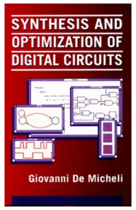
Modeling Languages and Abstract Models

Giovanni De Micheli

Integrated Systems Centre

EPF Lausanne


SYNTHESIS AND OPTIMIZATION OF DIGITAL CIRCUITS

Giovanni De Micheli

Module 1

- Objective
  - Modeling requirements in systems
  - Modeling styles

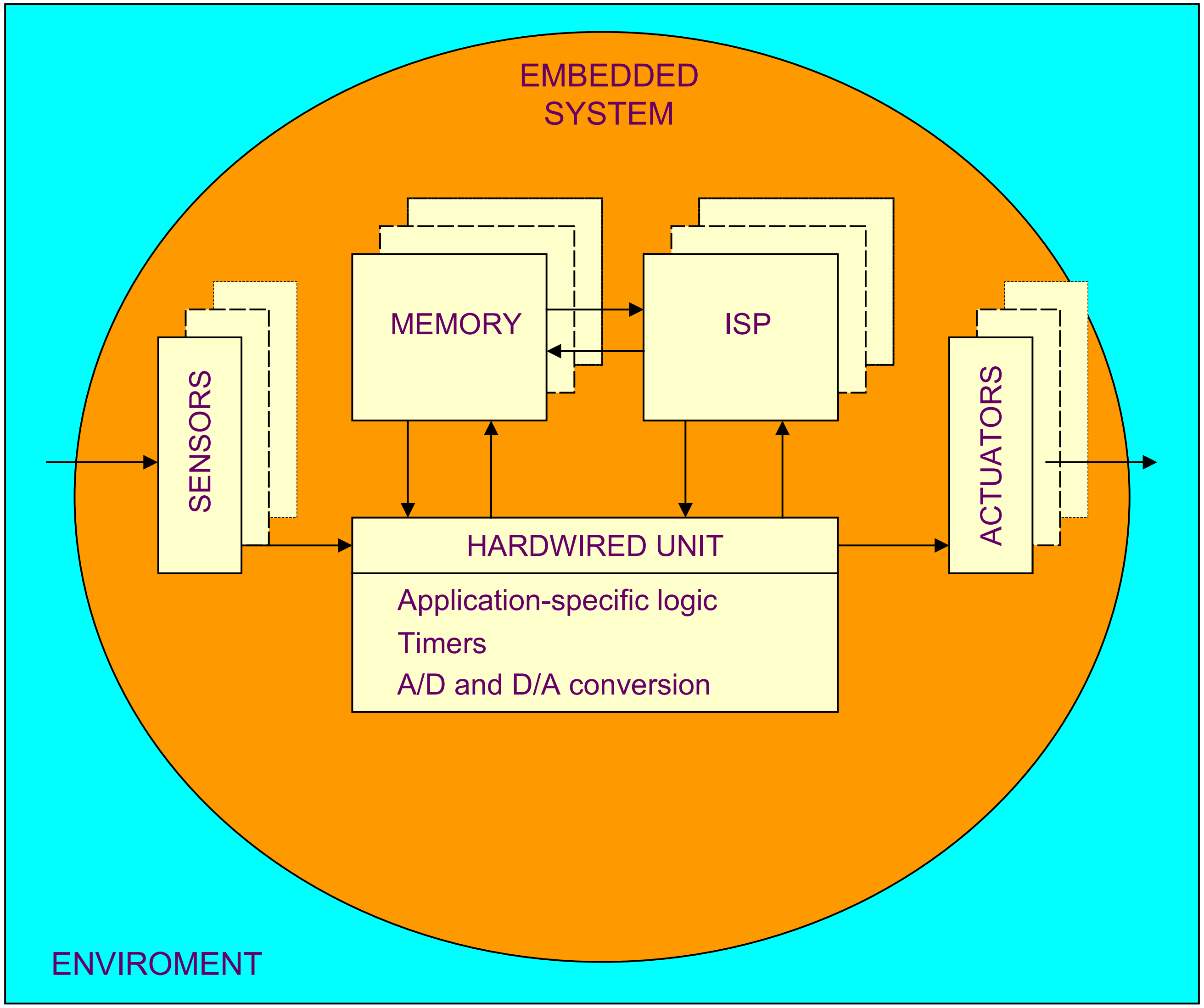
Electronic systems

- A system is a combination of:
  - Hardware platform:
    - Processors, memories, transducers
  - Software :
    - Application and system software
- Attributes:
  - Application domain
    - Computing, communication, consumer
  - Integration level
    - Chip, board, distributed/networked
  - Function
    - Autonomous, embedded

Trends and challenges

- Design increasingly more complex systems under higher *time to market* pressure
  - Raise level of abstraction
- Design starts are mainly for embedded system applications
- Use and re-use of high-level components
  - Processors, controllers, embedded memories
- Support concurrent Hw/Sw development
  - System customization via embedded software
- Support automated synthesis and verification

Embedded systems



Embedded system requirements

- **Reactive systems:**
  - The system never stops
  - The system responds to signals produced by the environment
- **Real-time systems:**
  - Timing constraints on task execution
  - Hard and soft constraints

System modeling

- Represent system functions while abstracting away unnecessary details
  - Software *programming languages*
  - *Hardware description languages*
  - *Flow* and *state-based diagrams*
  - Schematics
- No golden solution
  - System heterogeneity

The limits of my language mean the limits of my world

Wittgenstein

Circuit Modeling

- Formal methods:
  - Models in hardware languages
  - Flow and state diagrams
  - Schematics
- Informal methods:
  - Principles of operations
  - Natural-language descriptions

Hardware Description Languages

- Specialized languages with hardware design support
- Multi-level abstraction:
  - Behavior, RTL, structural
- Support for simulation
- Try to model hardware as designer likes to think of it

Software programming languages

- Software programming languages (C) can model functional behavior:
  - Example: processor models
- Software language models support marginally design and synthesis:
  - Unless extensions and overloading is used
  - Example: SystemC
- Different paradigms for hardware and software
- Strong trend in bridging the gap between software programming languages and HDLs

Hardware versus software models

- **Hardware:**
  - *Parallel* execution
  - I/O ports, building blocks
  - Exact event timing is *very* important
- **Software:**
  - Sequential execution (usually)
  - Structural information less important
  - Exact event timing is *not* important

Module 2

- Objectives
  - Language analysys
  - Procedural languages (Verilog)
  - Declarative languages (Silage)
  - Object-oriented languages (SystemC)

Language analysis

- **Syntax:**
  - External look of a language
  - Specified by a grammar
- **Semantics:**
  - Meaning of a language
  - Different ways of specifying it
- **Pragmatics:**
  - Other aspects of the language
  - Implementation issues

Language analysis

- ***Procedural*** **languages:**
  - ***Specify the action by a sequence of steps***
  - ***Examples: C, Pascal, VHDL, Verilog***
- ***Declarative*** **languages:**
  - **Specify the problem by a set of declaration**
  - **Example: Prolog**

Language analysis

- ***Imperative Semantics:***
  - **Dependence between the assignments and the values that variables can take**
  - **Examples C, Pascal**
- ***Applicative semantics:***
  - **Based on function invocation**
  - **Examples: Lisp, Silage**

