

EE382V: System-on-a-Chip (SoC) Design

Lecture 17 – Resource Allocation and Binding

*Source: G. De Micheli, Integrated Systems Center, EPFL
“Synthesis and Optimization of Digital Circuits”, McGraw Hill, 2001.*

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu



Lecture 17: Outline

- **Allocation, binding and sharing**
 - Problem formulation
- **Functional unit sharing**
 - Flat graphs
 - Hierarchical graphs
- **Register sharing**
 - Multi-port register files/memories
- **Bus sharing**
- **Extensions**
 - Module selection
 - Datapath and control synthesis

Allocation and Binding

- **Allocation**
 - Number of resources available
- **Binding**
 - Mapping of operations onto resources
- **Sharing**
 - Many-to-one relation
- **Selection**
 - Type to implement each operation

Binding

- **Limiting cases**
 - Dedicated resources
 - One resource per operation
 - No sharing
 - One multi-task resource
 - ALU
 - One resource per type
- **Closely related to scheduling**
- **Optimum binding/sharing**
 - Minimize the resource usage
 - Scheduled sequencing graphs
 - Operation concurrency well defined
 - Consider *operation types* independently
 - Problem decomposition
 - » Perform analysis for each resource type

ILP Formulation of Binding

- **Boolean variable** b_{ir}
 - Operation i bound to resource r

$$\sum_r b_{ir} = 1 \quad \text{for all operations } i$$

- **Boolean variables** x_{il}
 - Operation i scheduled to start at step l

$$\sum_i b_{ir} \sum_{m=l-di+1..l} x_{im} \leq 1 \quad \text{for all steps } l \text{ and resources } r$$

Compatibility and Conflicts

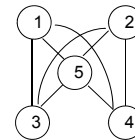
- **Operation compatibility:**

- Same type
- Non concurrent

t1	x=a+b	y=c+d	1	2
t2	s=x+y	t=x-y	3	4
t3	z=a+t		5	

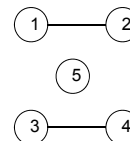
- **Compatibility graph:**
 - Vertices: operations
 - Edges: compatibility relation

Compatibility graph



- **Conflict graph:**
 - Complement of compatibility graph

Conflict graph



Compatibility and Conflicts

- **Compatibility graph clique partitioning**
 - Partition the graph into a minimum number of cliques
 - Find *clique cover number* $k(G_+)$
- **Conflict graph coloring**
 - Color the vertices by a minimum number of colors.
 - Find *chromatic number* $\chi(G_-)$
- **NP-complete problems**
 - Heuristic algorithms

EE382V: SoC Design, Lecture 17

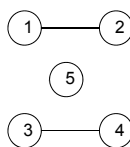
© G. De Micheli

7

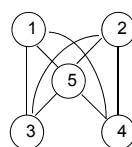
Compatibility and Conflict Example

t1	x=a+b	y=c+d	1	2
t2	s=x+y	t=x-y	3	4
t3	z=a+t		5	

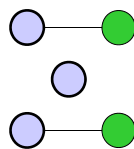
Conflict



Compatibility

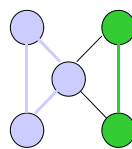


Coloring



ALU1: 1,3,5
ALU2: 2,4

Partitioning



EE382V: SoC Design, Lecture 17

© G. De Micheli

8

Perfect Graphs

- **Comparability graph**

- Graph $G(V, E)$ has an orientation $G(V, F)$ with the transitive property

$$(v_i, v_j) \in F \text{ and } (v_j, v_k) \in F \rightarrow (v_i, v_k) \in F$$

- **Interval graph**

- Vertices correspond to *intervals*
- Edges correspond to interval intersection
- Subset of *chordal* graphs
 - Every loop with more than three edges has a chord (edge that is not part of the cycle but connects two nodes in the loop)

Data Flow Graphs (DFGs)

