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

Lecture 7 – High-Level Synthesis

Sources: Jacob Abraham

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

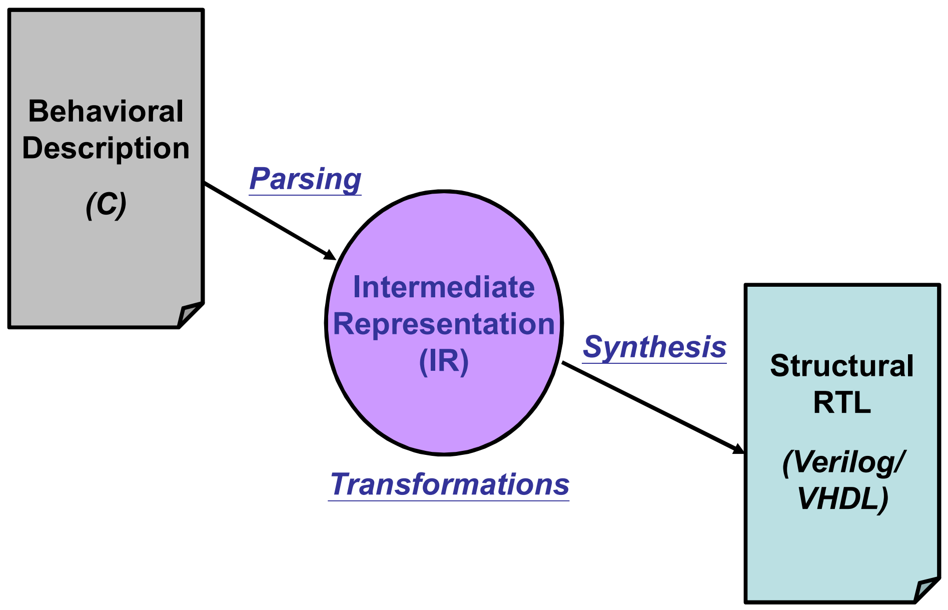
Lecture 7: Outline

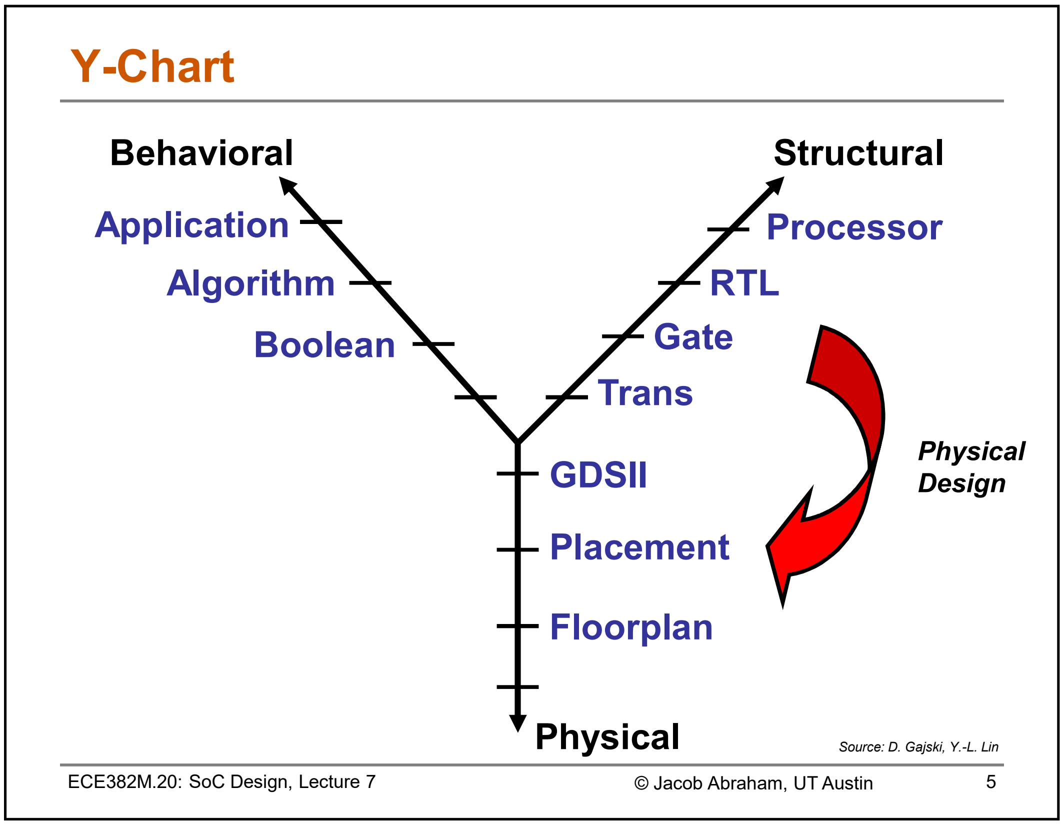
- **Introduction**
  - High-level synthesis (HLS)
- **Essential issues**
  - Behavioral specification languages
  - Target architectures
  - Intermediate representation
  - Scheduling/allocation/binding
  - Control generation
- **High-level synthesis flow**
  - Synthesis example

