COS 140: Foundations of Computer Science

Designing Circuits from Specifications

Fall 2018

Homework

- Reading: Chapter 6
- Homework:
  - All exercises at the end of Chapter 6
  - Due Monday, 9/24
  - All homework is due in class in *hardcopy* form on the due date!
- Cybersecurity team: See *umcst.maine.edu* for a description and email link.

Copyright © 2002–2018 UMaine Computer Science Department – 2 / 26

Circuit Specifications

Circuits

- Computer components realize *functions*
- Components are *logic circuits*
- How to describe a function – a *specification*
- Once there is a specification – how can we design a circuit for it?

Copyright © 2002–2018 UMaine Computer Science Department – 3 / 26

Circuit Specifications

- Two types: Boolean algebra and truth tables
- Boolean algebra expressions
  - Good when can state function in words, with logical operators

    E.g.: Create an expression that equals 1 when all of the inputs are 1, or when one, and only one, input is 0.
- Truth tables
  - Particularly good when relationship of input to output does not follow an easily-stated pattern.

    E.g.: Cards from a deck are assigned an input pattern arbitrarily.
  - Also good when you do not want to specify outputs for all inputs.

Example

Suppose our company makes jelly beans – 50 varieties of them total, but only some are made each month. How can we design a circuit that tells us, for any particular variety we are interested in, whether it is being made (1) or not being made (0) this month?

- Naïve way: 50 input lines, one input line for each variety
- Too many lines!

Example

50 varieties, not all made (1) each month.

- □ Better way: *code* the inputs
- □ Let each variety be a unique pattern of 1s in the input
- □ n input lines can represent 2^n different patterns
- □ How many input lines do we need?

n:	1	2	3	4	5	6
2^n:	2	4	8	16	32	64

- □ What about the leftovers?

Copyright © 2002–2018 UMaine Computer Science Department – 6 / 26

Example

50 varieties, not all made (1) each month.

- □ There are $64 - 50 = 14$ unused combinations
- □ We don't care whether their corresponding outputs are 1 or 0: *don't cares*
- □ We can leave these out of the truth table or mark with a special symbol (e.g., "–")

Copyright © 2002–2018 UMaine Computer Science Department – 7 / 26

Example

50 varieties, not all made (1) each month.

A	B	C	D	E	F	Being made?
...						
1	1	0	0	0	0	1
1	1	0	0	0	1	0
1	1	0	0	1	0	–
1	1	0	0	1	1	–
...						

With algebraic specification: have to decide on a value for each.

Copyright © 2002–2018 UMaine Computer Science Department – 8 / 26

Truth Tables with Don't Cares

- The circuit designer can choose the output value which is easiest to realize in the circuit for don't cares.
  - The circuit will have to have a value for each set of inputs.
  - Don't cares mean we "don't care" what that value is for some set of values for inputs variables.

Copyright © 2002–2018 UMaine Computer Science Department – 9 / 26

Creating Circuits from Algebraic Expressions

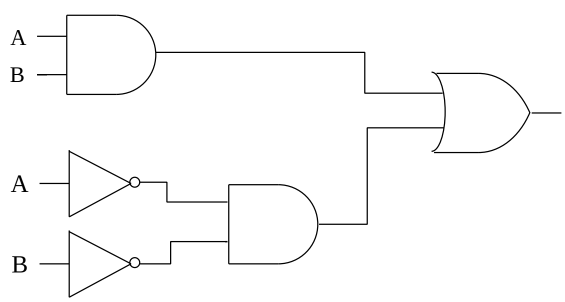
- Gates correspond to operators.
- Show input variables.
- Lines (representing physical connections) go from variables or output of a gate to input for a gate or output for the circuit.
- Gates should be visited in the order that the subexpressions are evaluated, as values "travel" along lines from input to output.
- Usually show circuits with input on left and output on right.

