



CS684

Embedded Systems (Software)

Models and Tools for Embedded Systems

Kavi Arya
CSE/ IIT Bombay

Problems with FSMs

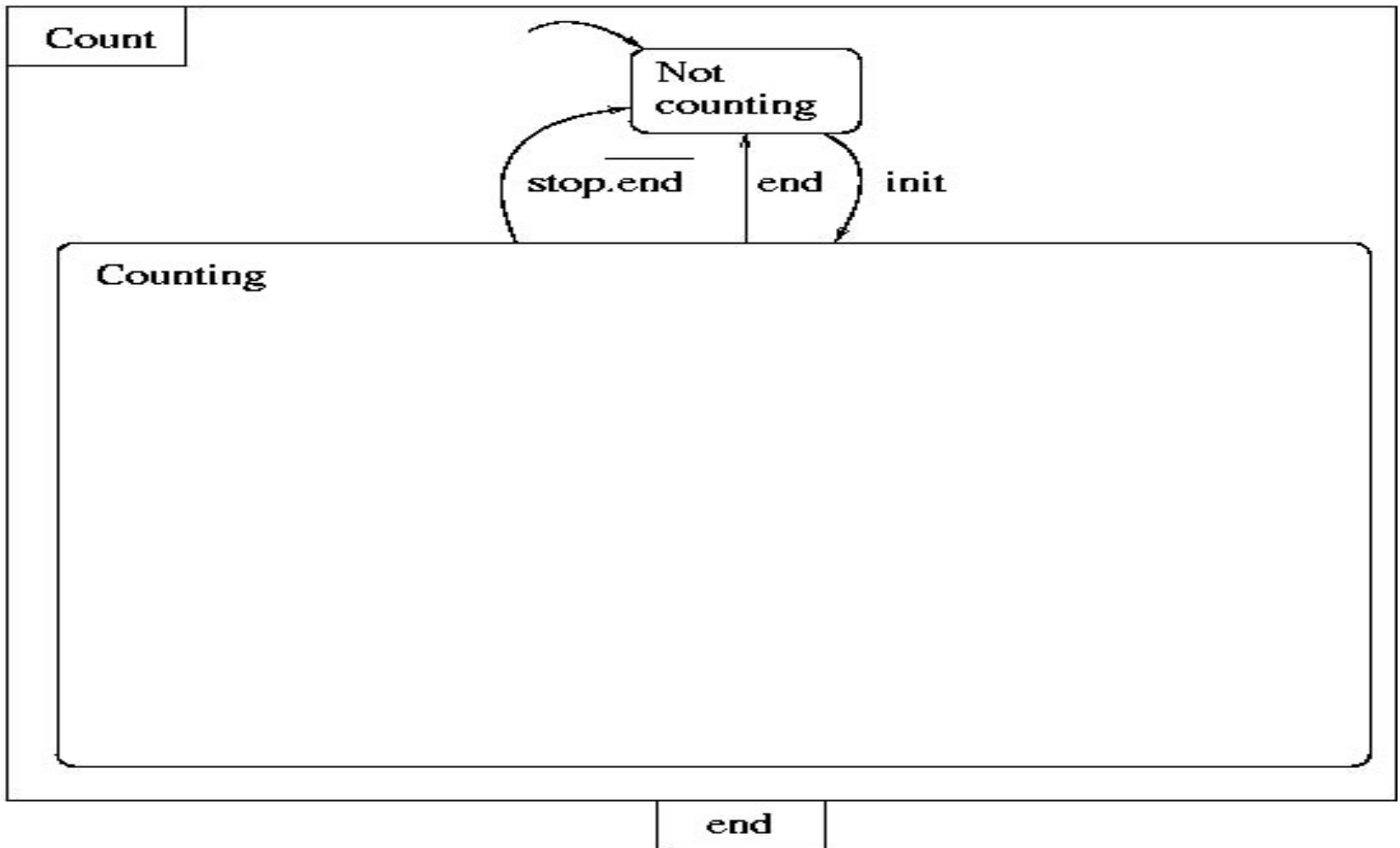
- All is not well with FSMs
- FSMs fine for small systems (10s of states)
- Imagine FSM with 100s and 10^{20} of states which is a reality
- Such large descriptions difficult to understand
- FSMs are flat and no structure
- Inflexible to add additional functionalities
- Need for structuring and combining different state machines

Statecharts

- Extension of FSMs to have these features
- Due to David Harel
- Retains the nice features
 - Pictorial appeal
 - States and transitions
- Enriched with two features
 - Hierarchy and Concurrency
- States are of two kinds
 - OR state (Hierarchy)
 - AND state (concurrency)

OR States

- An OR state can have a whole state machine inside it
- Example:



OR states

- When the system is in the state **Count**, it is either in **counting** or **not_counting**
- Exactly in ONE of the inner states
- Hence the term OR states
(more precisely XOR state)
- When **Count** is entered, it will enter **not_counting**
 - **default state**
- Inner states can be OR states (or AND states)

