Fortunately, we can get by with a few basic gates…

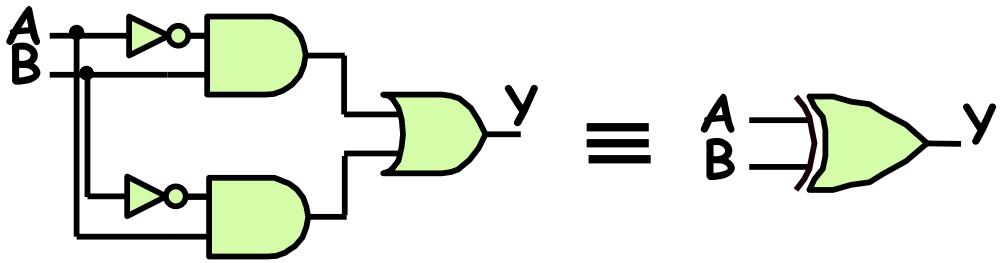
AND, OR, and NOT are sufficient… (cf Boolean Expressions):

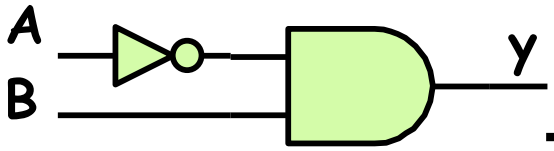
B>A

AB	Y
00	0
01	1
10	0
11	0

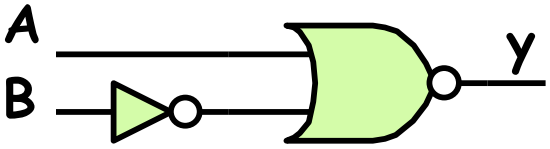
XOR

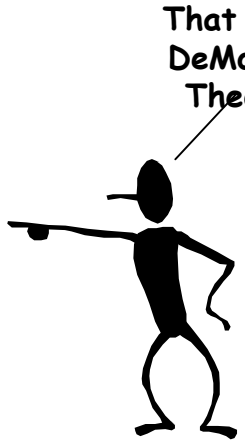
AB	Y
00	0
01	1
10	0
11	0





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





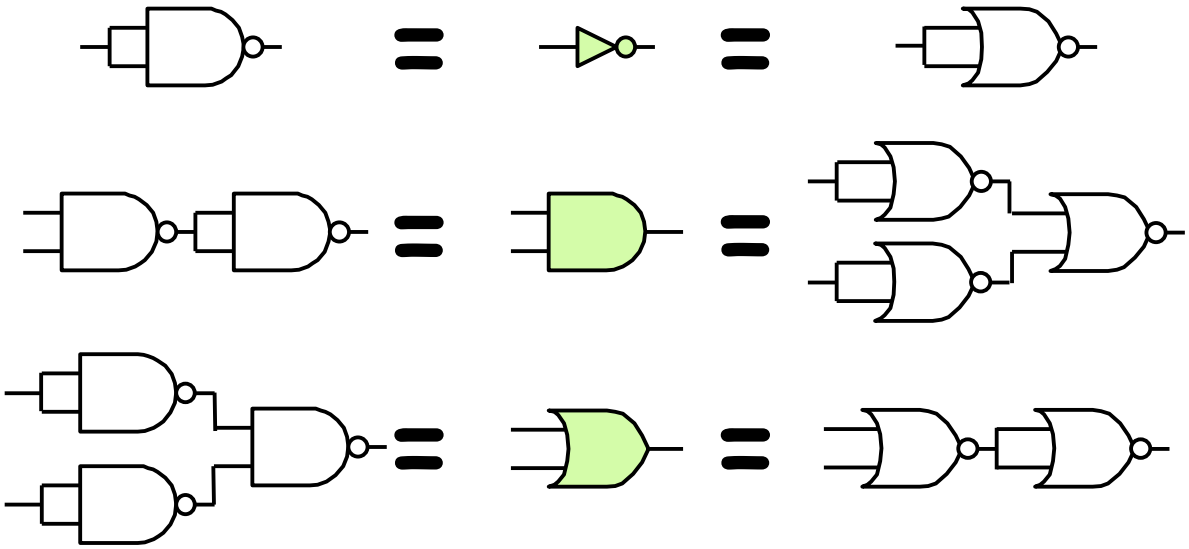
$\bar{A}\bar{B} = \overline{A + B}$

$\bar{A} + \bar{B} = \overline{AB}$

How many different gates do we *really* need?

One will do!

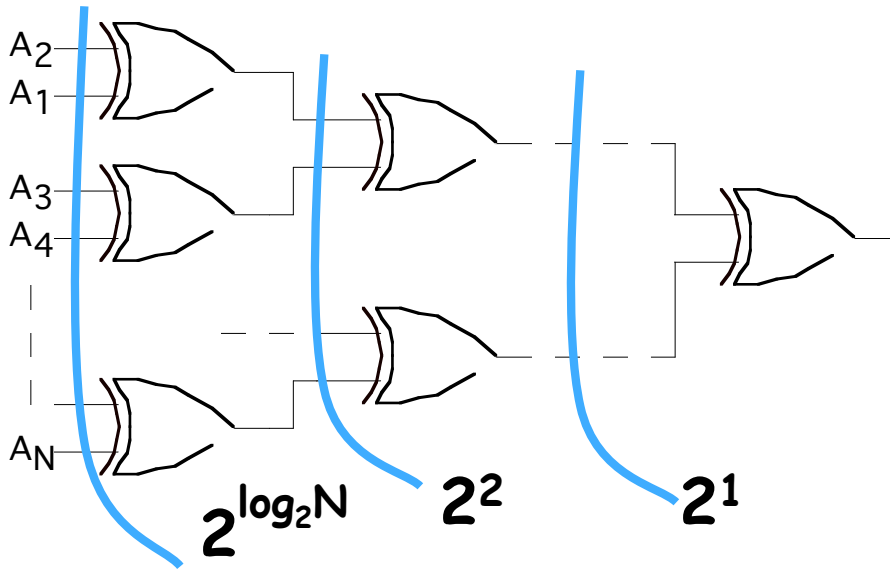
NANDs and NORs are universal:



Ah!, but what if we want more than 2 inputs?

I think that I shall never see

a circuit lovely as...



N-input TREE has O(log N) levels...

Signal propagation takes O(log N) gate delays.