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

High Level Synthesis

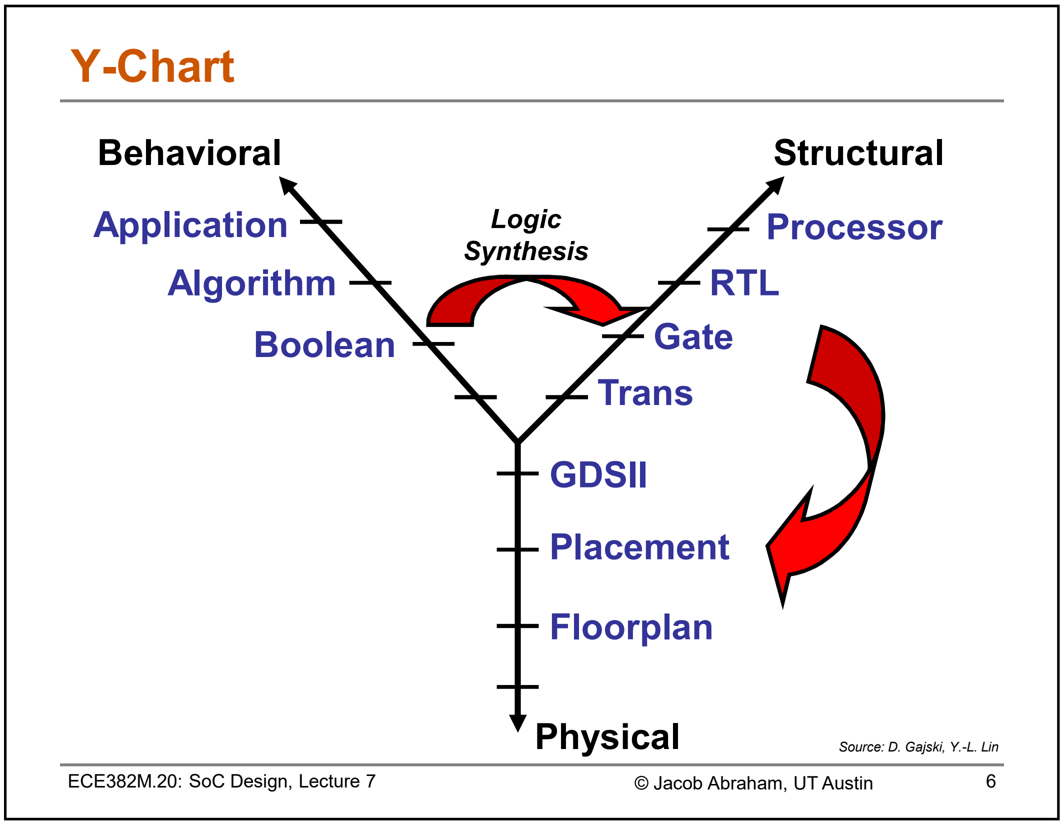




Y-Chart

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

ECE382M.20: SoC Design, Lecture 7 © Jacob Abraham, UT Austin 5

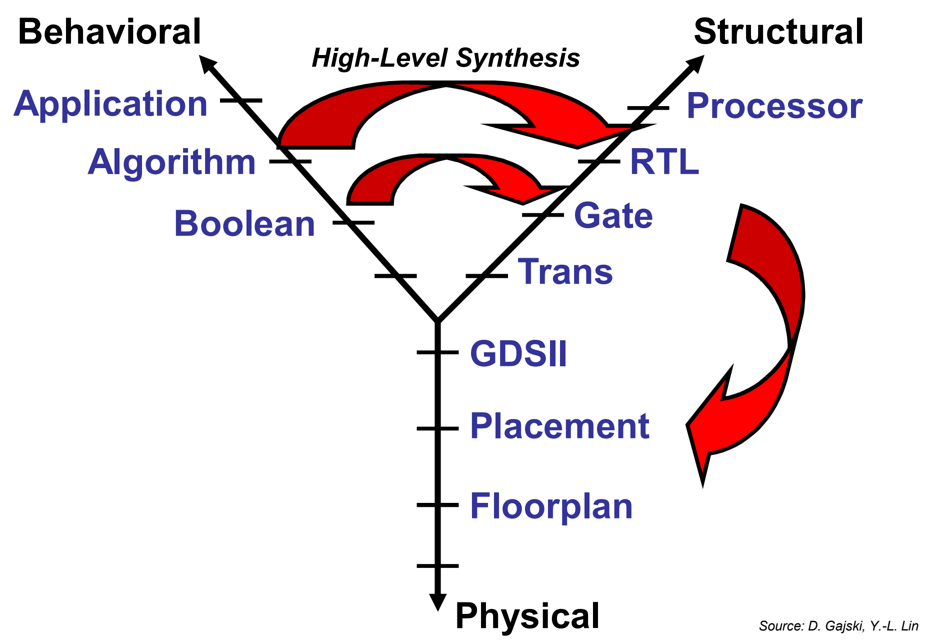


Y-Chart

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

ECE382M.20: SoC Design, Lecture 7 © Jacob Abraham, UT Austin 6

Y-Chart



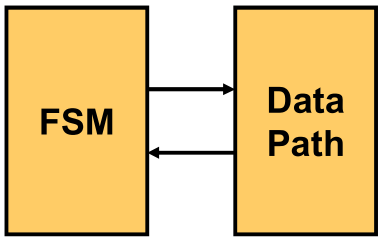
Lecture 7: Outline

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

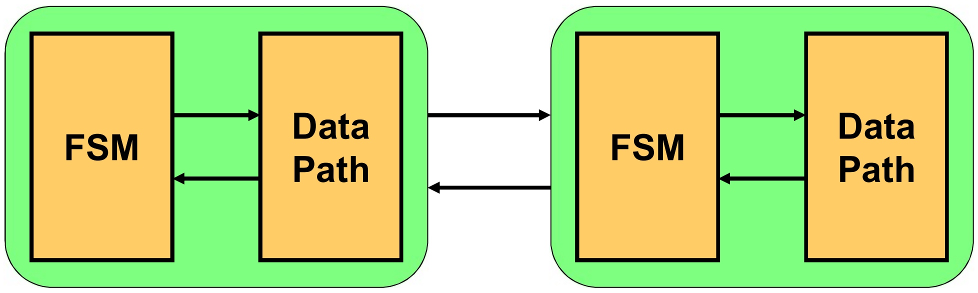
Behavioral Specification Languages

- **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

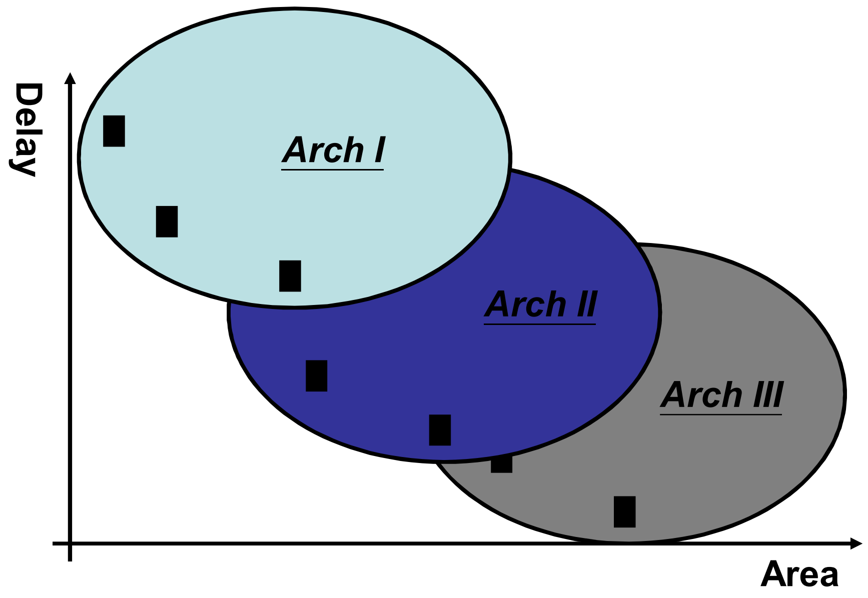
Target Architecture



Finite-State Machine with Data Path



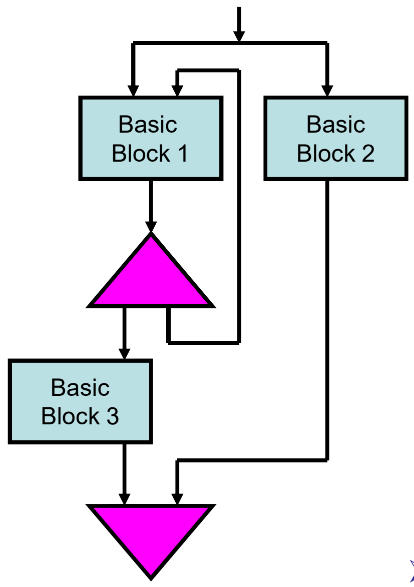
Design Space Exploration



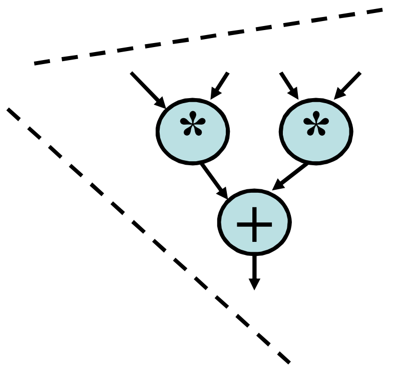
Design Space and Quality Measures

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

Intermediate Representation



Control Flow Graph (CFG)



Data Flow Graph (DFG)

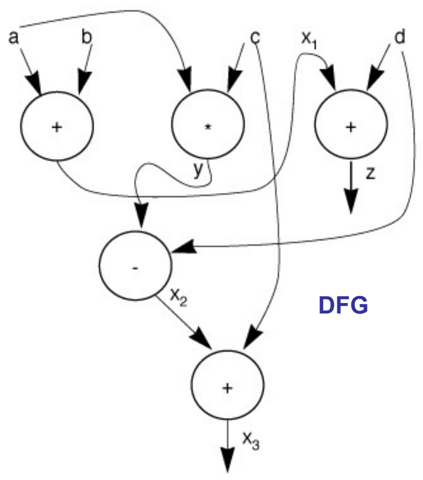
➢ Control/Data Flow Graph (CDFG)

