EECS1010 Logic Design
Lecture 4 Combinational Logic

Jenny Yi-Chun Liu
jennyliu@gapp.nthu.edu.tw

Outline

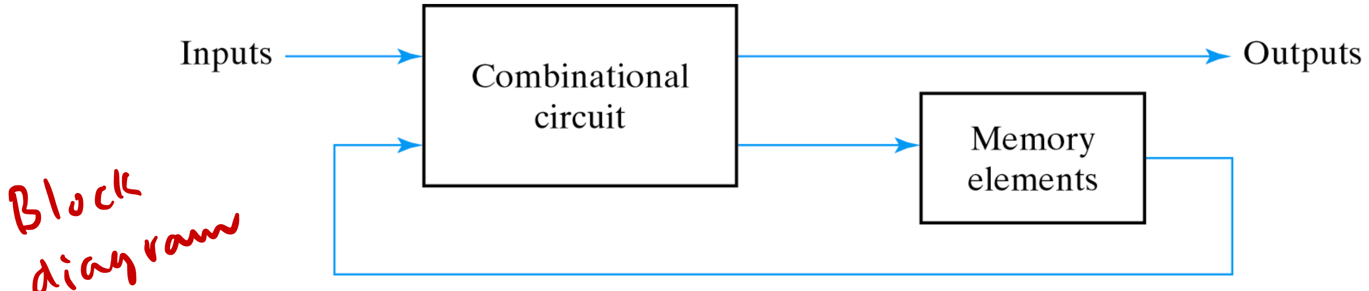
- Digital systems and information
- Boolean algebra and logic gates
- Gate-level minimization
- Combinational logic
- Sequential circuits
- Registers and counters
- Memory

Chapter Outline

- Combinational circuits
- Analysis of combinational circuits
- Design of combinational circuits
- Binary adders and subtractors
- Binary multiplier
- Decoders
- Encoders
- Multiplexers
- Arbiters
- Comparators
- Shifters

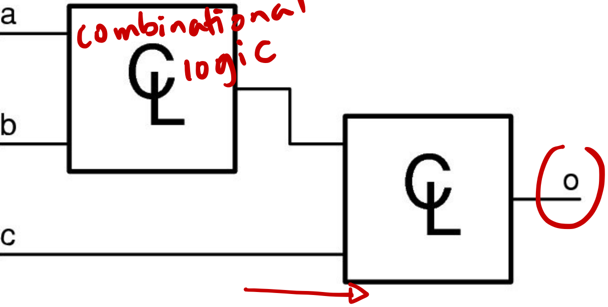
Logic Circuits for Digital Systems

1. • Combinational circuits
   – Outputs at any time are determined directly and only by the present inputs.
2. • Sequential circuits
   – Circuits that employ memory elements and combinational logic gates.
   – Outputs are determined by the present inputs and the state of the memory cells.



Closure

- Combinational circuits are closed under acyclic composition.



- Cyclic composition of combinational logic circuits
  - The feedback variable can remember the history of the circuits.
  - Sequential circuits.

Analyze of combinational circuits

Analyze a Combinational Circuit

- Analysis procedure
  - Make sure the given circuit is combinational
    - No **feedback path** or **memory element**
  - Derive the corresponding ***Boolean functions***
  - Derive the corresponding ***truth table***
  - Verify and analyze the design
    - Logic simulation (waveforms)
  - Explain the function

Derivation of Boolean Function

- Label all gate outputs that are functions of the input variables only. Determine the functions.
- Label all gate outputs that are functions of the input variables and previously labeled gate outputs, and find the functions.
- Repeat previous step until all the primary outputs are obtained.

Example: Derivation of Boolean Function

$F_1, F_2 (A, B, C)$

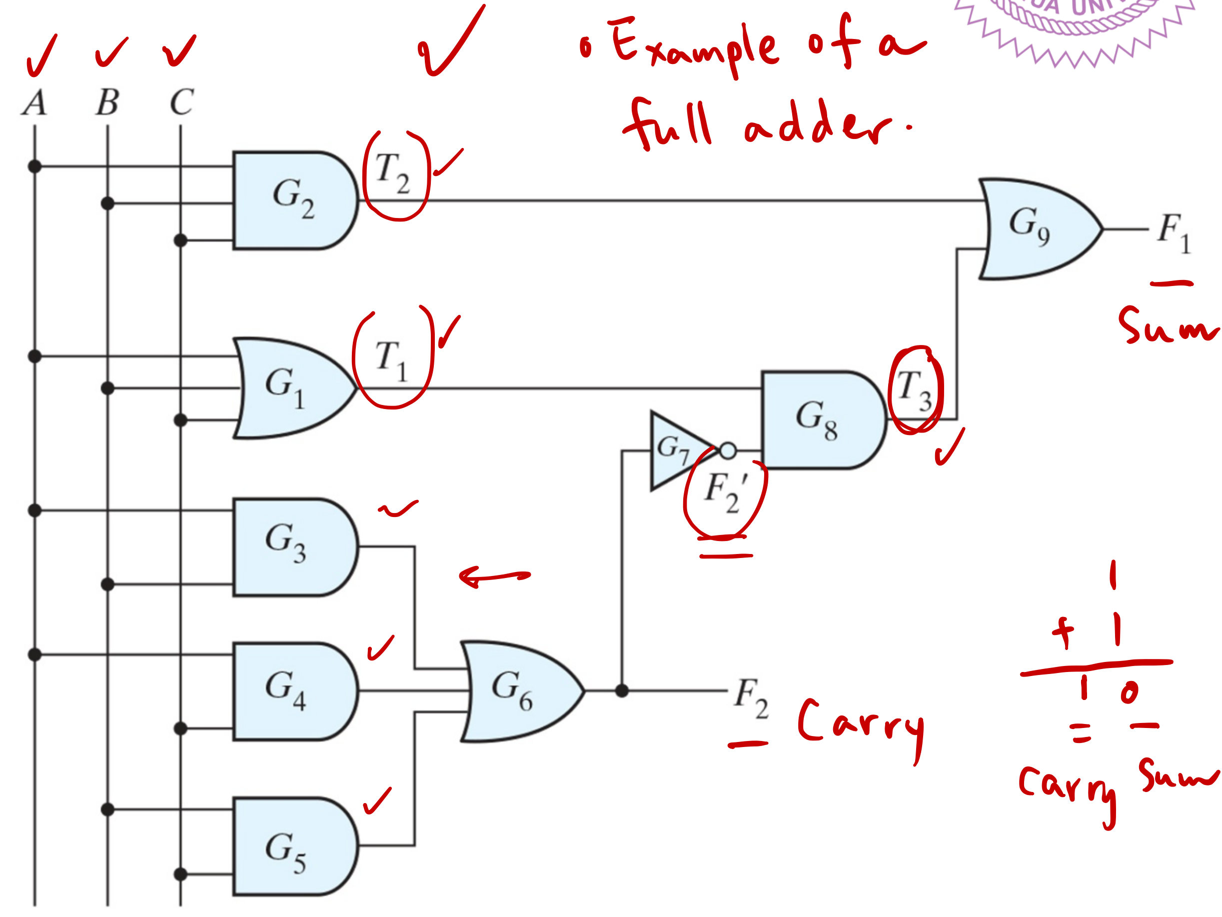
$F_1 (A, B, C) = T_2 + T_3$

$T_2 = A \cdot B \cdot C$

$T_2 = T_1 \cdot F_2'$

$T_1 = A + B + C$

$F_2 = AB + AC + BC$



$F_1 = ABC + T_1 \cdot \overline{F_2}$

$= ABC + (A+B+C) \cdot (AB+AC+BC)'$

$= ABC + (A+B+C)\left((\bar{A}+\bar{B})(\bar{A}+\bar{C})(\bar{B}+\bar{C})\right) = ABC + A\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C}$

Derivation of Truth Table

- For n input variables
  - List all the 2^n input combinations from 0 to 2^n-1.
  - Partition the circuit into small single-output blocks and label the output of each block.
  - Obtain the truth table of the blocks depending on the input variables only.
  - Proceed to obtain the truth tables for other blocks that depend on previously defined truth tables.

Design of combinational circuits

Design Procedure

1. Specification: From the specifications, determine the inputs, outputs, and their symbols.
2. Formulation: Derive the truth table (functions) from the relationship between the inputs and outputs
3. Optimization: Derive the simplified Boolean functions for each output function. Draw a logic diagram or provide a netlist for the resulting circuits using AND, OR, and inverters.
4. Technology Mapping: Transform the logic diagram or netlist to a new diagram or netlist using the available implementation technology.
5. Verification: Verify the design.

Design Example: A BCD-to-Excess-3 Converter (1/3)

1. Specifications
   - input (ABCD), output (wxyz) (MSB to LSB)
   - ABCD: 0000 ~ 1001 (0~9)
2. Formulation

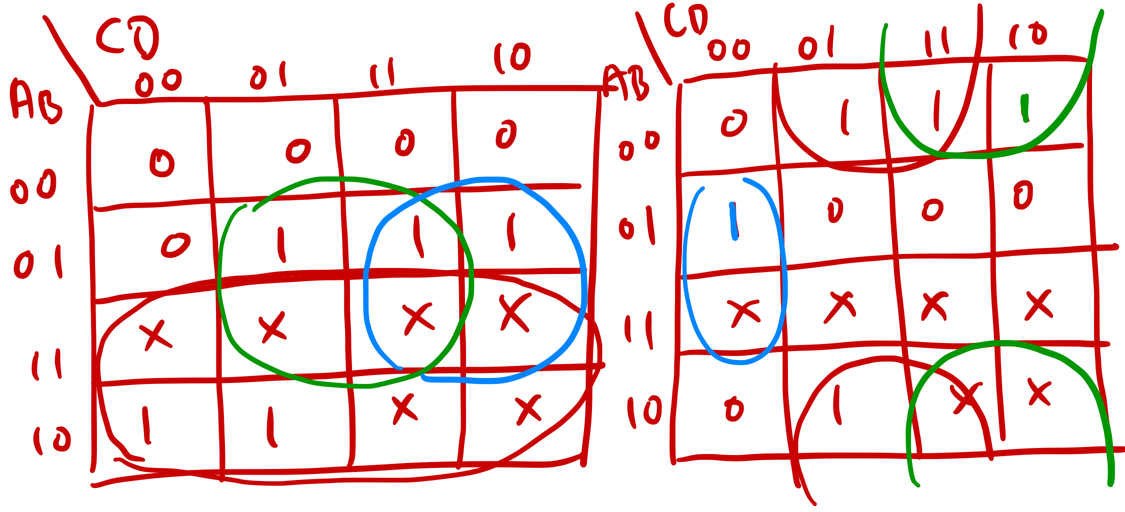
2. Truth table

Input BCD code				Output Excess-3 code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