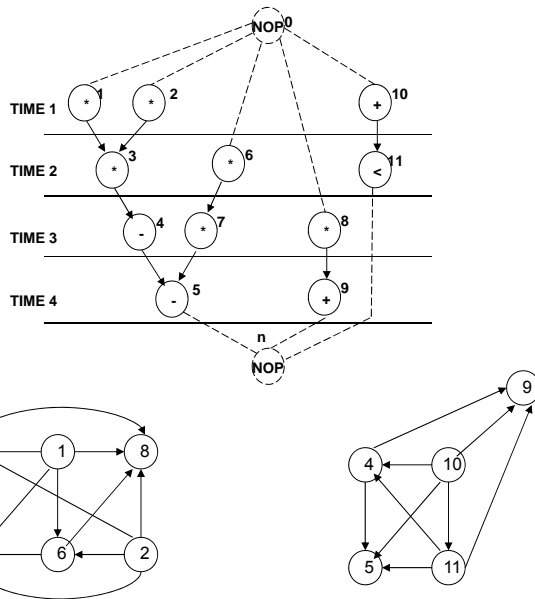
- **The compatibility/conflict graphs have special properties**

- Compatibility
 - Comparability graph
- Conflict
 - Interval graph

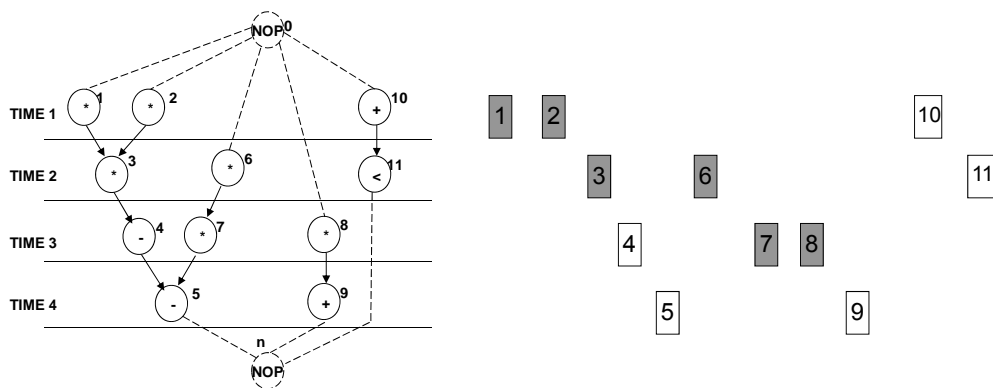
- **Polynomial time solutions**

- Golumbic's algorithm for partitioning of compatibility graphs
- Left-edge algorithm for coloring of interval graphs

DFG Example: Comparability Graphs



DFG Example: Interval Graphs



Left-Edge Algorithm

- **Input:**
 - Set of intervals with *left* and *right edge*
 - A set of *colors* (initially one color)
- **Rationale**
 - Sort intervals in a *list* by *left edge*
 - Assign non overlapping intervals to first color using the list
 - When possible intervals are exhausted, increase color counter and repeat

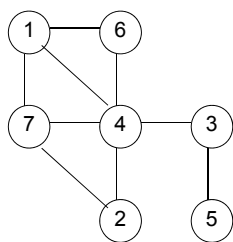
Left-Edge Algorithm

```

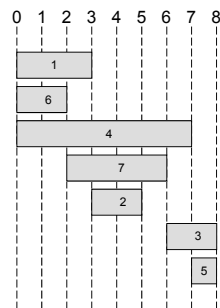
LEFT_EDGE(I)
{
  Sort elements of I in a list L in ascending order of  $l_i$ ;
   $c = 0$ ;
  while (some interval has not been colored) do {
     $S = \emptyset$ ;
     $r = 0$ ;
    while ( exists  $s \in L$  such that  $l_s > r$ ) do {
       $s =$  First element in the list L with  $l_s > r$ ;
       $S = S \cup \{s\}$ ;
       $r = r_s$ ;
      Delete  $s$  from L;
    }
     $c = c + 1$ ;
    Label elements of S with color  $c$ ;
  }
}

```

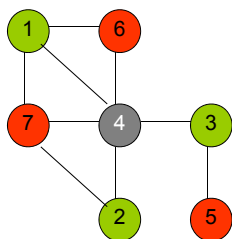
Left-Edge Example



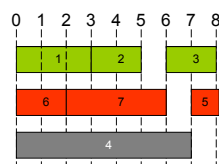
Conflict graph



Intervals



Colored conflict graph



Coloring

Lecture 17: Outline

- ✓ Allocation, binding and sharing
 - ✓ Problem formulation
- Functional unit sharing
 - ✓ Flat graphs
 - Hierarchical graphs
- Register sharing
 - Multi-port register files/memories
- Bus sharing
- Extensions
 - Module selection
 - Datapath and control synthesis