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;
```

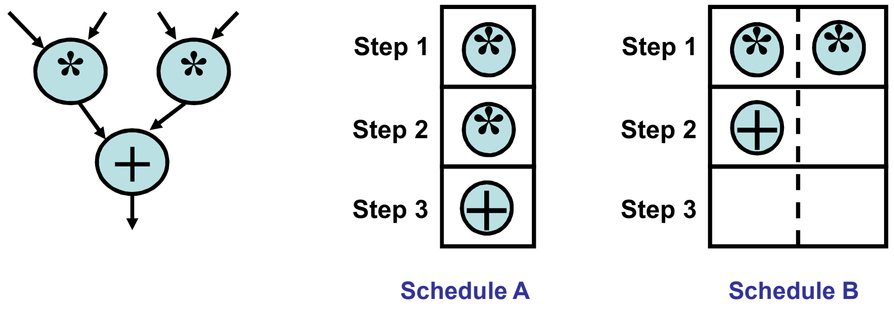
Single assignment form
```
x1 <= a + b;
y <= a * c;
z <= x1 + d;
x2 <= y - d;
x3 <= x2 + c;
```



DFG

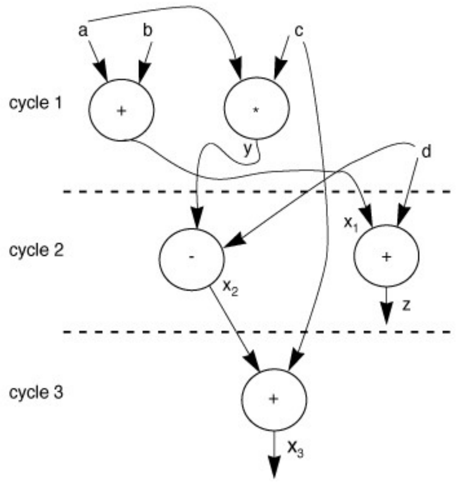
Scheduling

- **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

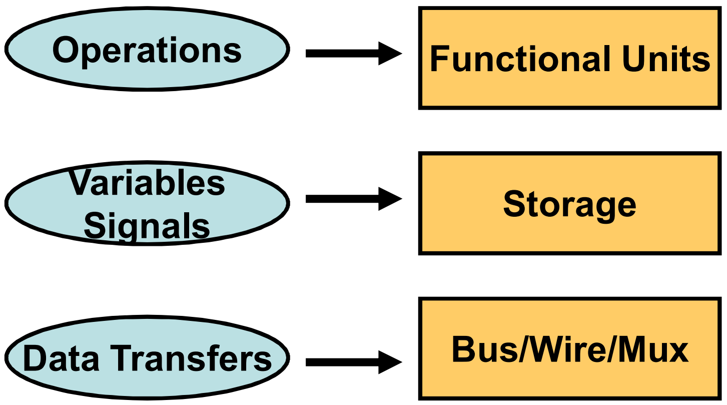


Scheduling Example

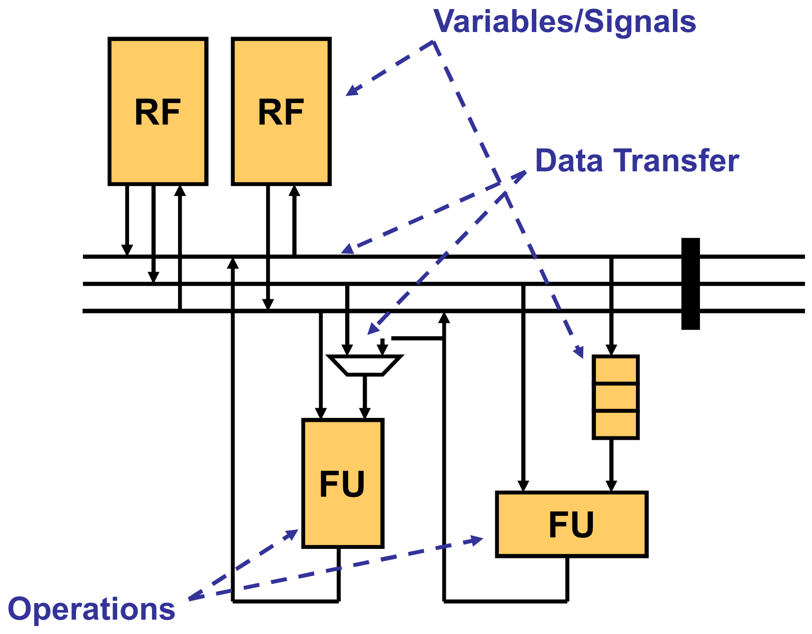
- **One feasible schedule for last DFG**



Allocation/Binding

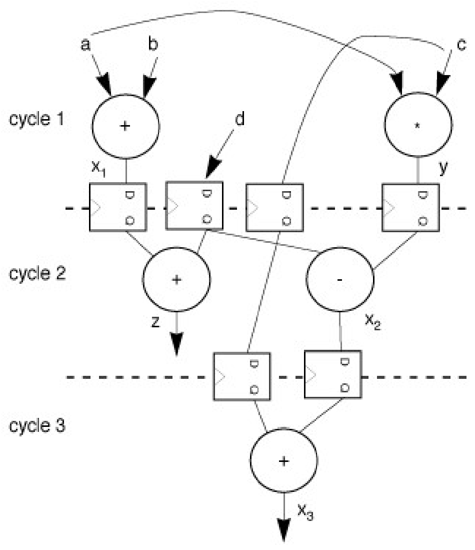


Datapath



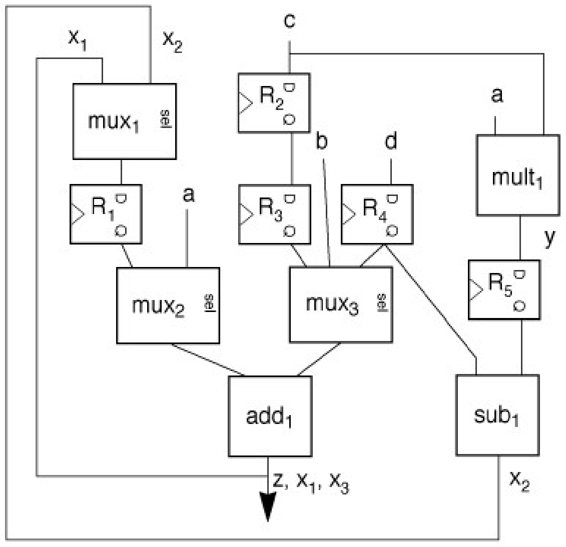
Binding Values to Registers

- **Registers fall on clock cycle boundaries**

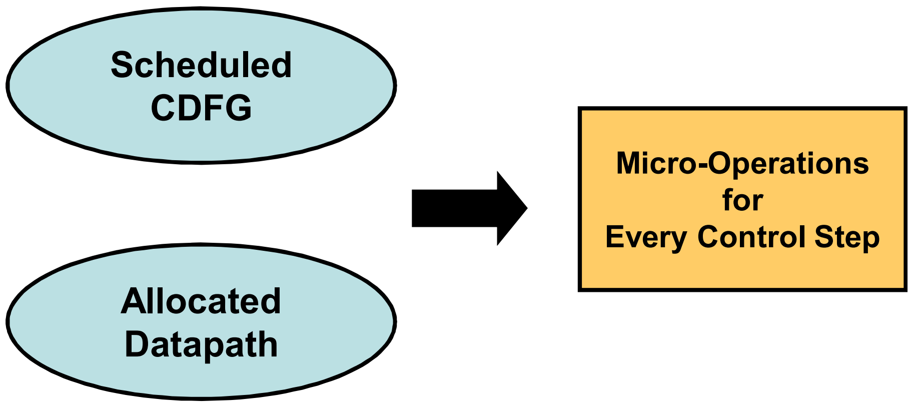


Binding Operations to Functional Units

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

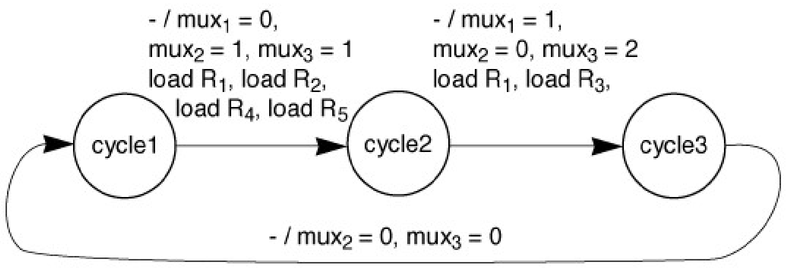


Controller Generation



Building the Sequencer

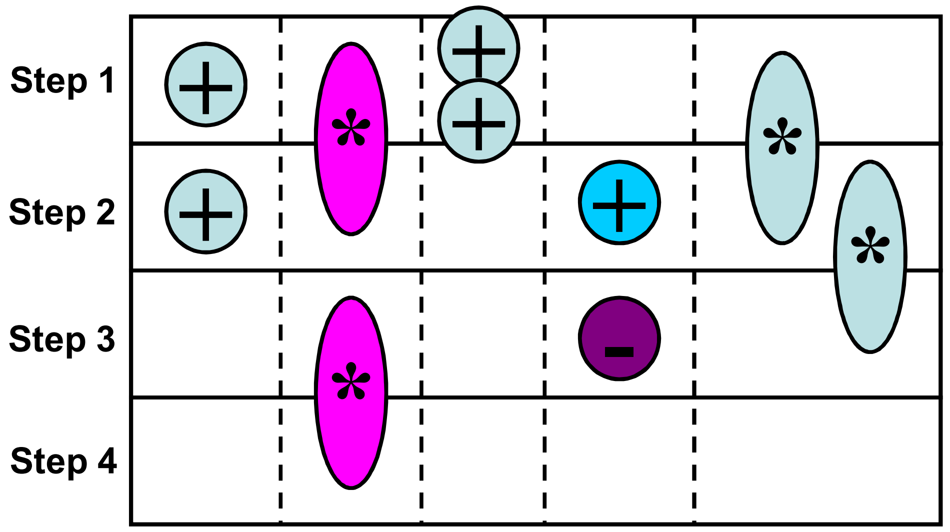
- **Finite state machine (FSM)**
  - Driving datapath control signals
