

Architectural-Level Synthesis

Giovanni De Micheli
Integrated Systems Centre
EPF Lausanne



This presentation can be used for non-commercial purposes as long as this note and the copyright footers are not removed

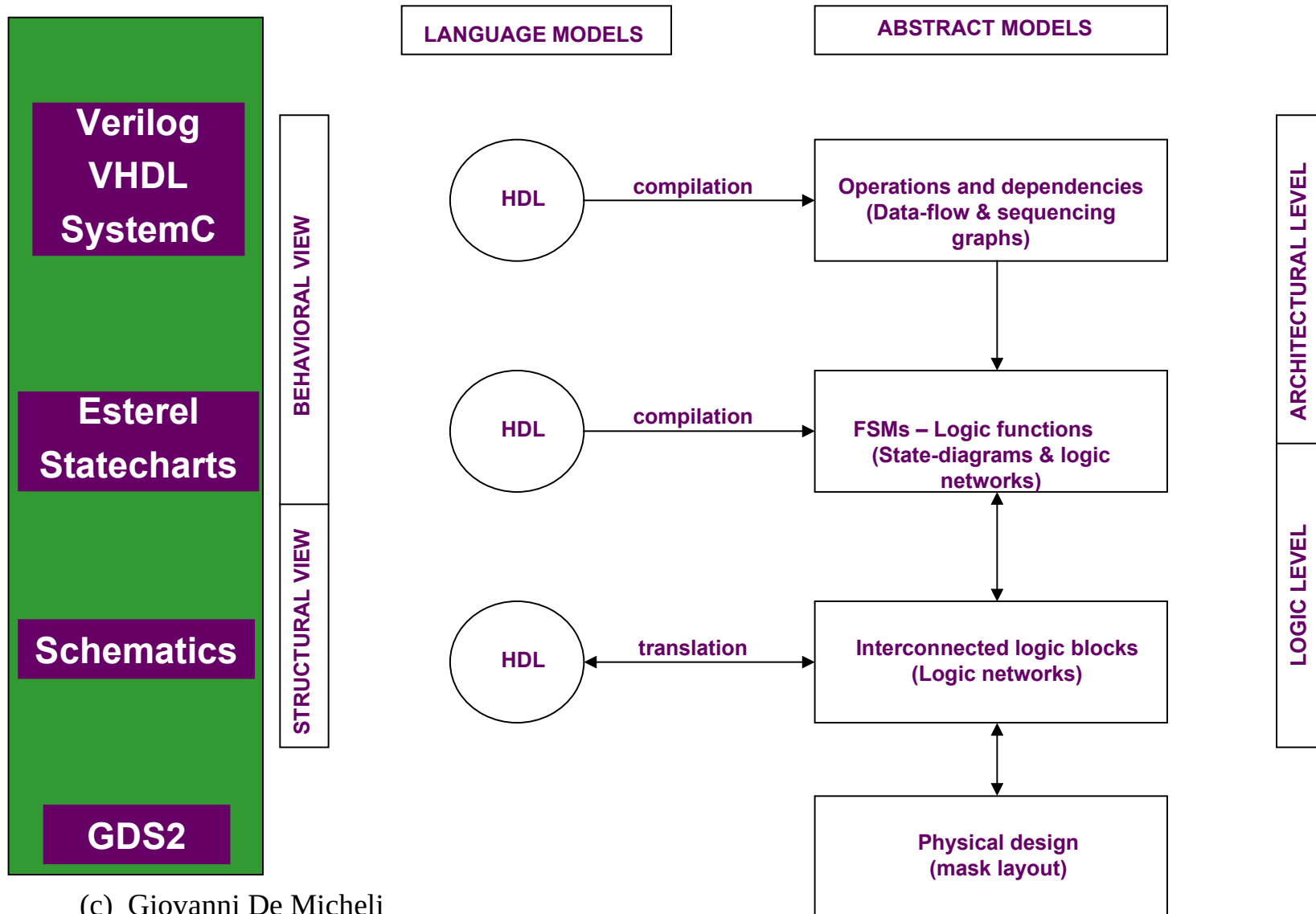
© Giovanni De Micheli – All rights reserved

Module1

◆ Objectives

- ▲ Motivation
- ▲ Compiling language models into abstract models
- ▲ Behavioral-level optimization and program-level transformations

Models and flows



Architectural-level synthesis motivation

- ◆ **Raise input abstraction level**
 - ▲ Reduce specification of details
 - ▲ Extend designer base
 - ▲ Self-documenting design specifications
 - ▲ Ease modifications and extensions
- ◆ **Reduce design time**
- ◆ **Explore and optimize macroscopic structure:**
 - ▲ Series/parallel execution of operations

Architectural-level synthesis

- ◆ **Translate HDL models into sequencing graphs**
- ◆ **Behavioral-level optimization:**
 - ▲ **Optimize abstract models independently from the implementation parameters**
- ◆ **Architectural synthesis and optimization:**
 - ▲ **Create macroscopic structure:**
 - ▼ **Data-path and control-unit**
 - ▲ **Consider area and delay information of the implementation**

