

ECE382M.20: System-on-Chip (SoC) Design

Lecture 6 – High-Level Synthesis

Sources: Jacob Abraham, Rajesh Gupta (UC San Diego)

*D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner,
“Embedded System Design: Modeling, Synthesis, Verification,”
Chapter 6: Hardware Synthesis, Springer, 2009.*

Andreas Gerstlauer

Electrical and Computer Engineering

The University of Texas at Austin

gerstl@ece.utexas.edu



The University of Texas at Austin

Chandra Department of Electrical
and Computer Engineering

Cockrell School of Engineering

Lecture 6: Outline

- **Introduction**
 - High-level synthesis (HLS)
- **Essential issues**
 - Behavioral specification languages
 - Target architectures
 - Intermediate representation
 - Scheduling/allocation/binding
 - Control generation
- **High-level synthesis flow**
 - Synthesis example
 - Formalization & automation

High Level Synthesis (HLS)

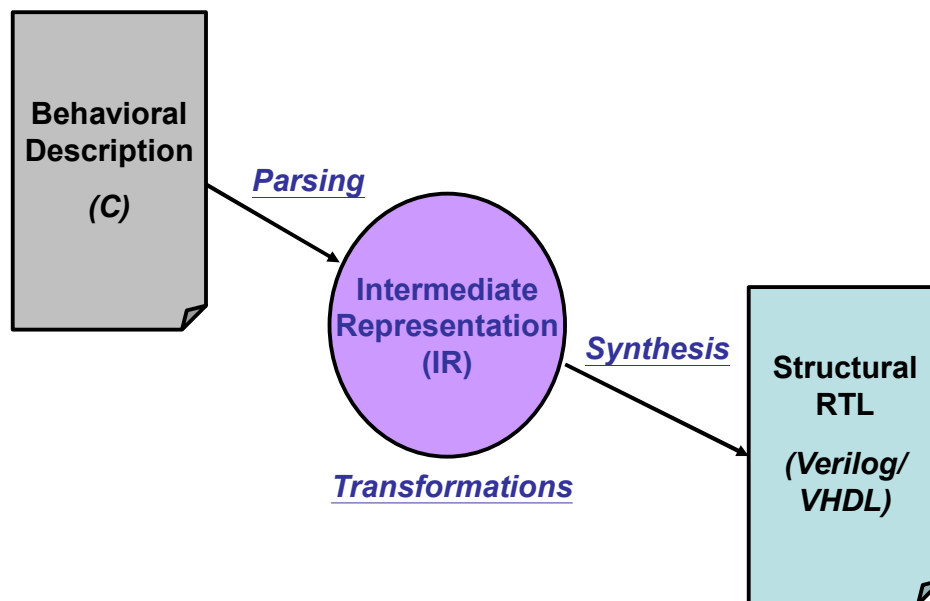
- **Convert a high-level description of a design to a RTL netlist**
 - Input:
 - High-level languages (e.g., C)
 - Behavioral hardware description languages (e.g., VHDL)
 - State diagrams / logic networks
 - Tools:
 - Parser
 - Library of modules
 - Constraints:
 - Area constraints (e.g., # modules of a certain type)
 - Delay constraints (e.g., set of operations should finish in λ clock cycles)
 - Output:
 - Operation scheduling (time) and binding (resource)
 - Control generation and detailed interconnections

ECE382M.20: SoC Design, Lecture 6

© Jacob Abraham, UT Austin

3

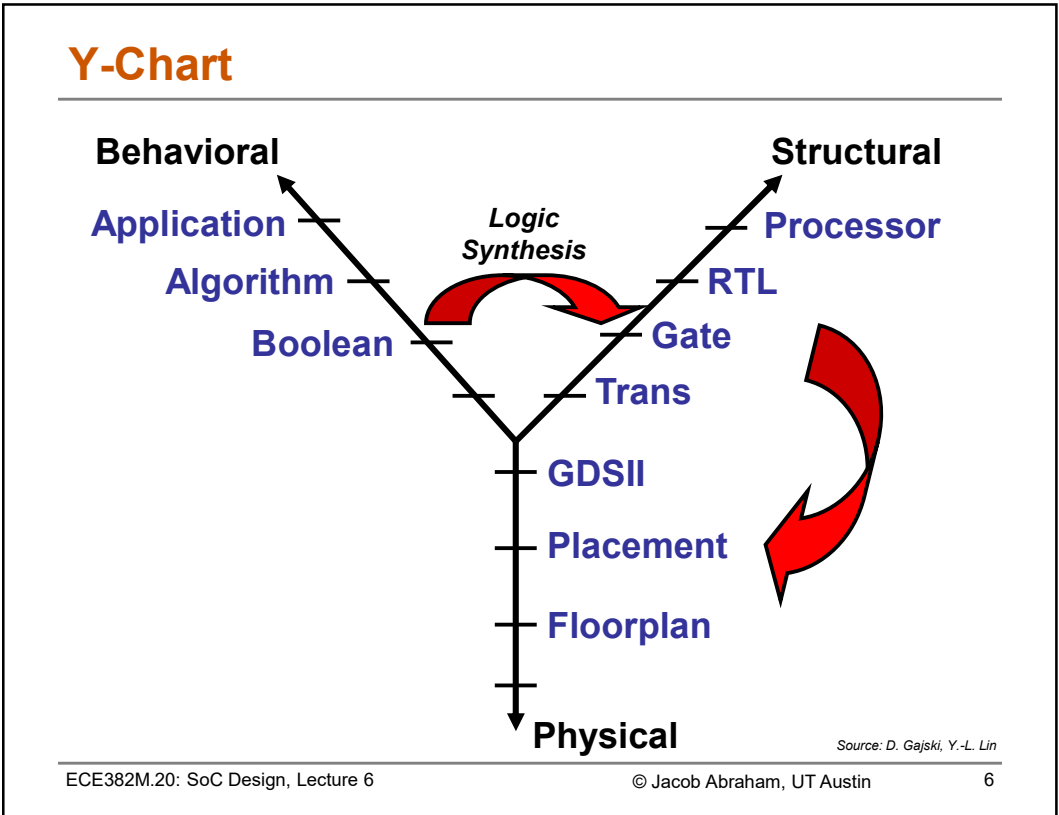
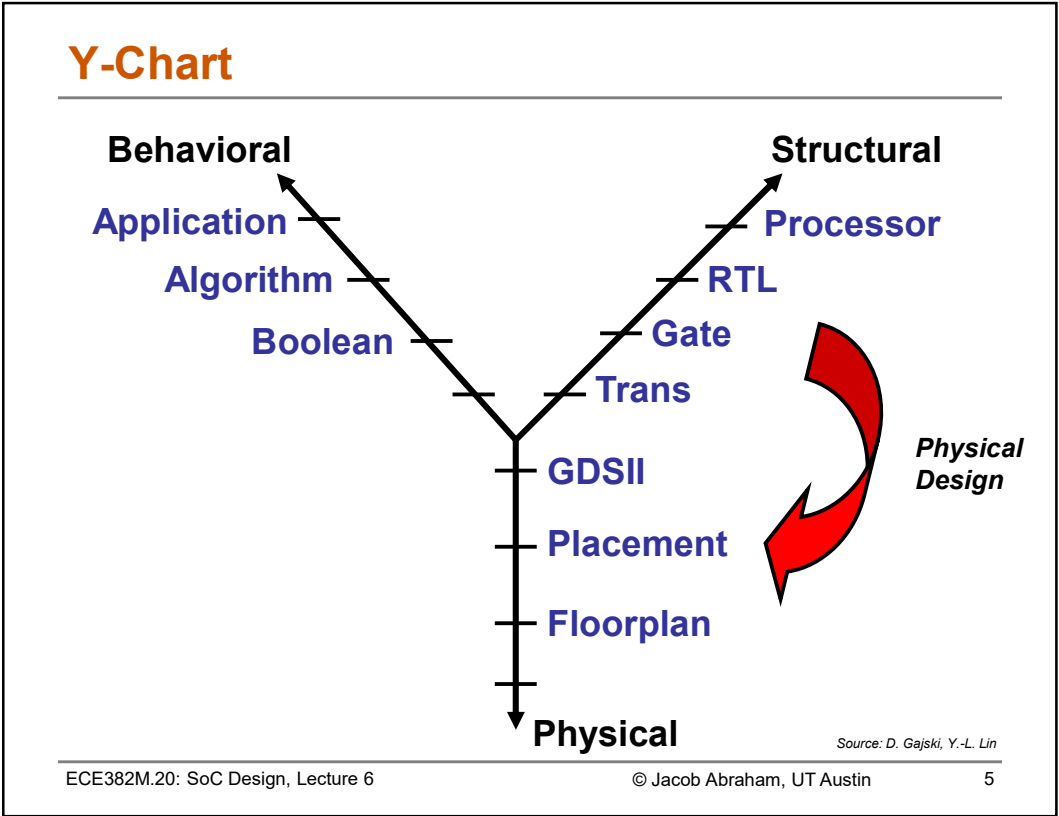
High Level Synthesis



