UNIVERSITY of WASHINGTON

CSS 422 Hardware and Computer Organization

Combinational Circuit

CPU Components and ALU Design

Ver. 3.1

Professor: Professor: Yang Peng

Topics

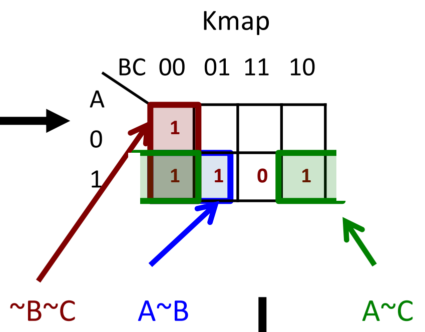
- Karnaugh Map: more simplification techniques
- Digital Components
  - Multiplexer
  - Decoder
  - Comparator
- ALU Design
  - Half/Full Adders and Subtraction
  - Multiplier
  - ALU

Berger: Ch 1, 2, and 3

Null: Ch3

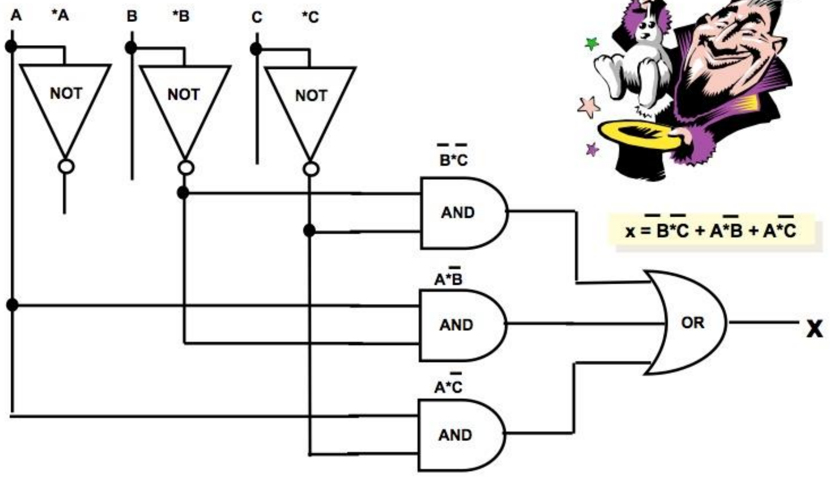
Boolean Equation Simplification K-Map vs. Algebraic Laws

A	B	C	x
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



$x = \overline{B}*\overline{C} + A*\overline{B} + A*\overline{C}$

Simplified equation



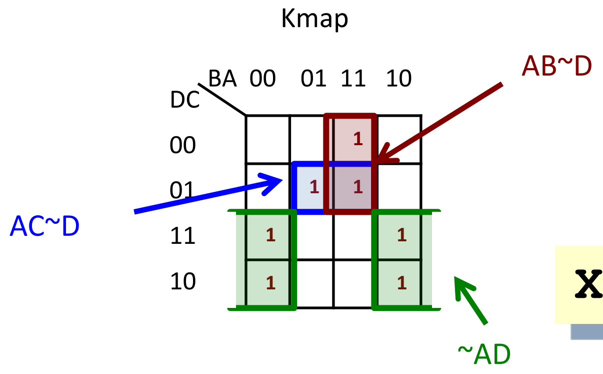
Example: 4-Input Circuit Simplification

A	B	C	D	X
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
1	1	0	0	1
0	0	1	0	0
1	0	1	0	1
0	1	1	0	0
1	1	1	0	1
0	0	0	1	1
1	0	0	1	0
0	1	0	1	1
1	1	0	1	0
0	0	1	1	1
1	0	1	1	0
0	1	1	1	1
1	1	1	1	0

1. X = A*B*~C*~D + A*~B*C*~D + A*B*C*~D + ~A*~B*~C*D + ~A*B*~C*D + ~A*~B*C*D + ~A*B*C*D

2. X = A*B*~D*(~C + C) + A*~B*C*~D + ~A*~C*D* (~B+ B) + ~A*C*D*(B + ~B)

3. X = A*B*~D + A*~B*C*~D + ~A*D*(C + ~C)
4. X = A*B*~D + A*~B*C*~D + ~A*D
5. X = A*~D*(B + ~B*C) + ~A*D
6. X = A*~D*(C + B) + ~A*D //Second law of Dist.



X = ~A*D + A*B*~D +A*C*~D

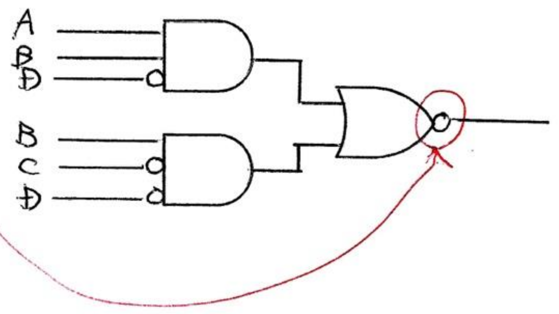
More Karnaugh Map Technique
Use of Negation

- If you find many 1s, ignore those 1s, instead, group 0s.
- Add a negation of the sum (= OR)

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

$$\bar{f} = AB\bar{D} + B\bar{C}\bar{D}$$

$$f = \overline{AB\bar{D} + B\bar{C}\bar{D}}$$



More Karnaugh Map Technique

Exclusive OR

- If a Kmap shows a **lattice with 0 at the upper left cell**, use an exclusive OR.

