

MODELING LANGUAGES
AND
ABSTRACT MODELS

© *Giovanni De Micheli*

Stanford University

Outline

© GDM

- Hardware modeling issues:
 - Representations and models.
- Issues in hardware languages.
- Abstract hardware models:
 - Dataflow and sequencing graphs.

Circuit modeling

© GDM

- Formal methods:
 - Models in hardware languages.
 - Flow and state diagrams.
 - Schematics.

- Informal methods:
 - Principles of operations.
 - Natural-language descriptions.

Hardware Description Languages

© GDM

- 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

© GDM

- 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

© GDM

- 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.

Language analysis

© GDM

- *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

© GDM

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

Language analysis

© GDM

- *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

© GDM

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

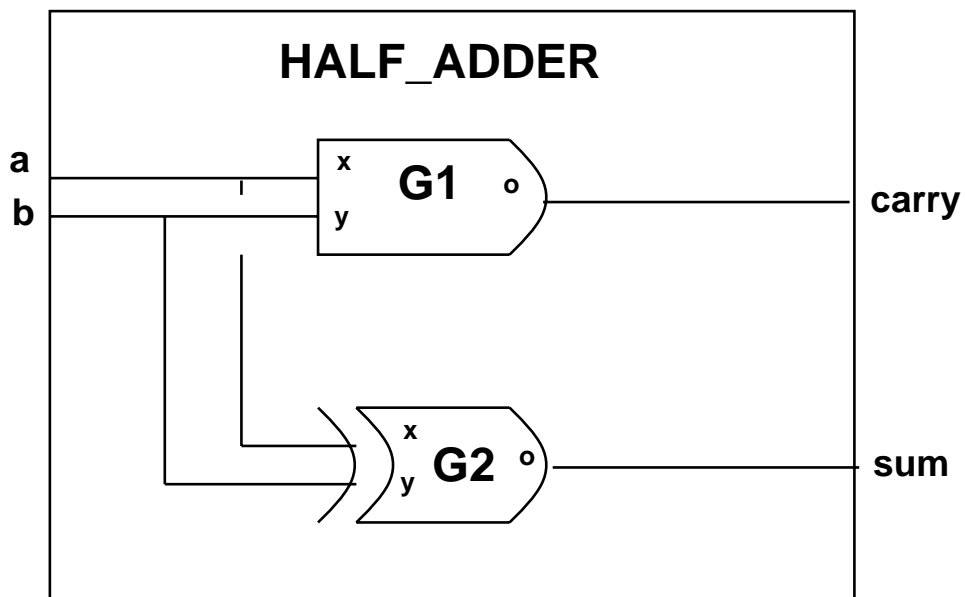
Structural view

© GDM

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

Example (half adder)

© GDM



Verilog example structural representation

© GDM

```
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

© GDM

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

Verilog example

Behavior of combinational logic circuit

© GDM

```
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

© GDM

```
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]  x1, u1, y1;
always
begin
    wait ( start);
    while ( x < a )
        begin
            x1 = x + dx;
            u1 = u - (3 * x * u * dx) - (3 * y * dx);
            y1 = y + (u * dx);
            @(posedge clock);
            x  = x1; u = u1 ; y = y1;
        end
endmodule
```