Compilation and behavioral optimization

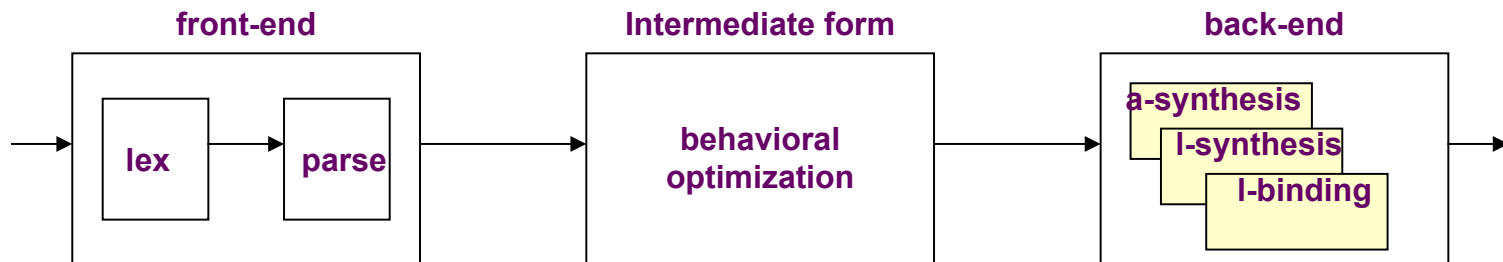
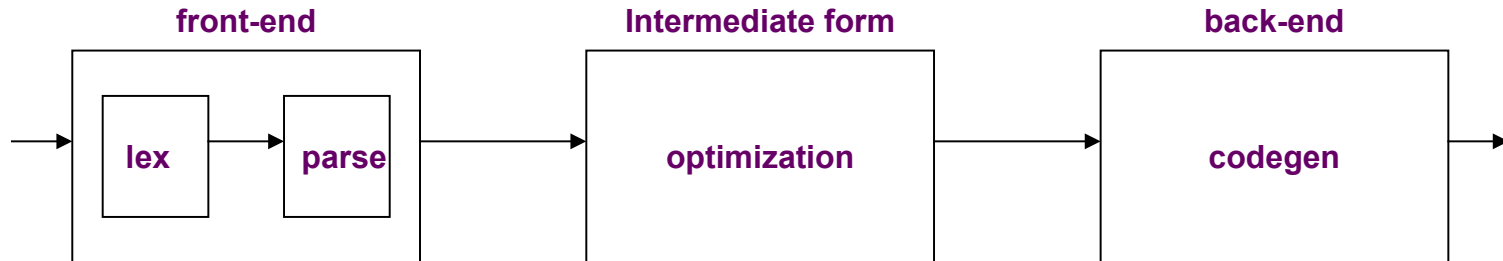
◆ Software compilation:

- ▲ Compile program into intermediate form
- ▲ Optimize intermediate form
- ▲ Generate target code for an architecture

◆ Hardware compilation:

- ▲ Compile HDL model into sequencing graph
- ▲ Optimize sequencing graph
- ▲ Generate gate-level interconnection for a cell library

Hardware and software compilation



Compilation

- ◆ **Front-end:**

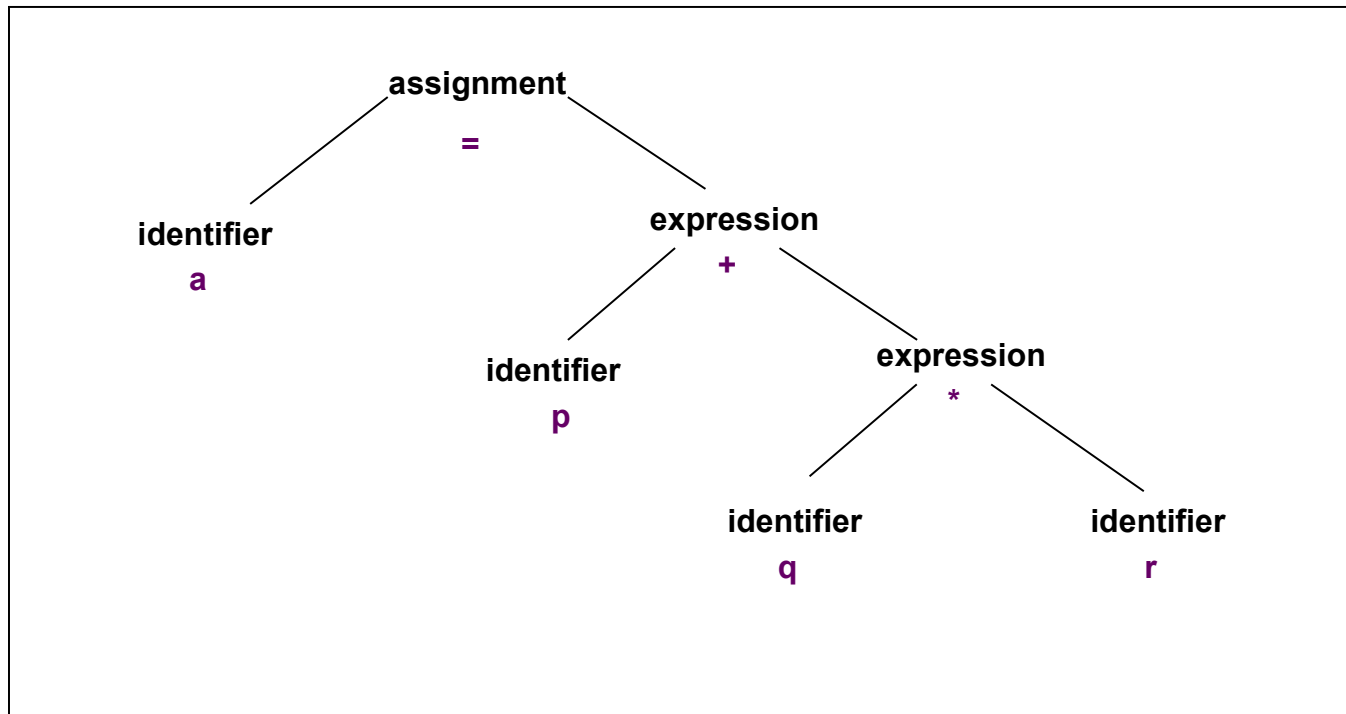
- ▲ Lexical and syntax analysis
- ▲ Parse-tree generation
- ▲ Macro-expansion
- ▲ Expansion of meta-variables

