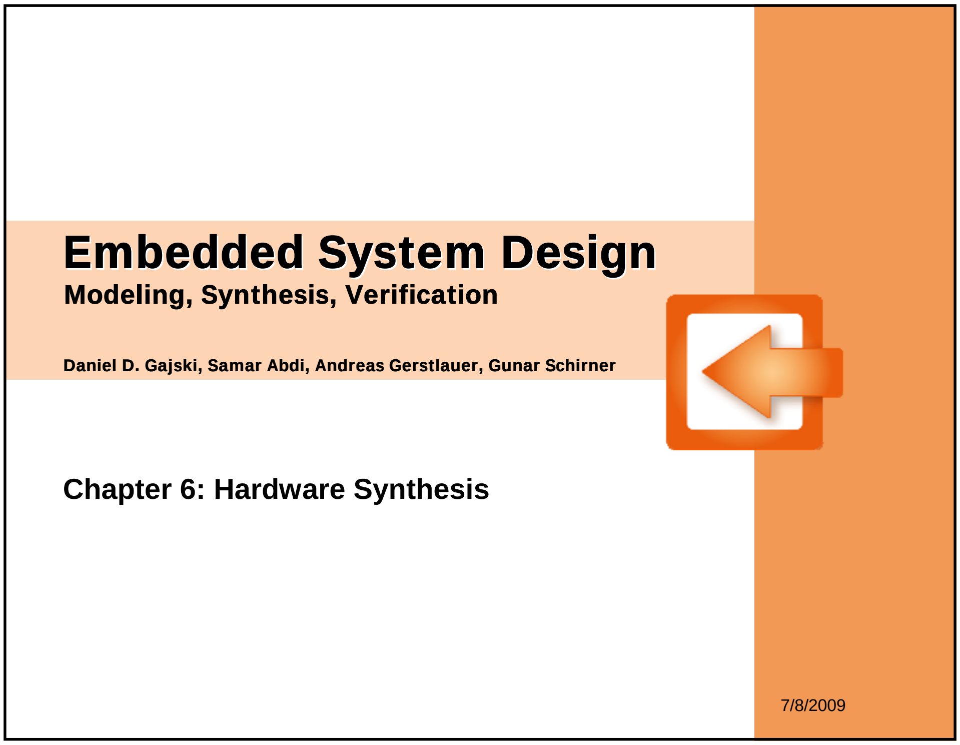


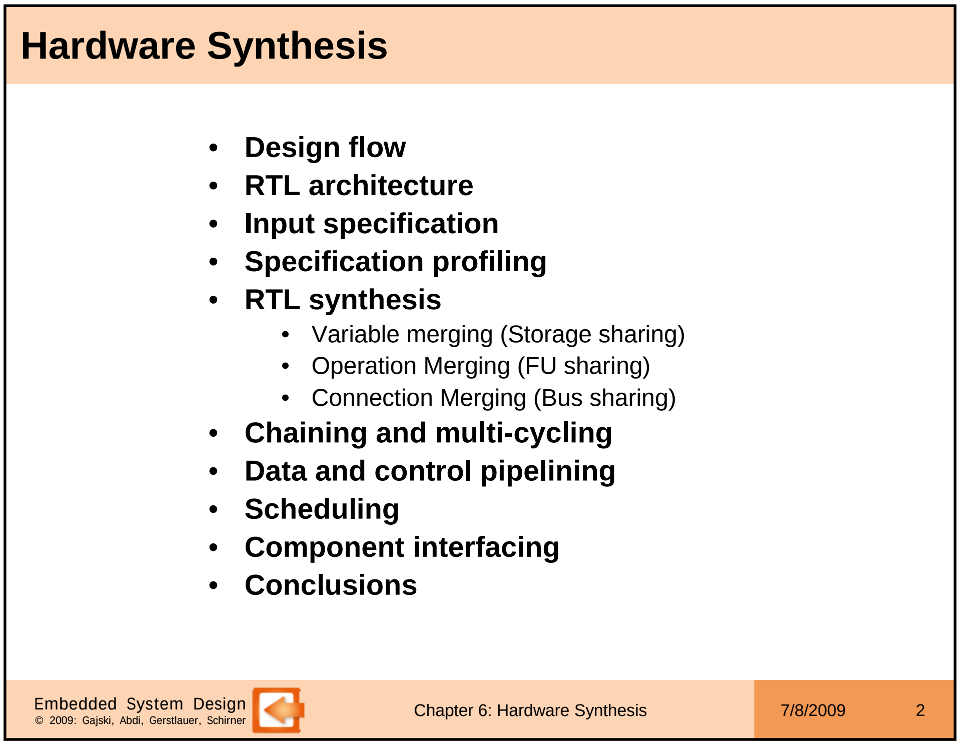
Embedded System Design

Modeling, Synthesis, Verification

Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, Gunar Schirner

Chapter 6: Hardware Synthesis

7/8/2009

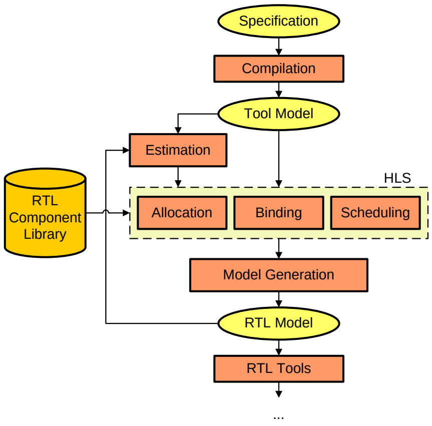


Hardware Synthesis

- Design flow
- RTL architecture
- Input specification
- Specification profiling
- RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- Chaining and multi-cycling
- Data and control pipelining
- Scheduling
- Component interfacing
- Conclusions

HW Synthesis Design Flow

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

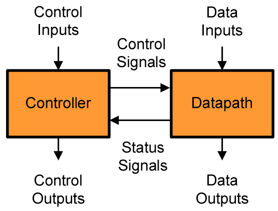


Hardware Synthesis

- ✓ Design flow
- RTL architecture
- Input specification
- Specification profiling
- RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- Chaining and multi-cycling
- Data and control pipelining
- Scheduling
- Component interfacing
- Conclusions