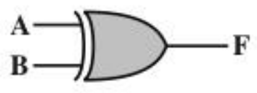
truth table

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

A \ B	0	1
0	0	1
1	1	0

Kmap

$F_1 = A\sim B + \sim A\ B$
$= A \oplus B$



A \ B	0	1
0	1	0
1	0	1

Kmap

$F_2 = \sim F$
$= \sim(A \oplus B)$

A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Kmap

$F_3 = A \oplus B \oplus C$

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

Kmap

$F_4 = A \oplus B \oplus C \oplus D$

More Karnaugh Map Technique

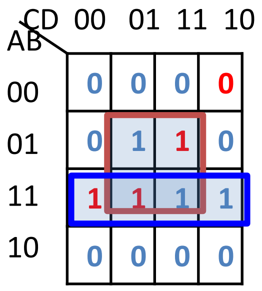
Don’t Care Terms

- If some products (outputs) can be either 0 or 1, **you can choose 0 or 1 as you create larger groups – We don’t care the logic value of these outputs.**

A	B	C	D	P
0	0	0	0	0
0	0	0	1	0
0	0	1	0	X
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	X
1	1	0	1	X
1	1	1	0	1
1	1	1	1	1

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

Kmap



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

Kmap

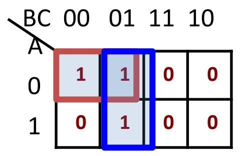
F = AB + BD

How to Simplify Product of Sums

- Simplification Steps
  1. Negate a given product-of-sums form.
  2. Apply De Morgan's law to the negation.
  3. Draw the corresponding truth table and Kmap.
  4. Get a simplified logic.
  5. Re-negate it back.

- Example
  1. F = (A + B + C)(B + ~C)
  2. ~F = ~{(A + B + C)(B + ~C)}
     = ~(A + B + C) +~(B + ~C)
     = ~A~B~C + ~BC
  3. Truth table and Kmap

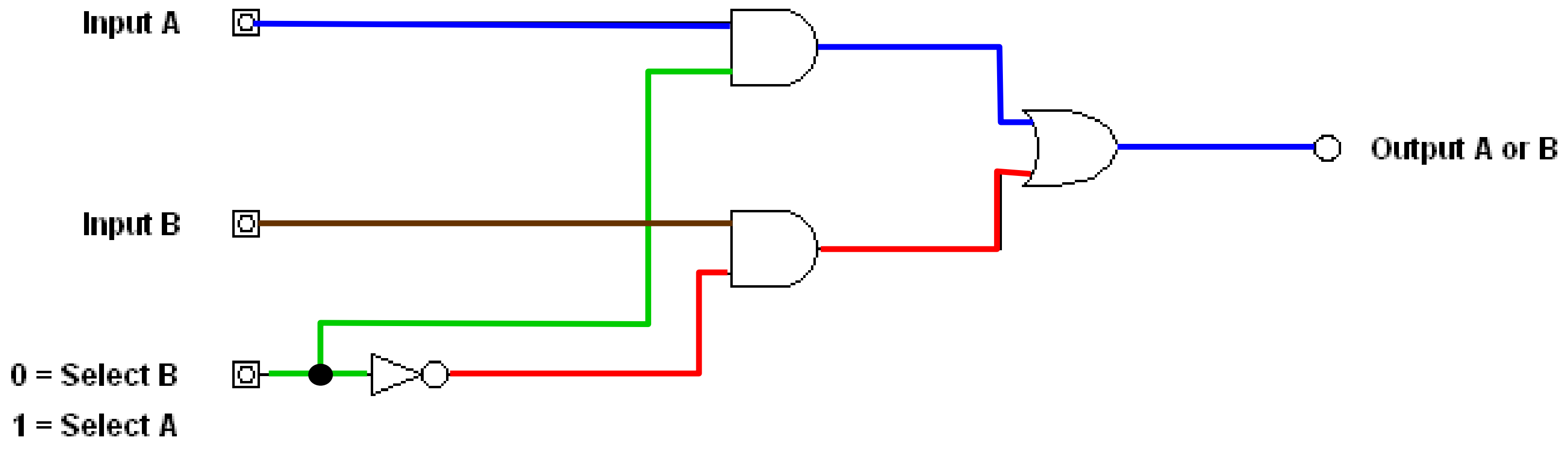
A	B	C	P
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