OR states

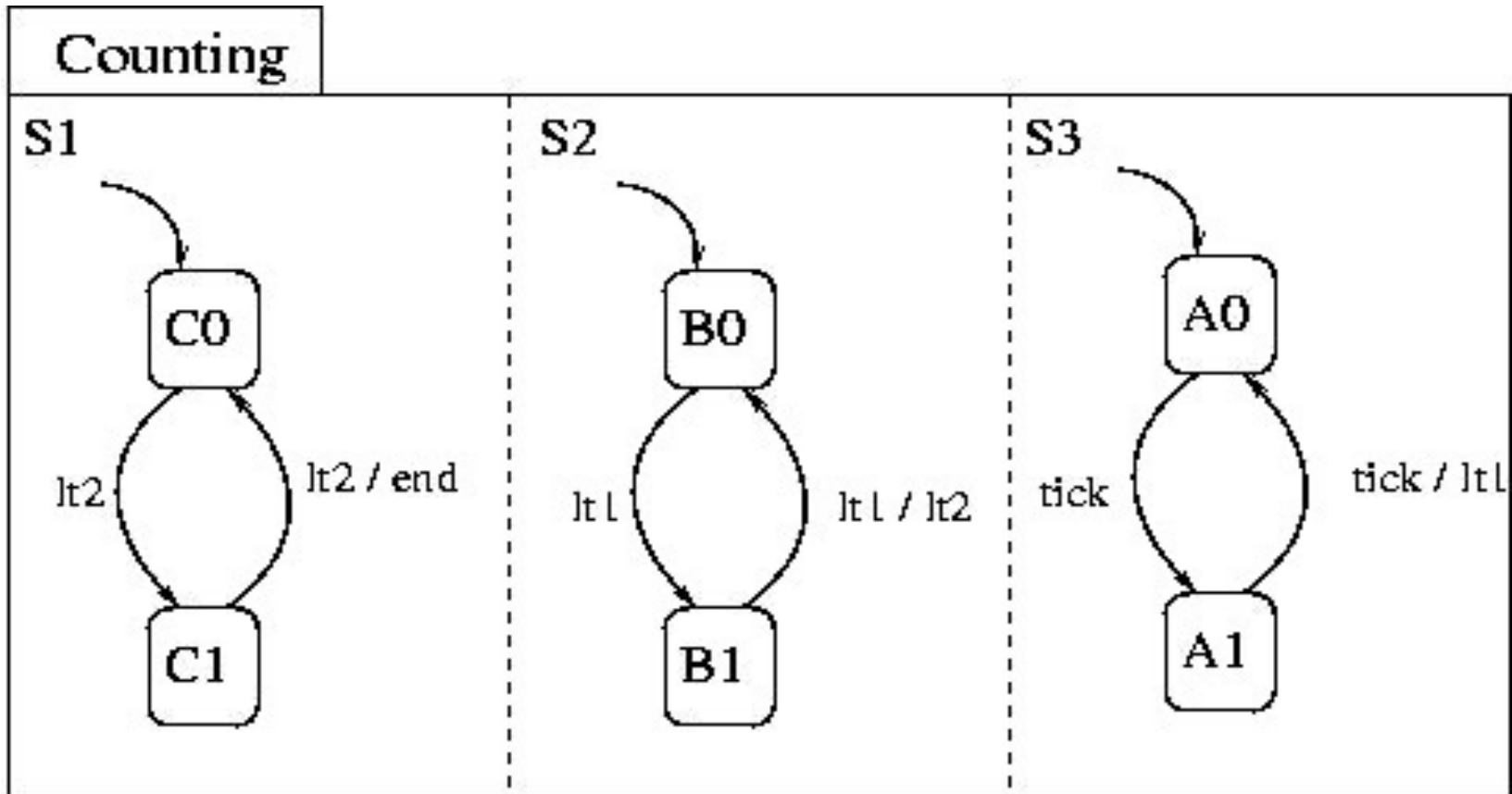
- Both outer and inner states active simultaneously
- When the outer state exits, inner states also exited
- Priorities of transitions
- **Preemption** (strong and weak)

Economy of Edges

- Every transition from outer state corresponds to many transitions from each of the inner states
- Hierarchical construct replaces all these into one single transition
- Edge labels can be complex

AND States

- An **Or** state contains exactly one state machine
- An **And** state contains two or more state machines
- Example:



Example

- Counting is an **And** state w/ 3 state machines
- **S1, S2, S3**, concurrent components of state
- When in state **Counting**, control resides **simultaneously** in all 3 state machines
- Initially, control is in **C0, B0** and **A0**
- Execution involves, in general, **simultaneous transitions** in all the state machines

Example (contd.)

- When in state **C0, B0, A1**, clock signal triggers the transition to **B1** and **A0** in **S2** and **S3**
- When in **C0, B1, A1**, clock signal input trigger the transitions to **C1, B0** and **A0** in all **S1, S2, S3**
- **And** state captures concurrency
- Default states in each concurrent component