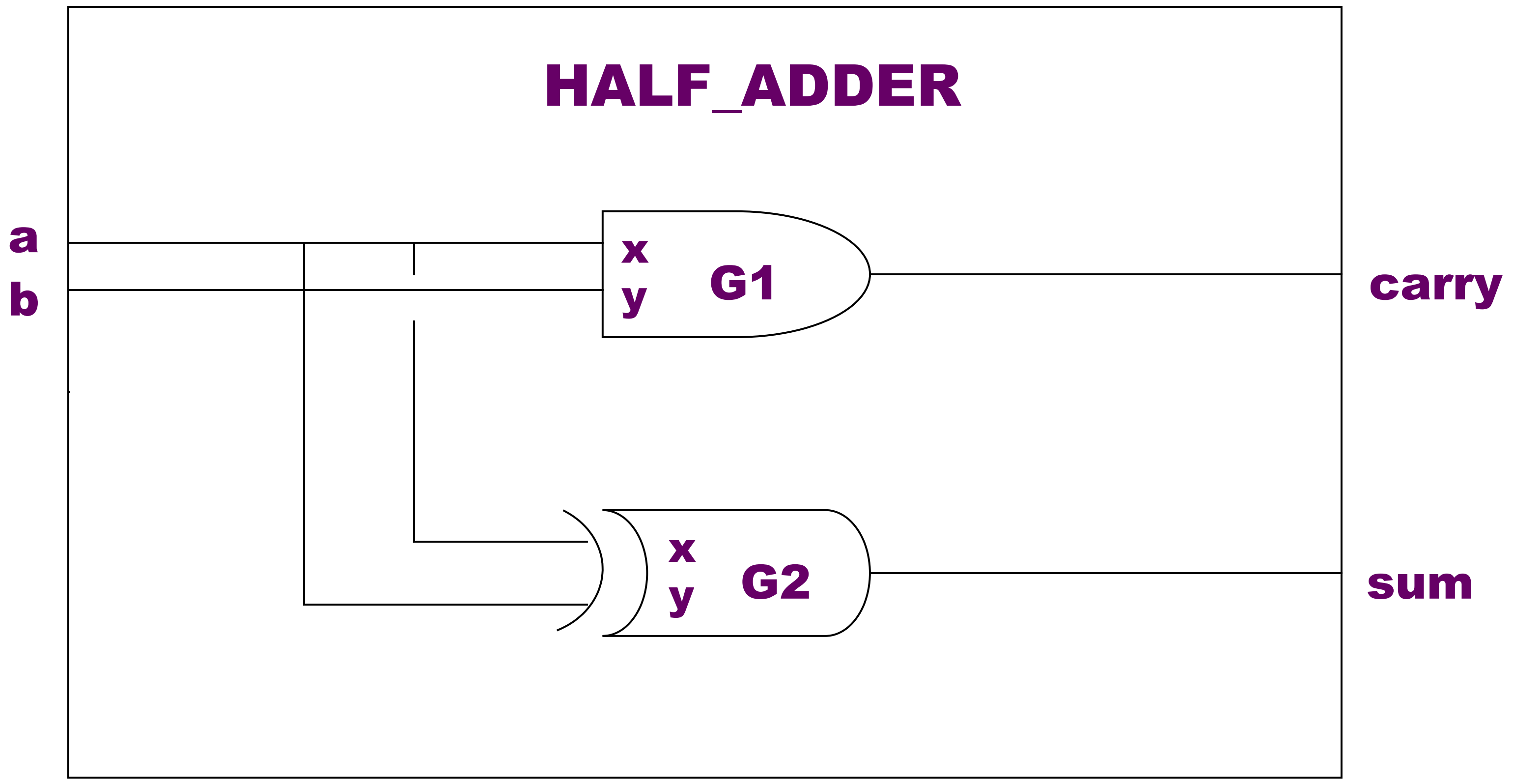
Hardware languages and views

- **Physical view:**
  - Physical layout languages
  - Declarative or procedural
- **Structural view:**
  - Structural languages
  - Declarative (with some procedural features)
- **Behavioral view:**
  - Behavioral languages
  - Mainly procedural

Structural view

- Composition of blocks
- Encoding of a schematic
- Incidence structure
- Hierarchy and instantiation
- HDL examples:
  - VHDL, Verilog HDL, ...

Example
(half adder)



Verilog example

Structural representation

```
module   HALF_ADDER (a , b , carry , sum);
                    input     a , b;
                    output   carry, sum;
                    and
                              g1 (carry, a , b);
                    xor
                              g2 (sum, a , b);
endmodule
```

Behavioral view
procedural languages

- Set of tasks with partial order:
  - *Architectural-level:*
    - Tasks: generic operations.
  - *Logic-level:*
    - Tasks: logic functions.
- Independent of implementation choices
- HDL examples:
  - VHDL, Verilog HDL, ...

Verilog example

Behavior of combinational logic circuit

```
module    HALF_ADDER(a , b , carry ,sum);

                    input       a , b;

                    output    carry, sum;

                                        assign carry = a & b ;

                                        assign sum  = a ^ b ;

endmodule
```

Verilog example
behavior of sequential logic circuit

```
module DIFFEQ (x, y, u , dx, a, clock, start);
Input          [7:0]          a, dx;
inout         [7:0]          x, y, u;
input                        clock, start;
reg   [7:0]       xl, ul, yl;
always
begin
             wait ( start);
             while ( x < a )
                         begin
                         xl = x + dx;
                         ul = u - (3 * x * u * dx) - (3 * y * dx);
                         yl = y + (u * dx);
                         @(posedge clock);
                         x = xl; u = ul ; y = yl;
                         end
endmodule
```

System Verilog

- Extensions to Verilog HDL
  - Modeling:
    - Transaction-level modeling
      - Higher abstraction level
    - Direct Programming interface
      - Enables calls to C/C++/SystemC
      - Co-simulation Verilog/SystemC
    - Interface modeling with encapsulation
      - Support bus-intensive design
      - IP protection by nesting modules
  - Verification:
    - Procedural assertions
      - Built into the language
      - Avoid recoding errors, increase test accuracy
- Proposed by Accellera
  - To be standardized by IEEE

Timing semantics
(event-driven semantics)

- Digital synchronous implementation
- An operation is triggered by some event:
  - If the inputs to an operation change, the operation is re-evaluated
- Used by simulators for efficiency reasons

Synthesis policy for VHDL and Verilog

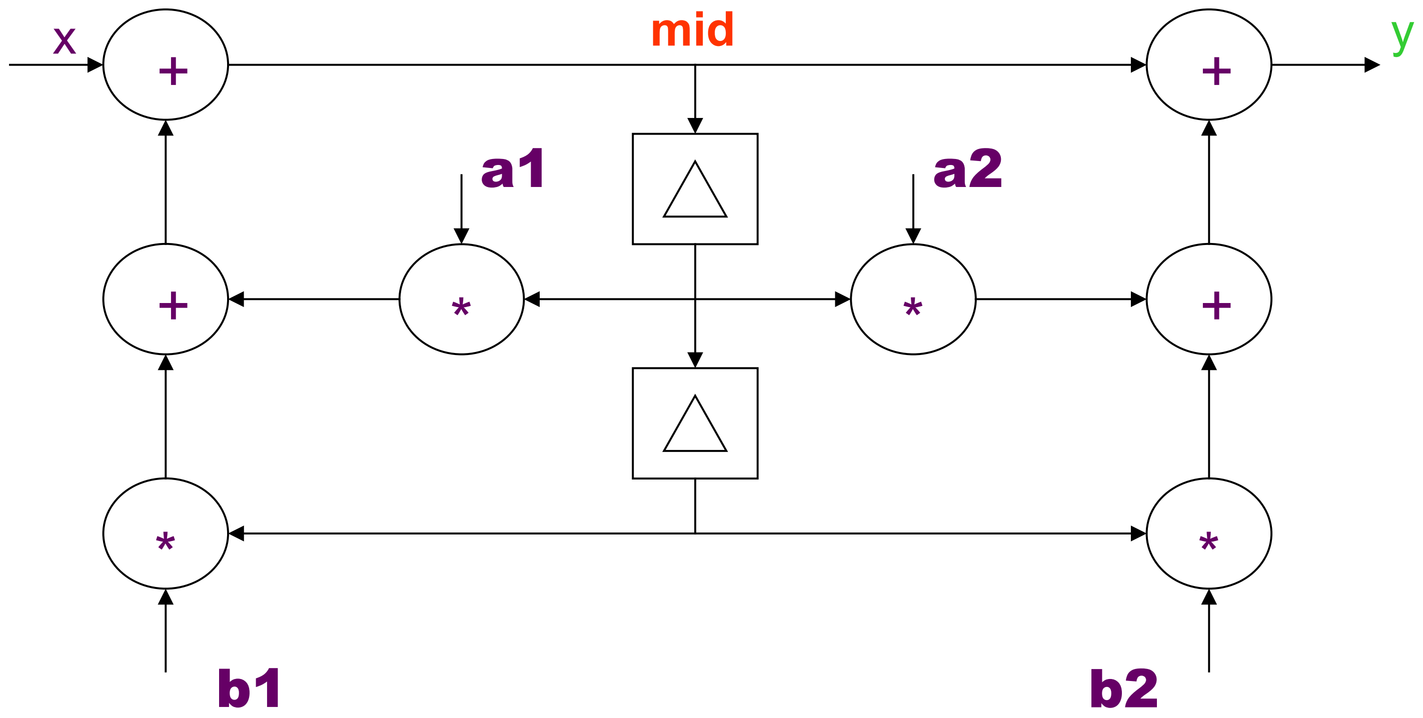
- **Operations are synchronized to a clock**

