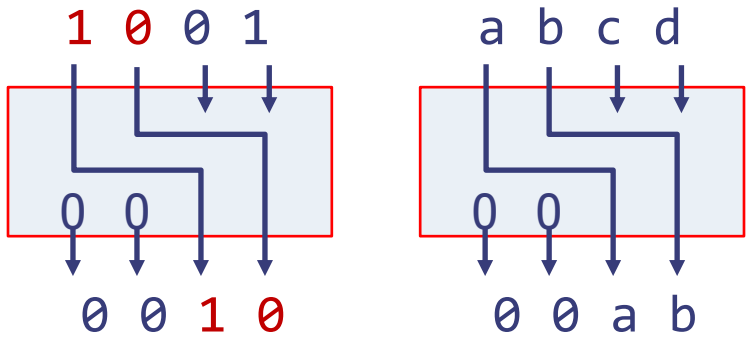
Barrel Shifter, Boolean Optimizations, and Logic Synthesis

Silvina Hanono Wachman
Computer Science & Artificial Intelligence Lab
M.I.T.

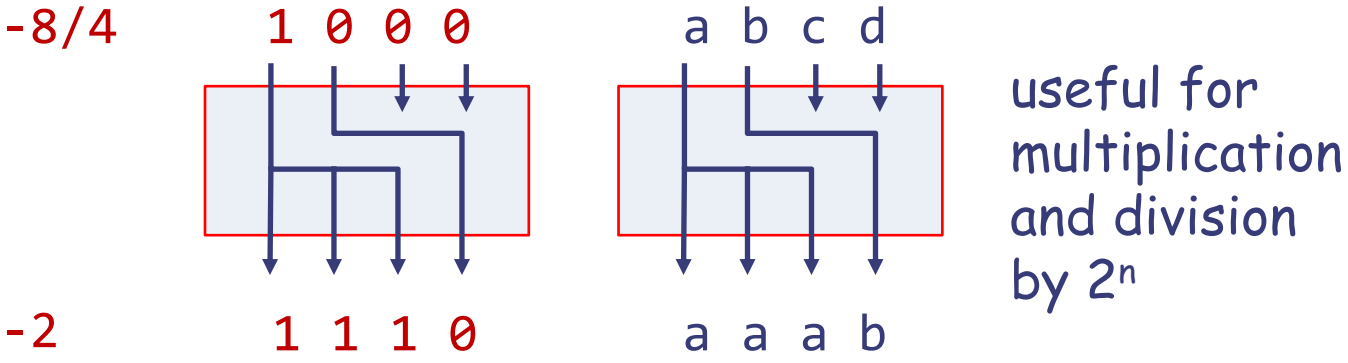
Reminders:
Lab 2 checkoff due Wed 2/27
Lab 3 due Thu 2/28
Quiz 1: March 7th 7:30-9:30PM

Shift operators

Logical right shift by 2



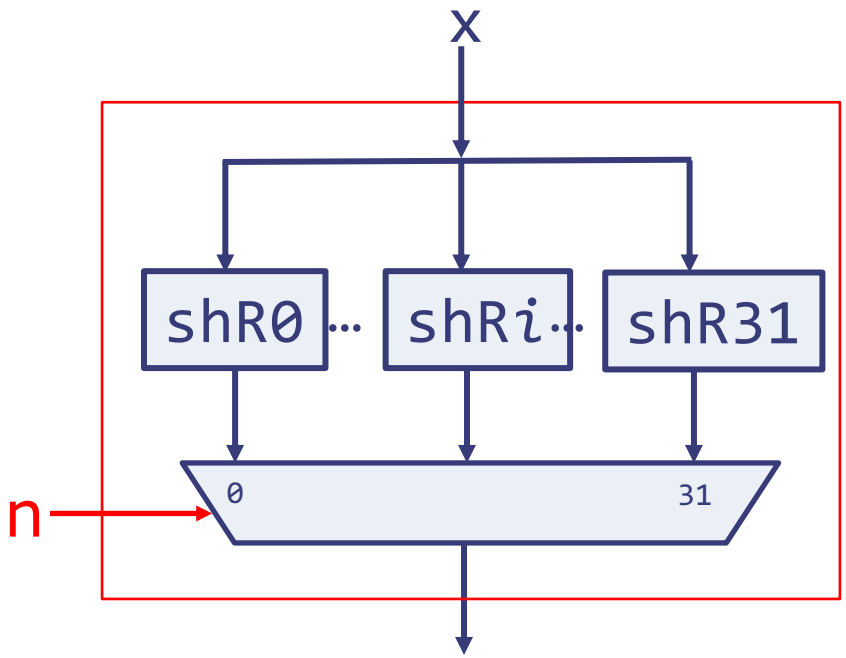
- Fixed size shift operation is cheap in hardware – just wire the circuit appropriately
- Arithmetic shifts are similar