  - Multiple states/cycles even without conditionals

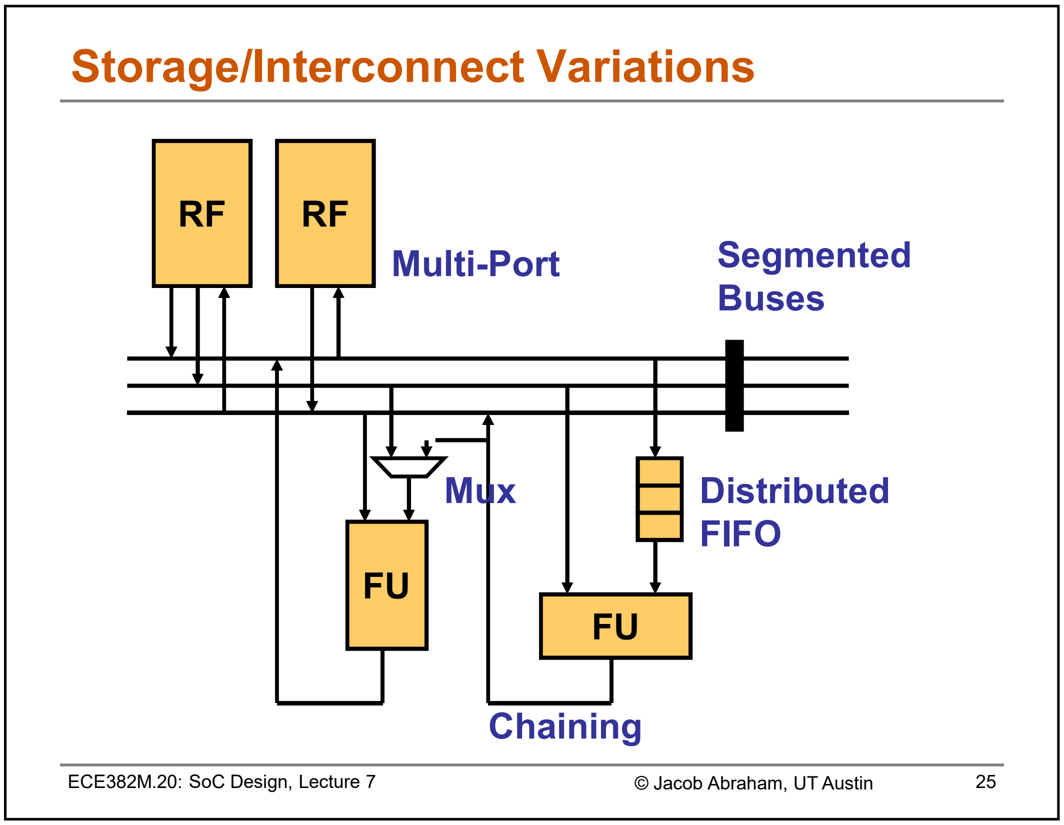

- / mux1 = 0, mux2 = 1, mux3 = 1 load R1, load R2, load R4, load R5
- / mux1 = 1, mux2 = 0, mux3 = 2 load R1, load R3,
cycle1
cycle2
cycle3
- / mux2 = 0, mux3 = 0

Hardware Variations

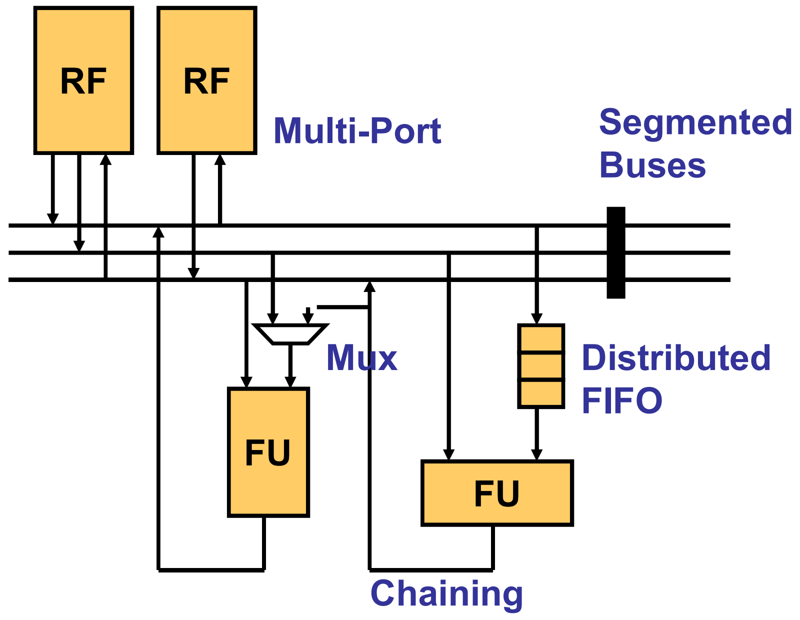
- **Functional Units**
  - Pipelined
  - Multi-cycle
  - Chained
  - Multi-function
- **Storage**
  - Register, register file
  - Single-/multi-ported RAM, ROM
  - FIFO, Scratchpad
- **Interconnect**
  - Bus-based
  - Mux-based
  - Protocol-based

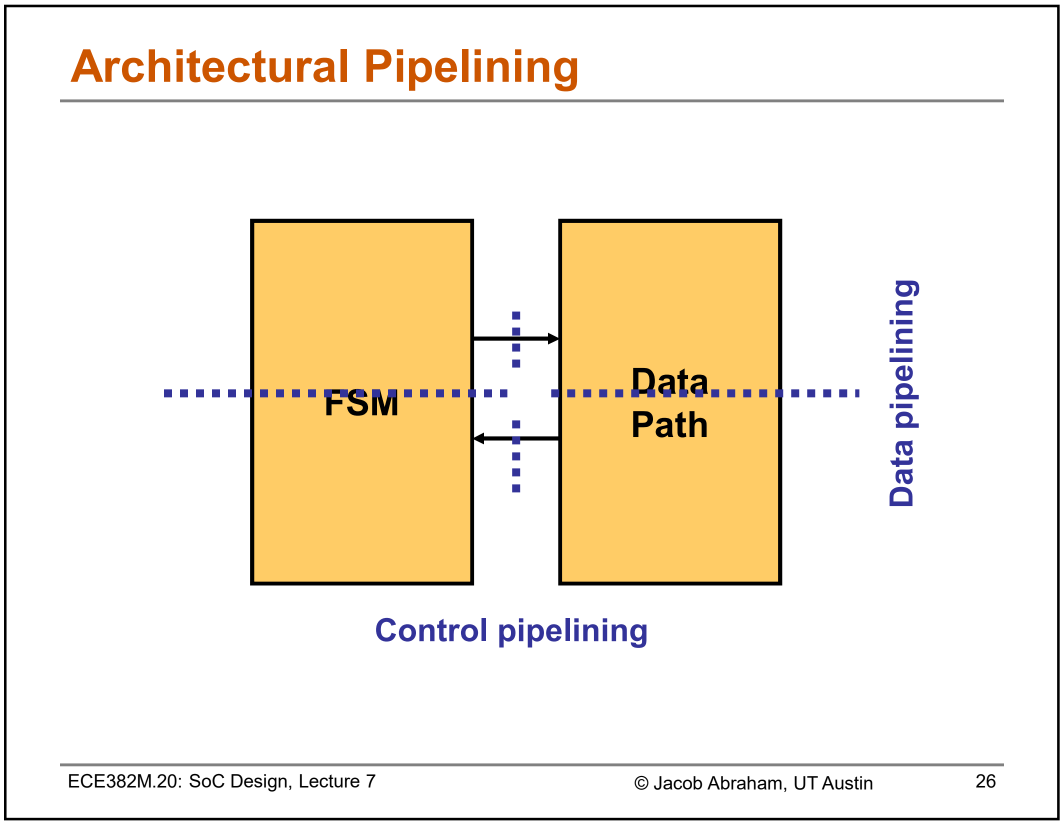
Functional Unit Variations



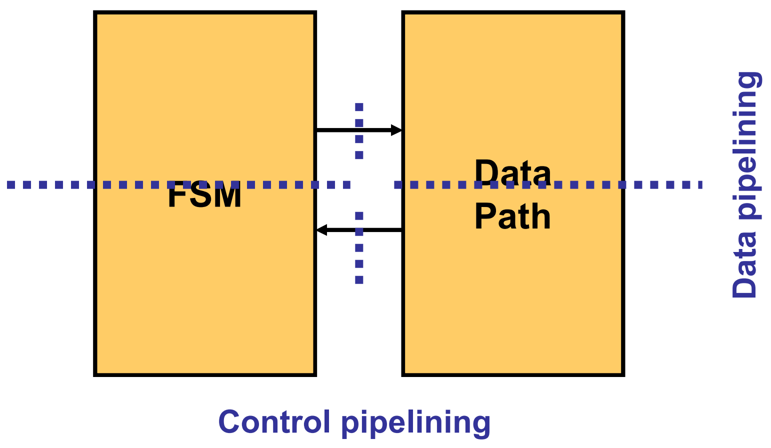


Storage/Interconnect Variations





Architectural Pipelining



Lecture 7: 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

Example 1: C Code for Ones Counter

- **Programming language semantics**
  - Sequential execution,
  - Coding style to minimize coding
- **HW design**
  - Parallel execution,
  - Communication through signals