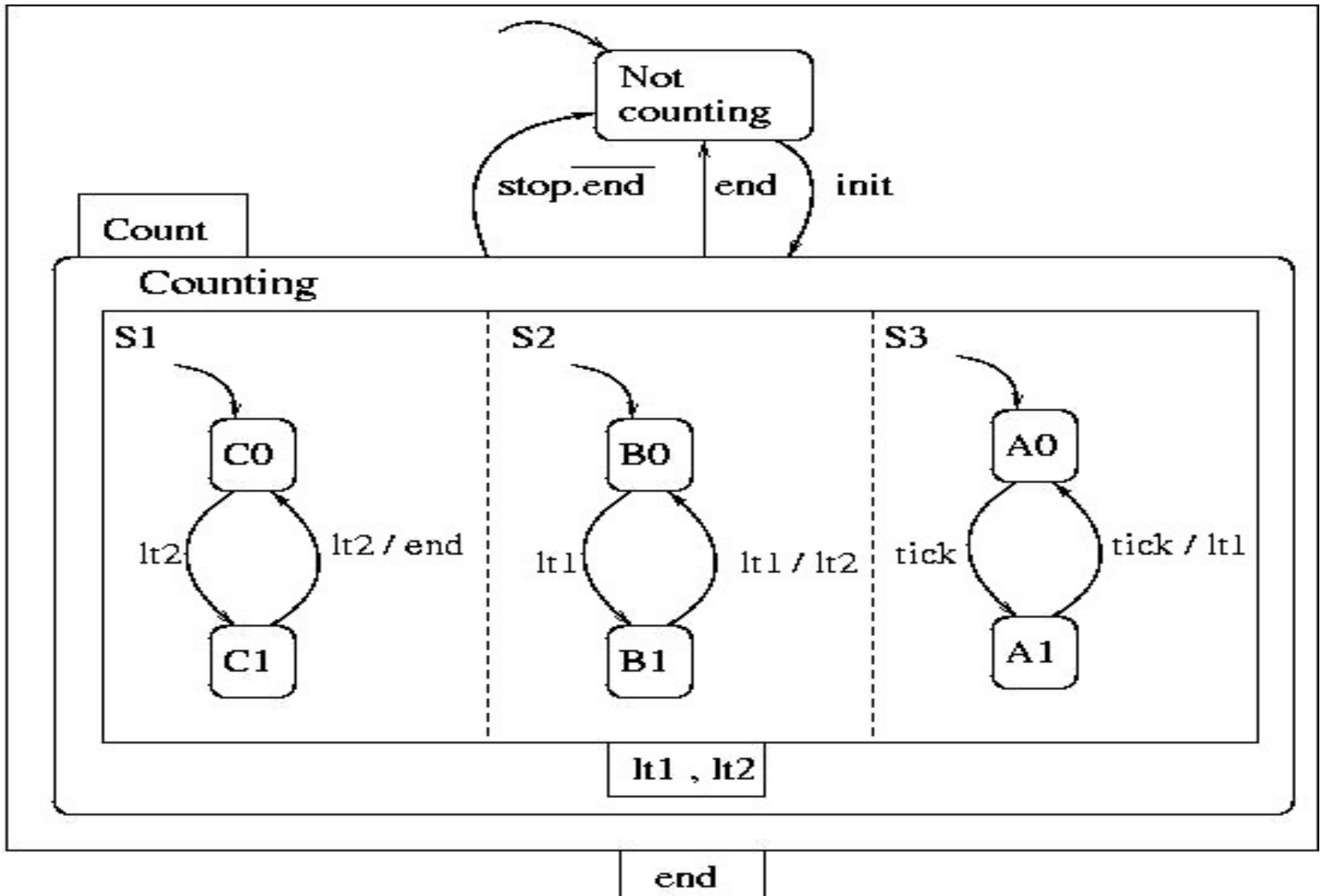
Economy of States

- **AND**-state can be flattened to single state mc
- Results in **exponential** number of states and transitions
- **AND** state is compact & intuitive representation

Counting

- What are the three components of the state?
- They represent behaviour of three bits of a counter
- **S3** –least significant bit, **S2** the middle & **S1** is MSB
- Compare this with flat and monolithic description of counter state machine given earlier
- Which is preferable?
- The present one is robust - can be redesigned to accommodate additional bits
- Look at the complete description of the counter

Complete Machine



Communication

- Concurrent components of AND state communicate with each other
- Taking an edge requires certain events to occur
- New signals are generated when an edge is taken