  by using a wait (or @) command
- **Wait and @ statements delimit clock boundaries**
- **Clock is a parameter of the model:**
  - **Model is updated at each clock cycle**

Behavioral view
declarative languages

- **Combinational circuits:**
  - Set of untimed assignments.
  - Each assignment represents a virtual logic gate
  - Very similar to procedural models
- **Sequential circuits:**
  - Use timing annotation for delayed signals
  - Set of assignments over (delayed) variables

Silage example



```
function IIR ( a1, a2 , b1, b2, x: num)  /* returns */   y : num =
begin
                    y = mid + a2 * mid@1 + b2 * mid@2;
                    mid = x + a1 * mid@1 + b1 * mid@2;
end
```

Hardware primitives

- Hardware basic units:
  - Logic gates
  - Registers
  - Black-boxes
    - e.g., Complex units, RAMs
- Connections
- Ports

Semantics of variables

- Variables are implemented in hardware by:
  - Registers
  - Wires
- The hardware can store information or not
- Two cases:
  - Combinational circuits
    - Resolution policy for multiple assignment to a variable
  - Sequential circuits
    - Variables keep values until reassigned

SystemC

- Objectives:
  - Model Hw with Sw programming language
  - Achieve fast simulation
  - Provide support for hw/sw system design
- Requirement:
  - Give hw semantics to sw models
- Supported by a large consortium of semiconductor and EDA companies

SystemC

- C++ class library and modeling methodology
  - Hw semantics defined through the class library
- Object-oriented style
  - Components and encapsulation
- No language restriction or addition
- Some hw synthesis support

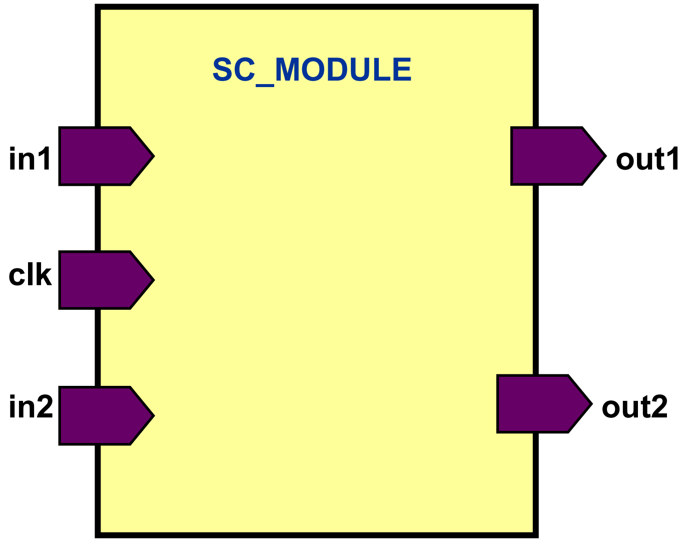
SystemC features

- Enable C++ without extending the language (syntax)

  - use classes

Concurrency	Processes
Hardware Data Types	bit vectors, arbitrary precision signed and unsigned integers, fixed-point numbers
Notion of Time	Clocks
Reactive Behavior	Watching
Communication	Signals, protocols

SystemC Classes Modules and Ports

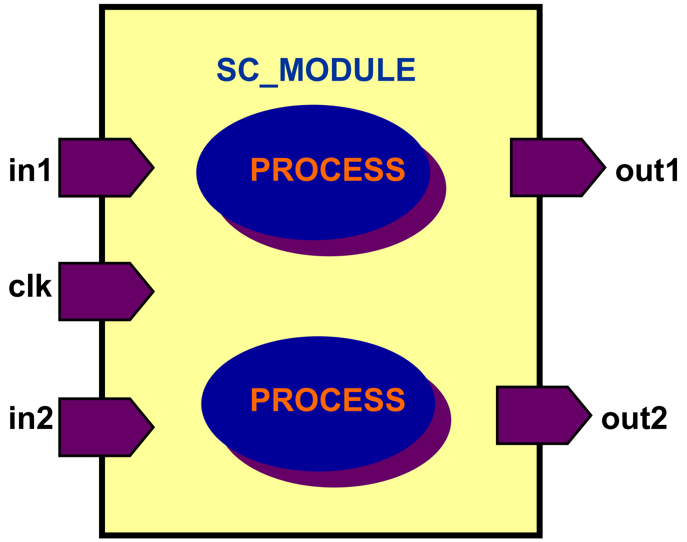


- Modules (`sc_module`)
  - Fundamental structural entity
  - Contain processes
  - Contain other modules (creating hierarchy)

- **Ports(`sc_in<>,sc_out<>,sc_inout<>`)**
  - **Modules have ports**
  - **Ports have types**
  - **A process can be made sensitive to ports/signals**

SystemC Classes - Processes

- Processes
  - Functionality is described in a process
  - Processes run concurrently
  - Code inside a process executes sequentially
  - SystemC has three different types of processes
    - SC_METHOD
    - SC_THREAD
    - SC_CTHREAD



Process types

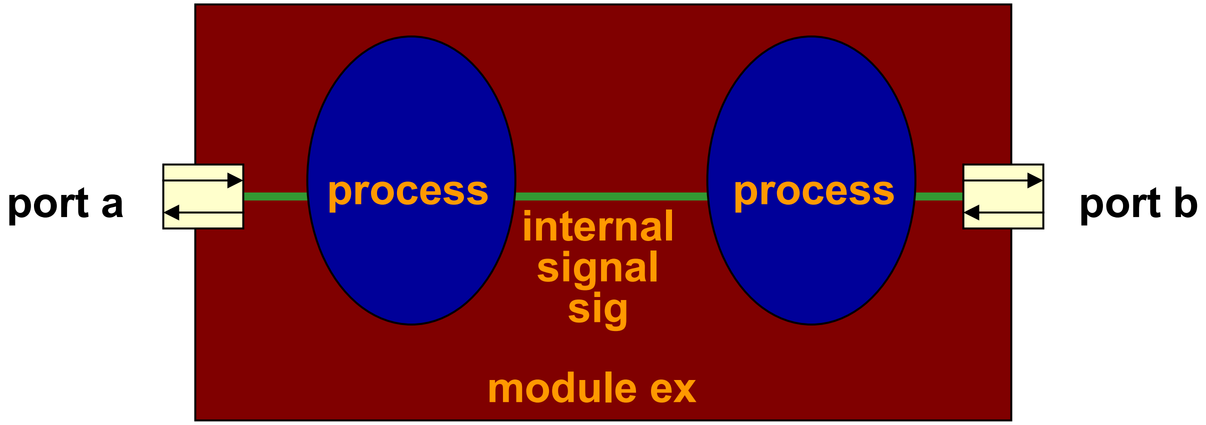
- *sc_method*: method process
  - sensitive to a set of signals
  - executed until it returns
- *sc_thread*: thread process
  - sensitive to a set of signals
  - executed until a *wait()*
- *sc_cthread*: clocked thread process
  - sensitive only to one edge of clock
  - execute until a *wait()* or a *wait_until()*
  - *watching(reset)* restarts from top of process body (reset evaluated on active edge)