Hierarchical Sequencing Graphs (CDFGs)

- **Hierarchical conflict/compatibility graphs**
 - Easy to compute
 - Prevent sharing across hierarchy
- **Flatten hierarchy**
 - Bigger graphs
 - Destroy nice properties

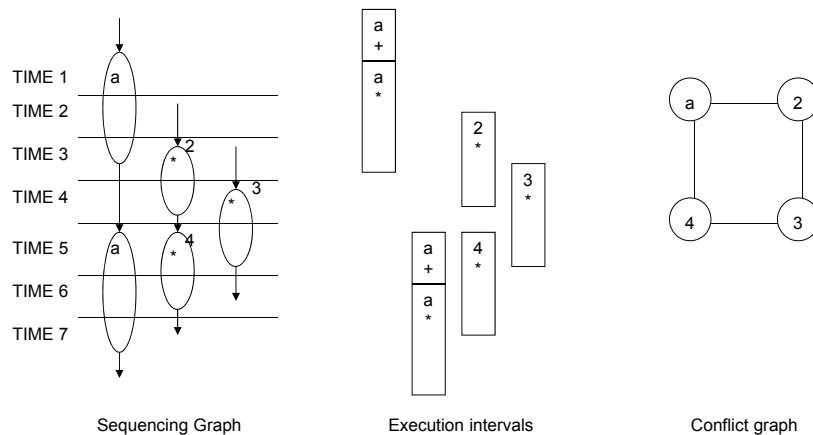
EE382V: SoC Design, Lecture 17

© G. De Micheli

17

Hierarchical Graph Example

- **Calls of sub-models**



- **Not perfect graphs any more**
 - Intractable, use of heuristics

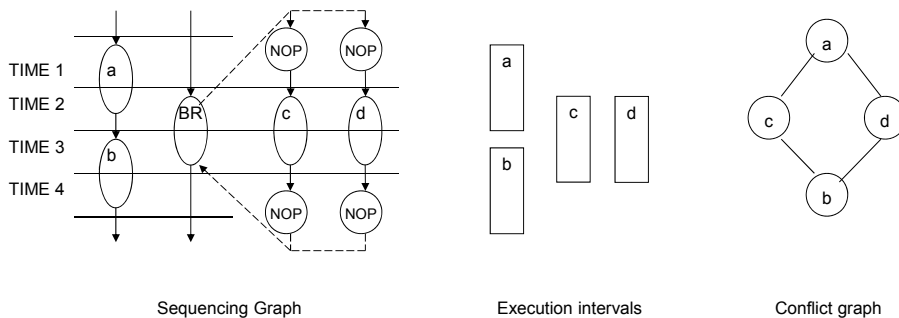
EE382V: SoC Design, Lecture 17

© G. De Micheli

18

Hierarchical Graph Example

- **Branching constructs**



- **Not perfect graphs any more**

- Intractable, use of heuristics

Lecture 17: Outline

- ✓ **Allocation, binding and sharing**
 - ✓ Problem formulation
- ✓ **Functional unit sharing**
 - ✓ Flat graphs
 - ✓ Hierarchical graphs
- **Register sharing**
 - Multi-port register files/memories
- **Bus sharing**
- **Extensions**
 - Module selection
 - Datapath and control synthesis

Storage Elements

- **Registers**
 - Hold data across cycles
 - Data: value of a variable
 - Variable lifetime in scheduled graph
 - Can be re-used (shared) across variables
- **Memory blocks, register files**
 - Limited number of read/write ports