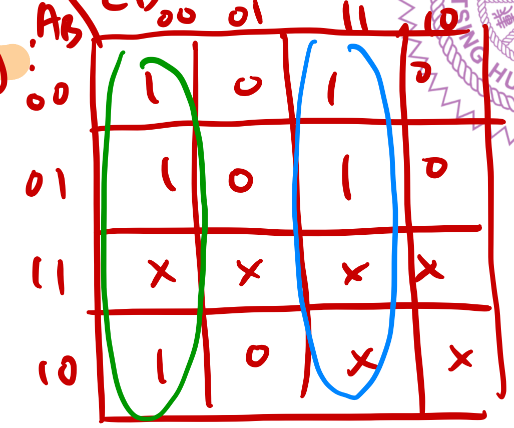
Design Example: A BCD-to-Excess-3 Converter (2/3)

3. Optimization

w: x:



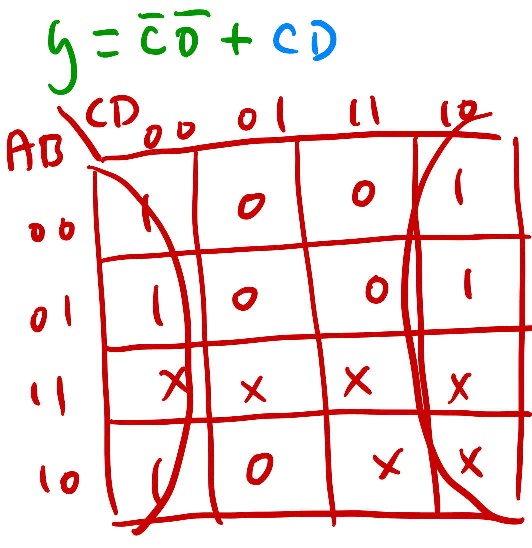
y:



$w = A + BD + BC$

$x = \bar{B}D + \bar{B}C + B\bar{C}\bar{D}$

$y = \bar{C}\bar{D} + CD$

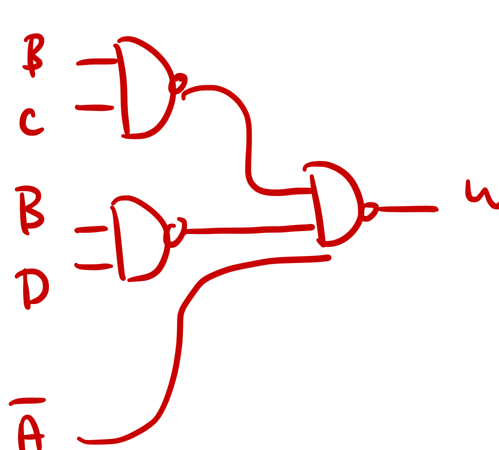


$z = \bar{D}$

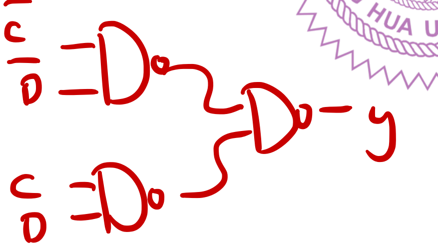
Design Example: A BCD-to-Excess-3 Converter (3/3)

4. Technology mapping

- NAND only



x circuit inputs: $\bar{B}$, D; $\bar{B}$, C; B, $\bar{C}$, D → x



$y = \bar{C}\bar{D} + CD$

D → z

$z = \bar{D}$

$w = A + BD + BC$

$x = \bar{B}D + \bar{B}C + B\bar{C}\bar{D}$

$A - B = A + (-B)$

- 4/25 (Tue): midterm 2
- 4/18 (Tue): quiz 2, 2:20pm –

Binary adders and subtractors

- Half adders (HA)
- Full adders (FA)
- Ripple-carry adders (RCA)
- Carry lookahead adders (CLA)

Half Adder and Full Adder (1/2)

- Half adder:
  - 2 inputs: X, Y
  - 2 outputs: S (sum), C(carry)

$$\begin{array}{r} x \\ +\ y \\ \hline C\ \ S \end{array}$$

Truth table

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$S = \bar{x}y + x\bar{y} = x \oplus y$

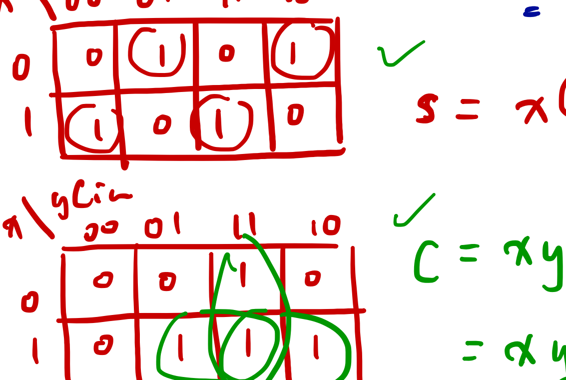
$C = x \cdot y$

- Full adder:
  - 3 inputs: X, Y, Cin (carry from previous digit)
  - 2 outputs: S (sum), C(carry)

$$\begin{array}{r} C_{in} \\ x \\ +\ y \\ \hline C\ \ S \end{array}$$

Truth table

x	y	Cin	S	C
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



$S = x \oplus y \oplus C_{in}$

$C = xy + xC_{in} + yC_{in}$

$= xy + (x \oplus y) \cdot C_{in}$

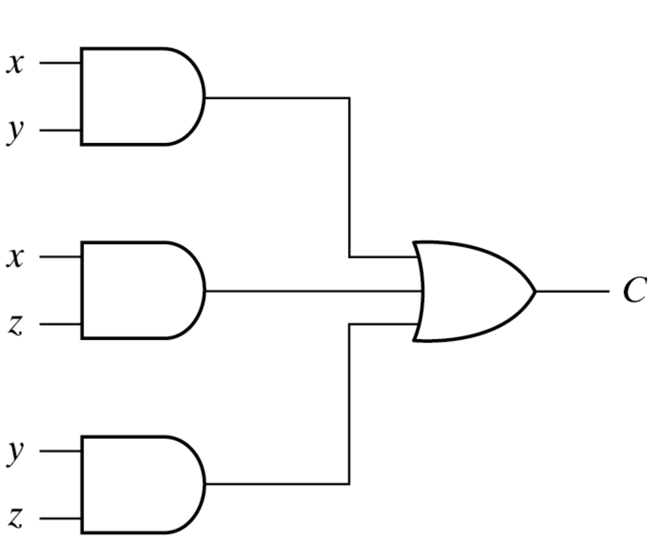
Half Adder and Full Adder (2/2)