RTL style

Testbench

Architectural style

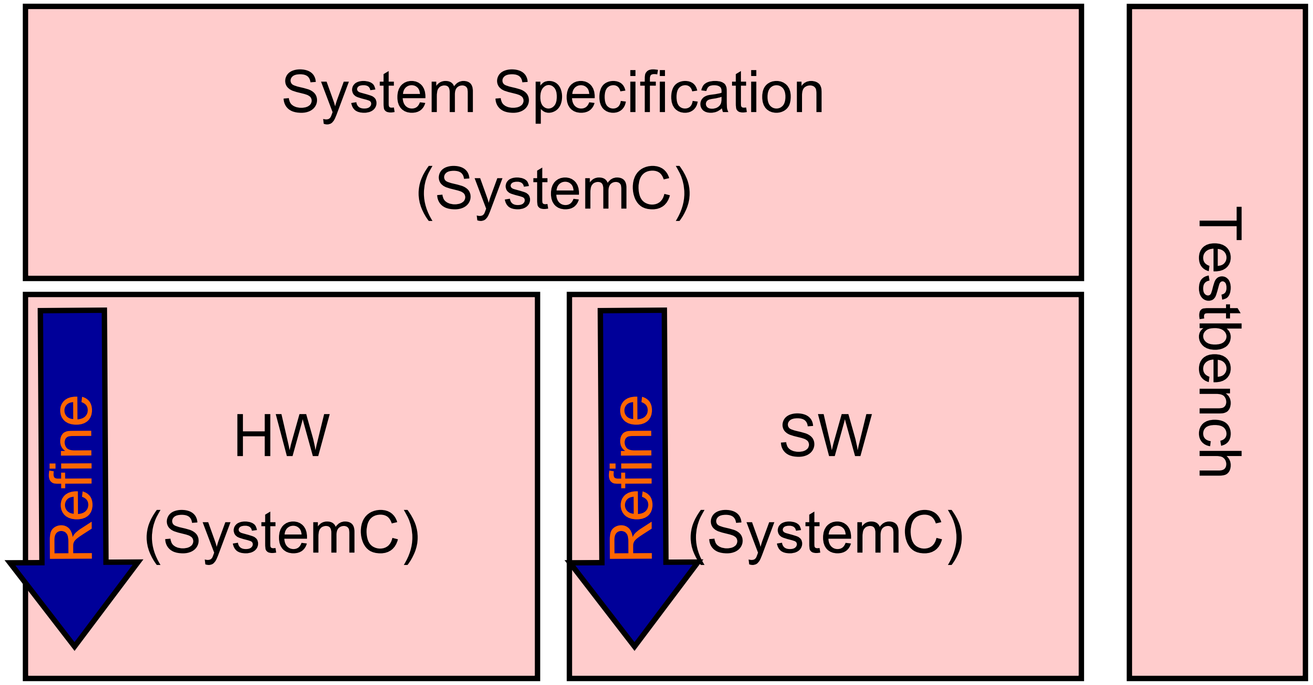
Execution of processes

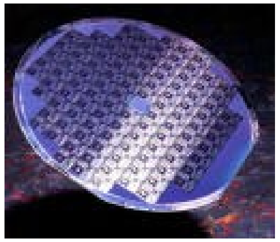


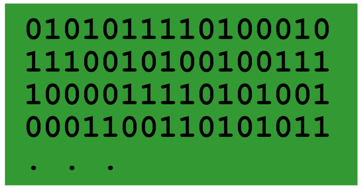
- Not hierarchical, communicate through signals
- Execution and signal updates
  - request-update semantics

    1. execute all processes that can be executed
    2. update the signals written by the processes
    3. => other processes to be executed

SystemC Design Vision







- **SystemC as a single design language**

Module 3

- **Objectives**
  - **Abstract models**
  - **The sequencing graph abstraction and its properties**

Abstract models and intermediate formats

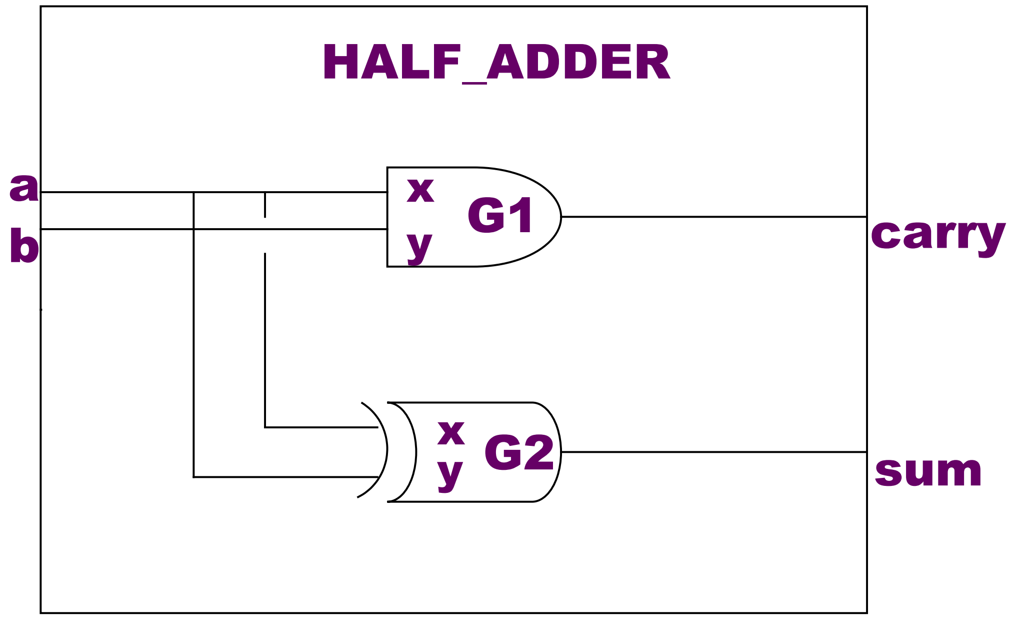
- **Abstract models:**
  - **Models based on graphs and discrete mathematics**
  - **Useful for problem formalization, algorithm development and reasoning about properties**
- **Intermediate forms:**
  - **ASCII or binary representations of abstract models**
  - **Derived from language models by compilation**

Abstract models
Examples

- **Netlists:**
  - **Structural views**
- **Logic networks:**
  - **Mixed structural/behavioral views**
- **State diagrams:**
  - **Behavioral views of sequential models**
- **Dataflow and sequencing graphs:**
  - **Abstraction of behavioral models**