- ◆ **Semantic analysis:**

- ▲ Data-flow and control-flow analysis
- ▲ Type checking
- ▲ Resolve arithmetic and relational operators

Parse tree example

$a = p + q * r$



Behavioral-level optimization

- ◆ Semantic-preserving transformations aiming at simplifying the model
- ◆ Applied to parse-trees or during their generation
- ◆ Taxonomy:
 - ▲ *Data-flow* based transformations
 - ▲ *Control-flow* based transformations

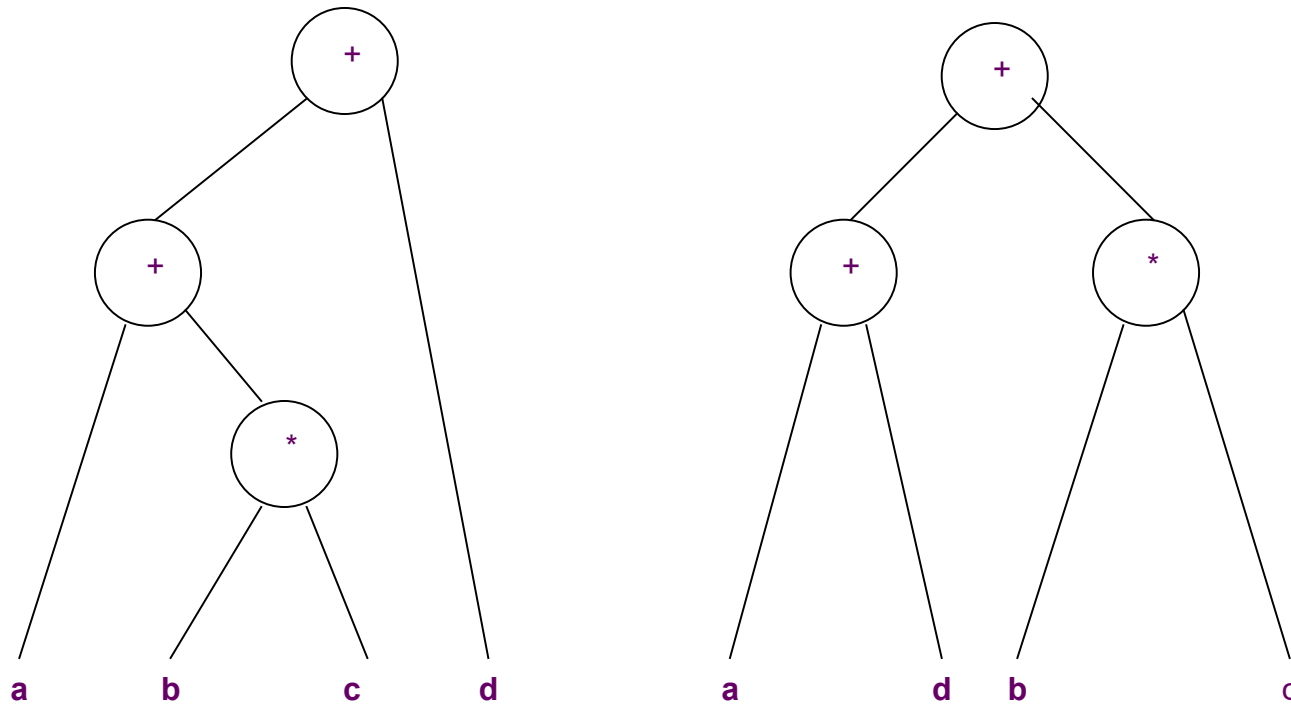
Data-flow based transformations

- ◆ Tree-height reduction
- ◆ Constant and variable propagation
- ◆ Common sub-expression elimination
- ◆ Dead-code elimination
- ◆ Operator-strength reduction
- ◆ Code motion

