

Logic

Logical Operators

Symbol	Meaning
\sim	Not
$\&$	Binary And
$\&\&$	Logical And
$ $	Binary Or
$ $	Logical Or
\ll	Left Shift ($\times 2^n$)
\gg	Right Shift (floor division: $\frac{num}{2^n}$)

Logical operators act on statements / prepositions, **Bit-wise** operators act on bit strings.

Karnaugh / K - Maps

Quick way to develop a logical statement that encapsulates a truth table.

1. **Change one bit at a time**
2. only fill cells where output is **1 / T**
3. **group** into powers of 2
4. **remove *unstable* variables** (change w/in the group)
5. **result**
 1. **sum of products (SOP)** like $F = BC + AB + AC$
 2. **product of sums (POS)**
 1. uses 0s instead of 1s
 2. result like $F = (A + B) * (B + C)$

Adders

Half-Adder (HA)

Boolean expressions represent a *sum* (s) and *carry* (cout).

Half-Adders use **XOR** and **AND** gates to “add” two bits.

Full-Adder (FA)

To add more than two bits, we need a full-adder, made from two half-adders.

Inputs: a, b, and cin

Outputs: sum (s) and carry out (cout)

Symbols for logic gates


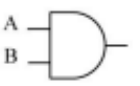

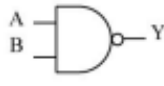


	Truth Table	Gate Symbol	Boolean	Verilog															
NOT	<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	Y	0	1	1	0		$Y = \overline{A}$	<code>assign y = ~a;</code>									
A	Y																		
0	1																		
1	0																		
AND	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1		$Y = A \wedge B$	<code>assign y = a & b;</code>
A	B	Y																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
OR	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1		$Y = A \vee B$	<code>assign y = a b;</code>
A	B	Y																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
NAND	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0		$Y = \overline{A \wedge B}$	<code>assign y = ~(a & b);</code>
A	B	Y																	
0	0	1																	
0	1	1																	
1	0	1																	
1	1	0																	
NOR	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0		$Y = \overline{A \vee B}$	<code>assign y = ~(a b);</code>
A	B	Y																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	0																	
XOR	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0		$Y = A \oplus B$	<code>assign y = a ^ b;</code>
A	B	Y																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	

Figure 1: image

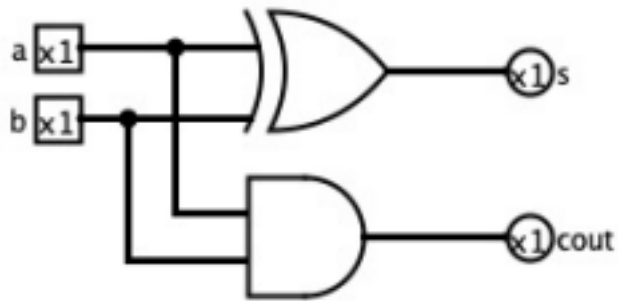


Figure 2: image

Inputs/Ouputs					K-Maps	
inputs			outputs			
a	b	cin	s	cout		
0	0	0	0	0		
0	0	1	1	0		
0	1	0	1	0		
0	1	1	0	1		
1	0	0	1	0		
1	0	1	0	1		
1	1	0	0	1		
1	1	1	1	1		

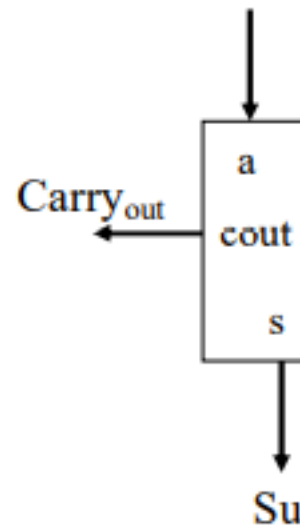
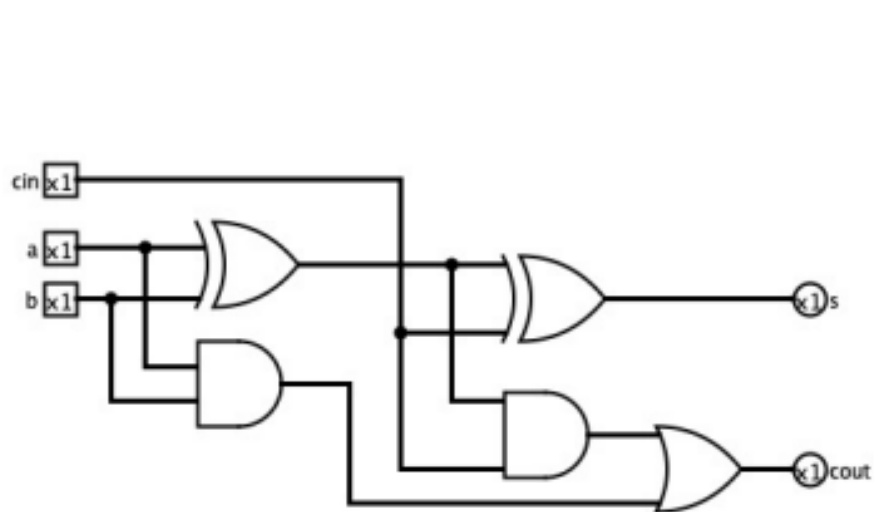
		cin	
		0	1
ab	00		1
	01	1	
	11		1
	10	1	

sum (s) k-map

		cin	
		0	1
ab	00		
	01		
	11		
	10		

co

Full Adder



$$s = (\neg a \wedge \neg b \wedge \text{cin}) \vee (\neg a \wedge b \wedge \neg \text{cin}) \vee (a \wedge \neg b \wedge \neg \text{cin}) \\ \vee (a \wedge b \wedge \text{cin})$$

$$s = a \oplus b \oplus \text{cin} \text{ (simplified)}$$

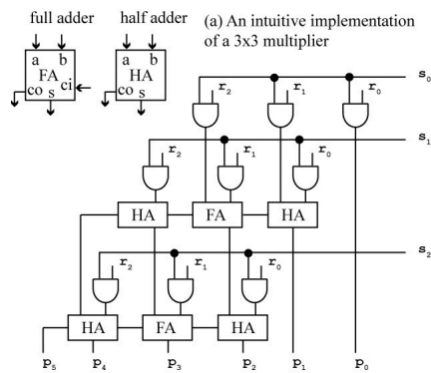
$$\text{cout} = (a \wedge b) \vee (a \wedge \text{cin}) \vee (b \wedge \text{cin})$$

Multiplication

Binary	Decimal
1 1 1	7
x 1 0 1	x 5
-----	-----
1 1 1	35
0 0 0	
+ 1 1 1	

1 0 0 0 1 1	= 35

Simple **AND** gate used to copy the top if the bottom is a **1**. Then, we use a combination of HAs and FAs to sum the multiplication.



Notice: we do not need FAs everywhere!