Netlist

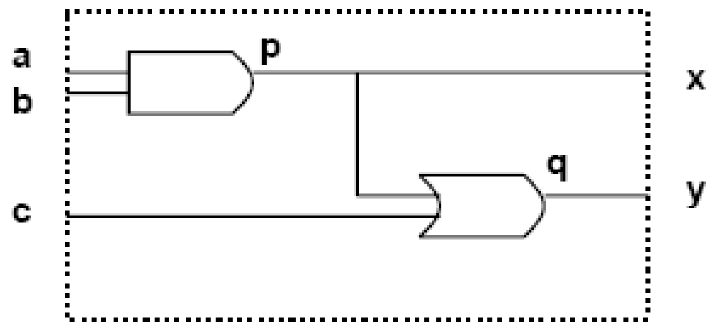
- Module-oriented netlist
  - G1: a,b,carry
  - G2: a,b,sum

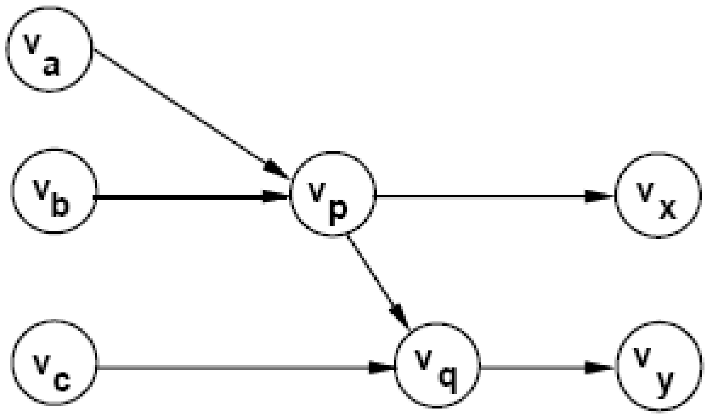


Logic network

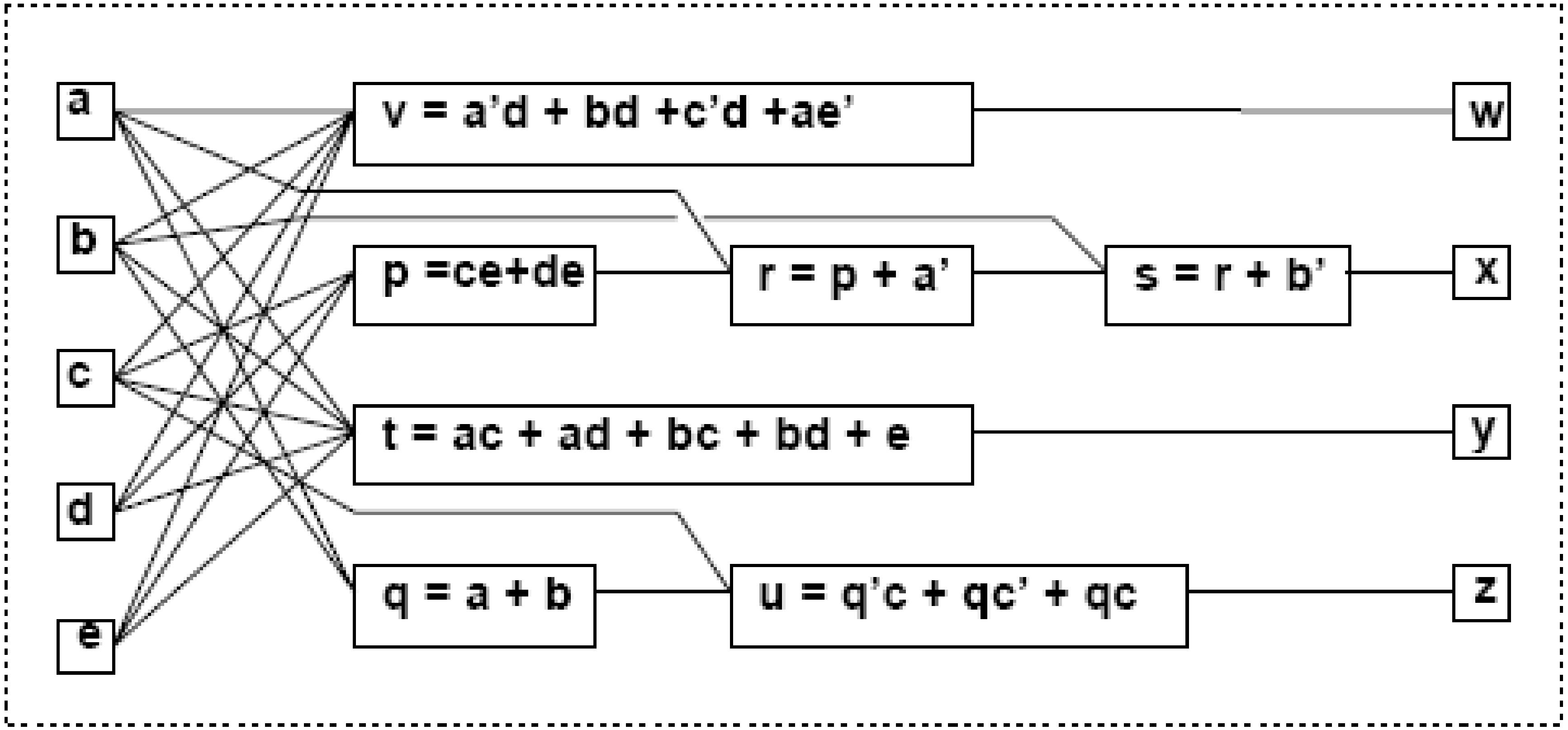
- Logic network
  - An interconnection of blocks
    - Each block modeled by a Boolean function
  - Usual restrictions:
    - Acyclic and memoryless
    - Single-output functions
- The model has a structural/behavioral semantics
  - The structure is induced by the interconnection
- Mapped network
  - Special case when the blocks correspond to library elements