Half adder

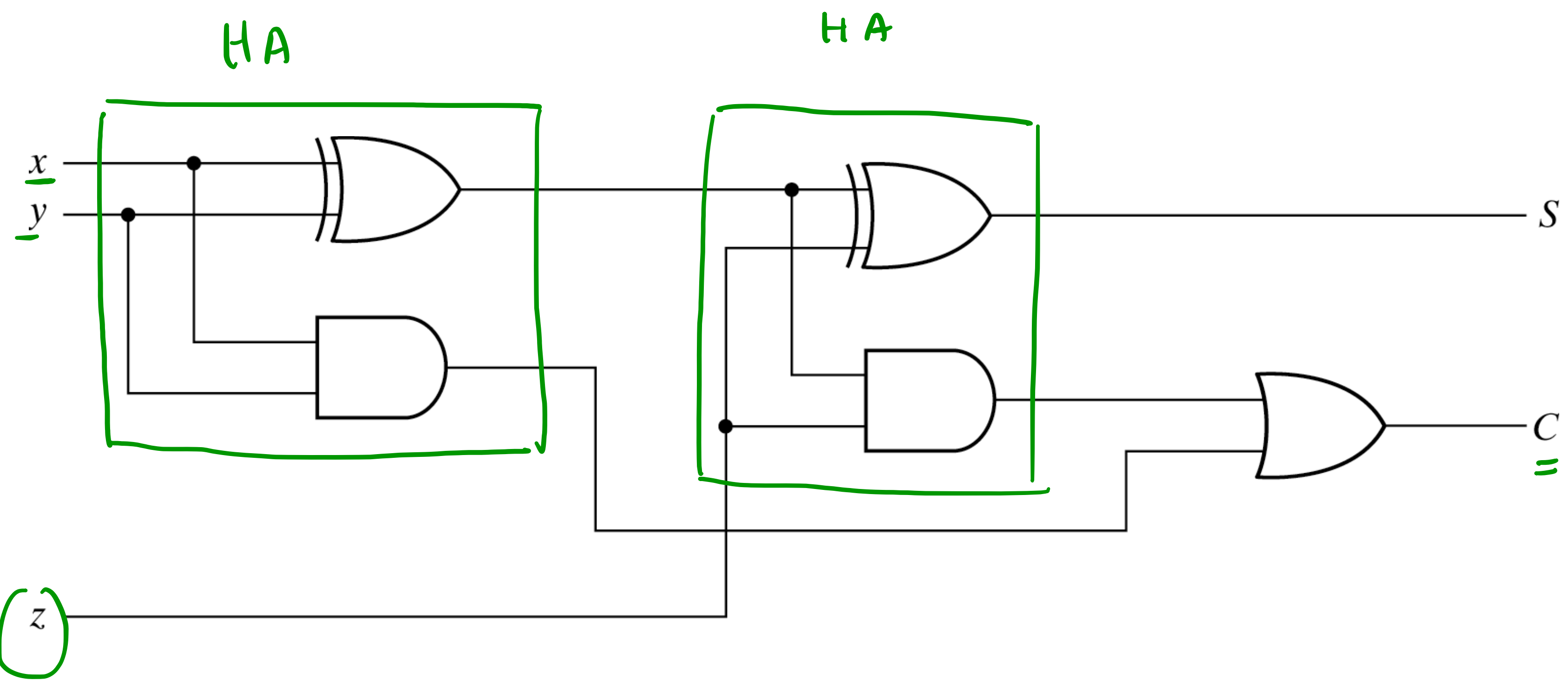
x
y'
x'
y
S
x
y
C

Full adder

x'
y'
z
x'
y
z'
x
y'
z'
x
y
z
S

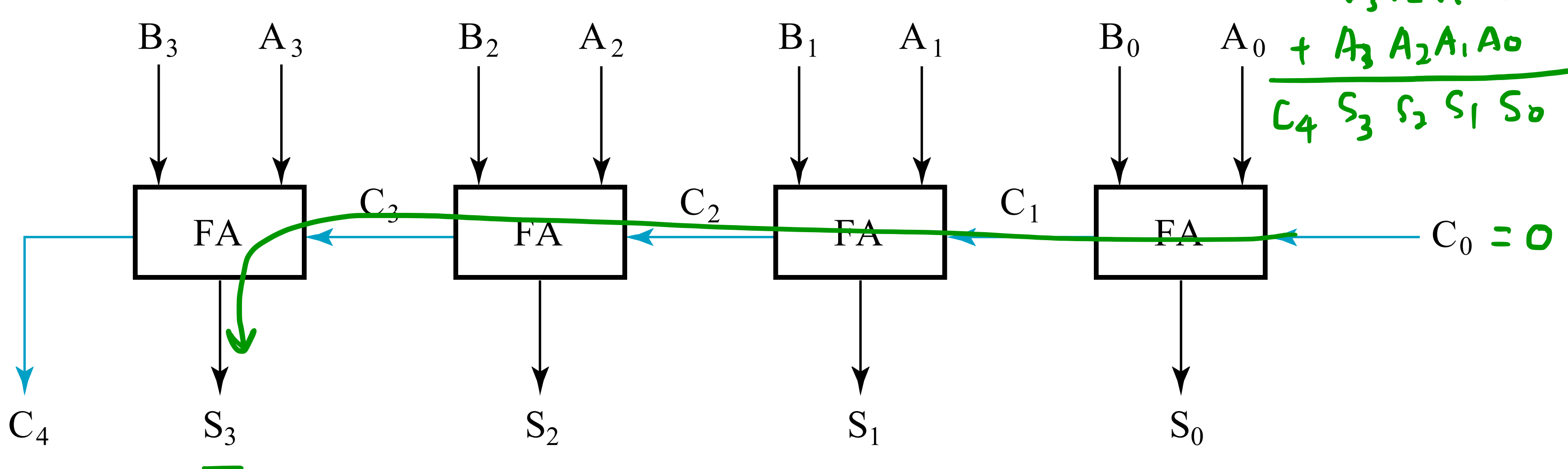


Full Adder Implementation with Half Adders



Unsigned 4-Bit Ripple Carry Adders (1/2)

- An unsigned four-bit ripple carry adder made from four 1-bit full adders:



- The computation time of an n-bit ripple carry adder grows linearly with length n due to the carry chain (critical path).

Unsigned 4-Bit Ripple Carry Adders (2/2)

TSING HUA UNIVERSITY

✓ $$S_i = f(A_i, B_i, C_i) = A_i \oplus B_i \oplus C_i$$

✓ $$C_{i+1} = g(A_i, B_i, C_i) = A_i \cdot B_i + B_i \cdot C_i + C_i \cdot A_i$$

✓ $$S_0 = f(A_0, B_0, C_0)$$

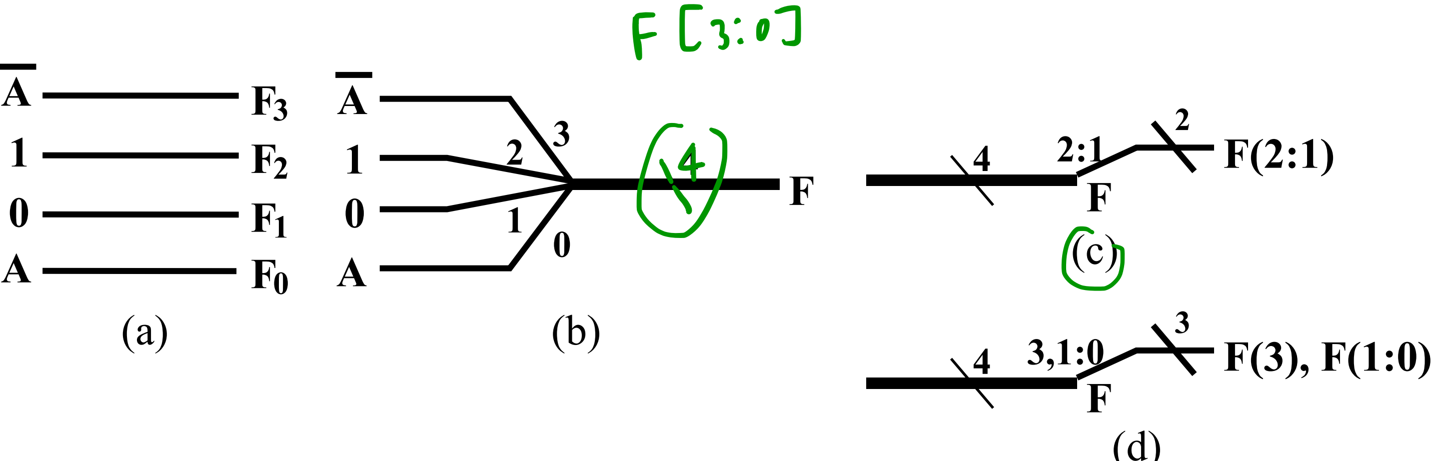
✓ $$C_1 = g(A_0, B_0, C_0)$$

$$S_1 = f(A_1, B_1, C_1)$$
$$C_2 = g(A_1, B_1, C_1)$$

$$S_2 = f(A_2, B_2, C_2)$$
$$C_3 = g(A_2, B_2, C_2)$$

$$S_3 = f(A_3, B_3, C_3)$$
$$C_4 = g(A_3, B_3, C_3)$$

Multiple Bit Notation



Carry Lookahead Adders (1/3)

- For a full adder, $C_{i+1} = x_i \cdot y_i + x_i \cdot C_i + y_i \cdot C_i$

$$= x_i \cdot y_i + (x_i \oplus y_i) \cdot C_i$$

$$= x_i \cdot y_i + (x_i \oplus y_i) \cdot [x_{i-1} \cdot y_{i-1} + (x_{i-1} \oplus y_{i-1}) \cdot C_{i-1}]$$

$$= G_i + P_i \cdot C_i$$

$G_i = x_i \cdot y_i$

Carry generate

$P_i = x_i \oplus y_i$

Carry propagate

$$S_i = x_i \oplus y_i \oplus C_i$$

$$= P_i \oplus C_i$$

	C_{i+1}	C_i: 1
		x_i: 0
+		y_i: 1
		S_i

Carry Lookahead Adders (2/3)

$$\begin{array}{rcccc} & c_2 & c_1 & c_0 & \\ & x_2 & x_1 & x_0 & \\ + & y_2 & y_1 & y_0 & \\ \hline c_3 & s_2 & s_1 & s_0 & \end{array}$$

$$c_3 = x_2 y_2 + (x_2 \oplus y_2) \cdot c_2$$

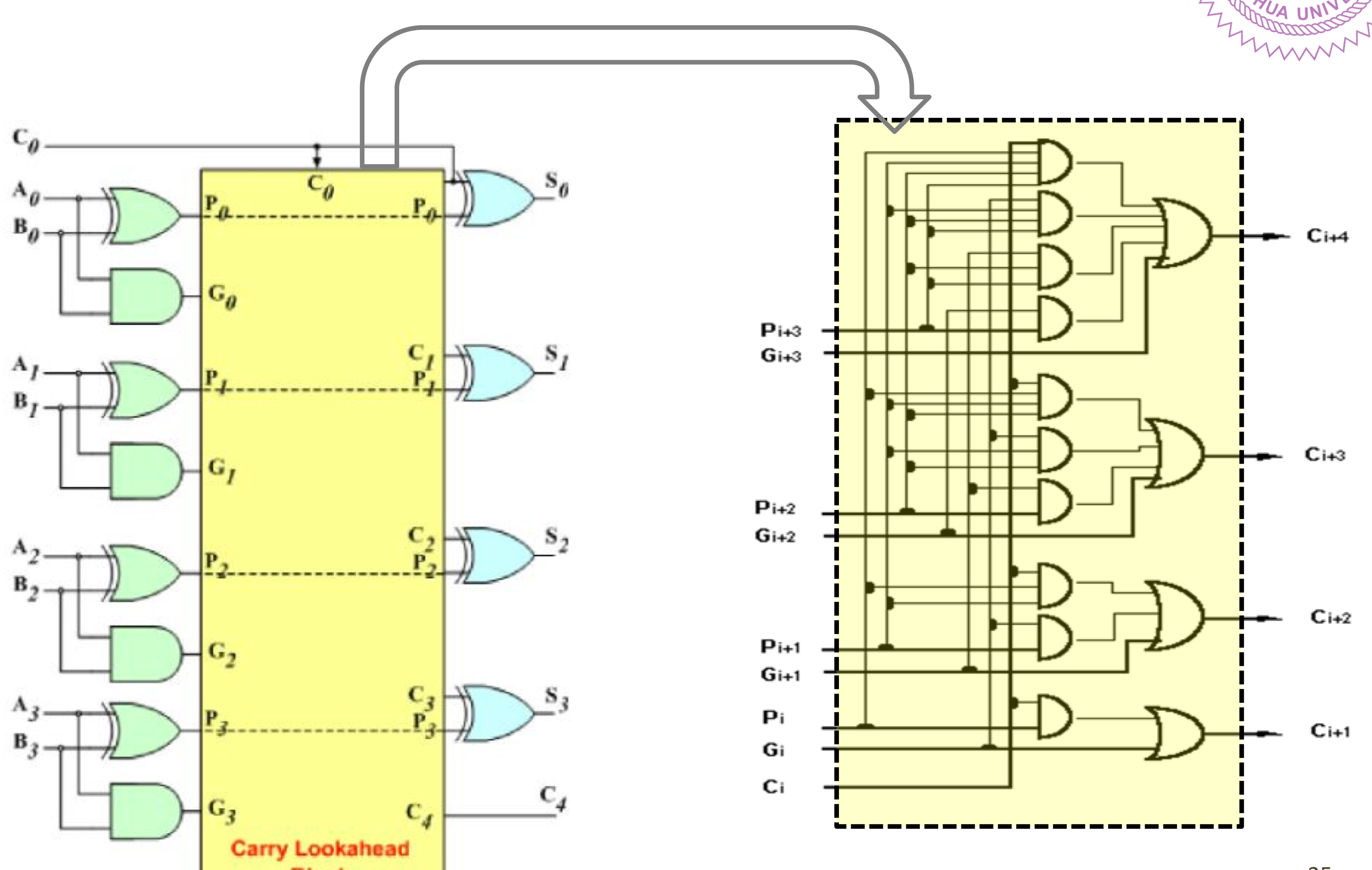
$$= x_2 y_2 + (x_2 \oplus y_2) \cdot [x_1 y_1 + (x_1 \oplus y_1) \cdot c_1]$$

$$= x_2 y_2 + (x_2 \oplus y_2) \cdot [x_1 y_1 + (x_1 \oplus y_1) \cdot (x_0 y_0 + (x_0 \oplus y_0) \cdot c_0)]$$

$$c_3 = G_2 + P_2 [G_1 + P_1 \cdot (G_0 + P_0 \cdot c_0)]$$

4-bit CLA

Carry Lookahead Adders (3/3)



Binary Adder/Subtractor

A + B

A − B = A + (−B)

- Binary subtraction can be performed by adding the minuend to the 2's complement of the subtrahend.