useful for multiplication and division by 2^n

Logical right shift by *n*

- Suppose we want to build a shifter which shifts a value x by *n* where n is between 0 and 31
- One way to do this is by connecting 32 different shifters via a mux



How many *2*-way one-bit muxes are needed to implement this structure?

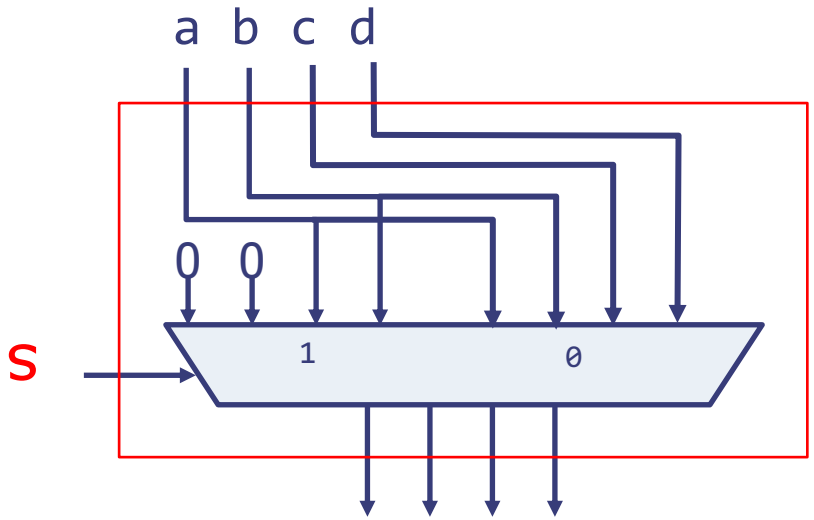
n*(n-1)

Can we do better?

Logical right shift by *n*

- Shift *n* can be broken down into log *n* steps of fixed-length shifts of size 1, 2, 4, …
  - For example, we can perform shift 5 (=4+1) by doing shifts of size 4 and 1
  - Thus, 8'b01100111 shift 5 can be performed in two steps:
    - 8'b01100111 $\overset{\text{shift 4}}{\Rightarrow}$ 8'b00000110 $\overset{\text{shift 1}}{\Rightarrow}$ 8'b00000011
- For a 32-bit number, a 5-bit *n* can specify all the needed shifts
  - $3_{10} = 00011_2$, $5_{10} = 00101_2$, $21_{10} = 10101_2$
  - The bit encoding of *n* tells us which shifters are needed; if the value of the i^{th} (least significant) bit is 1 then we need to shift by 2^i bits
- Tradeoff *delay* for *size*

Conditional operation: shift versus no-shift

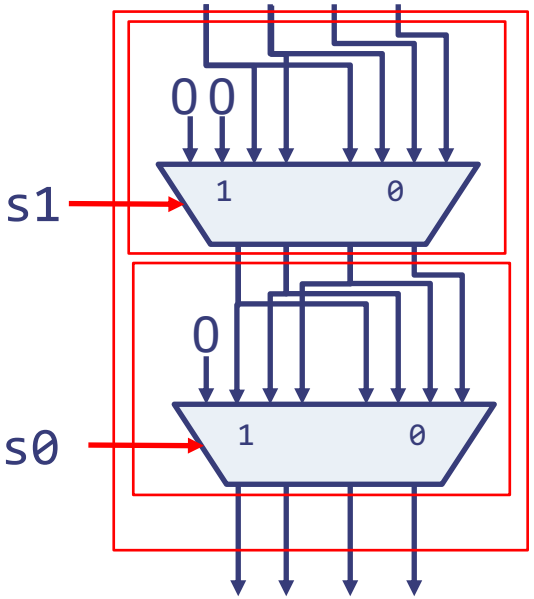


- We need a mux to select the appropriate wires: if s is one the mux will select the wires on the left (shift) otherwise it would select wires on the right (no-shift)

```
(s==1)? {2'b0,a,b}:{a,b,c,d};
```