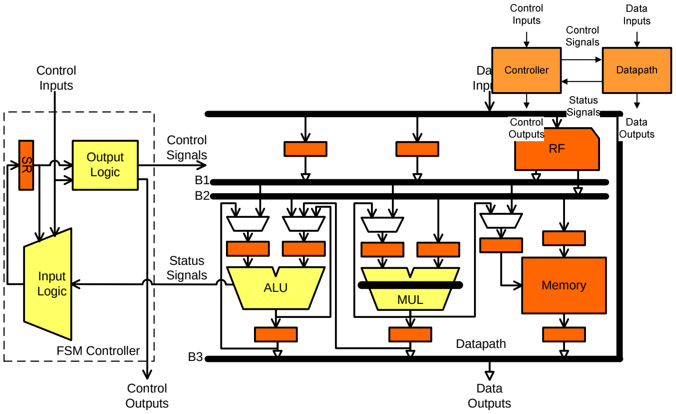
RTL Architecture

- **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



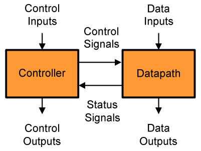
RTL Architecture with FSM Controller

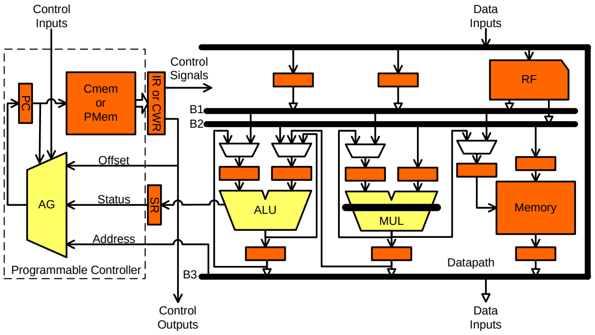
- **Simple architecture**
- **Small number of states**



RTL Architecture with Programmable Controller

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





Hardware Synthesis

- ✓ Design flow
- ✓ RTL architecture
- **Input specification**
- Specification profiling
- RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- Chaining and multi-cycling
- Data and control pipelining
- Scheduling
- Component interfacing
- Conclusions

Input Specification

- **Programming language (C/C++, …)**
  - Programming semantics requires pre-synthesis optimization
- **System description language (SystemC, …)**
  - Simulation semantics requires pre-synthesis optimization
- **Control/Data flow graph (CDFG)**
  - CDFG generation requires dependence analysis
- **Finite state machine with data (FSMD)**
  - State interpretation requires some kind of scheduling
- **RTL netlist**
  - RTL design that requires only input and output logic synthesis
- **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;
}
```

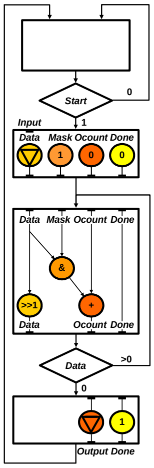
Function-based C code

```
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

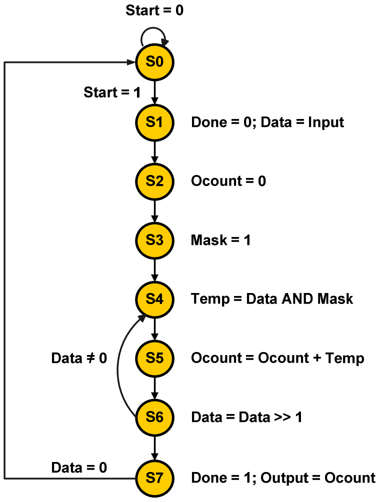
CDFG for Ones Counter

- **Control/Data flow graph**
  - 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 then 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**



Start = 0

S0

Start = 1

S1 Done = 0; Data = Input

S2 Ocount = 0

S3 Mask = 1

S4 Temp = Data AND Mask

Data ≠ 0

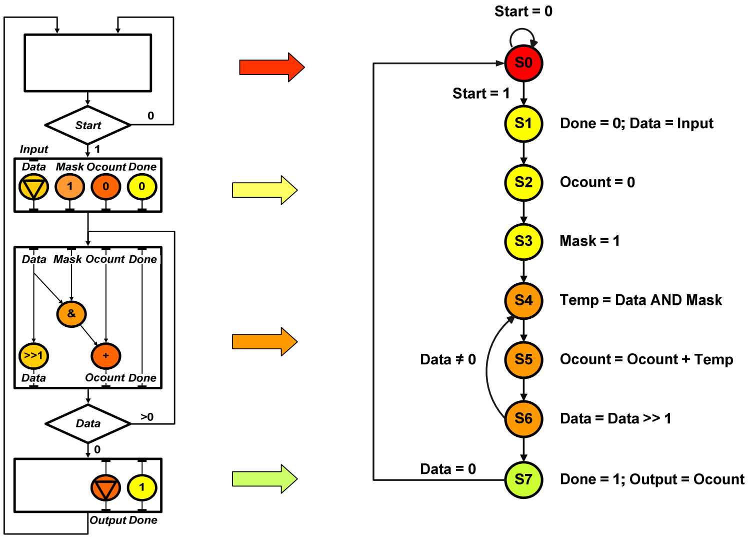
S5 Ocount = Ocount + Temp

S6 Data = Data >> 1

Data = 0

S7 Done = 1; Output = Ocount

CDFG and FSMD for Ones Counter



Start = 0

S0

Start = 1

S1 Done = 0; Data = Input

S2 Ocount = 0

S3 Mask = 1

S4 Temp = Data AND Mask

Data ≠ 0

S5 Ocount = Ocount + Temp

S6 Data = Data >> 1

Data = 0

S7 Done = 1; Output = Ocount

RTL Specification for Ones Counter

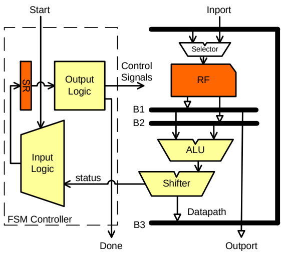
- **RTL Specification**
  - Controller and datapath netlist
  - Input and output tables for logic synthesis
  - RTL library needed for netlist