B	M	b
0	0	0
0	1	1
1	0	1
1	1	0

addition

$b_0 = B$

M = 0 add

M = 1 sub

B_3 A_3 B_2 A_2 B_1 A_1 B_0 A_0

M = 0

b_3 b_2 b_1 b_0

C C_4 C_3 C_2 C_1 C_0

FA FA FA FA

S_3 S_2 S_1 S_0

V overflow

Decimal Adders

Decimal Adders (1/3)

- Addition of 2 decimal digits in BCD
  - {Cout,S}=A+B+Cin
    - $S=S_8S_4S_2S_1$, $A=A_8A_4A_2A_1$, $B=B_8B_4B_2B_1$
  - A digit in BCD cannot exceed 9, add 6 (0110) for final correction.
- Case 1: sum $\leq$ 9, then the sum is correct.
- Case 2: sum > 9, add "6" (0110_2) to the sum.

Decimal value	BCD digit
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

∘ Case 1:

```
  0001 (+1)
+ 0100 (+4)
-----------
  0101 (+5)
```

∘ Case 2:

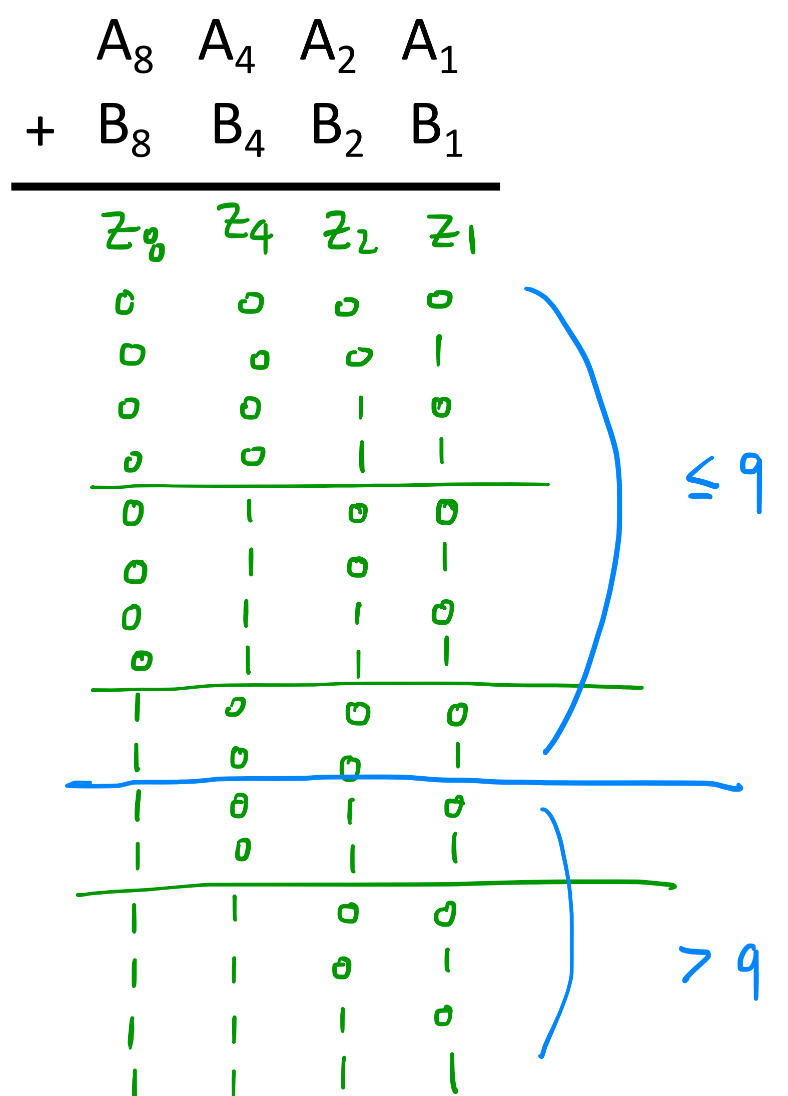
```
   1000 (+8)
+  1001 (+9)
------------
  10001 (+17) binary
+  0110
------------
0001 0111  BCD
   1    7
```

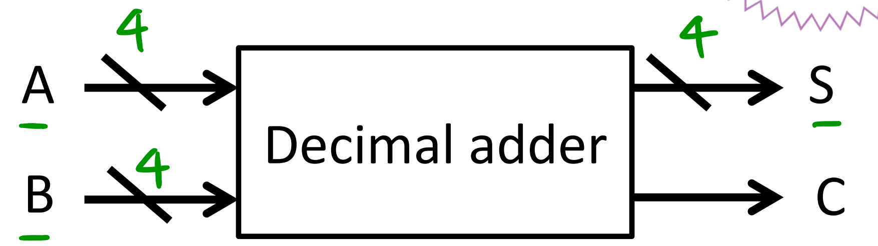
(+10) 1010 (binary)

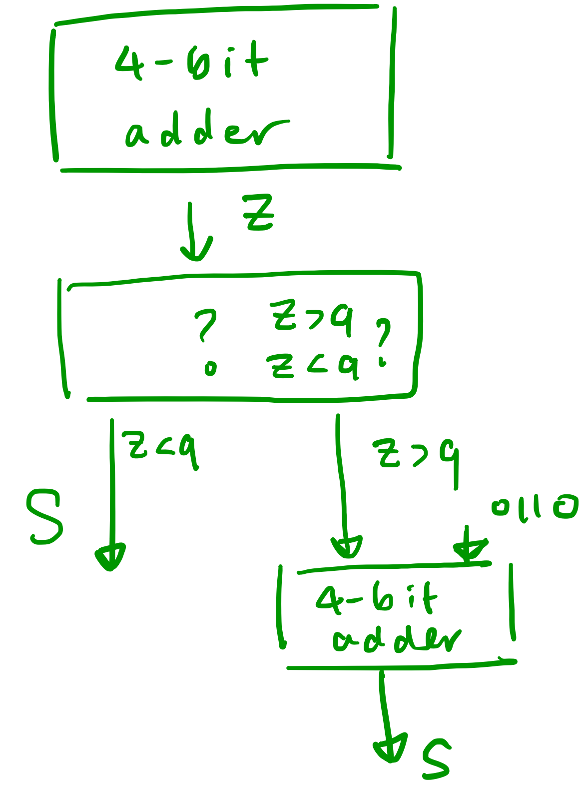
00010000 (BCD)

(+11)

Decimal Adders (2/3)







Decimal Adders (3/3)

