

CS 140 Lecture 14

Standard Combinational Modules

Professor CK Cheng
CSE Dept.
UC San Diego

Part III. Standard Modules

A. Interconnect

B. Operators. Adders Multiplier

- Adders
 - 1. Representation of numbers
 - 2. Full Adder
 - 3. Half Adder
 - 4. Ripple-Carry Adder
 - 5. Carry Look Ahead Adder
 - 6. Prefix Adder
 - 7. Carry Save Adder

ALU

Multiplier

Division

Operators

- Specification: Data Representations
- Arithmetic: Algorithms
- Logic: Synthesis
- Layout: Placement and Routing

1. Representation

- 2's Complement
-x: $2^n - x$
- 1's Complement
-x: $2^n - x - 1$

1. Representation

- 2's Complement

$$-x: 2^n - x$$

e.g. $16 - x$

- 1's Complement

$$-x: 2^n - x - 1$$

e.g. $16 - x - 1$

Id	2's comp.	1's comp.
0	0	15
-1	15	14
-2	14	13
-3	13	12
-4	12	11
-5	11	10
-6	10	9
-7	9	8
-8	8	

1. Representation

Id	-Binary	sign mag	2's comp	1's comp
0	0000	1000	0000	1111
-1	0001	1001	1111	1110
-2	0010	1010	1110	1101
-3	0011	1011	1101	1100
-4	0100	1100	1100	1011
-5	0101	1101	1011	1010
-6	0110	1110	1010	1001
-7	0111	1111	1001	1000
-8			1000	

Representation

1's Complement

For a negative number, we take the positive number and complement every bit.

2's Complement

For a negative number, we do 1s complement and plus one.

$$(b_{n-1}, b_{n-2}, \dots, b_0): -b_{n-1}2^{n-1} + \sum_{i < n-1} b_i 2^i$$

Representation

2's Complement

- $x+y$
- $x-y: x+2^n-y=2^n+x-y$
- $-x+y: 2^n-x+y$
- $-x-y: 2^n-x+2^n-y$
 $= 2^n+2^n-x-y$
- $-(-x)=2^n-(2^n-x)=x$

1's Complement

- $x+y$
- $x-y: x+2^n-y-1=2^n-1+x-y$
- $-x+y: 2^n-x-1+y=2^n-1-x+y$
- $-x-y: 2^n-x-1+2^n-y-1$
 $= 2^n-1+2^n-x-y-1$
- $-(-x)=2^n-(2^n-x-1)-1=x$

Examples

$$\begin{array}{r}
 2 + 3 = 5 \\
 \textcolor{red}{0} \text{ } \textcolor{red}{0} \text{ } \textcolor{red}{1} \text{ } \textcolor{red}{0} \\
 0 \text{ } 0 \text{ } 1 \text{ } 0 \\
 + 0 \text{ } 0 \text{ } 1 \text{ } 1 \\
 \hline
 0 \text{ } 1 \text{ } 0 \text{ } 1
 \end{array}$$

$$\begin{array}{r}
 2 - 3 = -1 \text{ (2's)} \\
 \textcolor{red}{0} \text{ } \textcolor{red}{0} \text{ } \textcolor{red}{0} \text{ } \textcolor{red}{0} \\
 0 \text{ } 0 \text{ } 1 \text{ } 0 \\
 + 1 \text{ } 1 \text{ } 0 \text{ } 1 \\
 \hline
 1 \text{ } 1 \text{ } 1 \text{ } 1
 \end{array}$$

$$\begin{array}{r}
 2 - 3 = -1 \text{ (1's)} \\
 0 \text{ } 0 \text{ } 1 \text{ } 0 \\
 + 1 \text{ } 1 \text{ } 0 \text{ } 0 \\
 \hline
 1 \text{ } 1 \text{ } 1 \text{ } 0
 \end{array}$$

Check for overflow (2's)

$$\begin{array}{r}
 -2 - 3 = -5 \text{ (2's)} \\
 \textcolor{red}{1} \text{ } \textcolor{red}{1} \text{ } \textcolor{red}{0} \text{ } \textcolor{red}{0} \\
 1 \text{ } 1 \text{ } 1 \text{ } 0 \\
 + 1 \text{ } 1 \text{ } 0 \text{ } 1 \\
 \hline
 1 \text{ } 0 \text{ } 1 \text{ } 1
 \end{array}$$

$$\begin{array}{r}
 -2 - 3 = -5 \text{ (1's)} \\
 \textcolor{red}{1} \text{ } \textcolor{red}{1} \text{ } \textcolor{red}{0} \text{ } \textcolor{red}{0} \\
 1 \text{ } 1 \text{ } 0 \text{ } 1 \\
 + 1 \text{ } 1 \text{ } 0 \text{ } 0 \\
 \hline
 1 \text{ } 0 \text{ } 0 \text{ } 1 \\
 \textcolor{red}{1} \\
 \hline
 1 \text{ } 0 \text{ } 1 \text{ } 0
 \end{array}$$

$$\begin{array}{r}
 3 + 5 = 8 \\
 \textcolor{red}{0} \text{ } \textcolor{red}{1} \text{ } \textcolor{red}{1} \text{ } \textcolor{red}{1} \\
 0 \text{ } 0 \text{ } 1 \text{ } 1 \\
 + 0 \text{ } 1 \text{ } 0 \text{ } 1 \\
 \hline
 1 \text{ } 0 \text{ } 0 \text{ } 0 \\
 C_4 C_3
 \end{array}$$

$$\begin{array}{r}
 -3 + -5 = -8 \\
 \textcolor{red}{1} \text{ } \textcolor{red}{1} \text{ } \textcolor{red}{1} \text{ } \textcolor{red}{1} \\
 1 \text{ } 1 \text{ } 0 \text{ } 1 \\
 + 1 \text{ } 0 \text{ } 1 \text{ } 1 \\
 \hline
 1 \text{ } 0 \text{ } 0 \text{ } 0 \\
 C_4 C_3
 \end{array}$$

Addition: 2's Complement Overflow

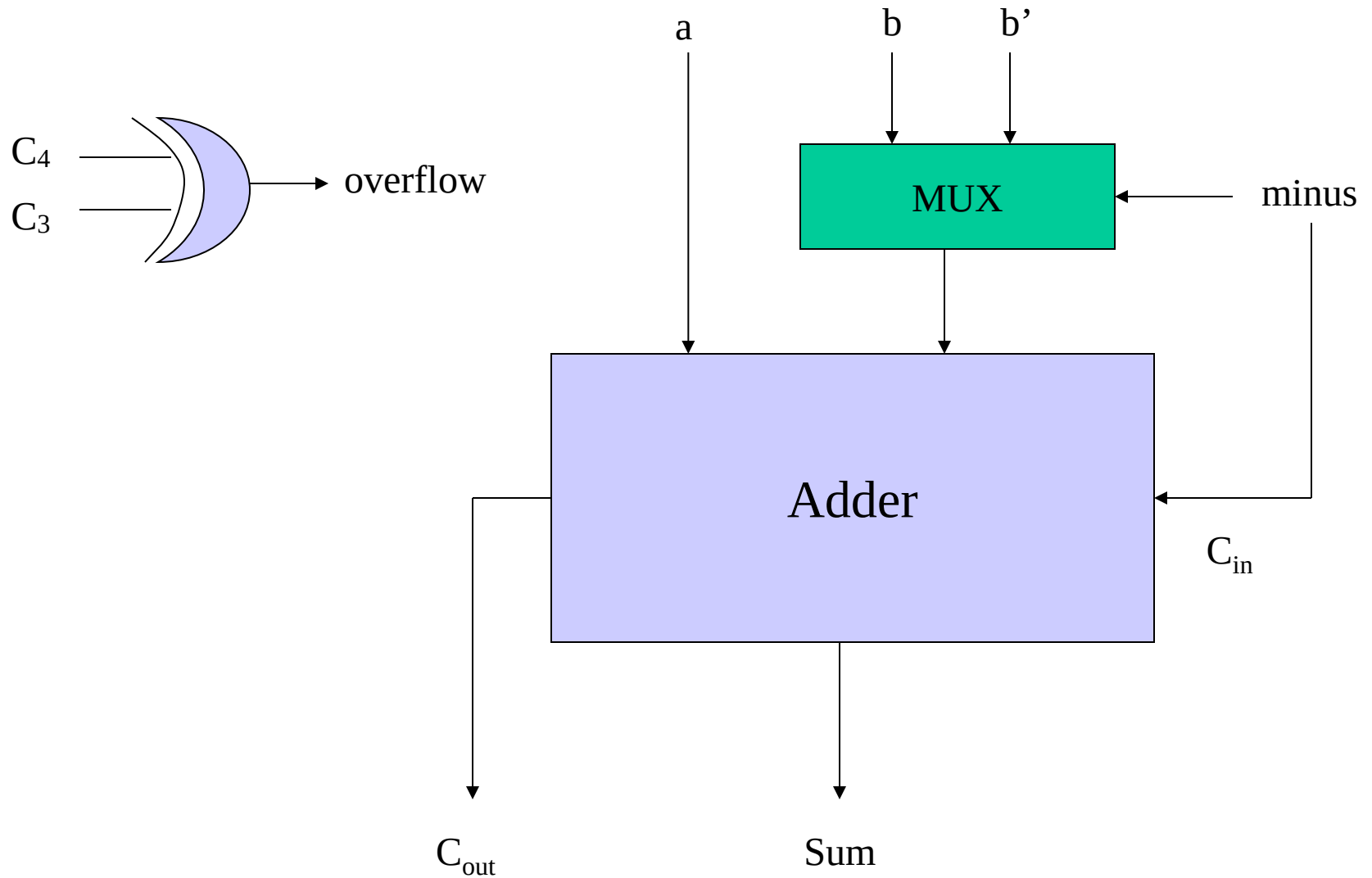
In 2's complement:

$$\text{overflow} = c_n \text{ xor } c_{n-1}$$

Exercise:

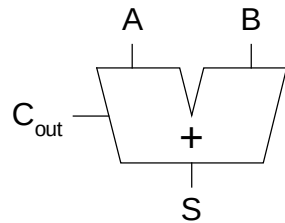
1. Demonstrate the overflow with more examples.
2. Prove the condition.