Question: Can EVERY N-Input Boolean function be implemented as a tree of 2-input gates?

Here's a Design Approach

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

1) Write out our functional spec as a truth table

2) Write down a Boolean expression for every '1' in the output

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

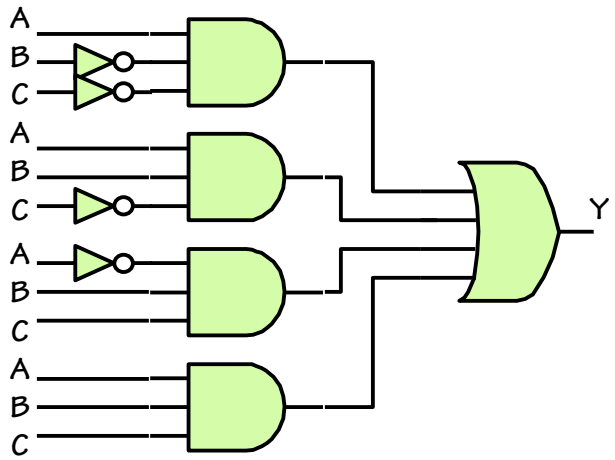
3) Wire up the gates, call it a day, and declare success!

This approach will always give us Boolean expressions in a particular form:

SUM-OF-PRODUCTS

Straightforward Synthesis

We can implement
SUM-OF-PRODUCTS
with just three levels of logic.

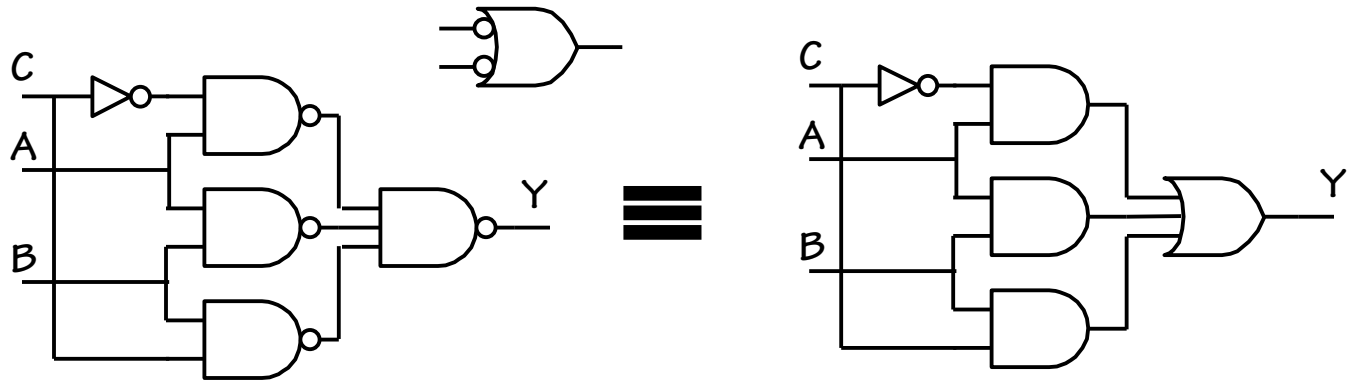


INVERTERS/AND/OR

Propagation delay --
No more than "3" gate delays
(well, it's actually O(log N) gate delays)

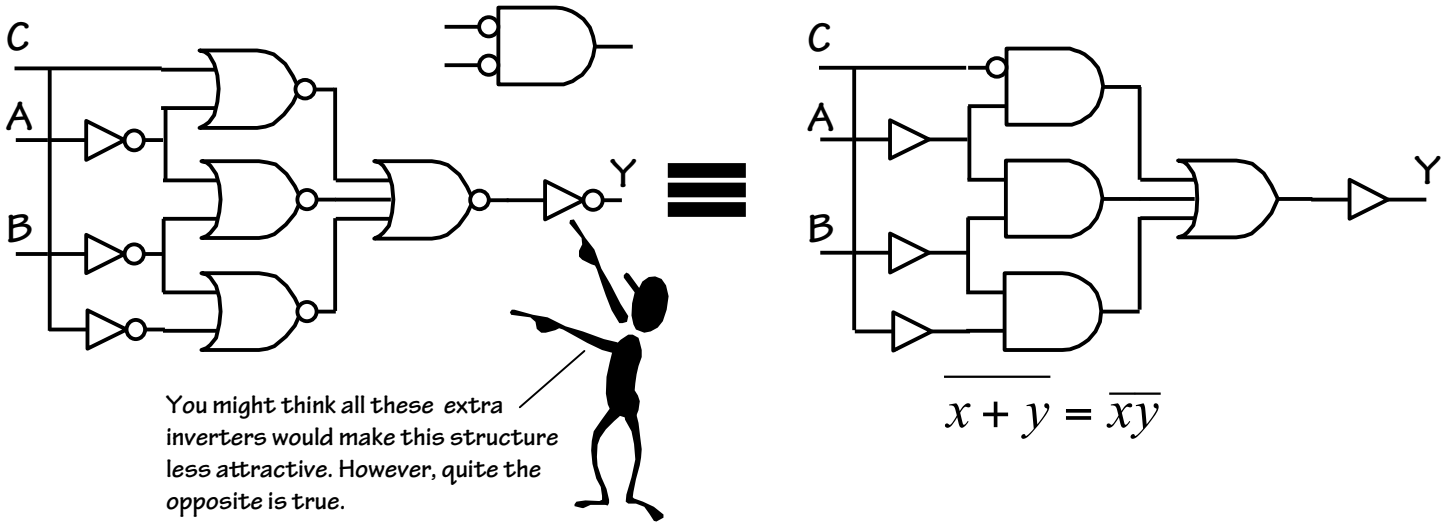
Practical SOP Implementation

- **NAND-NAND** $\overline{AB}=\bar{A}+\bar{B}$ "Pushing Bubbles"



$$\overline{xyz} = \bar{x} + \bar{y} + \bar{z}$$

- **NOR-NOR** $\bar{A}\bar{B}=\overline{A+B}$



$$\overline{x + y} = \overline{x}\,\overline{y}$$

Logic Simplification

Can we implement the same function with fewer gates? Before trying we'll add a few more tricks in our bag.