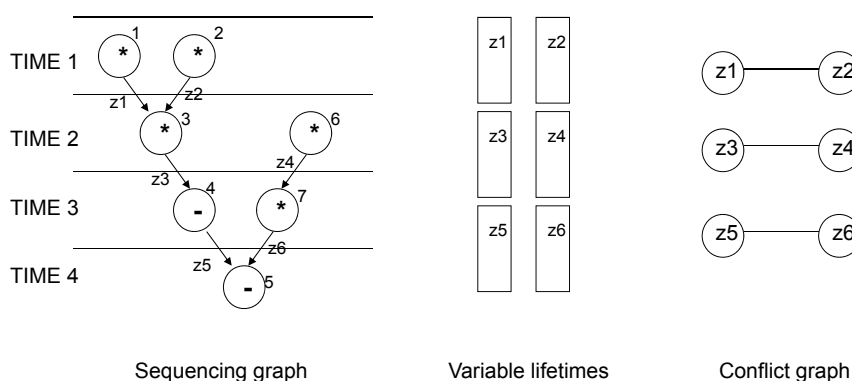
Register Binding Problem

- **Given a schedule**
 - *Lifetime intervals* for variables
 - Lifetime overlaps
- **Conflict graph (interval graph)**
 - Vertices \leftrightarrow variables
 - Edges \leftrightarrow overlaps
 - Interval graph
- **Compatibility graph (comparability graph)**
 - Complement of conflict graph

Register Sharing

- **Given**
 - Variable lifetime conflict graph
- **Find**
 - Minimum number of registers storing all the variables
- **Key point**
 - Interval graph
 - Left-edge algorithm (polynomial-time complexity)

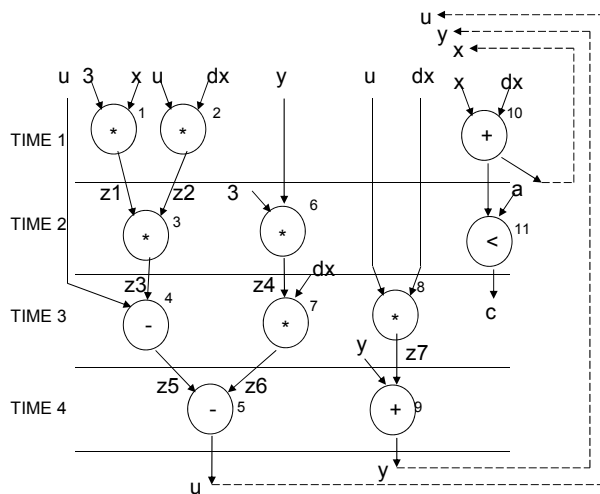
Register Sharing Example



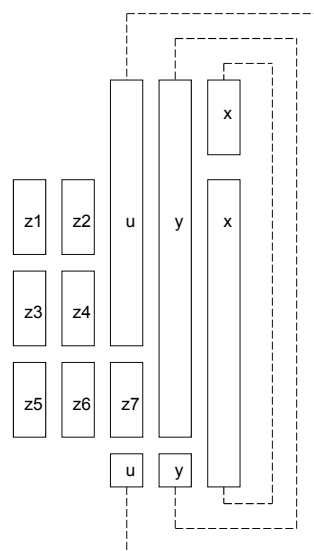
Register Sharing in Loops

- **Iterative conflicts through loop-carried dependencies**
 - Preserve values across iterations
 - Circular-arc conflict graph
 - Coloring is intractable
 - **Hierarchical graphs**
 - General conflict graphs
 - Coloring is intractable
- **Heuristic algorithms**

Loop Example

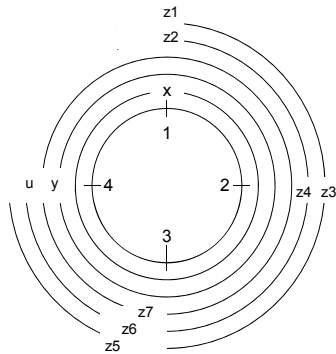


Sequencing graph

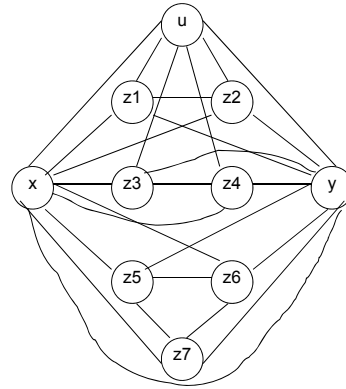


Variable lifetimes

Loop Variable Lifetimes



Variable lifetimes



Circular-arc conflict graph

Multi-Port Memory Binding

- Find *minimum number of ports* to access the required number of variables
- Variables use the same port
 - Port compatibility/conflict
 - Similar to resource binding
- Variables can use any port
 - Decision variable x_{il} is TRUE when variable i is accessed in step l
 - Optimum: $\max_{1 \leq l \leq \lambda+1} (\sum_{i=1..nvar} x_{il})$