Addition and Subtraction using 2's Complement



1-Bit Adders

Half Adder

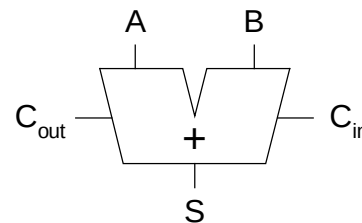


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$

$$C_{out} = AB$$

Full Adder

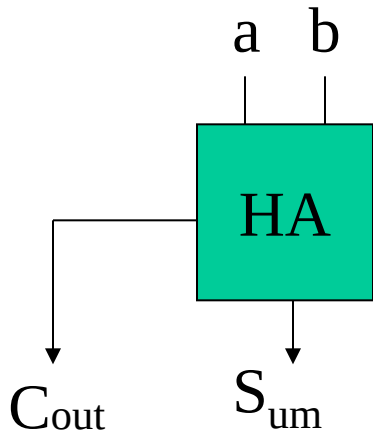


C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = A \oplus B \oplus C_{in}$$

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

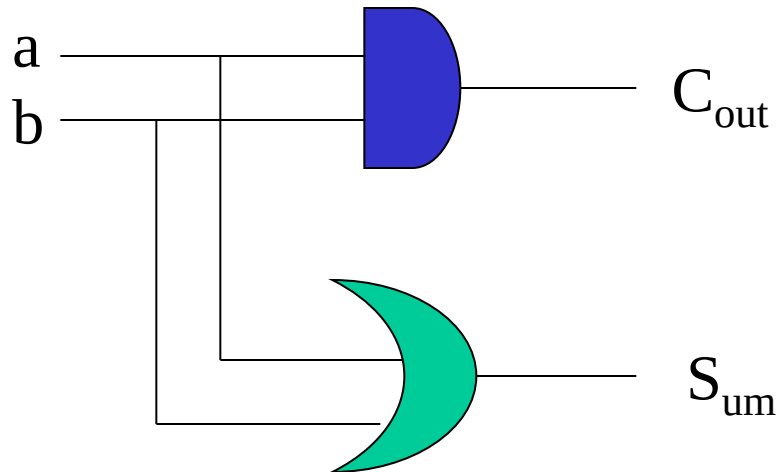
Half Adder



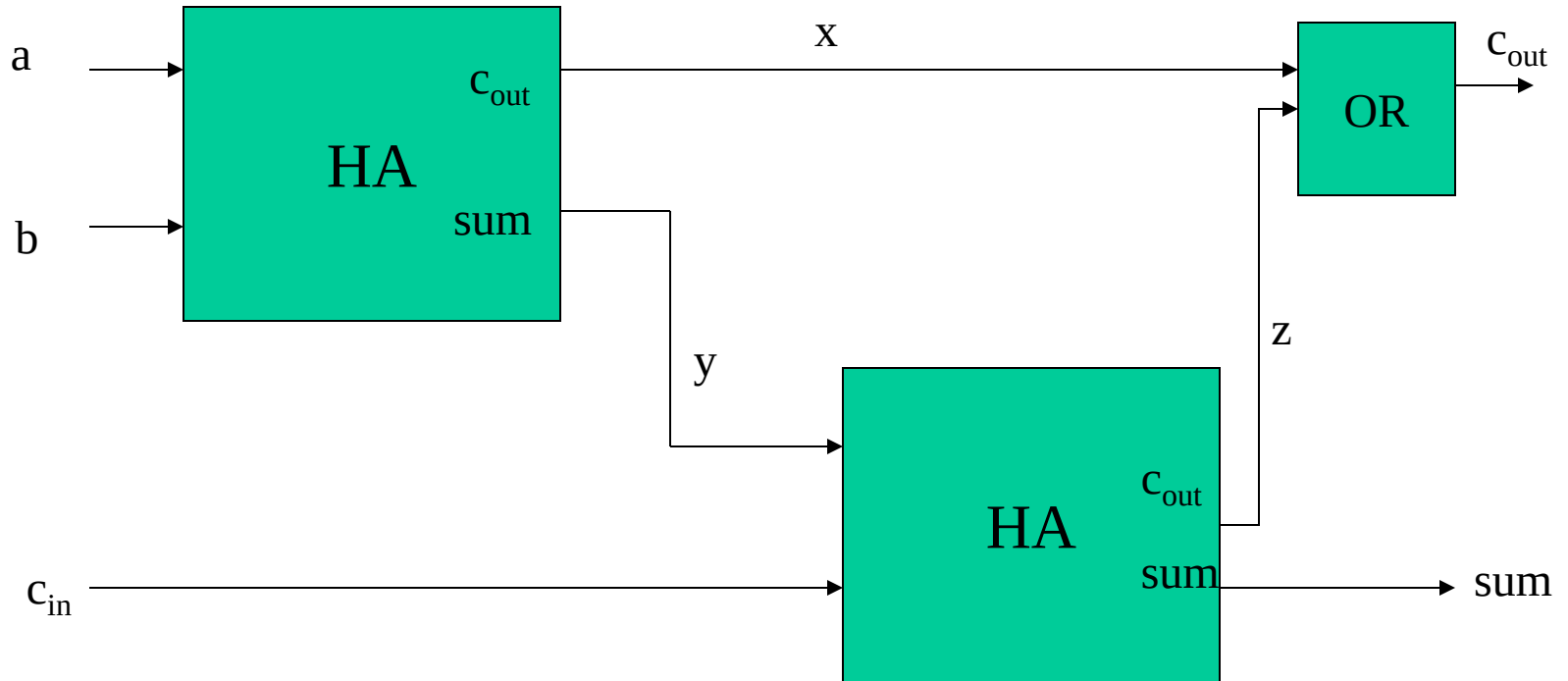
a	b	C _{out}	S _{um}
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S_{\text{um}} = ab' + a'b = a \oplus b$$

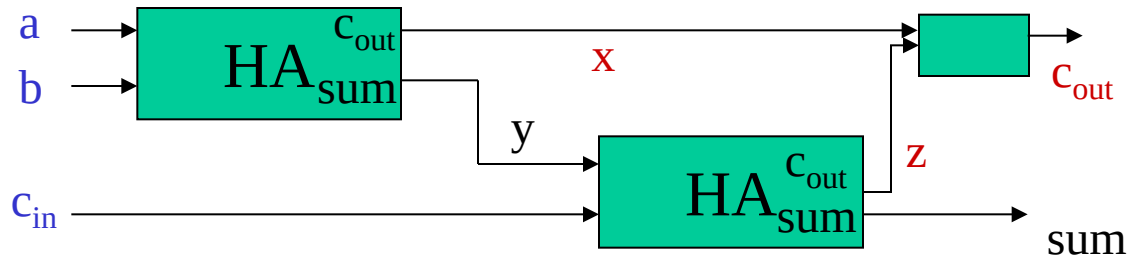
$$C_{\text{out}} = ab$$



Full Adder Composed of Half Adders



Full Adder Composed of Half Adders

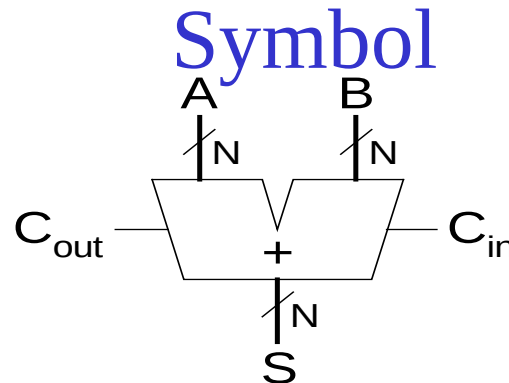


Id	a	b	c_{in}	x	y	z	c_{out}	s_{um}
0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	1
2	0	1	0	0	1	0	0	1
3	0	1	1	0	1	1	1	0
4	1	0	0	0	1	0	0	1
5	1	0	1	0	1	1	1	0
6	1	1	0	1	0	0	1	0
7	1	1	1	1	0	0	1	1