Input logic table

Present State	Inputs:		Next State	Output: Done
	Start	Data = 0		
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
```

Hardware Synthesis

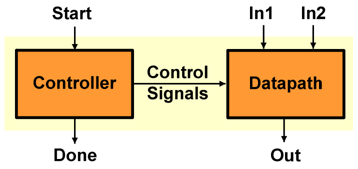
- ✓ Design flow
- ✓ RTL architecture
- ✓ Input specification
- **Specification profiling**
- RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- Chaining and multi-cycling
- Data and control pipelining
- Scheduling
- Component interfacing
- Conclusions

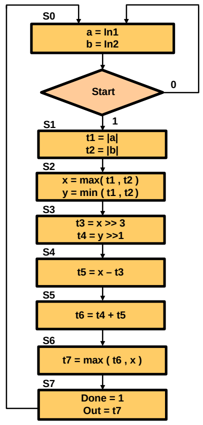
Profiling and Estimation

- **Pre-synthesis optimization**
- **Preliminary scheduling**
  - Simple scheduling algorithm
- **Profiling**
  - Operation usage
  - Variable life-times
  - Connection usage
- **Estimation**
  - Performance
  - Cost
  - Power

Square-Root Algorithm (SRA)

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





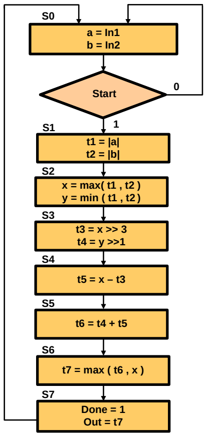
Variable and Operation Usage

Variable usage

	S1	S2	S3	S4	S5	S6	S7
a	X						
b	X						
t1		X					
t2		X					
x			X	X	X	X	
y			X				
t3				X			
t4				X	X		
t5					X		
t6						X	
t7							X
No. of live variables	2	2	2	3	3	2	1

Operation usage

	S1	S2	S3	S4	S5	S6	S7	Max. no. of units
abs	2							2
min		1						1
max		1				1		1
>>			2					2
-				1				1
+					1			1
No. of operations	2	1	2	1	1	1		



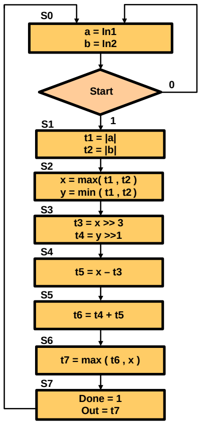
Connectivity usage

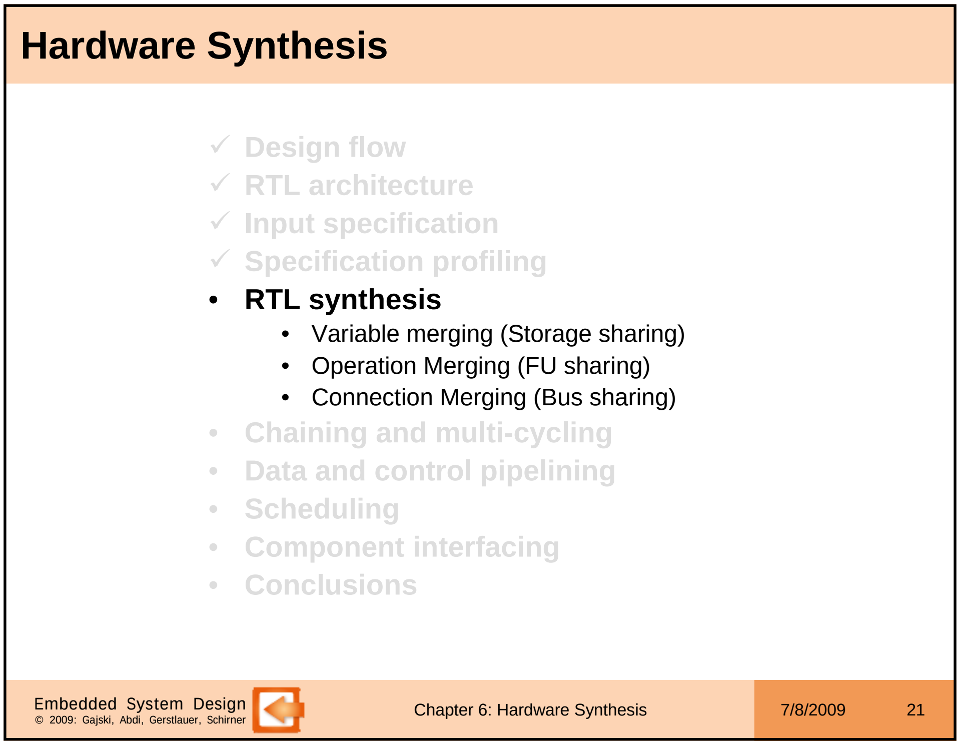
Operation usage

	S1	S2	S3	S4	S5	S6	S7	Max. no. of units
abs	2							2
min		1						1
max		1				1		1
>>			2					2
-				1				1
+					1			1
No. of operations	2	1	2	1	1	1		

Connectivity usage

	a	b	t1	t2	x	y	t3	t4	t5	t6	t7
abs1	I		O								
abs2		I		O							
min			I	I	O						
max			I	I	I	O				I	O
>>3					I		O				
>>1						I		O			
-					I		I		O		
+								I	I	O	

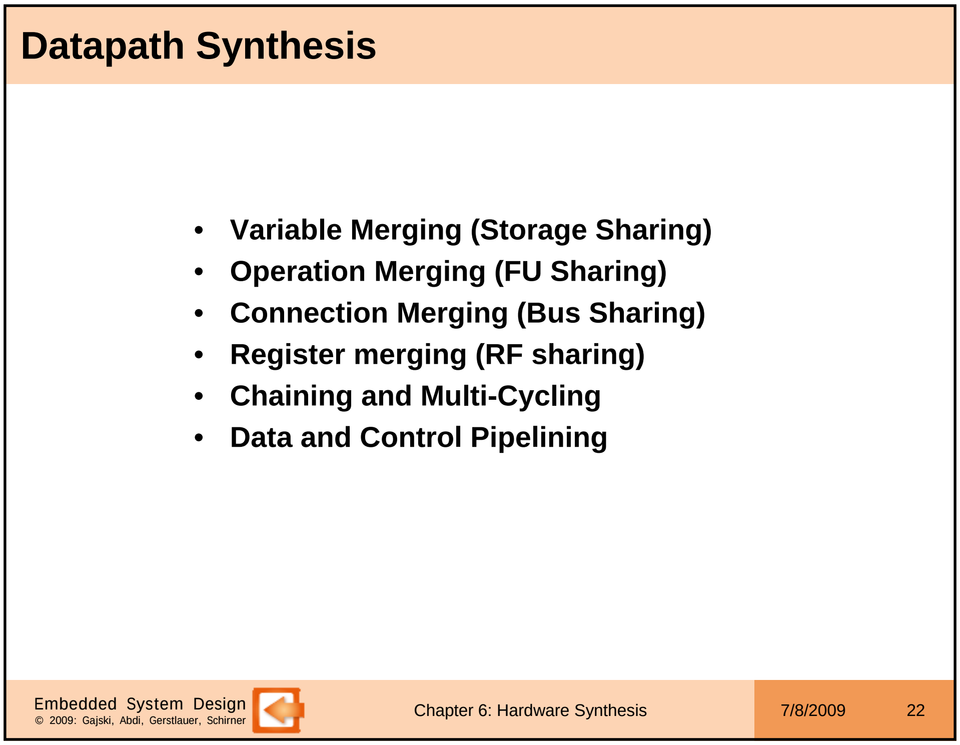




Hardware Synthesis

✓ Design flow
✓ RTL architecture
✓ Input specification
✓ Specification profiling

