CSEN102

Introduction to Computer Science

Lecture 10:
Boolean Logic

Prof. Dr. Slim Abdennadher
Dr. Aysha Alsafty, slim.abdennadher@guc.edu.eg,
aysha.alsafty@guc.edu.eg

German University Cairo, Department of Media Engineering and Technology

23.12.2017 - 28.12.2017

1 Synopsis

1.1 Boolean algebra

Synopsis

Recall the Boolean operations and their Truth tables:

AND (conjunction, product)

x	y	$x * y$
0	0	0
0	1	0
1	0	0
1	1	1

OR (disjunction, sum)

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT (complement, negation)

x	$\neg x$
0	1
1	0

Boolean algebra

A Boolean Algebra must satisfy these axioms:

$x + 0 = x$	$x * 1 = x$	
$x + 1 = 1$	$x * 0 = 0$	
$x + x = x$	$x * x = x$	
$x + x' = 1$	$x * x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutativity
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributivity
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan's Law

$x + 0 = x$	$x * 1 = x$	
$x + 1 = 1$	$x * 0 = 0$	
$x + x = x$	$x * x = x$	
$x + x' = 1$	$x * x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutativity
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributivity
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan's Law

$x + 0 = x$	$x * 1 = x$	
$x + 1 = 1$	$x * 0 = 0$	
$x + x = x$	$x * x = x$	
$x + x' = 1$	$x * x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutativity
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associativity
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributivity
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan's Law

- The axioms in magenta are the same as *regular algebraic rules*.
- The axioms in cyan deal with the *complement*.

2 Boolean logic

2.1 Encoding boolean logic in hardware

Circuits

Encoding boolean logic in hardware

Expressions and circuits

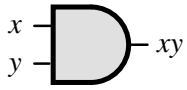
- A circuit is a *network of gates* that implements one or more boolean functions.
- We can build a circuit for any Boolean expression by *connecting primitive logic gates* in the correct order.
- Notice that the order of operations is explicit in the circuit.

Primitive logic gates

- A gate is an electronic device that operates on a collection of binary inputs to produce a binary output.
- Each basic operation can be *implemented in hardware* with a logic gate.

AND (conjunction)

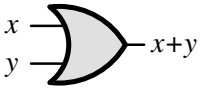
x	y	$x * y$
0	0	0
0	1	0
1	0	0
1	1	1



AND

OR (disjunction)

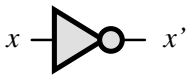
x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1



OR

NOT (complement)

x	$\neg x$
0	1
1	0



NOT

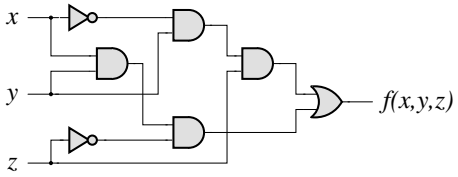
Encoding boolean logic in hardware

Example 1.

- *Truth table:*

X	Y	Z	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

- *Circuit:*



- *Expression:*
  $f(x, y, z) = x' * y * z +$
  $x * y * z'$

Equivalence proof with truth tables

- Two expressions are *equivalent* iff they always produce the same results for the same inputs

Example 2 ($(x + y)' = x'y'$).

x	y	$x + y$	$(x + y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x	y	x'	y'	$x'y'$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

3 Technical realization of Boolean logic

3.1 The transistor

Technical realization of Boolean logic

Building blocks of an actual computer

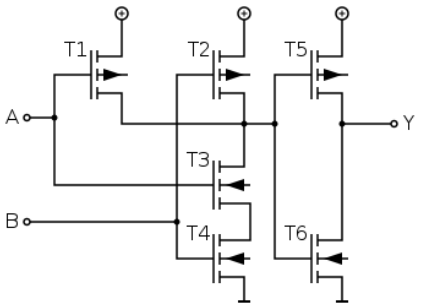
The transistor

- Computers can be build out of any *bistable device*.
- Today's "microprocessor revolution" was made possible by the invention of the *transistor*.
  - First patent in *1925* by Julius Edgar Lilienfeld.
  - First construction in *1934* by Oskar Heil
  - Different model in *1945* by Herbert F. Mataré, Heinrich Welker (FET)
  - In the same year by William B. Shockley and Walter H. Brattain (JFET) ($\rightarrow$ they receive the *Noble Price* in *1956*, so don't take Nobel prices too serious...)
- The name "transistor" is a concatenation of ***tran****sfer re****sistor***