Id	x	z	c_{out}
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	-

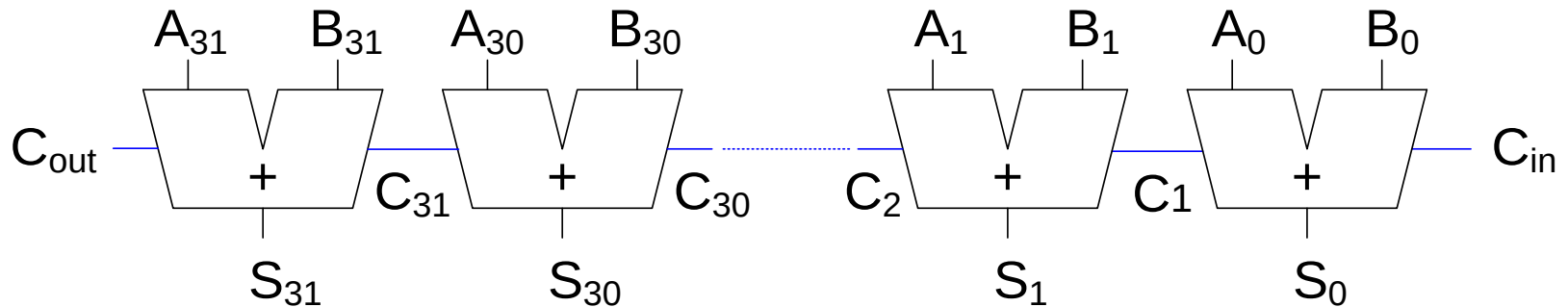
Adder

- Several types of carry propagate adders (CPAs) are:
 - Ripple-carry adders (slow)
 - Carry-lookahead adders (fast)
 - Prefix adders (faster)
- Carry-lookahead and prefix adders are faster for large adders but require more hardware.



Ripple-Carry Adder

- Chain 1-bit adders together
- Carry ripples through entire chain
- Disadvantage: **slow**



Ripple-Carry Adder Delay

- The delay of an N -bit ripple-carry adder is:

$$t_{\text{ripple}} = Nt_{FA}$$

where t_{FA} is the delay of a full adder

Carry-Lookahead Adder

- Compress the logic levels of C_{out}
- **Some definitions:**
 - Generate (G_i) and propagate (P_i) signals for each column:
 - A column will generate a carry out if A_i AND B_i are both 1.

$$G_i = A_i B_i$$

- A column will propagate a carry in to the carry out if A_i OR B_i is 1.

$$P_i = A_i + B_i$$

- The carry out of a column (C_i) is:

$$C_{i+1} = A_i B_i + (A_i + B_i)C_i = G_i + P_i C_i$$

Carry Look Ahead Adder

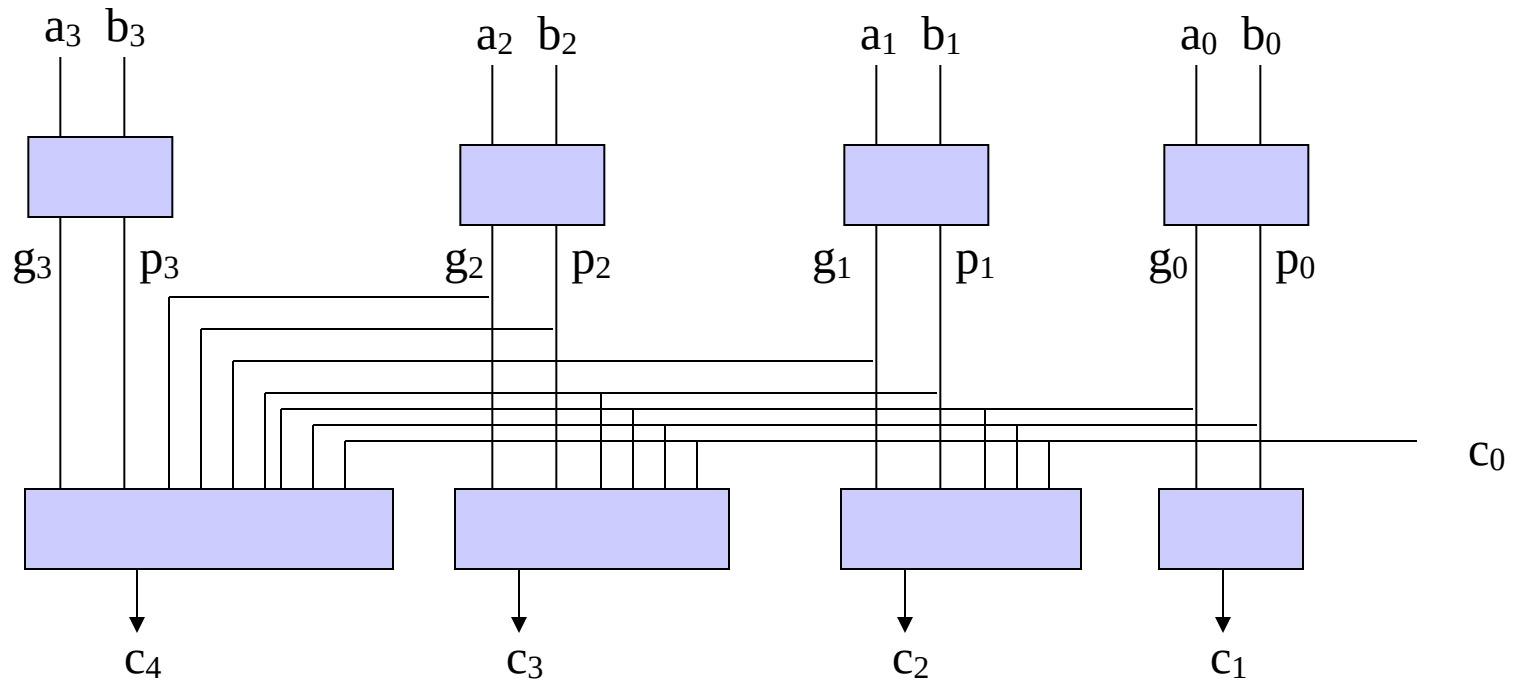
$$C_1 = a_0b_0 + (a_0+b_0)c_0 = g_0 + p_0c_0$$

$$C_2 = a_1b_1 + (a_1+b_1)c_1 = g_1 + p_1c_1 = g_1 + p_1g_0 + p_1p_0c_0$$

$$C_3 = a_2b_2 + (a_2+b_2)c_2 = g_2 + p_2c_2 = g_2 + p_2g_1 + p_2p_1g_0 + p_2p_1p_0c_0$$

$$C_4 = a_3b_3 + (a_3+b_3)c_3 = g_3 + p_3c_3 = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0 + p_3p_2p_1p_0c_0$$