Example of mapped network





Example of general network



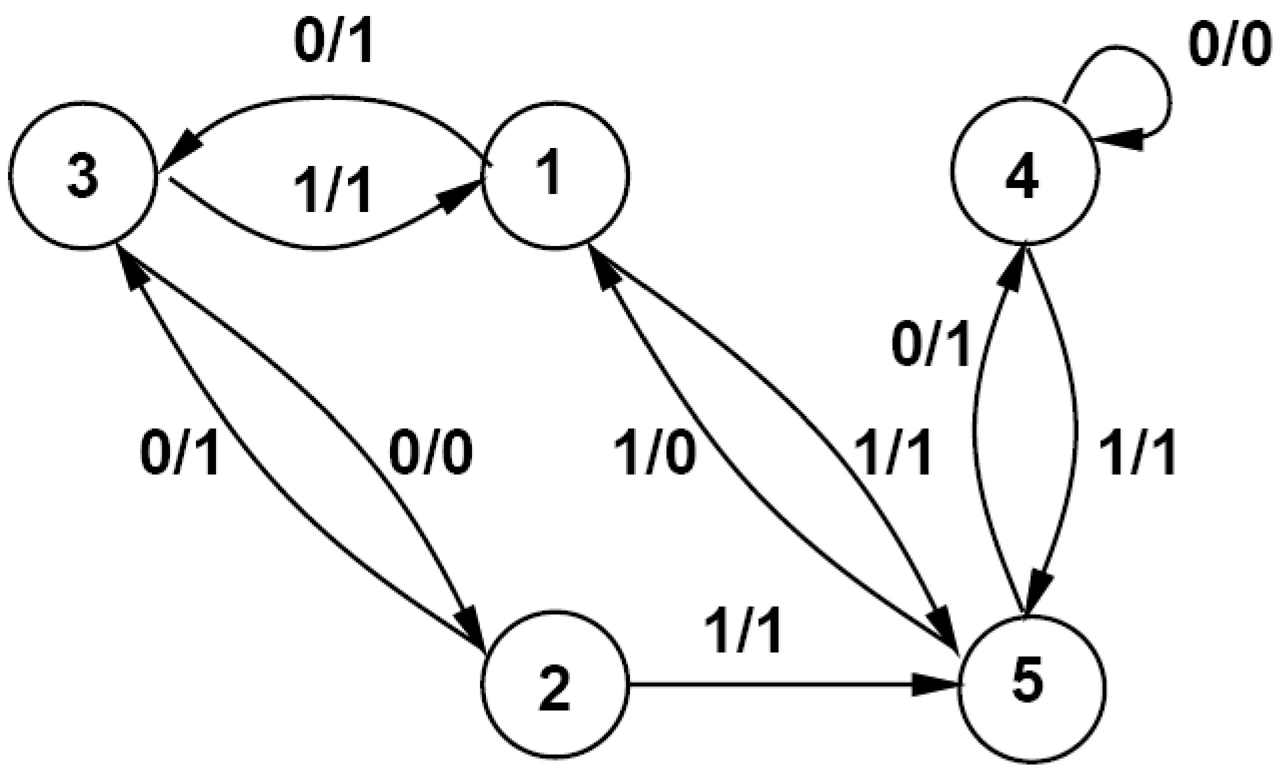
Formal finite-state machine model

- A set of primary input patterns X
- A set of primary output 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

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



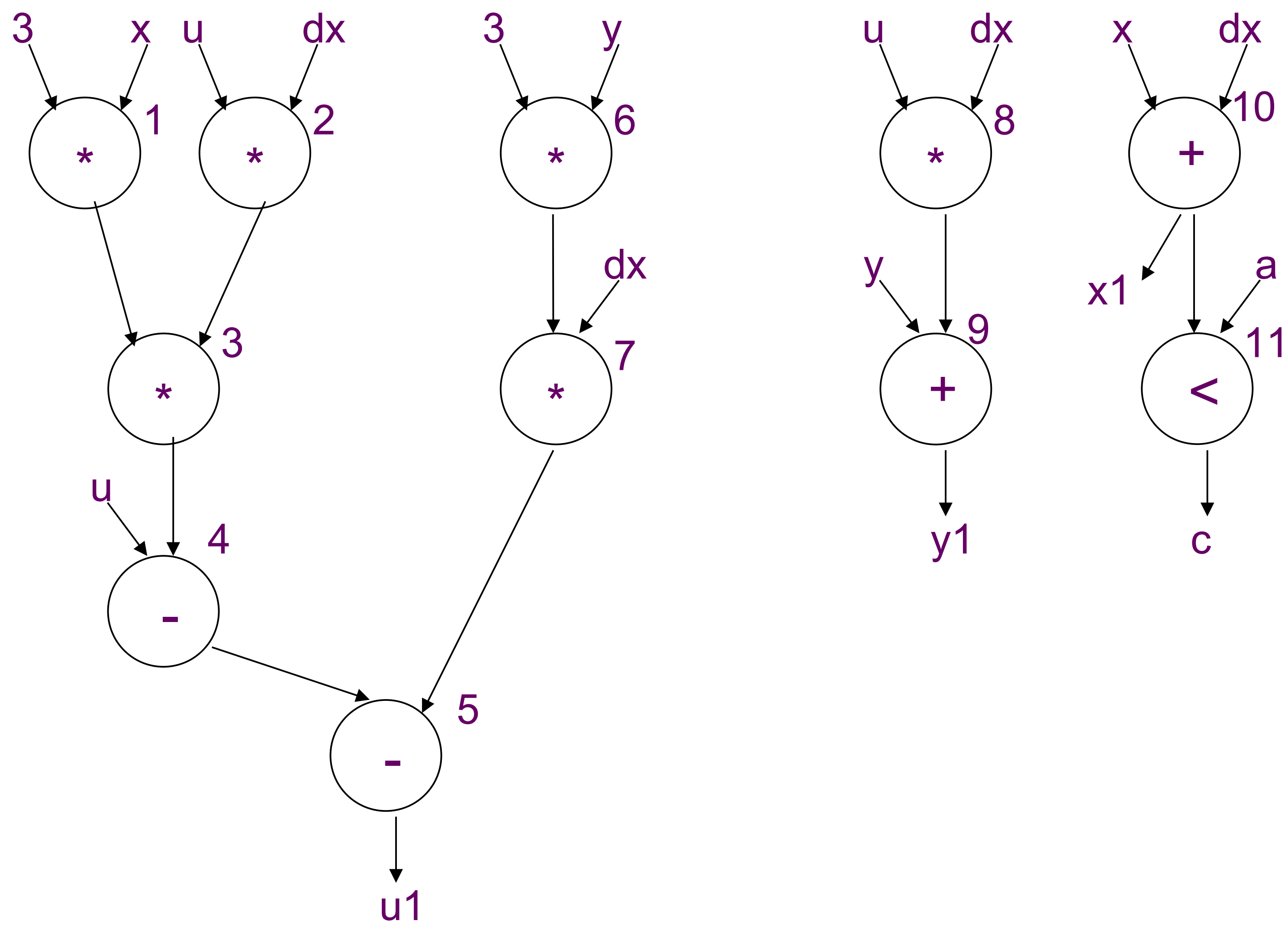
Dataflow graphs

- **Behavioral views of architectural models**
- **Useful to represent data-paths**
- **Graph:**
  - **Vertices = operations**
  - **Edges = dependencies**

Example
Differential equation solver -- loop body

```
diffeq {
    read ( x, y, u, dx, a ) ;
    repeat {
        xl = x + dx;
        ul = u – ( 3 . x . u . dx ) – ( 3 . y . dx ) ;
        yl = y + u . dx ;
        c = x < a ;
        x = xl; u = ul; y = yl ;
    until ( c );
write ( y )
}
```