```
int OnesCounter(int Data){
 int Ocount = 0;
 int Temp, Mask = 1;
 while (Data > 0) {
  Temp = Data & Mask;
  Ocount = Data + Temp;
  Data >>= 1;
 }
 return Ocount;
}
```

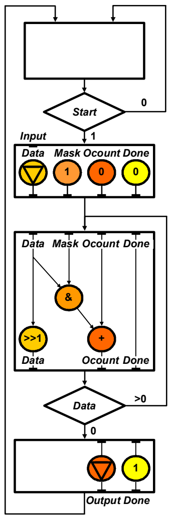
Function-based C code

```
while(1) {
  while (Start == 0);
  Done = 0;
  Data = Input;
  Ocount = 0;
  Mask = 1;
  while (Data>0) {
    Temp = Data & Mask;
    Ocount = Ocount + Temp;
    Data >>= 1;
  }
  Output = Ocount;
  Done = 1;
}
```

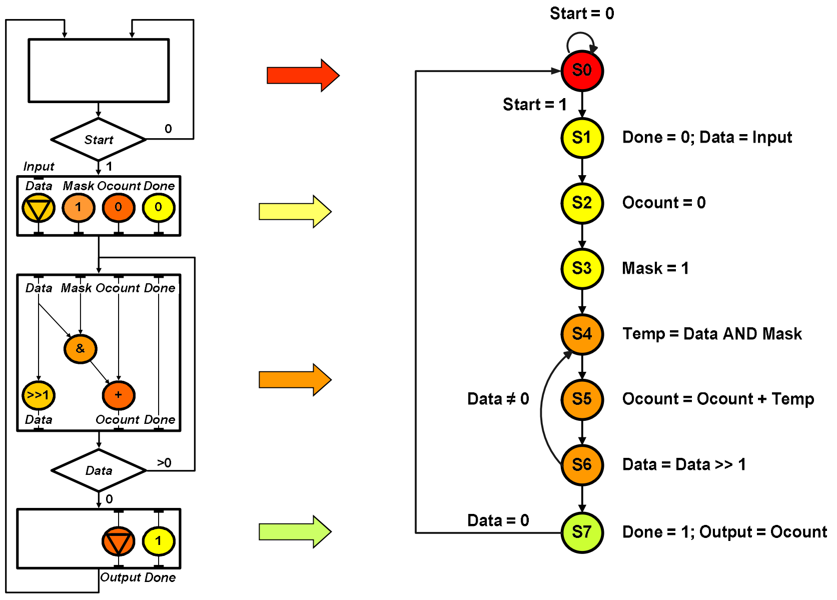
RTL-based C code

CDFG for Ones Counter

- **Control/Data Flow Graph (CDFG)**
  - Resembles programming language
    - Loops, ifs, basic blocks (BBs)
  - Explicit dependencies
    - Control dependences between BBs
    - Data dependences inside BBs
  - Missing dependencies between BBs

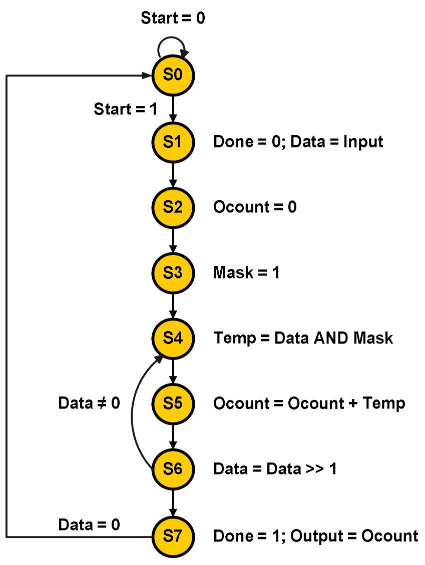


CDFG and FSMD for Ones Counter

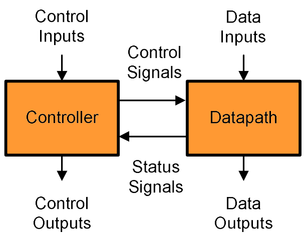


FSMD for Ones Counter

- **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**



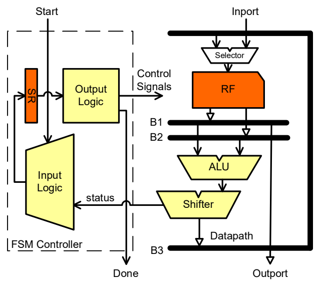
RTL Specification for Ones Counter



- **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	Outport
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

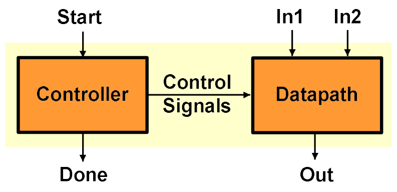
HDL description of Ones Counter

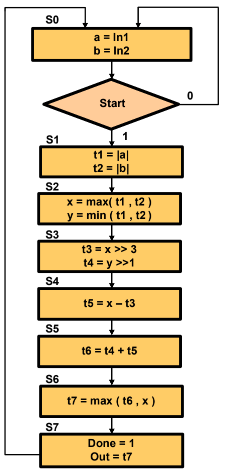
- **HDL description**
  - Same as RTL description
  - Several levels of abstraction
    - Variable binding to storage
    - Operation binding to FUs
    - Transfer binding to connections
- **Netlist must be synthesized**
  - Logic synthesis