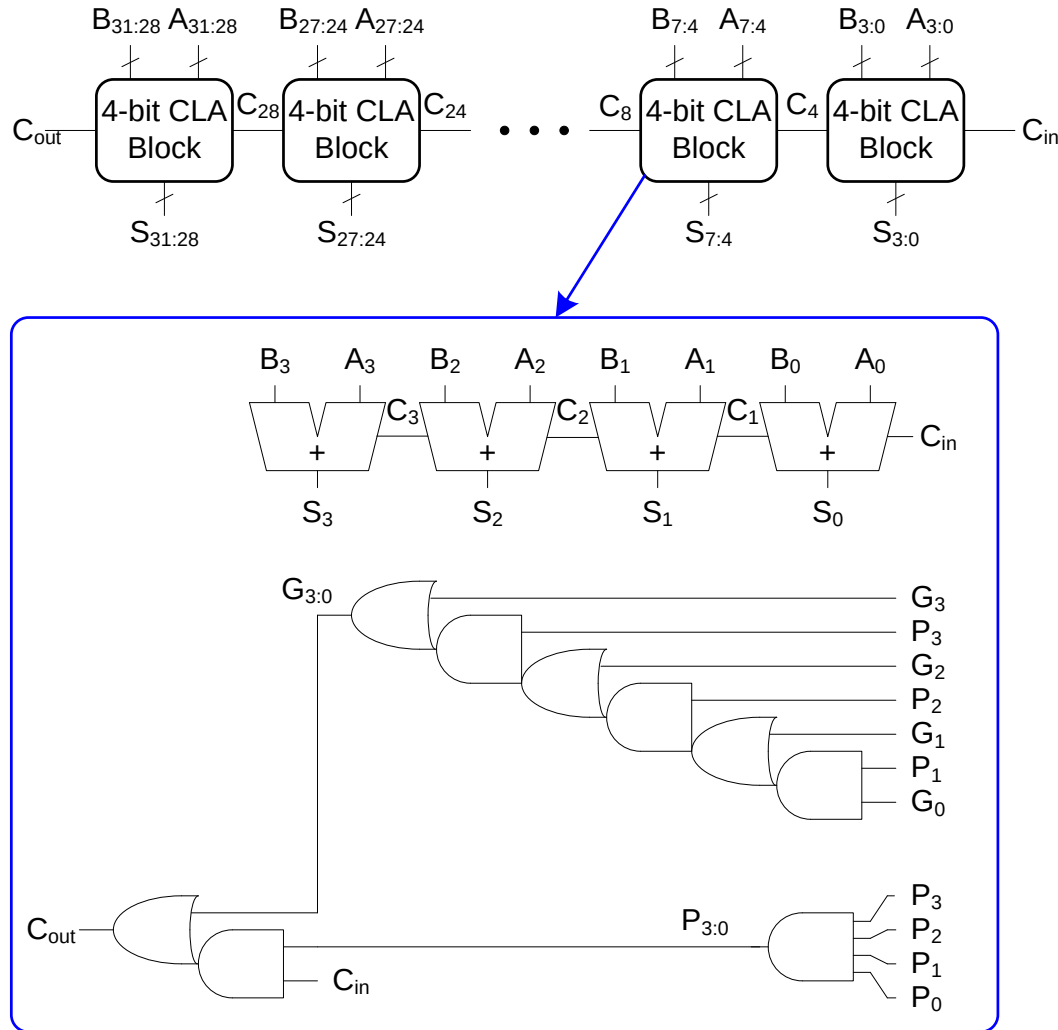
$$q_i = a_i b_i \quad p_i = a_i + b_i$$



Carry-Lookahead Addition

- Step 1: compute *generate* (G) and *propagate* (P) signals for columns (single bits)
- Step 2: compute G and P for k -bit blocks
- Step 3: C_{in} propagates through each k -bit propagate/generate block

32-bit CLA with 4-bit blocks



Carry-Lookahead Adder Delay

- Delay of an N -bit carry-lookahead adder with k -bit blocks:

$$t_{CLA} = t_{pg} + t_{pg_block} + (N/k - 1)t_{AND_OR} + kt_{FA}$$

where

- t_{pg} : delay of the column generate and propagate gates
 - t_{pg_block} : delay of the block generate and propagate gates
 - t_{AND_OR} : delay from C_{in} to C_{out} of the final AND/OR gate in the k -bit CLA block
- An N -bit carry-lookahead adder is generally much faster than a ripple-carry adder for $N > 16$

Prefix Adder

- Computes the carry in (C_{i-1}) for each of the columns as fast as possible and then computes the sum:

$$S_i = (A_i \oplus B_i) \oplus C_i$$

- Computes G and P for 1-bit, then 2-bit blocks, then 4-bit blocks, then 8-bit blocks, etc. until the carry in (generate signal) is known for each column
- Has $\log_2 N$ stages

Prefix Adder

- A carry in is produced by being either *generated* in a column or *propagated* from a previous column.
- Define column -1 to hold C_{in} , so $G_{-1} = C_{\text{in}}, P_{-1} = 0$
- Then, the carry in to col. i = the carry out of col. $i-1$:

$$C_{i-1} = G_{i-1:-1}$$

$G_{i-1:-1}$ is the generate signal spanning columns $i-1$ to -1 .

There will be a carry out of column $i-1$ (C_{i-1}) if the block spanning columns $i-1$ through -1 generates a carry.

- Thus, we rewrite the sum equation: $S_i = (A_i \oplus B_i) \oplus G_{i-1:-1}$
- **Goal:** Compute $G_{0:-1}, G_{1:-1}, G_{2:-1}, G_{3:-1}, G_{4:-1}, G_{5:-1}, \dots$
(These are called the *prefixes*)

Prefix Adder

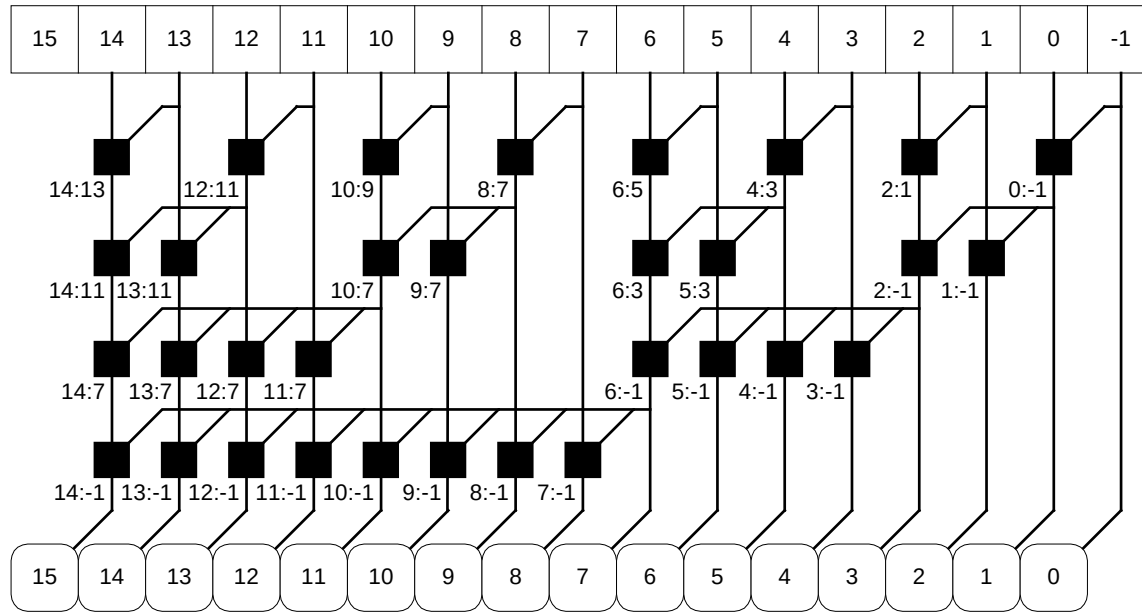
- The generate and propagate signals for a block spanning bits $i:j$ are:

$$G_{i:j} = G_{i:k} + P_{i:k} G_{k-1:j}$$

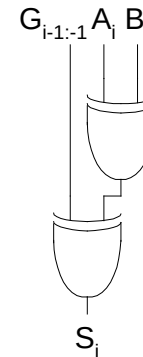
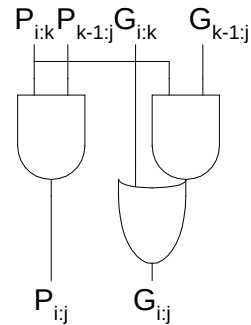
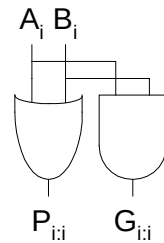
$$P_{i:j} = P_{i:k} P_{k-1:j}$$

- In words, these prefixes describe that:
 - A block will generate a carry if the upper part ($i:k$) generates a carry or if the upper part propagates a carry generated in the lower part ($k-1:j$)
 - A block will propagate a carry if both the upper and lower parts propagate the carry.

Prefix Adder Schematic



Legend



Carry Save Adder

Fast and low power addition for multiple additions

- A Redundant Number System
- Save the Partial Solution for Multiple Additions
- One Bit Propagation for Each Addition

Carry Save Adder: A Redundant Number System

- For each digit, we have two bits
 - $(0,0)$ $(0,1)$ $(1,0)$ $(1,1)$
- Weight 0 $(0,0)$
- Weight 1 $(0,1)$ $(1,0)$
- Weight 2 $(1,1)$