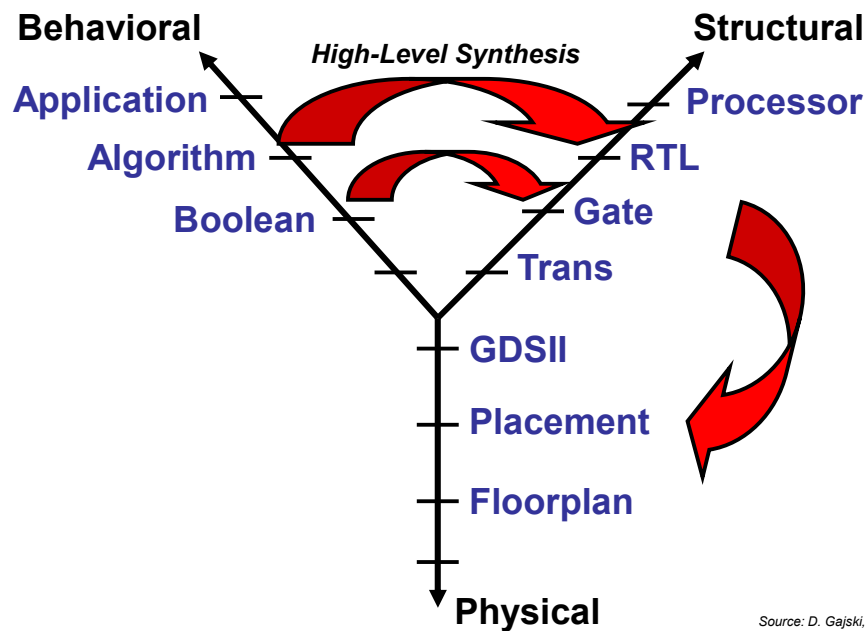
ECE382M.20: SoC Design, Lecture 6

© Jacob Abraham, UT Austin

4



Y-Chart



Source: D. Gajski, Y.-L. Lin

ECE382M.20: SoC Design, Lecture 6

© Jacob Abraham, UT Austin

7

Lecture 6: Outline

✓ Introduction

✓ High-level synthesis

• Essential issues

- Behavioral specification languages
- Target architectures
- Intermediate representation
- Scheduling/allocation/binding
- Control generation

• High-level synthesis flow

- Synthesis example
- Formalization & automation

ECE382M.20: SoC Design, Lecture 6

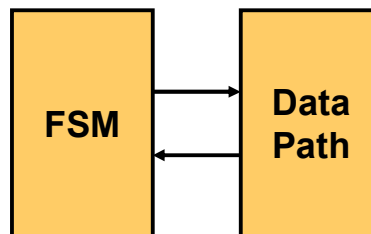
© 2026 A. Gerstlauer

8

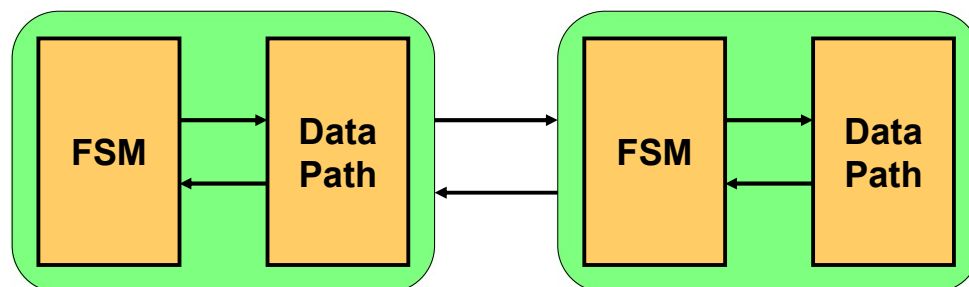
Behavioral Specification Languages

- **First HLS approaches (90's)**
 - Popular HDL
 - Verilog, VHDL
 - Synthesis-oriented HDLs
 - UDL/I
- **Recent resurgence (00's)**
 - Popular legacy programming languages
 - C/C++
 - Add hardware-specific constructs to existing languages
 - SystemC

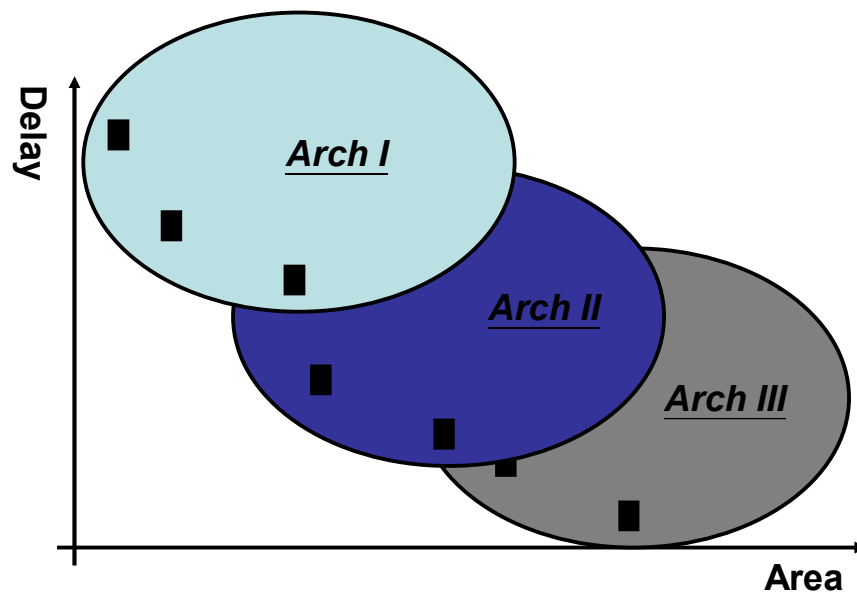
Target Architecture



Finite-State Machine with Data Path



Design Space Exploration



ECE382M.20: SoC Design, Lecture 6

© Jacob Abraham, UT Austin

11

Design Space and Quality Measures

- **Design space**
 - Set of all feasible implementations
- **Quality Measures**
 - Performance
 - Cycle-time
 - Latency
 - Throughput
 - Area cost
 - Power Consumption
 - Testability
 - Reusability

ECE382M.20: SoC Design, Lecture 6

© Jacob Abraham, UT Austin

12

Intermediate Representation

The CFG diagram shows a flow starting from an entry point, branching to Basic Block 1 and Basic Block 2. Basic Block 1 flows into a pink triangle (branch), which then flows into Basic Block 3. Basic Block 2 flows into another pink triangle (join), which then flows into a final pink triangle (exit). The DFG diagram shows a dashed box containing three nodes: two multiplication nodes (*) and one addition node (+). Arrows indicate data flow from inputs into the multiplication nodes, and from the multiplication nodes into the addition node.

Control Flow Graph (CFG)

Data Flow Graph (DFG)

➤ Control/Data Flow Graph (CDFG)

ECE382M.20: SoC Design, Lecture 6

© Jacob Abraham, UT Austin

13

Data Flow Graph (DFG)

- Directed acyclic graph (DAG)
 - Data dependencies, no control/conditionals, no other order

Original code

```
x = a + b;
y = a * c;
z = x + d;
x = y - d;
x = x + c;
```

Single assignment form

```
x1 <= a + b;
y <= a * c;
z <= x1 + d;
x2 <= y - d;
x3 <= x2 + c;
```

The DFG diagram shows a directed acyclic graph with nodes representing operations. Inputs a and b flow into an addition node (+), which outputs x1. Inputs a and c flow into a multiplication node (*), which outputs y. Inputs x1 and d flow into an addition node (+), which outputs z. Inputs y and d flow into a subtraction node (-), which outputs x2. Inputs x2 and c flow into an addition node (+), which outputs x3. The graph is labeled DFG.