Using transistors for logical gates

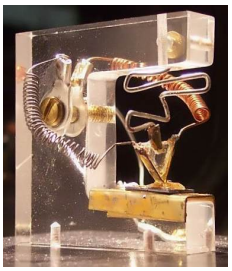
- Transistors can be used to implement the basic logical gates

Example 3 (AND). Schematic of an AND gate with *Metal Oxide Semiconductor Field Effect Transistors* (MOSFET)

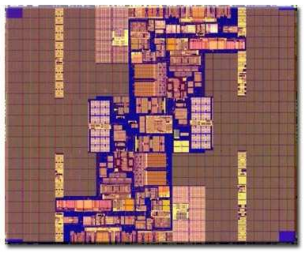


State of the art

Comparison



Copy of the first transistor by Shockley, Brattain



Die of the Itanium 2 Montecito

- A Quad-core Itanium Tuwika has about *2 billion* transistors (state-of-the-art in mid 2010)

4 Applying Boolean algebra

4.1 Translating the truth table

Step 1 – Translating the truth table

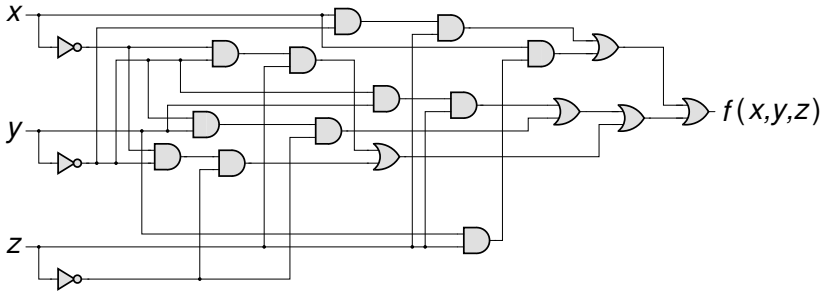
x	y	z	$f(x,y,z)$	
0	0	0	*1*	$\leftarrow$
0	0	1	*1*	$\leftarrow$
0	1	0	*1*	$\leftarrow$
0	1	1	*1*	$\leftarrow$
1	0	0	0	
1	0	1	*1*	$\leftarrow$
1	1	0	0	
1	1	1	*1*	$\leftarrow$

$$f(x,y,z) \;=\; x'y'z' + x'y'z + x'yz' + x'yz + xy'z + xyz$$

Next step – Transcribing the formula?

$$f(x,y,z) \;=\; x'y'z' + x'y'z + x'yz' + x'yz + xy'z + xyz$$

Immediate translation



- We used *20 gates*. We can do better…

Step 2 – Reducing the formula

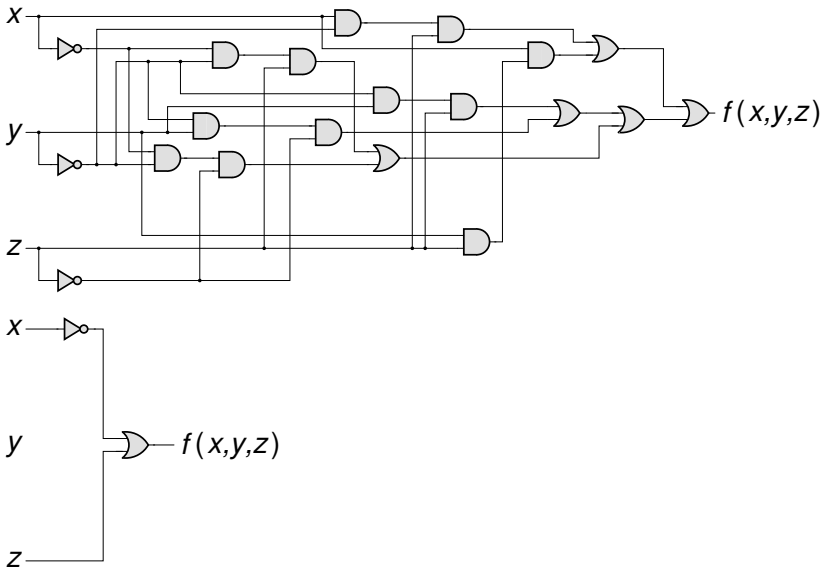
Using the rules and axioms of Boolean algebra:

$$
\begin{aligned}
f(x,y,z) &= x'y'z' + x'y'z + x'yz' + x'yz + xy'z + xyz \\
f(x,y,z) &= (x'y'z' + x'y'z) + (x'yz' + x'yz) + (xy'z + xyz) \\
f(x,y,z) &= ((x'y')z' + (x'y')z) + ((x'y)z' + (x'y)z) + ((xz)y' + (xz)y) \\
f(x,y,z) &= ((x'y' + x'y')(z' + z)) + ((x'y + x'y)(z' + z)) + \\
&\quad ((xz + xz)(y' + y)) \\
f(x,y,z) &= (x'y')1 + (x'y)1 + (xz)1 \\
f(x,y,z) &= x'y' + x'y + xz \\
f(x,y,z) &= x'(y' + y) + xz \\
f(x,y,z) &= x' + xz \\
f(x,y,z) &= (x' + x)(x' + z) \\
f(x,y,z) &= 1(x' + z) \\
f(x,y,z) &= x' + z
\end{aligned}
$$

Step 3 – Transcribing the formula

$$f(x,y,z) = x'y'z' + x'y'z + x'yz' + x'yz + xy'z + xyz f(x,y,z) = x' + z$$

Transcription before*after* algebraic transformation



- Now we used *2 gates*. That's better.

Summary and observations

- Algebraic expressions can be *simplified* through transformations
- Simplified expressions have multiple *advantages*
  - fewer *building blocks*
  - fewer *crossings*
- Having a *systematic* approach would be nice...

4.2 Systematic reduction of components

A systematic approach

- The form of the expression transcribed from the truth table is called *disjunctive normal form*.

$$f(x,y,z) \;=\; x'y'z' + x'y'z + x'yz' + x'yz + xy'z + xyz$$

- In general, for n inputs $x_1, \ldots, x_n$, the disjunctive normal form is

$$f(x_1,\ldots,x_n) \;=\; x_1^{s_{1,1}} * \cdots * x_n^{s_{1,n}} + \cdots + x_1^{s_{m,1}} * \cdots * x_n^{s_{m,n}}$$

  - Where m is the number of "1s" in the truth table, and
  - $s_{i,j}$ is the "sign" of input x_j in conjunct i

A systematic approach

Idea:

$$f(x,y,z) \;=\; x'y'z' + x'y'z + x'yz' + x'yz + xy'z + xyz$$

- Search for conjuncts that *vary in only one component*
- Apply:
  - Associativity: $x'y'z' + x'y'z = (x'y')z' + (x'y')z$
  - Distributivity: $(x'y')z' + (x'y')z = ((x'y') + (x'y')) * (z + z')$
  - $x + x' = 1$, $x + x = x$: $(x'y') * 1$
  - $x * 1 = x$: $x'y'$
- *Replace* the old conjuncts by the new one

$$f(x,y,z) \;=\; x'y' + x'yz' + x'yz + xy'z + xyz$$

- *Repeat* until no further reductions can be found

A systematic approach

Example 4.

$$\begin{aligned}
f(x,y,z) &= x'y'z' + x'y'z + x'yz' + x'yz + xy'z + xyz \\
f(x,y,z) &= x'y' + x'yz' + x'yz + xy'z + xyz \\
f(x,y,z) &= x'y' + x'y + xy'z + xyz \\
f(x,y,z) &= x'y' + x'y + xz \\
f(x,y,z) &= x' + xz \\
f(x,y,z) &= x' + z
\end{aligned}$$

Even more structured approaches include:

- The *Quine-McCluskey algorithm*
- The *Veitch diagram* or *Karnaugh map*