Carry Save Adder: A Simple Example

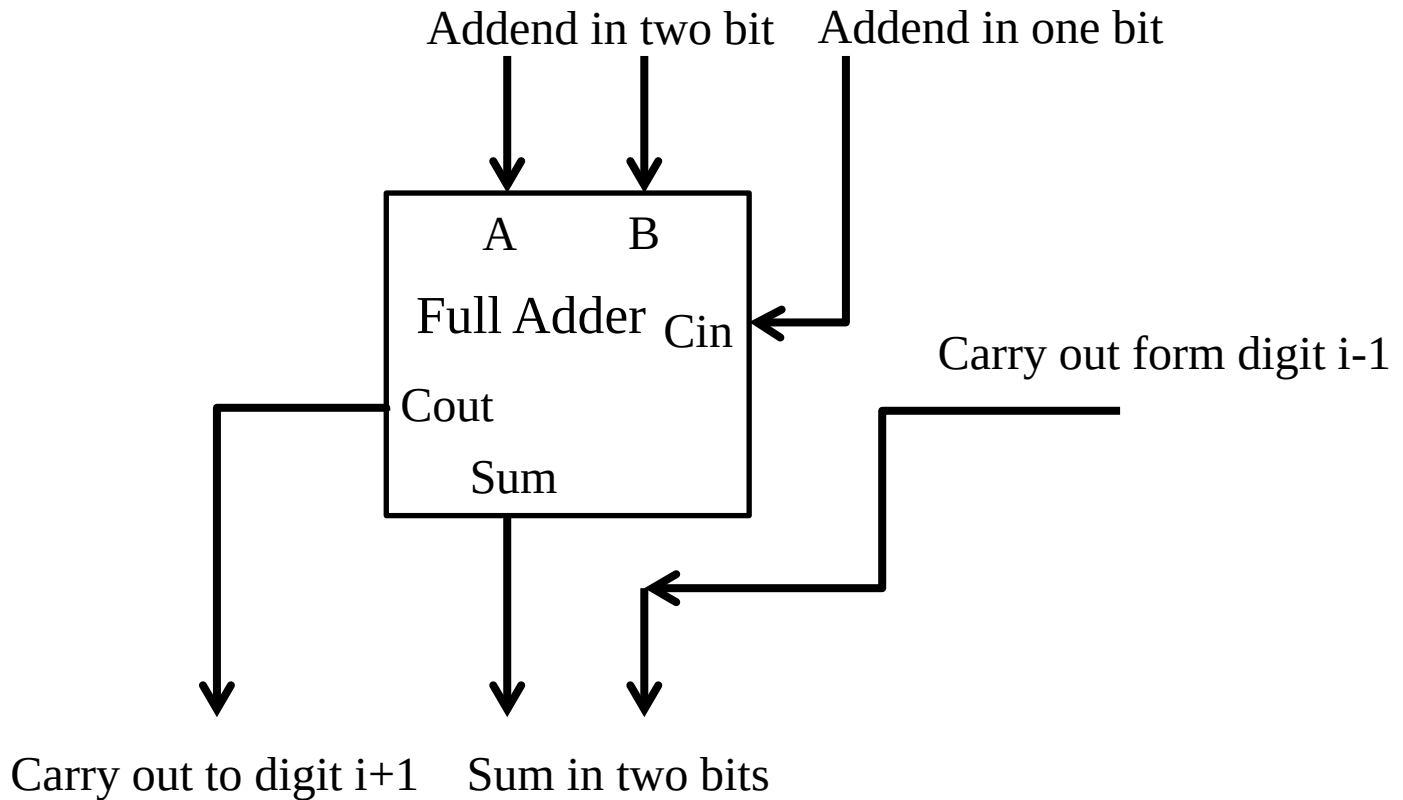
Weight	32	16	8	4	2	1
n1	0	0	1	0	0	1
n2	0	1	1	1	1	0
n3	0	1	1	1	0	1
n4	0	0	1	0	1	1

Carry Save Adder: A Simple Example

Weight	64	32	16	8	4	2	1
n1	0	0	0	1	1	0	1
n2	0	0	1	1	1	1	0
n3	0	0	1	1	1	0	1
n4	0	0	0	1	1	1	1

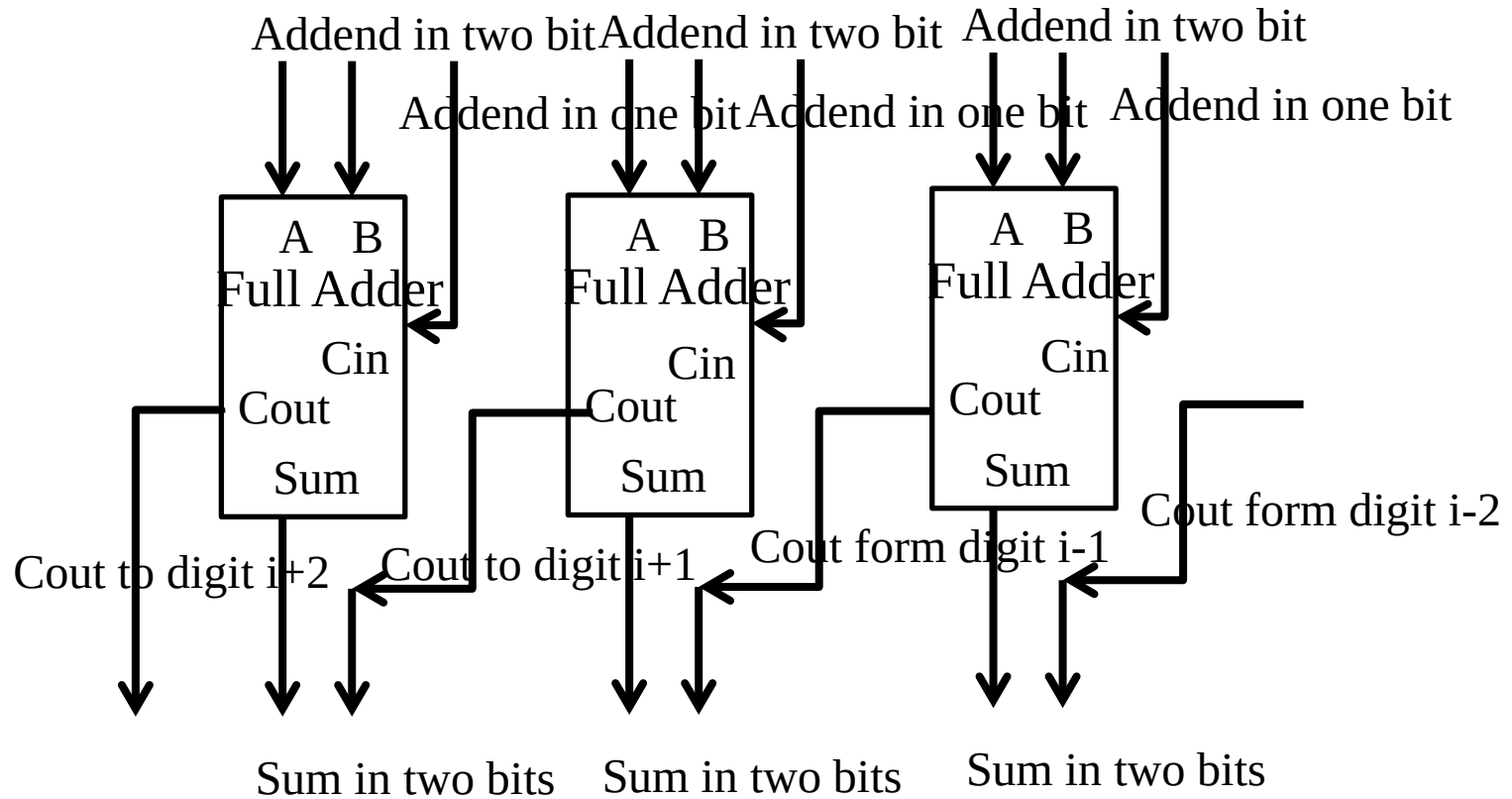
n1	00	00	00	10	10	00	10
n2	0	0	1	1	1	1	0
A=n1+n2	00	00	11	01	00	10	10
n3	0	0	1	1	1	0	1
B=A+n3	00	01	11	00	10	11	00
n4	0	0	0	1	1	1	1
C=B+n4	00	10	11	11	01	10	10
Sum	1	0	1	0	1	1	1

Carry Save Adder: One Digit Logic Diagram



- For digit i , the addition of addend in two bits and addend in one bit produces the carry out and sum.
- The sum and the carry out from digit $i-1$ become the two bit output at digit i .

Carry Save Adder: Three Digit Logic Diagram



- For digit i , the addition of addend in two bits and addend in one bit produces the carry out and sum.
- The sum and the carry out from digit $i-1$ become the two bit output at digit i .

Carry Save Adder

Procedure of carry save addition system

1. Convert the first addend to the redundant number system
2. For intermediate solution, perform carry save addition.
3. Split the redundant number into two numbers.
4. Perform the addition on the two numbers to general the solution.

Prefix Adder Delay

- The delay of an N -bit prefix adder is:

$$t_{PA} = t_{pg} + \log_2 N(t_{pg_prefix}) + t_{XOR}$$

where

- t_{pg} is the delay of the column generate and propagate gates (AND or OR gate)
- t_{pg_prefix} is the delay of the black prefix cell (AND-OR gate)

Adder Delay Comparisons

- Compare the delay of 32-bit ripple-carry, carry-lookahead, and prefix adders. The carry-lookahead adder has 4-bit blocks. Assume that each two-input gate delay is 100 ps and the full adder delay is 300 ps.

Adder Delay Comparisons

- Compare the delay of 32-bit ripple-carry, carry-lookahead, and prefix adders. The carry-lookahead adder has 4-bit blocks. Assume that each two-input gate delay is 100 ps and the full adder delay is 300 ps.