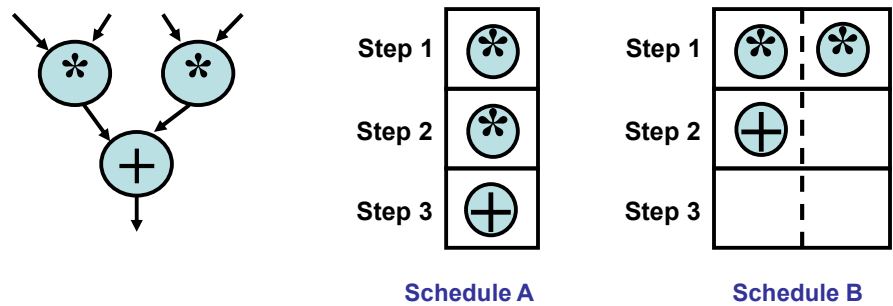
ECE382M.20: SoC Design, Lecture 6

© Jacob Abraham, UT Austin

14

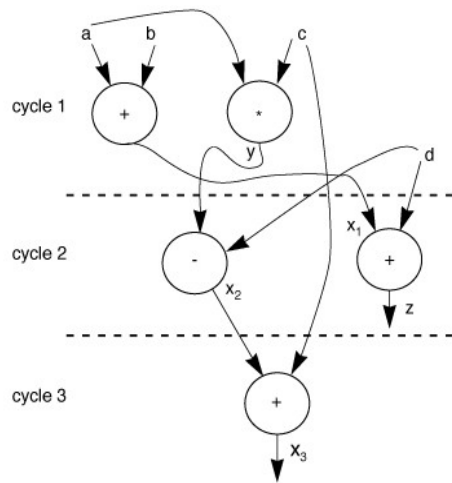
Scheduling

- **Assign operations to clock cycles**
 - Single basic block / data flow graph (DFG) at a time
 - Blocks concatenated according to control flow graph (CFG)
 - Advanced scheduling techniques go beyond single blocks

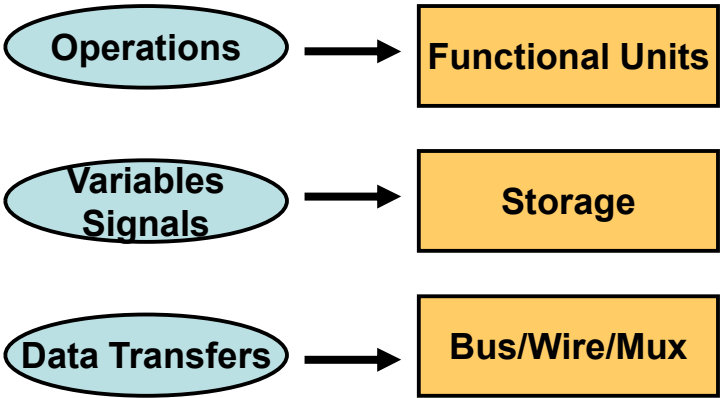


Scheduling Example

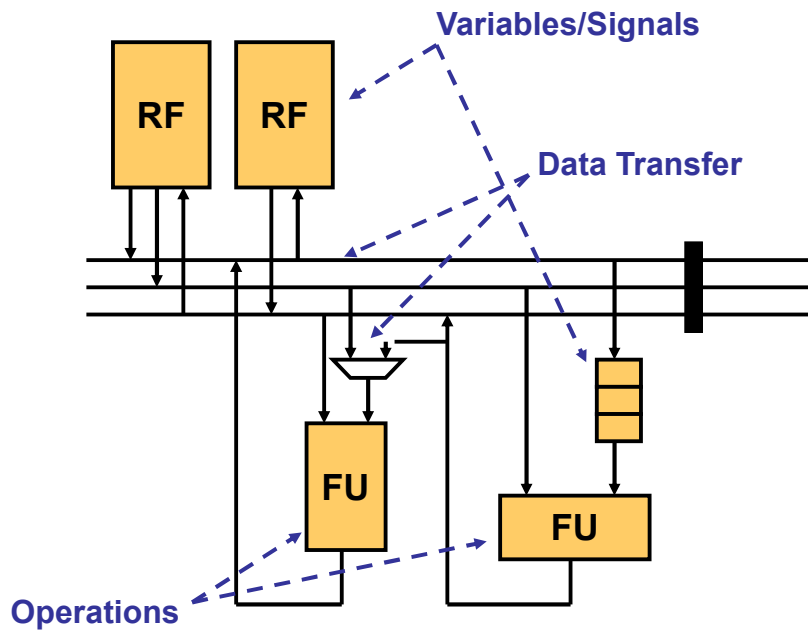
- **One feasible schedule for last DFG**



Allocation/Binding

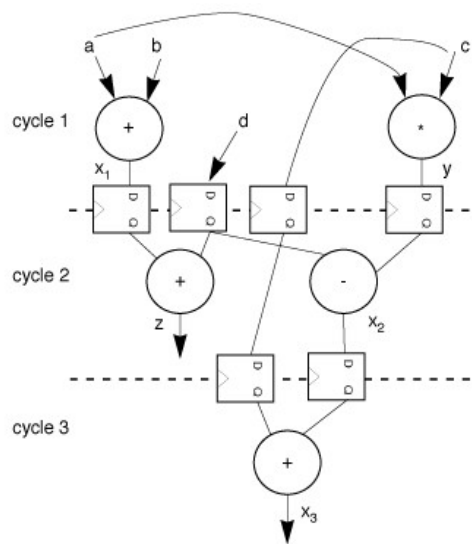


Datapath



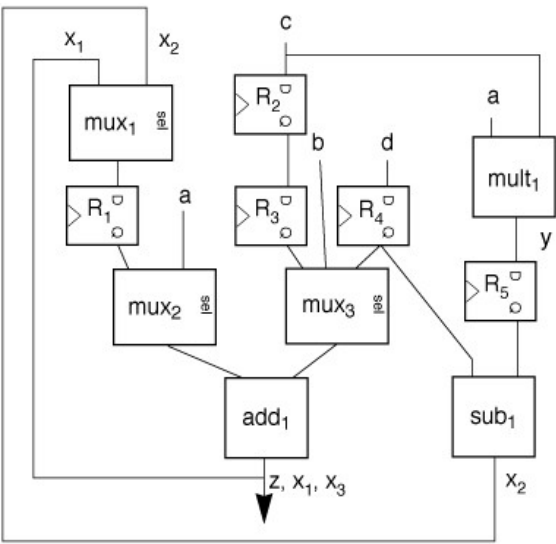
Binding Values to Registers

- Registers fall on clock cycle boundaries

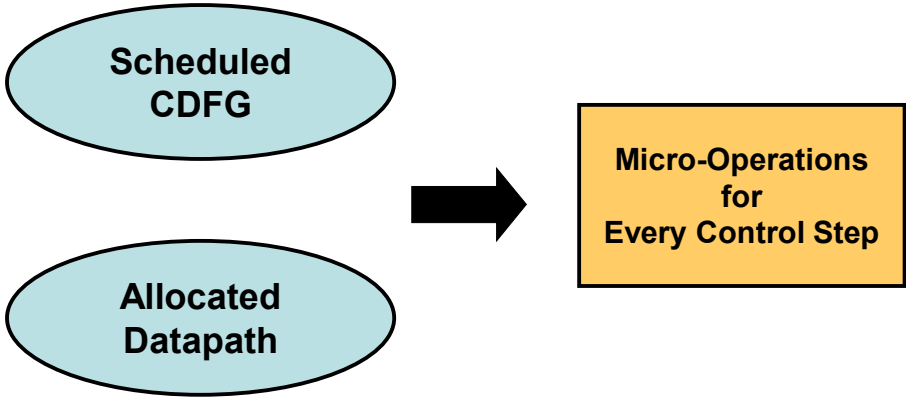


Binding Operations to Functional Units

- Muxes allow sharing
 - Functional units (FUs) for several operations
 - Registers for several variables

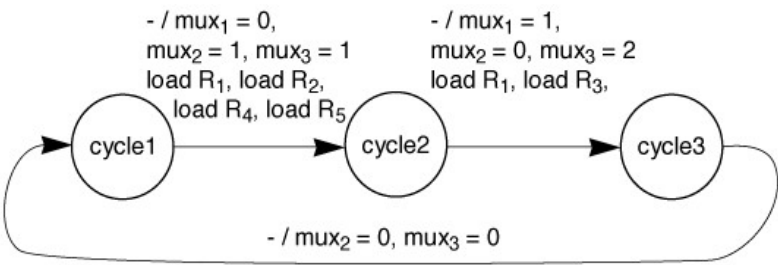


Controller Generation



Building the Sequencer

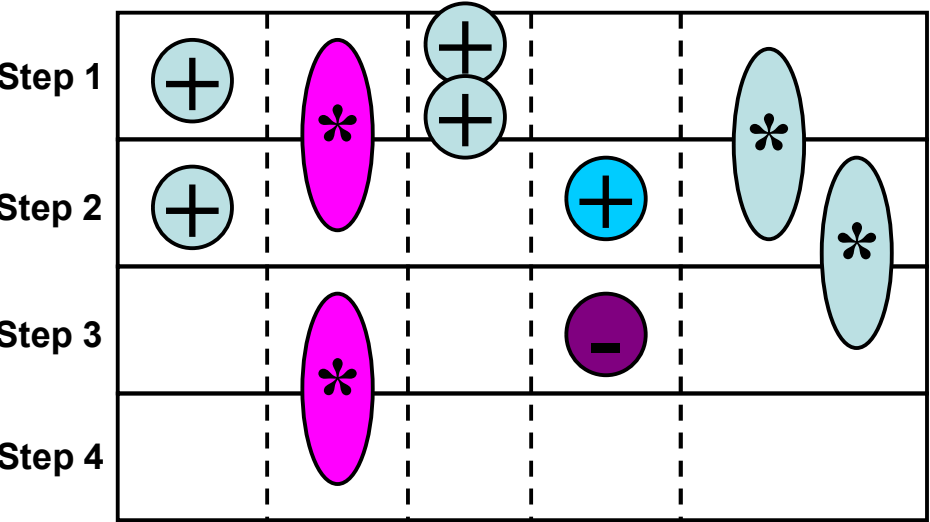
- **Finite state machine (FSM)**
 - Driving datapath control signals
 - Multiple states/cycles even without conditionals