Tree-height reduction

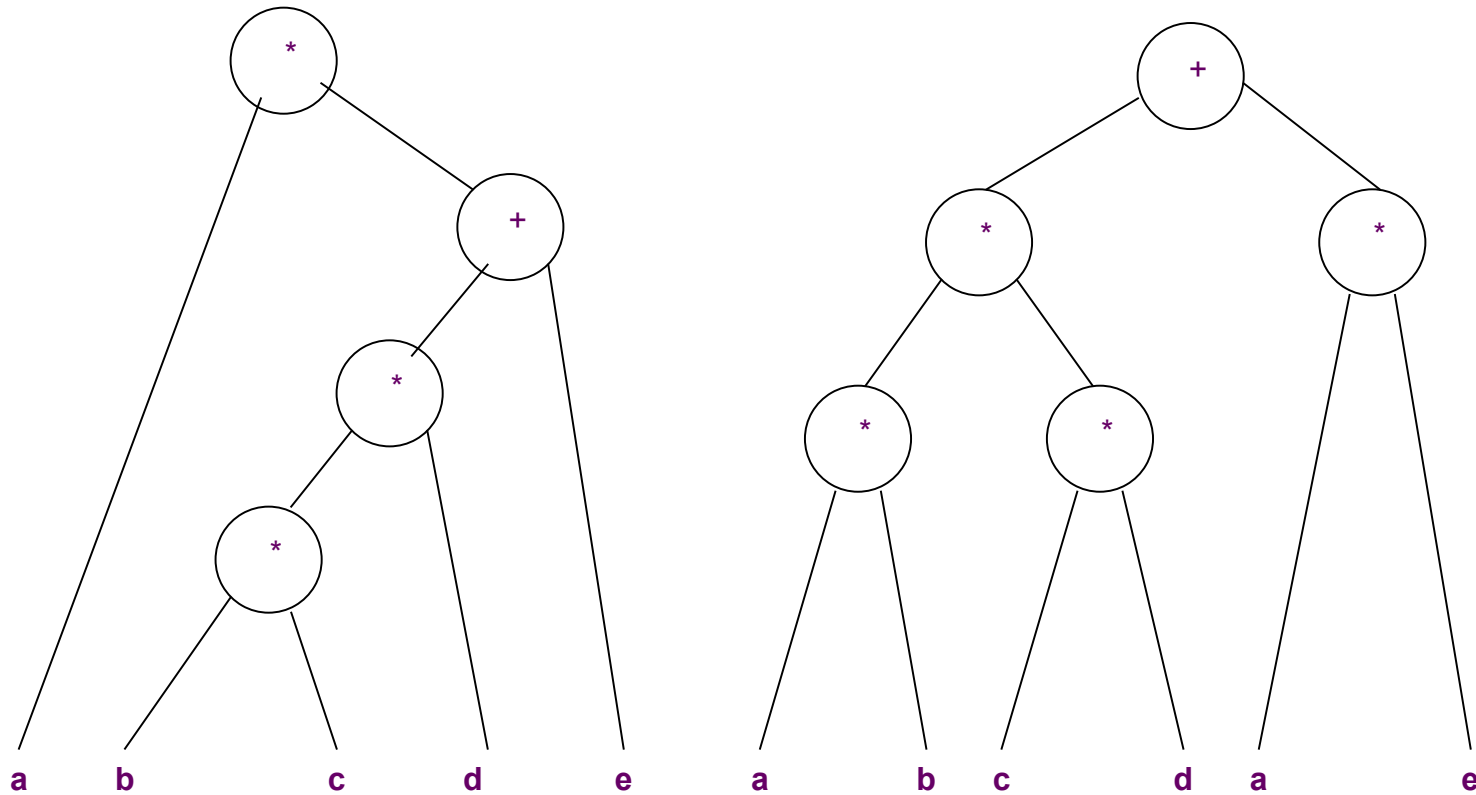
- ◆ Applied to arithmetic expressions
- ◆ Goal:
 - ▲ Split into two-operand expressions to exploit hardware parallelism at best
- ◆ Techniques:
 - ▲ Balance the expression tree
 - ▲ Exploit *commutativity*, *associativity* and *distributivity*

Example of tree-height reduction using commutativity and associativity



$$x = a + b * c + d \rightarrow x = (a + d) + b * c$$

Example of tree-height reduction using distributivity



$$x = a * (b * c * d + e) \rightarrow x = a * b * c * d + a * e;$$

Examples of propagation

◆ Constant propagation

$a = 0; b = a + 1; c = 2 * b;$

$a = 0; b = 1; c = 2;$

◆ Variable propagation:

$a = x; b = a + 1; c = 2 * x;$

$a = x; b = x + 1; c = 2 * x;$

Sub-expression elimination

- ◆ Logic expressions:

- ▲ Performed by logic optimization
- ▲ Kernel-based methods

- ◆ Arithmetic expressions:

- ▲ Search isomorphic patterns in the parse trees
- ▲ Example:

$a = x + y; \quad b = a + 1; \quad c = x + y$

$a = x + y; \quad b = a + 1; \quad c = a;$

Examples of other transformations

- ◆ **Dead-code elimination:**

$a = x; b = x + 1; c = 2 * x;$

$a = x;$ can be removed if not referenced

- ◆ **Operator-strength reduction:**

$a = x^2, b = 3 * x;$

$a = x * x; t = x \ll 1; b = x + t;$

- ◆ **Code motion:**

$\text{for } (i = 1; i < a * b) \{ \}$

$t = a * b; \text{ for } (i = 1; i < t) \{ \}$

Control-flow based transformations

- ◆ **Model expansion**
- ◆ **Conditional expansion**
- ◆ **Loop expansion**
- ◆ **Block-level transformations**