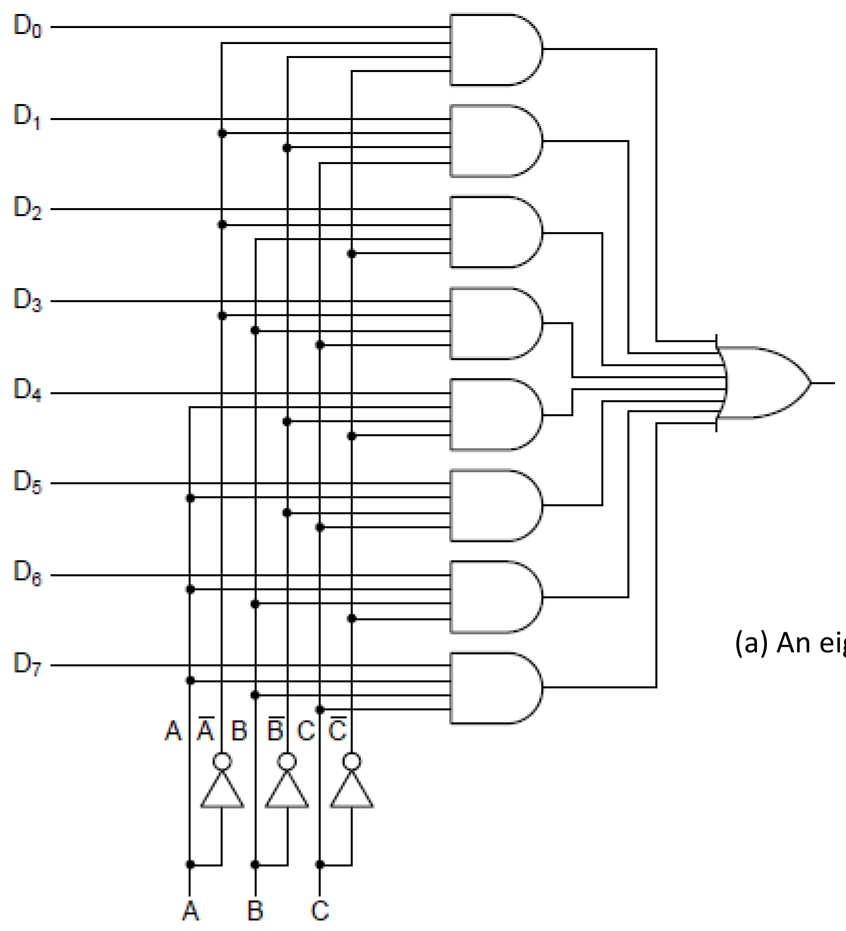
  4. ~ F = ~A~B +~BC
  5. F = ~(~A~B + ~BC)
     = ~(~A~B)~(~BC)
     = (A + B)(B + ~C)

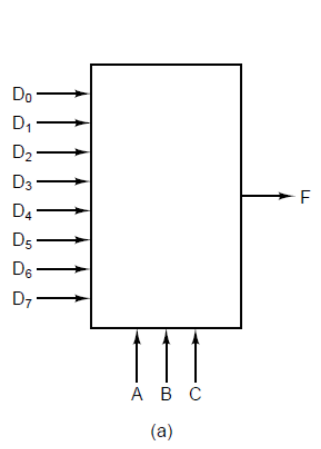
Simple Multiplexer (MUX)

- The MUX circuit allows one of two (or more) logical signals to be selected

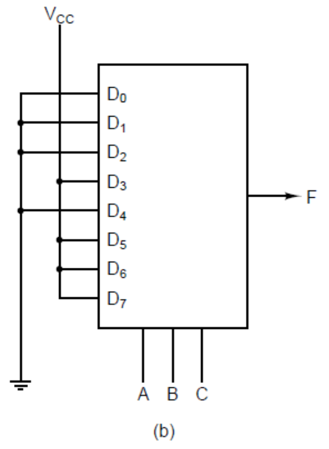


Multiplexers





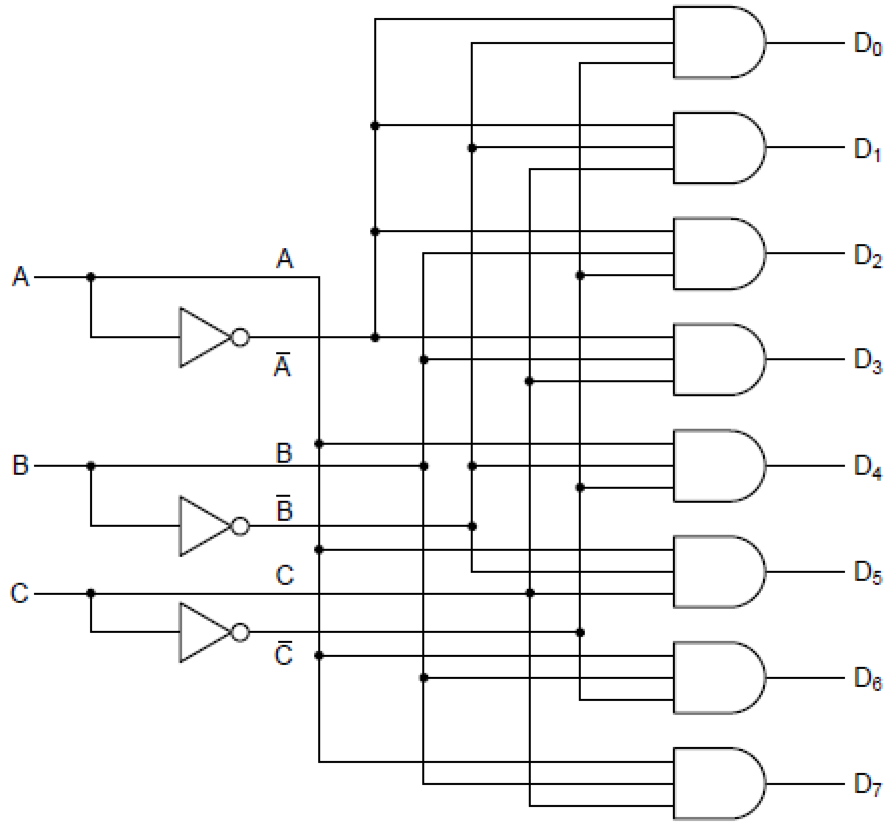
(a) An eight-input multiplexer.



(b) The same multiplexer wired to compute the majority function.

A, B, C controls which of the eight input lines to be gated.

