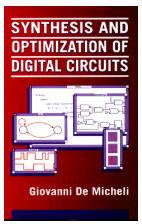
FSM-based Specification Formalisms

Giovanni De Micheli

Integrated Systems Laboratory



Module 1

- **Objectives:**
  - Finite-state machines
  - Synchronous languages
  - State Charts

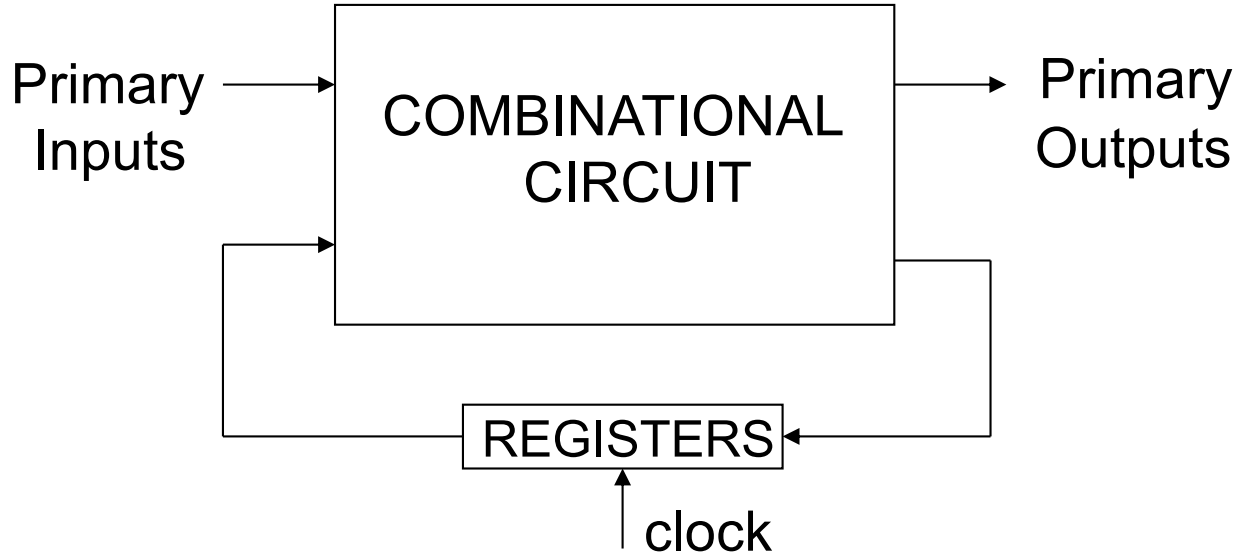
Models of computation

- **Data-flow** oriented models
  - Focus on computation
  - Data-flow graphs and derivatives
- **Control-flow** oriented models
  - Focus on control
  - Based on *finite-state machine models*
- DF and CF model complementary aspects

Formal FSM model

- A set of primary inputs patterns *X*
- A set of primary outputs patterns *Y*
- A set of states *S*
- A *state* transition function:
  - $\delta: X \times S \rightarrow S$
- An output function:
  - $\lambda : X \times S \rightarrow Y$ for *Mealy* models
  - $\lambda : S \rightarrow Y$ for *Moore* models

Finite-state machines



- A finite-state machine is an abstraction
- Computation takes no time
  - Outputs are available as soon as inputs are
- A finite-state machine implementation is a sequential circuit with a finite cycle-time

