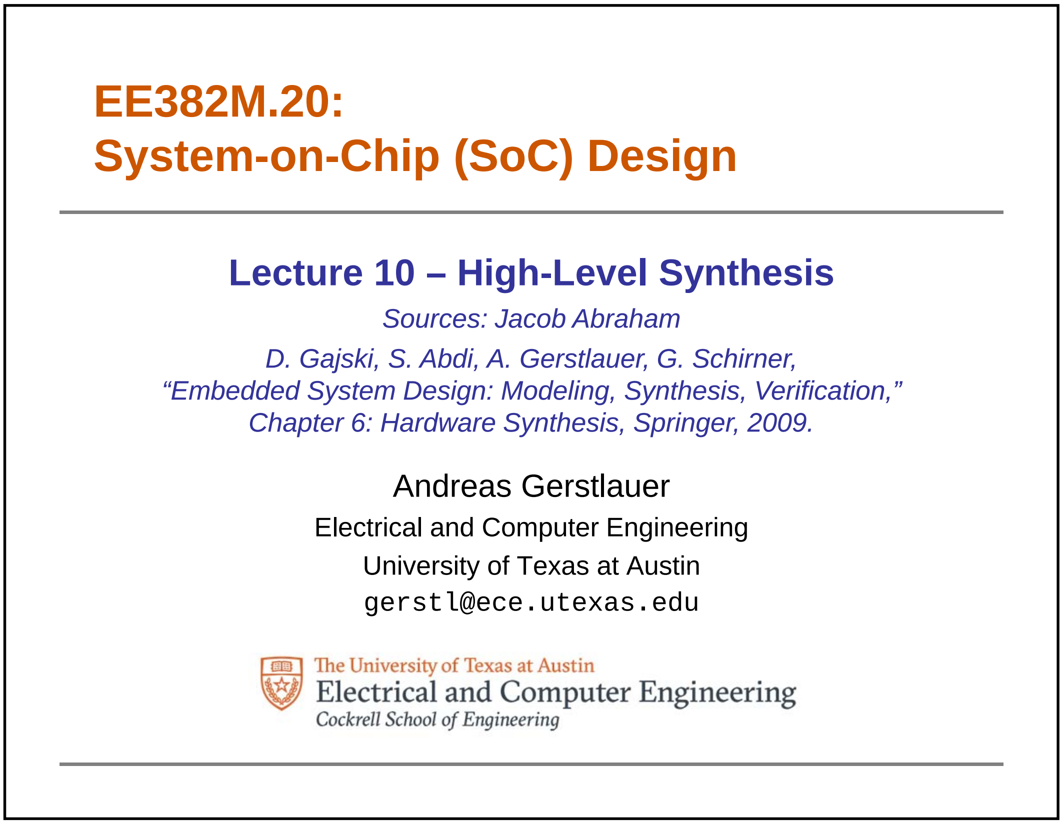


EE382M.20: System-on-Chip (SoC) Design

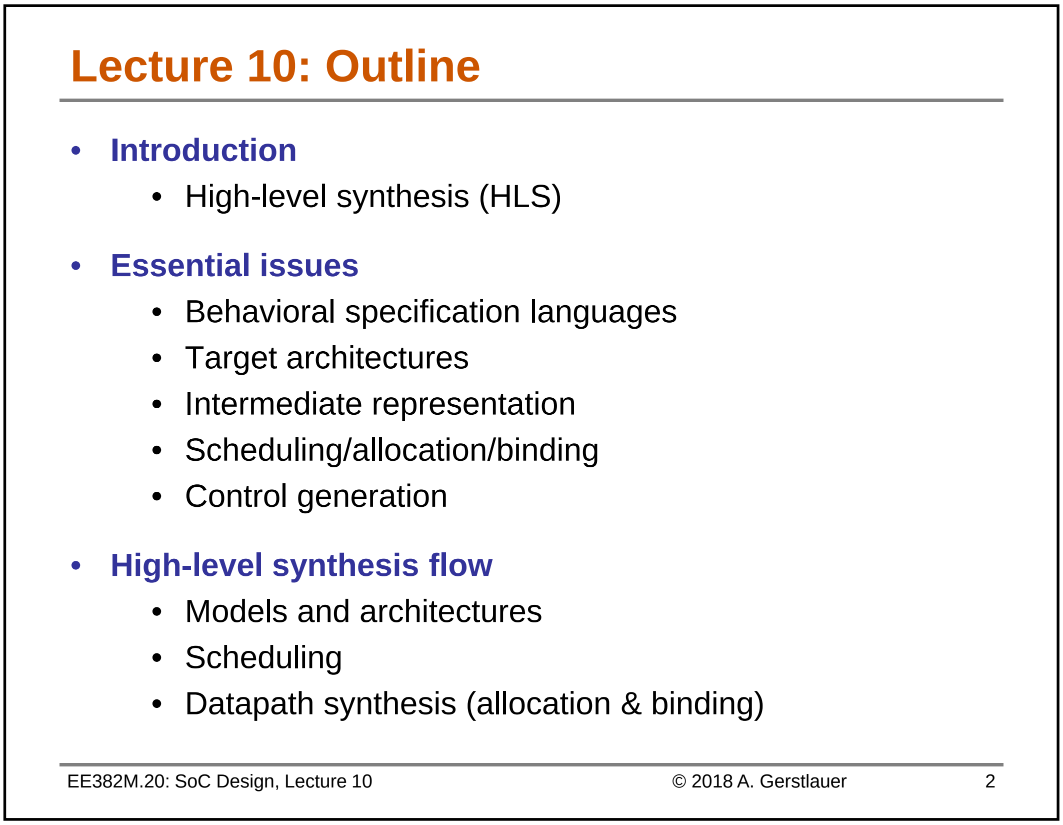
Lecture 10 – High-Level Synthesis

Sources: Jacob Abraham

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

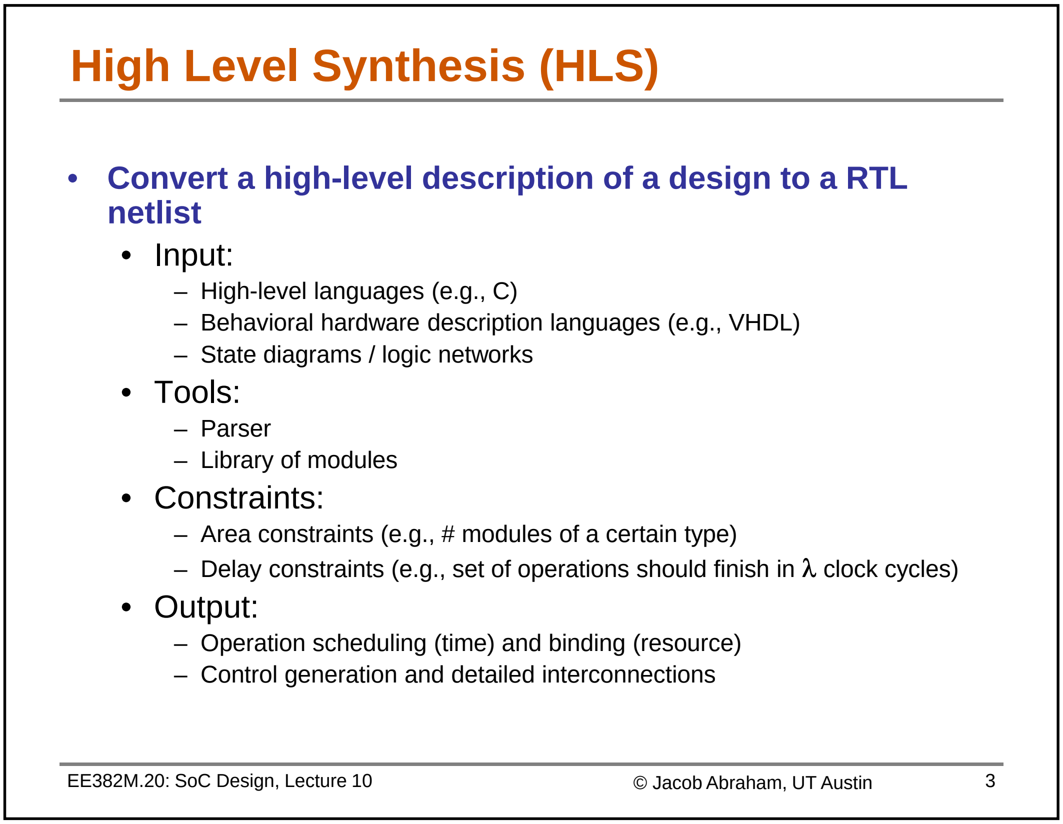
Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering



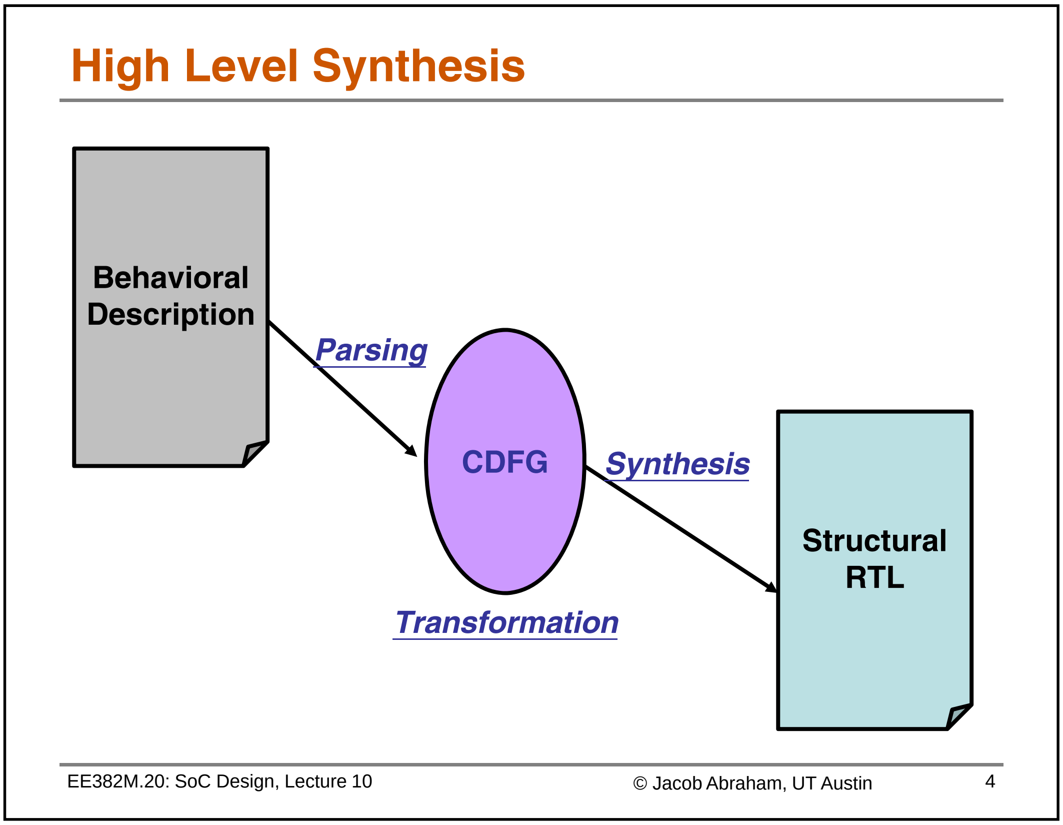
Lecture 10: Outline

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



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

Behavioral Description

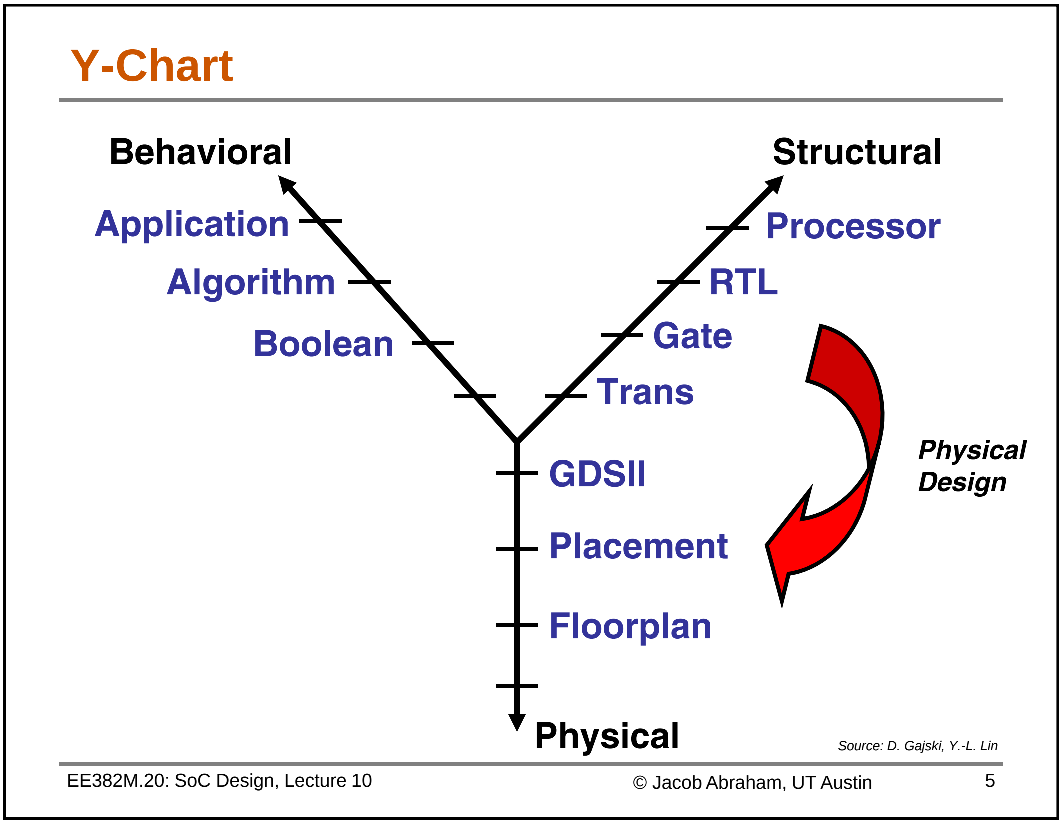
Parsing

CDFG

Synthesis

Structural RTL

Transformation



Y-Chart

Behavioral

Application

Algorithm

Boolean

Structural

Processor

RTL

Gate

Trans

GDSII

Placement

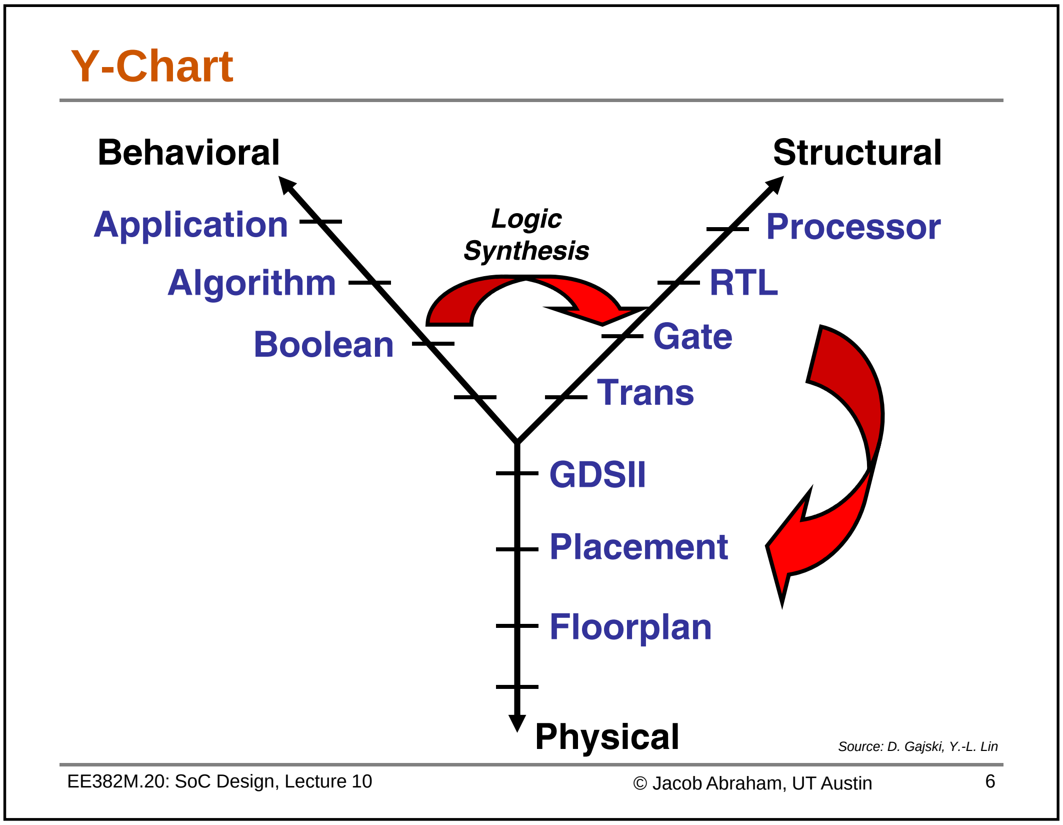
Floorplan

Physical

Physical Design

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

EE382M.20: SoC Design, Lecture 10 © Jacob Abraham, UT Austin 5



Y-Chart

Behavioral

Application

Algorithm

Boolean

Logic Synthesis

Structural

Processor

RTL

Gate

Trans

GDSII

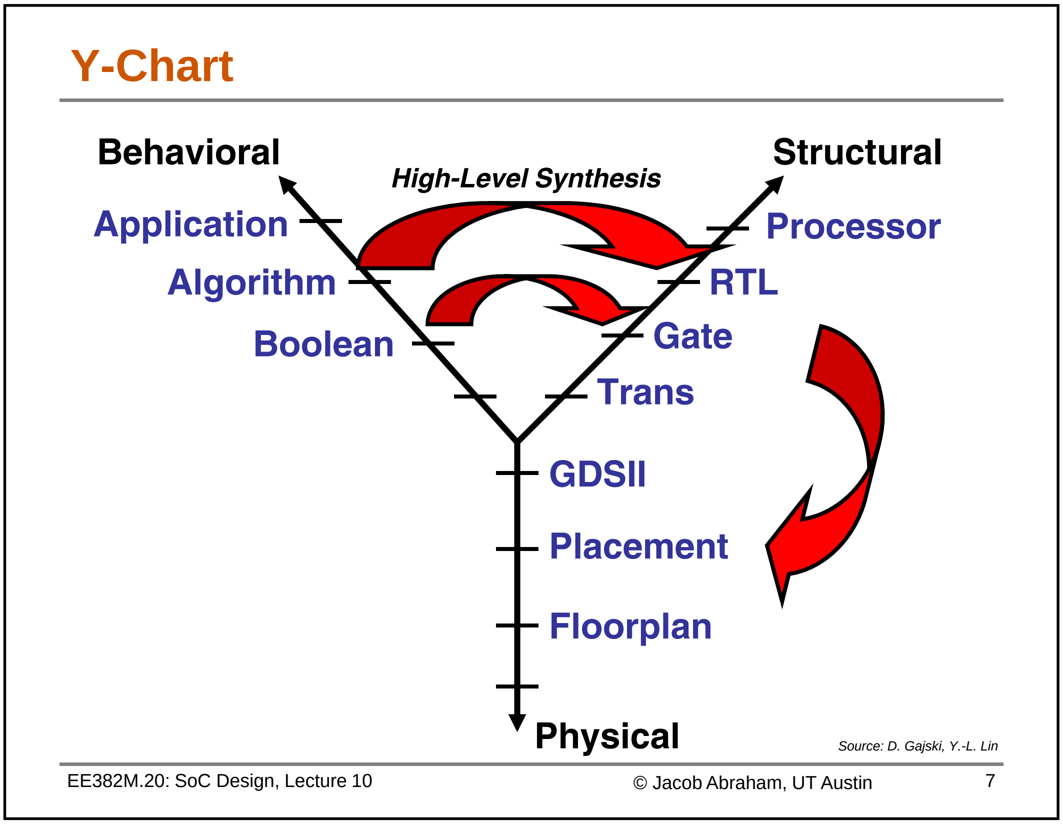
Placement

Floorplan

Physical

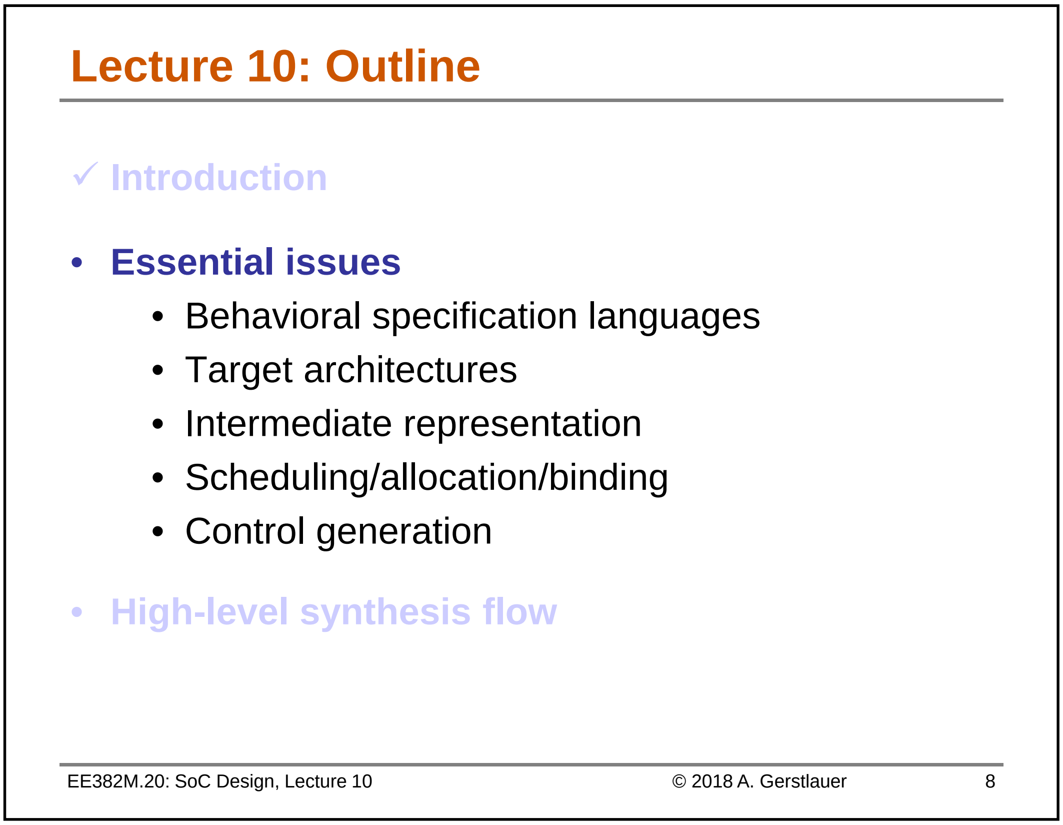
Source: D. Gajski, Y.-L. Lin

EE382M.20: SoC Design, Lecture 10 © Jacob Abraham, UT Austin 6



Y-Chart

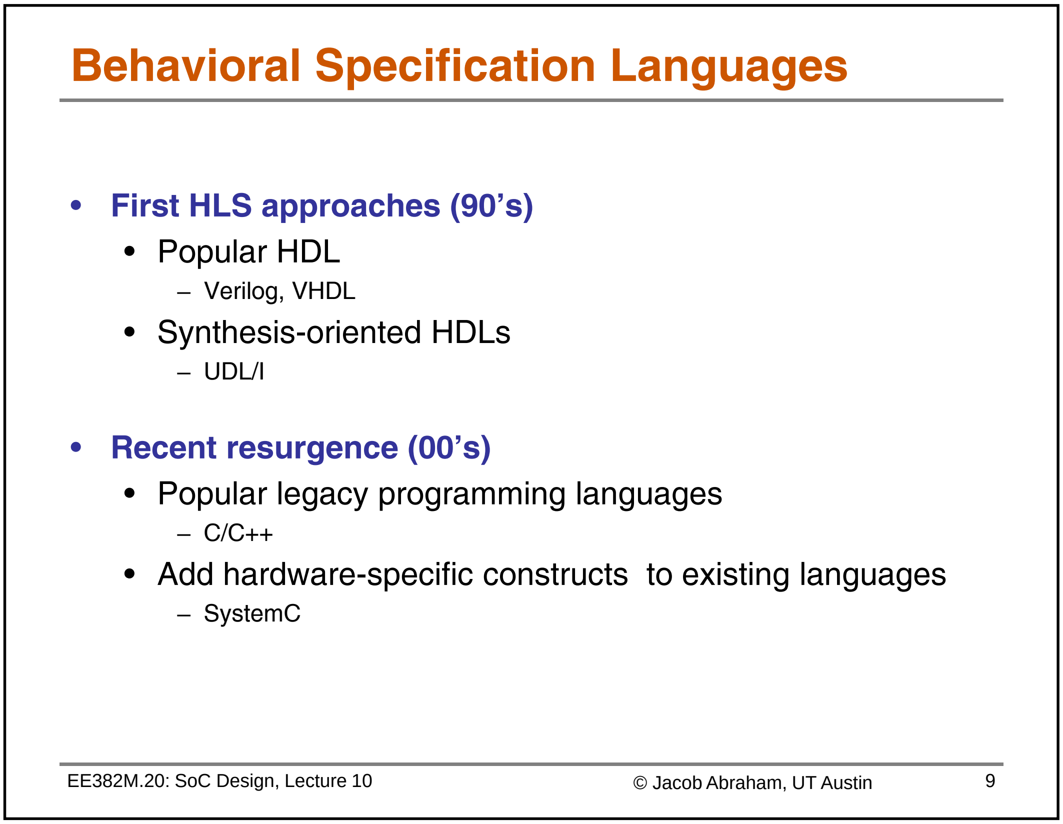
Source: D. Gajski, Y.-L. Lin



Lecture 10: Outline

✓ Introduction

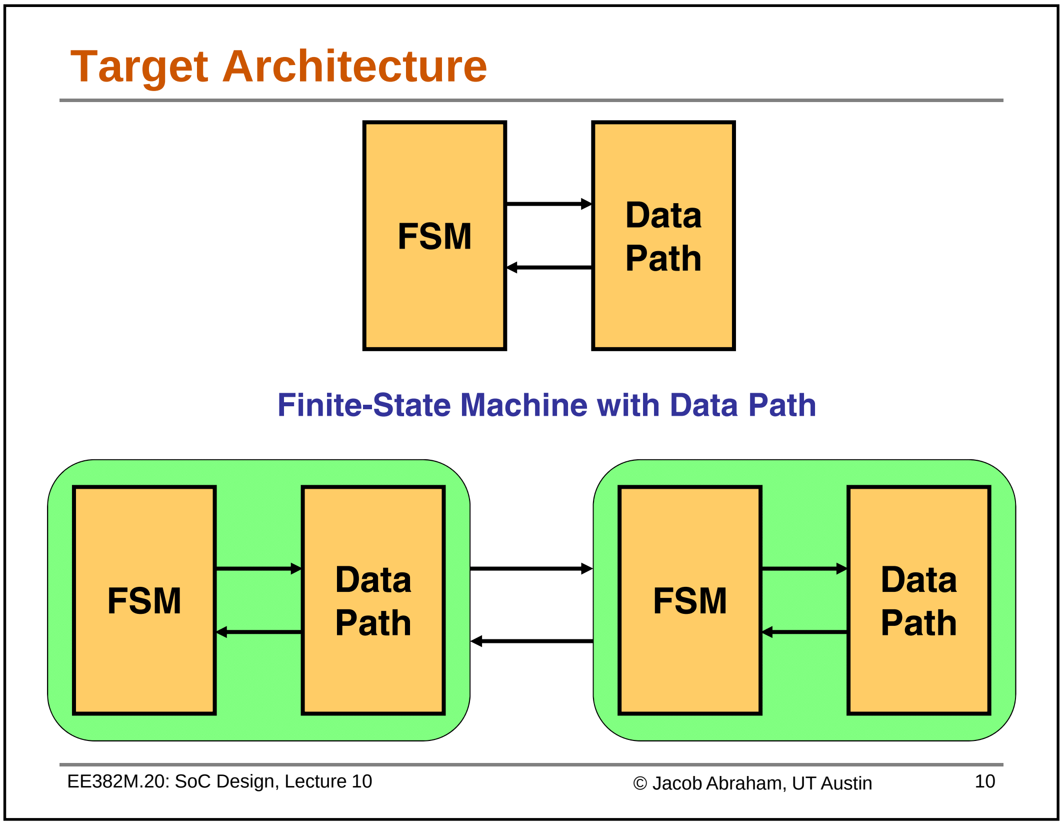
- Essential issues
  - Behavioral specification languages
  - Target architectures
  - Intermediate representation
  - Scheduling/allocation/binding
  - Control generation
- High-level synthesis flow



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

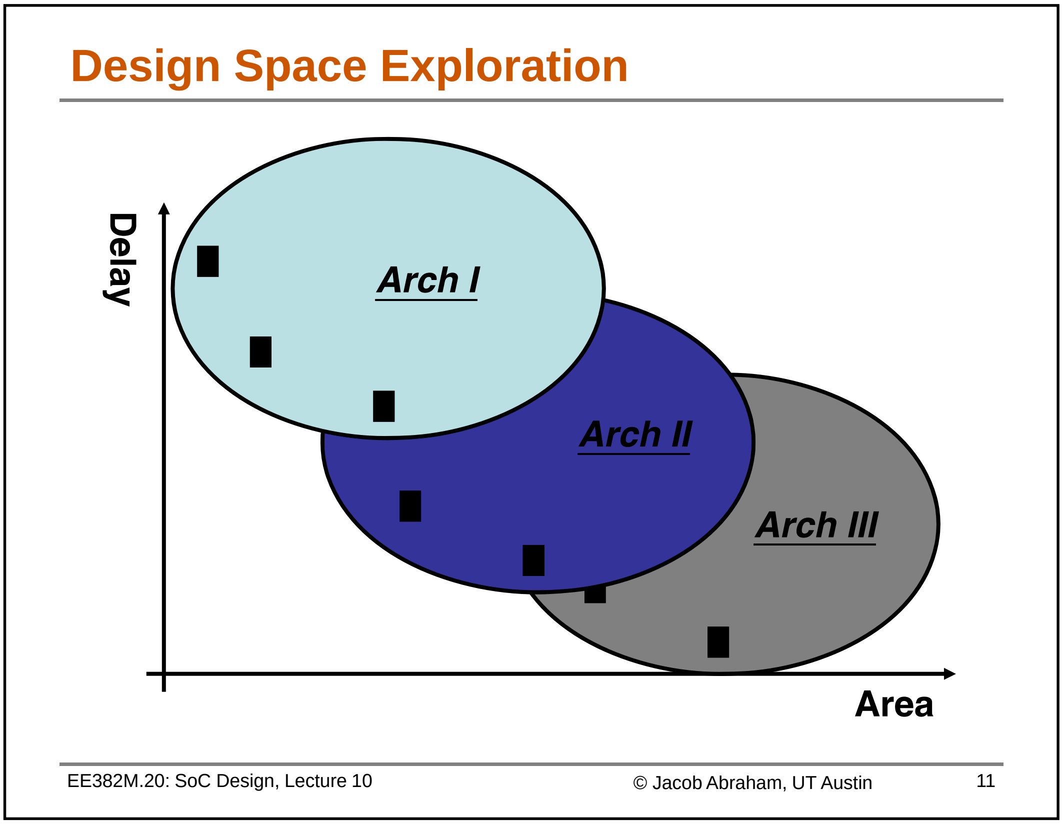
EE382M.20: SoC Design, Lecture 10 © Jacob Abraham, UT Austin 9