- **RTL synthesis**
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- Chaining and multi-cycling
- Data and control pipelining
- Scheduling
- Component interfacing
- Conclusions

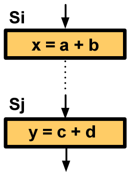


Datapath Synthesis

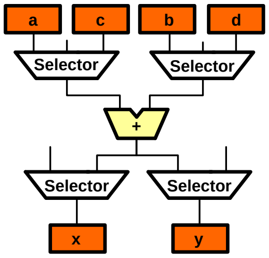
- **Variable Merging (Storage Sharing)**
- **Operation Merging (FU Sharing)**
- **Connection Merging (Bus Sharing)**
- **Register merging (RF sharing)**
- **Chaining and Multi-Cycling**
- **Data and Control Pipelining**

Gain in register sharing

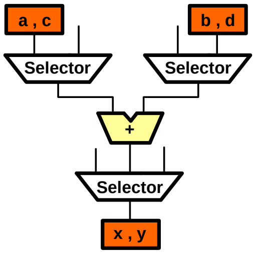
- **Register sharing**
  - Grouping variables with non-overlapping lifetimes
  - Sharing reduces connectivity cost



Partial FSMD



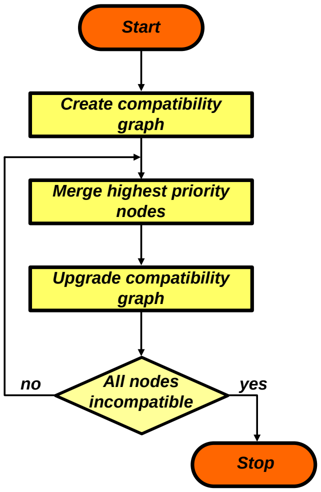
Datapath without register sharing



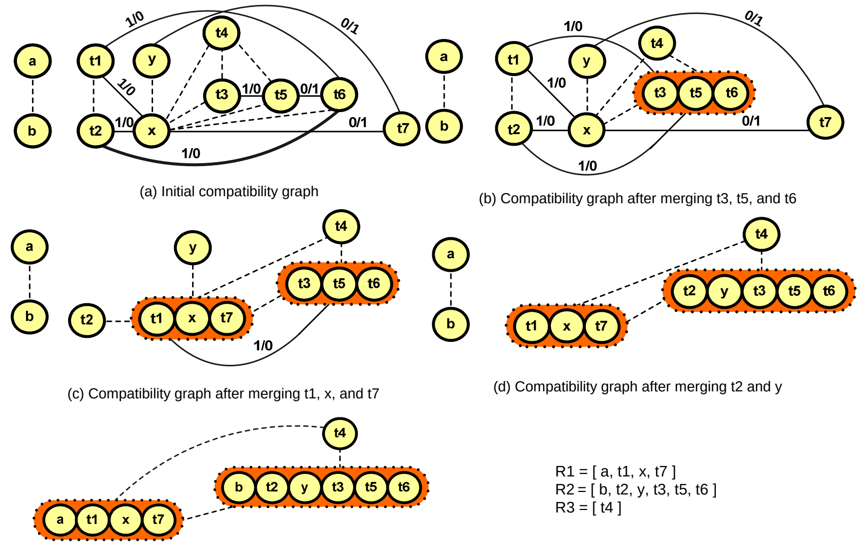
Datapath with register sharing

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

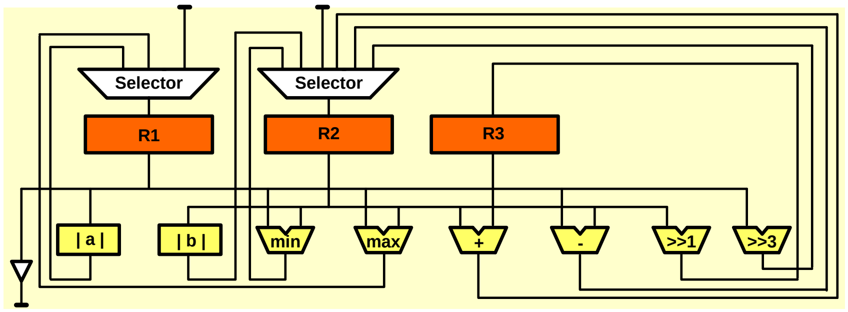
R1 = [a, t1, x, t7]
R2 = [b, t2, y, t3, t5, t6]
R3 = [t4]

(f) Final register assignments

Datapath with Shared Registers

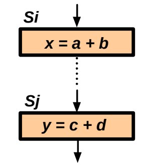
- Variables combined into registers
- One functional unit for each operation

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

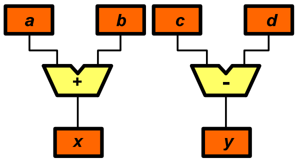


Gain in Functional Unit Sharing

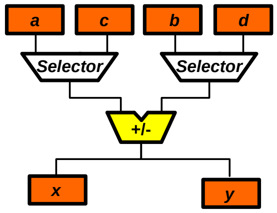
- **Functional unit sharing**
  - Smaller number of FUs
  - Larger connectivity cost



Partial FSMD

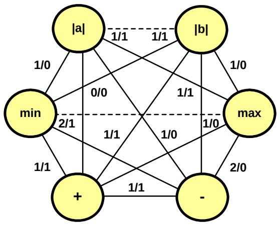


Non-shared design

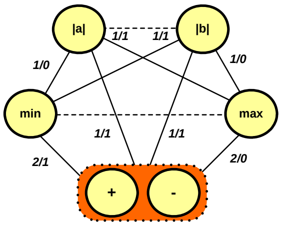


Shared design

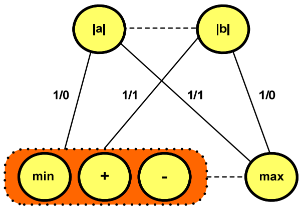
Operation Merging for SRA



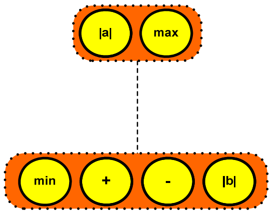
Initial compatibility graph



Compatibility graph after merging of + and -



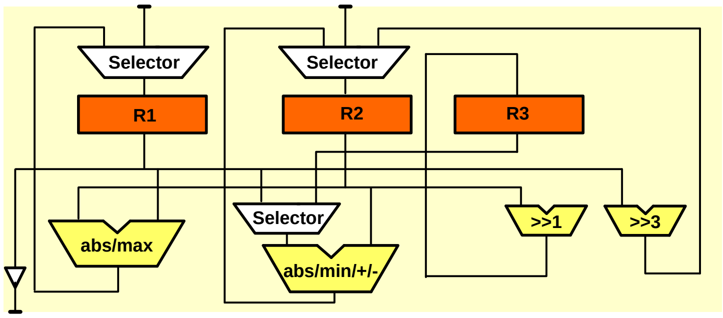
Compatibility graph after merging of min, +, and -