Logical right shift circuit

- Define log *n* shifters of sizes 1, 2, 4, …
- Define log *n* muxes to perform a particular size shift
- Suppose s = {s1,s0} is a two bit number. Shift circuit to shift a number by s can be expressed as two conditional expressions where the second uses the output of the first



tmp[3:1]

```
Bit#(4) input = {a,b,c,d}
Bit#(4) tmp = (s1==1)? {2'b0,a,b}:input;
Bit#(4) output = (s0==1)? {1'b0,tmp[3],tmp[2],tmp[1]}:tmp;
```

Boolean Optimizations

Equivalence and Normal Form

- Every function can be described as a truth table
- There is a Boolean expression corresponding to every truth table

C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

sum-of-products (SOP) representation

$$Y = \bar{C}\bar{B}A + \bar{C}B\bar{A} + CB\bar{A} + CBA$$

- This representation is called the function's normal form
  - It is *unique*, but there may be simpler expressions

Simplify Boolean Expressions using Boolean Algebra

- Manipulate a Boolean Expression using the following properties:

commutative	$a \cdot b = b \cdot a$	$a + b = b + a$
associative	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$	$a + (b + c) = (a + b) + c$
distributive	$a \cdot (b + c) = a \cdot b + a \cdot c$	$a + b \cdot c = (a + b) \cdot (a + c)$
complements	$a \cdot \overline{a} = 0$	$a + \overline{a} = 1$
absorption	$a \cdot (a + b) = a$	$a + a \cdot b = a$
reduction	$a \cdot b + a \cdot \overline{b} = a$	$(a + b) \cdot (a + \overline{b}) = a$
DeMorgan's Law	$\overline{a \cdot b} = \overline{a} + \overline{b}$	$\overline{a + b} = \overline{a} \cdot \overline{b}$

Boolean Simplification of SOPs

- A minimal sum-of-products is a sum-of-products expression that has the smallest possible number of AND and OR operators

$$Y = \overline{C}\,\overline{B}A + CB\overline{A} + CBA + \overline{C}BA$$

reduction

$$Y = \overline{C}\,\overline{B}A + CB + \overline{C}BA$$

reduction

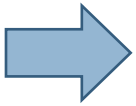
$$Y = \overline{C}A + CB$$

  - Minimal SOPs can be implemented with fewer and smaller gates
  - Unlike the normal form, it is not unique (a function may have multiple minimal SOPs)

Truth Tables with "Don't Cares"

Another way to reveal simplification is to rewrite the truth table using "don't cares" (-- or X) to indicate when the value of a particular input is irrelevant in determining the value of the output.

C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



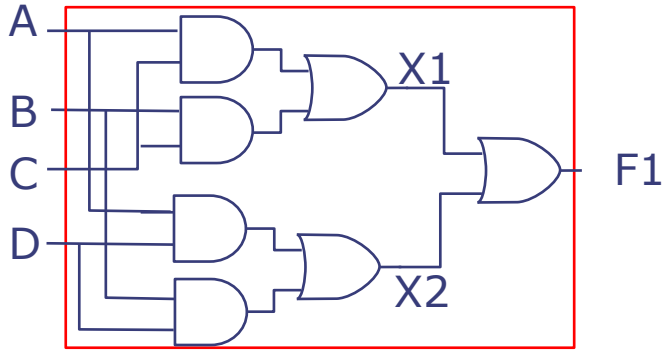
C	B	A	Y	
0	X	0	0	
0	X	1	1	$\rightarrow \overline{C}A$
1	0	X	0	
1	1	X	1	$\rightarrow CB$
X	0	0	0	
X	1	1	1	$\rightarrow BA$

Note: Some input combinations (e.g., 000) are matched by more than one row in the "don't care" table. For a well-defined Truth Table, all matching rows must specify the same output value!