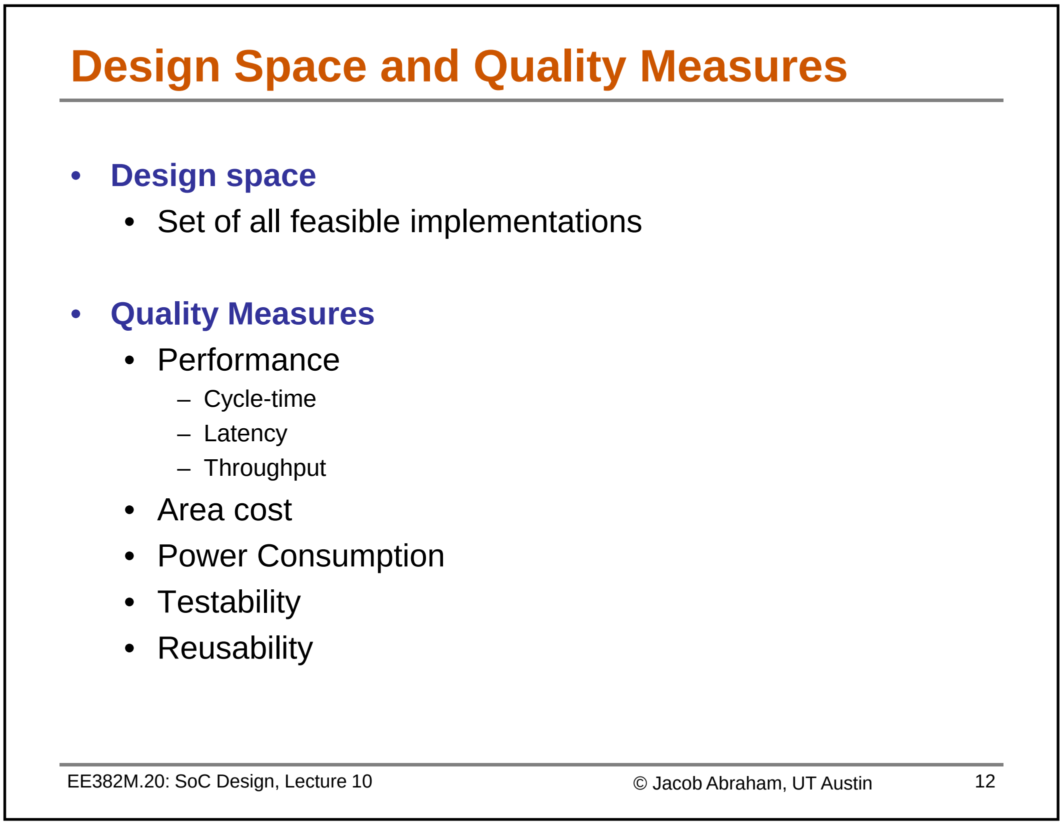
Target Architecture

Finite-State Machine with Data Path

EE382M.20: SoC Design, Lecture 10 © Jacob Abraham, UT Austin 10



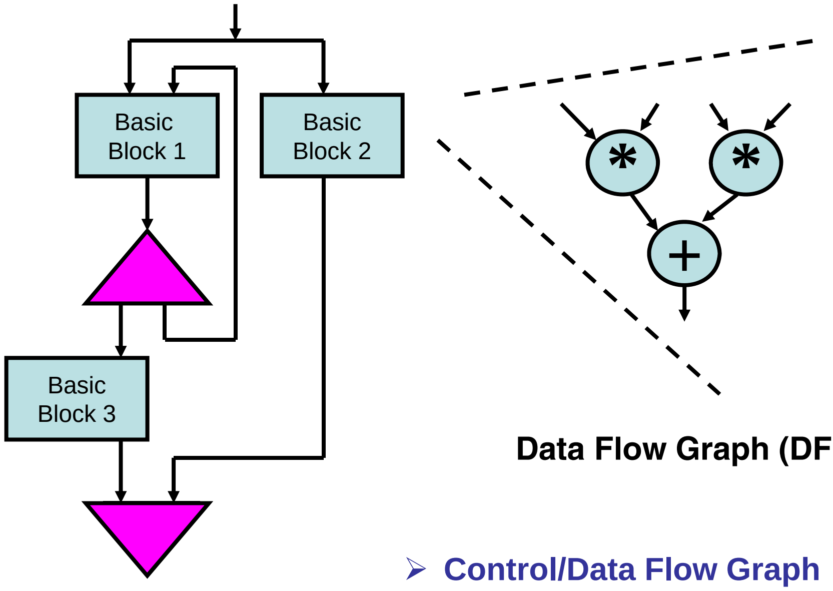
Design Space Exploration



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



Data Flow Graph (DFG)

Control Flow Graph (CFG)

- ➢ **Control/Data Flow Graph (CDFG)**

Data Flow Graph (DFG)

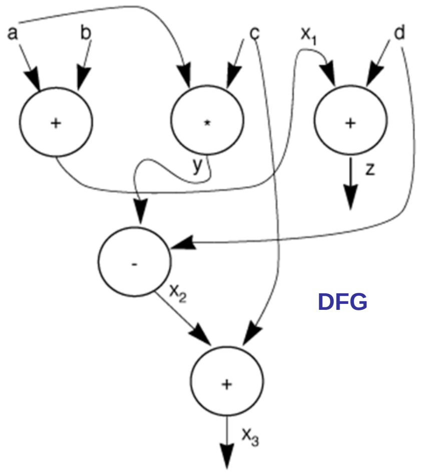
- **Directed acyclic graph (DAG)**
  - Data dependencies, no control/conditionals, no other order

Original code

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

Single assignment form

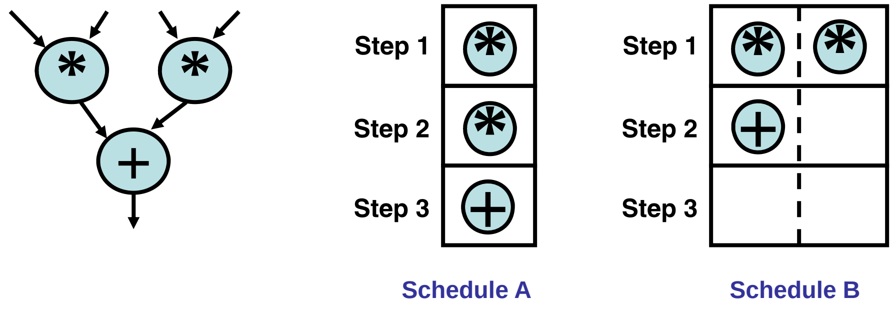
```
x1 <= a + b;
y <= a * c;
z <= x1 + d;
x2 <= y - d;
x3 <= x2 + c;
```



DFG

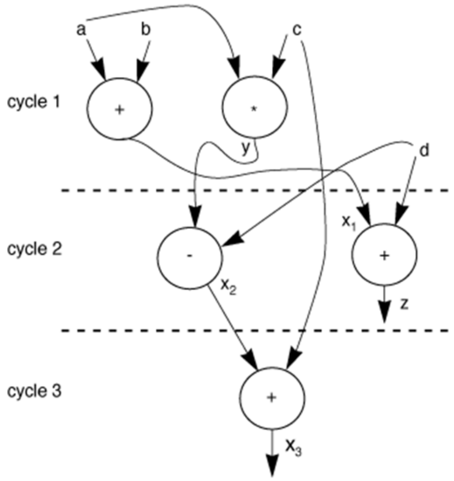
Scheduling

- **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

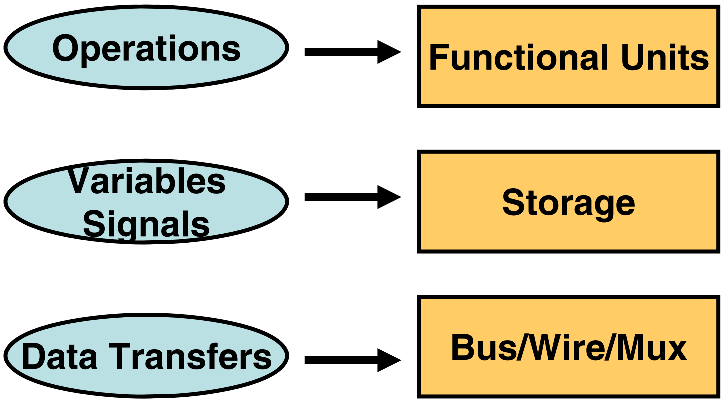


Scheduling Example

- **One feasible schedule for last DFG**



Allocation/Binding

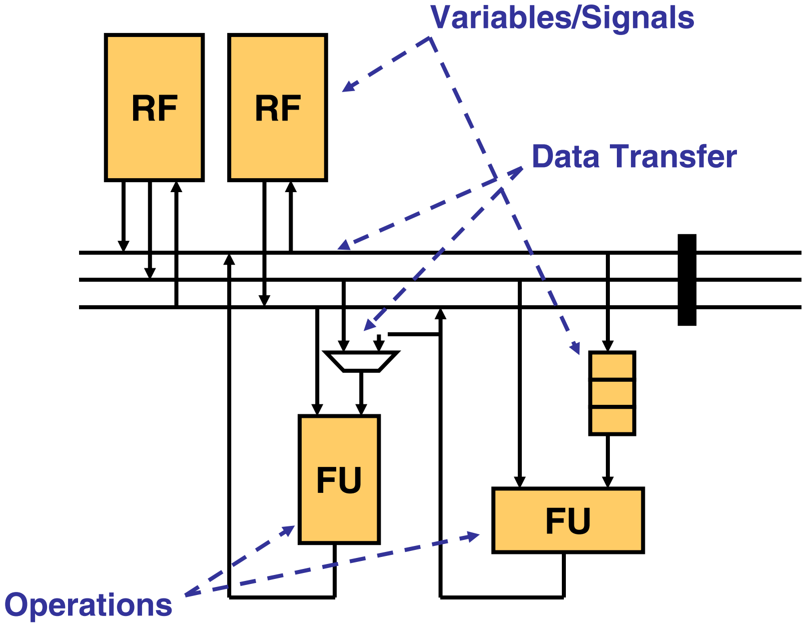


Operations → Functional Units

Variables Signals → Storage

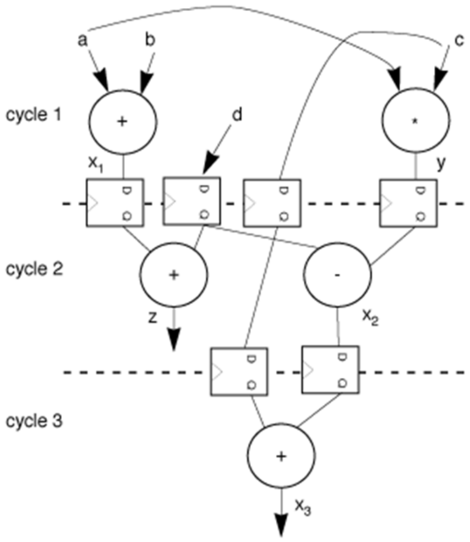
Data Transfers → Bus/Wire/Mux

Datapath



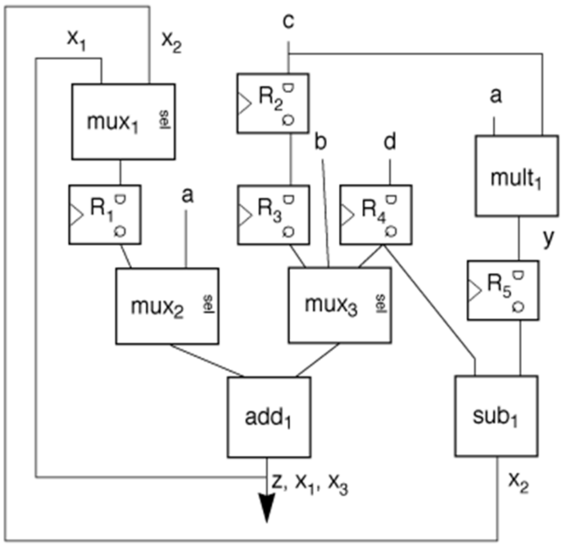
Binding Values to Registers

- **Registers fall on clock cycle boundaries**

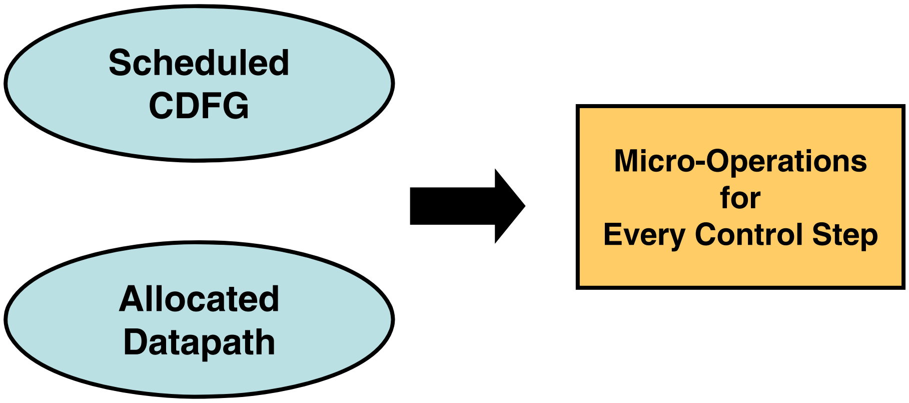


Binding Operations to Functional Units

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



Controller Generation



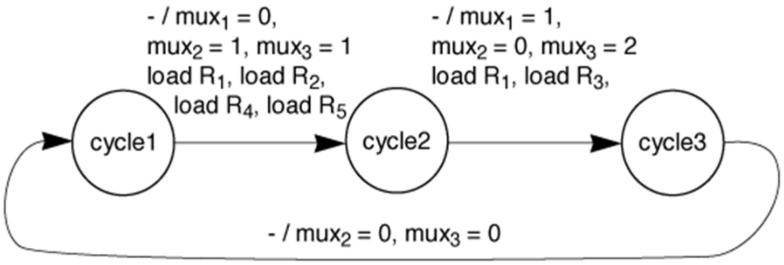
Scheduled CDFG

Allocated Datapath

Micro-Operations for Every Control Step

Building the Sequencer

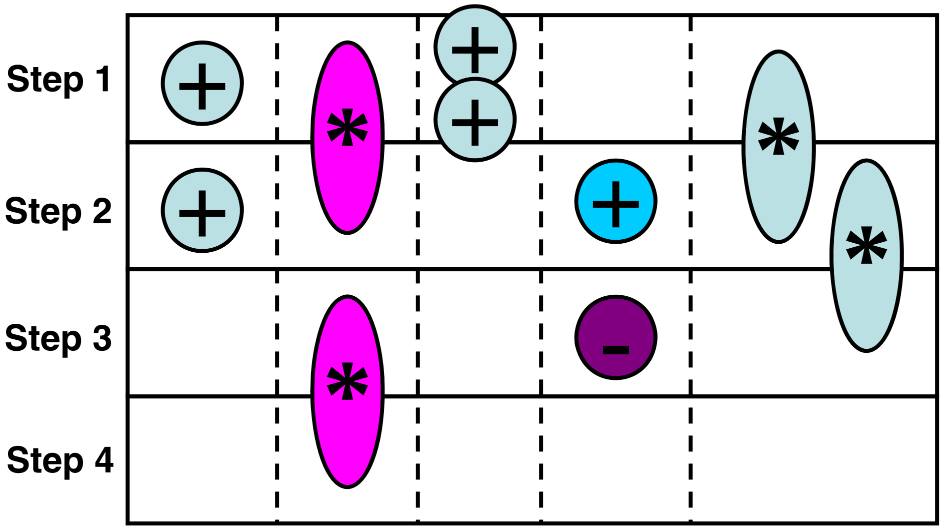
- **Finite state machine (FSM)**
  - Driving datapath control signals
  - Multiple states/cycles even without conditionals



