ...



97

Copyright 2001 © Mani Srivastava

## Synchronous vs. Asynchronous

- Synchronous: StateChart, Esterel
  - ◆ communication by shared variables that are read & written in 0 time
  - ◆ communication and computation happens instantaneously at discrete time instants (e.g. clock ticks & 0-delay computation)
  - ◆ all FSMs make a transition simultaneously (lock step)
    - simultaneous signals: ordering imposed by dependencies
  - ◆ may be difficult to implement
    - verify synchronous assumption on final design
    - Cycle time > max computation time
- Asynchronous FSMs
  - ◆ free to proceed independently
  - ◆ do not execute a transition at the same time (except for rendezvous)
  - ◆ may need to share notion of time: synchronization
  - ◆ easy to implement but difficult to analyze

98

Copyright 2001 © Mani Srivastava

## Asynchronous Communications

- Blocking vs. non-Blocking
  - ◆ Blocking read
    - process can not test for emptiness of input
    - must wait for input to arrive before proceed
  - ◆ Blocking write
    - process must wait for successful write before continue
  - ◆ blocking write/blocking read (CSP, CCS)
  - ◆ non-blocking write / blocking read (FIFO, CFSMs, SDL)
  - ◆ non-blocking write / non-blocking read (shared variables)
- Buffers used to adapt when sender and receiver have different rates
  - ◆ what size?
- Lossless vs. lossy
  - ◆ events/tokens may be lost
  - ◆ bounded memory: overwrite or overflow
  - ◆ need to block the sender
- Single vs. multiple read
  - ◆ result of each write can be read at most or several times

99

Copyright 2001 © Mani Srivastava

## Communication Models

	Transmitters	Receivers	Buffer Size	Blocking Reads	Blocking Writes	Single Reads
Unsynchronized	many	many	one	no	no	no
Read-Modify-write	many	many	one	yes	yes	no
Unbounded FIFO	one	one	unbounded	yes	no	yes
Bounded FIFO	one	one	bounded	no	maybe	yes
Single Rendezvous	one	one	one	yes	yes	yes
Multiple Rendezvous	many	many	one	no	no	yes

[Vincentelli]

100

Copyright 2001 © Mani Srivastava

## Modeling Approaches based on Software Design Methods

- No systematic design in 60s
- From 70s, many different s/w design strategies
  - ◆ Design methods based on functional decomposition
    - Real-Time Structured Analysis and Design (RTSAD)
  - ◆ Design methods based on concurrent task structuring
    - Design Approach for Real-Time Systems (DARTS)
  - ◆ Design methods based on information hiding
    - Object-Oriented Design method (OOD)
  - ◆ Design methods based on modeling the domain
    - Jackson System Development method (JSD)
    - Object-Oriented Design method (OOD)
- UML is the latest manifestation
  - ◆ becoming prevalent in complex embedded system design

101

Copyright 2001 © Mani Srivastava

## How Models Influence an Application Design?

- Example: given input from a camera, digitally encode it using MPEG II encoding standards.
  - this task involves: storing the image for processing going through a number of processing steps, e.g., Discrete cosine transform (DCT), Quantization, encoding (variable length encoding), formatting the bit stream, Inverse Discrete Cosine transform (IDCT), ...
- Is this problem appropriate for
  - Reactive Systems, Synchronous Data flow, CSP, ...
- More than one model be reasonable.

102

Copyright 2001 © Mani Srivastava

## Choice of Model

- Model Choice: depends on
  - ◆ application domain
    - DSP applications use data flow models
    - Control applications use finite state machine models
    - Event driven applications use reactive models
  - ◆ efficiency of the model
    - in terms of simulation time
    - in terms of synthesized circuit/code.
- Language Choice: depends on
  - ◆ underlying semantics
    - semantics in the model appropriate for the application.
  - ◆ available tools
  - ◆ personal taste and/or company policy