- These can trigger further transitions in other components
- A series of transitions can be taken as a result of one transition triggered by environment event
- Different kinds of communication primitives
- More on this later

Flat State Machines

- Capture the behaviour of the counter using FSMs
 - Huge number of states and transitions
 - Explosion of states and transitions
- Statechart description is compact
 - Easy to understand
 - Robust
 - Can be simulated
 - Code generation is possible
 - Execution mechanism is more complex

Exercise

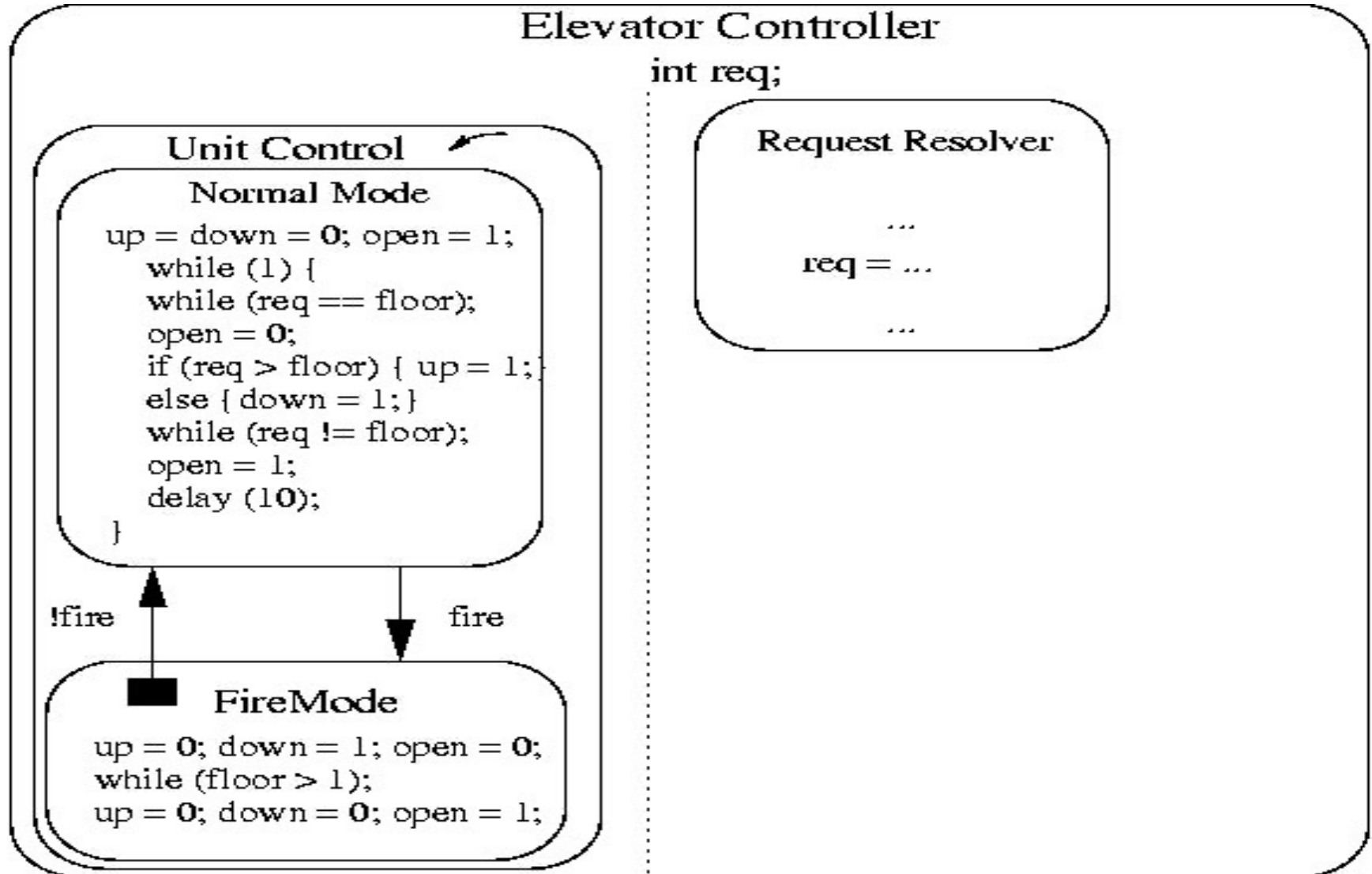
- Extend the lift controller example
 - Control for closing and opening the door
 - Control for indicator lamp
 - Avoid movement of the lift when the door is open
 - Include states to indicate whether lift in service or not
 - Controller for multiple lifts
- Give a Statechart description