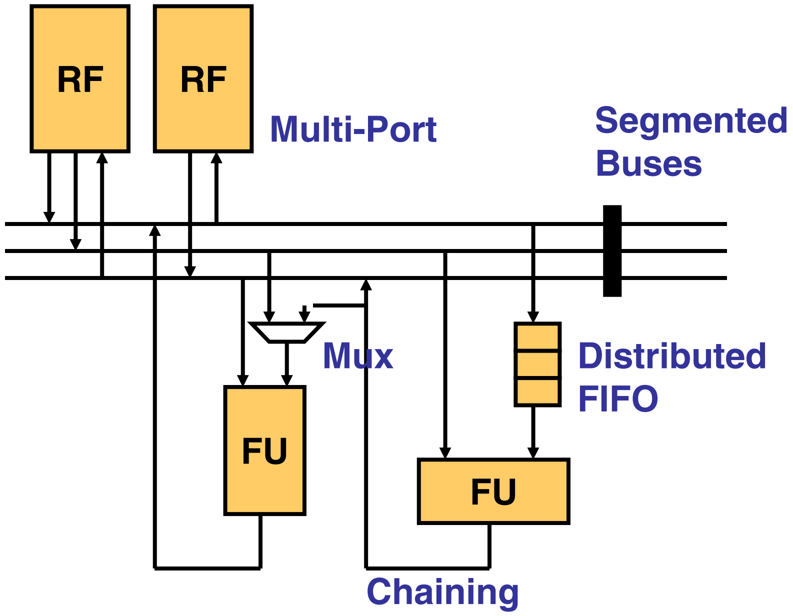
Hardware Variations

- **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

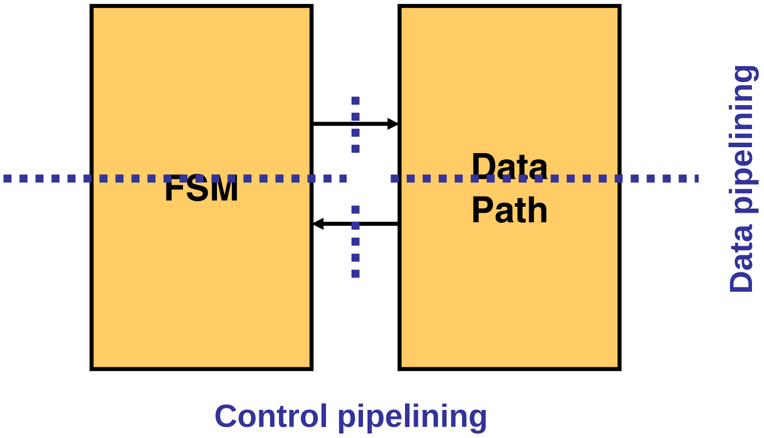
Functional Unit Variations



Storage/Interconnect Variations



Architectural Pipelining

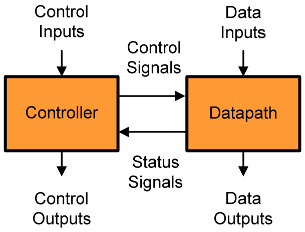


Lecture 10: Outline

- ✓ Introduction
- ✓ Essential issues
- **High-level synthesis flow**
  - RTL architectures & synthesis models
  - Resource- and time-constrained scheduling
  - Variable/operation/connection merging (storage/FU/bus resource sharing)
  - Chaining and multi-cycling
  - Data and control pipelining

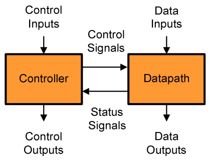
RTL Processor Architecture

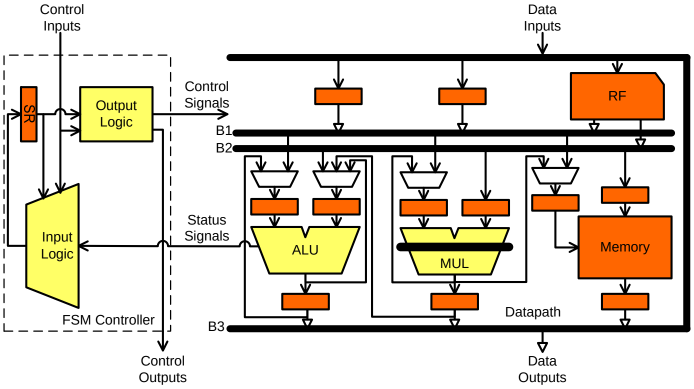
- **Controller**
  - FSM controller
  - Programmable controller
- **Datapath components**
  - Storage components
  - Functional units
  - Connection components
- **Pipelining**
  - Functional unit
  - Datapath
  - Control
- **Structure**
  - Chaining
  - Multicycling
  - Forwarding
  - Branch prediction
  - Caching



RTL Processor with FSM Controller

- **Simple architecture**
- **Small number of states**

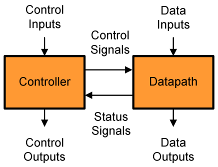


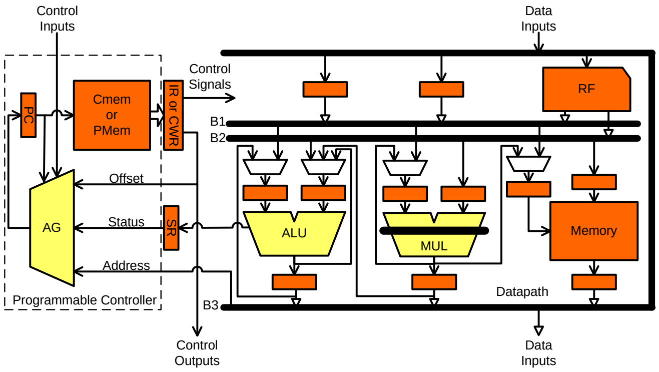


EE382M.20: SoC Design, Lecture 10 © 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner 29

RTL with Programmable Control

- **Complex architecture**
  - Control and datapath pipelining
  - Advanced structural features
- **Large number of states (CW or IS)**

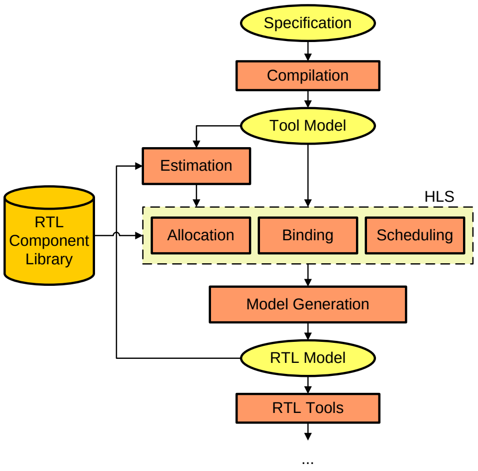




EE382M.20: SoC Design, Lecture 10 © 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner 30

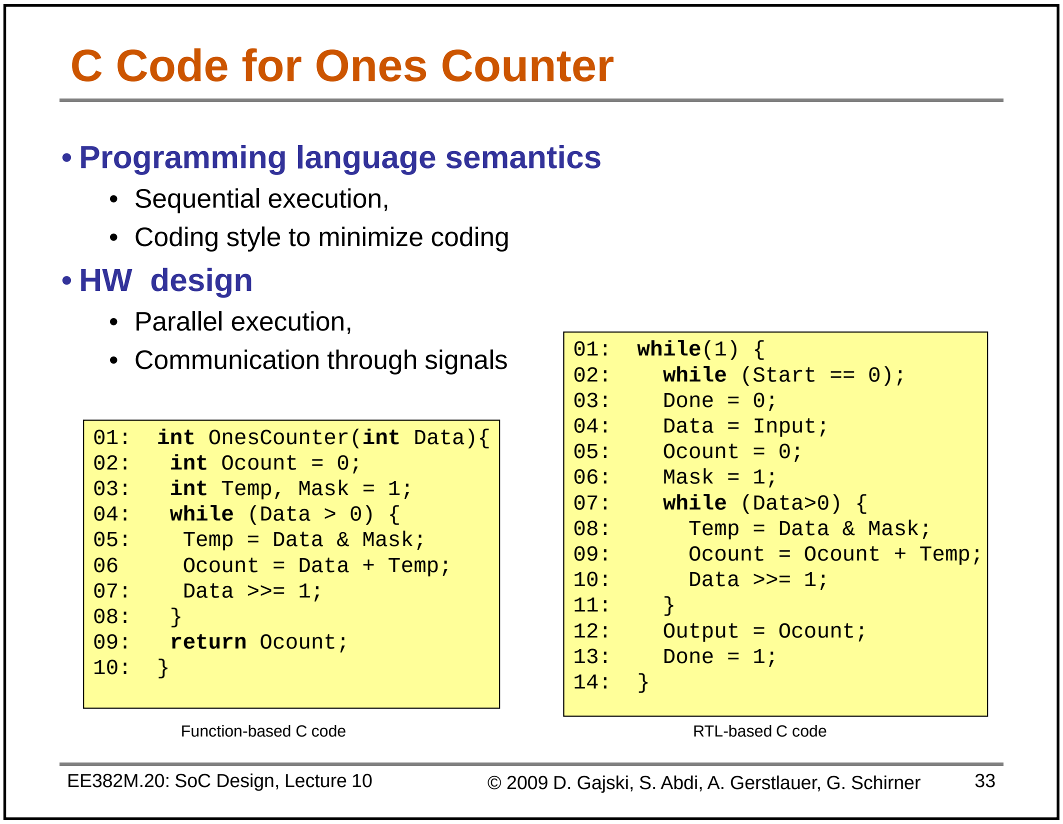
Hardware Synthesis Design Flow

- **Compilation**
- **Estimation**
- **HLS**
- **Model generation**
- **RTL synthesis**
- **Logic synthesis**
- **Layout**



Synthesis Models

1. **Input specification**
   - Programming language (C/C++, …)
     - Programming semantics requires pre-synthesis optimization
   - System description language (SystemC, …)
     - Simulation semantics requires pre-synthesis optimization
2. **Control/Data flow graph (CDFG)**
   - CDFG generation requires dependence analysis
3. **Finite state machine with data (FSMD)**
   - State interpretation requires some kind of scheduling
4. **RTL netlist**
   - RTL design that requires only input and output logic synthesis
5. **Hardware description language (Verilog / VHDL)**
   - HDL description requires RTL library and logic synthesis