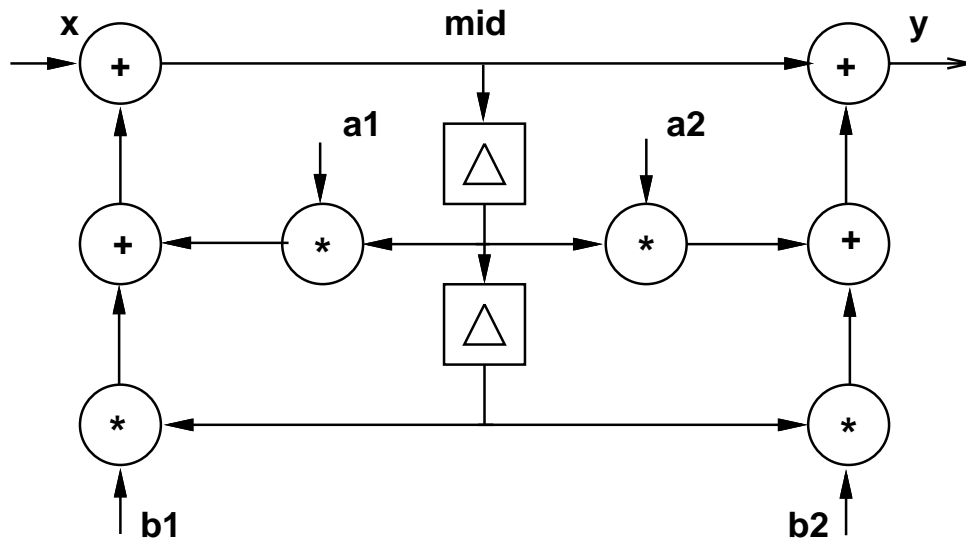

Behavioral view declarative languages

© GDM

- 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

© GDM



Silage example

© GDM

```
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
```

Issues in hardware languages

© GDM

- Mixing behavior and structure.
 - Controlling some implementation details.
- Primitive elements and variable semantics.
 - Multiple-assignment problem.
- Timing semantics.
 - Synthesis policies.

Behavior versus structure

© GDM

- Express partitions in design.
- Pure behavior is hard to specify.
 - I/O ports imply a structure.
 - Hierarchy may imply structure.
- Hybrid representations.

Example

© GDM

- Pipelined processor design
- Pipeline is an implementation issue.
- A behavioral representation should **not** specify the pipeline.
- Most processor *instruction sets* are conceived with an implementation in mind.
- The behavior is defined to fit an implementation model.

Hardware primitives

© GDM

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

Semantics of variables

© GDM

- Variables are implemented in hardware by:
 - Registers.
 - Wires.
- The hardware can store information or not.
- Cases:
 - Combinational circuits.
 - Sequential circuits.

Semantics of variables

© GDM

- Combinational circuits.
- Multiple-assignment to a variable.
- Conflict resolution.
 - Oring (YLL).
 - Last assignment.

Semantics of variables

© GDM

- Sequential circuits.
- Multiple-assignment to a variable.
- Variable retains its value until reassigned.
- Problem:
 - Variable propagation and observability.

Example

© GDM

- Multiple reassignments:
 - $x = 0 ; x = 1 ; x = 0 ;$
- Interpretations:
 - Each assignment takes a cycle. → pulse.
 - x assumes value 0.
 - x assumes value 0 after a short glitch.

Timing semantics

© GDM

- Most procedural HDLs specify a *partial order* among operations.
- What is the timing of an operation?
 - A posteriori model:
 - * Delay annotation.
 - A priori model:
 - * Timing constraints.
 - * Synthesis policies.

Timing semantics (event-driven semantics)

© GDM

- 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

© GDM

- 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.

Verilog example

behavior of sequential logic circuit

© GDM

```
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]  x1, u1, y1;
always
begin
    wait ( start);
    while ( x < a )
        begin
            x1 = x + dx;
            u1 = u - (3 * x * u * dx) - (3 * y * dx);
            y1 = y + (u * dx);
            @(posedge clock);
            x  = x1; u = u1 ; y = y1;
        end
endmodule
```

Abstract models

© GDM

- Models based on graphs.
- Useful for:
 - Machine-level processing.
 - Reasoning about properties.
- Derived from language models by compilation.