Dual Multiport-Memory Binding Problem

- Find *max number of variables to be stored through a fixed number of ports a*
 - Boolean variables $\{ b_i, i = 1, 2, \dots, n_{var} \}$:
 - Variable i with $b_i=1$ will be stored in register file
 - maximize $\sum_{i=1..nvar} b_i$ such that

$$\sum_{i=1..nvar} b_i x_{il} \leq a \quad l = 1, 2, \dots, \lambda + 1$$

Multi-Port Memory Binding Example

Time – step 1 : $r_3 = r_1 + r_2 ; r_{12} = r_1$

*Time – step 2 : $r_5 = r_3 + r_4 ; r_7 = r_3 * r_6 ; r_{13} = r_3$*

Time – step 3 : $r_8 = r_3 + r_5 ; r_9 = r_1 + r_7 ; r_{11} = r_{10} / r_5$

Time – step 4 : $r_{14} = r_{11} \& r_8 ; r_{15} = r_{12} | r_9$

Time – step 5 : $r_1 = r_{11} ; r_2 = r_{15}$

$\max \sum_{i=1}^{15} b_i$ such that

$$b_1 + b_2 + b_3 + b_{12} \leq a$$

$$b_3 + b_4 + b_5 + b_6 + b_7 + b_{13} \leq a$$

$$b_1 + b_3 + b_5 + b_7 + b_8 + b_9 + b_{10} + b_{11} \leq a$$

$$b_8 + b_9 + b_{11} + b_{12} + b_{14} + b_{15} \leq a$$

$$b_1 + b_2 + b_{14} + b_{15} \leq a$$

- **One port $a = 1$:**
 - $\{ b_2, b_4, b_8 \}$ non-zero
 - 3 variables stored: v_2, v_4, v_8
- **Two ports $a = 2$:**
 - 6 variables stored: $v_2, v_4, v_5, v_{10}, v_{12}, v_{14}$
- **Three ports $a = 3$:**
 - 9 variables stored: $v_1, v_2, v_4, v_6, v_8, v_{10}, v_{12}, v_{13}$

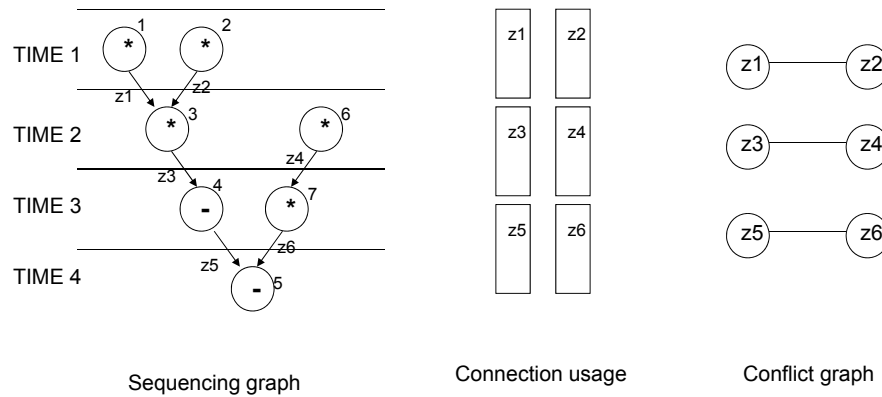
Lecture 17: Outline

- ✓ Allocation, binding and sharing
 - ✓ Problem formulation
- ✓ Functional unit sharing
 - ✓ Flat graphs
 - ✓ Hierarchical graphs
- ✓ Register sharing
 - ✓ Multi-port register files/memories
- Bus sharing
- Extensions
 - Module selection
 - Datapath and control synthesis

Bus Sharing and Binding

- Find the *minimum number of busses* to accommodate all data transfer
- Find the *maximum number of data transfers* for a fixed number of busses
- Similar to memory binding problem
- ILP formulation or heuristic algorithms

Bus Sharing Example



- **One bus:**
 - 3 variables can be transferred
- **Two busses:**
 - All variables can be transferred