Extensions to Statecharts

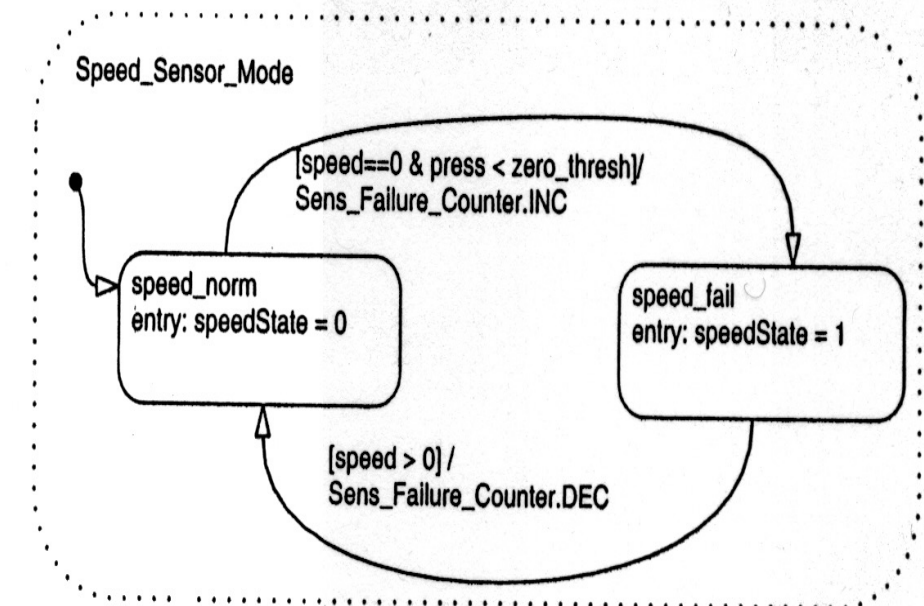
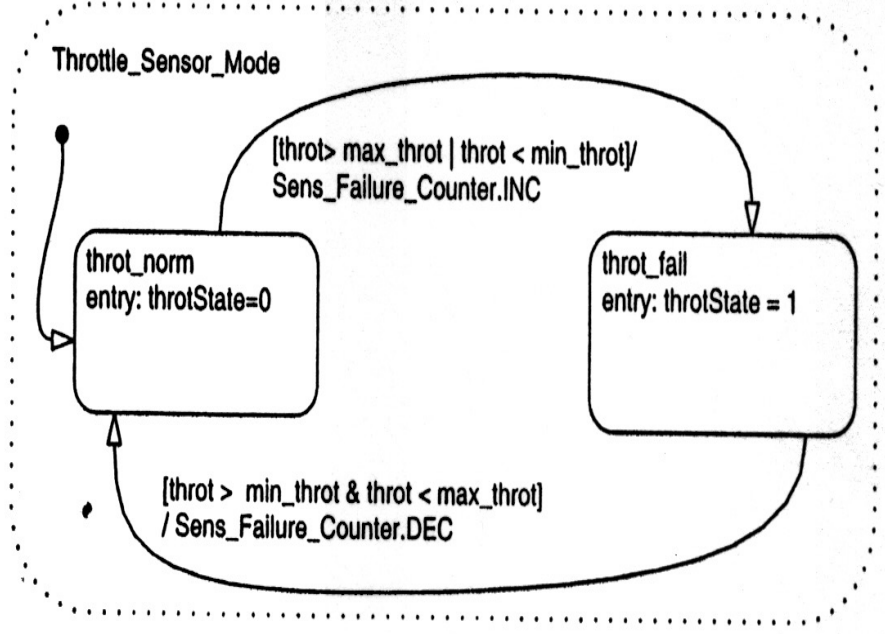
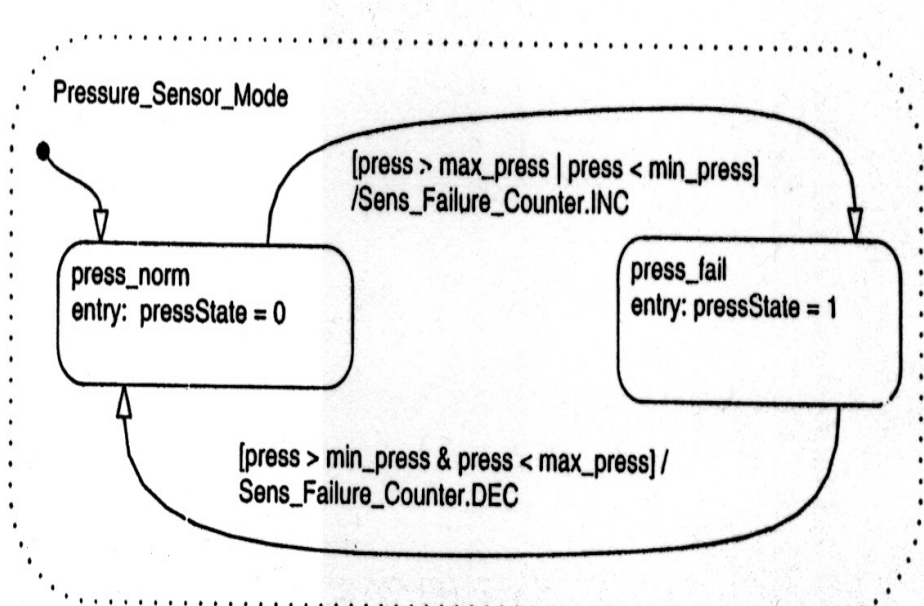
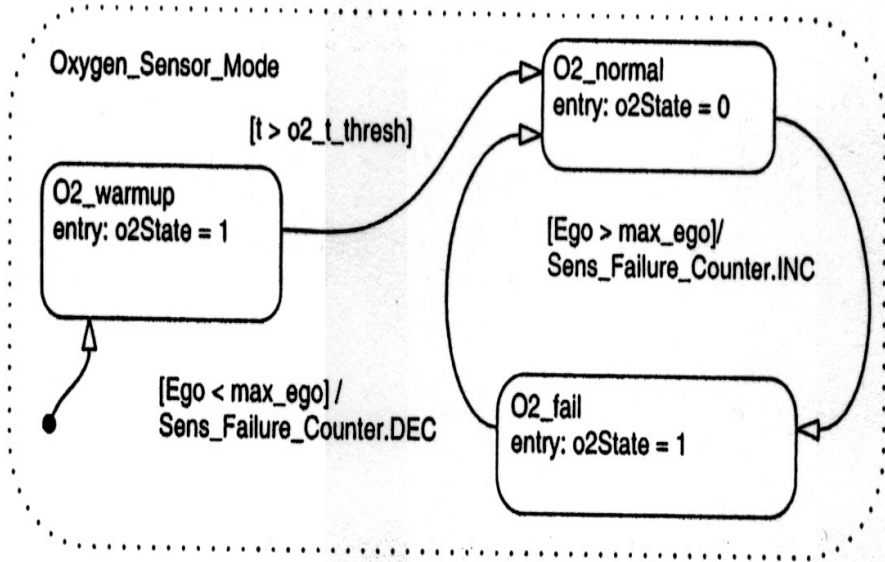
- Various possibilities explored
- Adding code to transitions, to states
- Complex data types and function calls
- Combining textual programs with statecharts
- Various commercial tools exist
 - Statemate and Rhapsody (ilogix)
 - UML tools (Rational rose)
 - Stateflow (Mathworks)
 - SynchCharts (Esterel Technologies)