Abstract models

Examples

© GDM

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

Dataflow graphs

© GDM

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

Example

© GDM

$$xl = x + dx$$

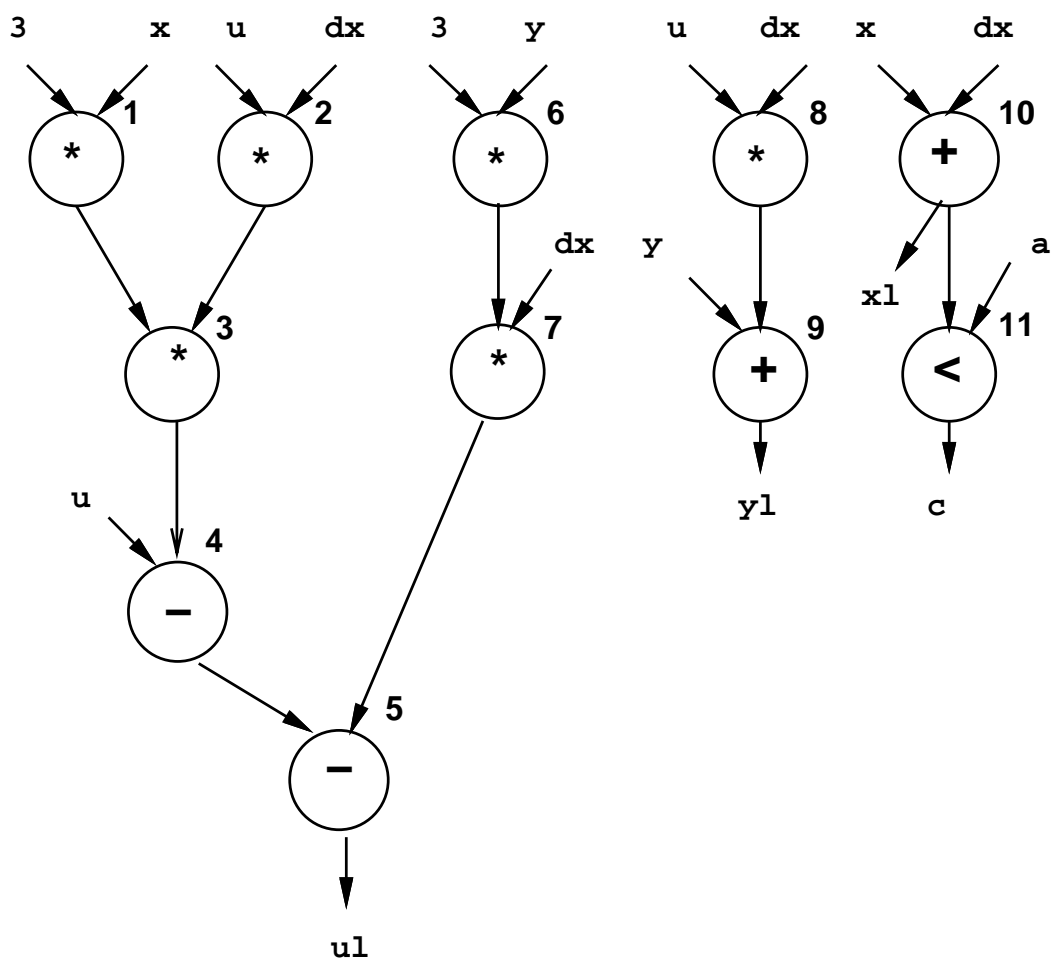
$$ul = u - (3 \cdot x \cdot u \cdot dx) - (3 \cdot y \cdot dx)$$