Multi-level circuits

- Suppose we are restricted to using only two input AND and OR gates then a sum-of product circuit cannot always to be realized using two levels of logic
  - Example: $F = A \cdot C + B \cdot C + A \cdot D + B \cdot D$

$X1 = A \cdot C + B \cdot C$
$X2 = A \cdot D + B \cdot D$
$F = X1 + X2$



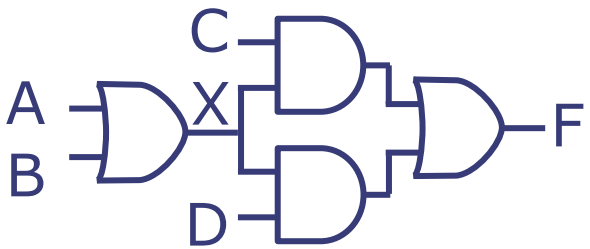
- Fast microprocessors typically use 8-levels of logic, while slower processors use as many as 14-18 levels of logic!

Common subexpression (CSE) optimization

- We can often reduce the number of gates by factoring out common subexpressions
- Example: $F = A \cdot C + B \cdot C + A \cdot D + B \cdot D$ (minimal SOP)

$$F = (A+B) \cdot C + (A+B) \cdot D$$

$$X = A + B$$
$$F = X \cdot C + X \cdot D$$



- CSE optimization increases the number of logic levels
- Until now our synthesized circuit always looked like a tree but the introduction of CSE optimization can turn a tree into a directed acyclic graph
- Multi-level simplification has no well-defined optimum

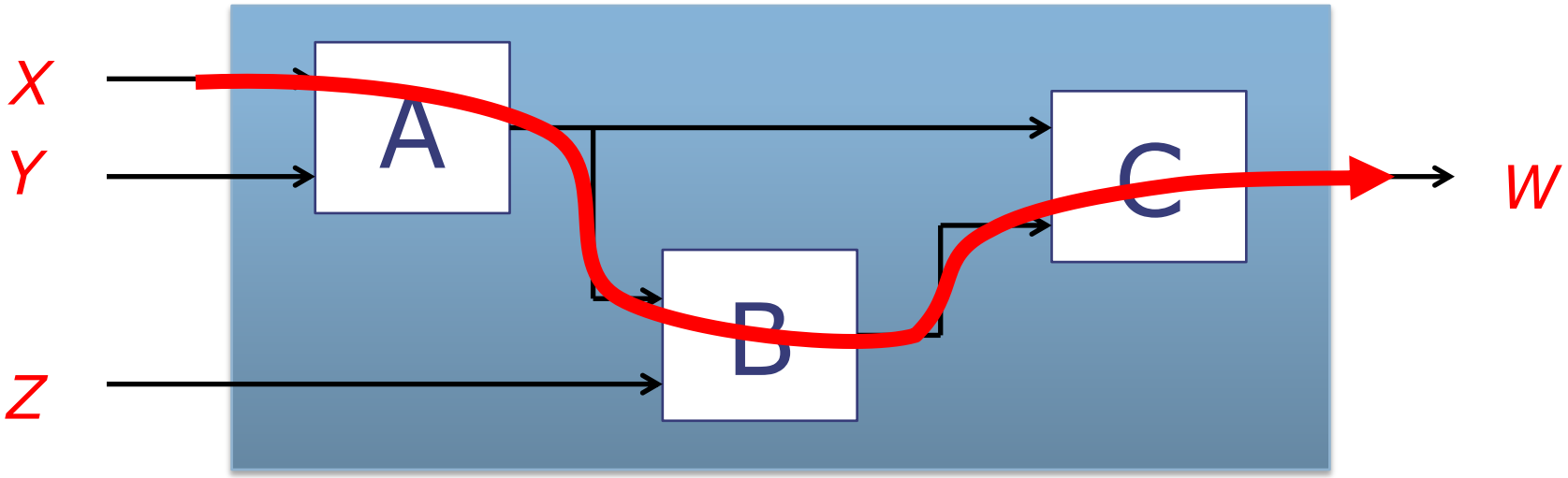
Logic Synthesis

Combinational Circuit Properties

- In addition to their functionality, combinational circuits have two important physical properties: *area* and *propagation delay* (t_{PD}), *i.e.*, the time it takes for the output to stabilize after stable inputs are available
- Both these properties are derived from the properties of the gates used to build the circuit
- Most circuits are built from some *standard cell library* implemented in a specific CMOS technology