Example

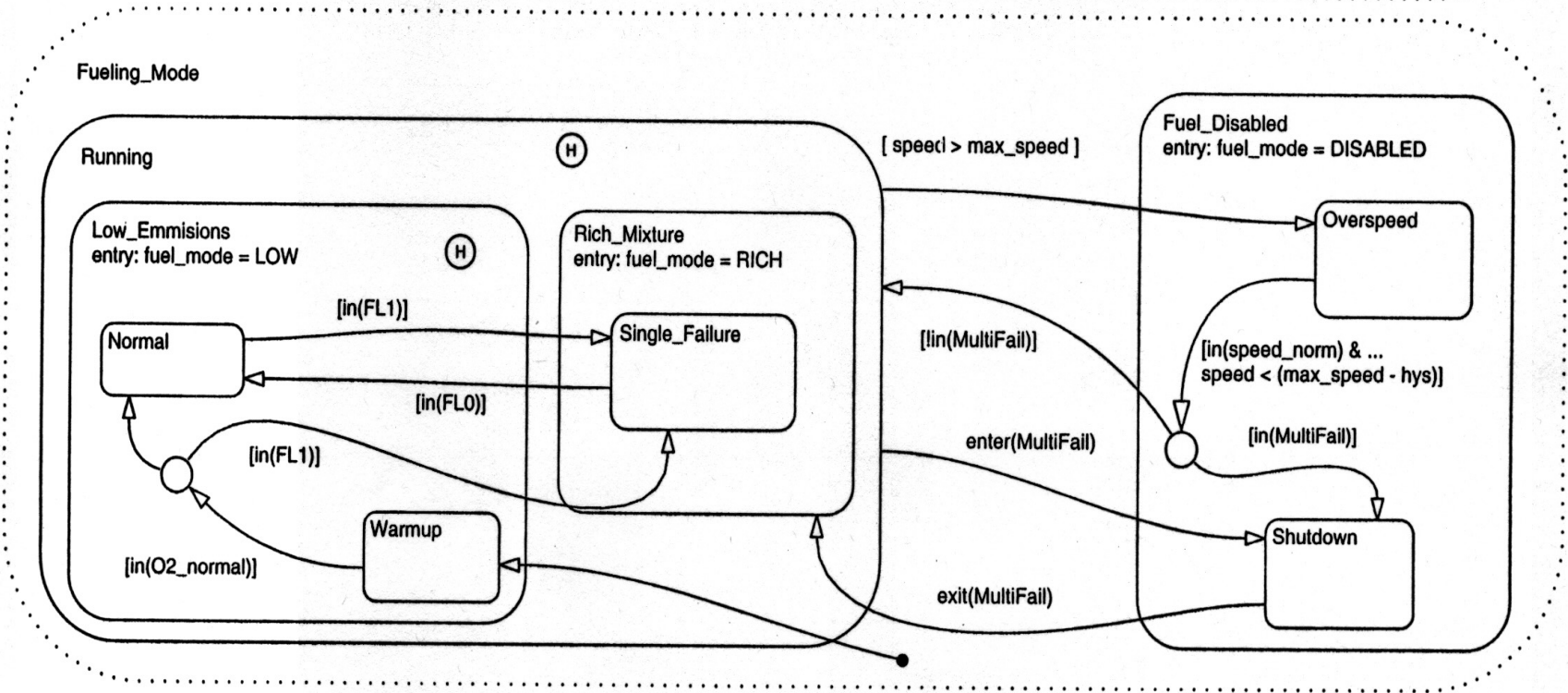
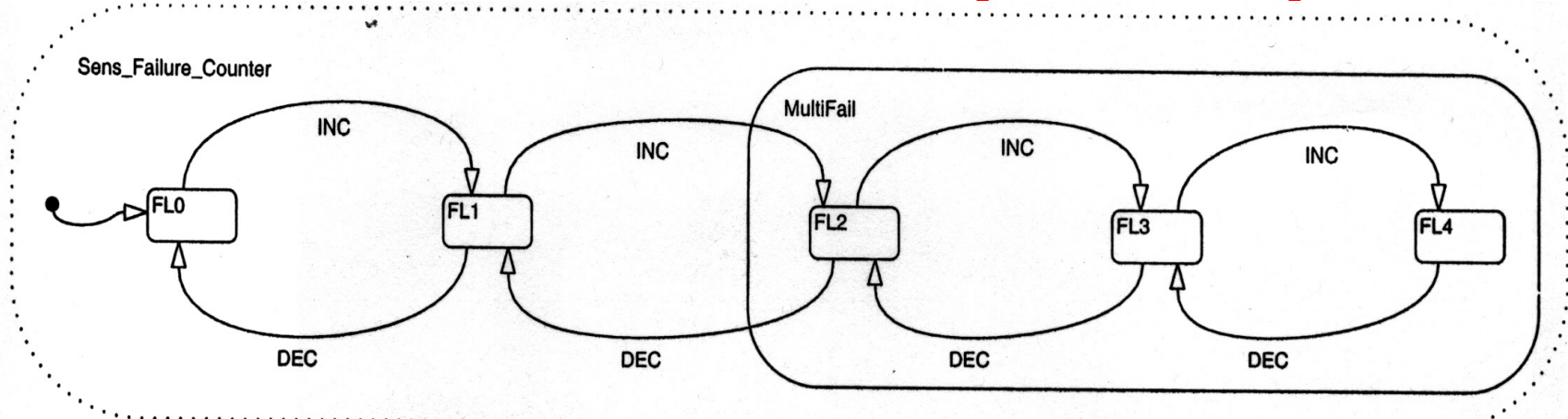
- Program State Machine model



Fuel Controller



Fuel Controller (Contd.)



Other Models

- Synchronous Reactive Models
 - Useful for expressing control dominated application
 - Rich primitives for expressing complex controls
 - Esterel (Esterel Technologies)
 - More on this later

Design Features

- **Two broad classifications**
 - Control-dominated designs
 - Data-dominated Designs

- **Control-dominated designs**
 - Input events arrive at irregular & unpredictable times
 - Time of arrival and response more crucial than values