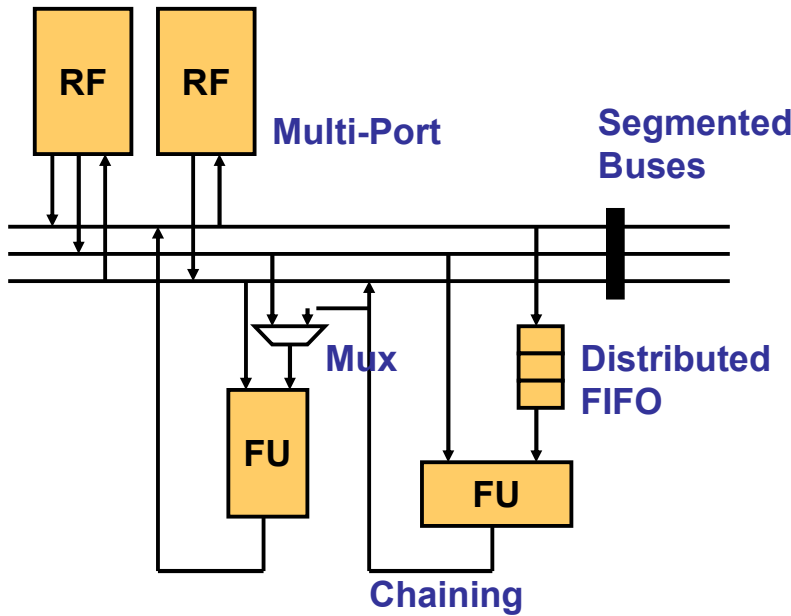
Hardware Variations

- **Functional Units**
 - Pipelined
 - Multi-cycle
 - Chained
 - Multi-function
- **Storage**
 - Register, register file
 - Single-/multi-ported RAM, ROM
 - FIFO, Scratchpad
- **Interconnect**
 - Bus-based
 - Mux-based
 - Protocol-based

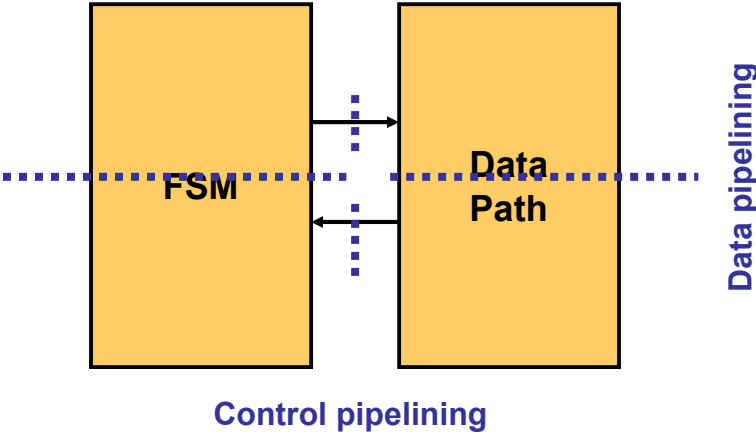
Functional Unit Variations



Storage/Interconnect Variations



Architectural Pipelining



Lecture 6: Outline

✓ Introduction

- ✓ High-level synthesis (HLS)

✓ Essential issues

- ✓ Behavioral specification languages
- ✓ Target architectures
- ✓ Intermediate representation
- ✓ Scheduling/allocation/binding
- ✓ Control generation

• High-level synthesis flow

- Synthesis example
- Formalization

ECE382M.20: SoC Design, Lecture 6

© 2026 A. Gerstlauer

27

Example 1: C Code for Ones Counter

• Programming language semantics

- Sequential execution,
- Coding style to minimize coding

• HW design

- Parallel execution,
- Communication through signals

```

01: int OnesCounter(int Data) {
02:     int Ocount = 0;
03:     int Temp, Mask = 1;
04:     while (Data > 0) {
05:         Temp = Data & Mask;
06:         Ocount = Data + Temp;
07:         Data >>= 1;
08:     }
09:     return Ocount;
10: }

```

Function-based C code

```

01: while(1) {
02:     while (Start == 0);
03:     Done = 0;
04:     Data = Input;
05:     Ocount = 0;
06:     Mask = 1;
07:     while (Data>0) {
08:         Temp = Data & Mask;
09:         Ocount = Ocount + Temp;
10:         Data >>= 1;
11:     }
12:     Output = Ocount;
13:     Done = 1;
14: }

```

RTL-based C code

ECE382M.20: SoC Design, Lecture 6

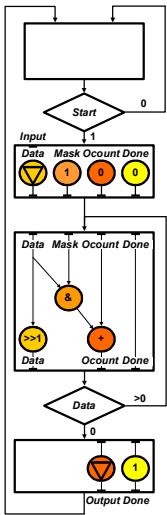
© 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner

28

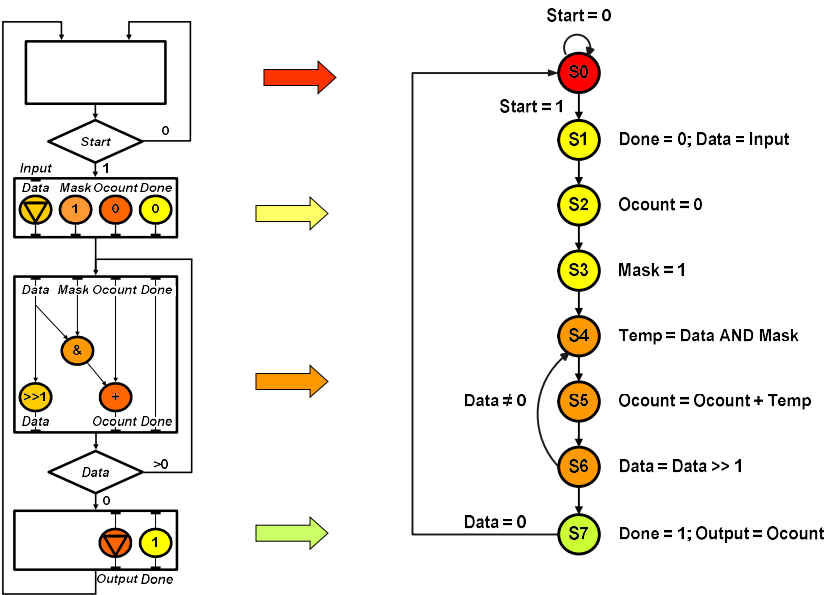
CDFG for Ones Counter

• **Control/Data Flow Graph (CDFG)**

- Resembles programming language
 - Loops, ifs, basic blocks (BBs)
- Explicit dependencies
 - Control dependencies between BBs
 - Data dependencies inside BBs
- Missing dependencies between BBs

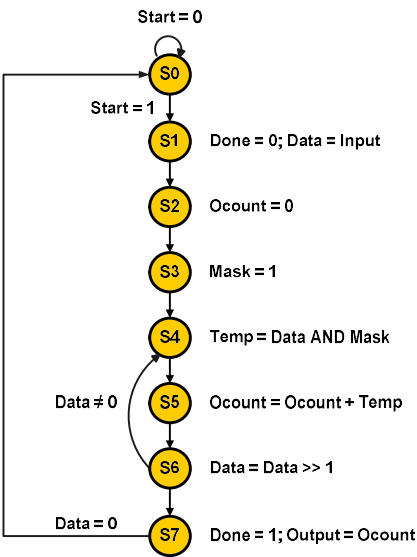


CDFG and FSMD for Ones Counter

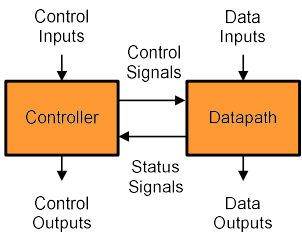


FSMD for Ones Counter

- **FSMD more detailed than CDFG**
 - States may represent clock cycles
 - Conditionals and statements executed concurrently
 - All statement in each state executed concurrently
 - Control signal and variable assignments executed concurrently
- **FSMD includes scheduling**
- **FSMD doesn't specify binding or connectivity**