Model expansion

- ◆ Expand subroutine
 - ▲ Flatten hierarchy
 - ▲ Expand scope of other optimization techniques
- ◆ Problematic when model is called more than once
- ◆ Example:

$x = a + b; \quad y = a * b; \quad z = \text{foo}(x, y);$

$\text{foo}(p,q) \{ t=q - p; \text{return}(t); \}$

By expanding **foo**:

$x = a + b; \quad y = a*b; \quad z = y - x;$

Conditional expansion

- ◆ Transform conditional into parallel execution with test at the end
- ◆ Useful when test depends on late signals
- ◆ May preclude hardware sharing
- ◆ Always useful for logic expressions
- ◆ Example:

$y = ab; \text{ if } (a) \{ x = b + d; \} \text{ else } \{ x = bd; \}$

▲ Can be expanded to: $x = a(b + d) + a'bd$

▲ And simplified as: $y = ab; x = y + d(a + b)$

Loop expansion

- ◆ Applicable to loops with data-independent exit conditions
- ◆ Useful to expand scope of other optimization techniques
- ◆ Problematic when loop has many iterations

- ◆ Example:

$x = 0; \text{ for } (i = 1; i < 3; i++) \{ x = x + 1; \}$

- ◆ Expanded to:

$x = 0; x = x + 1; x = x + 2; x = x + 3$

Module2

◆ Objectives

- ▲ Architectural optimization
- ▲ Scheduling, resource sharing, estimation

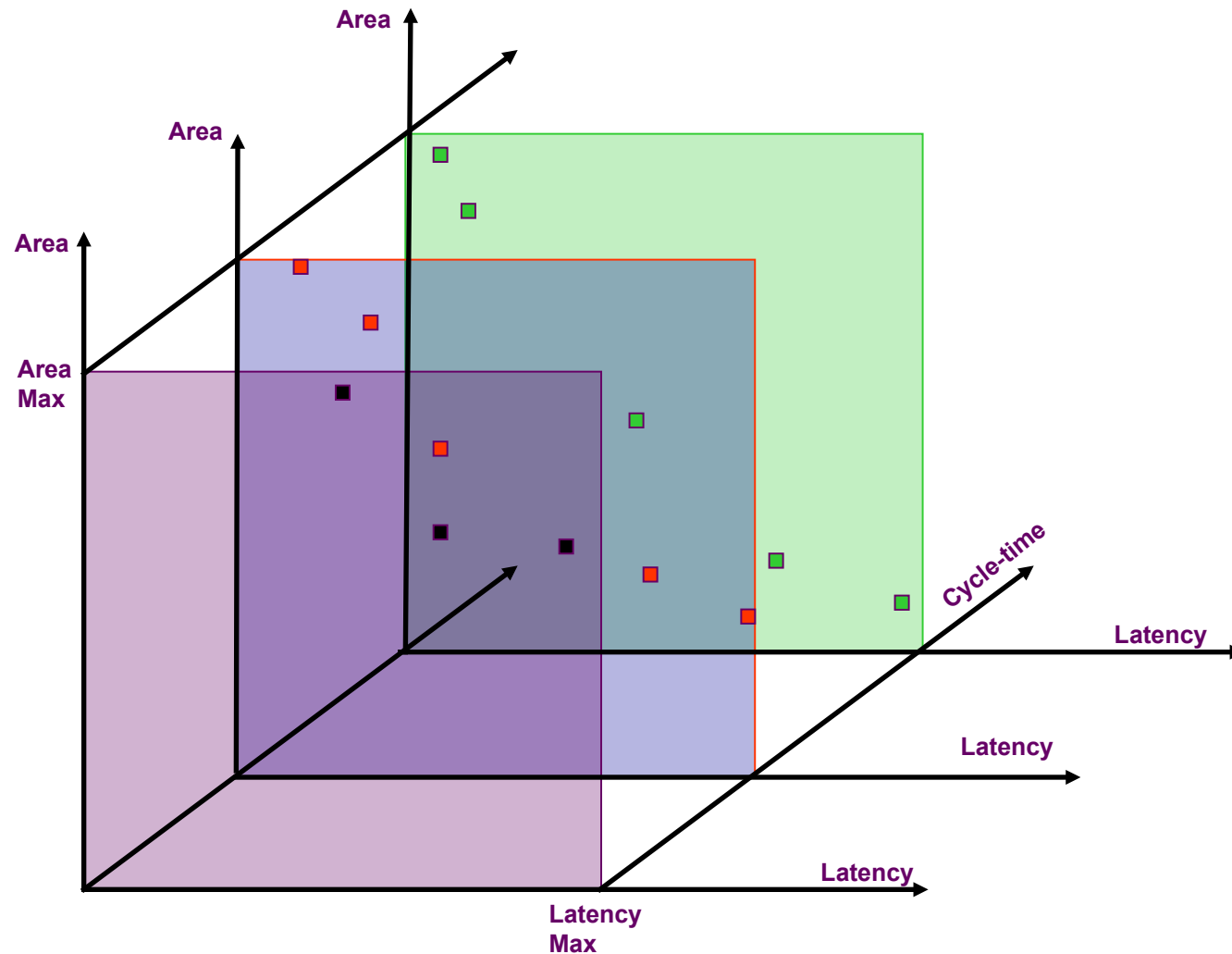
Architectural synthesis and optimization