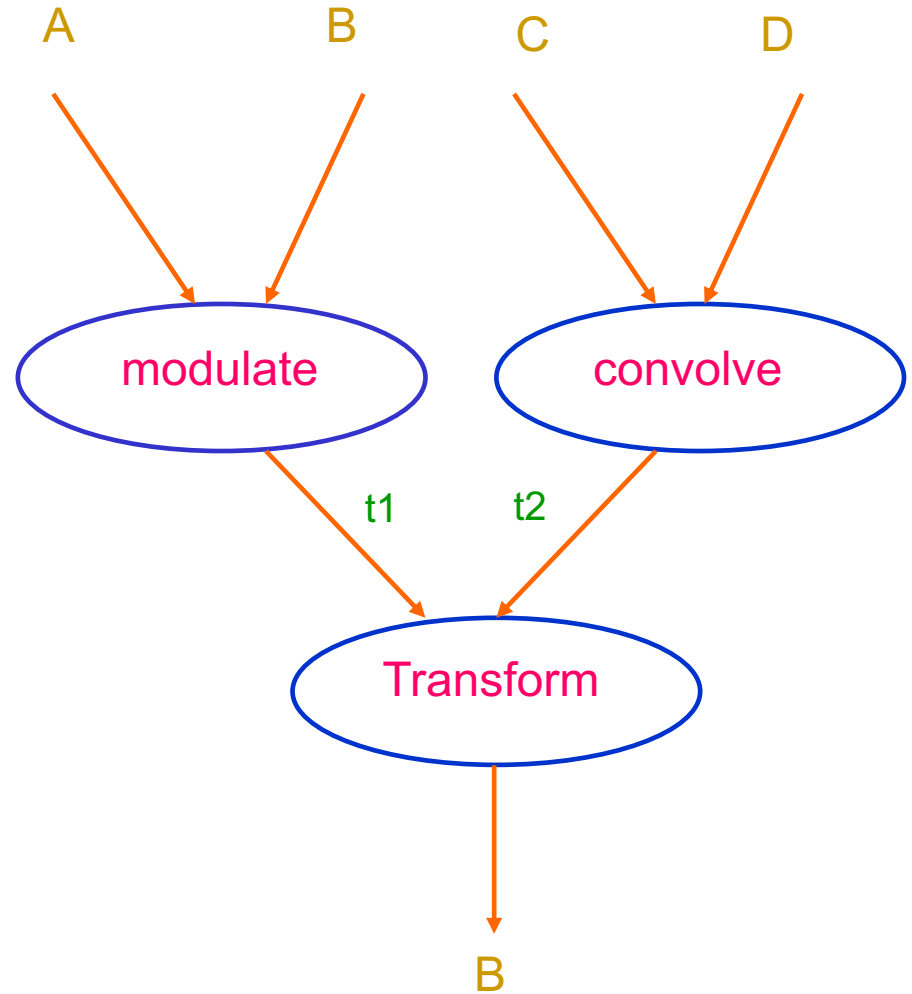
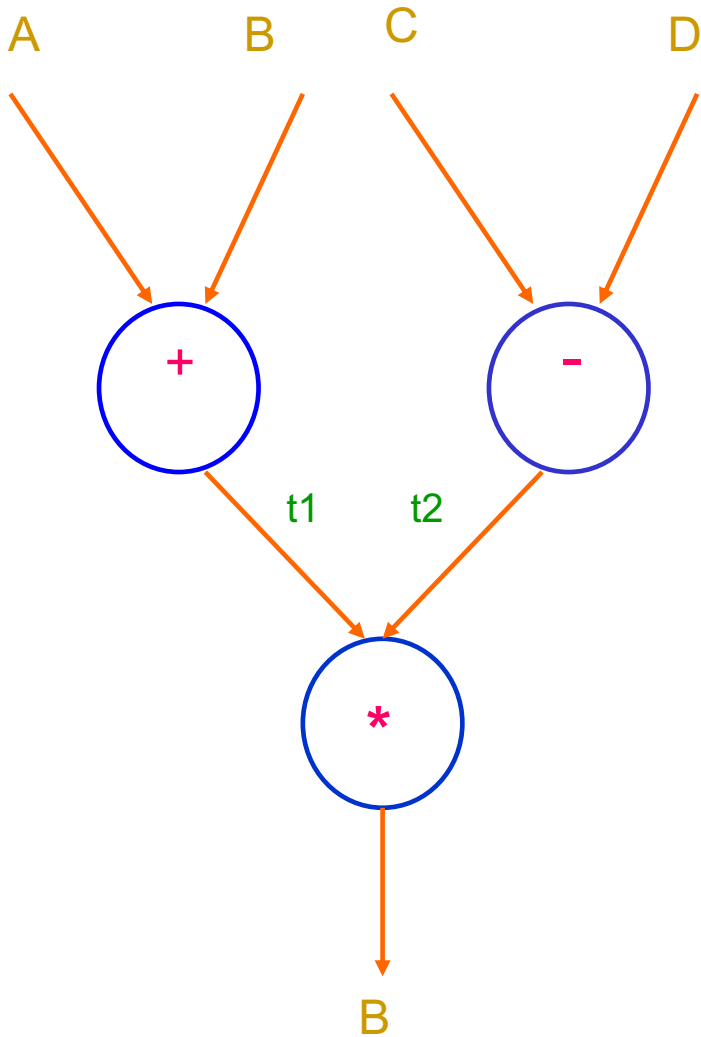
Design Features

- **Data-dominated designs**
 - Inputs are streams of data coming at regular intervals (sampled data)
 - Values are more crucial
 - Outputs are complex mathematical functions of inputs
 - numerical computations and digital signal processing computations

Data flow Models

- State machines, Statecharts, Esterel are good for control-dominated designs
- Data flow models for data-dominated systems
- Special case of concurrent process models
- System behaviour described as an interconnection of nodes
- Each node describes transformation of data
- Connection between a pair of nodes describes the flow of data from one node to the other

Example



Data Flow Models

- Graphical Languages with support for
 - Simulation, debugging, analysis
 - Code generation onto DSP and micro processors
- Analysis support for hw/sw partitioning
- Many commercial tools and languages
 - Lustre, Signal
 - SCADE
 - Matlab, Scilab