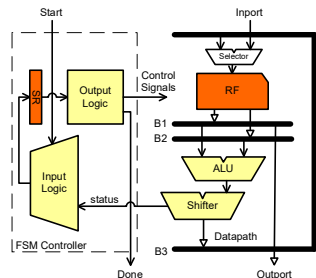


RTL Specification for Ones Counter



- **RTL**
 - FSM control
 - Datapath netlist

Next state logic table				
Present State	Inputs: Start	Data = 0	Next State	Output: Done
S0	0	X	S0	X
S0	1	X	S1	X
S1	X	X	S2	0
S2	X	X	S3	0
S3	X	X	S4	0
S4	X	X	S5	0
S5	X	X	S6	0
S6	X	0	S4	0
S6	X	1	S7	0
S7	X	X	S0	1



Output logic table							
State	RF Read Port A	RF Read Port B	ALU	Shifter	RF selector	RF Write	Output
S0	X	X	X	X	X	X	Z
S1	X	X	X	X	Inport	RF[0]	Z
S2	RF[2]	RF[2]	subtract	pass	B3	RF[2]	Z
S3	RF[2]	X	increment	pass	B3	RF[1]	Z
S4	RF[0]	RF[1]	AND	pass	B3	RF[3]	Z
S5	RF[2]	RF[3]	add	pass	B3	RF[2]	Z
S6	RF[0]	X	pass	shift right	B3	RF[0]	Z
S7	RF[2]	X	X	X	X	disable	enable

RF[0] = Data, RF[1] = Mask, RF[2] = Ocount, RF[3] = Temp

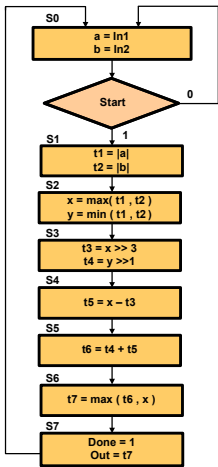
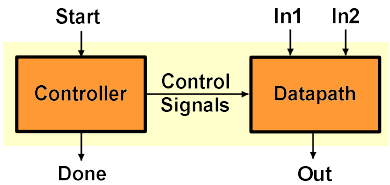
HDL description of Ones Counter

- **HDL description**
 - Same as RTL description
 - Several levels of abstraction
 - Variable binding to storage
 - Operation binding to FUs
 - Transfer binding to connections
- **Netlist must be synthesized**
 - Logic synthesis

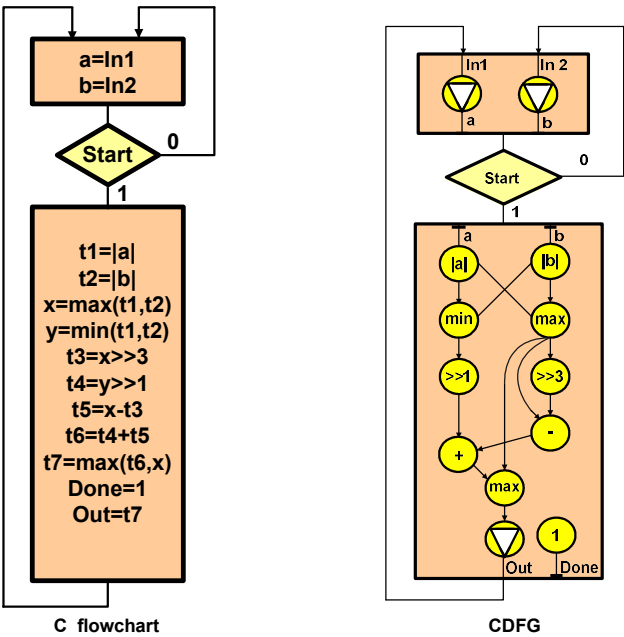
```
01: // ...
02: always@(posedge clk)
03: begin : output_logic
04:     case (state)
05:         // ...
06:         S4: begin
07:             B1 = RF[0];
08:             B2 = RF[1];
09:             B3 = alu(B1, B2, 1_and);
10:             RF[3] = B3;
11:             next_state = S5;
12:         end
13:         // ...
14:         S7: begin
15:             B1 = RF[2];
16:             Outport <= B1;
17:             done <= 1;
18:             next_state = S0;
19:         end
20:     endcase
21: end
22: endmodule
```

Example 2: Square-Root Algorithm (SRA)

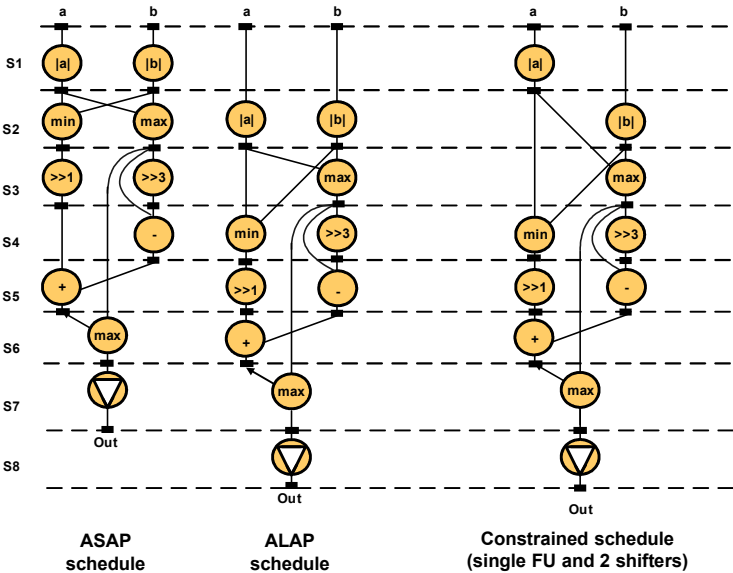
- **SQR = max ((0.875x + 0.5y), x)**
 - $x = \max (|a|, |b|)$
 - $y = \min (|a|, |b|)$



C and CDFG for SRA Algorithm

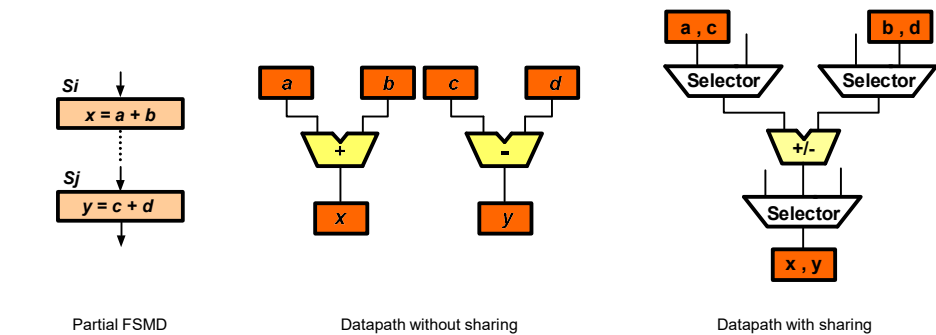


SRA Scheduling



Resource Sharing

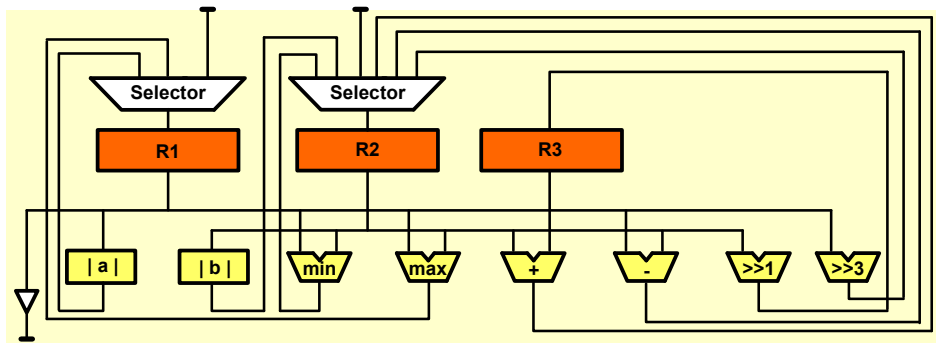
- **Register sharing**
 - Variables with non-overlapping lifetime sharing the same register
- **Functional unit sharing**
 - Non-overlapping (sequential) operations sharing the same unit
- **Interconnect (wire & port) sharing**
 - Non-overlapping transfers sharing busses
 - Non-overlapping register accesses sharing register files