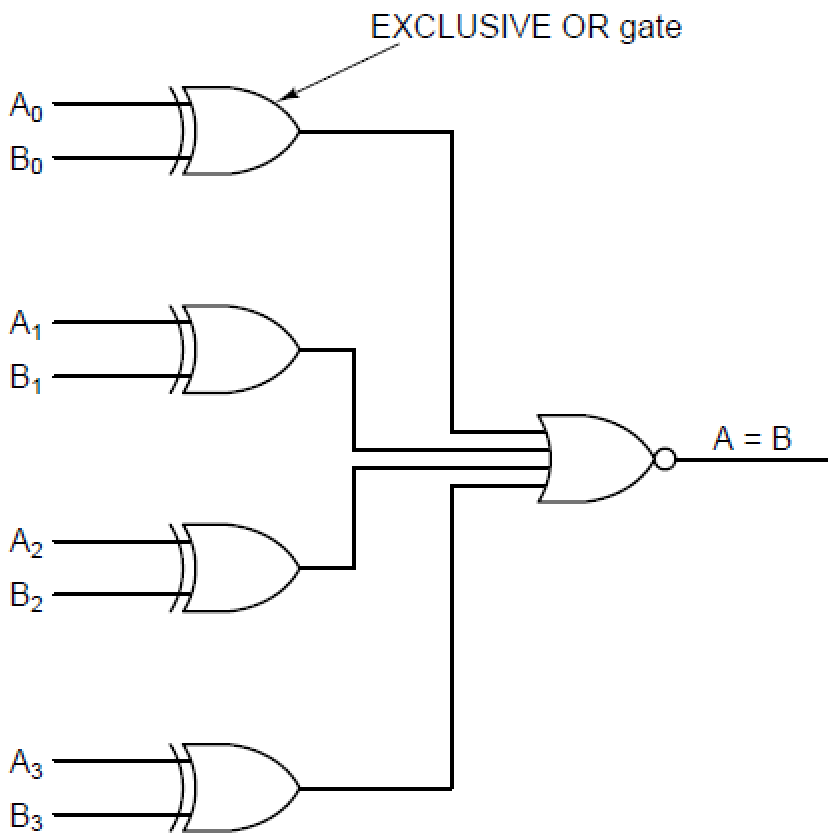
Decoder



A 3-to-8 decoder circuit: Depending on the inputs, only one of eight outputs is 1.

UNIVERSITY *of* WASHINGTON

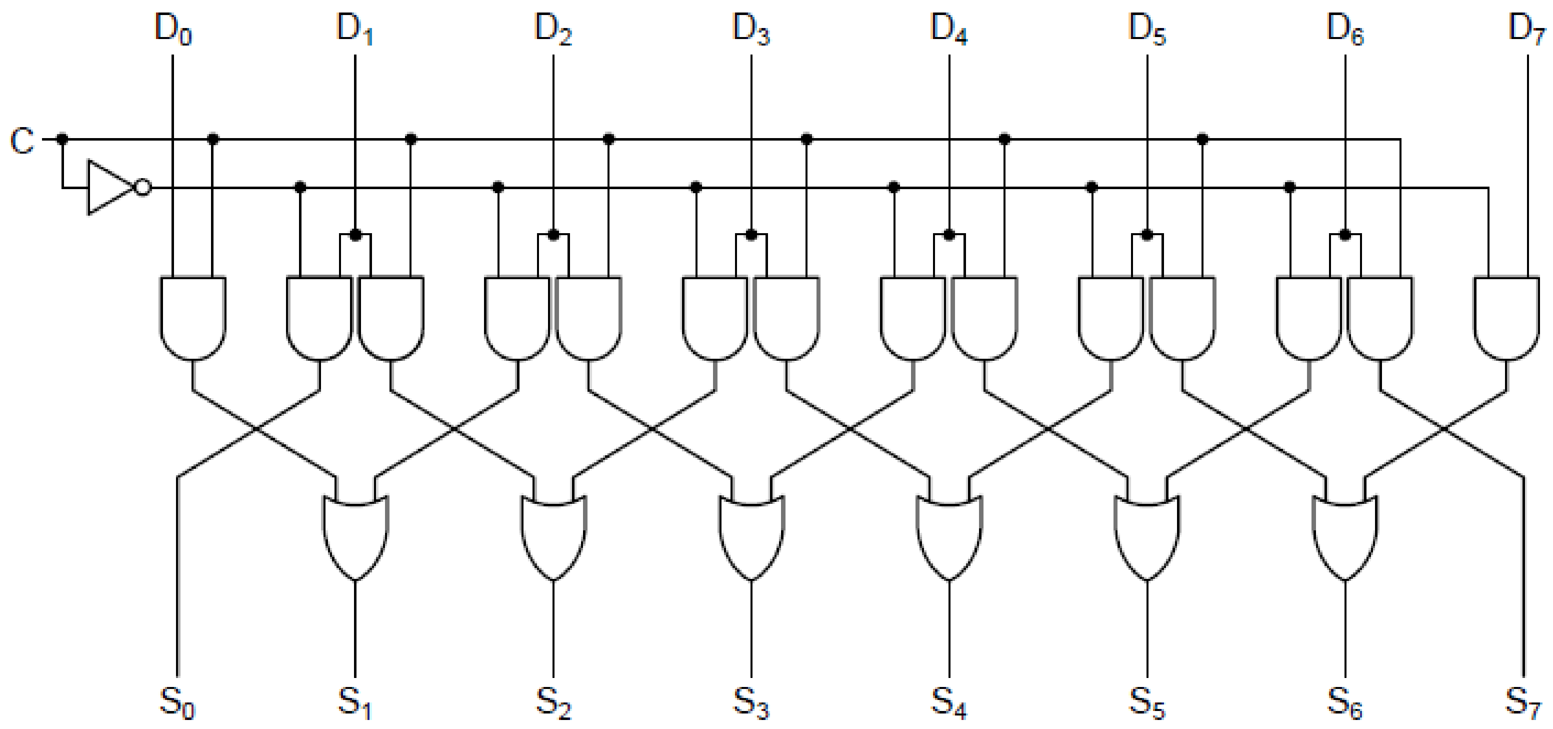
Comparator



A simple 4-bit comparator.