- ◆ Synthesize macroscopic structure in terms of building-blocks
- ◆ Explore area/performance trade-off:
 - ▲ *maximize performance* implementations subject to *area* constraints
 - ▲ *minimize area* implementations subject to *performance* constraints
- ◆ Determine an optimal implementation
- ◆ Create logic model for data-path and control

Design space and objectives

- ◆ **Design space:**
 - ▲ Set of all feasible implementations
- ◆ **Implementation parameters:**
 - ▲ Area
 - ▲ Performance:
 - ▼ Cycle-time
 - ▼ Latency
 - ▼ Throughput (for pipelined implementations)
 - ▲ Power consumption

Design evaluation space



Hardware modeling

- ◆ **Circuit behavior:**
 - ▲ Sequencing graphs
- ◆ **Building blocks:**
 - ▲ Resources
- ◆ **Constraints:**
 - ▲ Timing and resource usage

Resources

- ◆ **Functional resources:**
 - ▲ Perform operations on data
 - ▲ Example: arithmetic and logic blocks
- ◆ **Storage resources:**
 - ▲ Store data
 - ▲ Example: memory and registers
- ◆ **Interface resources:**
 - ▲ Example: busses and ports

Synthesis in the temporal domain

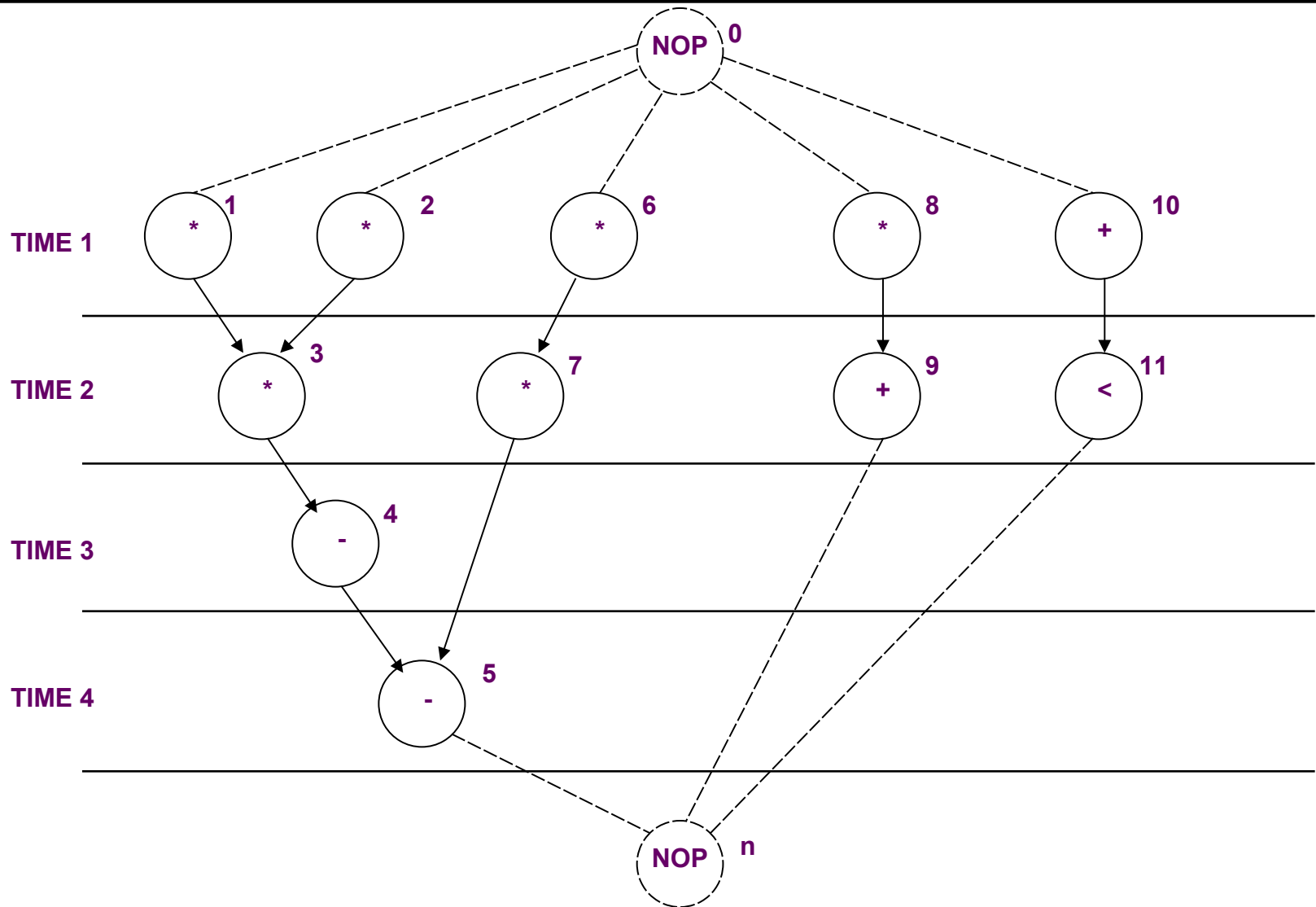
- ◆ *Scheduling:*

- ▲ Associate a **start-time** with each operation
- ▲ Determine **latency** and parallelism of the implementation