Computing area and propagation delay

A, B and C are combinational circuits



Functional description? $W = f_C(f_A(X, Y), f_B(f_A(X, Y), Z))$

Propagation delay? $t_{PD} = t_{PD,A} + t_{PD,B} + t_{PD,C}$

Area? $area = area_A + area_B + area_C$

Ignoring the wires

Standard Cell Library

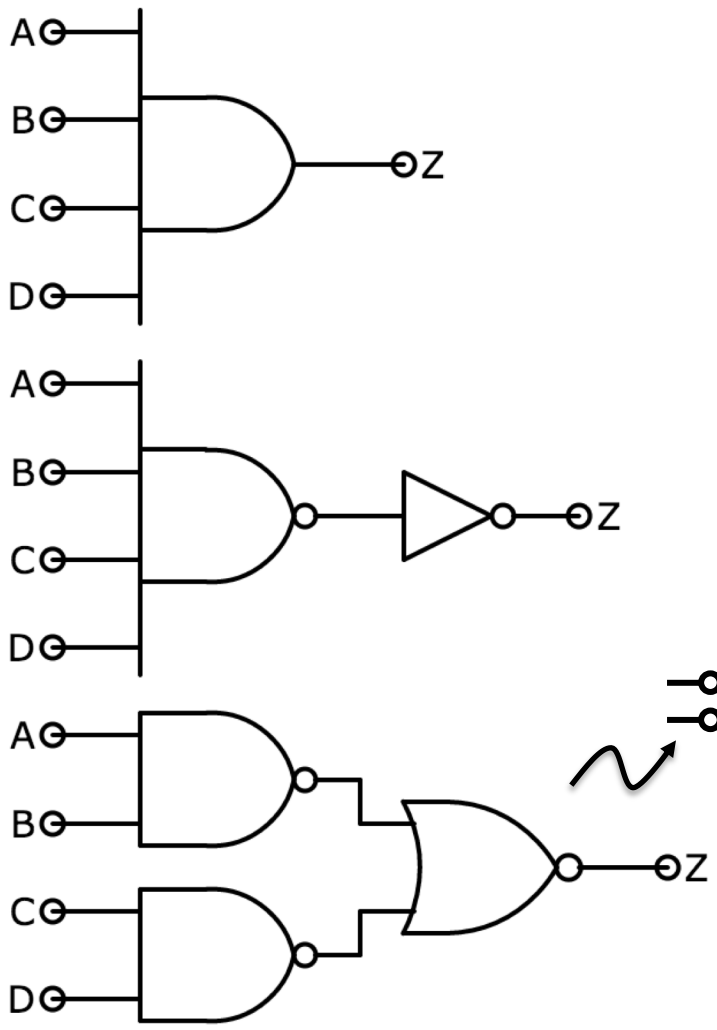
- Library of gates and their physical characteristics
- Example:

Gate	Delay (ps)	Area (μ^2)
Inverter	20	10
AND2	50	25
NAND2	30	15
OR2	55	26
NOR2	35	16
AND4	90	40
NAND4	70	30
OR4	100	42
NOR4	80	32

Observations:

1. In current technology (CMOS), inverting gates are faster and smaller
2. Delay and area grow with number of inputs

Design Tradeoffs: Delay vs Size



AND4:
t_{PD} = 90 ps, size = 40μ²

NAND4 + INV:
t_{PD} = 90 ps, size = 40μ²

Demorgan's Laws:

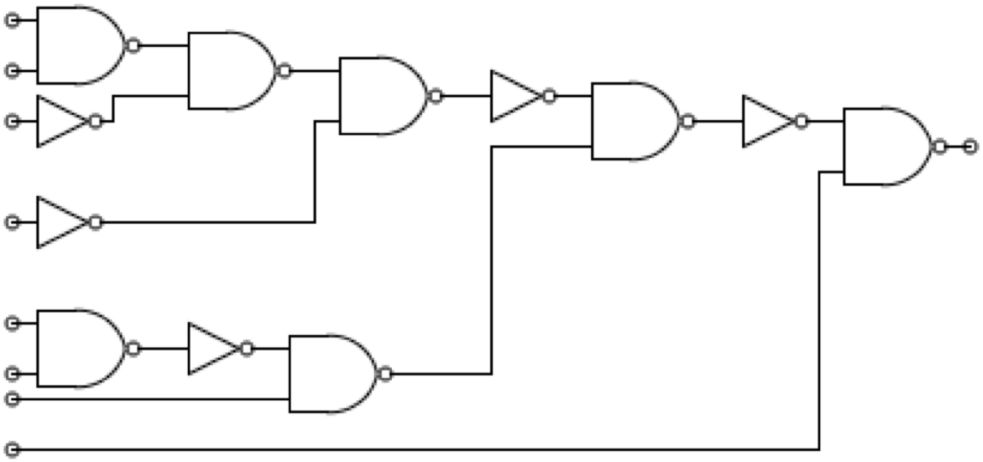
$$\overline{A} \cdot \overline{B} = \overline{A + B}$$

$$\overline{A} + \overline{B} = \overline{A \cdot B}$$

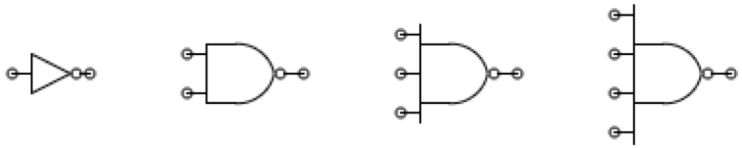
2*NAND2 + NOR2:
t_{PD} = 1 NAND2 + NOR2 = 65 ps,
size = 2 NAND2 + NOR2 = 46μ²

Example: Mapping a Circuit to a Standard Cell Library

Find an implementation of a circuit, e.g.,



Using gates from a standard cell library, e.g.,

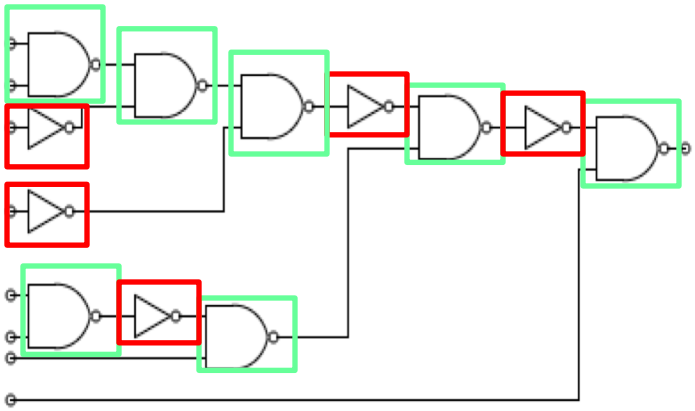


Area 2 3 4 5

That optimizes for some goal, e.g., minimum area

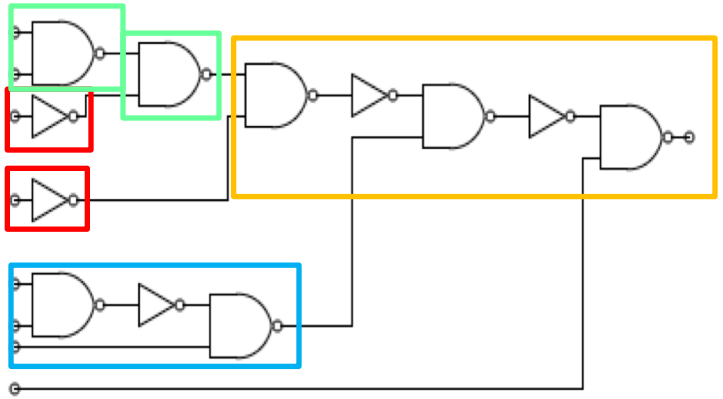
Example: Mapping a Circuit to a Standard Cell Library