SRA Datapath with Shared Registers

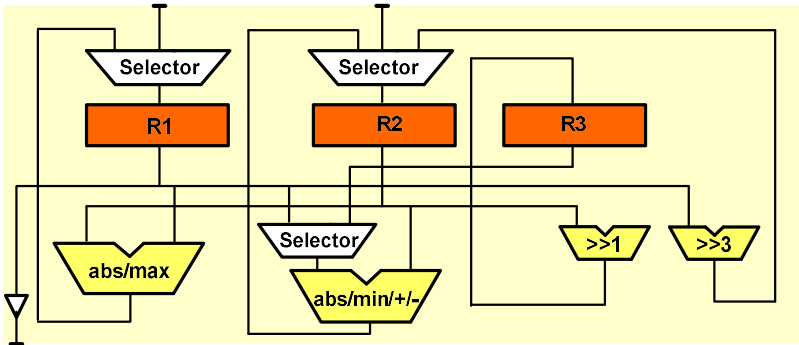
- **Variables combined into registers**
- **One functional unit for each operation**

R1 = [a, t1, x, t7]
R2 = [b, t2, y, t3, t5, t6]
R3 = [t4]



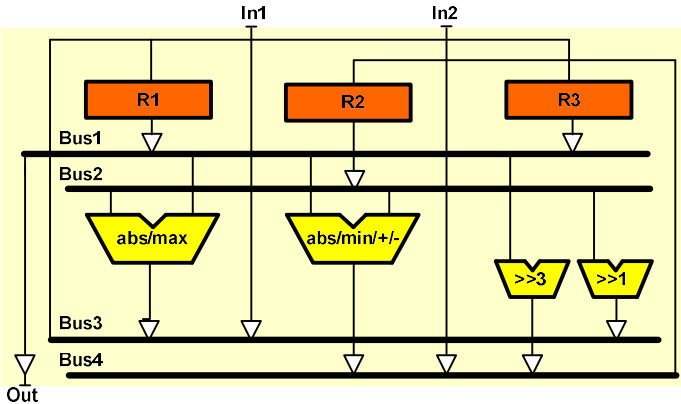
Datapath with Shared Registers and FUs

- Variables combined into registers
- Operations combined into functional units



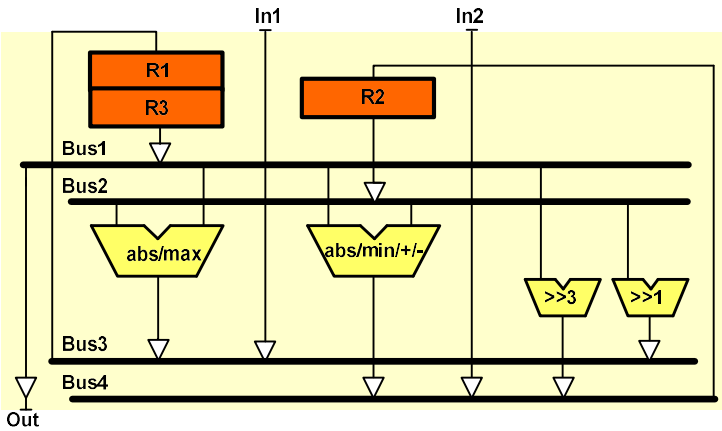
Datapath with Shared Registers, FUs, Buses

- Minimal SRA architecture
 - 3 registers
 - 4 (2) functional units
 - 4 buses



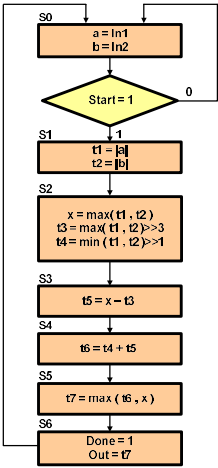
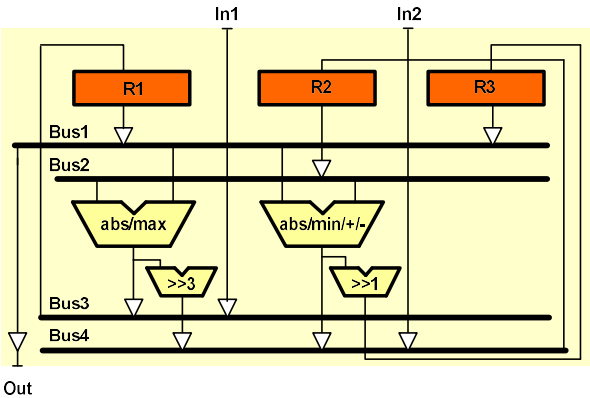
Datapath with Shared RF

- RF minimize connectivity cost by sharing ports



Datapath with Chaining

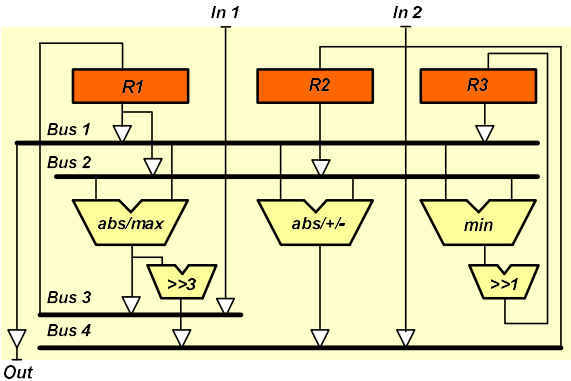
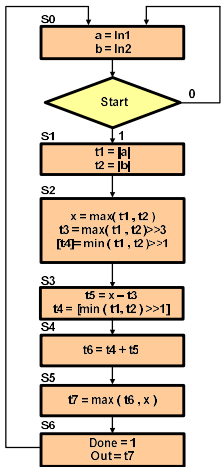
- Chaining connects two or more FUs
- Allows execution of two or more operation in a single clock cycle
- Improves performance at no cost



Datapath with Chaining and Multi-Cycling

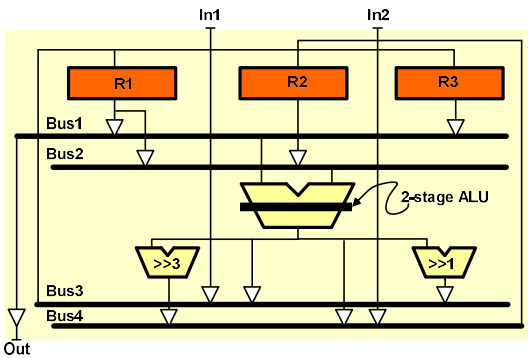
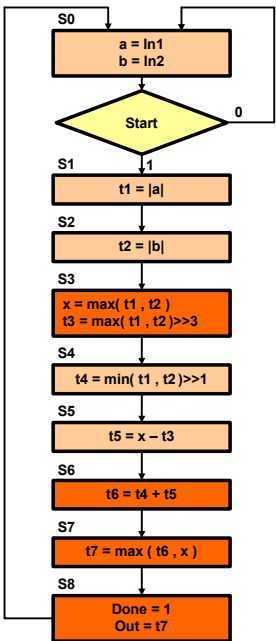
• Multi-cycling

- Operations that take more than one cycle
- Allows use of slower FUs
- Allows faster clock-cycle

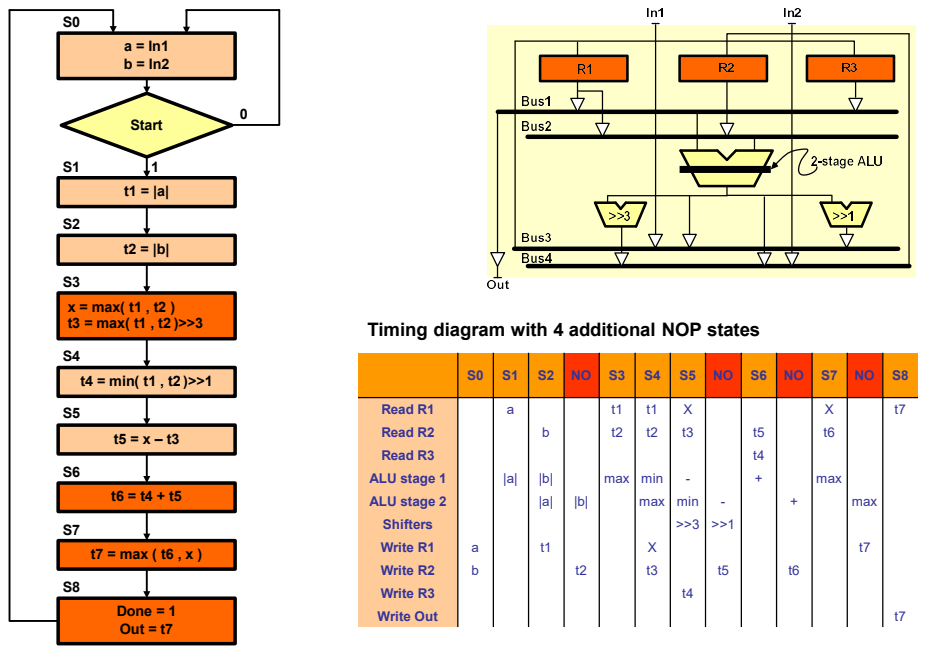


Functional Unit Pipelining (1)