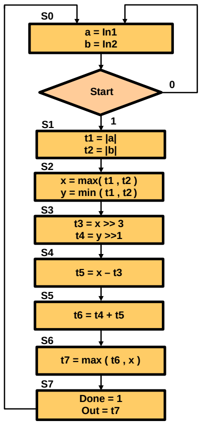
Final graph partitions

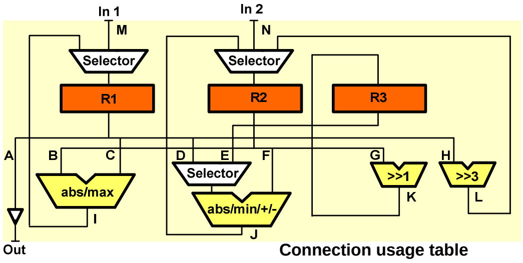
Datapath with Shared Registers and FUs

- Variables combined into registers
- Operations combined into functional units



Connection usage for SRA





Connection usage table

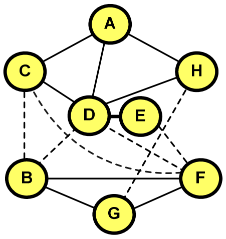
	S0	S1	S2	S3	S4	S5	S6	S7
A								X
B			X				X	
C		X	X				X	
D			X		X			
E						X		
F		X	X		X	X		
G				X				
H				X				
I		X	X				X	
J		X	X		X	X		
K				X				
L				X				
M	X							
N	X							

- Find compatible connections for merging into buses

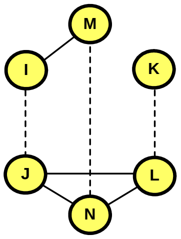
Connection Merging for SRA

- **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



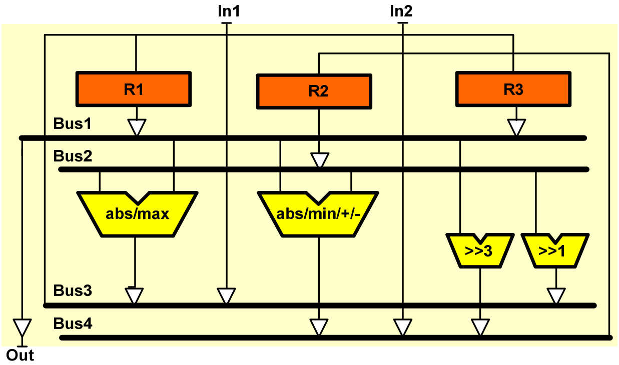
Compatibility graph for output buses

Bus assignment

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

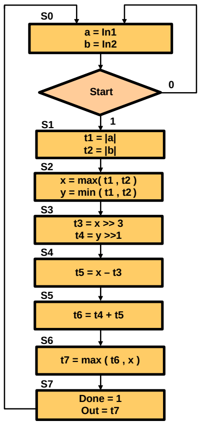
Datapath with Shared Registers, FUs and Buses

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



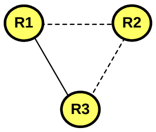
Register Merging into RFs

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

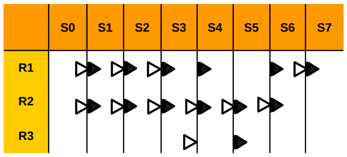


R1 = [a, t1, x, t7]
R2 = [b, t2, y, t3, t5, t6]
R3 = [t4]

Register assignment



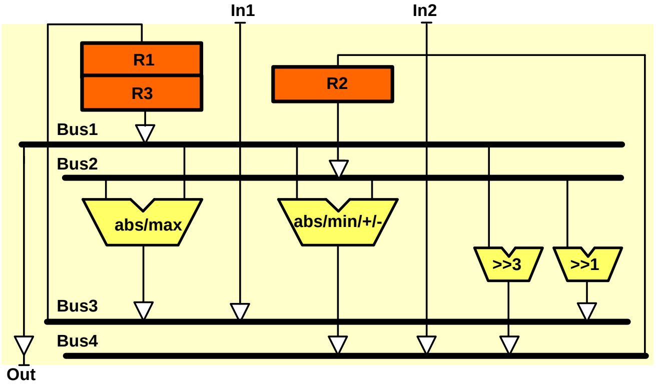
Compatibility graph



Register access table

Datapath with Shared RF

- **RF minimize connectivity cost by sharing ports**

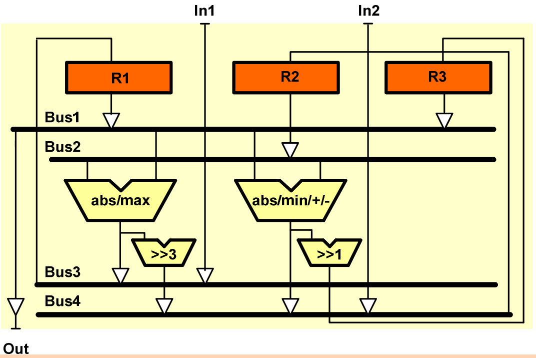


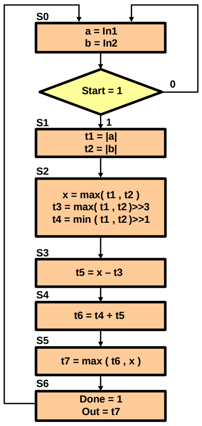
Hardware Synthesis

- ✓ Design flow
- ✓ RTL architecture
- ✓ Input specification
- ✓ Specification profiling
- ✓ RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- **Chaining and multi-cycling**
- Data and control pipelining
- Scheduling
- Component interfacing
- Conclusions

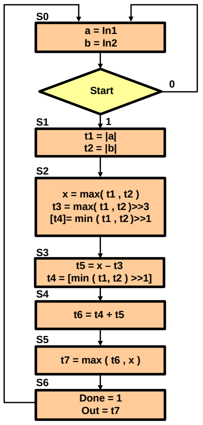
Datapath with Chaining

- 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 Chained and Multi-Cycled FUs