BOOLEAN ALGEBRA:

OR rules:	$a + 1 = 1,\ a + 0 = a,\ a + a = a$
AND rules:	$a1 = a,\ a0 = 0,\ aa = a$
Commutative:	$a + b = b + a,\ ab = ba$
Associative:	$(a + b) + c = a + (b + c),\ (ab)c = a(bc)$
Distributive:	$a(b+c) = ab + ac,\ a + bc = (a+b)(a+c)$
Complements:	$a + \overline{a} = 1, \quad a\overline{a} = 0$
Absorption:	$a + ab = a, \quad a + \overline{a}b = a + b$
	$a(a + b) = a, \quad a(\overline{a} + b) = ab$
Reduction:	$ab + \overline{a}b = b, \quad (a + b)(\overline{a} + b) = b$
DeMorgan's Law:	$\overline{a} + \overline{b} = \overline{ab}, \quad \overline{a}\overline{b} = \overline{a + b}$

Boolean Minimization:

An Algebraic Approach

Can't he come up with a *new* example???

Lets (again!) simplify

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

Using the identity

$$\alpha A + \alpha \overline{A} = \alpha$$

For any expression α and variable A:

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

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

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

Hey, I could write A *program* to do That!

Karnaugh Maps: A Geometric Approach

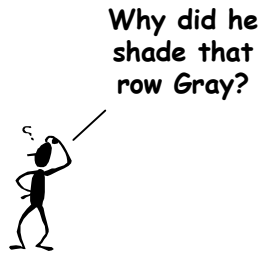
K-Map: a truth table arranged so that terms which differ by exactly one variable are adjacent to one another so we can see potential reductions easily.

Here's the layout of a 3-variable K-map filled in with the values from our truth table:

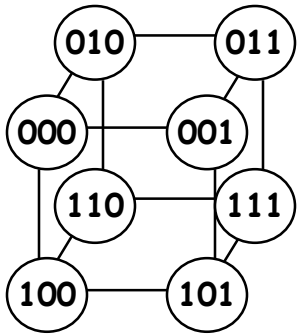
Truth Table

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

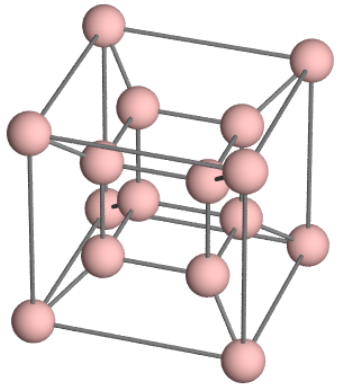
C\AB	00	01	11	10
0	0	0	1	1
1	0	1	1	0



It's cyclic. The left edge is adjacent to the right edge. (It's really just a flattened out cube).



On to Hyperspace



4-variable K-map for a multipurpose logic gate:

$$Y = \begin{cases} A \cdot B & \text{if } CD = 00 \\ A + B & \text{if } CD = 01 \\ \overline{B} & \text{if } CD = 10 \\ A \oplus B & \text{if } CD = 11 \end{cases}$$

CD\AB	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	0	1	0	1
10	1	0	0	1

Again it's cyclic. The left edge is adjacent to the right edge, and the top is adjacent to the bottom.

Finding Subcubes

We can identify clusters of "irrelevent" variables by circling adjacent subcubes of 1s. A subcube is just a lower dimensional cube.

C\AB	00	01	11	10
0	0	0	1	1
1	0	1	1	0

\AB CD\	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	0	1	0	1
10	1	0	0	1

The best strategy is generally a greedy one.

- Circle the largest N-dimensional subcube (2^N adjacent 1's)
  4x4, 4x2, 4x1, 2x2, 2x1, 1x1
- Continue circling the largest remaining subcubes (even if they overlap previous ones)
- Circle smaller and smaller subcubes until no 1s are left.

Write Down Equations

Write down a product term for the portion of each cluster/subcube that is invariant. You only need to include enough terms so that all the 1's are covered. Result: a minimal sum of products expression for the truth table.

C\AB	00	01	11	10
0	0	0	1	1
1	0	1	1	0

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

We're done!

\AB CD\	00	01	11	10
00	0	0	1	0
01	0	1	1	1
11	0	1	0	1
10	1	0	0	1

$Y = AB\bar{C} + \bar{A}BD + A\bar{B}D + \bar{B}C\bar{D}$

Recap: K-map Minimization

1) Copy truth table into K-Map
2) Identify subcubes,

   selecting the largest available subcube at each step, even if it involves some overlap with previous cubes, until all ones are covered. (Try: 4x4, 2x4 and 4x2, 1x4 and 4x1, 2x2, 2x1 and 1x2, finally 1x1)
3) Write down the minimal SOP realization

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

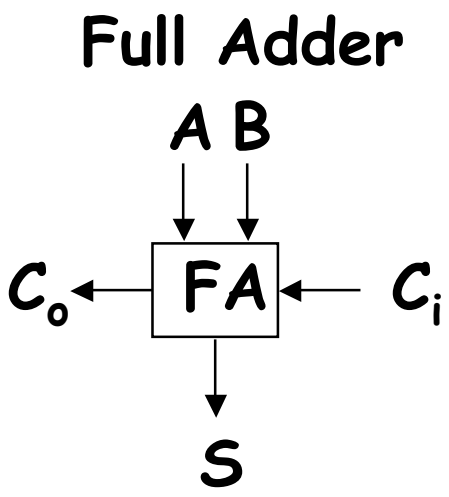
JARGON: The circled terms are called *implicants*. An implicant not completely contained in another implicant is called a *prime implicant*.

C\BA	00	01	11	10
0	0	1	1	0
1	0	0	1	1

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

Logic that defies SOP simplification

C_i	A	B	S	C_o
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



S

C/AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

C_O

C/AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$S = A\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + ABC$$

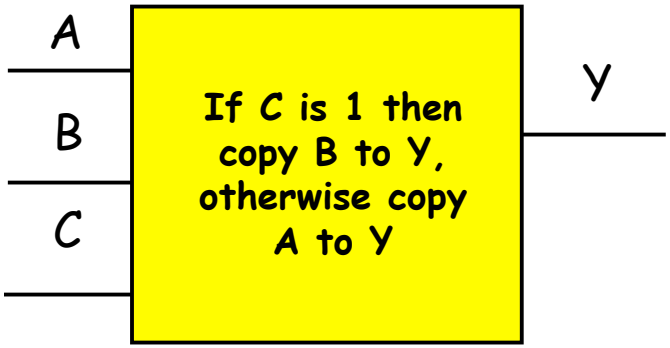
$$C_o = AB\bar{C} + A\bar{B}C + \bar{A}BC + ABC$$

Can simplify the carry out easily enough, eg...

$$C_o = BC + AB + AC$$

But, the sum, S, doesn't have a simple sum-of-products implementation even though it can be implemented using only two 2-input XOR gates.