UNIVERSITY of WASHINGTON

1-Bit Left/Right Shift Register



Half Adder

```
1-bit addition
A + B = Sum
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 1 0
```

Truth Table

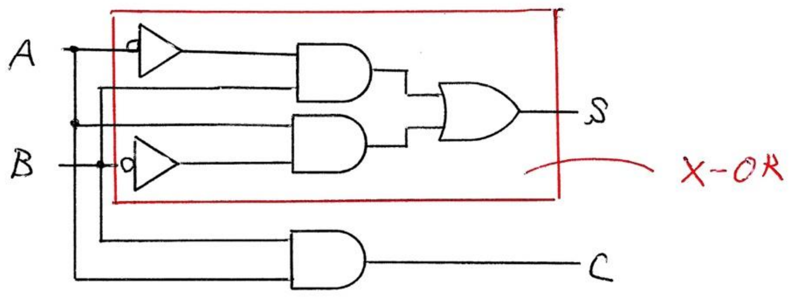
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

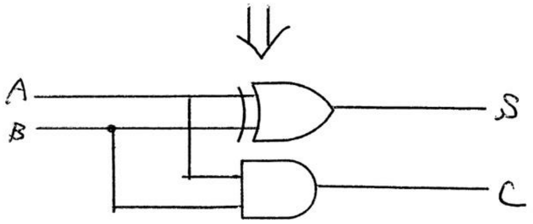
~AB

A~B

$S = \sim AB + A\sim B = A \oplus B$

$C = AB$

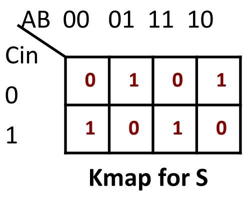




Full Adder (1)

Truth Table

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



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

Kmap for S

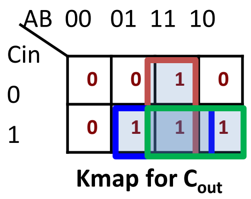
Can't simplify the logic at all:

$S = \sim A\sim BC_{in} + \sim AB\sim C_{in} + A\sim B\sim C_{in} + ABC_{in}$

Use the algebraical laws:

$S = (\sim AB + A\sim B) \sim C_{in} + (\sim A\sim B + AB)\ C_{in}$
$= (A \oplus B)\sim C_{in} + (\sim(A+B) + \sim(\sim A + \sim B))C_{in}$
$= (A \oplus B)\sim C_{in} + \sim((A+B)(\sim A + \sim B))C_{in}$
$= (A \oplus B)\sim C_{in} + \sim(A\sim A + A\sim B + \sim AB + B\sim B)C_{in}$
$= (A \oplus B)\sim C_{in} + \sim(A\sim B + \sim AB)C_{in}$
$= (A \oplus B)\sim C_{in} \ + \ \sim(A \oplus B)C_{in}$
$= A \oplus B \oplus C_{in}$

If a Kmap became a lattice,
It means exclusive ORs



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

Kmap for C_{out}

$C_{out} = AB + BC_{in} + C_{in}A$

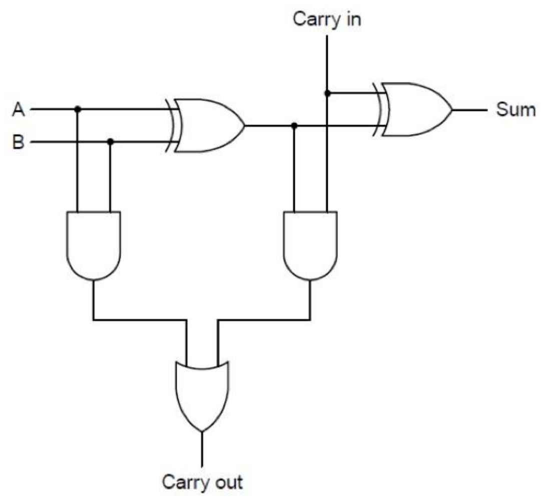
Use the algebraical laws:

$C_{out} = AB(C_{in} + \sim C_{in}) + (A + \sim A)BC_{in} + A(B + \sim B)C_{in}$
$= ABC_{in} + AB\sim C_{in} + ABC_{in} + \sim ABC_{in} + ABC_{in} + A\sim BC_{in}$
$= ABC_{in} + AB\sim C_{in} + (\sim AB + A\sim B)C_{in}$
$= AB(C_{in} + \sim C_{in}) + (A \oplus B)C_{in}$
$= AB + (A \oplus B)\ C_{in}$