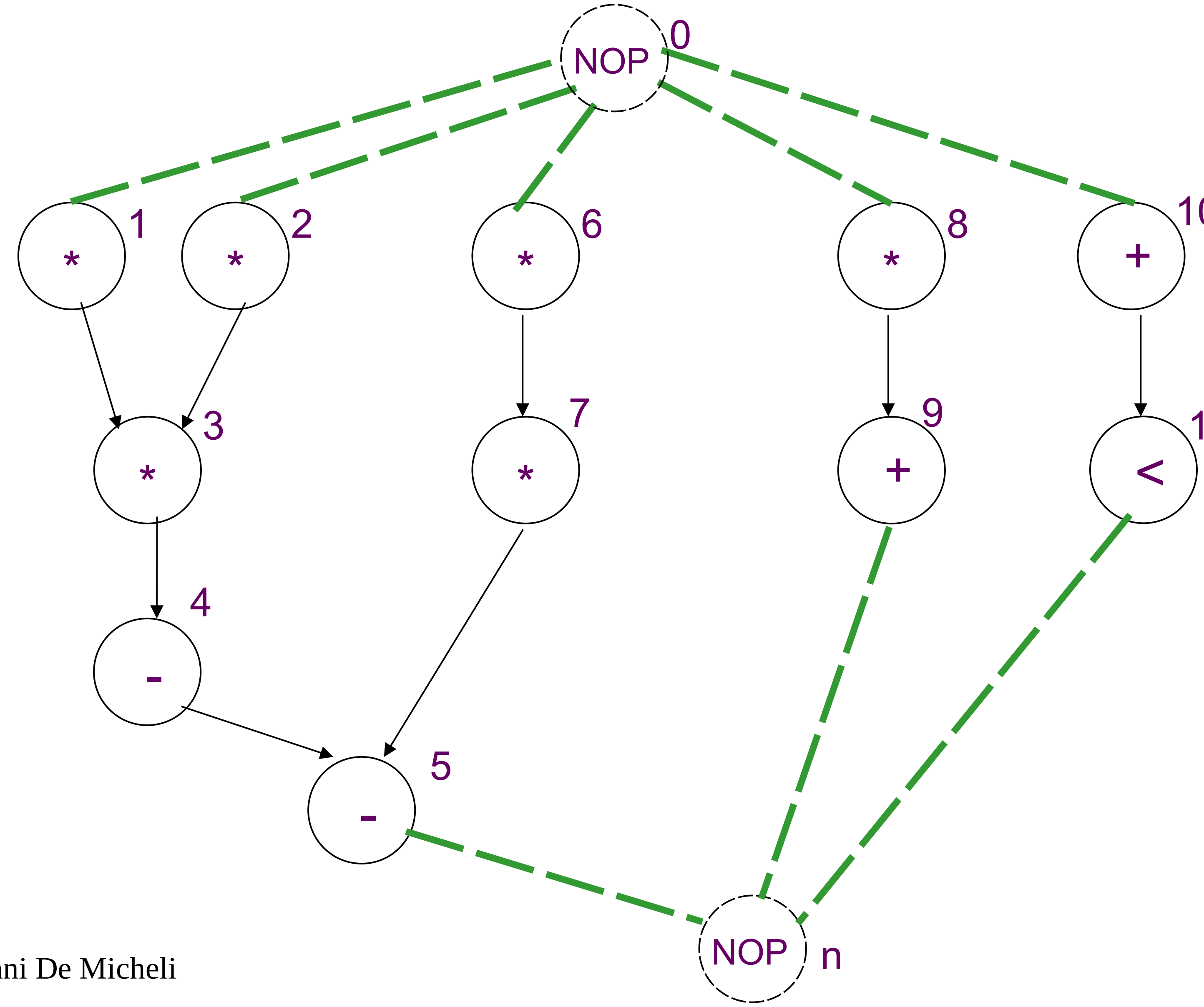
Example



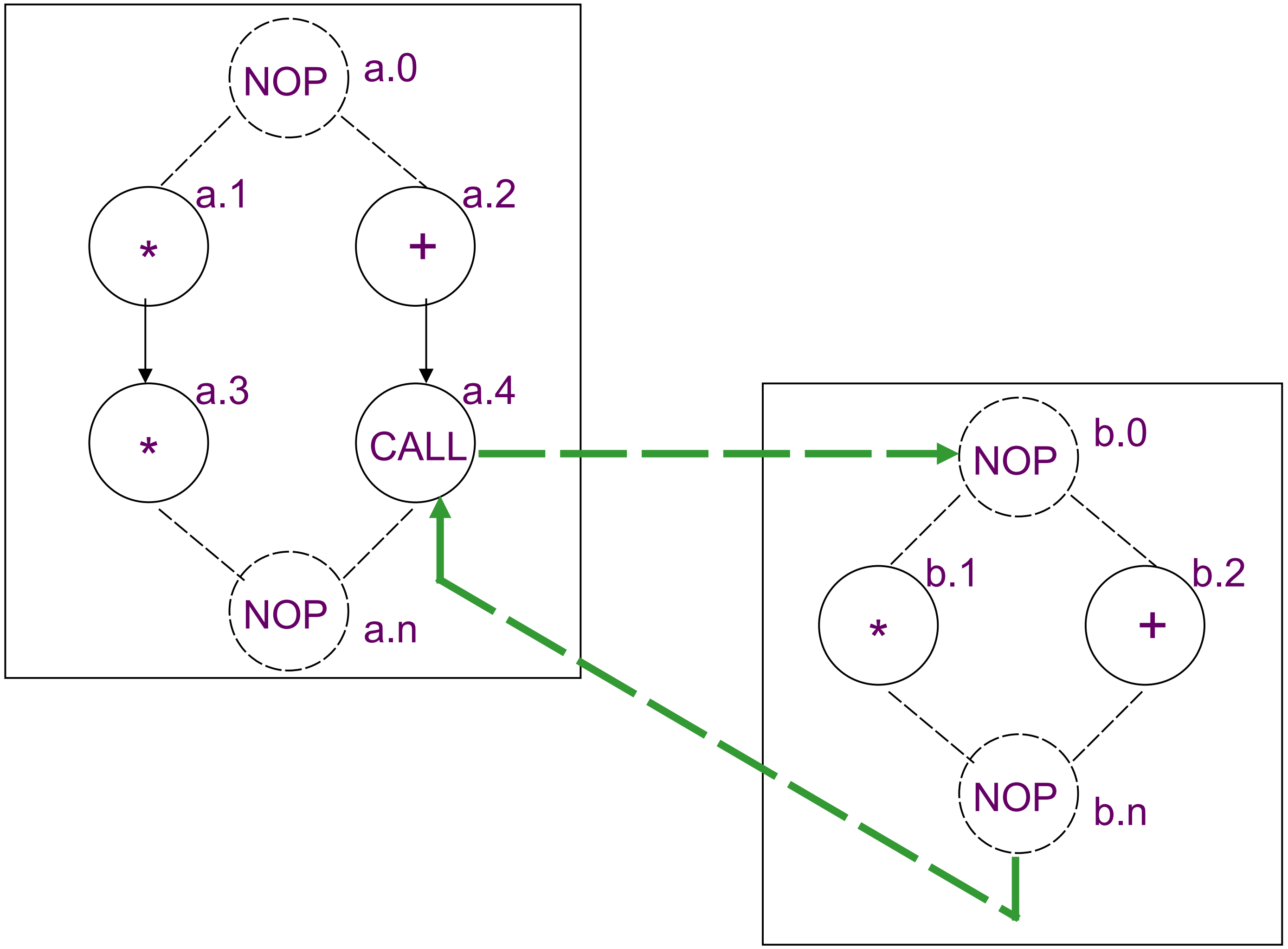
Sequencing graphs

- Behavioral views of architectural models
- Useful to represent data-path and control
- Extended dataflow graphs:
  - Operation serialization
  - Hierarchy
  - Control-flow commands:
    - Branching and iteration.
- Polar graphs:
  - Source and sink