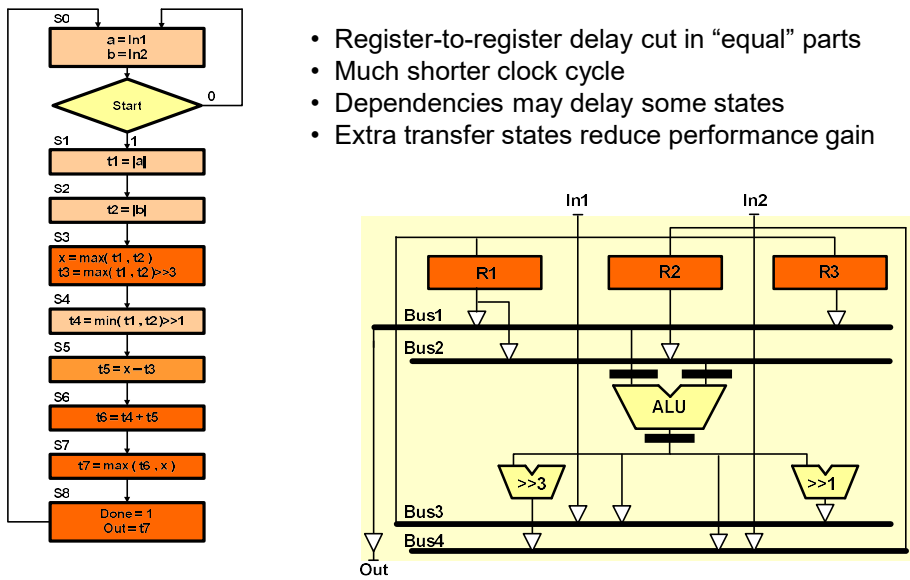
- Operation delay cut in "half"
- Shorter clock cycle
- Dependencies may delay (stall) some states
- Extra stall states reduce performance gain



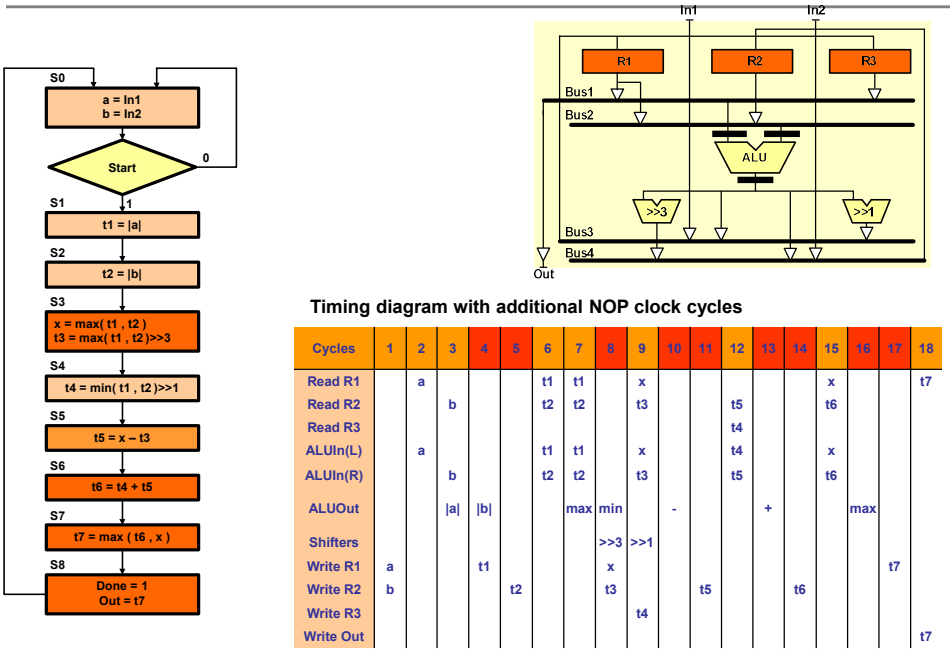
Functional Unit Pipelining (2)



Datapath Pipelining (1)

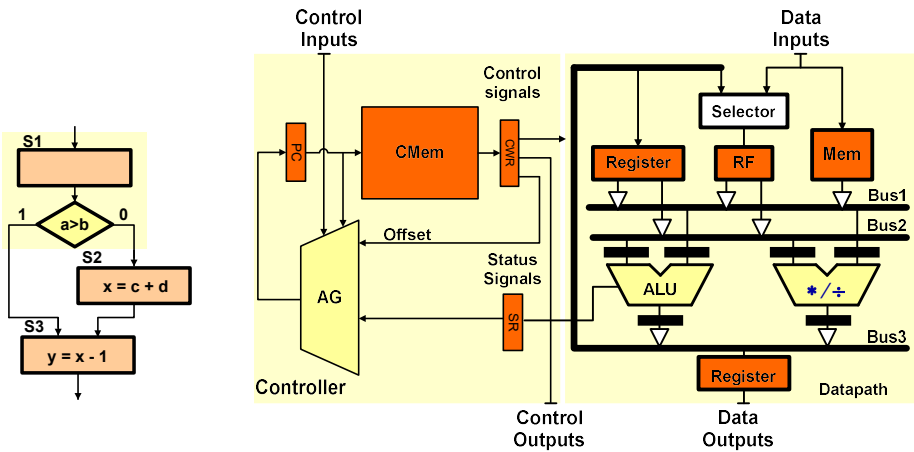


Datapath Pipelining (2)



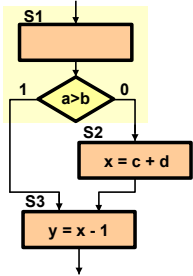
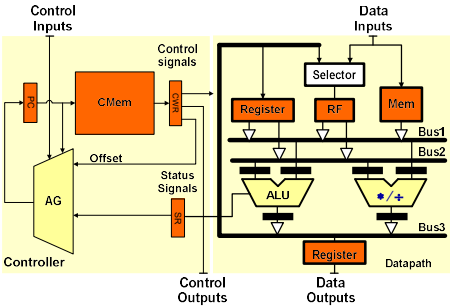
Datapath and Control Pipelining (1)

- Fetch delay cut into several parts
- Shorter clock cycle
- Conditionals may delay some states
- Extra NOP states reduce performance gain



Datapath and Control Pipelining (2)

- 3 NOP cycles for the branch
- 2 NOP cycles for data dependence



Timing diagram with additional NOP clock cycles

Operation	Cycle	0	1	2	3	4	5	6	7	8	9	10
Read PC		10	11	12	13	14	15	16	17	18	19	20
Read CWR			S1	NO	NO	NO	S2	NO	NO	S3		
Read RF(L)			a				c			x		
Read RF(R)			b				d			1		
Write ALUIn(L)			a				c			x		
Write ALUIn(R)			b				d			1		
Write ALUOut							c+d				x-1	
Write RF								x				y
Write SR												
Write PC		11	12	13	14/17	15	16	17	18	19	20	

Lecture 6: Outline

- ✓ Introduction
 - ✓ High-level synthesis
- ✓ Essential issues
 - ✓ Behavioral specification languages
 - ✓ Target architectures
 - ✓ Intermediate representation
 - ✓ Scheduling/allocation/binding
 - ✓ Control generation
- High-level synthesis flow
 - ✓ Synthesis example
 - Formalization & automation

High-Level Synthesis Flow

Lex

Parse

Behavioral Optimization

Arch synth

Logic synth

Lib Binding

Compilation front-end

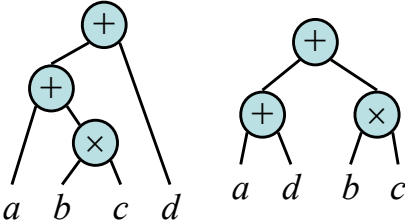
Intermediate form

HLS backend

• Expressions

$$x = a + b \times c + d$$

• Parse tree



➤ Data-flow graph (DFG)

➤ Control/data-flow of basic blocks (CDFG)

ECE382M.20: SoC Design, Lecture 8

© 2023 A. Gerstlauer

51

Source: R. Gupta

Behavioral Optimization

• Data-flow transformations from software compilation

• Tree height reduction

- Balance expression tree, expose parallelism

• Constant and variable propagation ($a = 1; c = 2 * a; \rightarrow c = 2;$)