$$yl = y + u \cdot dx$$

$$c = xl < a$$

Example

© GDM



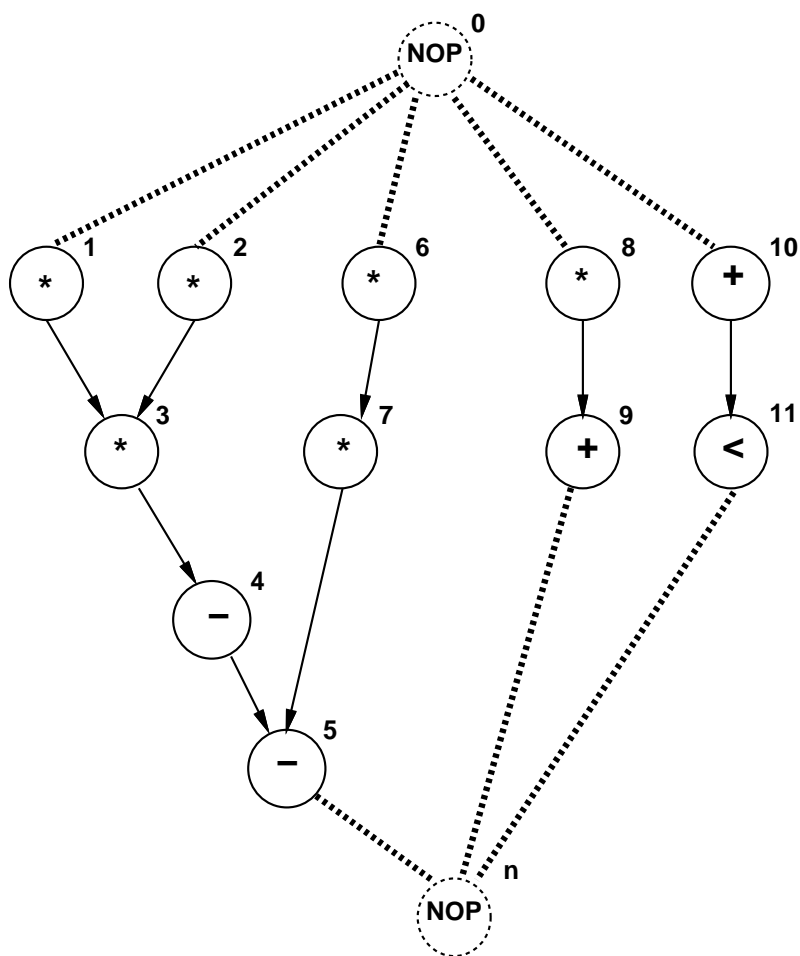
Sequencing graphs

© GDM

- 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: *source* and *sink*.