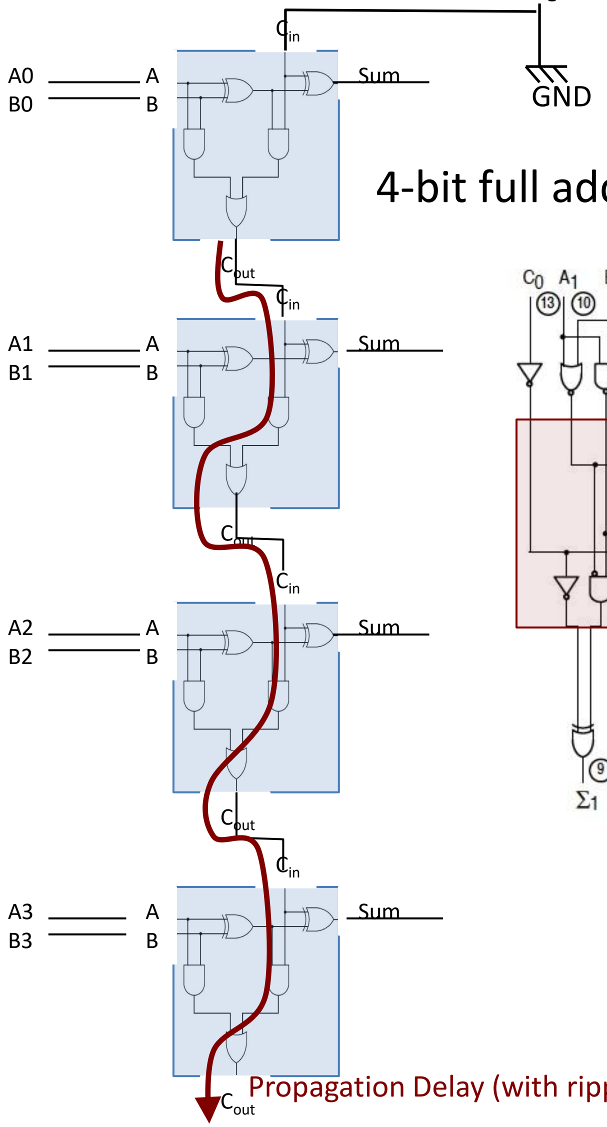
UNIVERSITY of WASHINGTON

Full Adder (2)

1-bit full addr



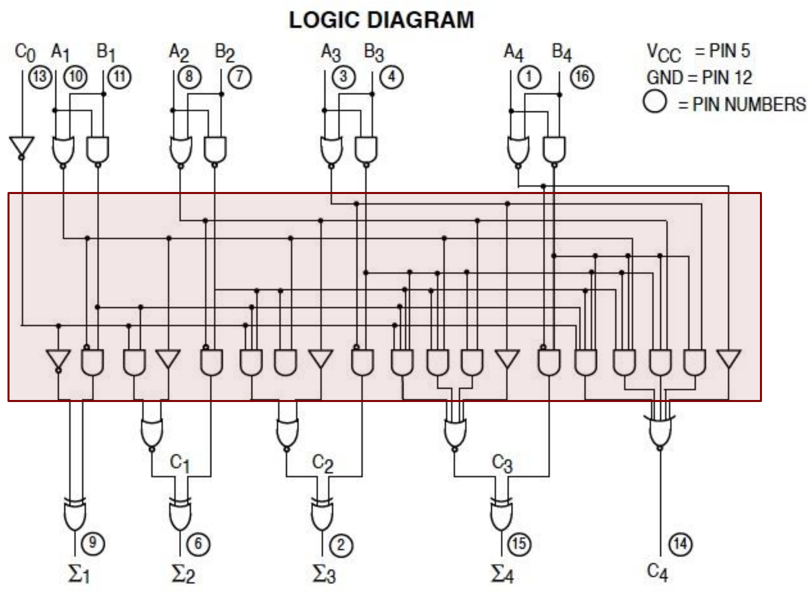
4-bit full addr



Propagation Delay (with ripple carry)

Carry look ahead

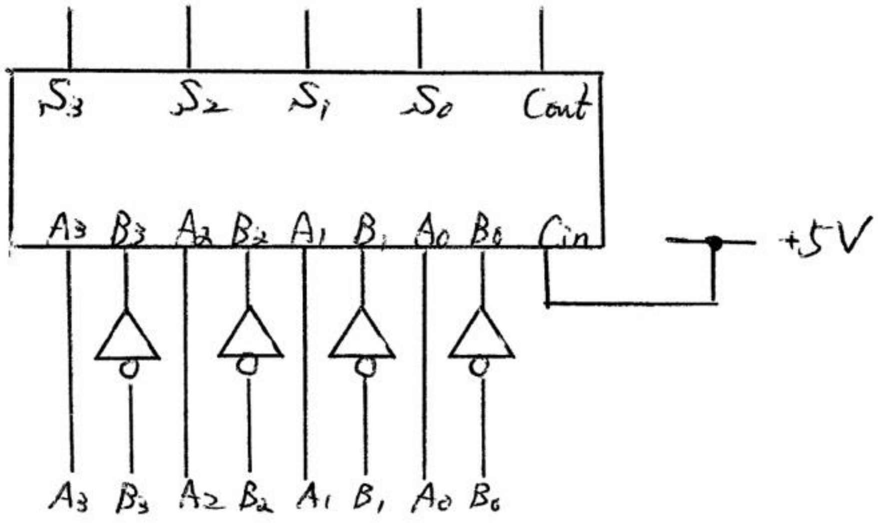
7483 4-bit full addr



Subtraction

- A – B = A + (–B) = A + (~B + 1)