Logic Synthesis Using MUXes

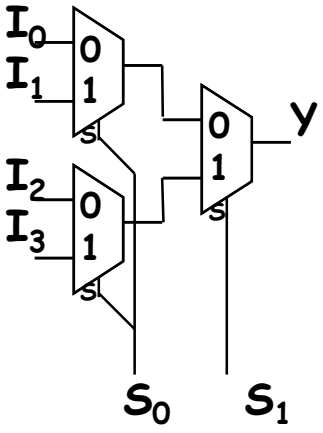


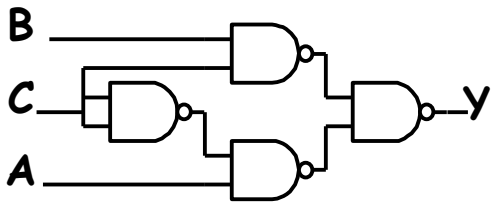
2-input Multiplexer

Truth Table

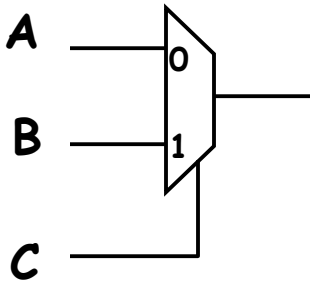
C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

A 4-input Mux implemented as a tree





schematic



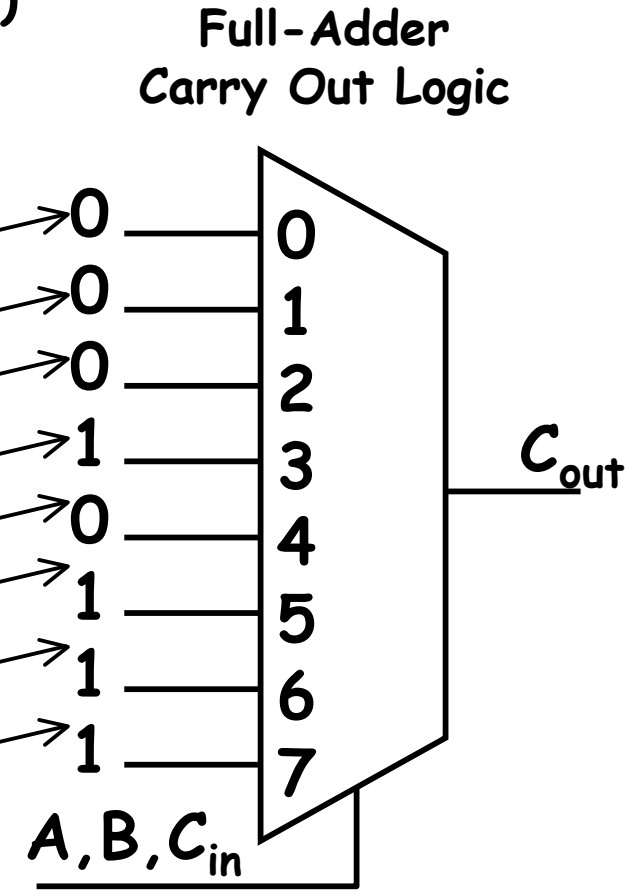
Gate symbol

Systematic Implementation of Combinational Logic

Consider implementation of some arbitrary Boolean function, F(A,B)

... using a MULTIPLEXER as the only circuit element:

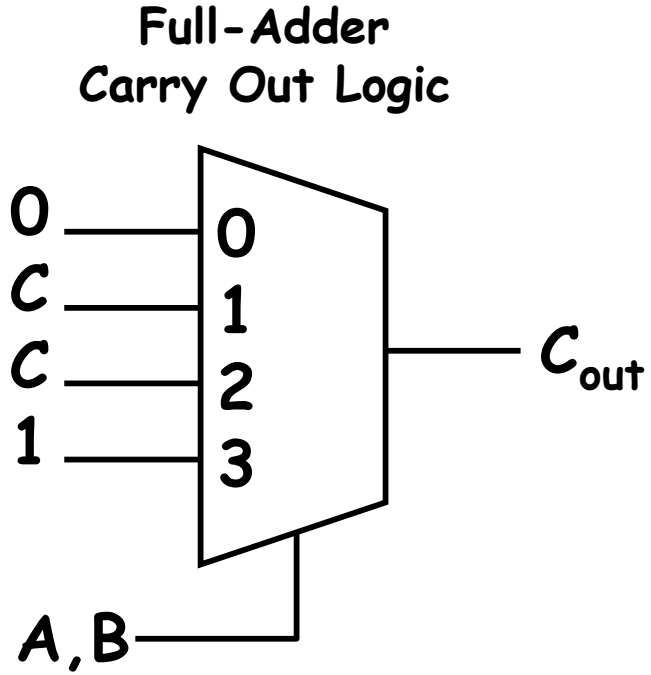
A	B	C_{in}	C_{out}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