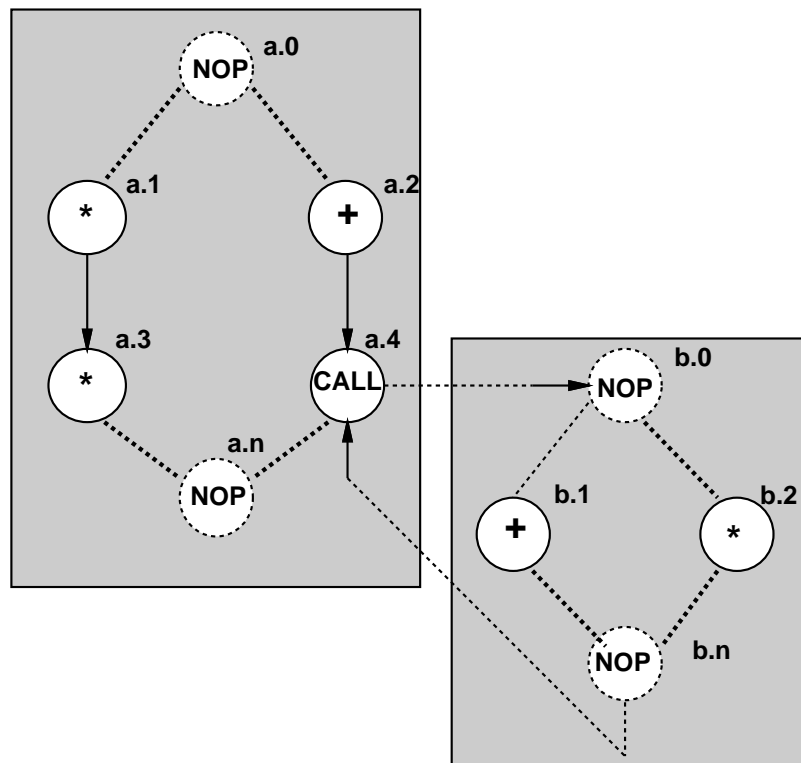
Example

© GDM



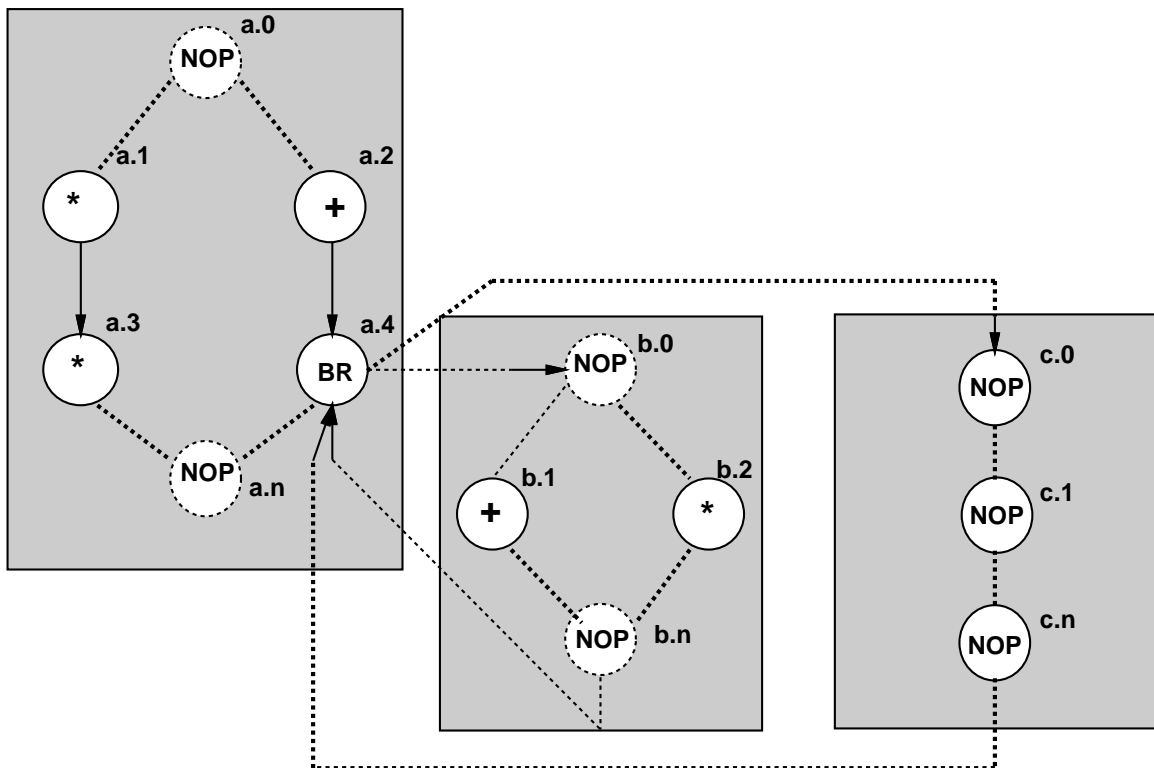
Example of hierarchy

© GDM



Example of branching

© GDM



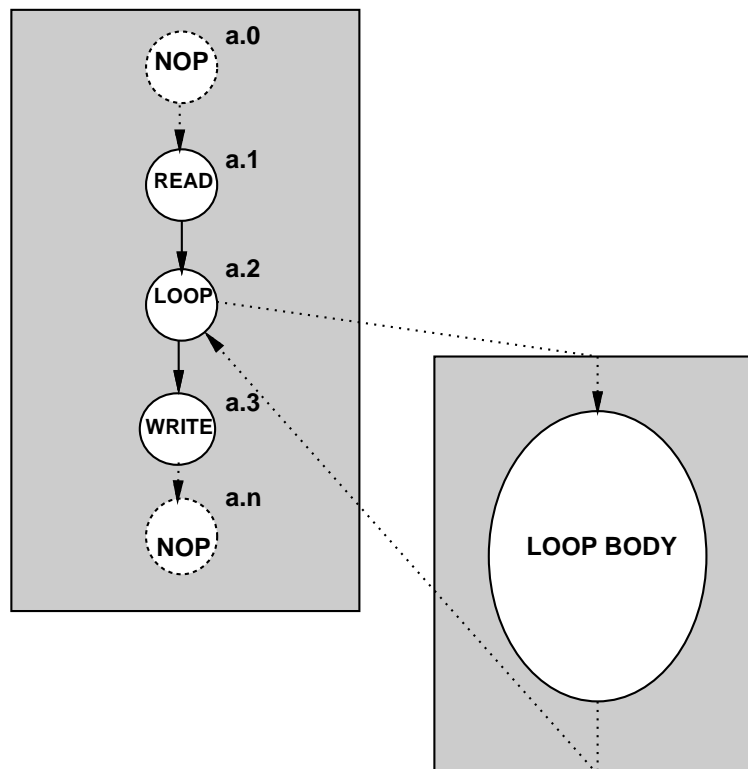
Example of iteration

© GDM

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

Example of iteration

© GDM



Semantics of sequencing graphs

© GDM

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

Vertex attributes

© GDM

- 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

© GDM

- 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.

Summary

© GDM

- 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 hardware semantics need to be associated with programming languages.
- Abstract models:
 - Capture essential information.
 - Derivable from HDL models.
 - Useful to prove properties.