- ◆ *Scheduled sequencing graph:*

- ▲ Sequencing graph with start-time annotation

Example



Synthesis in the spatial domain

- ◆ ***Binding:***

- ▲ Associate a resource with each operation with the same type
- ▲ Determine the area of the implementation

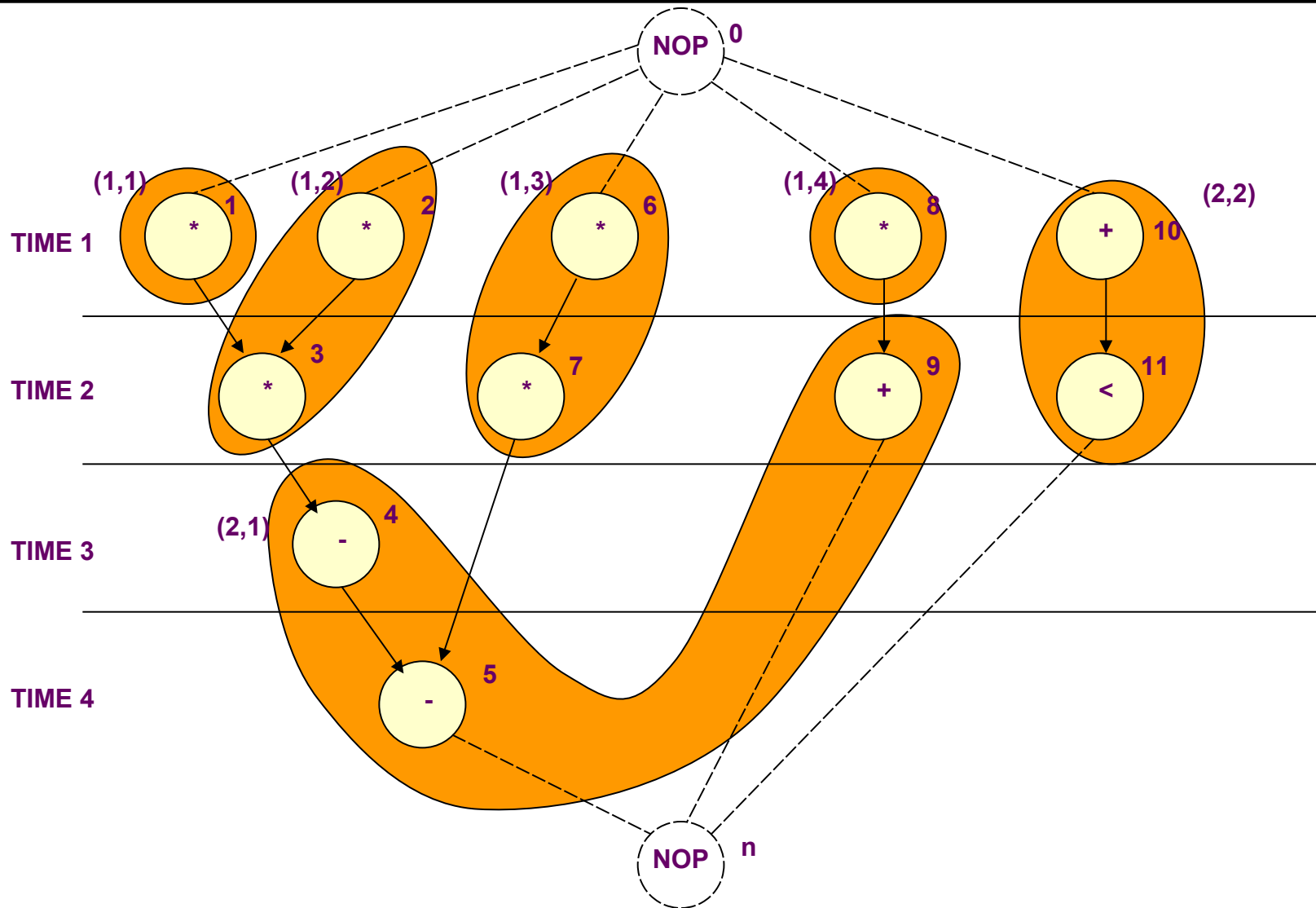
- ◆ ***Sharing:***

- ▲ Bind a resource to more than one operation
- ▲ Operations must not execute concurrently

- ◆ ***Bound sequencing graph:***

- ▲ Sequencing graph with resource annotation

Example



Estimation

- ◆ Resource-dominated circuits.
 - ▲ Area = sum of the area of the resources bound to the operations
 - ▼ Determined by *binding*
 - ▲ Latency = start time of the sink operation (minus start time of the source operation)
 - ▼ Determined by *scheduling*
- ◆ Non resource-dominated circuits
 - ▲ Area also affected by:
 - ▼ Registers, steering logic, wiring and control
 - ▲ Cycle-time also affected by:
 - ▼ Steering logic, wiring and (possibly) control