S0
a = In1
b = In2

Start
0
1

S1
t1 = |a|
t2 = |b|

S2
x = max(t1 , t2)
t3 = max(t1 , t2)>>3
[t4]= min (t1 , t2)>>1

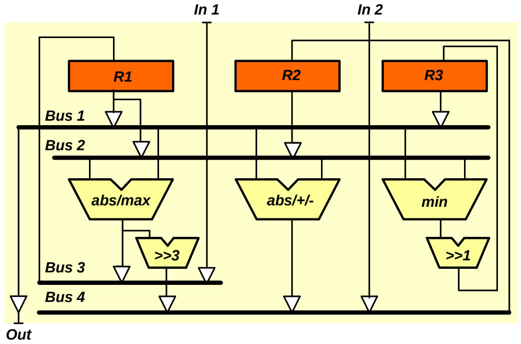
S3
t5 = x – t3
t4 = [min (t1, t2) >>1]

S4
t6 = t4 + t5

S5
t7 = max (t6 , x)

S6
Done = 1
Out = t7

- Multi-cycling allows use of slower FUs
- Multi-cycling allows faster clock-cycle



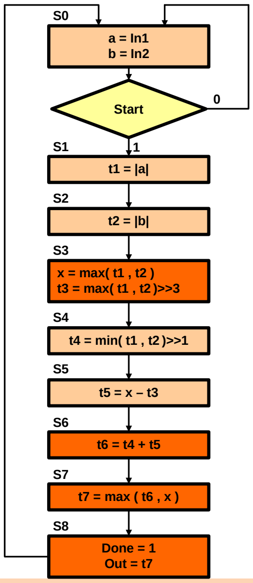
Hardware Synthesis

- ✓ Design flow
- ✓ RTL architecture
- ✓ Input specification
- ✓ Specification profiling
- ✓ RTL synthesis
    - Variable merging (Storage sharing)
    - Operation Merging (FU sharing)
    - Connection Merging (Bus sharing)
- ✓ Chaining and multi-cycling
- **Data and control pipelining**
- Scheduling
- Component interfacing
- Conclusions

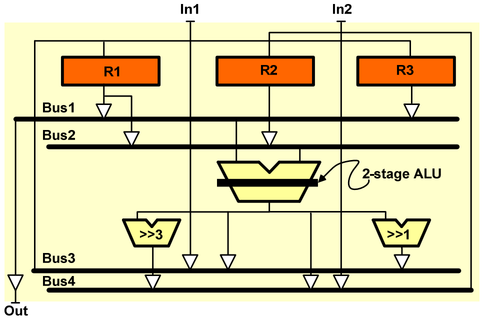
Pipelining

- **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 generaqted at the same time

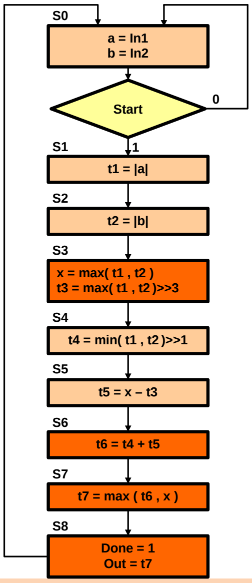
Functional Unit Pipelining (1)

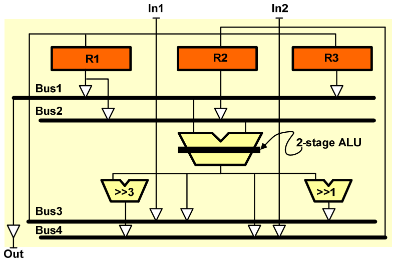


- •Operation delay cut in ”half”
- •Shorter clock cycle
- •Dependencies may delay some states
- •Extra NO states reduce performance gain



Functional Unit Pipelining (2)

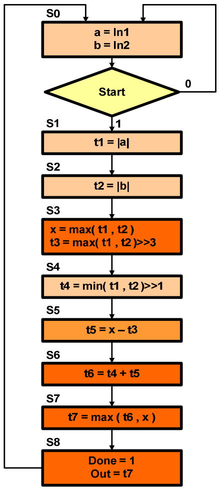




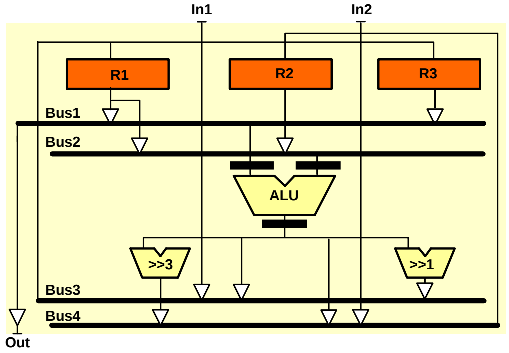
Timing diagram with 4 additional NO states

	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

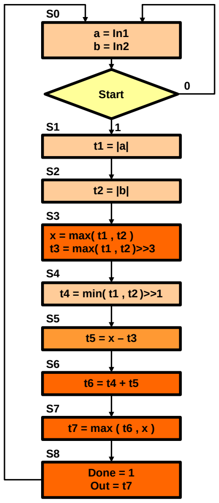
Datapath Pipelining (1)

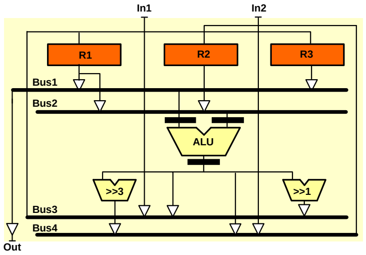


- •Register-to-register delay cut in “equal” parts
- •Much shorter clock cycle
- •Dependencies may delay some states
- •Extra NO states reduce performance gain



Datapath pipelining (2)



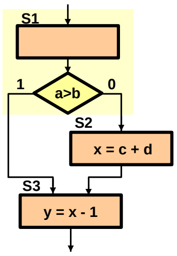


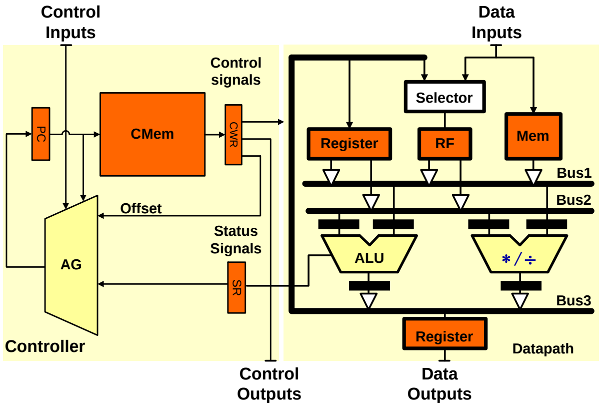
Timing diagram with additional NO clock cycles

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)

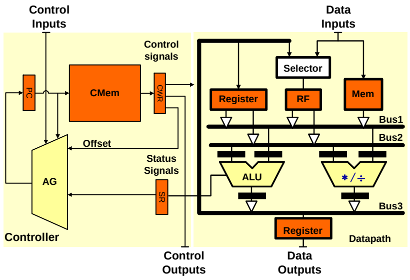