C Code for Ones Counter

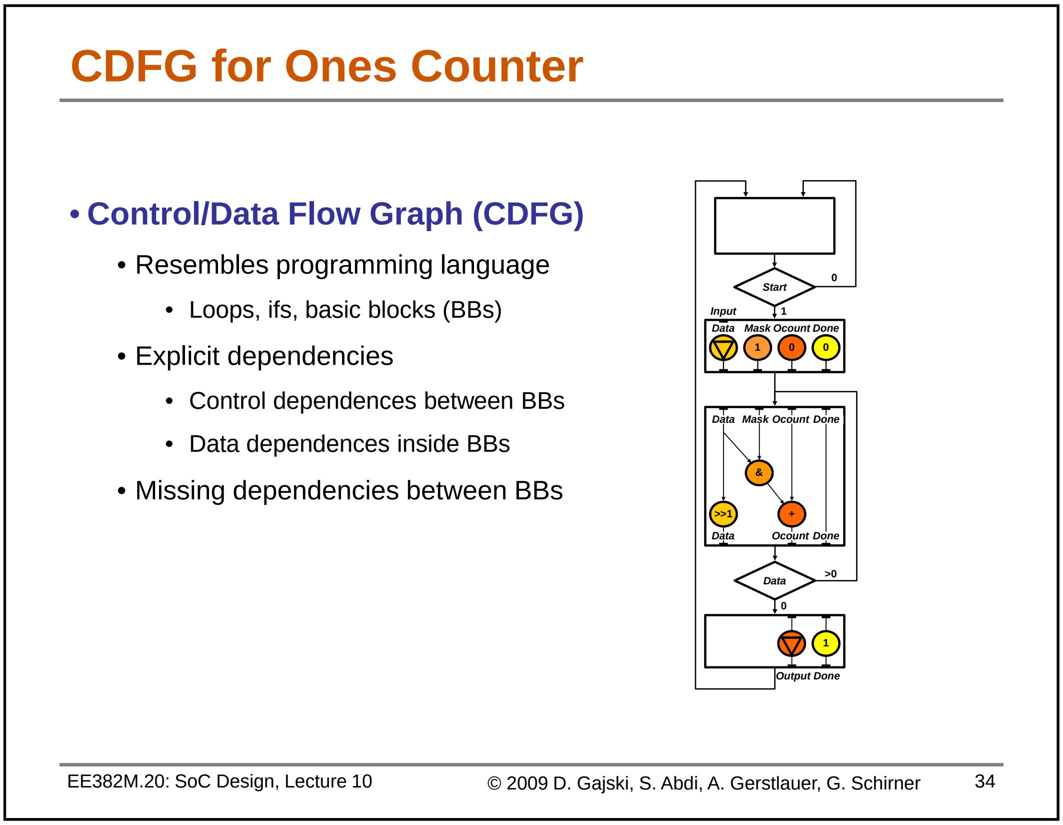
- **Programming language semantics**
  - Sequential execution,
  - Coding style to minimize coding
- **HW design**
  - Parallel execution,
  - Communication through signals

```
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

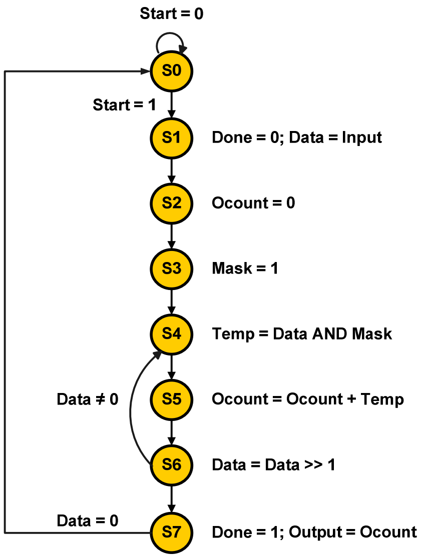


CDFG for Ones Counter

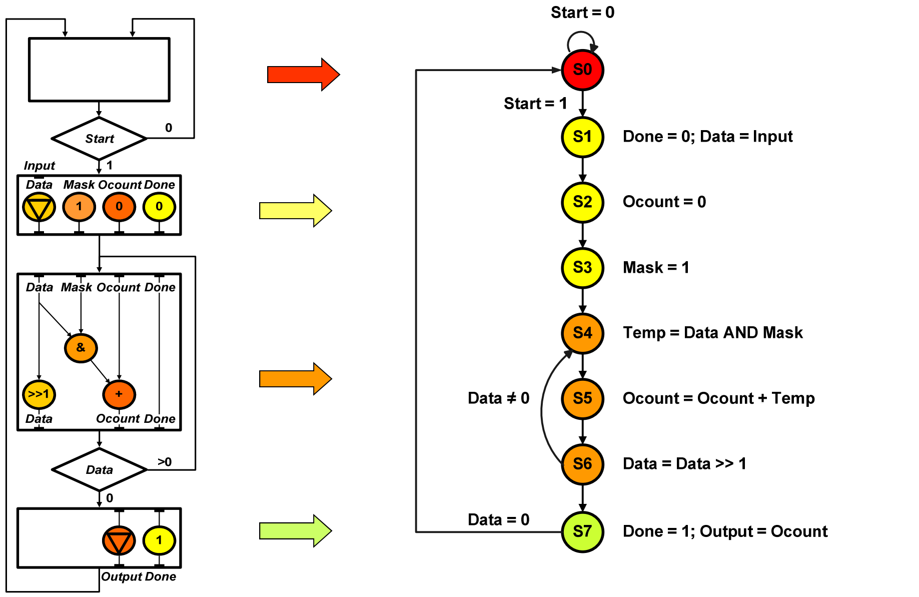
- **Control/Data Flow Graph (CDFG)**
  - Resembles programming language
    - Loops, ifs, basic blocks (BBs)
  - Explicit dependencies
    - Control dependences between BBs
    - Data dependences inside BBs
  - Missing dependencies between BBs

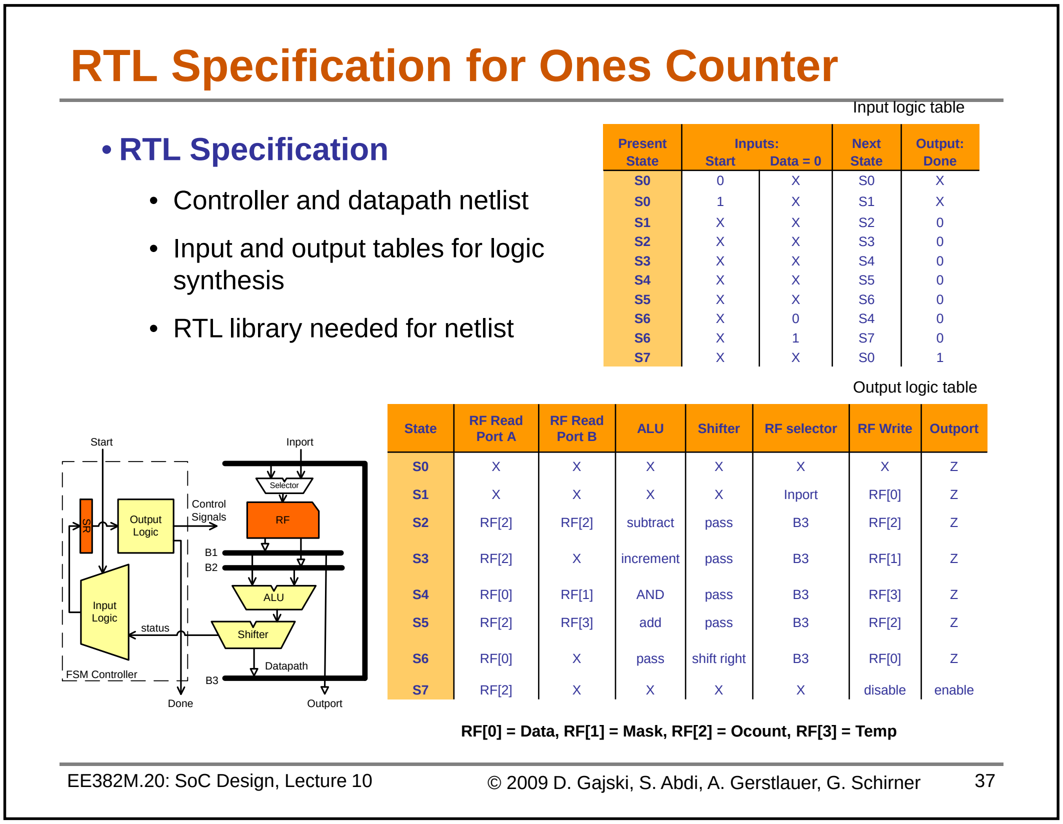
FSMD for Ones Counter

- **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**



CDFG and FSMD for Ones Counter





RTL Specification for Ones Counter

- **RTL Specification**
  - Controller and datapath netlist
  - Input and output tables for logic synthesis
  - RTL library needed for netlist

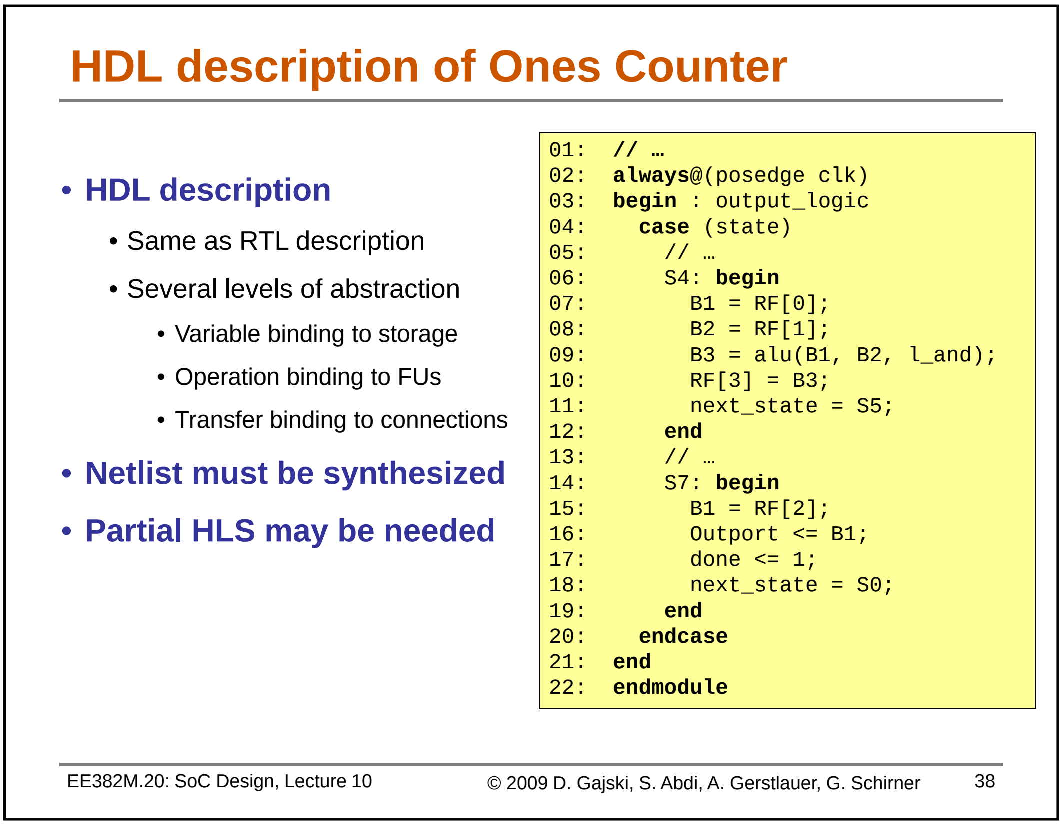
Input logic table

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

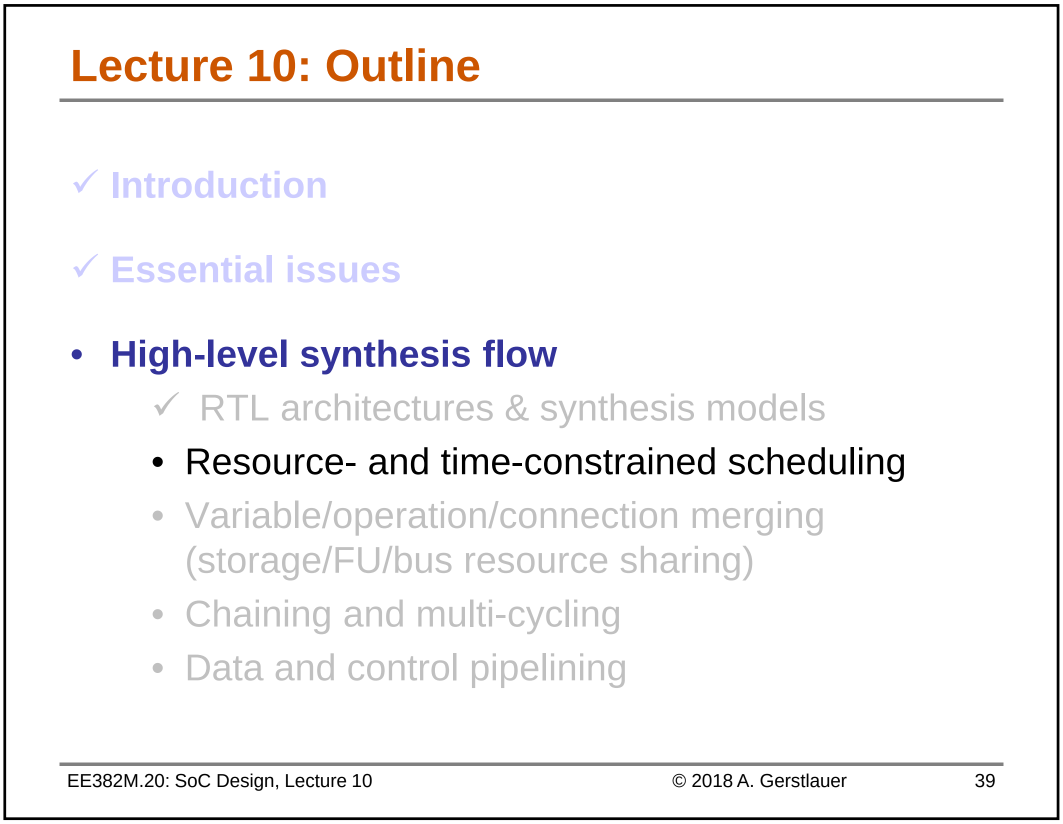
RF[0] = Data, RF[1] = Mask, RF[2] = Ocount, RF[3] = Temp



HDL description of Ones Counter

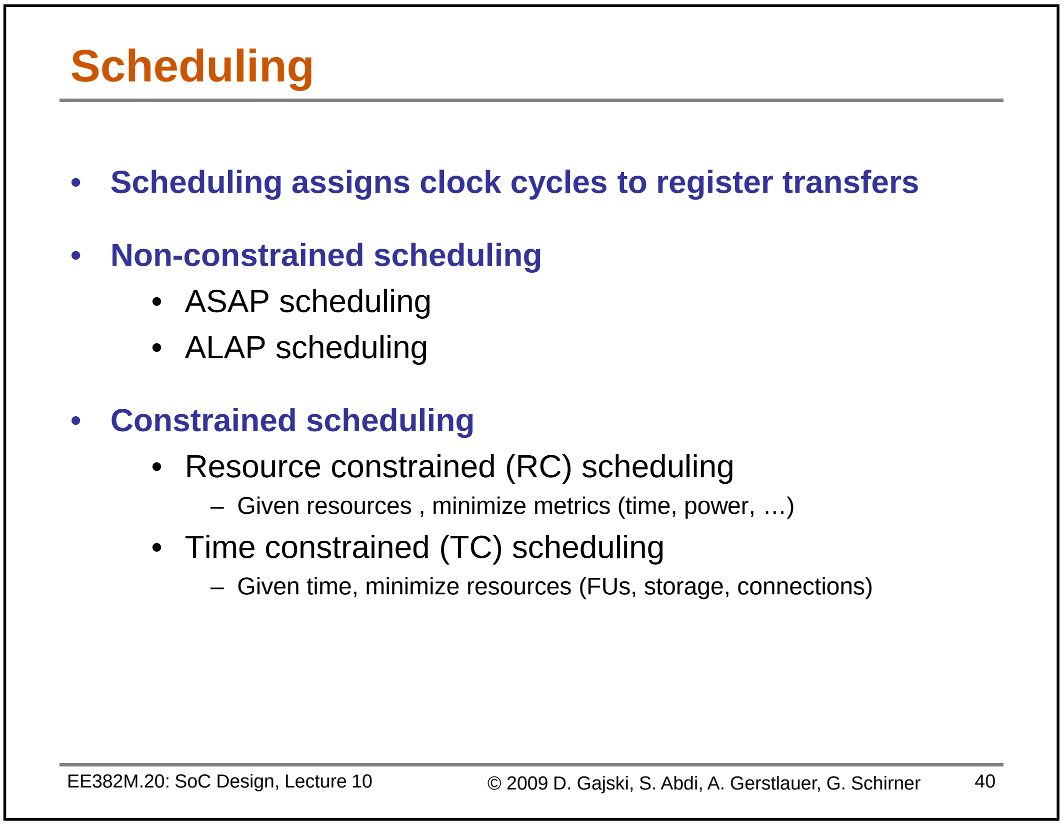
- **HDL description**
  - Same as RTL description
  - Several levels of abstraction
    - Variable binding to storage
    - Operation binding to FUs
    - Transfer binding to connections
- **Netlist must be synthesized**
- **Partial HLS may be needed**