Copyright © 2002–2018 UMaine Computer Science Department – 10 / 26

Example: Creating a Simple Circuit

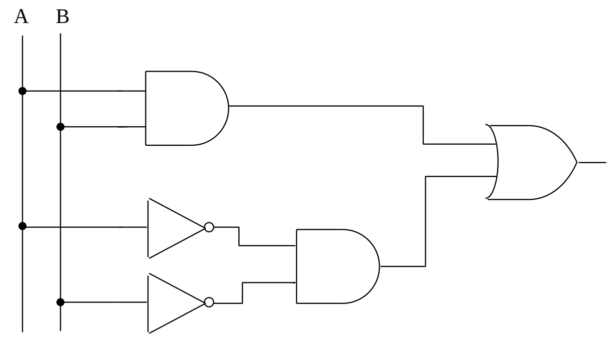
Create a circuit with output 1 if and only if the value of both of its input variables are 1 or both are 0.

Algebraic specification: $AB + \overline{A}\,\overline{B}$



Copyright © 2002–2018 UMaine Computer Science Department – 11 / 26

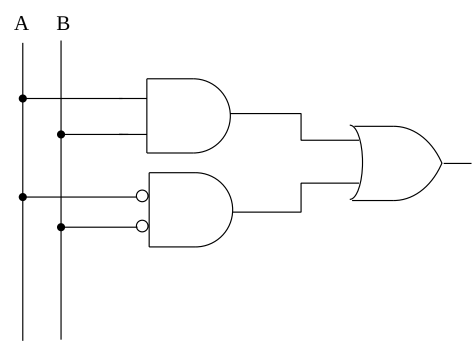
Example: Other Ways to Draw Circuits



Input values can be shown once, connected to input for appropriate gates.

Copyright © 2002–2018 UMaine Computer Science Department – 12 / 26

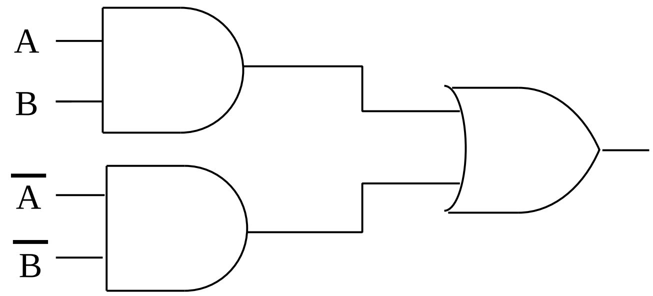
Example: Other Ways to Draw Circuits (cont'd)



NOT can be represented by a circle anywhere on the input line.

Copyright © 2002–2018 UMaine Computer Science Department – 13 / 26

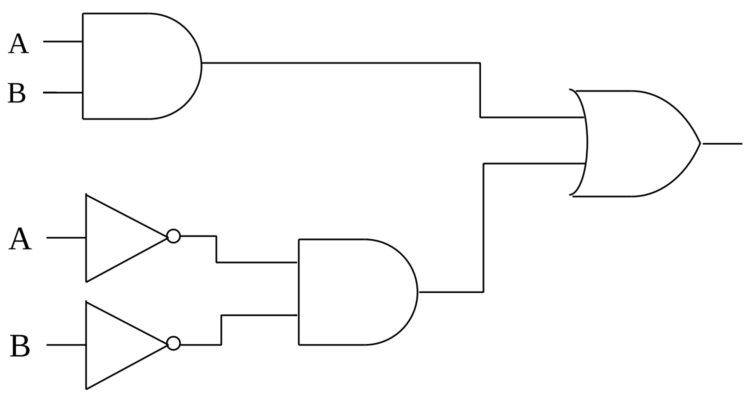
Example: Other Ways to Draw Circuits (cont'd)



Input values can be shown as negated.

Copyright © 2002–2018 UMaine Computer Science Department – 14 / 26

Example (cont'd): Following the Values



(Fill in values during lecture)

Copyright © 2002–2018 UMaine Computer Science Department – 15 / 26

Creating Circuits from Truth Tables

- Have truth table with inputs and values for sets of inputs given.

A	B	C	F
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

  - For each set of inputs (i.e., each row) with an output of 1, create a *product term*.
  - The product term contains each input variable (or its negation).
  - If the value of the input variable is 1, it appears in the product term. If the value of the input variable is 0, its negation appears in the product term.

Copyright © 2002–2018 UMaine Computer Science Department – 16 / 26

Creating Circuits from Truth Tables (cont'd)

A	B	C	F	Product Term
0	0	0	0	
0	0	1	0	
0	1	0	1	$\overline{A}B\overline{C}$
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	$AB\overline{C}$
1	1	1	1	ABC

$$\Rightarrow \overline{A}B\overline{C} + AB\overline{C} + ABC$$

Sum of products (SOP) form.