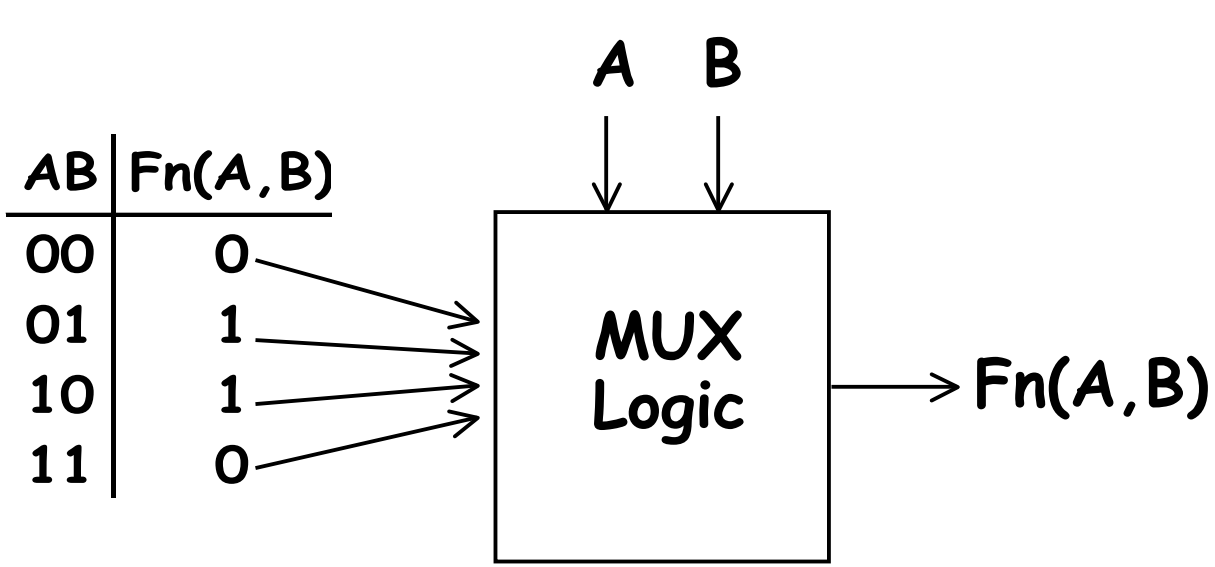
Systematic Implementation of Combinational Logic

Same function as on previous slide, but this time let's use a 4-input mux

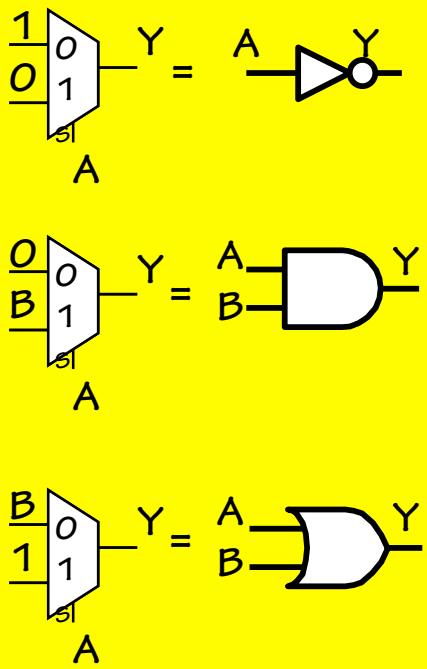
A	B	C_{in}	C_{out}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



General Table Lookup Synthesis



Muxes are UNIVERSAL!



In future technologies muxes might be the "natural gate".

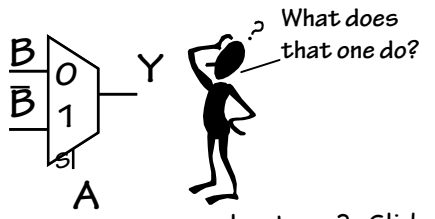
Generalizing:

In theory, we can build any 1-output combinational logic block with multiplexers.

For an N-input function we need a 2^N input mux.

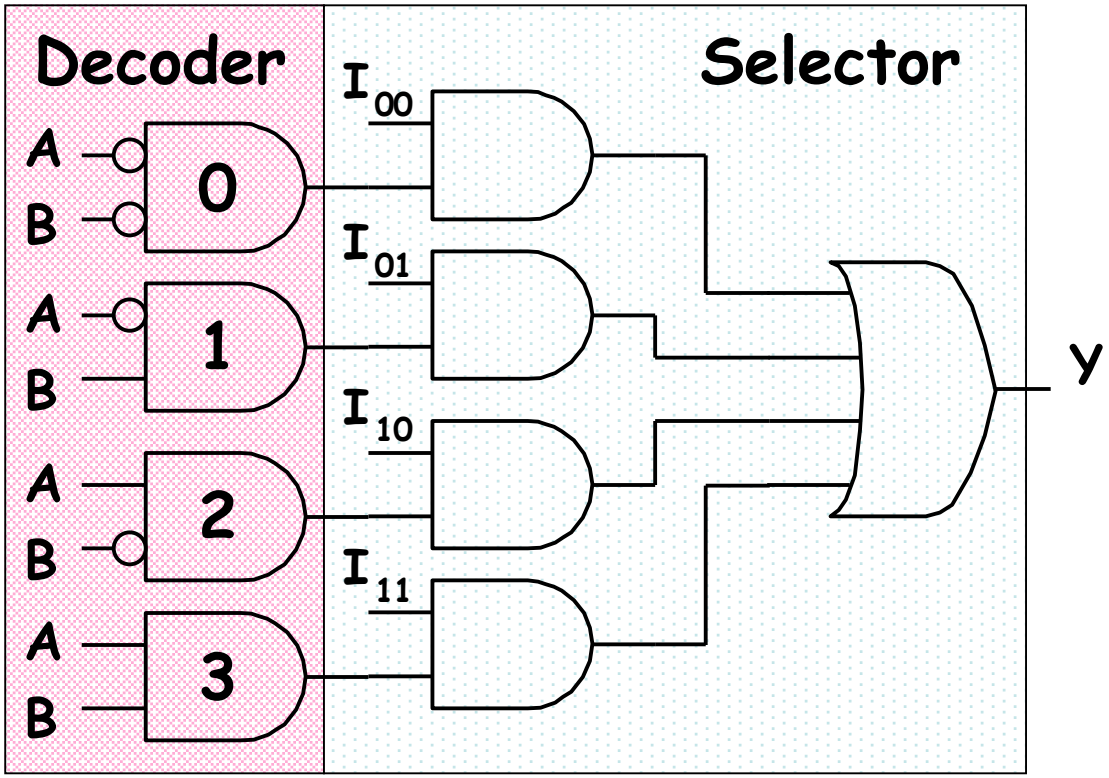
BIG Multiplexers?

How about 10-input function? 20-input?