```
// …
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
```



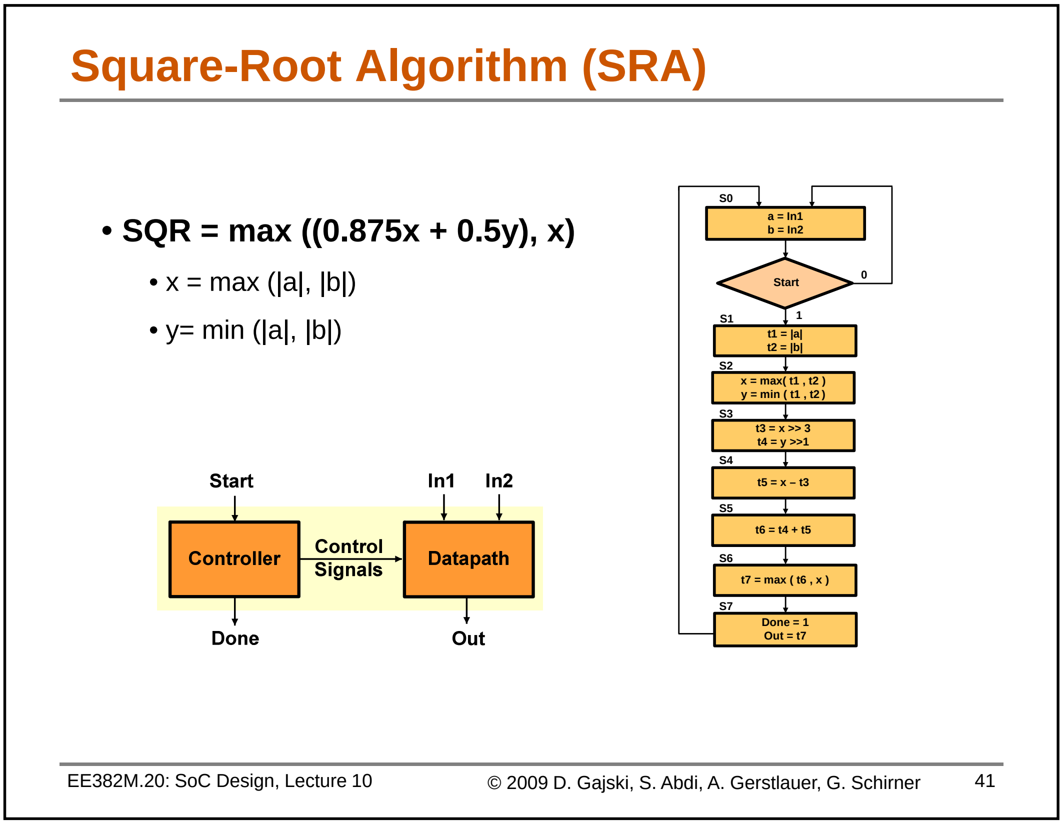
Lecture 10: Outline

- ✓ Introduction
- ✓ Essential issues
- High-level synthesis flow
  - ✓ RTL architectures & synthesis models
  - Resource- and time-constrained scheduling
  - Variable/operation/connection merging (storage/FU/bus resource sharing)
  - Chaining and multi-cycling
  - Data and control pipelining



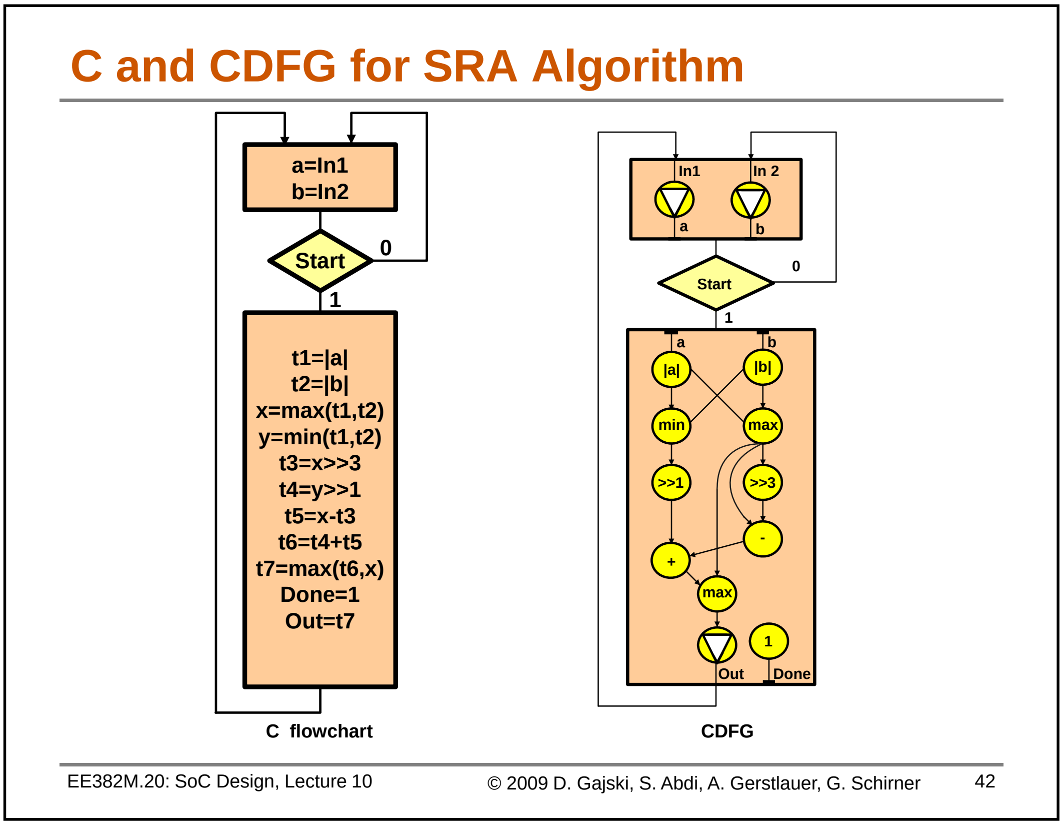
Scheduling

- **Scheduling assigns clock cycles to register transfers**
- **Non-constrained scheduling**
  - ASAP scheduling
  - ALAP scheduling
- **Constrained scheduling**
  - Resource constrained (RC) scheduling
    - Given resources , minimize metrics (time, power, …)
  - Time constrained (TC) scheduling
    - Given time, minimize resources (FUs, storage, connections)



Square-Root Algorithm (SRA)

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

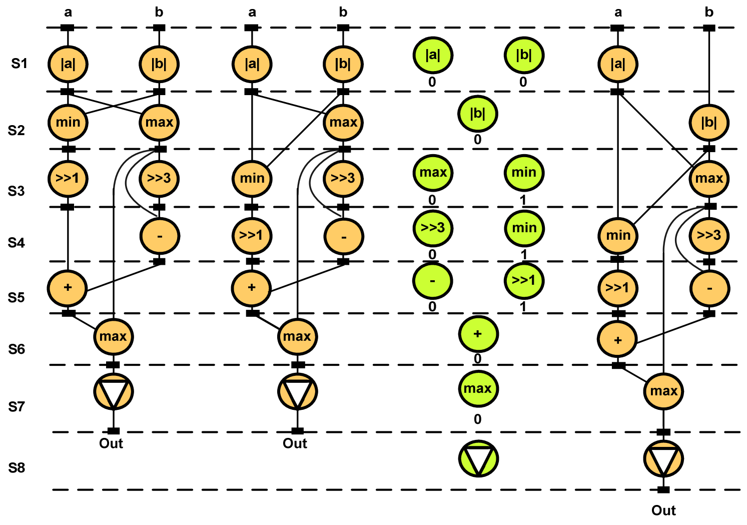


C and CDFG for SRA Algorithm

C flowchart

CDFG

RC Scheduling



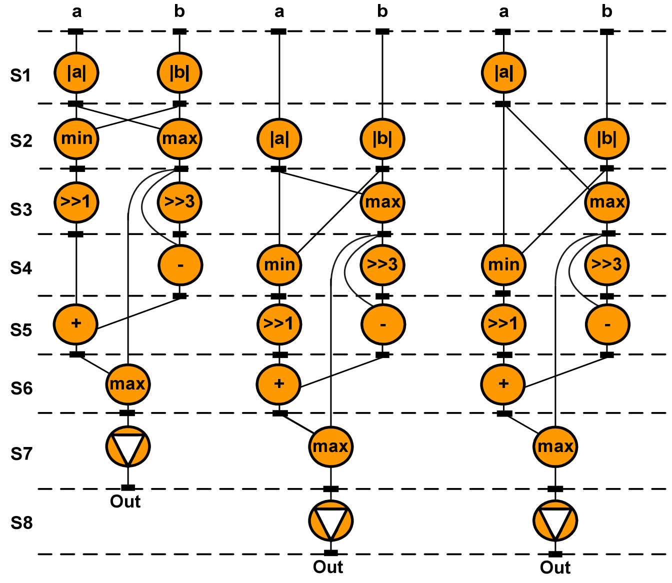
ASAP schedule

ALAP schedule

Ready list with mobilities (ALAP – ASAP)

RC schedule (for single FU and 2 shifters)

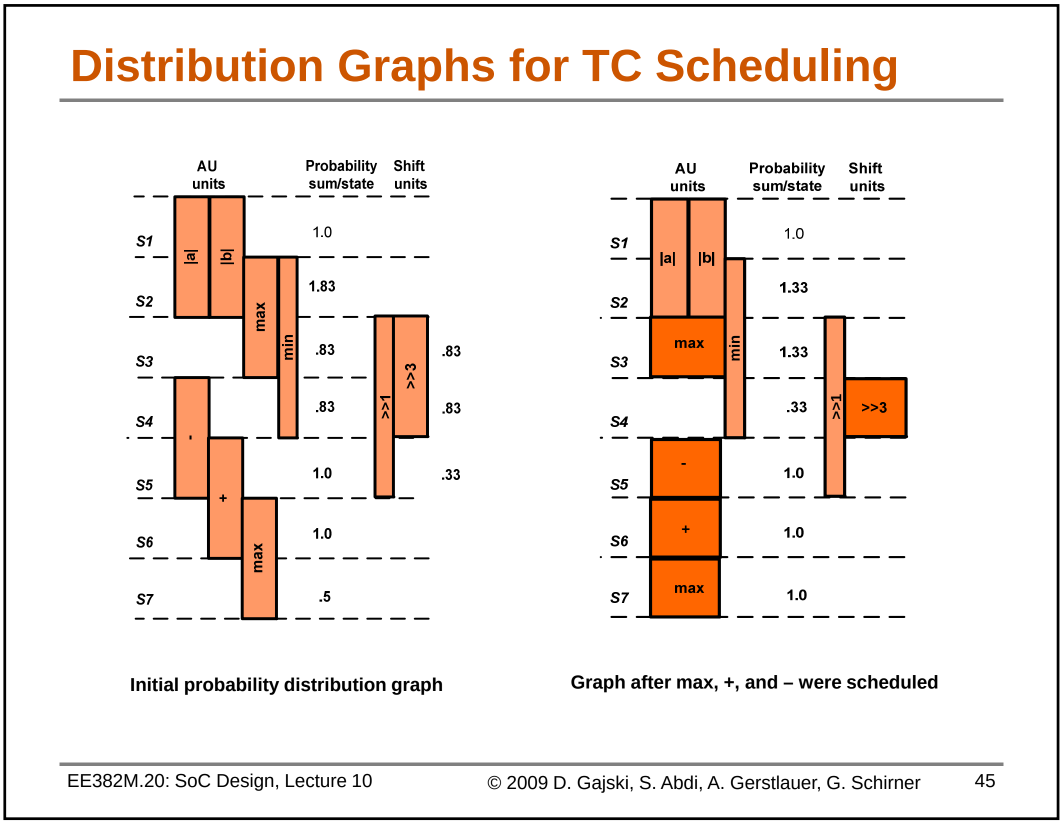
TC Scheduling



ASAP

ALAP

TC schedule

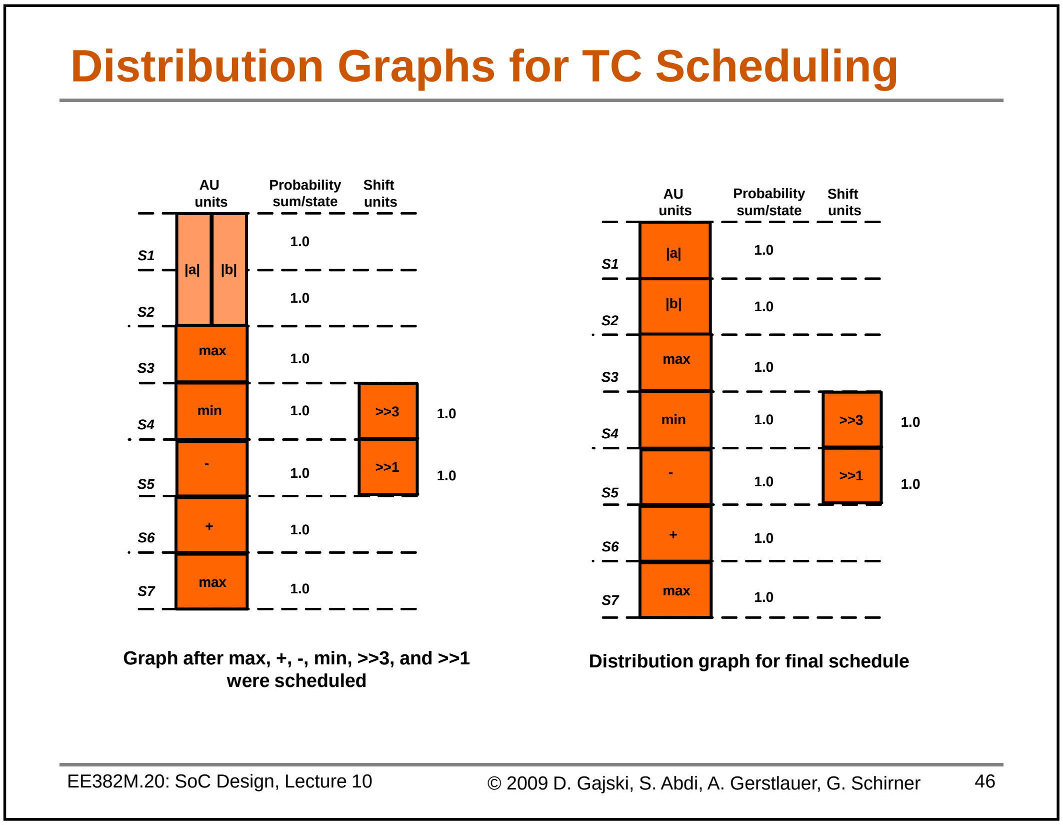


Distribution Graphs for TC Scheduling

Initial probability distribution graph

Graph after max, +, and – were scheduled

EE382M.20: SoC Design, Lecture 10 © 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner 45



Distribution Graphs for TC Scheduling

Graph after max, +, -, min, >>3, and >>1 were scheduled

Distribution graph for final schedule

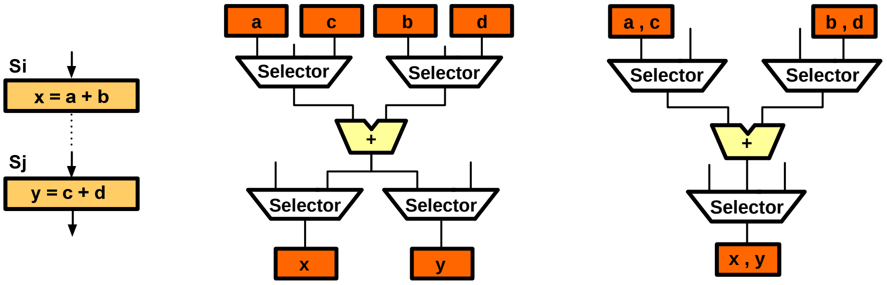
EE382M.20: SoC Design, Lecture 10 © 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner 46

Lecture 10: Outline

- ✓ Introduction
- ✓ Essential issues
- High-level synthesis flow
  - ✓ RTL architectures & synthesis models
  - ✓ Resource- and time-constrained scheduling
  - Variable/operation/connection merging (storage/FU/bus resource sharing)
  - Chaining and multi-cycling
  - Data and control pipelining

Register Sharing

- **Register sharing**
  - Grouping variables with non-overlapping lifetimes
  - Sharing reduces connectivity cost



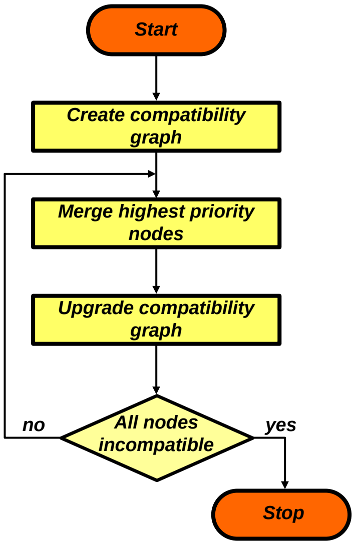
Partial FSMD

Datapath without register sharing

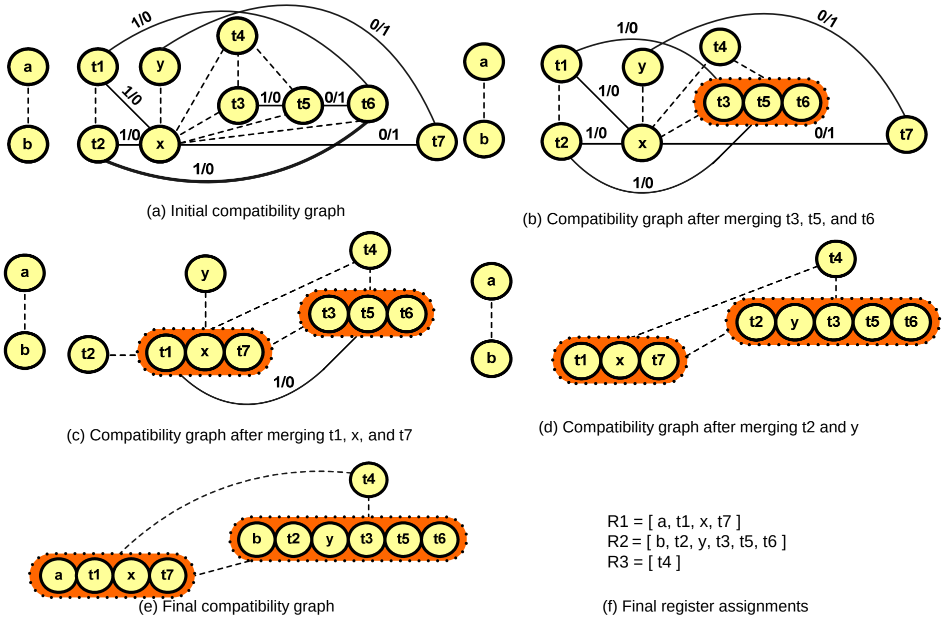
Datapath with register sharing

General Partitioning Algorithm

- **Compatibility graph**
  - Compatibility:
    - Non-overlapping in time
    - Not using the same resource
  - Non-compatible:
    - Overlapping in time
    - Using the same resource
- **Priority**
  - Critical path
  - Same source, same destination


Start
Create compatibility graph
Merge highest priority nodes
Upgrade compatibility graph
All nodes incompatible
no
yes
Stop

Variable Merging for SRA



(a) Initial compatibility graph

(b) Compatibility graph after merging t3, t5, and t6

(c) Compatibility graph after merging t1, x, and t7