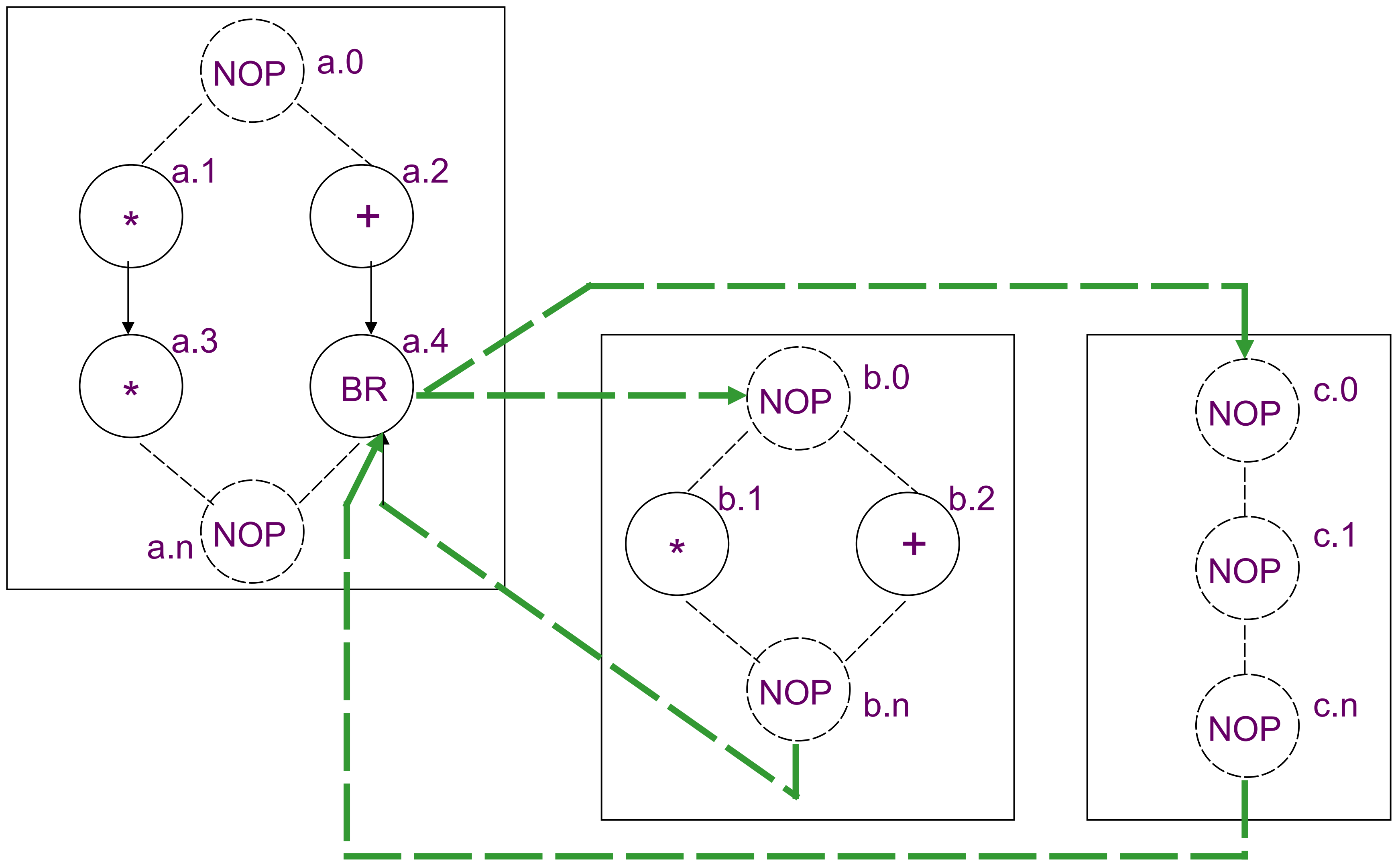
Example



Example of hierarchy



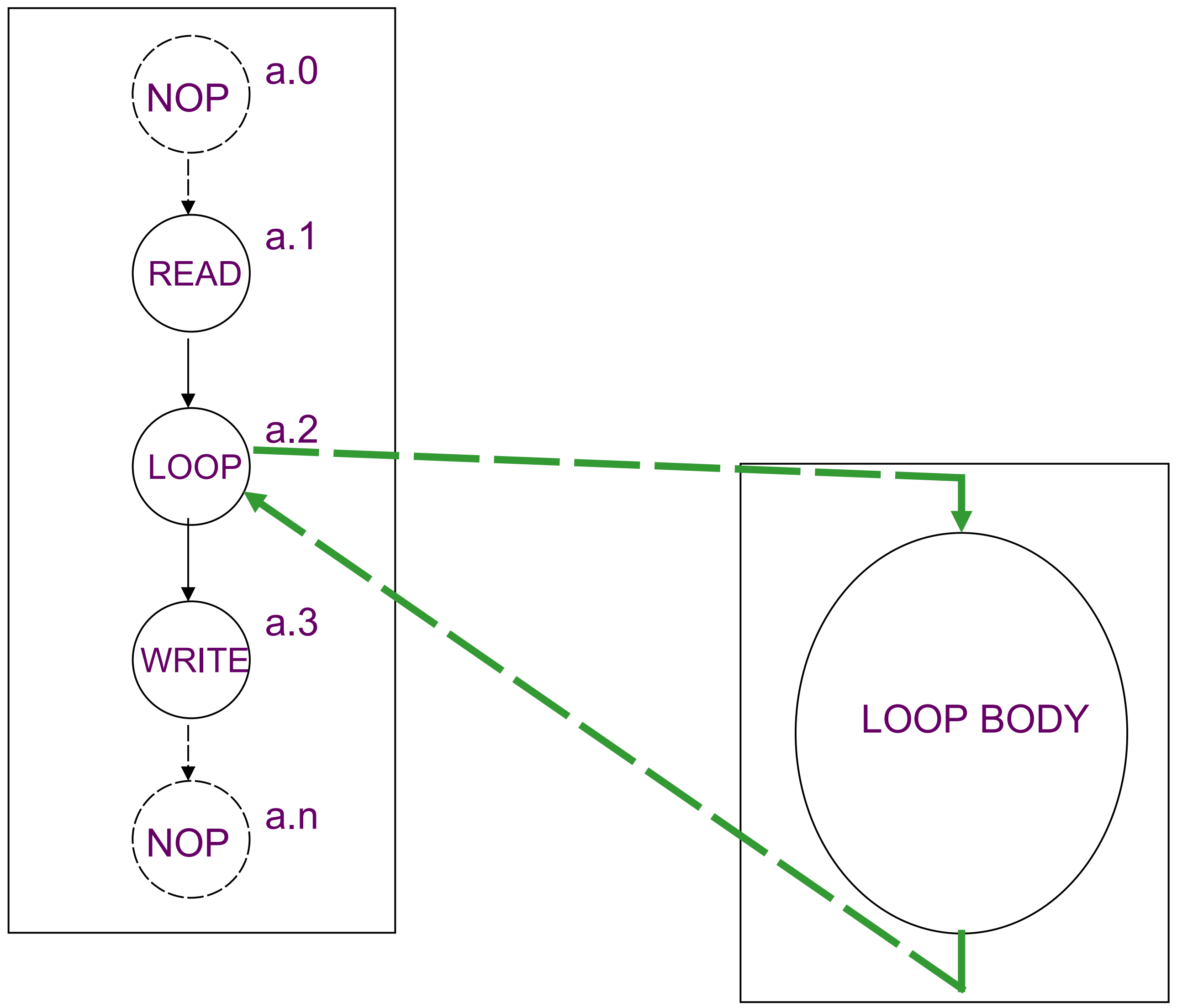
Example of branching



Example of iteration

```
diffeq {
        read (x; y; u; dx; a);
        repeat {
            xl = x + dx;
            ul = u – (3 · x · u · dx ;
            yl = y + u · dx ;
            c = x > a;
            x = xl ; u = ul ; y = yl ;
            }
        until ( c )
write (y) ;
}
```

Example of iteration



Semantics of sequencing graphs

- Marking of vertices:
  - Waiting for execution
  - Executing
  - Having completed execution
- Execution semantics:
  - An operation can be fired as soon as all its immediate predecessors have completed execution

Vertex attributes

- Area cost
- Delay cost:
  - Propagation delay
  - Execution delay
- Data-dependent execution delays:
  - Bounded (e.g. branching)
  - Unbounded (e.g. iteration, synchronization)

Properties of sequencing graphs

- Computed by visiting hierarchy bottom-up
- Area estimate:
  - Sum of the area attributes of all vertices
  - Worst-case – no sharing
- Delay estimate (latency):
  - Bounded-latency graphs
  - Length of longest path from source to sink

Summary

- Hardware synthesis requires specialized language support:
  - VHDL and Verilog HDL are mainly used today
    - Similar features
    - Simulation-oriented
- Synthesis from programming languages is also possible:
  - Hardware and software models of computation are different
  - Appropriate hw semantics need to be associated with programming languages
    - SystemC
- Abstract models:
  - Capture essential information
  - Derivable from HDL models
  - Useful to prove properties and for algorithm development.