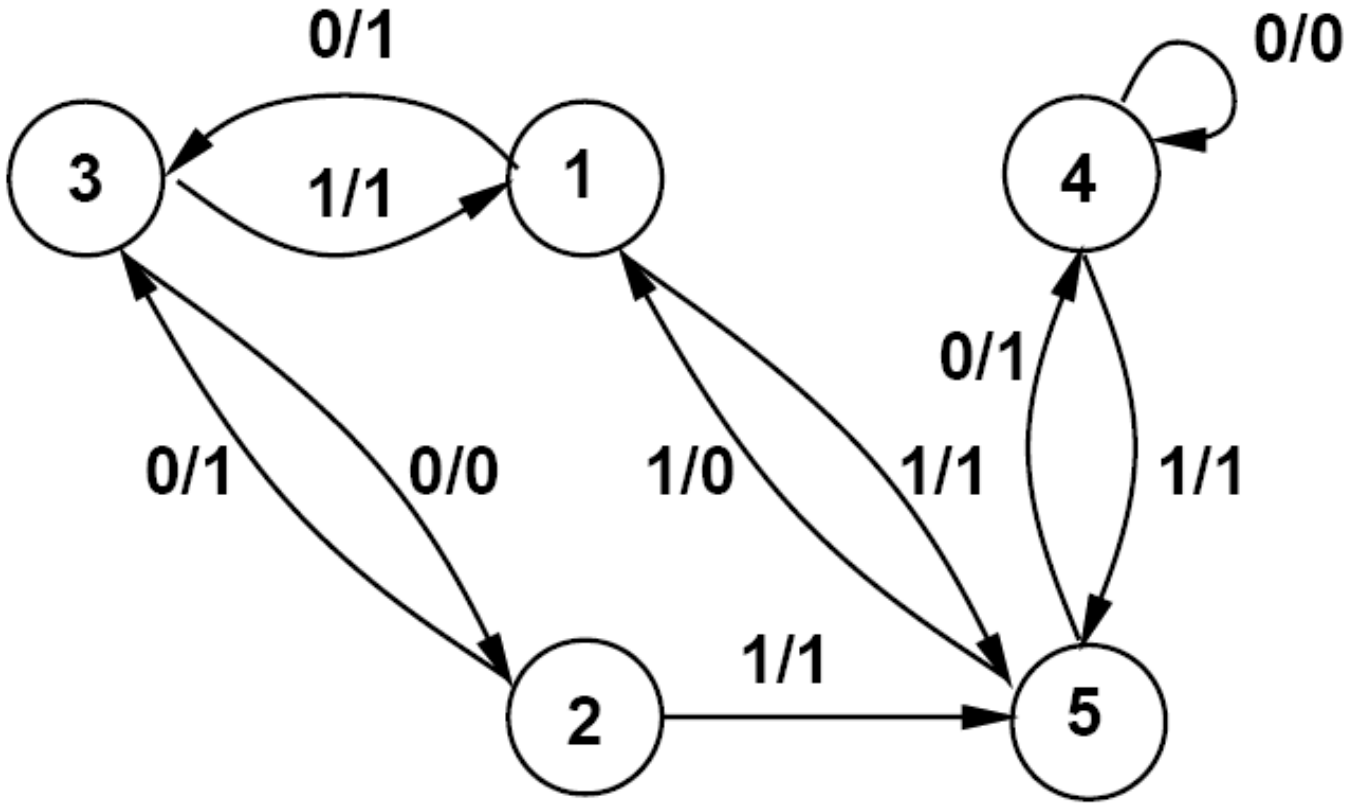
State diagrams

- Directed graph
  - Vertices = states
  - Edges = transitions
- Equivalent to state transition tables