(d) Compatibility graph after merging t2 and y

(e) Final compatibility graph

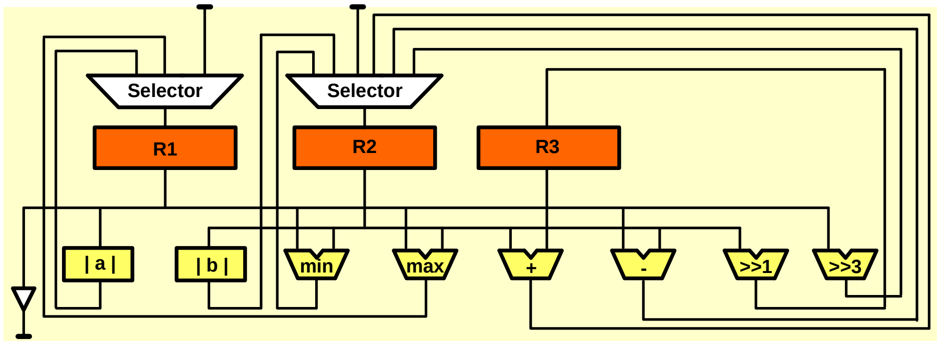
R1 = [a, t1, x, t7]
R2 = [b, t2, y, t3, t5, t6]
R3 = [t4]

(f) Final register assignments

Datapath with Shared Registers

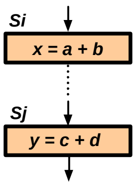
- **Variables combined into registers**
- **One functional unit for each operation**

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

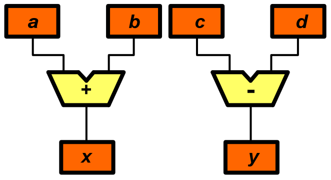


Functional Unit Sharing

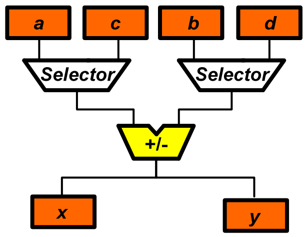
- **Functional unit sharing**
  - Smaller number of FUs
  - Larger connectivity cost



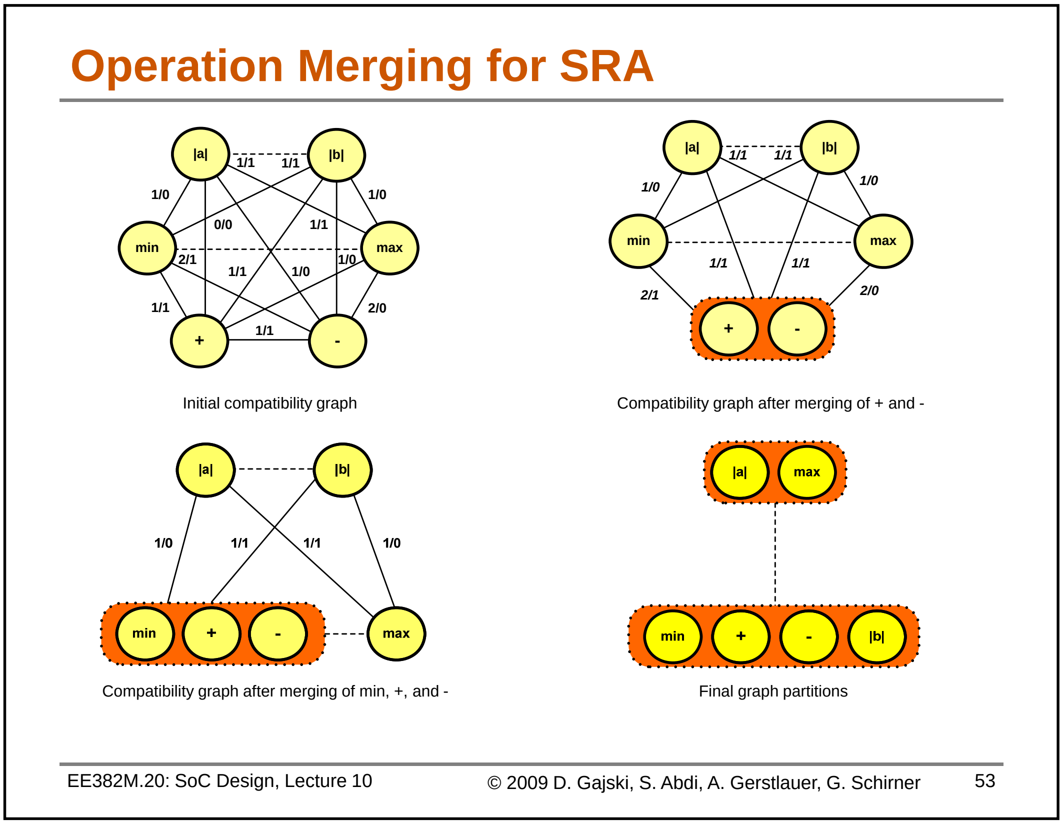
Partial FSMD



Non-shared design



Shared design



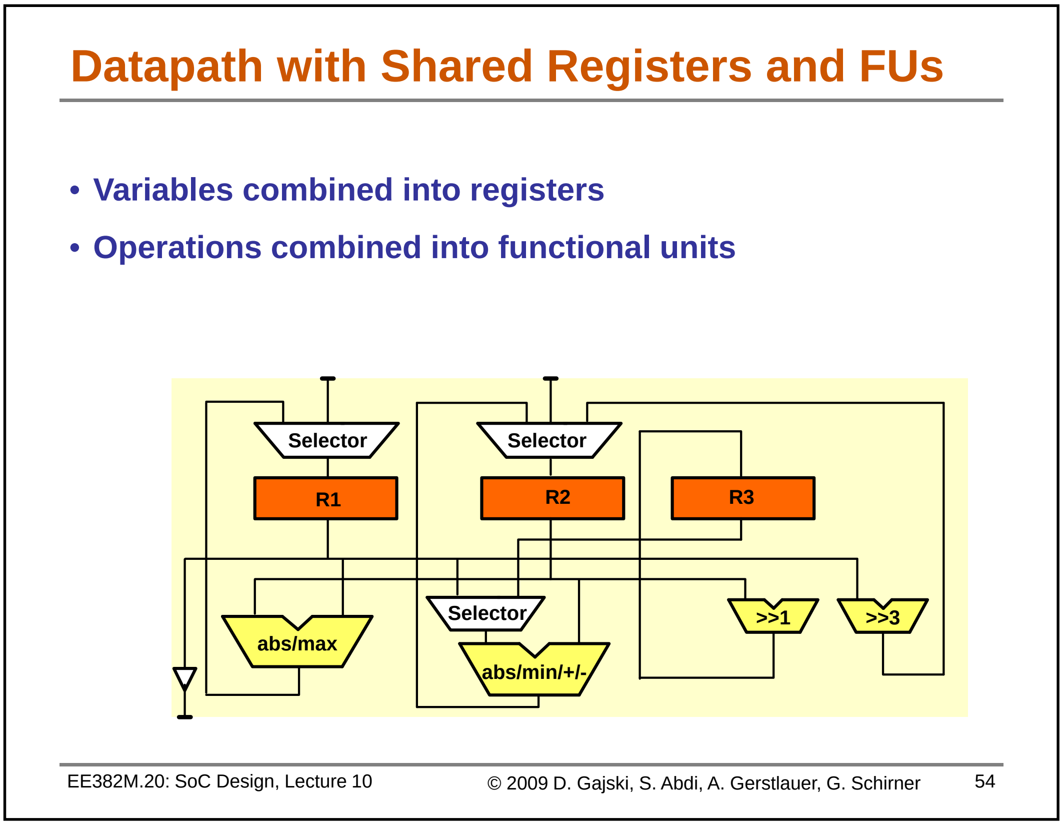
Operation Merging for SRA

Initial compatibility graph

Compatibility graph after merging of + and -

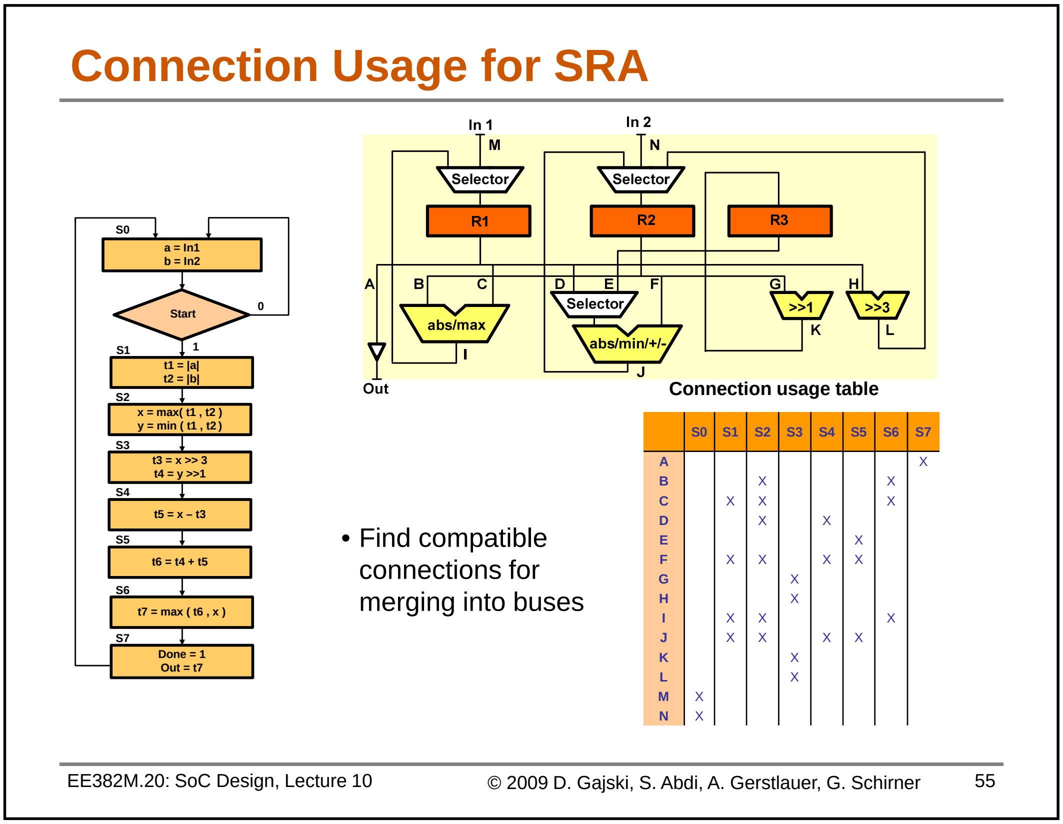
Compatibility graph after merging of min, +, and -

Final graph partitions



Datapath with Shared Registers and FUs

- **Variables combined into registers**
- **Operations combined into functional units**

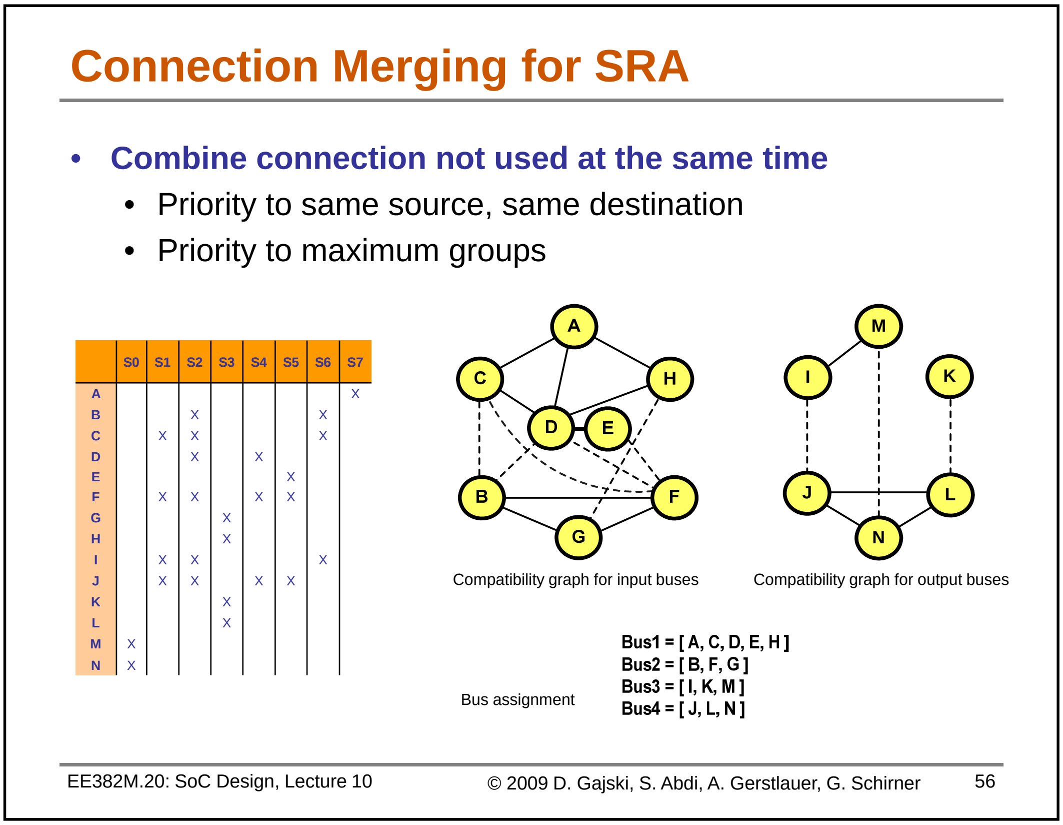


Connection Usage for SRA

- Find compatible connections for merging into buses

Connection usage table

	S0	S1	S2	S3	S4	S5	S6	S7
A								X
B			X				X	
C		X	X				X	
D			X		X			
E						X		
F		X	X		X	X		
G				X				
H				X				
I		X	X				X	
J		X	X		X	X		
K				X				
L				X				
M	X							
N	X							



Connection Merging for SRA

- **Combine connection not used at the same time**
  - Priority to same source, same destination
  - Priority to maximum groups

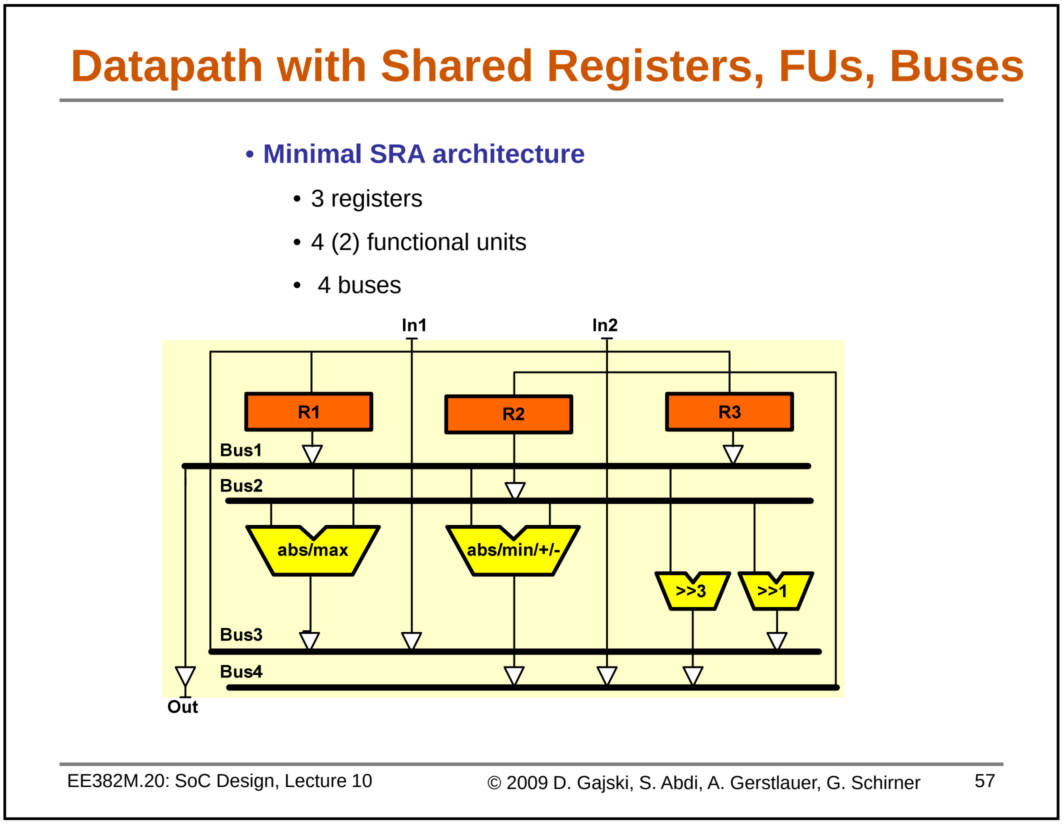
	S0	S1	S2	S3	S4	S5	S6	S7
A								X
B			X				X	
C		X	X				X	
D			X		X			
E						X		
F		X	X		X	X		
G				X				
H				X				
I		X	X				X	
J		X	X		X	X		
K				X				
L				X				
M	X							
N	X							

Compatibility graph for input buses

Compatibility graph for output buses

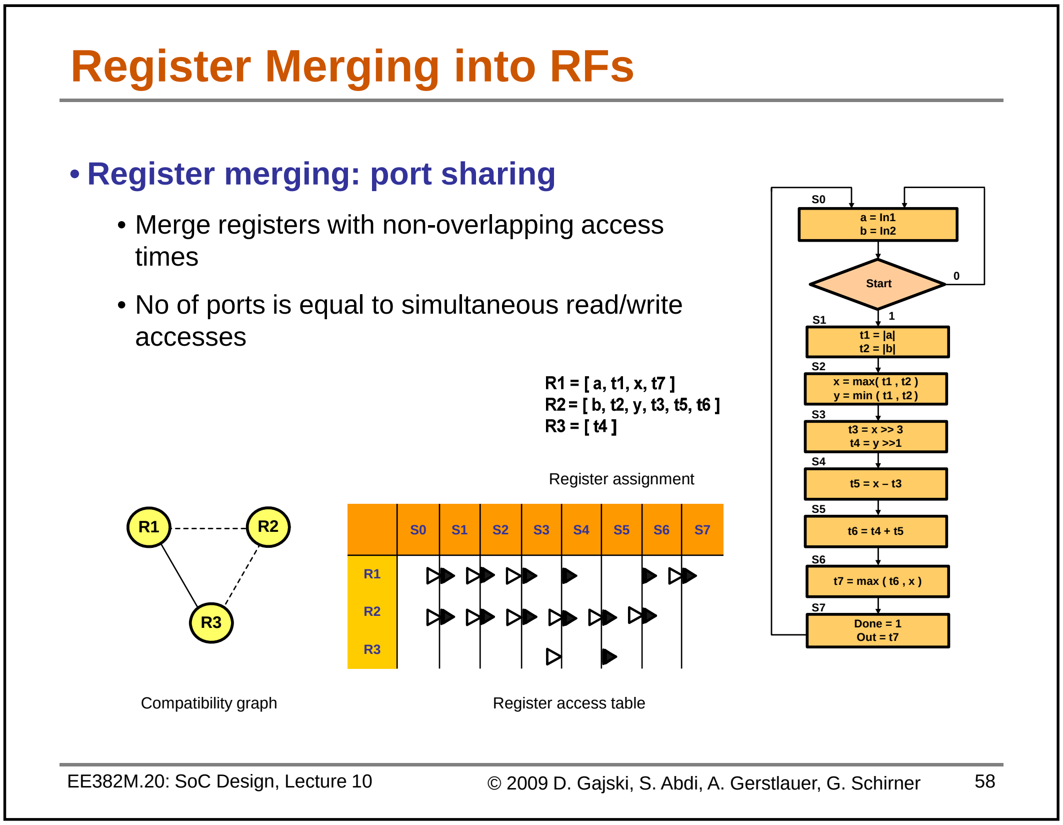
Bus assignment

Bus1 = [A, C, D, E, H]
Bus2 = [B, F, G]
Bus3 = [I, K, M]
Bus4 = [J, L, N]



Datapath with Shared Registers, FUs, Buses

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



Register Merging into RFs

- **Register merging: port sharing**
  - Merge registers with non-overlapping access times
  - No of ports is equal to simultaneous read/write accesses

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

Register assignment

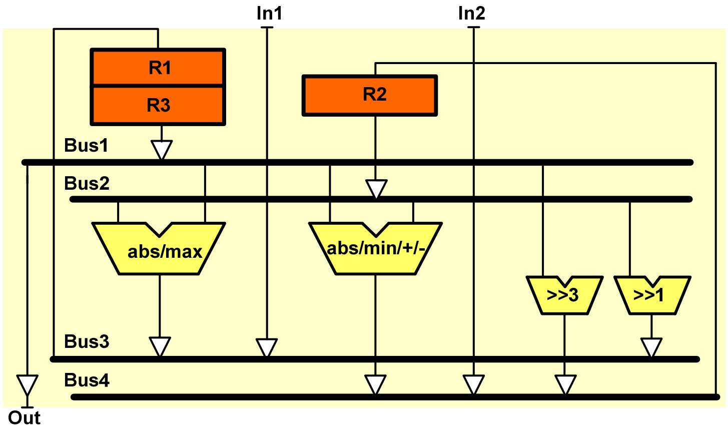
Compatibility graph

	S0	S1	S2	S3	S4	S5	S6	S7
R1	▷	▶ ▷	▶ ▷	▶	▶		▶ ▷	▶
R2	▷	▶ ▷	▶ ▷	▶ ▷	▶ ▷	▶ ▷	▶	
R3				▷		▶		

Register access table

Datapath with Shared RF

- RF minimize connectivity cost by sharing ports

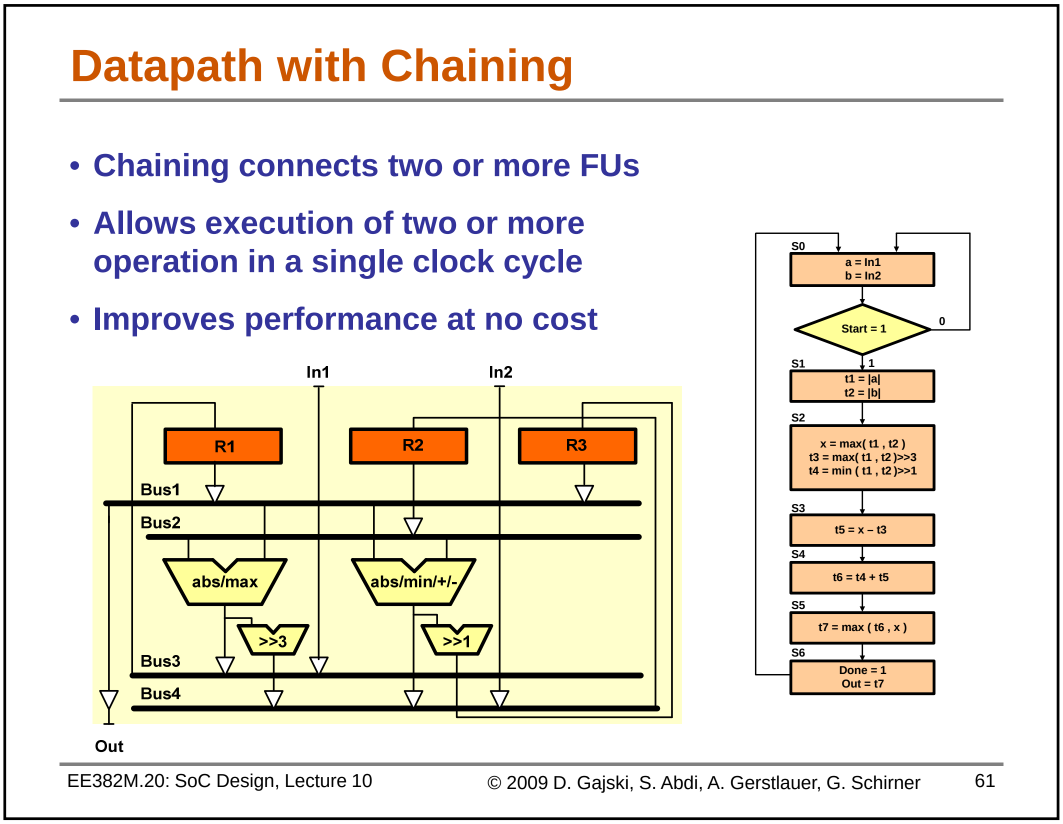


Lecture 10: Outline

✓ Introduction

✓ Essential issues

- High-level synthesis flow
  - ✓ RTL architectures & synthesis models
  - ✓ Resource- and time-constrained scheduling
  - ✓ Variable/operation/connection merging (storage/FU/bus resource sharing)
  - Chaining and multi-cycling
  - Data and control pipelining



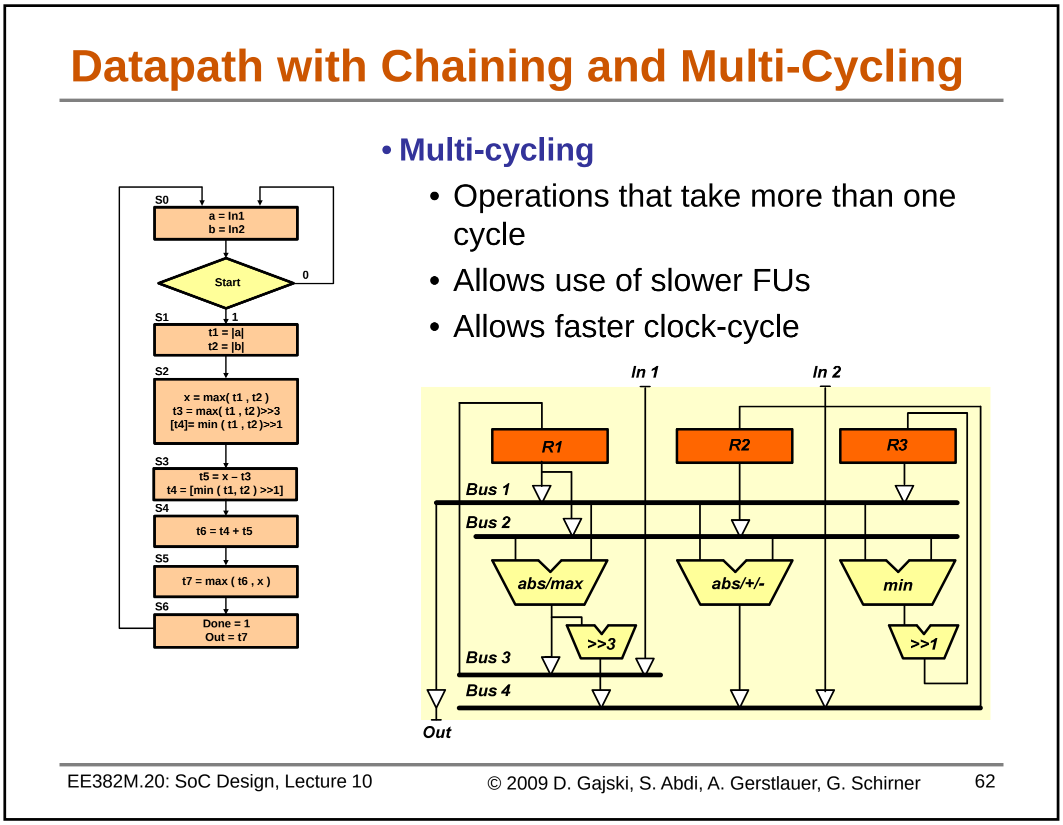