```
// …
always@(posedge clk)
begin : output_logic
  case (state)
    // …
    S4: begin
      B1 = RF[0];
      B2 = RF[1];
      B3 = alu(B1, B2, l_and);
      RF[3] = B3;
      next_state = S5;
    end
    // …
    S7: begin
      B1 = RF[2];
      Outport <= B1;
      done <= 1;
      next_state = S0;
    end
  endcase
end
endmodule
```

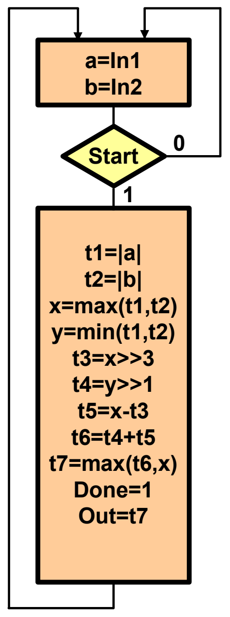
Example 2: Square-Root Algorithm (SRA)

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

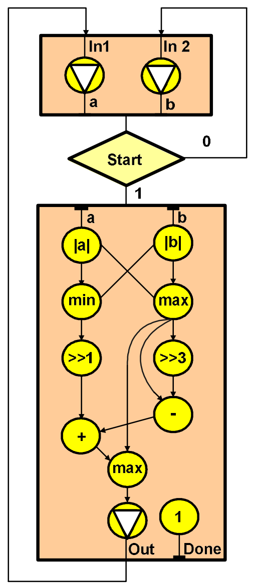




C and CDFG for SRA Algorithm

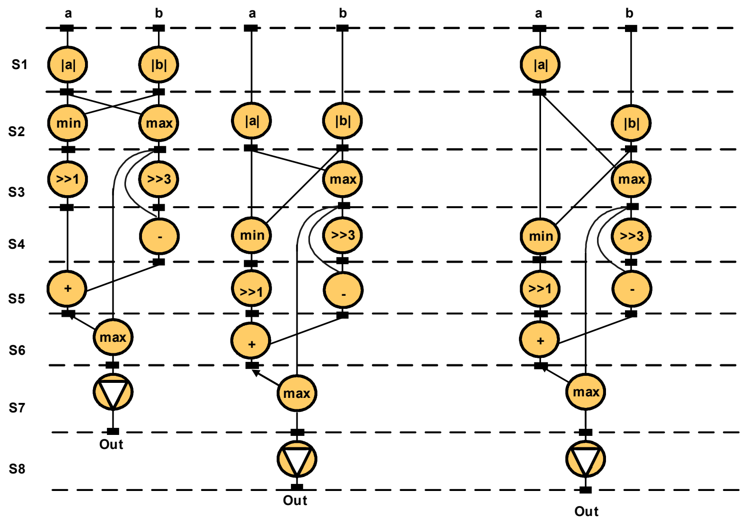


C flowchart



CDFG

SRA Scheduling



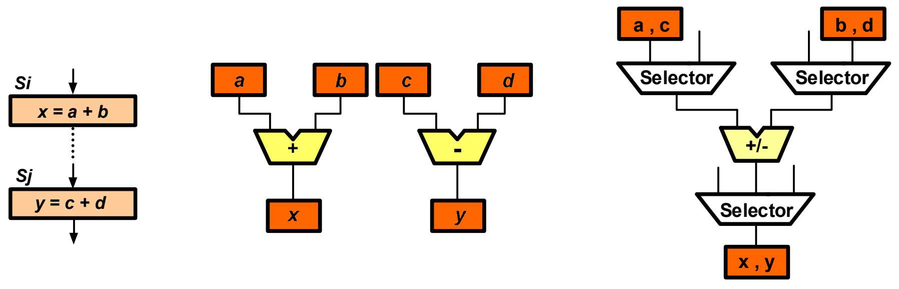
ASAP schedule

ALAP schedule

Constrained schedule (single FU and 2 shifters)

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

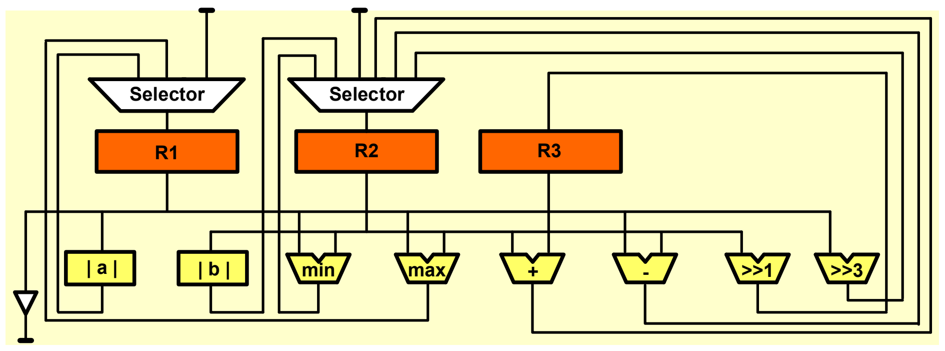


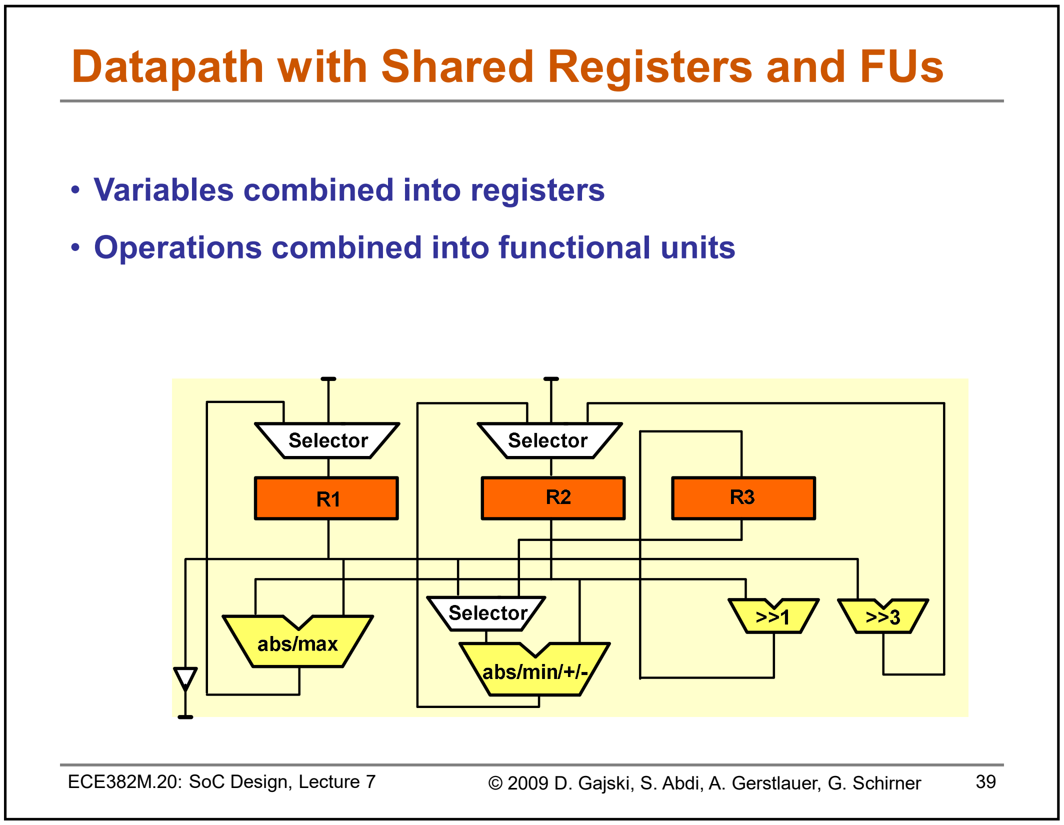
Partial FSMD Datapath without sharing Datapath with sharing

SRA Datapath with Shared Registers

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

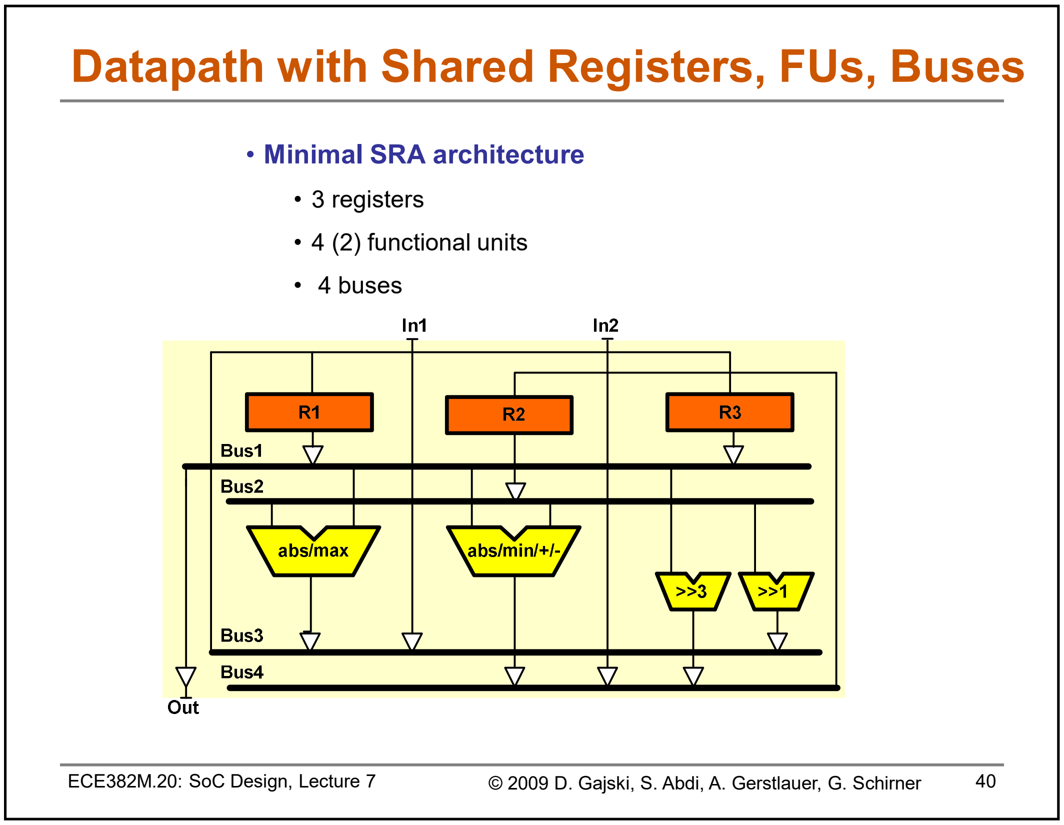
R1 = [a, t1, x, t7]
R2 = [b, t2, y, t3, t5, t6]
R3 = [t4]





Datapath with Shared Registers and FUs

- Variables combined into registers
- Operations combined into functional units

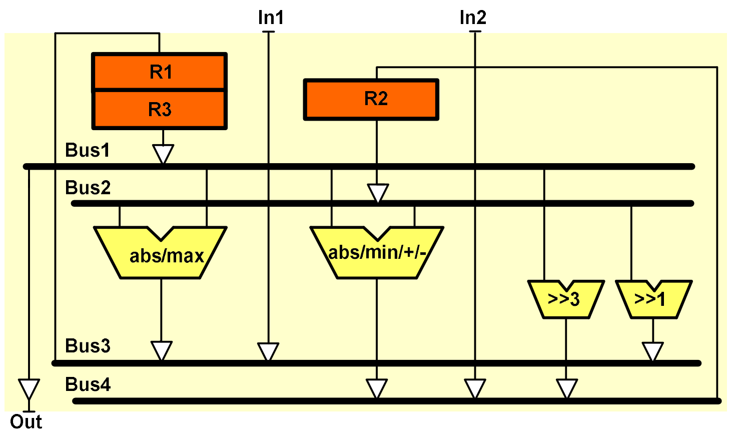


Datapath with Shared Registers, FUs, Buses

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

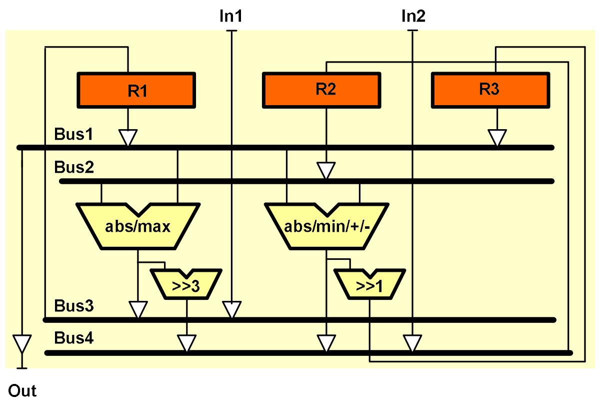