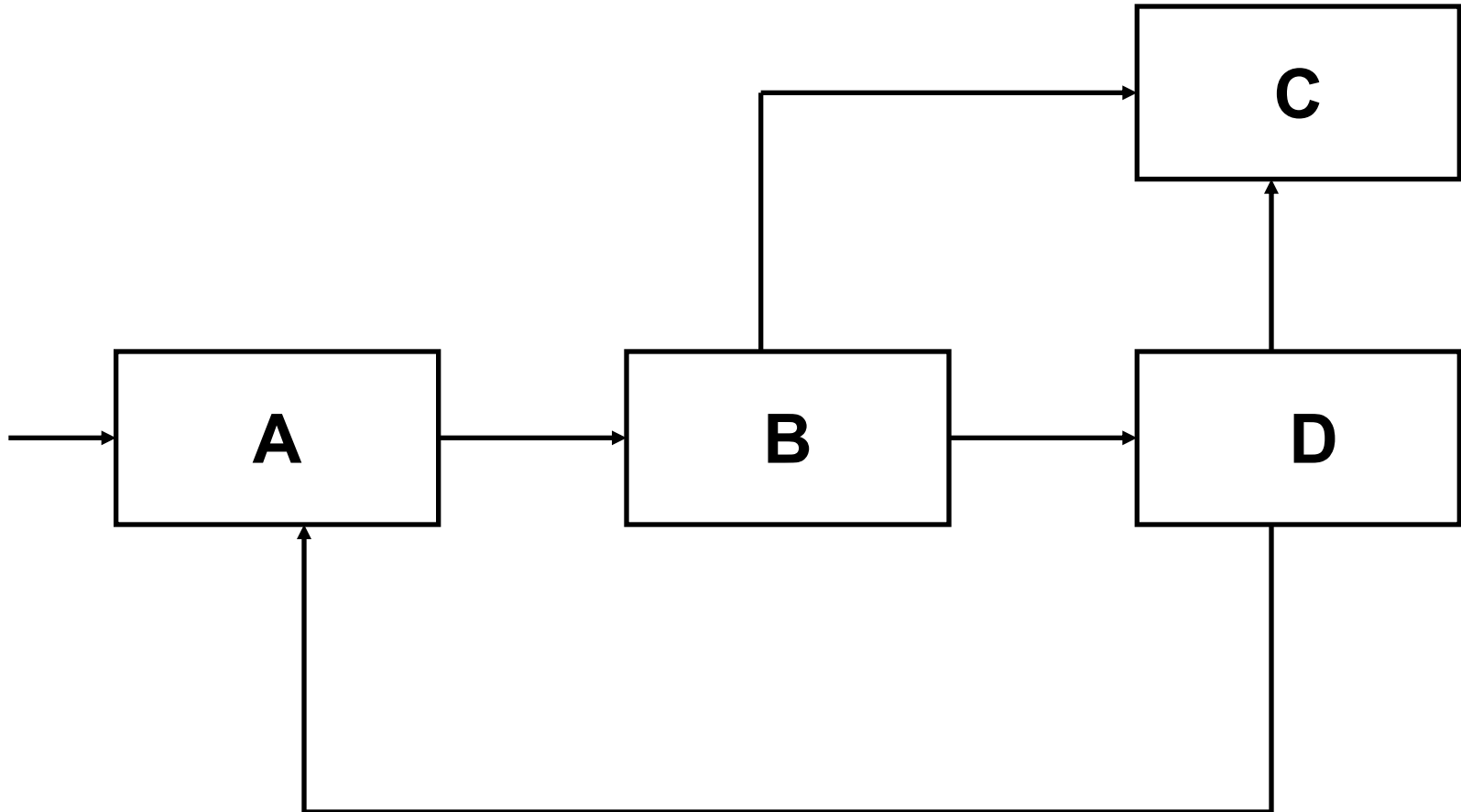
Discrete Event Models

- Used for HW systems
- VHDL, Verilog
- Models are interconnection of nodes
- Each node reacts to events at their inputs
- Generates output events which trigger other nodes

Discrete Event Models

- External events initiate a reaction
- Delays in nodes modeled as delays in event generation
- Simulation
- Problems with cycles
- Delta cycles in VHDL

Discrete Event Models



Realtime Embedded Systems

Embedded Software

Typical structure of a simple embedded system (Software)

loop

read inputs/sensors;

compute response;

generate actuator outputs

forever

Embedded Software (contd.)

- **Design Decisions**

- How to read inputs?
- How often to read inputs?
- Which order to read the inputs?
- How to compute responses?
- How to generate the responses?
- How often to generate?

The Simplest Approach

Round Robin Scheme

loop

await tick;

read S1; take_action(S1);

read S2; take_action(S2);

read S3; take_action(S3);

forever

Tick is a time interrupt

The Most General Scheme

- **Task1 || Task2 || ... || Task8**
- **Tasks**
 - Sequential threads
 - Concurrently executed
 - Can be scheduled and suspended
 - Wait for specific time period or events
 - Communicate with each other

The Most General Scheme

- **Real-time OS (RTOS kernel)**
 - Manages the tasks
 - Task communications
 - Timer services
 - Schedules the tasks for execution using various
 - Scheduling strategies

Summary

- Various models reviewed
 - Sequential programming models
 - Hierarchical and Concurrent State Machines
 - Data Flow Models, Discrete Event Models
- Each model suitable for particular application
- State Machines for **event-oriented** control systems
- Sequential prog. model, **data flow** model for fcn computation
- Real systems often require **mixture** of models
- Modeling tools/ lang. should have combination of all the features
 - **Ptolemy** (Berkeley) project studies modeling, simulation, and design of concurrent, real-time, embedded systems (Java based). <http://ptolemy.eecs.berkeley.edu/>
 - **POLIS** (Berkeley) framework for hw-sw Co-Design of Embedded Systems.
 - **LUSTRE/SCADE** of Esterel Technologies (from INRIA, France)

References

- F. Balarin et al., [Hardware – Software Co-design of Embedded Systems: The POLIS approach](#), Kluwer, 1997
- N. Halbwachs, [Synch. Prog. Of Reactive Systems](#), Kluwer, 1993
- D. Harel et al., [STATEMATE: a working environment for the development of complex reactive systems](#), IEEE Trans. Software Engineering, Vol. 16 (4), 1990.
- J. Buck, et al., [Ptolemy: A framework for simulating and prototyping heterogeneous systems](#), Int. Journal of Software Simulation, Jan. 1990
- Edward A. Lee, [Overview of the Ptolemy Project](#), Technical Memorandum No. UCB/ERL M03/25, University of California, Berkeley, CA, 94720, USA, July 2, 2003
- Gerard Berry, [The Esterel v5 Language Primer Version v591](#), Centre de Mathematiques Appliques Ecole des Mines and INRIA 2004, June 5, 2000. Available from <https://www.researchgate.net/publication/242374294> [The Esterel v5 Language Primer Version v5 91](#)
- Edward A. Lee and Yang Zhao, "[Reinventing Computing for Real Time](#) in Proceedings of the Monterey Workshop 2006, LNCS 4322, pp. 1-25, 2007, F. Kordon and J. Sztipanovits (Eds.) © Springer-Verlag Berlin Heidelberg 2007
- N. Halbwachs et al. The Synchronous Data Flow Programming Language LUSTRE. In Proc. IEEE 1991 Vol. 79, No. 9. Accessed 17 March 2014.
- J. Colaço, B. Pagano and M. Pouzet, "SCADE 6: A formal language for embedded critical software development (invited paper)," 2017 International Symposium on Theoretical Aspects of Software Engineering (TASE), Sophia Antipolis, 2017, pp. 1-11, doi: 10.1109/TASE.2017.8285623