B_8 B_4 B_2 B_1 A_8 A_4 A_2 A_1

Carry out

K

4- bit binary adder

Z_8 Z_4 Z_2 Z_1

Carry in

$Z_8 \cdot Z_4$

Output carry

$Z_8 \cdot Z_2$

0

0 1 1 0

Z_8 Z_4 Z_2 Z_1

4- bit binary adder

S_8 S_4 S_2 S_1

- Multiplication/ division by 2^n

$\Rightarrow$ shift left / right n digits

$b[3:0] \times 100_{\text{binary}} = b[3:0]00$

$b[3:0] \div 100 = b_3 b_2 . b_1 b_0$

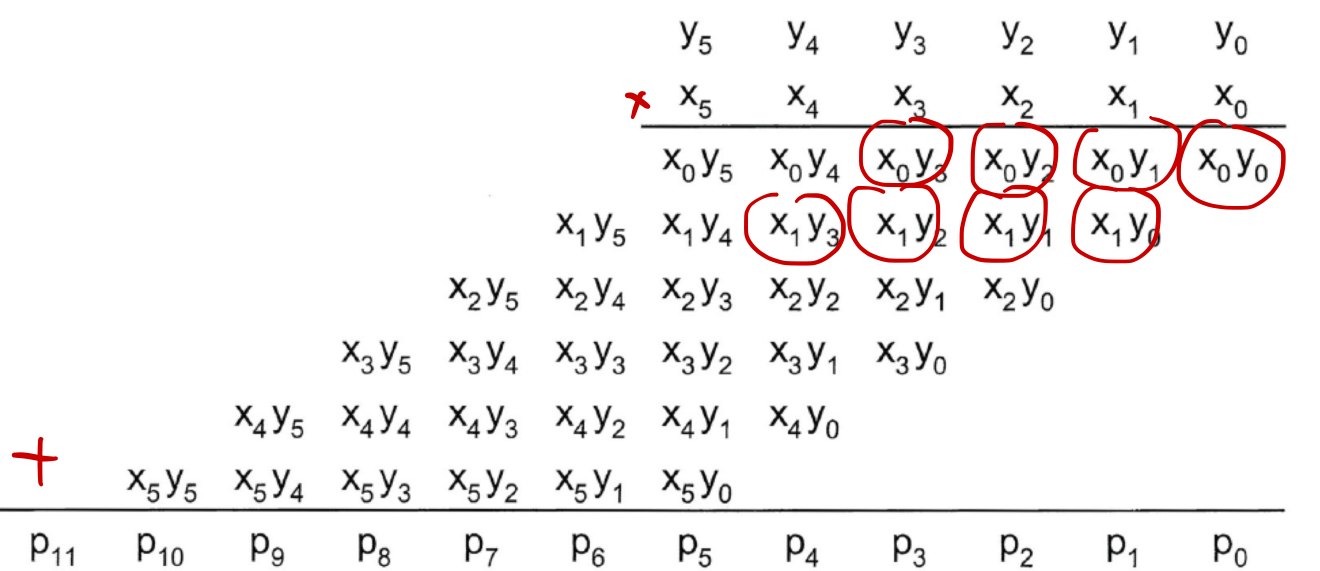
Binary Multiplier

Multiply/Divide by 2

- Multiplication/division by 2^n
  - Shift left/right

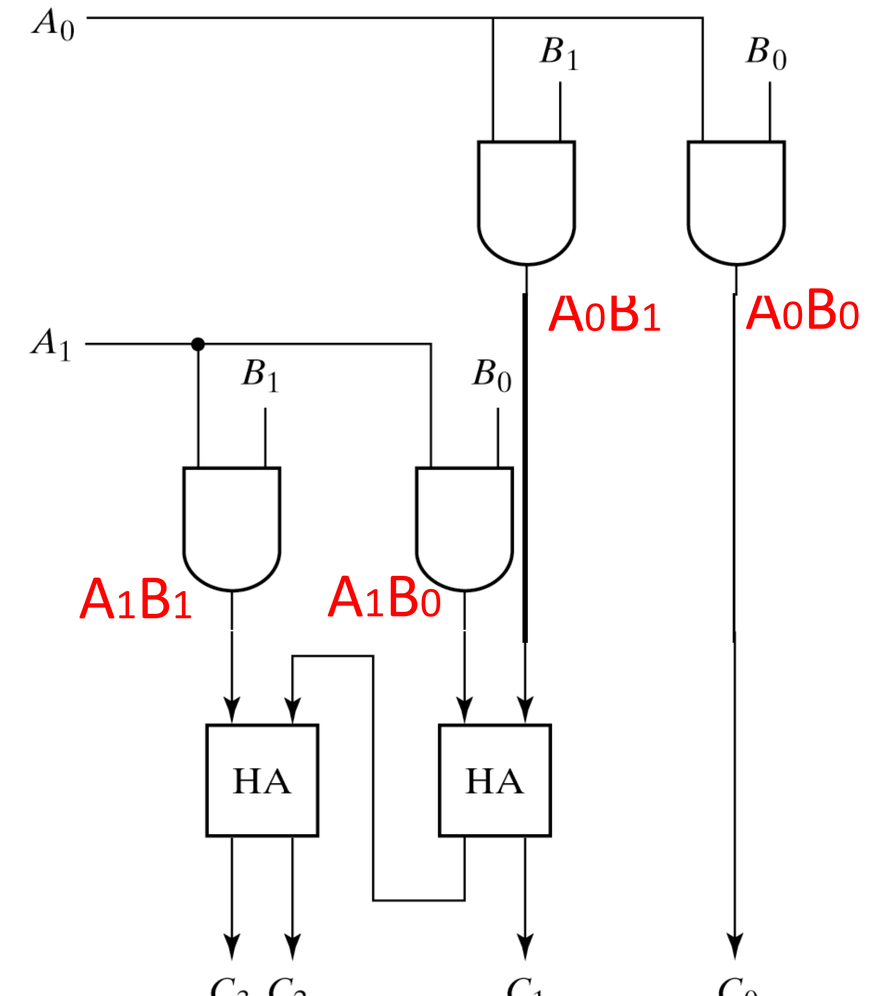
M-Bit x N-Bit Multiplication

$$P = \left(\sum_{j=0}^{M-1} y_j 2^j\right)\left(\sum_{i=0}^{N-1} x_i 2^i\right) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_i y_j 2^{i+j}$$



2-Bit x 2-Bit Binary Multiplier

		B_1	B_0
	x	A_1	A_0
		B_1A_0	B_0A_0
+	B_1A_1	B_0A_1	
C_3	C_2	C_1	C_0



B[3:0] A[2:0]

4-Bit x 3-Bit Binary Multiplier

A_0 B_3 B_2 B_1 B_0

A_1 B_3 B_2 B_1 B_0

0

Addend Augend

4-bit adder

Sum and output carry

A_2 B_3 B_2 B_1 B_0

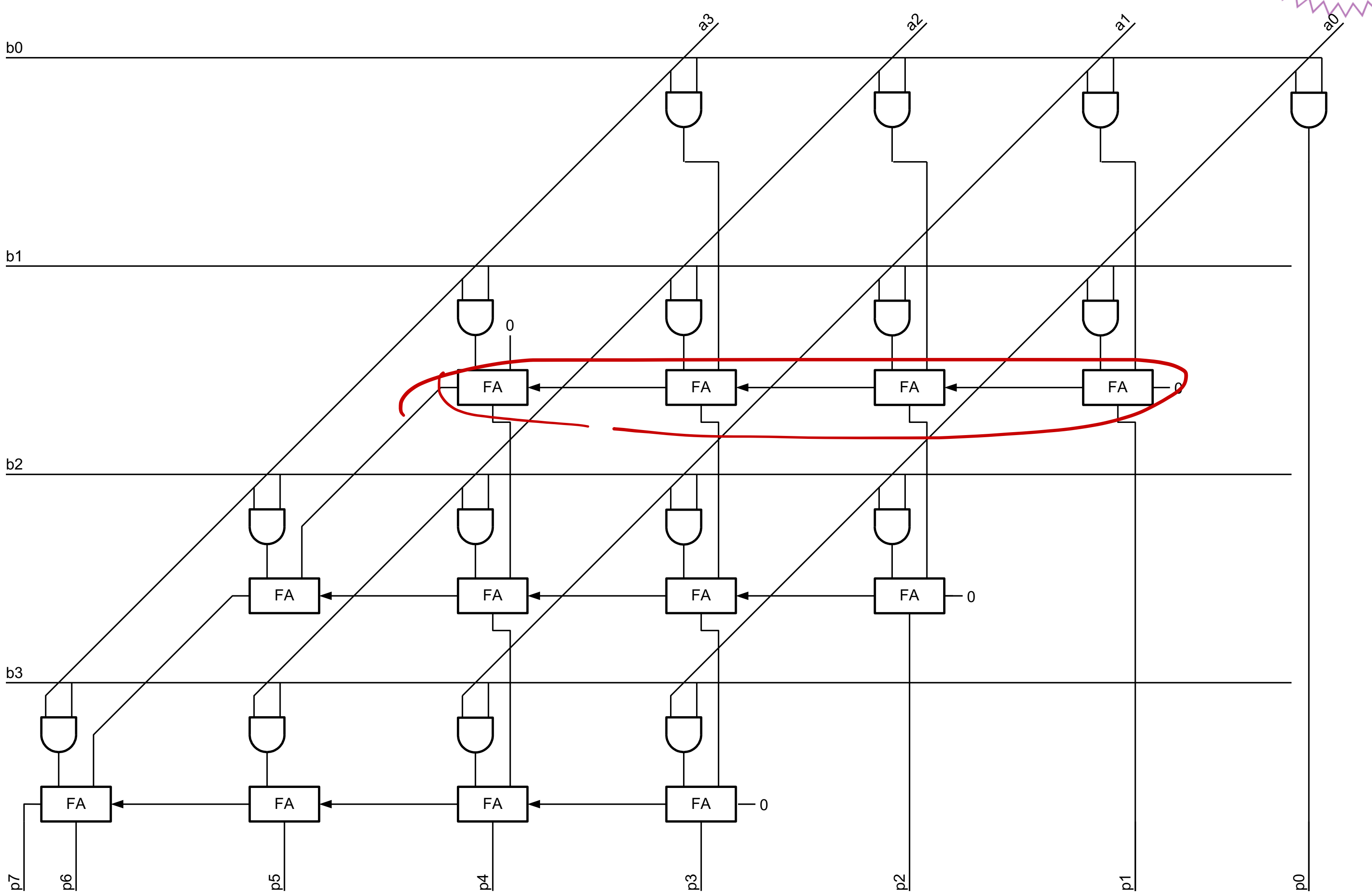
Addend Augend

4-bit adder

Sum and output carry

C_6 C_5 C_4 C_3 C_2 C_1 C_0

Array Multiplier



Decoders

One-Hot Representation

- Represent a set of N elements with N bits.
- Exactly one bit is set.

Binary	One-hot
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

Enabling Function

- Enabling permits an input signal to pass through to an output.

EN	X	F
0	0	0
0	1	0
1	0	0
1	1	1

- When EN = 0, buffer is disabled, F = 0.
- When EN = 1, buffer is enabled, F = X.

- In general, when a block is disabled, the output can be a fixed value or high impedance.

disabled

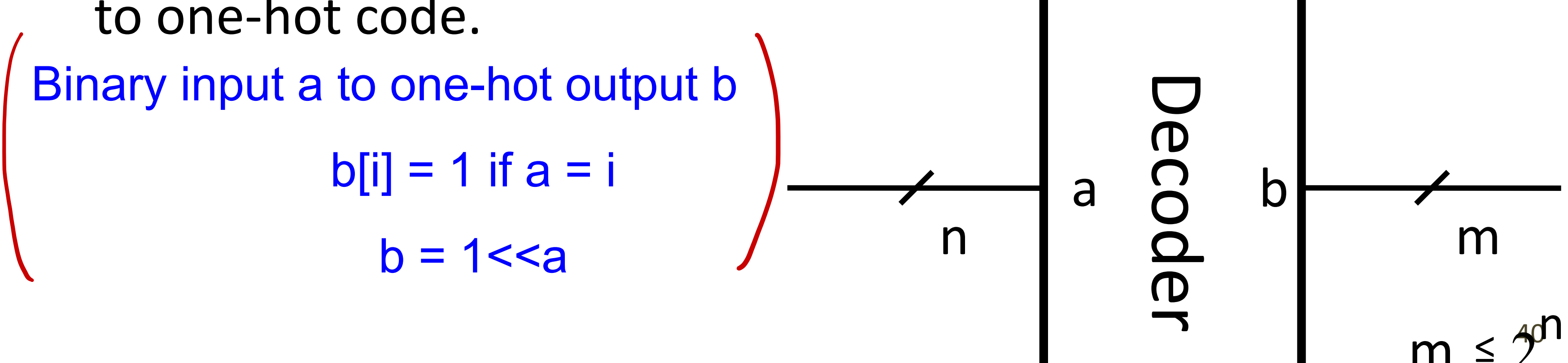
enabled $F = x$

$F = EN \cdot X$

Decoding

binary code → one-hot code

- n-to-m line decoder: the conversion of an *n*-bit input code to an *m*-bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code.
  - Output variables are mutually exclusive. Only one output can be 1 at a time.
  - Binary to one-hot decoder.
- Circuits that perform decoding are called *decoders.*
- A binary one-hot decoder converts a symbol from binary code to one-hot code.