Approaches to architectural optimization

- ◆ ***Multiple-criteria* optimization problem:**
 - ▲ Area, latency, cycle-time
- ◆ **Determine *Pareto optimal* points:**
 - ▲ Implementations such that no other has all parameters with inferior values
- ◆ **Draw trade-off curves:**
 - ▲ Discontinuous and highly nonlinear

Area-latency trade-off

- ◆ **Rationale:**

- ▲ Cycle-time dictated by system constraints

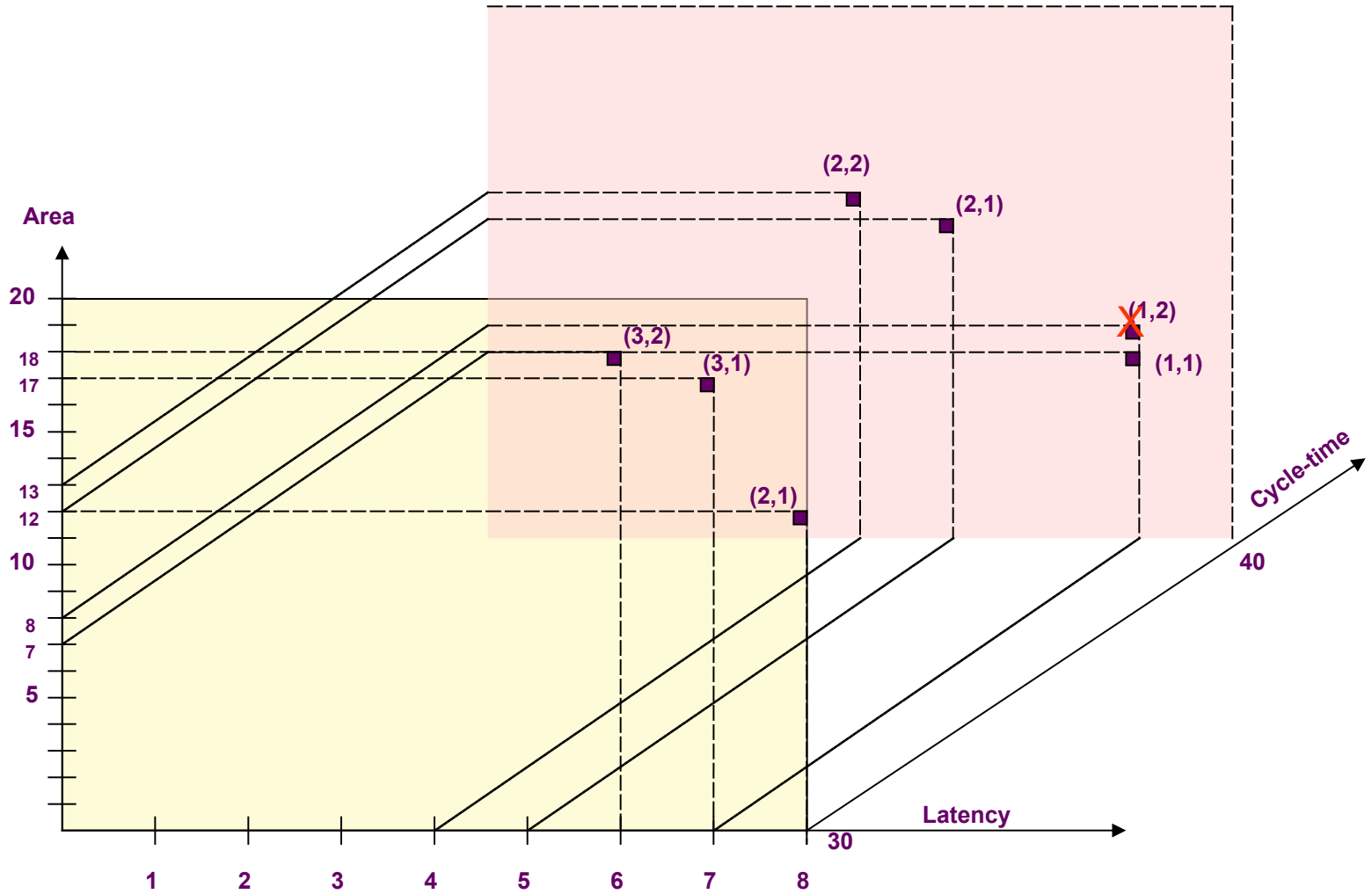
- ◆ **Resource-dominated circuits:**

- ▲ Area is determined by resource usage

- ◆ **Approaches:**

- ▲ *Schedule* for minimum latency under resource usage constraints
 - ▲ *Schedule* for minimum resource usage under latency constraints
 - ▼ for varying cycle-time constraints

Area/latency trade-off



Summary

- ◆ **Behavioral optimization:**
 - ▲ Create abstract models from HDL models
 - ▲ Optimize models without considering implementation parameters
- ◆ **Architectural synthesis and optimization**
 - ▲ Consider resource parameters
 - ▲ Multiple-criteria optimization problem:
 - ▼ area, latency, cycle-time