Lecture 17: Outline

- ✓ **Allocation, binding and sharing**
 - ✓ Problem formulation
- ✓ **Functional unit sharing**
 - ✓ Flat graphs
 - ✓ Hierarchical graphs
- ✓ **Register sharing**
 - ✓ Multi-port register files/memories
- ✓ **Bus sharing**
- **Extensions**
 - Module selection
 - Datapath and control synthesis

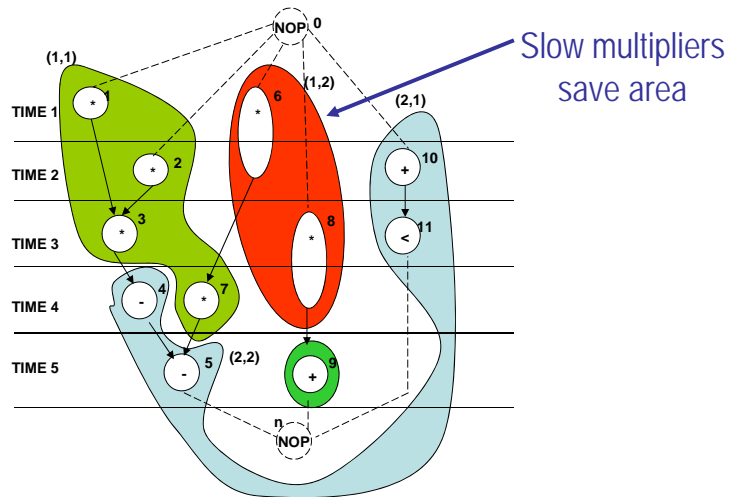
Module Selection Problem

- **Extension of resource sharing**
 - Library of resources
 - More than one resource per type
- **Example**
 - Ripple-carry adder
 - Carry look-ahead adder
- **Resource modeling**
 - Resource *subtypes* with
 - (area, delay) parameters

Module Selection Solutions

- **ILP formulation**
 - Decision variables
 - Select resource sub-type
 - Determine (*area*, *delay*)
- **Heuristic algorithm**
 - Determine *minimum latency* with fastest resource subtypes
 - Recover area by using slower resources on non-critical paths

Module Selection Example



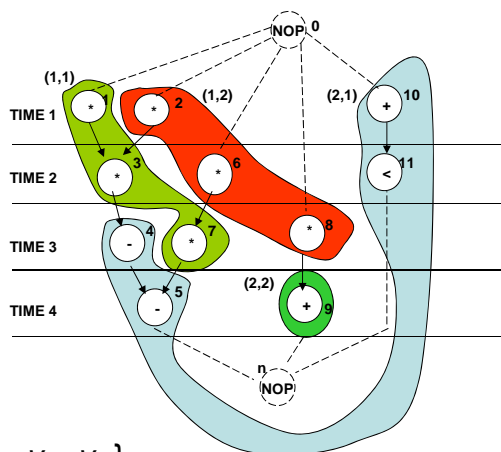
- **Multipliers with**
 - (Area, delay) = (5,1) and (2,2)
- **Latency bound of 5**

EE382V: SoC Design, Lecture 17

© G. De Micheli

37

Module Selection Example 2



- **Latency bound of 4**
 - Fast multipliers for $\{ v_1, v_2, v_3 \}$
 - Slower multiplier can be used elsewhere
 - Less sharing
- **Minimum-latency design uses fast multipliers only**
 - Impossible to use slow multipliers