A Mux's Guts

A decoder generates all possible product terms for a set of inputs



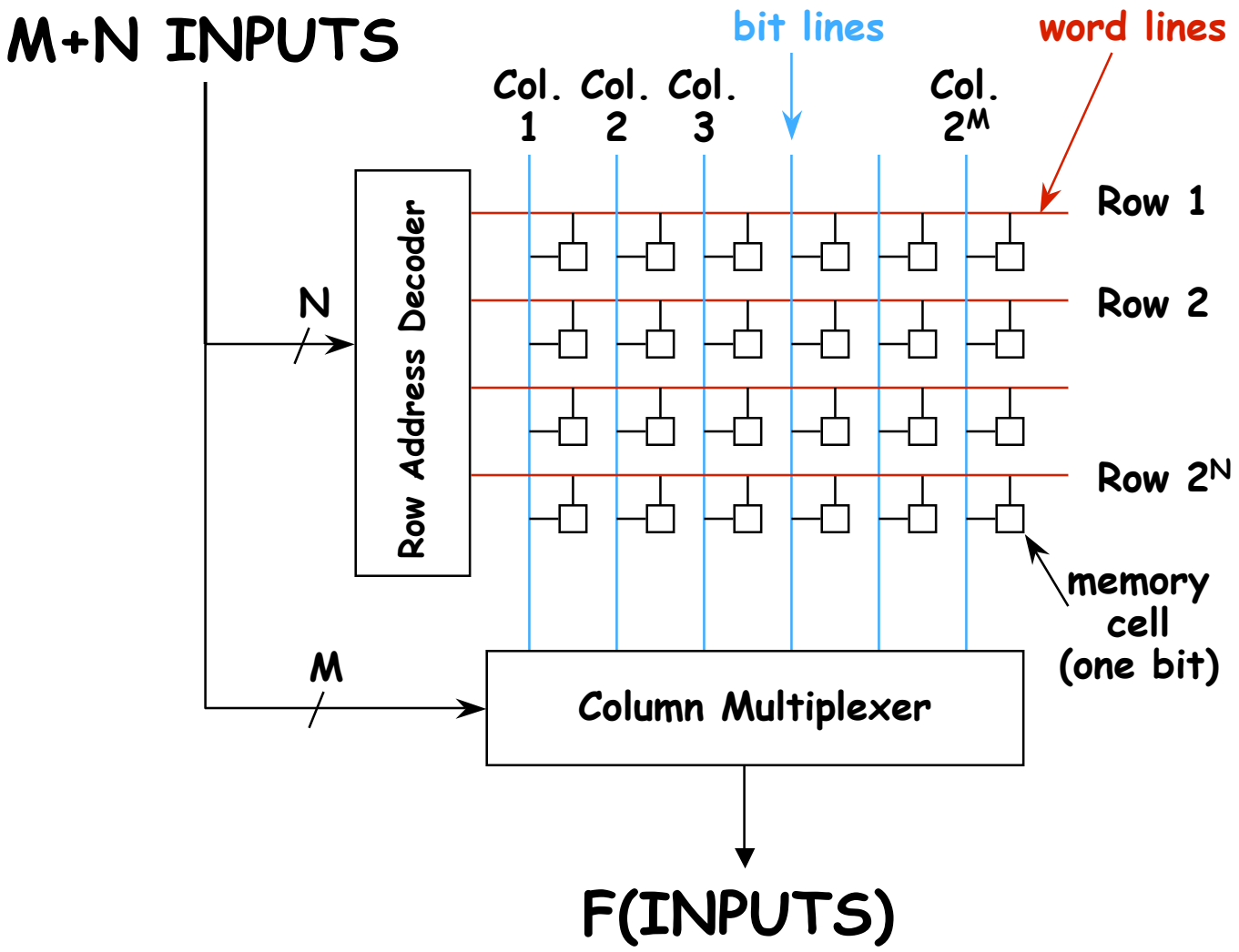
Multiplexers can be constructed into two sections:

A DECODER that identifies the desired input, and

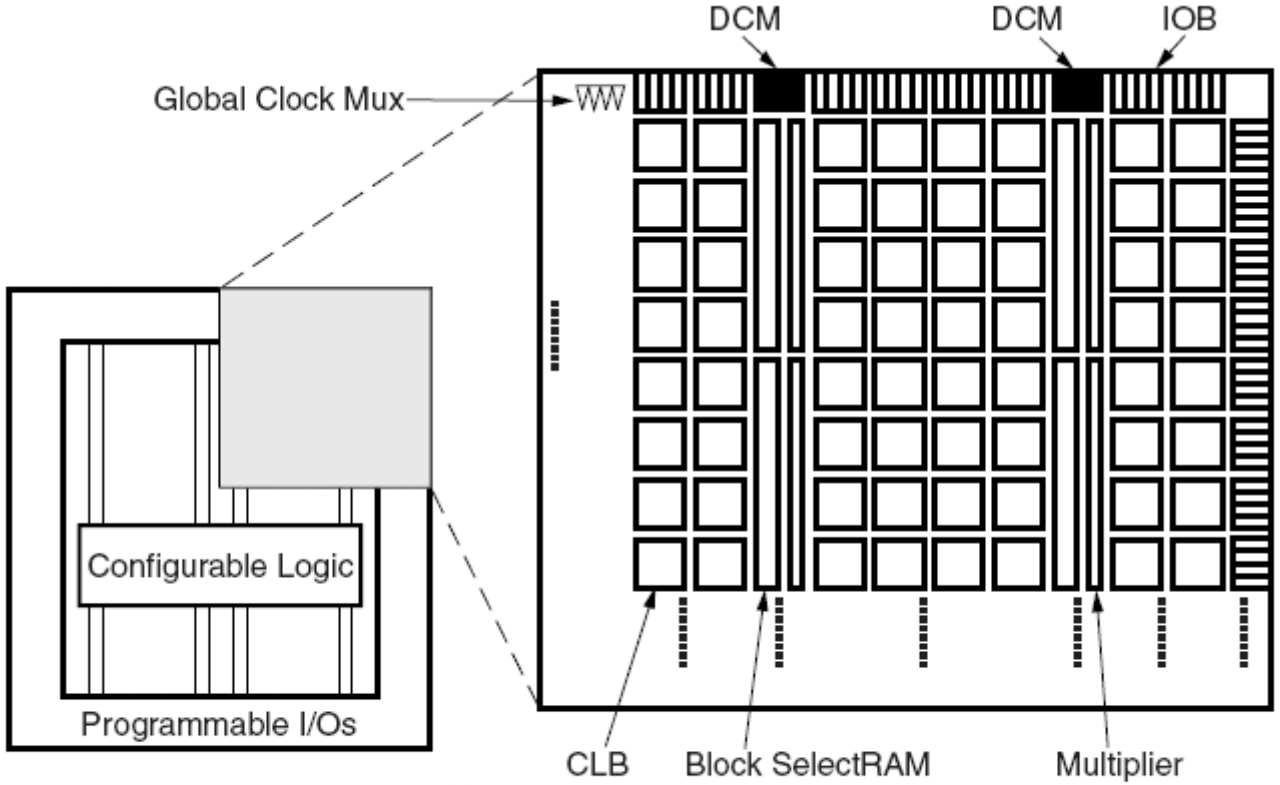
a SELECTOR that enables that input onto the output.