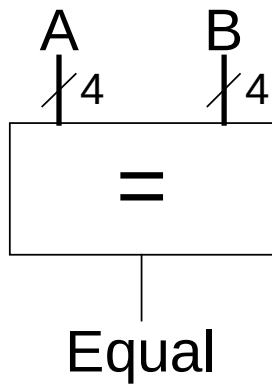
$$t_{\text{ripple}} = Nt_{FA} = 32(300 \text{ ps}) = 9.6 \text{ ns}$$

$$\begin{aligned} t_{CLA} &= t_{pg} + t_{pg_block} + (N/k - 1)t_{AND_OR} + kt_{FA} \\ &= [100 + 600 + (7)200 + 4(300)] \text{ ps} \\ &= 3.3 \text{ ns} \end{aligned}$$

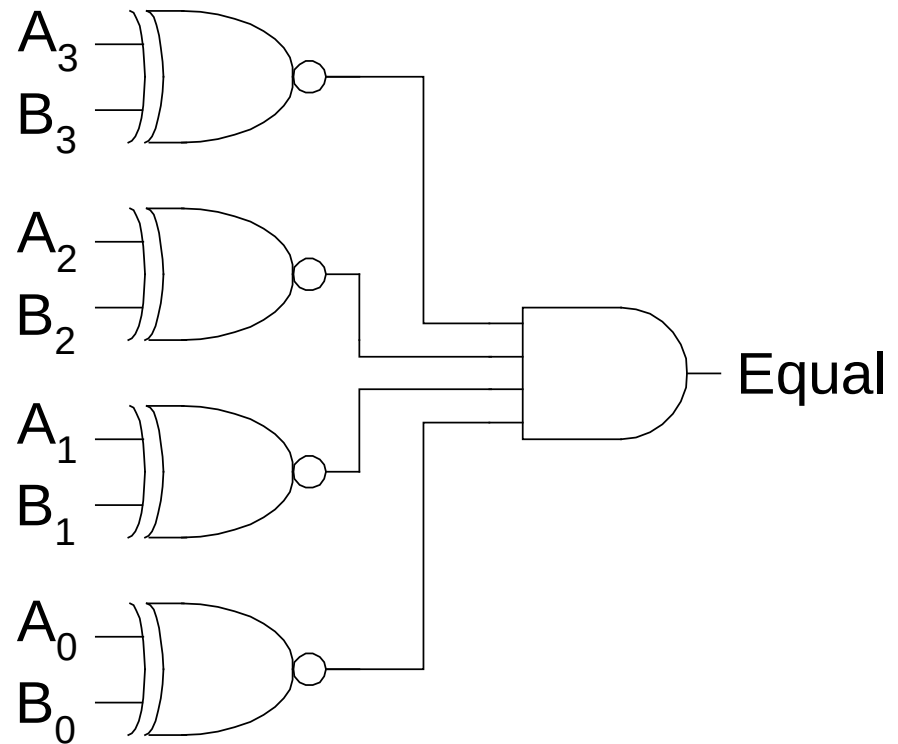
$$\begin{aligned} t_{PA} &= t_{pg} + \log_2 N(t_{pg_prefix}) + t_{XOR} \\ &= [100 + \log_2 32(200) + 100] \text{ ps} \\ &= 1.2 \text{ ns} \end{aligned}$$

Comparator: Equality

Symbol

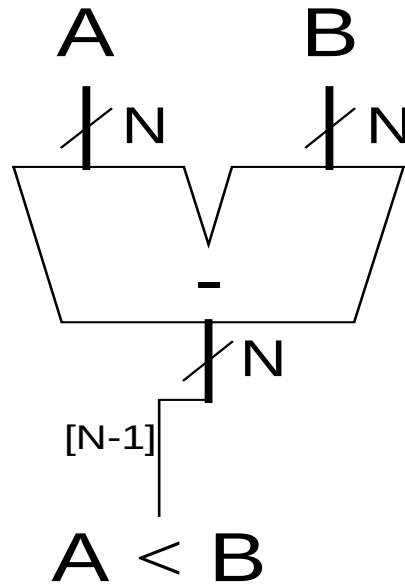


Implementation

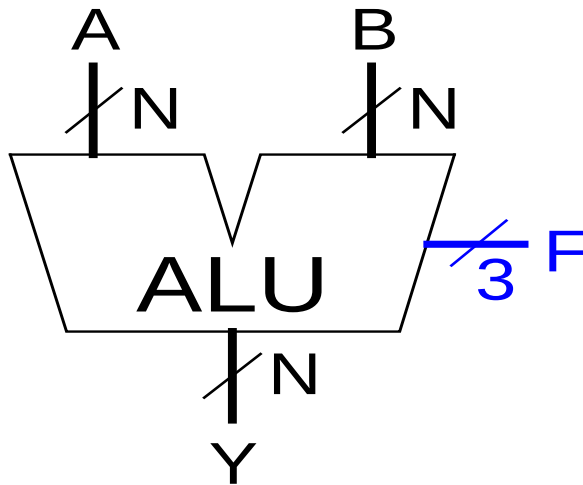


Comparator: Less Than

- For unsigned numbers

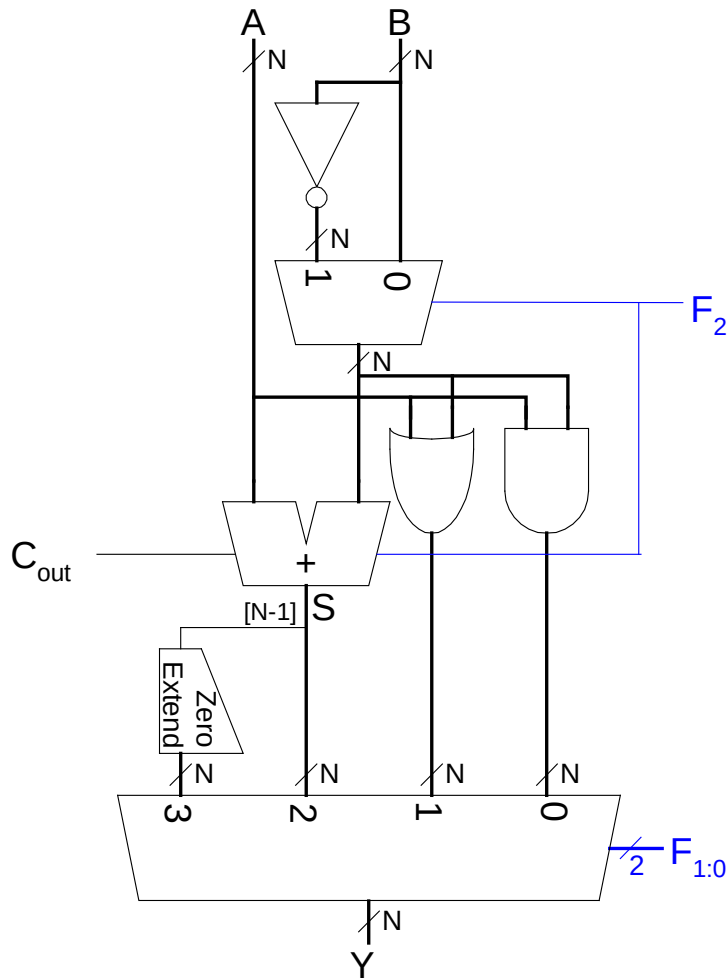


Arithmetic Logic Unit (ALU)



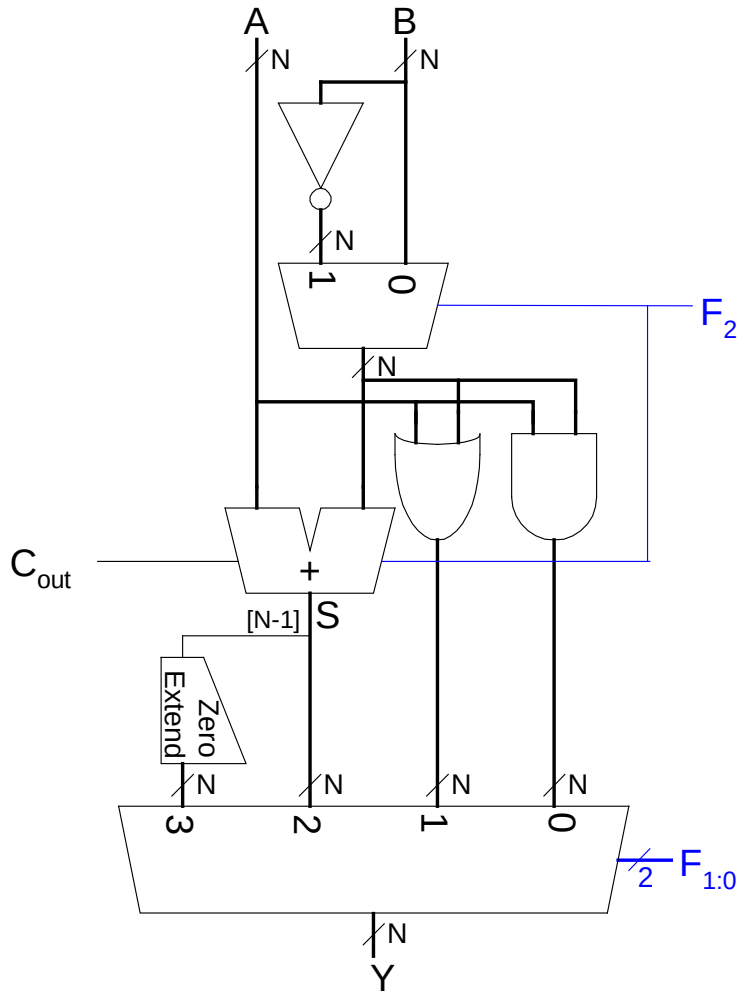
$F_{2:0}$	Function
000	$A \& B$
001	$A \mid B$
010	$A + B$
011	not used
100	$A \& \sim B$
101	$A \mid \sim B$
110	$A - B$
111	SLT