Datapath with Chaining

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

In1
In2
R1
R2
R3
Bus1
Bus2
abs/max
abs/min/+/-
>>3
>>1
Bus3
Bus4
Out

S0
a = In1
b = In2
Start = 1
0
S1
1
t1 = |a|
t2 = |b|
S2
x = max(t1 , t2)
t3 = max(t1 , t2)>>3
t4 = min (t1 , t2)>>1
S3
t5 = x – t3
S4
t6 = t4 + t5
S5
t7 = max (t6 , x)
S6
Done = 1
Out = t7

EE382M.20: SoC Design, Lecture 10 © 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner 61



Datapath with Chaining and Multi-Cycling

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

S0
a = In1
b = In2
Start
0
S1
1
t1 = |a|
t2 = |b|
S2
x = max(t1 , t2)
t3 = max(t1 , t2)>>3
[t4]= min (t1 , t2)>>1
S3
t5 = x – t3
t4 = [min (t1, t2) >>1]
S4
t6 = t4 + t5
S5
t7 = max (t6 , x)
S6
Done = 1
Out = t7

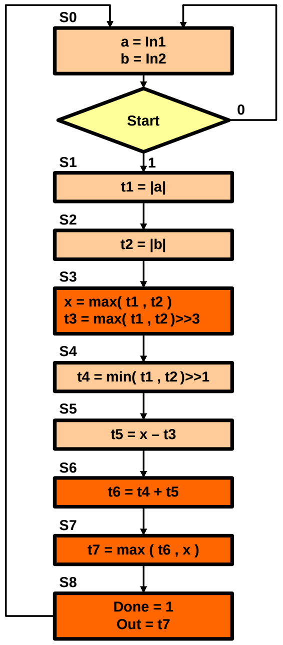
In 1
In 2
R1
R2
R3
Bus 1
Bus 2
abs/max
abs/+/-
min
>>3
>>1
Bus 3
Bus 4
Out

EE382M.20: SoC Design, Lecture 10 © 2009 D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner 62

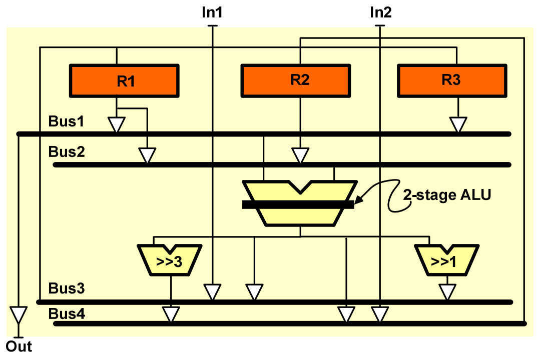
Pipelining

- **Functional Unit pipelining**
  - Two or more operation executing at the same time
- **Datapath pipelining**
  - Two or more register transfers executing at the same time
- **Control Pipelining**
  - Two or more instructions generated at the same time

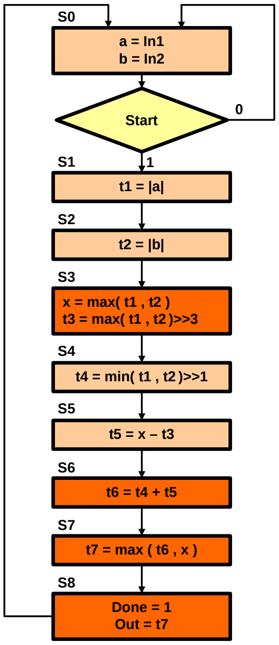
Functional Unit Pipelining (1)

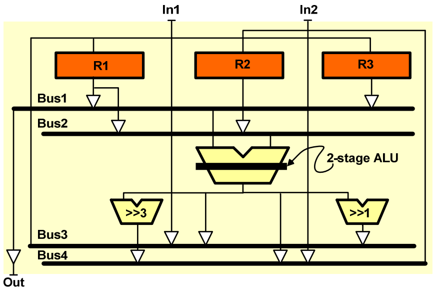


- Operation delay cut in "half"
- Shorter clock cycle
- Dependencies may delay some states
- Extra NO states reduce performance gain



Functional Unit Pipelining (2)

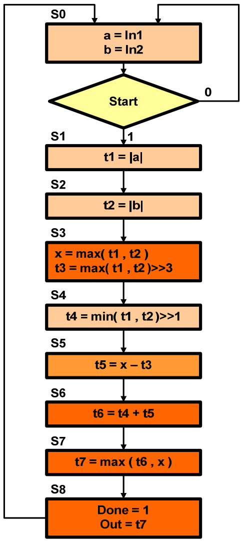




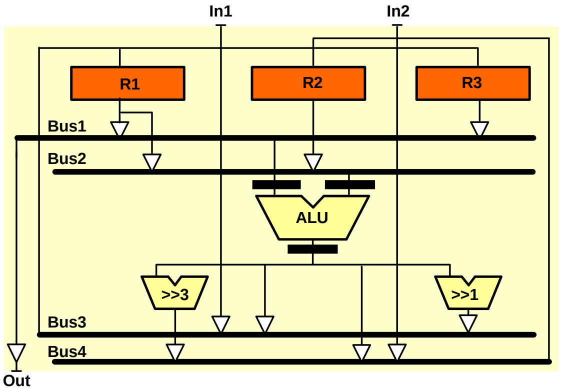
Timing diagram with 4 additional NOP states

	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

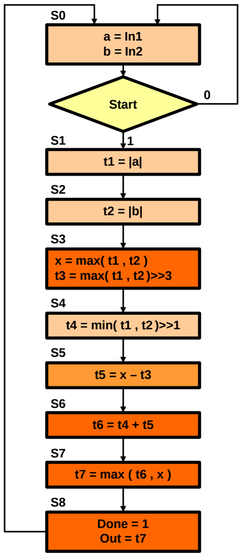
Datapath Pipelining (1)

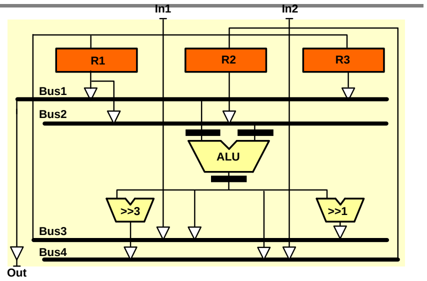


- Register-to-register delay cut in “equal” parts
- Much shorter clock cycle
- Dependencies may delay some states
- Extra NOP states reduce performance gain



Datapath Pipelining (2)



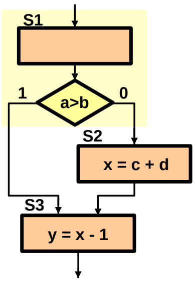


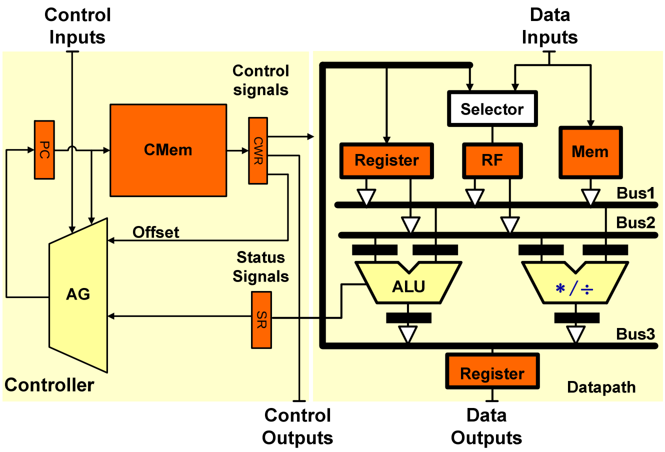
Timing diagram with additional NOP clock cycles

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)

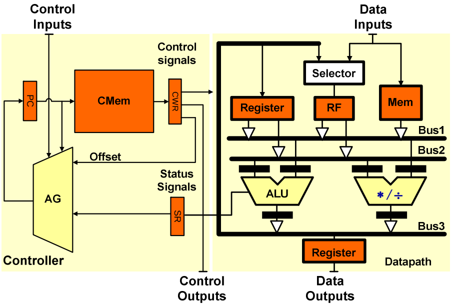
- Fetch delay cut into several parts
- Shorter clock cycle
- Conditionals may delay some states
- Extra NOP states reduce performance gain

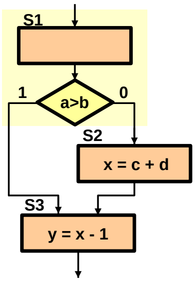




Data and Control Pipelining (2)

- 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 10: Summary

- **Synthesis techniques (binding)**
  - Variable merging (storage sharing)
  - Operation merging (FU sharing)
  - Connection merging (bus sharing)
- **Scheduling**
  - Metric constrained scheduling
- **Architecture techniques**
  - Chaining and Multi-Cycling
  - Data and Control Pipelining
  - Forwarding and Caching

➢ **Design automation (algorithms)**
  - Formalization / problem definition
  - Scheduling and binding combined?
    - If too complex, use partial order