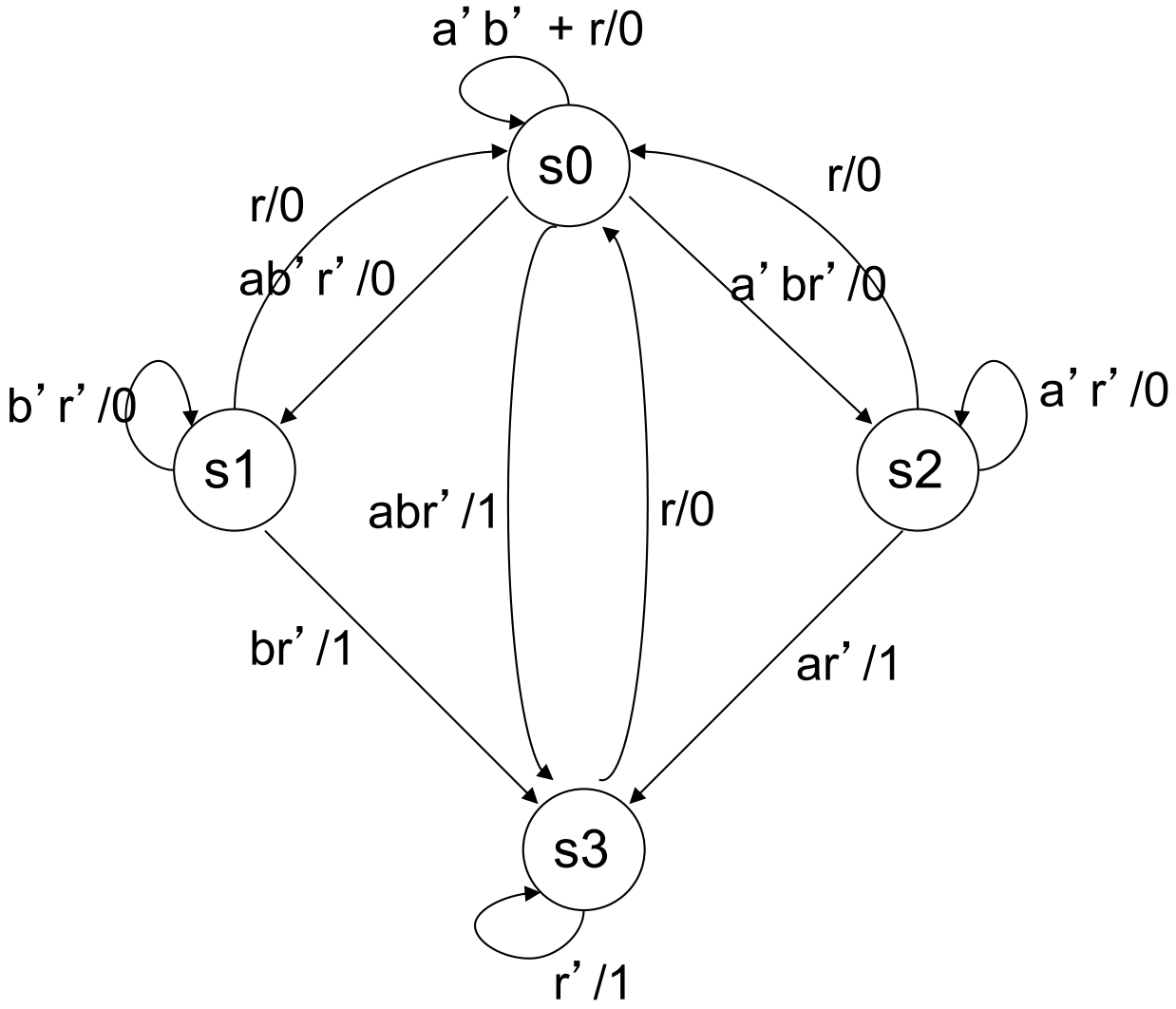
Example

INPUT	STATE	N-STATE	OUTPUT
0	s_1	s_3	1
1	s_1	s_5	1
0	s_2	s_3	1
1	s_2	s_5	1
0	s_3	s_2	0
1	s_3	s_1	1
0	s_4	s_4	0
1	s_4	s_5	1
0	s_5	s_4	1
1	s_5	s_1	0

Example



Example



FSM-based models

- Synchronous languages:
  - Esterel, Argos, Lustre, SDL
- Graphical formalisms:
  - FSMs, hierarchical FSMs, concurrent FSMs
  - StateCharts
  - Program-state machines
  - SpecCharts