2's complement


S_3
S_2
S_1
S_0
Cout
A_3 B_3 A_2 B_2 A_1 B_1 A_0 B_0 Cin
+5V
A_3 B_3 A_2 B_2 A_1 B_1 A_0 B_0

Overflow Bit

- If the sum of two numbers with the sign bits off yields a result number with the sign bit on, the overflow flag is turned on.
  - 0100 + 0100 = 1000 (overflow flag is turned on)
- If the sum of two numbers with the sign bits on yields a result number with the sign bit off, the overflow flag is turned on.
  - 1000 + 1000 = 0000 (overflow flag is turned on)
- How the ALU calculates the Overflow Flag?
  - Overflow happens when there is a carry into the sign bit but no carry out of the sign bit
  - The OVERFLOW flag is the XOR of the carry coming into the sign bit (if any) with the carry going out of the sign bit (if any). Overflow happens if the carry in does not equal the carry out.

```
 11
+01
===
 00

- carry in is 1
- carry out is 1
- 1 XOR 1 = NO OVERFLOW
```

```
 01
+01
===
 10

- carry in is 1
- carry out is 0
- 1 XOR 0 = OVERFLOW!
```

```
 11
+10
===
 01

- carry in is 0
- carry out is 1
- 0 XOR 1 = OVERFLOW!
```

```
 10
+01
===
 11

- carry in is 0
- carry out is 0
- 0 XOR 0 = NO OVERFLOW
```

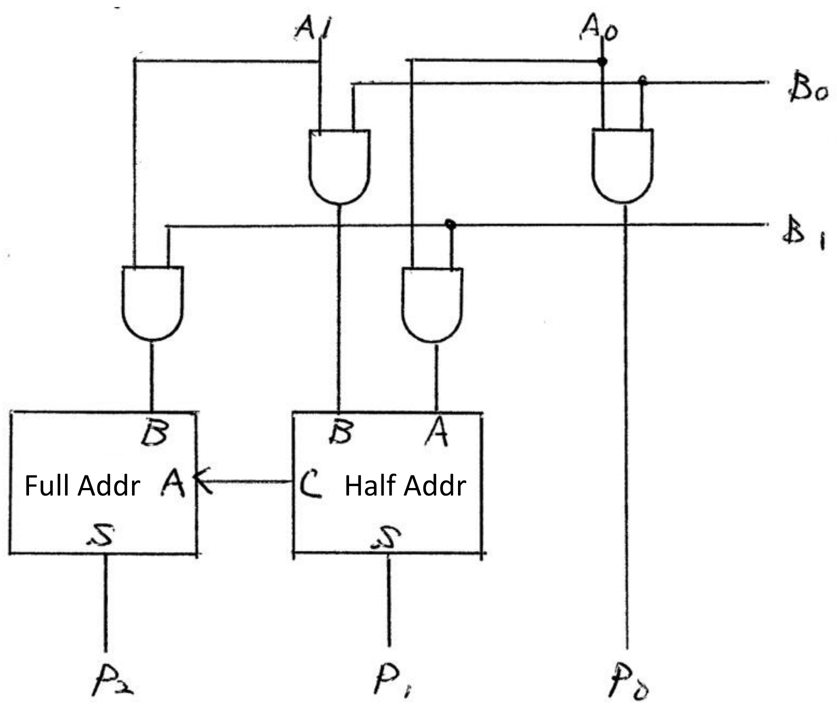
Reference: https://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt

UNIVERSITY of WASHINGTON

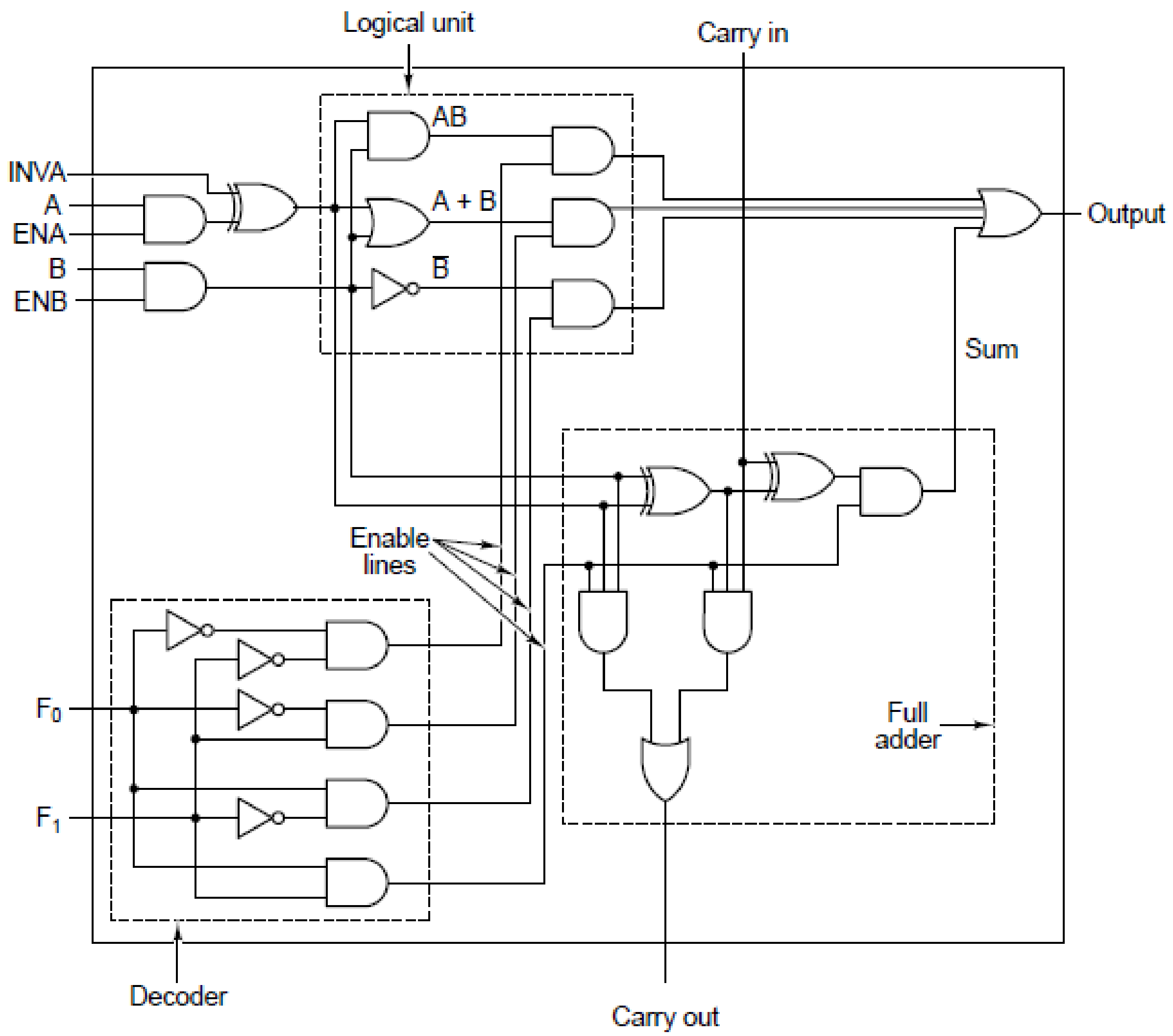
Multiplier

Binary Number	Multiplication Value
0 × 0 =	0
0 × 1 =	0
1 × 0 =	0
1 × 1 =	1

$$\begin{array}{rcccc}
 & & A_1 & A_0 \\
x) & & B_1 & B_0 \\
\hline
 & C_1 & A_1B_0 & A_0B_0 \\
 & + & + & \\
+) & A_1B_1 & A_0B_1 & \\
\hline
 & P_2 & P_1 & P_0
\end{array}$$



ALU

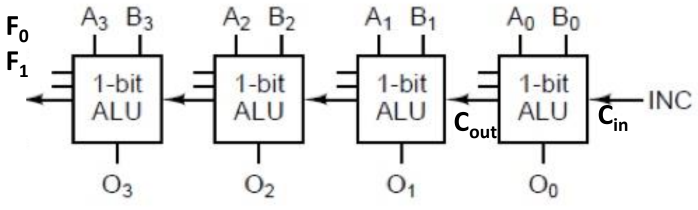


F_0	F_1	Operations
0	0	AB
0	1	A + B
1	0	~B
1	1	A plus B

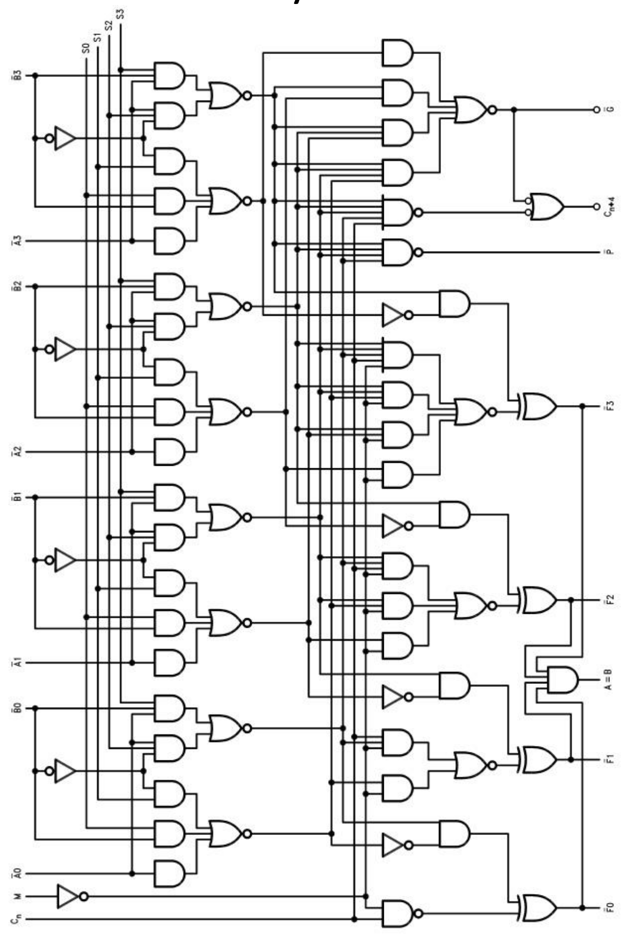
UNIVERSITY *of* WASHINGTON

4-bit ALU

Ripple Carry



74181: Carry Look Ahead



Summary

- More Kmap Techniques
  - Use the negation, XORs and “don’t care” terms
- Digital Components
  - Multiplexer: choosing a data line from memory chips
  - Decoder: choosing a memory chip
- Design of ALU
  - Comparator and shift register
  - Half/Full adder
  - Subtraction using a full adder (with ~B + C_{in})
  - Multiplication with two half adders
  - Ripple carry versus carry look ahead Adder/ALU