Hmmm, by sharing the decoder part of the logic MUXs could be adapted to make lookup tables with any number of outputs

Using Memory as a Programmable Logic Device



Xilinx Virtex II FPGA

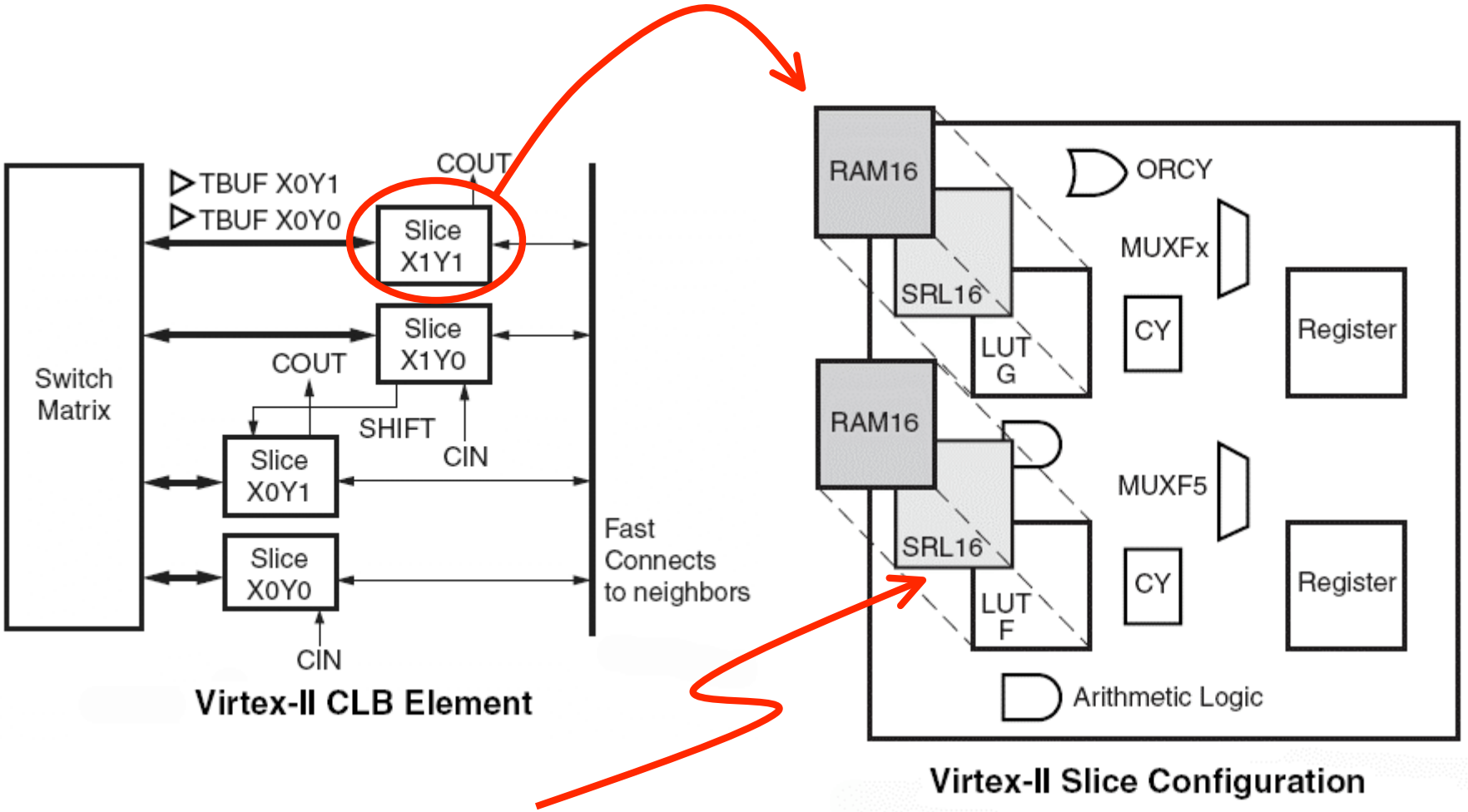


Virtex-II Architecture Overview

XC2V6000:

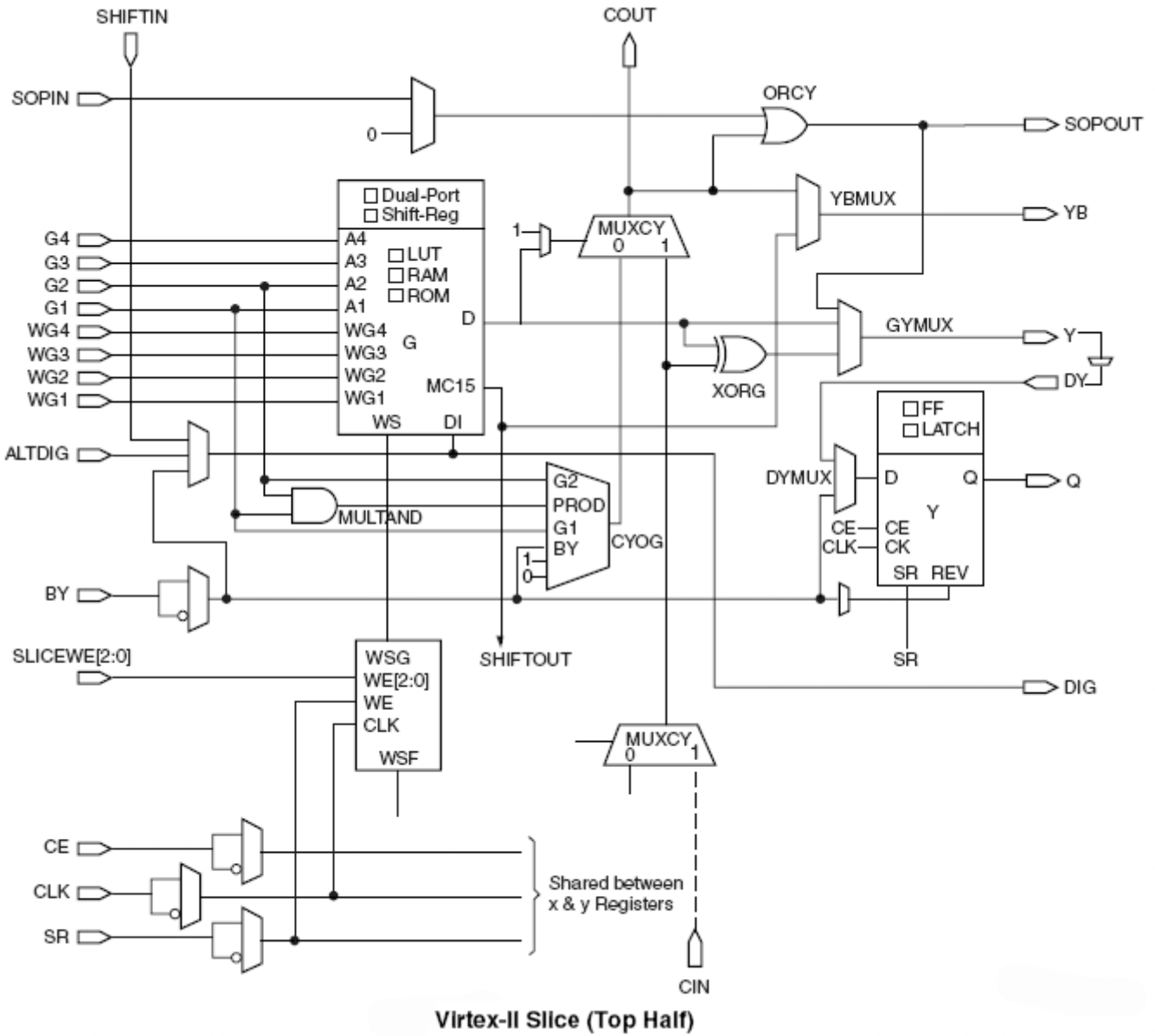
- 957 pins, 684 IOBs
- CLB array: 88 cols x 96/col = 8448 CLBs
- 18Kbit BRAMs = 6 cols x 24/col = 144 BRAMs = 2.5Mbits
- 18x18 multipliers = 6 cols x 24/col = 144 multipliers

Virtex II CLB



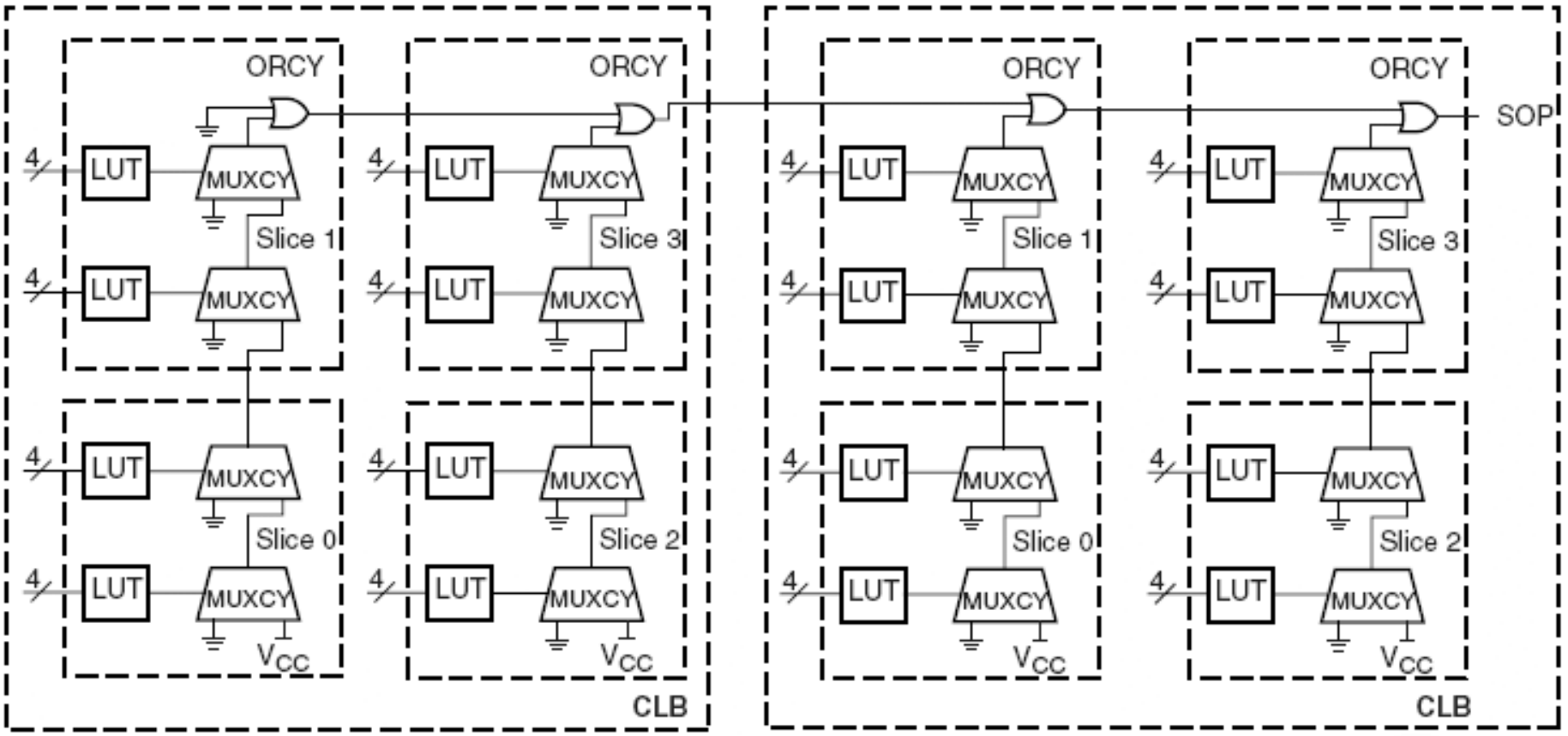
16 bits of RAM which can be configured as a 16x1 single- or dual-port RAM, a 16-bit shift register, or a 16-location lookup table

Virtex II Slice Schematic



Virtex-II Slice (Top Half)

Virtex II Sum-of-products



Horizontal Cascade Chain