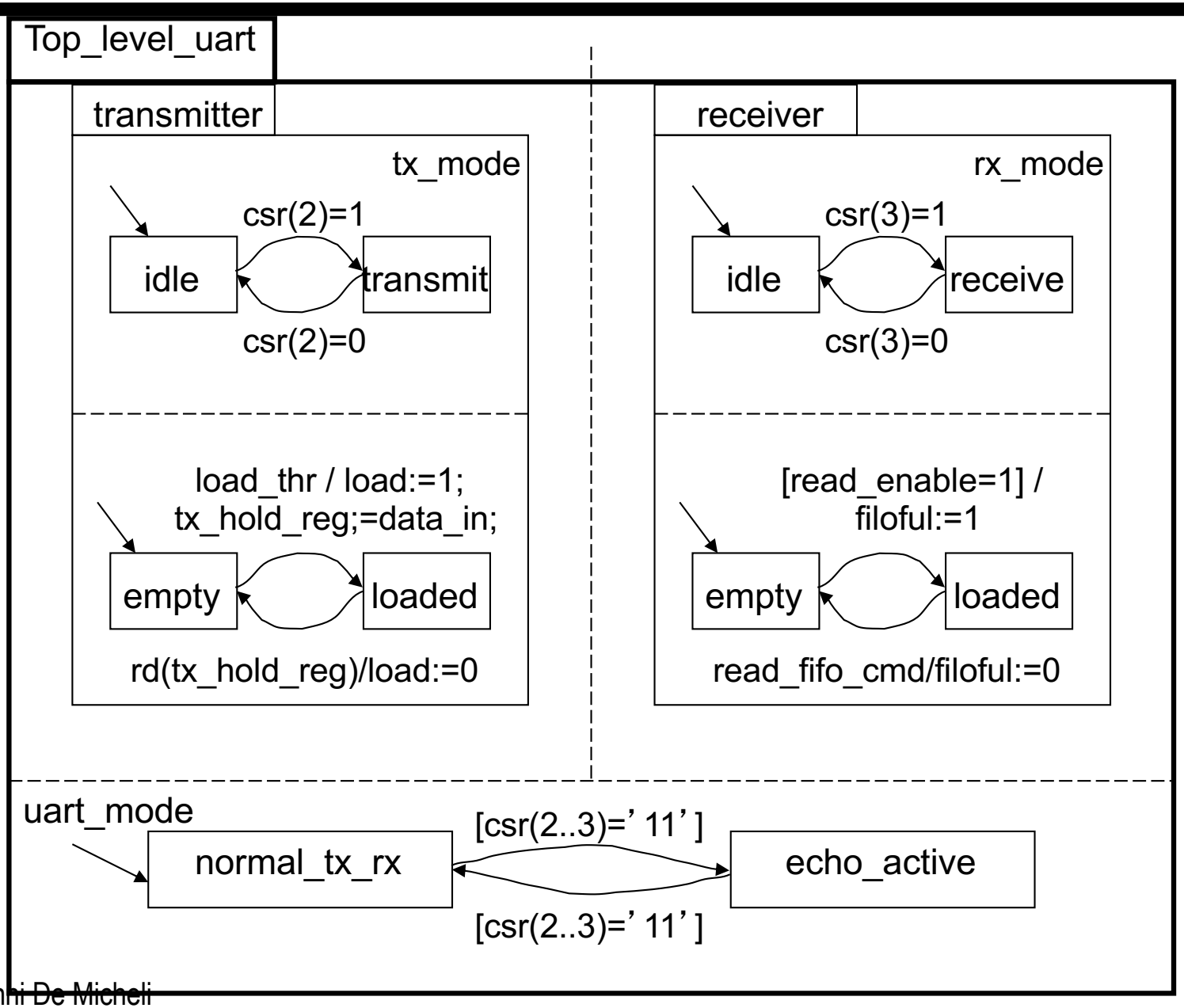
State Charts

- Proposed by Harel
- Graphic formalism to specify FSMs with:
  - Hierarchy
  - Concurrency
  - Communication
- Tools for simulation, animation and synthesis

State Charts

- States
- Transitions
- Hierarchy
  - OR (sequential) decomposition
    - State → a sequence of states
  - AND (concurrent) decomposition
    - State → a set of concurrent states

State charts



State Charts
Additional features

- State transitions across multiple levels
- Timeouts:
  - Notation on transition arcs denoting the max/min time in a given state
- Communication:
  - Broadcast mechanism based on event generation and reception
- History feature:
  - Keep track of visited states

StateCharts

- **Advantages:**
  - Formal basis
  - Easy to learn
  - Support of hierarchy, concurrency and exceptions
    - Avoid exponential blow up of states
- **Disadvantages:**
  - No description of data-flow computation