ALU Design



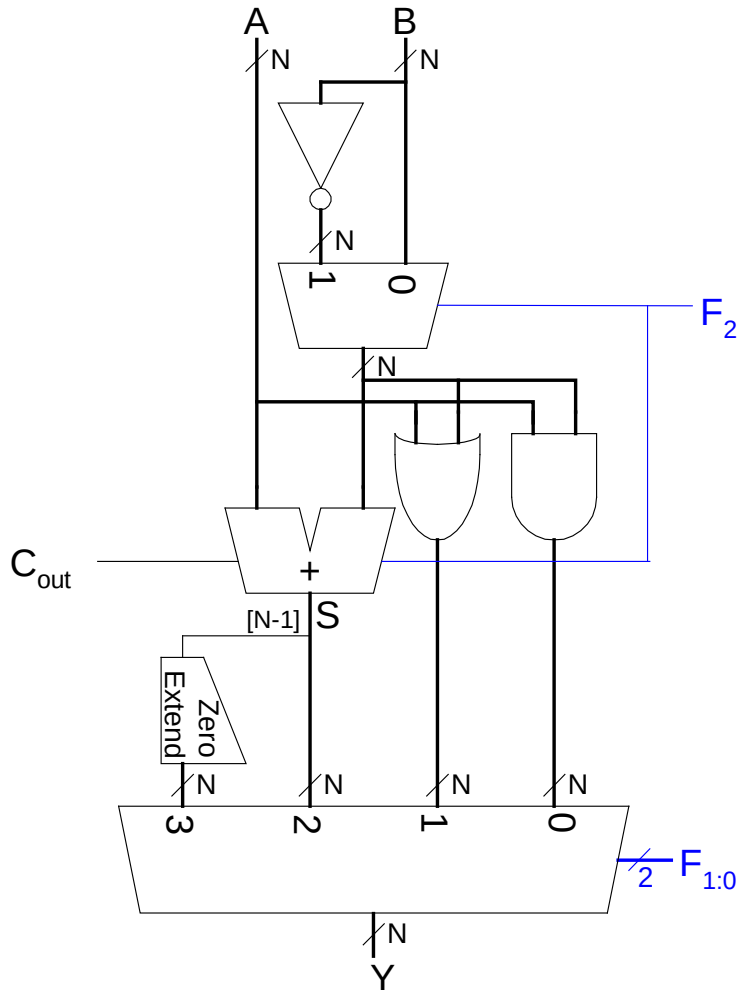
$F_{2:0}$	Function
000	$A \& B$
001	$A B$
010	$A + B$
011	not used
100	$A \& \sim B$
101	$A \sim B$
110	$A - B$
111	SLT

Set Less Than (SLT) Example



- Configure a 32-bit ALU for the set if less than (SLT) operation. Suppose $A = 25$ and $B = 32$.

Set Less Than (SLT) Example

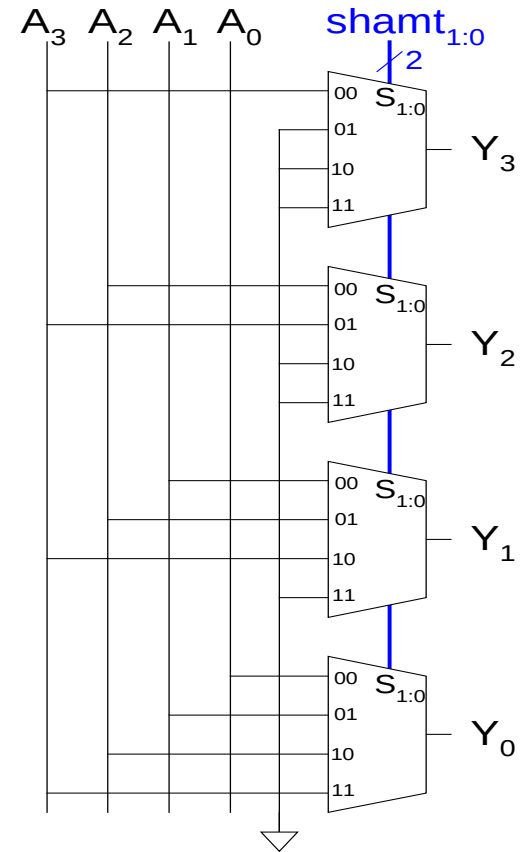
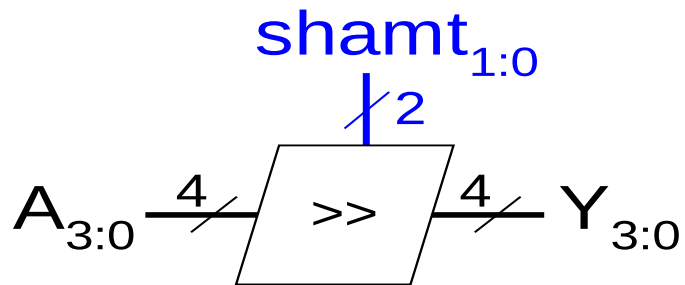


- Configure a 32-bit ALU for the set if less than (SLT) operation. Suppose $A = 25$ and $B = 32$.
 - A is less than B , so we expect Y to be the 32-bit representation of 1 ($0x00000001$).
 - For SLT, $F_{2:0} = 111$.
 - $F_2 = 1$ configures the adder unit as a subtractor. So $25 - 32 = -7$.
 - The two's complement representation of -7 has a 1 in the most significant bit, so $S_{31} = 1$.
 - With $F_{1:0} = 11$, the final multiplexer selects $Y = S_{31}$ (zero extended) = $0x00000001$.

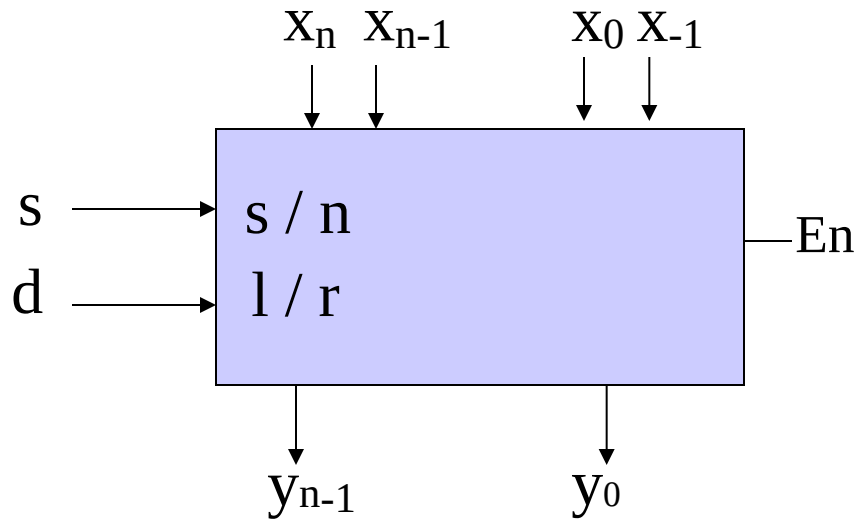
Shifters

- **Logical shifter:** shifts value to left or right and fills empty spaces with 0's
 - Ex: $11001 \gg 2 = 00110$
 - Ex: $11001 \ll 2 = 00100$
- **Arithmetic shifter:** same as logical shifter, but on right shift, fills empty spaces with the old most significant bit (msb).
 - Ex: $11001 \ggg 2 = 11110$
 - Ex: $11001 \lll 2 = 00100$
- **Rotator:** rotates bits in a circle, such that bits shifted off one end are shifted into the other end
 - Ex: $11001 \text{ ROR } 2 = 01110$
 - Ex: $11001 \text{ ROL } 2 = 00111$