Datapath with Shared RF

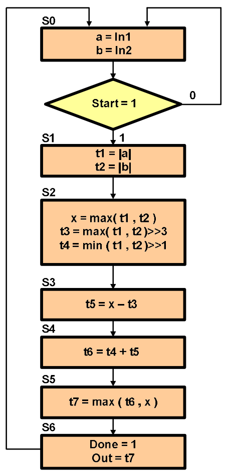
- RF minimize connectivity cost by sharing ports



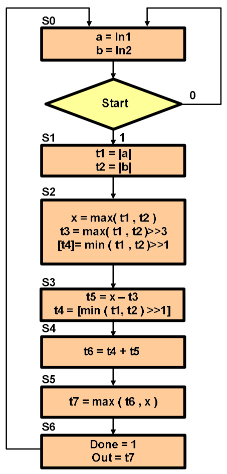
Datapath with Chaining

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

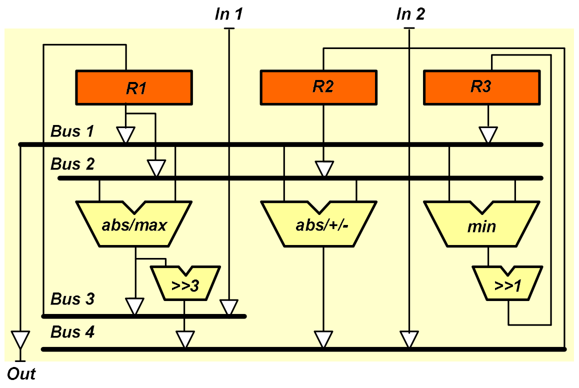




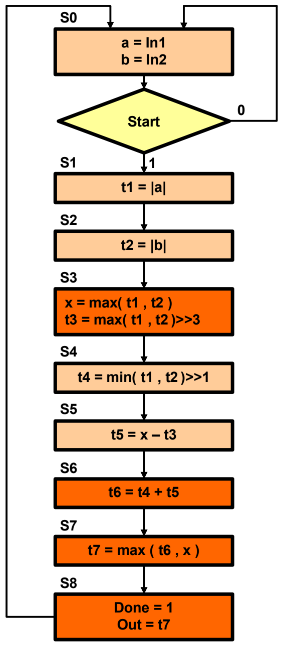
Datapath with Chaining and Multi-Cycling



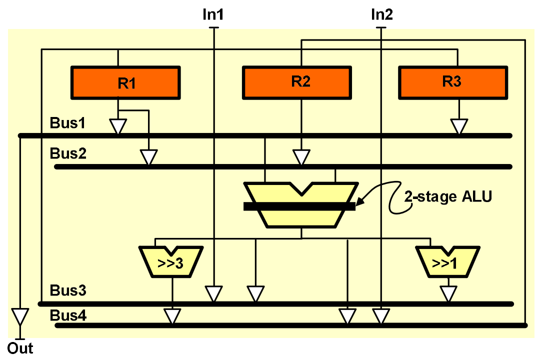
- **Multi-cycling**
  - Operations that take more than one cycle
  - Allows use of slower FUs
  - Allows faster clock-cycle



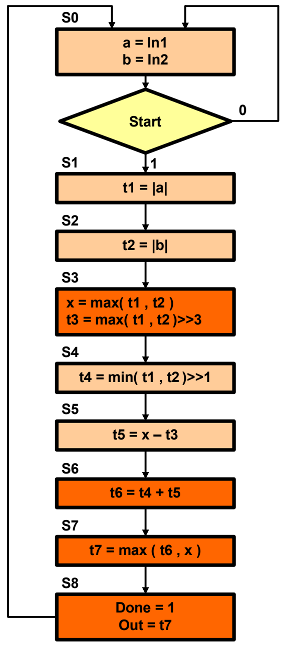
Functional Unit Pipelining (1)

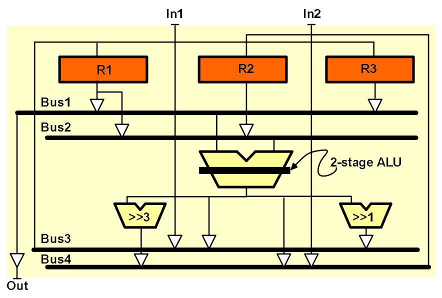


- Operation delay cut in "half"
- Shorter clock cycle
- Dependencies may delay (stall) some states
- Extra stall states reduce performance gain



Functional Unit Pipelining (2)



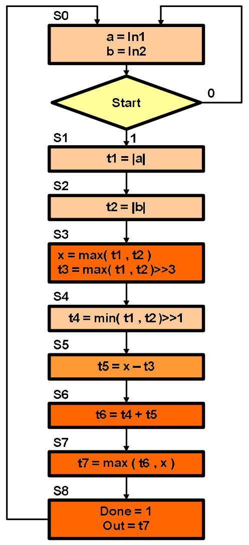


Timing diagram with 4 additional NOP states

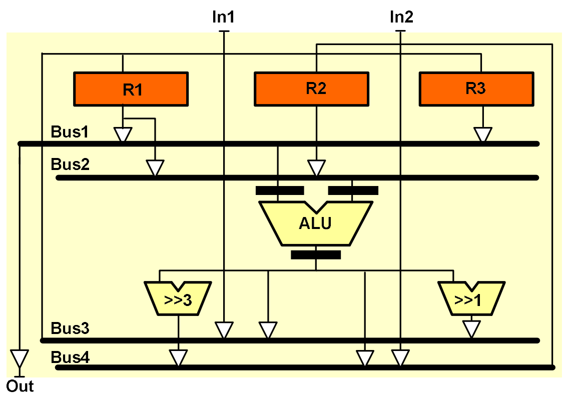
	S0	S1	S2	NO	S3	S4	S5	NO	S6	NO	S7	NO	S8
Read R1		a			t1	t1	X				X		t7
Read R2			b		t2	t2	t3		t5		t6		
Read R3									t4				
ALU stage 1		|a|	|b|		max	min	-		+		max		
ALU stage 2			|a|	|b|		max	min	-		+		max	
Shifters							>>3	>>1					
Write R1	a		t1			X						t7	
Write R2	b			t2		t3		t5		t6			
Write R3							t4						
Write Out													t7

ECE382M.20: SoC Design, Lecture 7   © 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner   45

Datapath Pipelining (1)

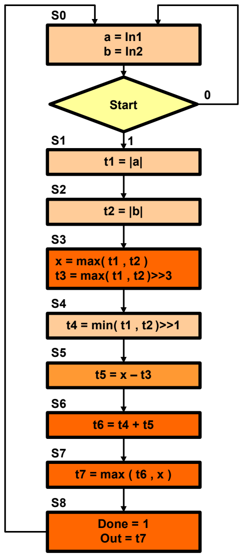