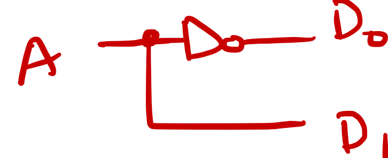
Decoder Examples

- 1-to-2-Line Decoder

A	D_1	D_0
0	0	1
1	1	0

$D_0 = \overline{A}$

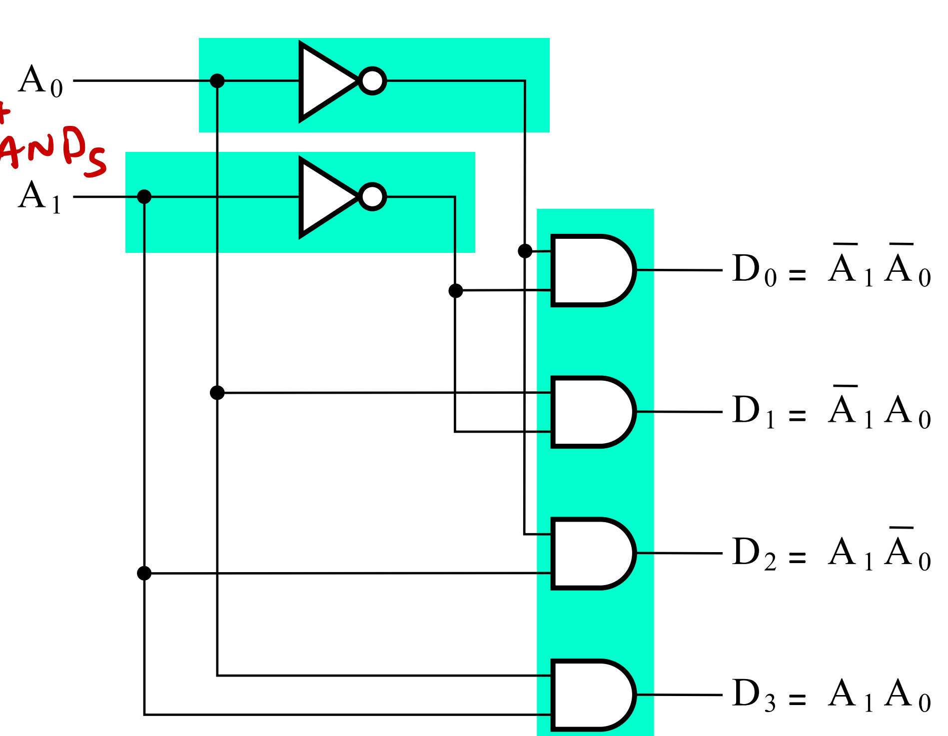
$D_1 = A$



- 2-to-4-Line Decoder
  - Two 1-to-2 line decoders + 4 ANDs

A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

(a)



(b)

Decoder Expansion

- 3-to-8-line decoder
  - Number of output ANDs = 8
  - Number of inputs to decoders driving output ANDs = 3
  - Closest possible split to equal
    - 2-to-4-line decoder
    - 1-to-2-line decoder
  - 2-to-4-line decoder
    - Number of output ANDs = 4
    - Number of inputs to decoders driving output ANDs = 2
    - Closest possible split to equal
      - Two 1-to-2-line decoders

3-to-8 Line Decoder (1/2)

Truth table page 33.

$A_2 \rightarrow \overline{A_2}$

$A_1 \rightarrow \overline{A_1}$

$A_0 \rightarrow \overline{A_0}$

$D_7 = A_2 A_1 A_0$

$D_6 = A_2 A_1 \overline{A_0}$

$D_5 = A_2 \overline{A_1} A_0$

$D_4 = A_2 \overline{A_1} \overline{A_0}$

$D_3 = \overline{A_2} A_1 A_0$

$D_2 = \overline{A_2} A_1 \overline{A_0}$

$D_1 = \overline{A_2} \overline{A_1} A_0$

$D_0 = \overline{A_2} \overline{A_1} \overline{A_0}$

3-to-8 Line Decoder (2/2)

4 2-input ANDs

8 2-input ANDs

A_0

A_1

A_2

D_0

D_1

D_2

D_3

D_4

D_5

D_6

D_7

2-to-4-Line decoder

1-to-2-Line decoders

3-to-8 Line decoder

$D_0 = \bar{A}_2\bar{A}_1\bar{A}_0$

$D_4 = A_2\bar{A}_1\bar{A}_0$

7-to-128 Line Decoder (1/2)

- 7-to-128-line decoder
  - Number of output ANDs = 128
  - Number of inputs to decoders driving output ANDs = 7
  - Closest possible split to equal
    - 4-to-16-line decoder
    - 3-to-8-line decoder
  - 4-to-16-line decoder
    - Number of output ANDs = 16
    - Number of inputs to decoders driving output ANDs = 2
    - Closest possible split to equal
      - 2 2-to-4-line decoders

7-to-128 Line Decoder (2/2)

A6, A5, A4, A3 → 4-to-16 dec → x_{15}, x_{14}, ..., x_0

A2, A1, A0 → 3-to-8 dec → y_7, y_6, ..., y_0

x_{15}, y_7 → AND → D_{127}

x_{15}, y_6 → AND → D_{126}

⋮

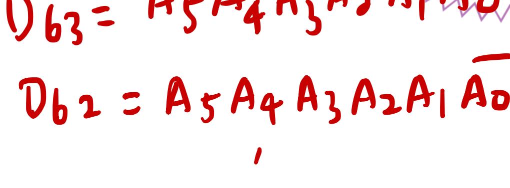
x_0, y_1 → AND → D_1

x_0, y_0 → AND → D_0

Practice :

gate input ?

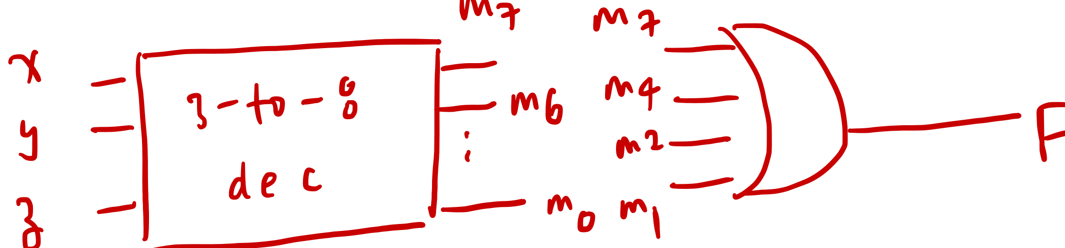
Advantages of Dividing Large Decoder



- 6-to-64 decoder requires
  - 64 6-input AND gates (384 inputs)
- 6-to-64 decoder using 2-to-4 decoders requires
  - 12 2-input AND gates (24 inputs)
  - 64 3-input AND gates (192 inputs)
- Faster, smaller, lower power

Combinational Logic Implementation with Decoders (1/2)

- Any combinational circuit with n inputs and m outputs can be implemented with an n-to-2^n decoder and m OR gates.
- Example: $F(x,y,z)=\Sigma(1,2,4,7)$



Combinational Logic
Practice Implementation with Decoders (2/2)

- Example: 3-bit prime detector

$F(x,y,z)=\Sigma(1,2,3,5,7)$

Encoders

Encoding

- Encoding: the opposite of decoding - the conversion of an *m*-bit input code to a *n*-bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code.
- Circuits that perform encoding are called *encoders.*
- An encoder has 2^n (or fewer) input lines and *n* output lines which generate the binary code corresponding to the input values.
- Typically, an encoder converts a code containing exactly one bit that is 1 to a binary code corresponding to the position in which the 1 appears.