Copyright © 2002–2018 UMaine Computer Science Department – 17 / 26

Creating Circuits from Truth Tables (cont'd)

- For these inputs, if each variable has the correct value, the output should be 1. AND ensures that all variables have the correct input value.
- OR together the product terms that were created in the previous step, so that *any* set of correct inputs produces a 1 as output.

Copyright © 2002–2018 UMaine Computer Science Department – 18 / 26

Creating Circuits from Truth Tables (cont'd)

- Once an SOP expression is formed, algebraic substitution or any method for creating equivalent expressions, can be used to create an equivalent expression that will be better for realizing in a circuit.
- A circuit is created from the resulting logical expression.

Copyright © 2002–2018 UMaine Computer Science Department – 19 / 26

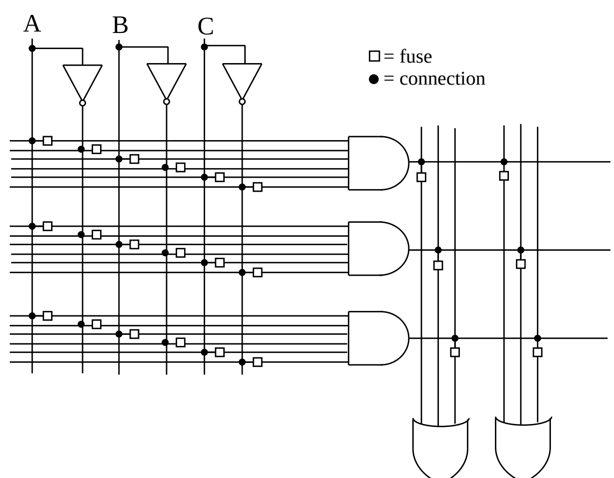
Programmable Logic Arrays

What is a PLA?

- Programmable Logic Array (PLA) is a chip designed with NOT, AND, and OR gates so it can handle arbitrary SOP expressions.
- A subarray of ANDs creates product terms from the inputs.
  - All inputs and their negations are available to all AND gates.
- A subarray of ORs takes input from the ANDs and creates outputs.
  - The chip can calculate several functions at once because of the different outputs.
- Large chips on the order of 25 inputs and 15 outputs.

Copyright © 2002–2018 UMaine Computer Science Department – 21 / 26

Example: A PLA



A small PLA with 3 inputs and 2 outputs.

This is a small PLA for illustrative purposes only. Can have any number of AND and OR gates.

Copyright © 2002–2018 UMaine Computer Science Department – 22 / 26

Using PLAs for SOPs

Advantages of a PLA

- Can capture all SOPs (within constraints of input size), and SOPs can be used to express any Boolean function.
- Several outputs
  - Can have one function for each output through an OR gate.
  - Can use calculation of a product in more than one function.
- Take advantage of the efficiency (cost, size, speed) of large-scale integration in a general-purpose way.

Copyright © 2002–2018 UMaine Computer Science Department – 23 / 26

Programming the PLA

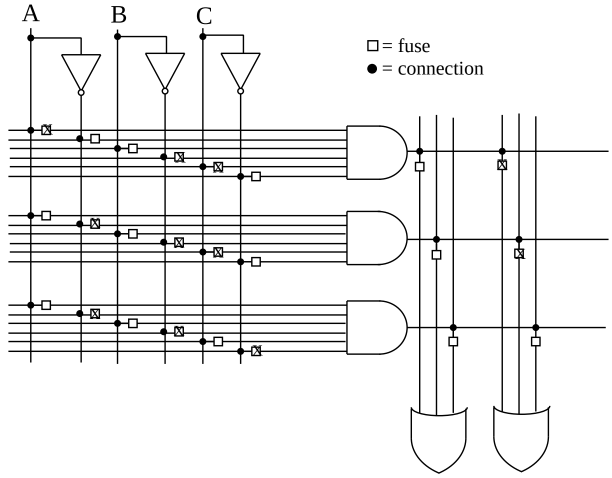
Programming the PLA

- Field-programmable logic array
  - All connections are made using fuses at the intersection points.
  - Fuses are blown when the connection is not wanted in the circuit.
- Or PLA can be programmed during chip fabrication