- •Fetch delay cut into several parts
- •Shorter clock cycle
- •Conditionals may delay some states
- •Extra NO states reduce performance gain

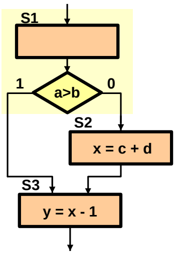




Data and Control Pipelining (2)

- 3 NO cycles for the branch
- 2 NO cycles for data dependence





Timing diagram with additional NO 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	

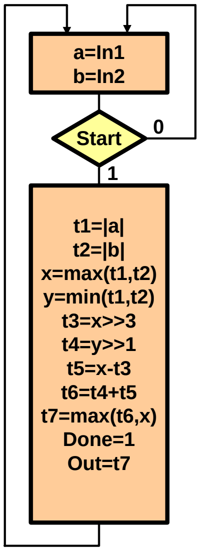
Hardware Synthesis

- ✓ Design flow
- ✓ RTL architecture
- ✓ Input specification
- ✓ Specification profiling
- ✓ RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- ✓ Chaining and multi-cycling
- ✓ Data and control pipelining
- **Scheduling**
- Component interfacing
- Conclusions

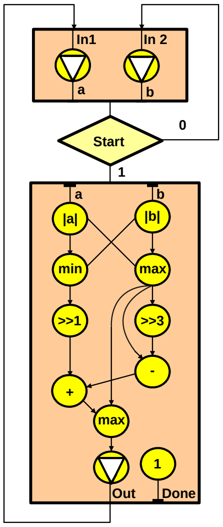
Scheduling

- **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)

C and CDFG for SRA Algorithm

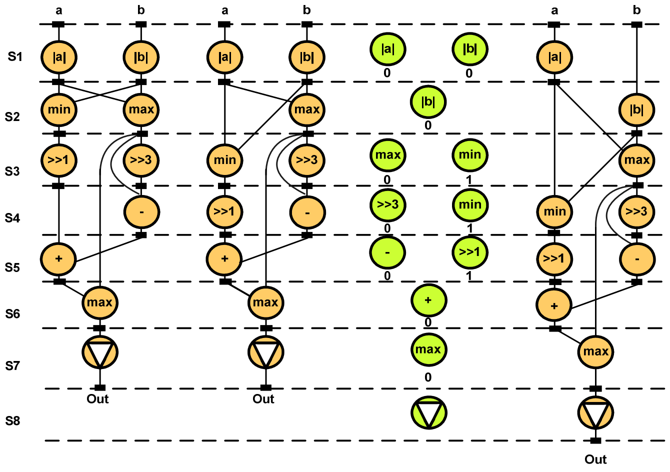


C flowchart



CDFG

RC Scheduling



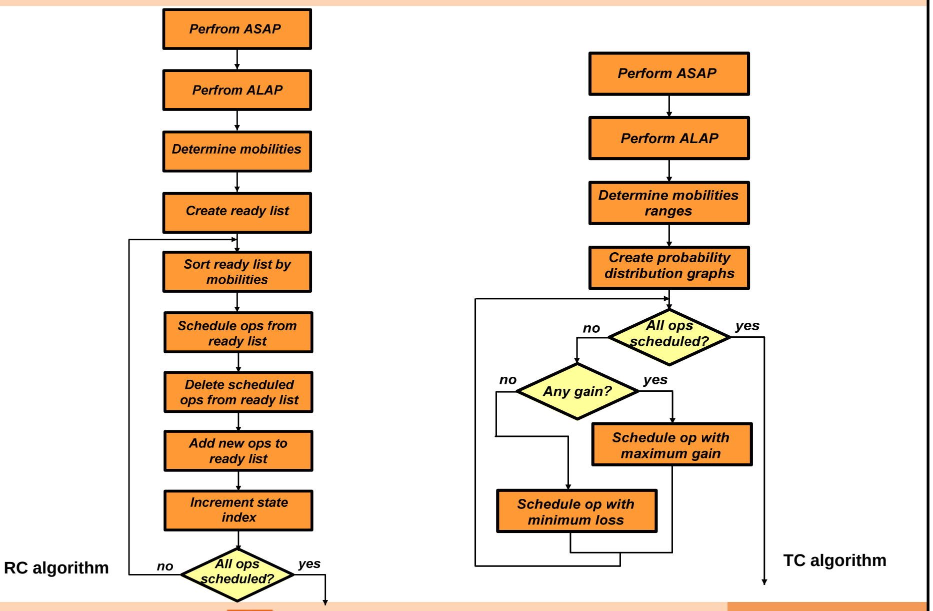
ASAP schedule

ALAP schedule

Ready list with mobilities (ALAP – ASAP)

RC schedule (for single FU and 2 shifters)

Scheduling Algorithms



Perfrom ASAP

Perfrom ALAP

Determine mobilities

Create ready list

Sort ready list by mobilities

Schedule ops from ready list

Delete scheduled ops from ready list

Add new ops to ready list

Increment state index

All ops scheduled?

no

yes

RC algorithm

Perform ASAP

Perform ALAP

Determine mobilities ranges

Create probability distribution graphs

All ops scheduled?

no

yes

Any gain?

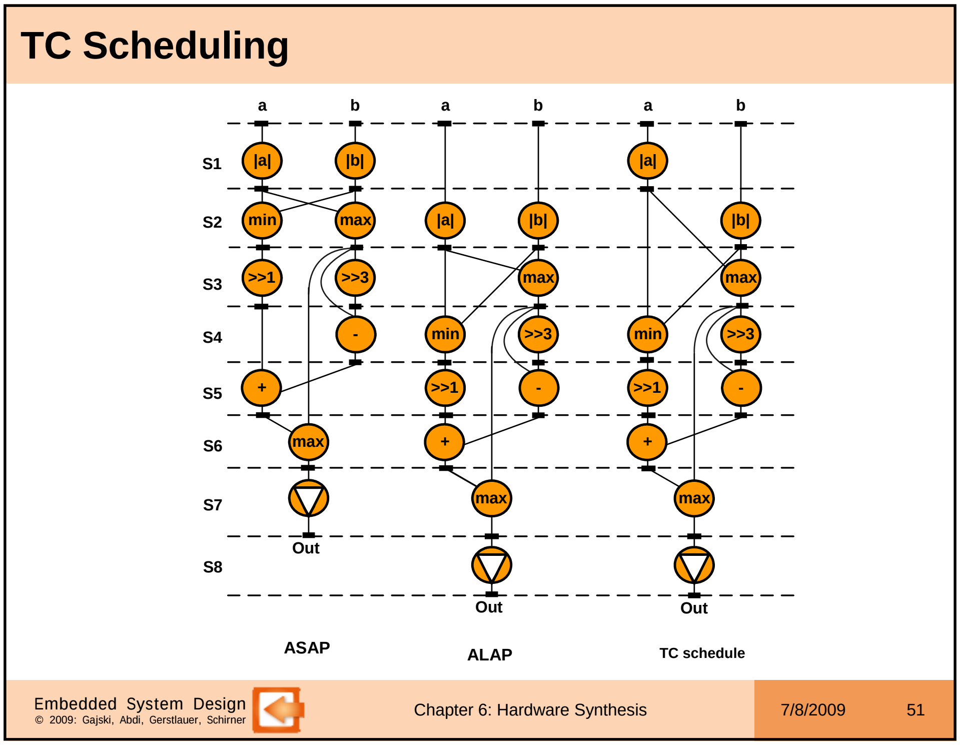
no

yes

Schedule op with maximum gain

Schedule op with minimum loss

TC algorithm

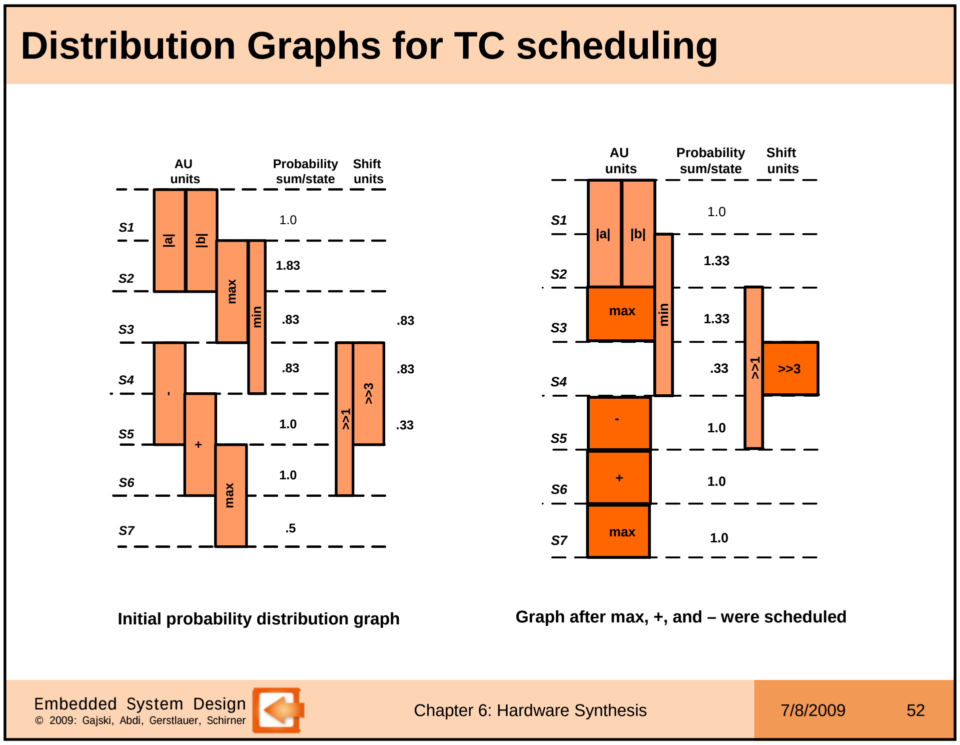


TC Scheduling

ASAP

ALAP

TC schedule

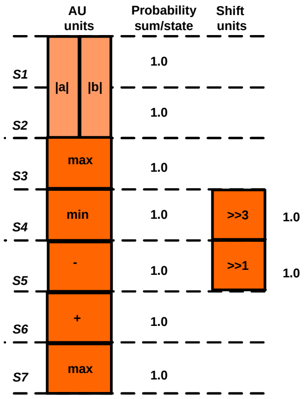


Distribution Graphs for TC scheduling

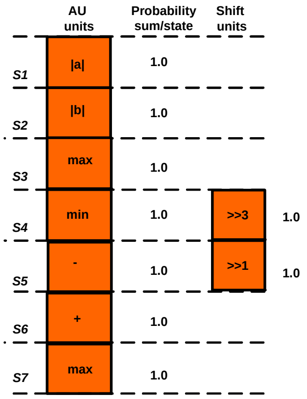
Initial probability distribution graph