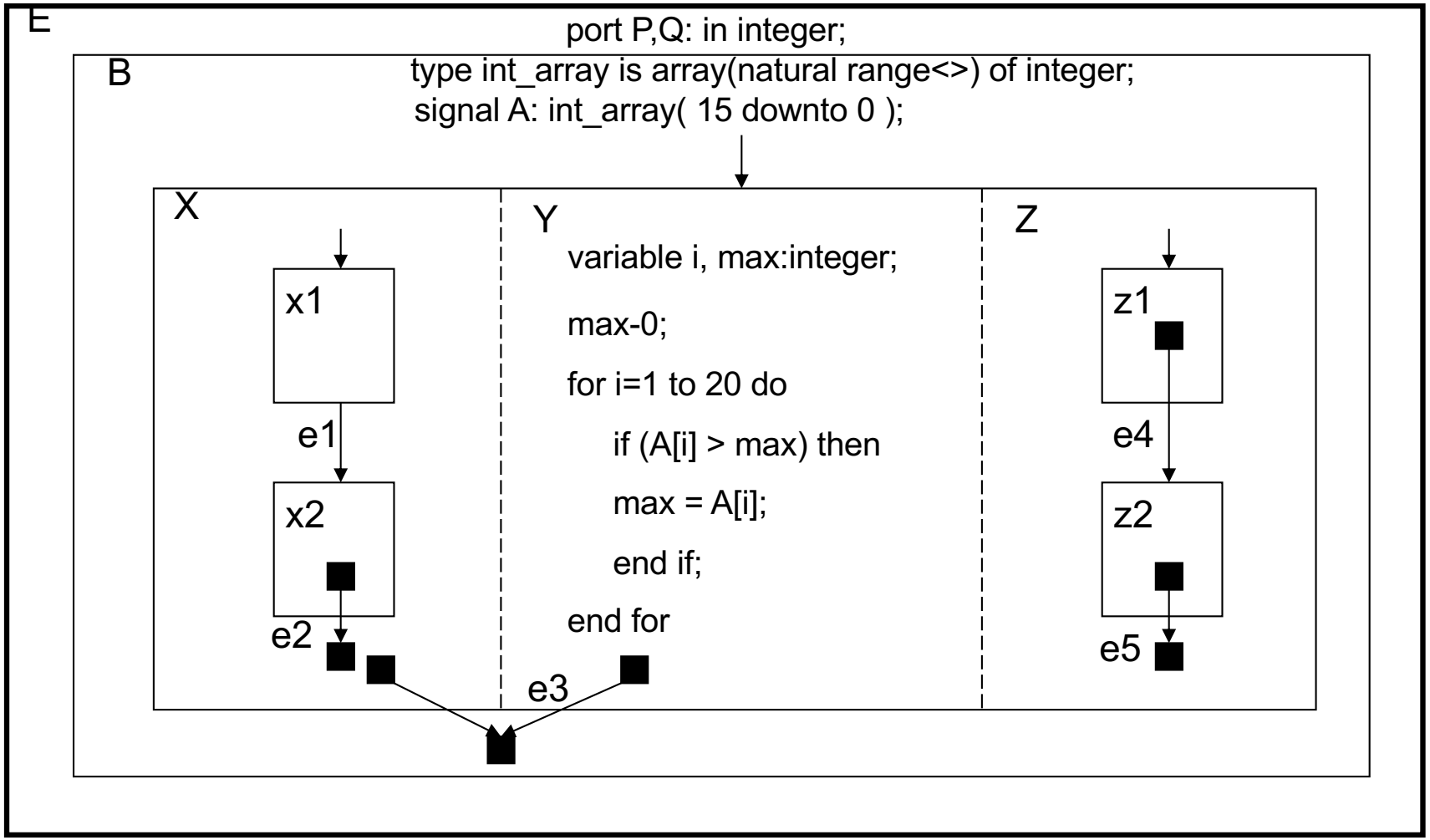
Program State Machines

- Combining FSM formalism with program execution
- In each state a specific program is active
- Hierarchy:
  - Sequential states
  - Concurrent states
- In a hierarchical state, several programs may be active

SpecCharts

- Based on Program State Machines
  - Introduced by Gajski et al.
- Extension of VHDL:
  - Compilable into VHDL for simulation and synthesis
  - Behavioral hierarchy
- Combining FSM and VHDL formalisms
  - Leaves of the hierarchy are VHDL models