EE382V: SoC Design, Lecture 17

© G. De Micheli

38

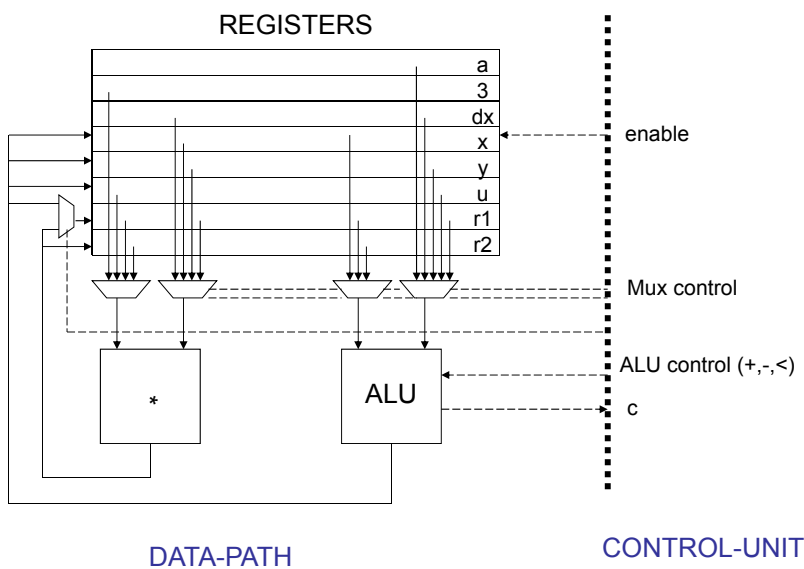
Lecture 17: Outline

- ✓ Allocation, binding and sharing
 - ✓ Problem formulation
- ✓ Functional unit sharing
 - ✓ Flat graphs
 - ✓ Hierarchical graphs
- ✓ Register sharing
 - ✓ Multi-port register files/memories
- ✓ Bus sharing
- Extensions
 - ✓ Module selection
 - Datapath and control synthesis

Data Path Synthesis

- Applied after resource binding
- Connectivity synthesis
 - Connection of resources to *multiplexers, busses* and *registers*
 - Control unit interface
 - I/O ports
- Physical data path synthesis
 - Specific techniques for regular datapath design
 - Regularity extraction

Data Path Synthesis Example



EE382V: SoC Design, Lecture 17

© G. De Micheli

41

Control Synthesis

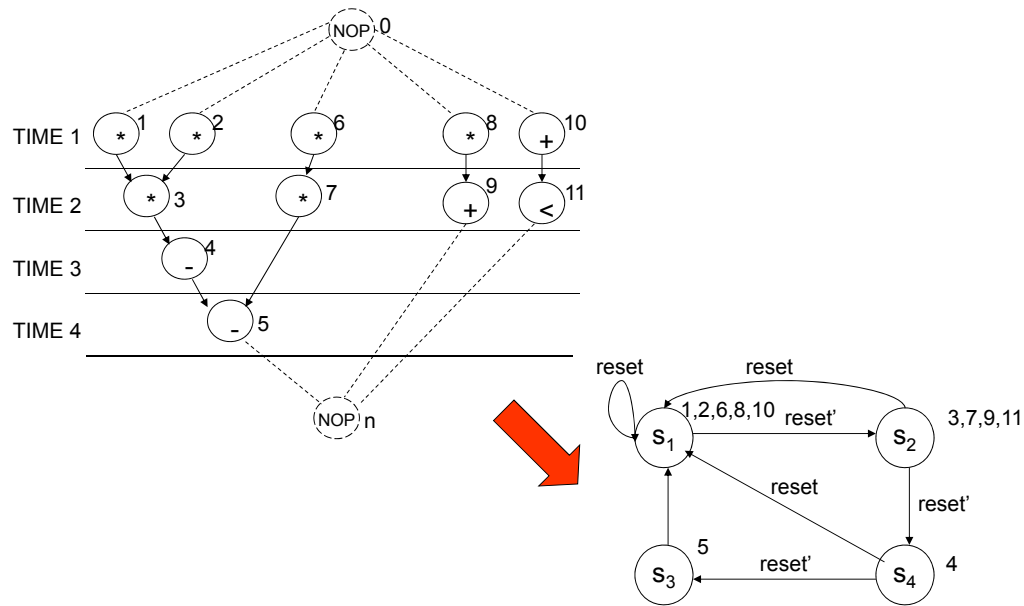
- **Synthesis of the control unit**
 - Sequencer
- **Logic model**
 - Synchronous FSM
- **Physical implementation**
 - Hard-wired or distributed FSM
 - Microcode

EE382V: SoC Design, Lecture 17

© G. De Micheli

42

Control Synthesis Example



EE382V: SoC Design, Lecture 17

© G. De Micheli

43

Lecture 17: Summary

- **Resource sharing is reducible to vertex coloring or to clique covering**
 - Simple for flat graphs
 - Intractable, but still easy in practice, for other graphs
 - Resource sharing has several extensions
 - Module selection
- **Data path design and control synthesis are conceptually simple but still important steps**
 - Generated data path is an interconnection of blocks
 - Control is one or more finite-state machines

EE382V: SoC Design, Lecture 17

© G. De Micheli

44