A 4-to-2 Encoder

A_3	A_2	A_1	A_0	B_1	B_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

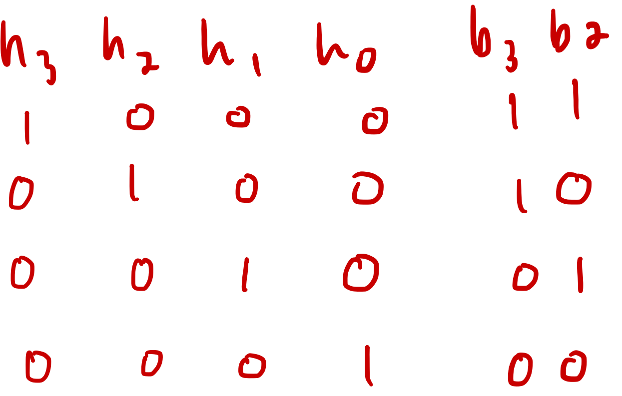
$B_1 = A_3 + A_2$

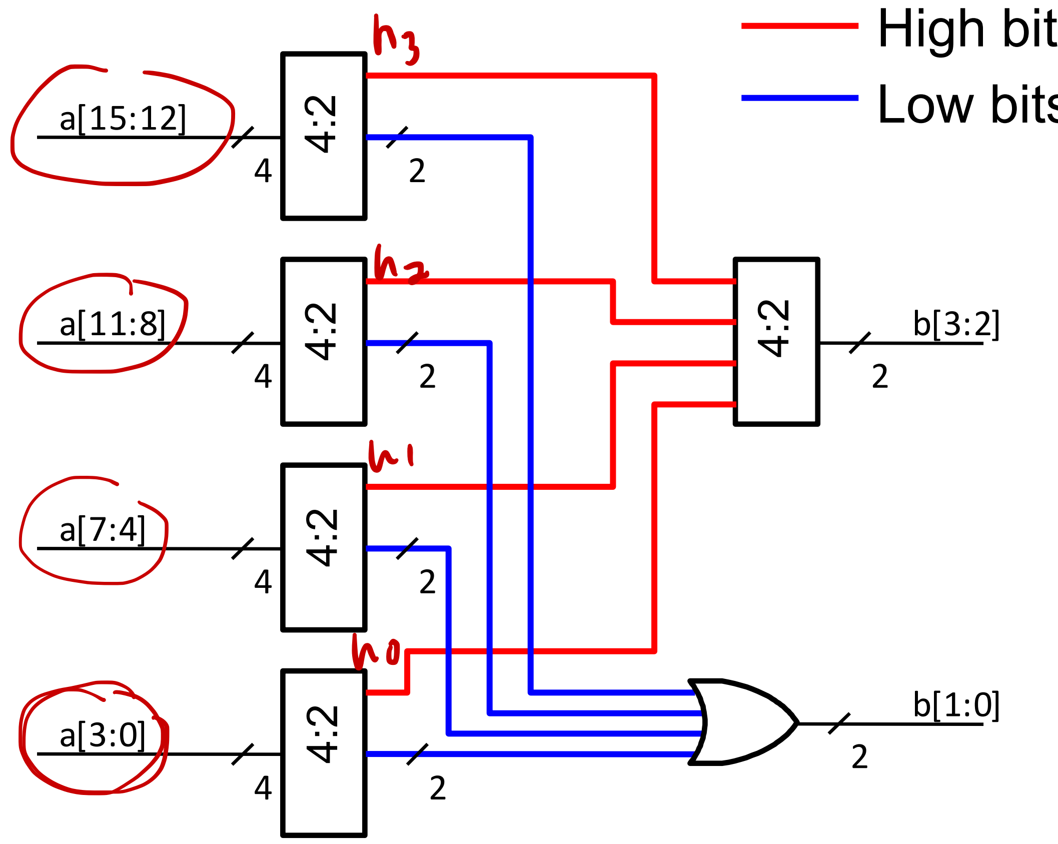
$B_0 = A_3 + A_1$

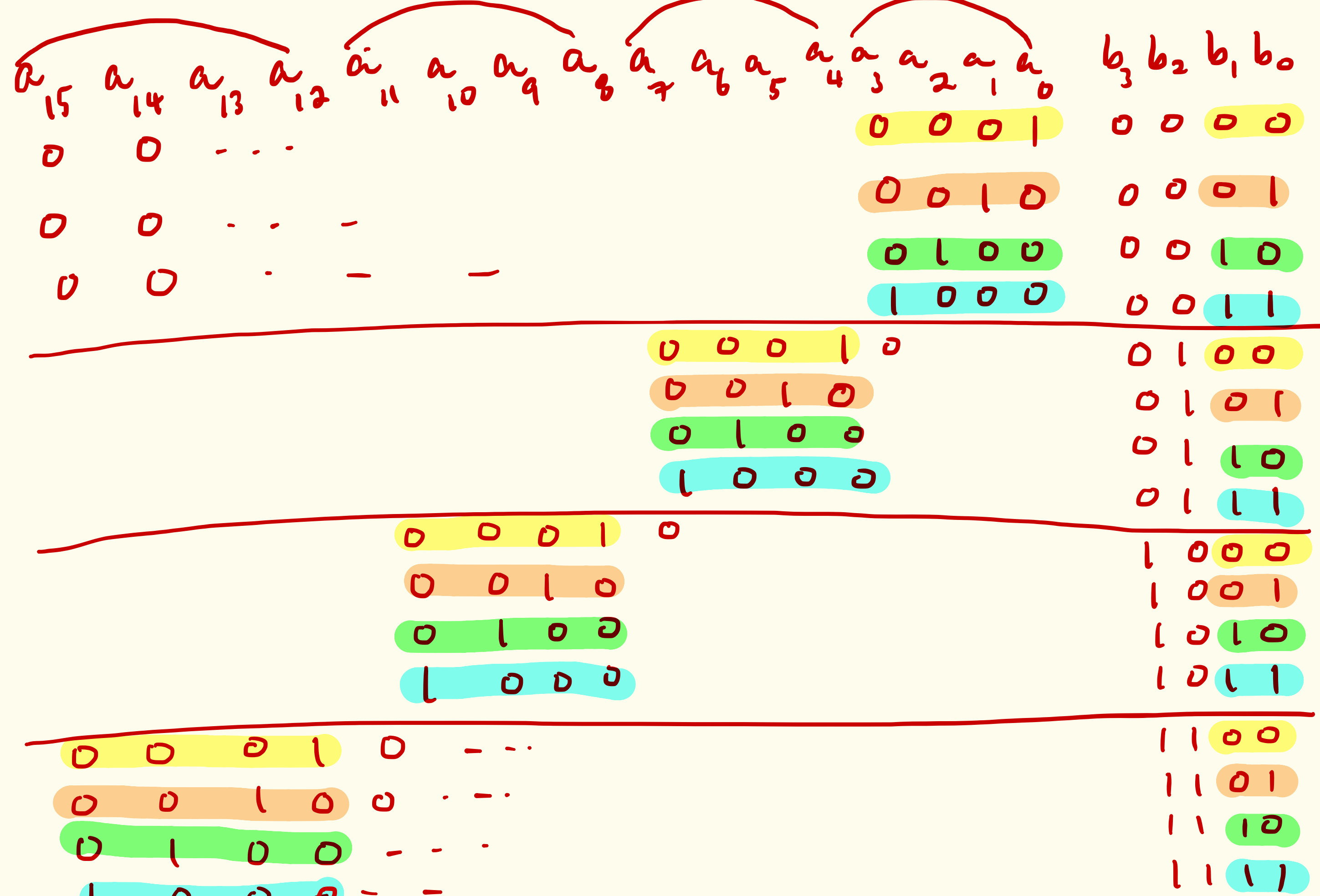
Design a Larger Encoder

16-to-4

- Additional summary output (High bits) is true if any input of the encoder is true.
- First rank encodes low bits, second rank encodes high bits.





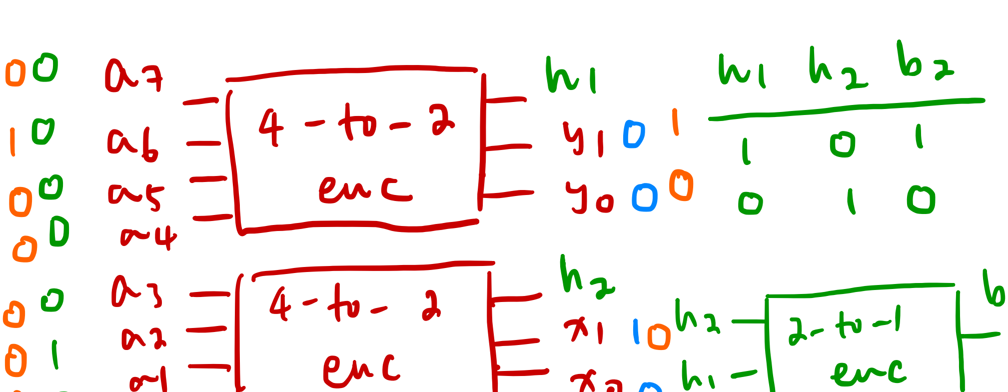


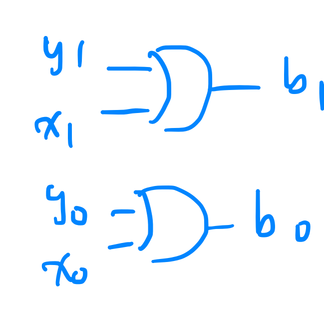
a_{15} a_{14} a_{13} a_{12} a_{11} a_{10} a_9 a_8 a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
b_3 b_2 b_1 b_0

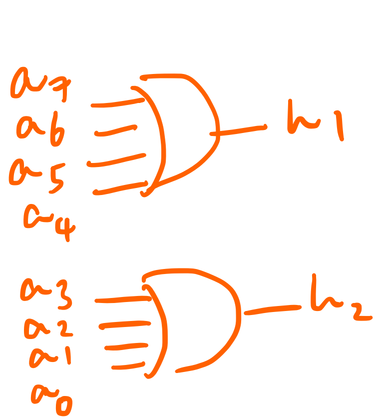
8-to-3 Line Encoder

input a[7:0] , output b[2:0]   h : high bit

a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	b_2	b_1	b_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1





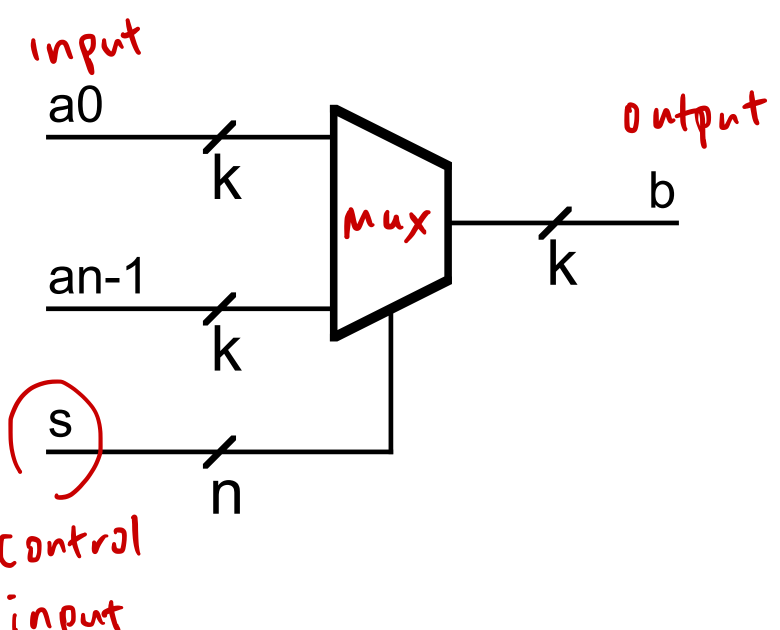


多工

Multiplexers

Multiplexers

- Multiplexer: select one binary information from n inputs.
  - n k-bit inputs
  - n-bit one hot select signal s
- Multiplexers are used as data selectors.