Shifter Design

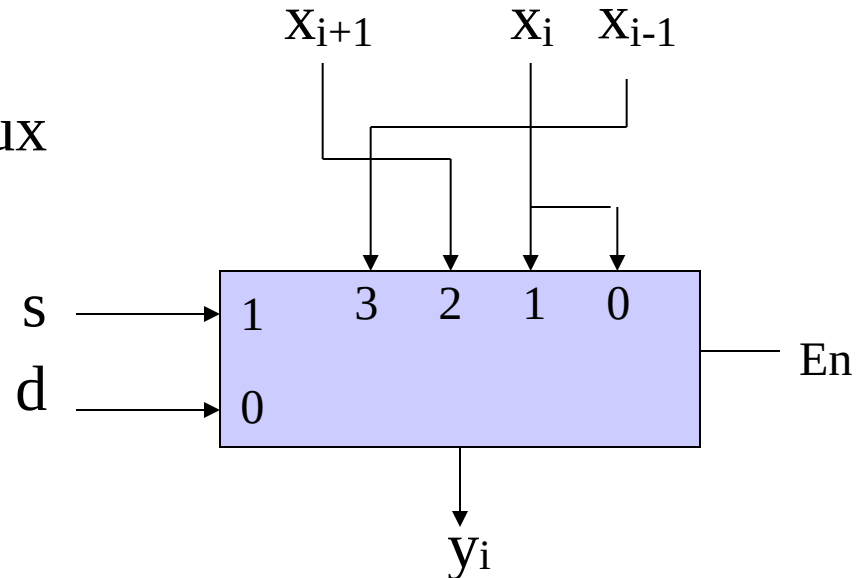


Shifter

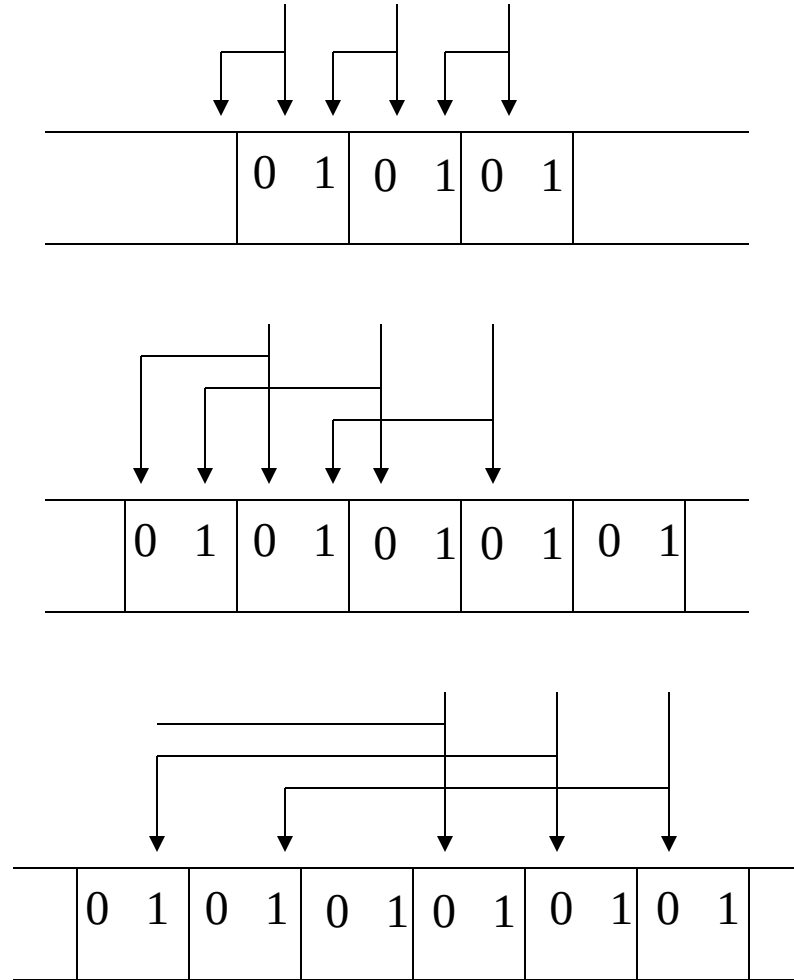
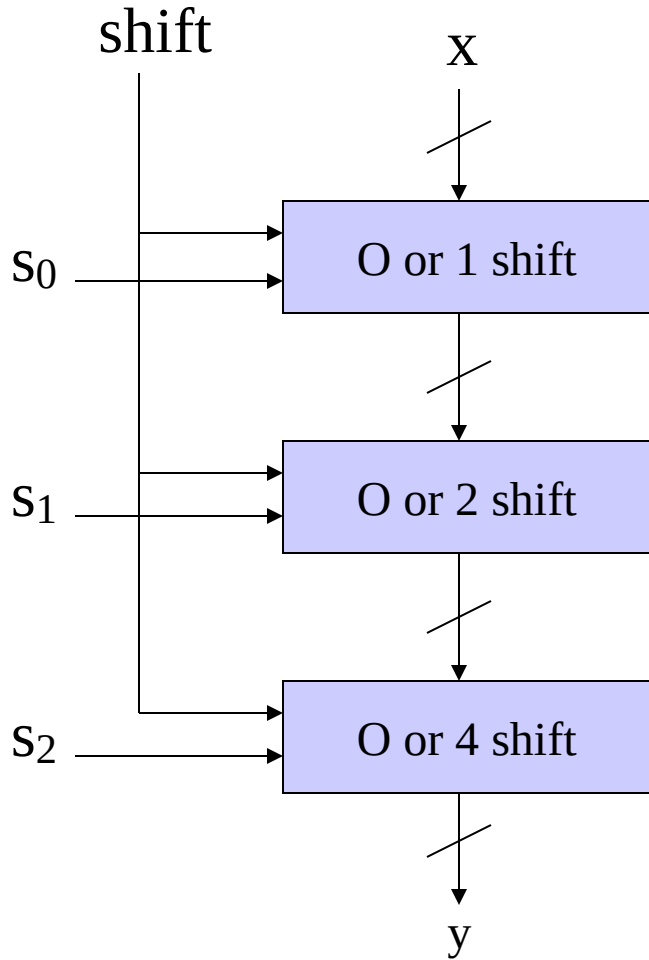


$$\begin{aligned}
 y_i &= x_{i-1} \text{ if } En = 1, s = 1, \text{ and } d = L \\
 &= x_{i+1} \text{ if } En = 1, s = 1, \text{ and } d = R \\
 &= x_i \text{ if } En = 1, s = 0 \\
 &= 0 \text{ if } En = 0
 \end{aligned}$$

Can be implemented with a mux



Barrel Shifter



Shifters as Multipliers and Dividers

- A left shift by N bits multiplies a number by 2^N
 - Ex: $00001 \ll 2 = 00100$ ($1 \times 2^2 = 4$)
 - Ex: $11101 \ll 2 = 10100$ ($-3 \times 2^2 = -12$)
- The arithmetic right shift by N divides a number by 2^N
 - Ex: $01000 \ggg 2 = 00010$ ($8 \div 2^2 = 2$)
 - Ex: $10000 \ggg 2 = 11100$ ($-16 \div 2^2 = -4$)

Multipliers

- Steps of multiplication for both decimal and binary numbers:
 - Partial products are formed by multiplying a single digit of the multiplier with the entire multiplicand
 - Shifted partial products are summed to form the result

Decimal

$$\begin{array}{r}
 230 \\
 \times 42 \\
 \hline
 460 \\
 + 920 \\
 \hline
 9660
 \end{array}$$

multiplicand
 multiplier
 partial
 products
 result

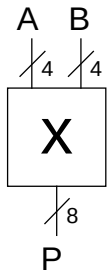
$$230 \times 42 = 9660$$

Binary

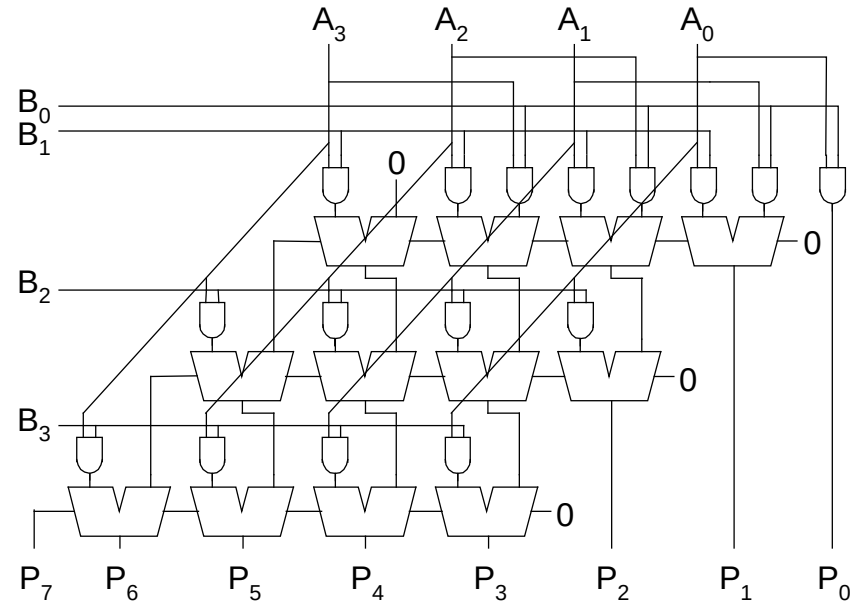
$$\begin{array}{r}
 0101 \\
 \times 0111 \\
 \hline
 0101 \\
 0101 \\
 0101 \\
 + 0000 \\
 \hline
 0100011
 \end{array}$$

$$5 \times 7 = 35$$

4 x 4 Multiplier



$$\begin{array}{r}
 \begin{array}{cccc}
 & A_3 & A_2 & A_1 & A_0 \\
 \times & B_3 & B_2 & B_1 & B_0 \\
 \hline
 & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 \\
 A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & \\
 A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 & \\
 + & A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 \\
 \hline
 P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
 \end{array}
 \end{array}$$



Division Algorithm

- $Q = A/B$
- R : remainder
- D : difference

$$R = A$$

for $i = N-1$ to 0

$$D = R - B$$

if $D < 0$ then $Q_i = 0, R' = R$ $// R < B$

else $Q_i = 1, R' = D$ $// R \geq B$

$$R = 2R'$$

4 x 4 Divider