• Common sub-expression elimination ($a=x+y; c=x+y; \rightarrow c = a;$)

• Dead-code elimination

• Operator strength reduction (e.g., $*4 \rightarrow << 2$)

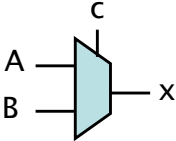
• Control-flow transformations for hardware

• Conditional expansion

- If (c) then $x=A$ else $x=B$
 - compute A and B in parallel, $x=(C)?A:B$

• Loop expansion/unrolling

- Instead of three iterations of a loop, replicate the loop body three times



ECE382M.20: SoC Design, Lecture 8

© 2023 A. Gerstlauer

52

Source: R. Gupta

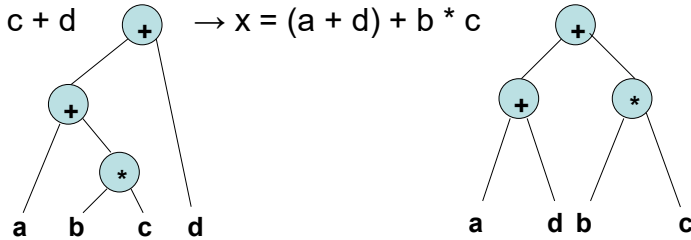
© 2026 A. Gerstlauer

26

Tree-Height Reduction

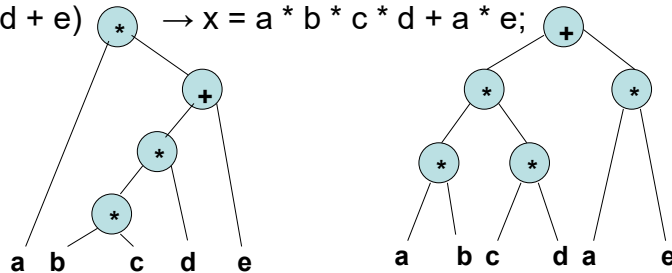
- Commutativity and associativity**

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



- Distributivity**

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



ECE382M.20: SoC Design, Lecture 8

© 2023 A. Gerstlauer

53

Architectural Synthesis

- Deals with “computational” behavioral descriptions**

- Behavior as sequencing graph (called dependency graph, or data flow graph DFG)
- Hardware resources as library elements
 - Pipelined or non-pipelined
 - Resource performance in terms of execution delay
- Constraints on operation timing & clock period
- Constraints on hardware resource availability
- Storage as registers, data transfer using wires

- Objective**

- Generate a synchronous, single-phase clock circuit
- Might have multiple feasible solutions (explore tradeoff)
- Satisfy constraints, minimize objective:
 - Maximize performance subject to area constraint
 - Minimize area subject to performance constraints

Source: R. Gupta

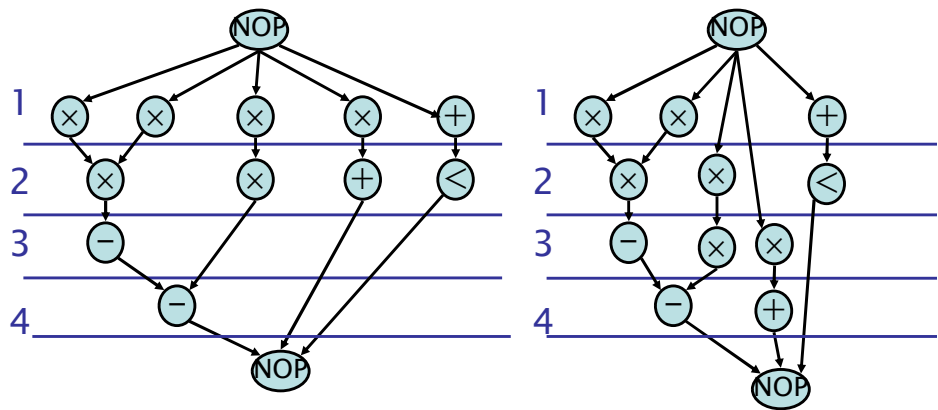
ECE382M.20: SoC Design, Lecture 8

© 2023 A. Gerstlauer

54

Synthesis in Temporal Domain

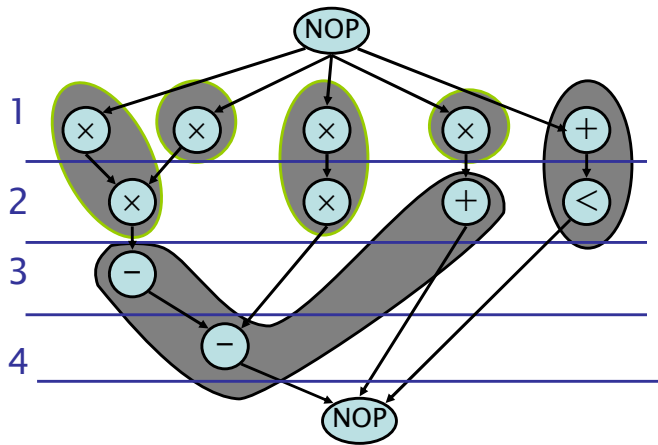
- **Scheduling**
 - Schedule is a mapping of operations to time slots (cycles)
 - Scheduled sequencing graph is a labeled graph



Source: R. Gupta

Synthesis in Spatial Domain

- **Resource binding and sharing**
 - Map each operation to a resource of the same type
 - More than one serial operation bound to same resource
 - Can be represented using hyperedges (vertex partition)

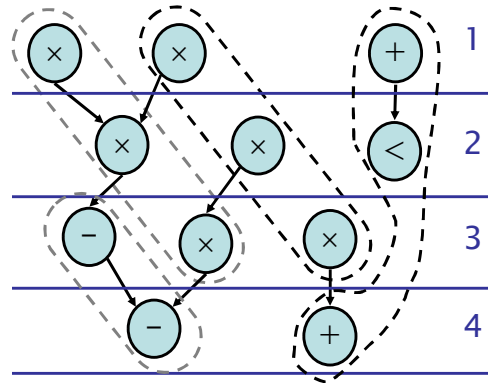


Source: R. Gupta

How Is the Datapath Implemented?

- Assuming the following schedule and binding

- Wires between modules?
- Input selection?
- How does binding/scheduling affect congestion?
- How does binding/scheduling affect steering logic?



ECE382M.20: SoC Design, Lecture 8

© 2023 A. Gerstlauer

57

Scheduling and Binding

- Resource constraints:**

- Mapping of operations into resource types $T(v_2) = adder$
- Number of resource instances of each type $a_{adder} = 2$

- Scheduling:**

- Labeled vertices $\phi(v_2) = 1$

- Binding:**

- Hyperedges (or vertex partitions) $\beta(v_2) = adder1$

- Cost:**

- Number of resources $\sum a_k \approx \text{area}$
- Registers, steering logic (Muxes, busses), wiring, control unit

- Delay:**

- Start time of the “sink” node $\phi(v_{sink})$
- Might be affected by steering logic and schedule (control)
- Resource-dominated vs. ctrl-dominated

ECE382M.20: SoC Design, Lecture 8

© 2023 A. Gerstlauer

58

Architectural Optimization

- **Optimization in view of design space flexibility**
- **A multi-criteria optimization problem**
 - Determine schedule ϕ and binding β .
 - Under area A , latency λ and cycle time τ objectives
- **Find non-dominated points in solution space**
 - Solution space tradeoff curves
 - Non-linear, discontinuous
- **Evaluate (estimate) cost functions**
- **Unconstrained optimization problems**
 - Min area: solve for minimal A binding
 - Min latency: solve for minimum λ scheduling
- **Single-objective optimization problems**
 - Minimize A under λ constraint
 - Minimize λ under A ($\approx$ resource) constraint

ECE382M.20: SoC Design, Lecture 8

© 2023 A. Gerstlauer

59

Scheduling and Binding

- **Cost λ and A determined by both ϕ and β**
 - Also affected by floorplan and detailed routing
- **β affected by ϕ :**
 - Resources cannot be shared among concurrent ops
- **ϕ affected by β :**
 - Resources cannot be shared among concurrent ops
 - When register and steering logic delays added to execution delays, might violate cycle time
- **Order?**
 - Apply either one (scheduling, binding) first

ECE382M.20: SoC Design, Lecture 8

© 2023 A. Gerstlauer

60

Lecture 6: Summary

- **High-level synthesis**
 - Scheduling
 - Resource- or time-constrained
 - Binding and resource sharing
 - Variable merging (storage sharing)
 - Operation merging (FU sharing)
 - Connection merging (bus sharing)
 - Architecture techniques
 - Chaining and multi-cycling
 - FU, datapath and control pipelining
- **Design automation**
 - Formalization & algorithms
 - Scheduling & binding formalism
 - Optimization problems