Possible implementations:



7 NAND2 (3) = 21
5 INV (2) = 10

Total area cost: 31



2 INV = 4
2 NAND2 = 6
1 NAND3 = 4
1 NAND4 = 5

Total area cost: 19

Logic Optimization

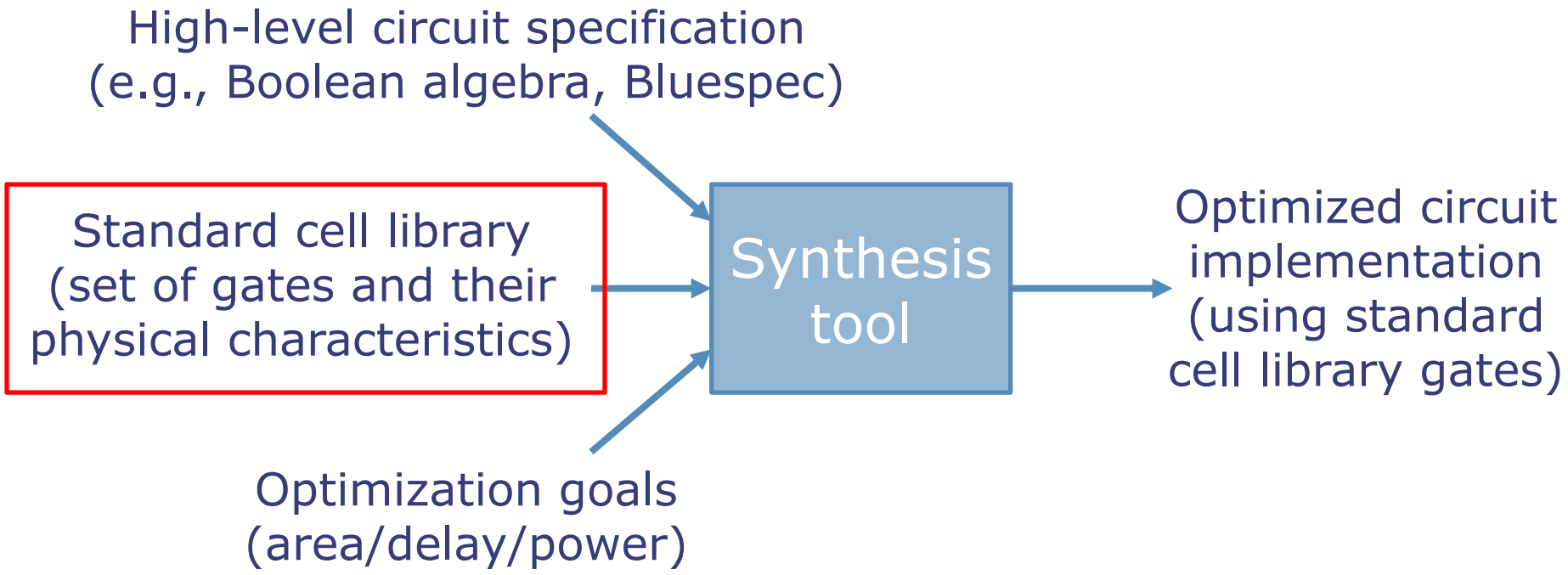
- Synthesizing an optimized circuit is a very complex problem
  - Boolean simplification
  - Mapping to cell libraries with many gates
  - Multidimensional tradeoffs (e.g., minimize area-delay-power product)

- Infeasible to do by hand for all but the smallest circuits!

- Instead, hardware designers write circuits in a hardware description language, and use a synthesis tool to derive optimized implementations

Logic Synthesis Tool

- In practice, tools use Boolean simplification and other techniques to synthesize a circuit that meets certain area, delay, and power goals:



Take-Home Problem

1. Find a minimal SOP expression for the following Boolean (SOP) expression:

$$\overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C$$

2. Using the gates in slide #18, find an implementation of this minimal SOP that minimizes area.

Thank you!

Next lecture:
Complex combinational circuits in Bluespec