Example



- **TOC: e2, e3**
- **TI: e1**

State transitions

- Sequencing between sub-behaviors are controlled by transition arcs
  - *TOC - Transition on completion*
    - Program terminates AND transition condition is true
  - *TI - Transition immediate*
    - Transition condition is true
- A transition arc is labeled by a triple:
  - (transition type, triggering event, next behavior)

SpecCharts semantics

- Timing semantics similar to VHDL
- Synchronization:
  - Use *wait statement*
  - Use TOC looping back to the top of the program
- Communication:
  - Using variables and signals
  - Message passing (send/receive)
- Simulation;
  - Model can be flattened to VHDL

Module 2

- **Objectives:**
  - **Expression-based formalisms**
  - **Control-flow expressions**

Expression-based formalisms

- Represent sequential behavior by expressions
- Advantages:
  - Symbolic manipulation
  - Translation into FSM models
- Disadvantages:
  - Loss of data-flow information

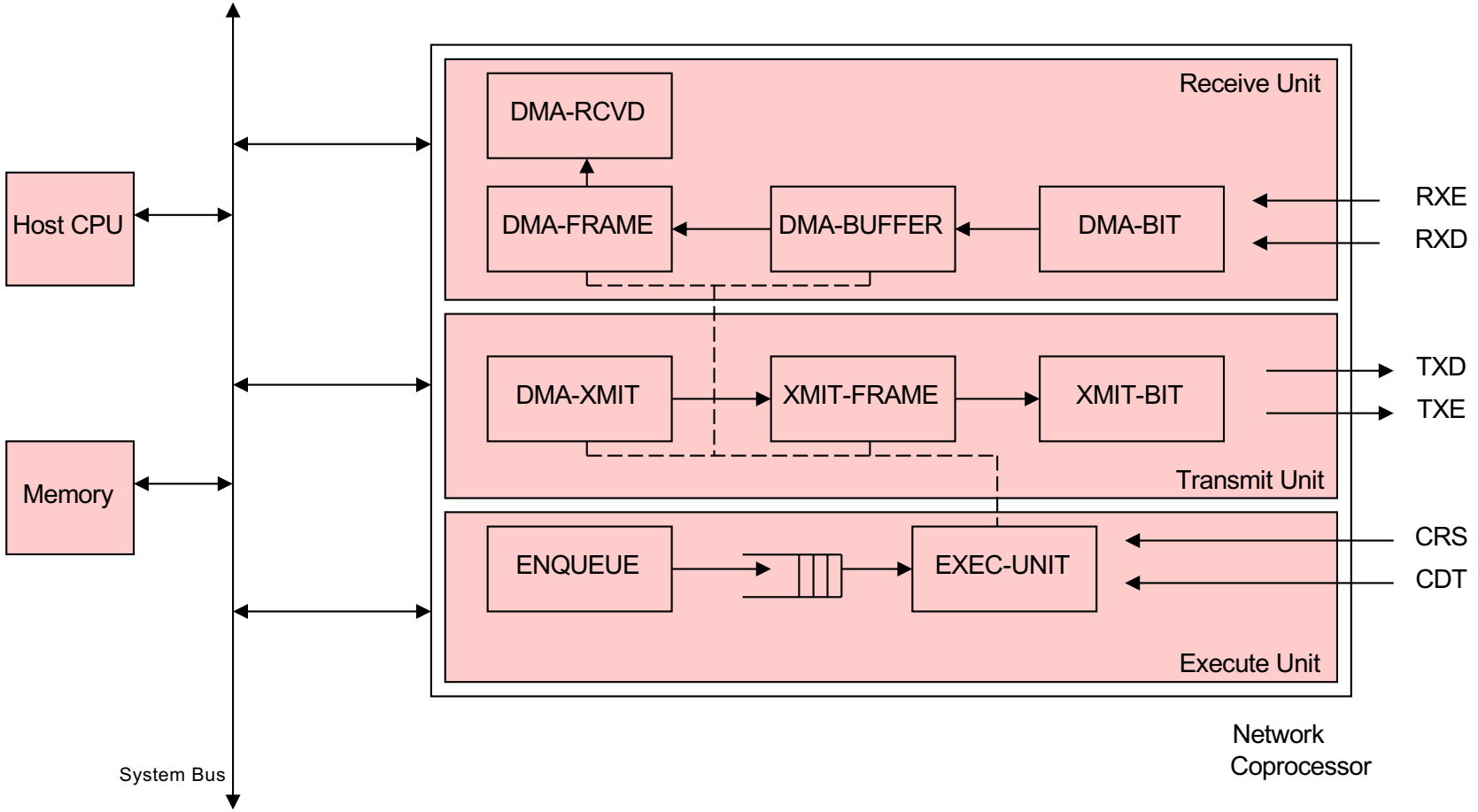
Control-flow expression formalism

- Expressions capturing a *high-level view of control-flow* while abstracting data-flow information
- Expressions are extracted directly from HDL or programming language specifications
- *Cycle-based semantics* provides a formal interpretation of HDLs
- Based on the *algebra of synchronous processes* (Process Algebra)

Control-Flow Expressions

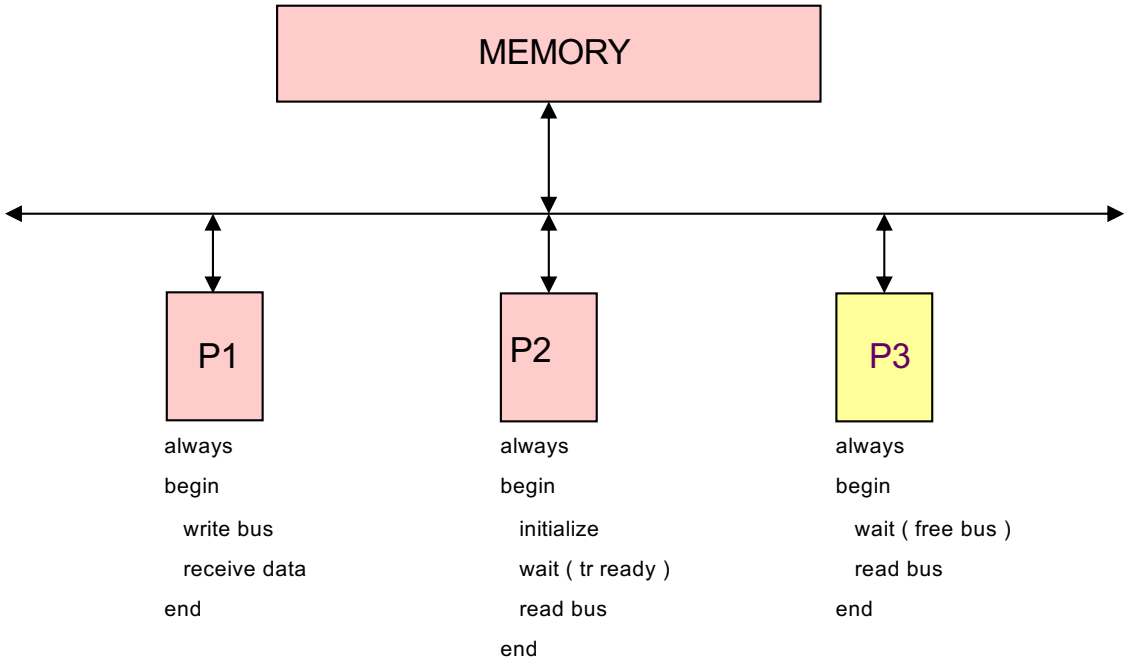
Composition	HDL	CFE
Sequential	**begin** P; Q **end**	$p \cdot q$
Parallel	**fork** P; Q **join**	$p \amalg q$
Alternative	**if** (c) P ; **else** Q ;	$c : p + \overline{c} : q$
Loop	**while** (c) P ; **wait (!c)** P ;	$(c: p)^*$ $(c: 0)^* .p$
Infinite	**always** P ;	p^ω