Graph after max, +, and – were scheduled

Distribution Graphs for TC scheduling



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



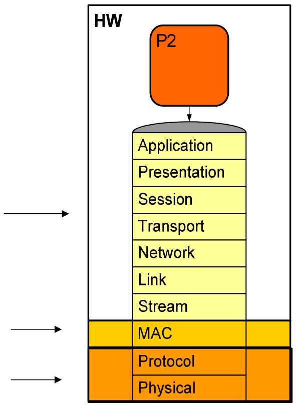
Distribution graph for final schedule

Hardware Synthesis

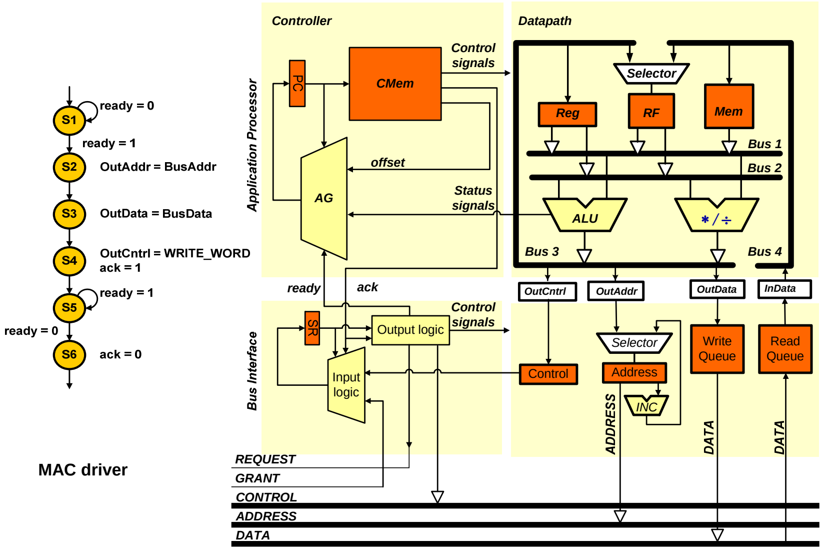
- ✓ Design flow
- ✓ RTL architecture
- ✓ Input specification
- ✓ Specification profiling
- ✓ RTL synthesis
  - Variable merging (Storage sharing)
  - Operation Merging (FU sharing)
  - Connection Merging (Bus sharing)
- ✓ Chaining and multi-cycling
- ✓ Data and control pipelining
- ✓ Scheduling
- **Component interfacing**
- Conclusions

Interface Synthesis

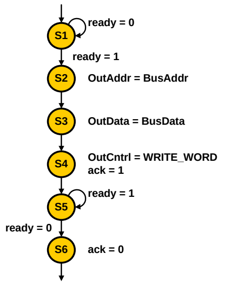
- •Combine process and channel codes
- •HW and protocol clock cycles may differ
- •Insert a bus-interface component
- •Communication in three parts:
  - •Freely schedulable code
    - •Scheduled with process code
  - •Schedule constrained code
    - •MAC driver from library for selected bus interface
  - •Bus interface
    - •Implemented by bus interface component from library



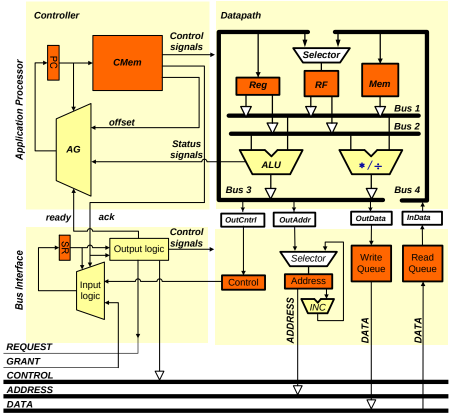
Bus Interface Controller (1)



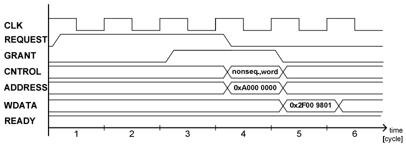
Bus Interface Controller (2)



MAC driver

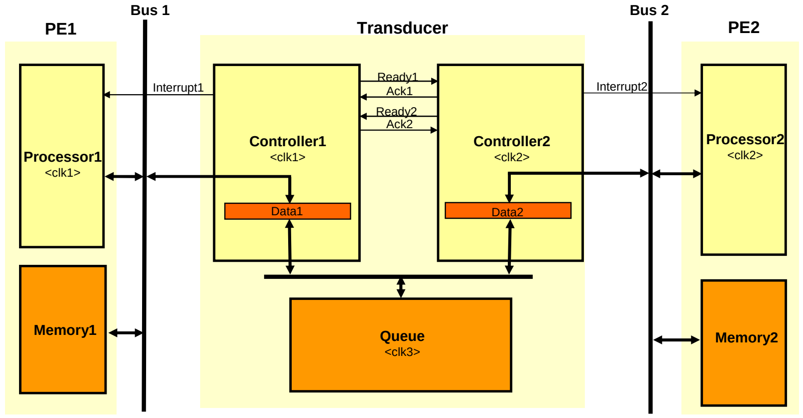


Bus protocol



Transducer/ Bridge

- Translates one protocol into another
  - Controller1 receives data with protocol1 and writes into queue
  - Controller2 reads from queue and sends data with protocol2



Conclusion

- **Synthesis techniques**
  - Variable Merging (Storage Sharing)
  - Operation Merging (FU Sharing)
  - Connection Merging (Bus Sharing)
- **Architecture techniques**
  - Chaining and Multi-Cycling
  - Data and Control Pipelining
  - Forwarding and Caching
- **Scheduling**
  - Metric constrained scheduling
- **Interfacing**
  - Part of HW component
  - Bus interface unit
- **If too complex, use partial order**