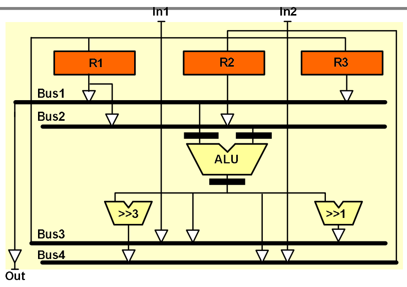
- Register-to-register delay cut in “equal” parts
- Much shorter clock cycle
- Dependencies may delay some states
- Extra transfer states reduce performance gain



ECE382M.20: SoC Design, Lecture 7   © 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner   46

Datapath Pipelining (2)



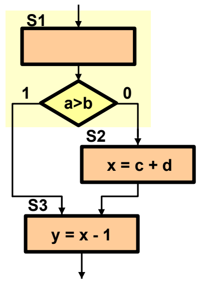


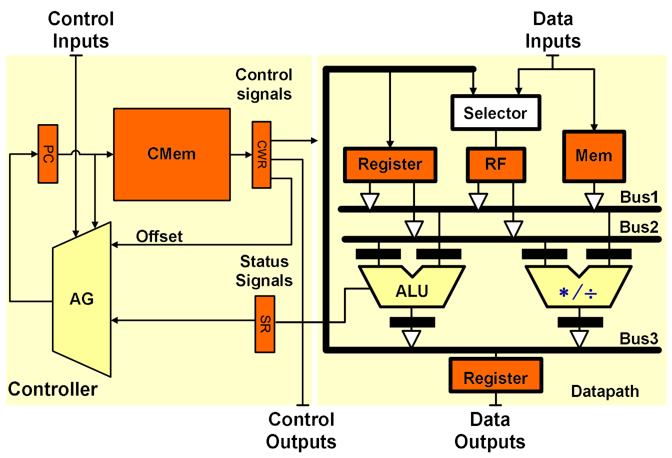
Timing diagram with additional NOP clock cycles

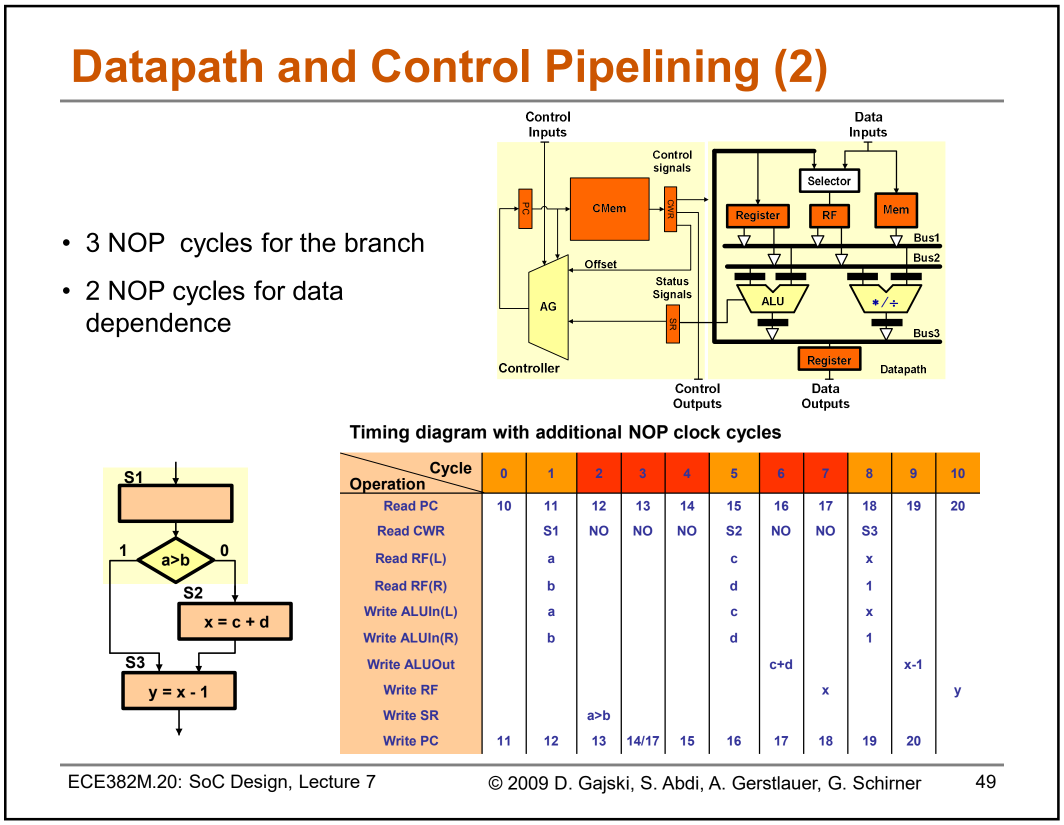
Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Read R1		a				t1	t1		x						x			t7
Read R2			b			t2	t2		t3			t5			t6			
Read R3												t4						
ALUIn(L)		a				t1	t1		x			t4			x			
ALUIn(R)			b			t2	t2		t3			t5			t6			
ALUOut			|a|	|b|			max	min		-			+			max		
Shifters								>>3	>>1									
Write R1	a			t1				x									t7	
Write R2	b				t2			t3			t5			t6				
Write R3									t4									
Write Out																		t7

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

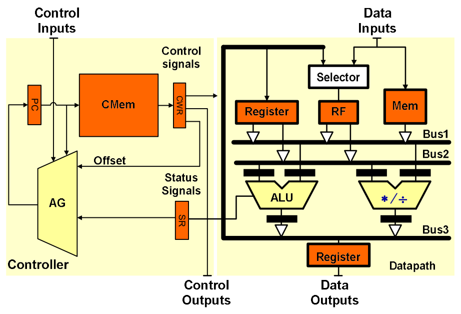


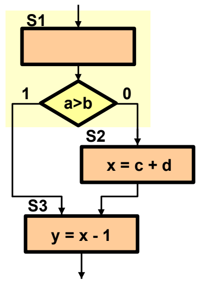




Datapath and Control Pipelining (2)

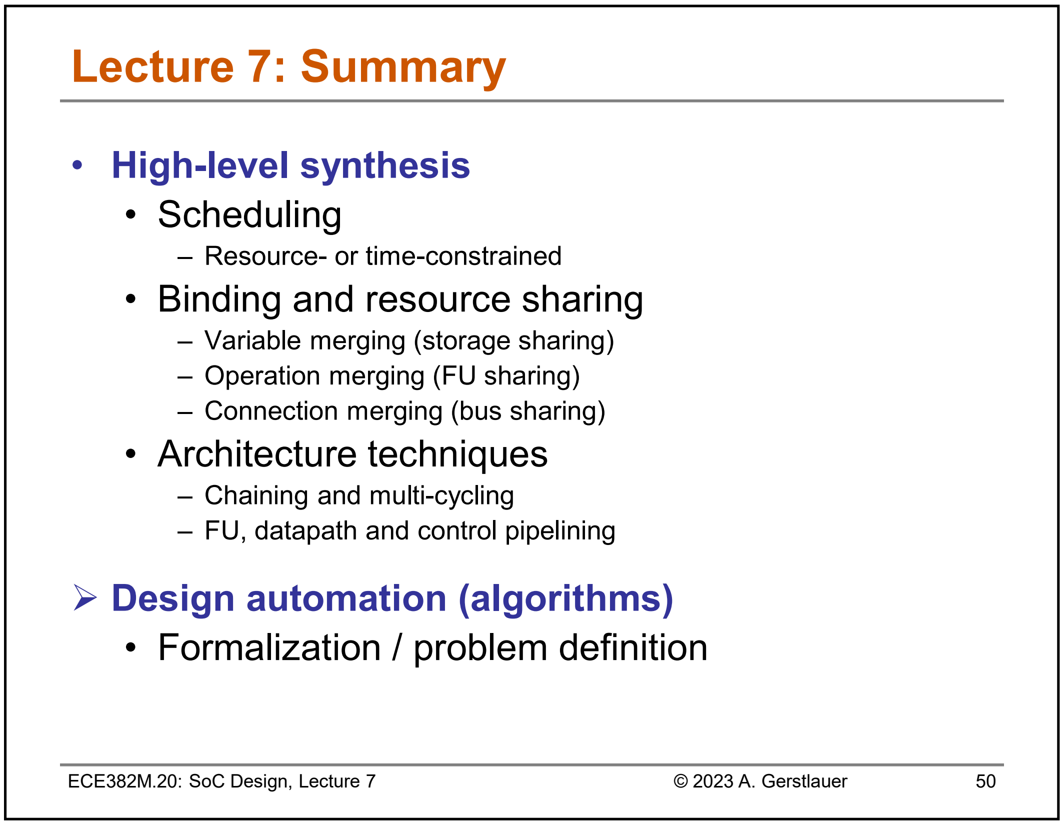
- 3 NOP cycles for the branch
- 2 NOP cycles for data dependence





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			a>b								
Write PC	11	12	13	14/17	15	16	17	18	19	20	



Lecture 7: 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 (algorithms)**
  - Formalization / problem definition