Example of design problem
Ethernet controller



- **Problem**
  - **Avoid bus conflicts**

Example of design problem
Ethernet controller



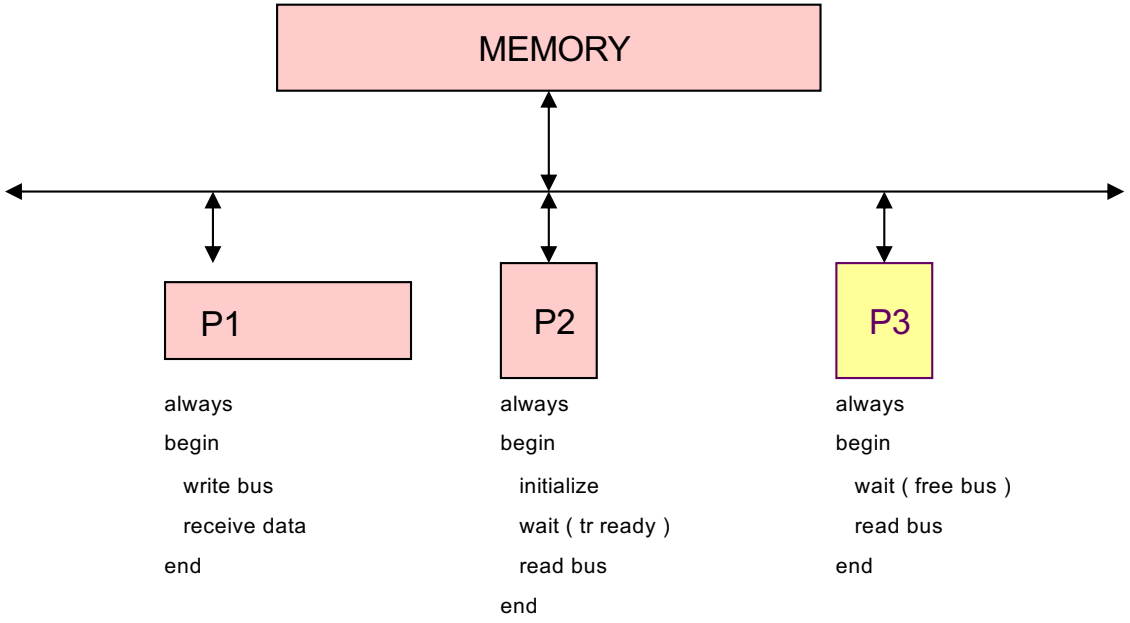
$p = p1 \text{ ıı } p2 \text{ ıı } p3$

$p1 = [a.0]^{\omega}$

$p2 = [0.(c:0)^*.a]^{\omega}$

$p3 = [(x:0)^*.a]^{\omega}$

Example
Control-Flow Expressions



$p = p1 \text{ ıı } p2 \text{ ıı } p3$

$p1 = [a.0]^{\omega}$

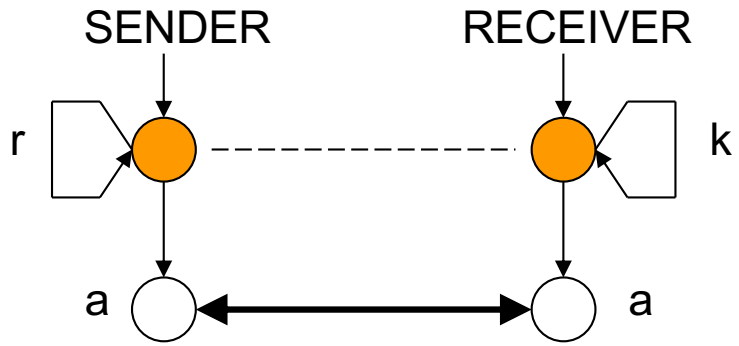
$p2 = [0.(c:0)^*.a]^{\omega}$

$p3 = [(x:0)^*.a]^{\omega}$

$NEVER = \{a,a\}$

- **Never access the bus twice simultaneously**

Example Synchronization



- Synchronization between a sender and a receiver in a blocking protocol
  - Sender = (x : r)*.a
  - Receiver = (y : k)*.a
  - ALWAYS = {{a; a}}
  - NEVER = {{r; k}}

Design with CFEs

- **Representation:**
  - A CFE can be compiled into a *specification automaton*
  - Representing all feasible behaviors
- **Synthesis:**
  - A control-unit implementation is a FSM
  - Derivable from a specification automaton by assigning values to decision variables over time
- **Optimization:**
  - Minimize a cost function defined over the decision variables

CFE Summary

- Control-flow expression are a modeling tool
- Formal semantic:
  - Support for synthesis and verification
- Synthesis path from CFEs to control-unit