Copyright © 2002–2018 UMaine Computer Science Department – 24 / 26

Example

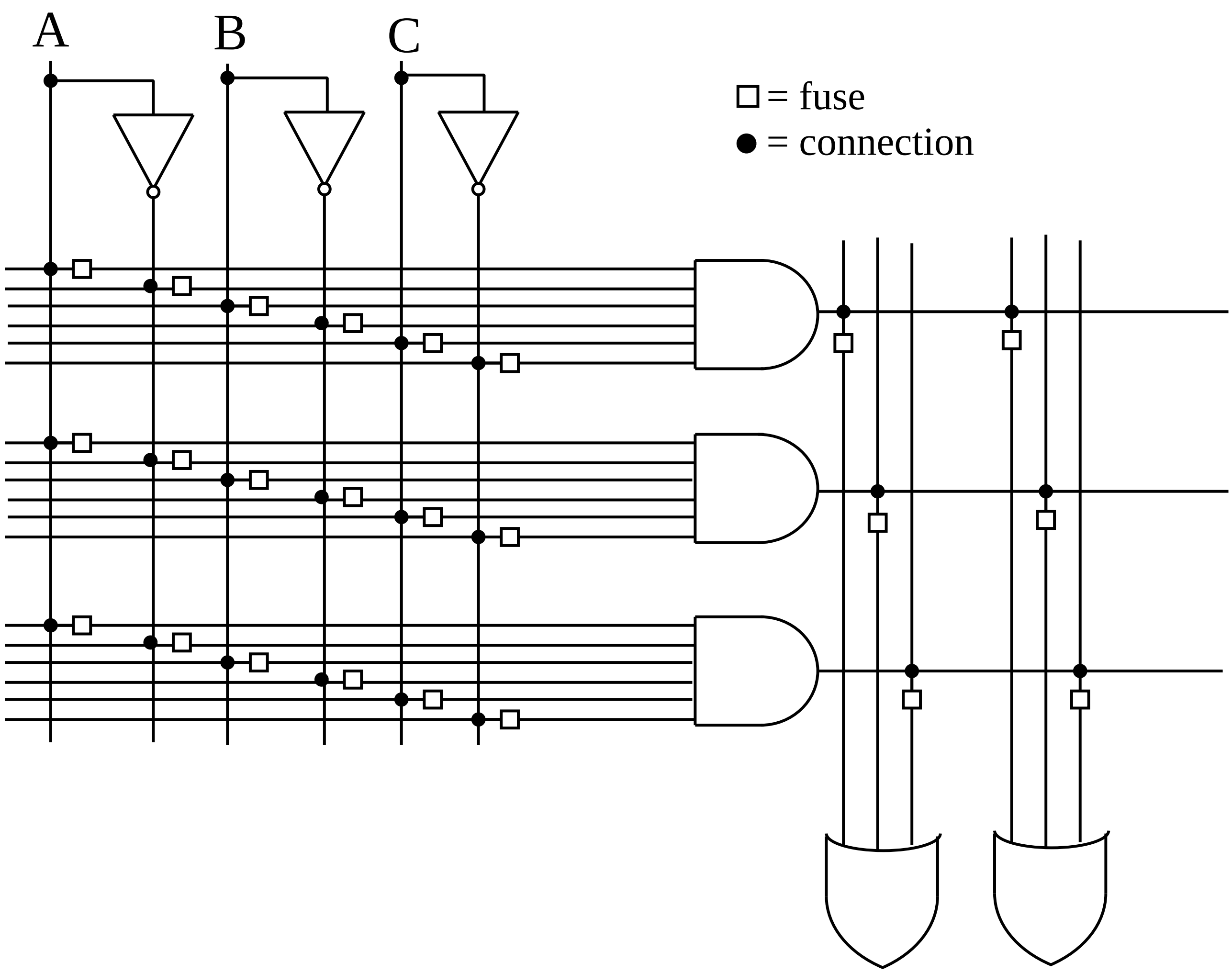
A simple PLA programmed for $\overline{A}B\overline{C} + AB\overline{C} + ABC$ and ABC.



Fuses marked X will have been blown and will *not* be part of the input to the next gate.

Copyright © 2002–2018 UMaine Computer Science Department – 25 / 26

Example



Copyright © 2002–2018 UMaine Computer Science Department – 26 / 26