One Hot vs. Binary Select

- One hot: n k-bit input lines, one n-bit control line

3-to-1 mux

s	b
0 0 1	a_0
0 1 0	a_1
1 0 0	a_2

- Binary select: n k-bit input lines, one m-bit control line

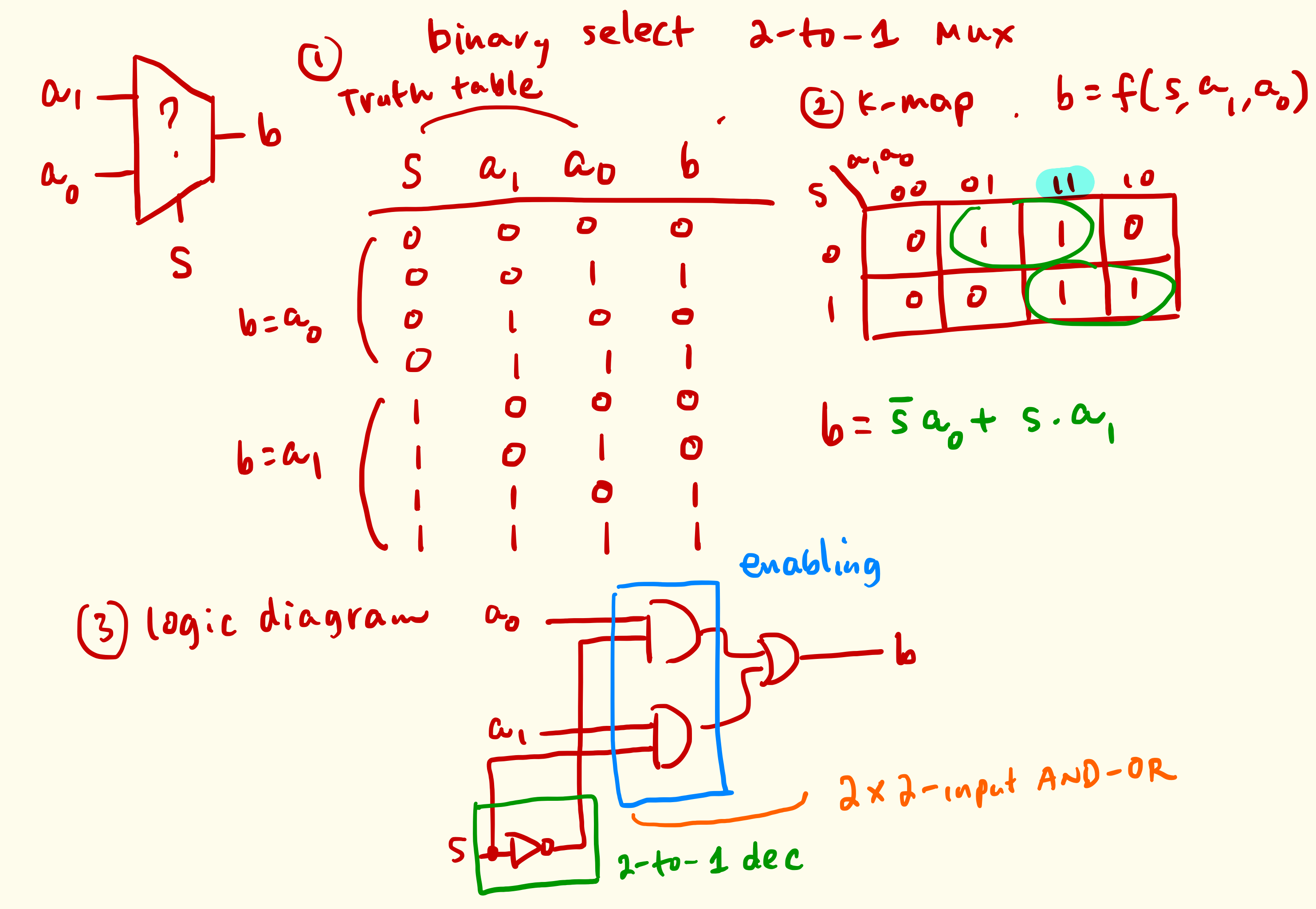
3-to-1 mux

s	b
0 0	a_0
0 1	a_1
1 0	a_2
1 1	x

binary

2-to-1 Multiplexer (1/2)

- The multiplexer circuit shown:
  - 1:2-line decoder
  - 2 enabling circuits
  - 2-input OR gate
- To obtain a basis for multiplexer expansion, we combine the Enabling circuits and OR gate into a 2 x 2 AND-OR circuit:
  - 1:2-line decoder
  - 2 x 2 AND-OR
- In general, for an 2^n:1-line multiplexer:
  - n:2^n-line decoder
  - 2^n x 2 AND-OR


binary select 2-to-1 mux
① Truth table
s	a1	a0	b
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
b = a0
b = a1
a1
a0
?
b
s
② K-map . b = f(s, a1, a0)
a1a0
s
00 01 11 10
0: 0 1 1 0
1: 0 0 1 1
b = s̄ a0 + s . a1
③ logic diagram
a0
a1
s
b
enabling
2x2-input AND-OR
2-to-1 dec

2-to-1 Multiplexer (2/2)

4:1 Multiplexer (1/2)

binary select

- 2:2^2-line decoder
- 2^2 x 2 AND-OR

2-to-4 dec

Decoder

S_1

S_0

S[1:0]: binary select

d_0

d_1

d_2

d_3

4 x 2 AND-OR

I_0

I_1

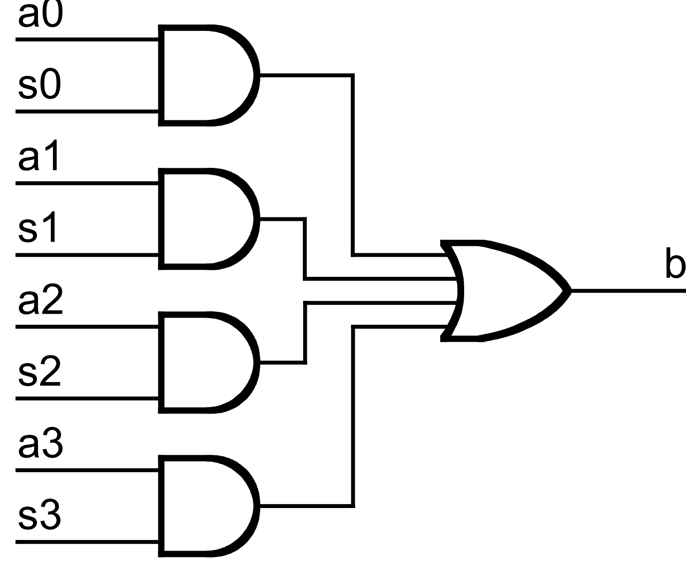
I_2

I_3

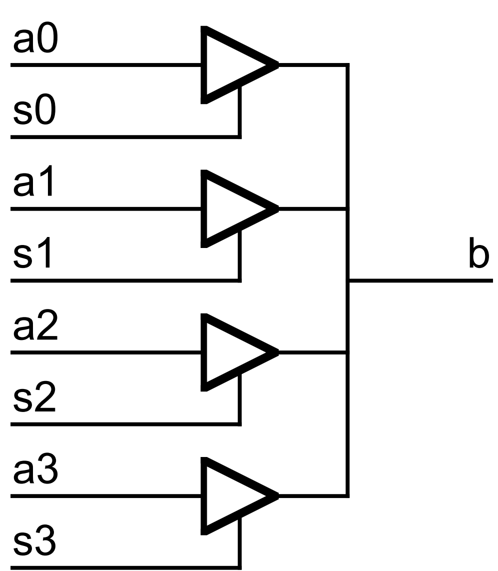
Y

4:1 Multiplexer (2/2)

S[3:0]: one hot

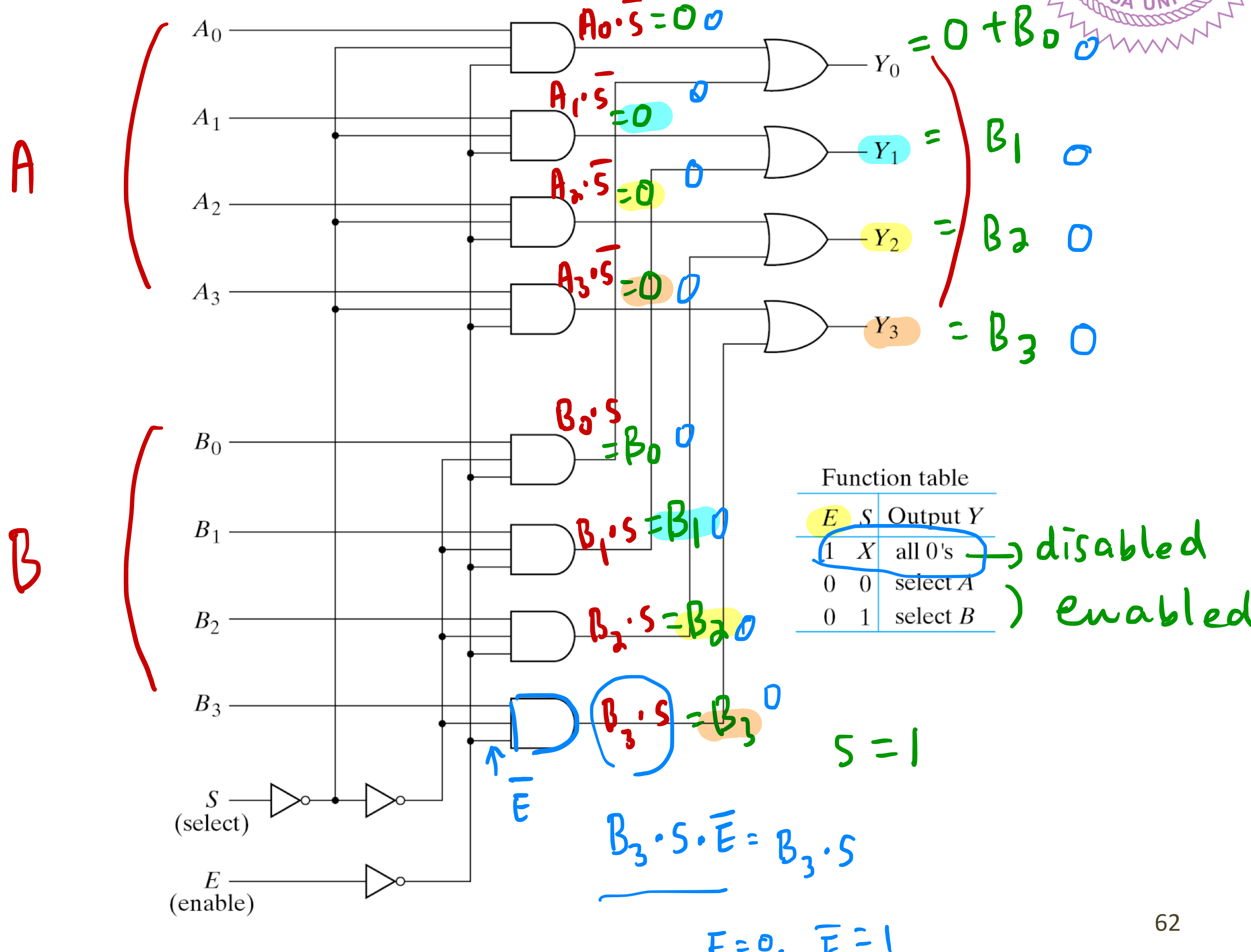


AND-OR implementation



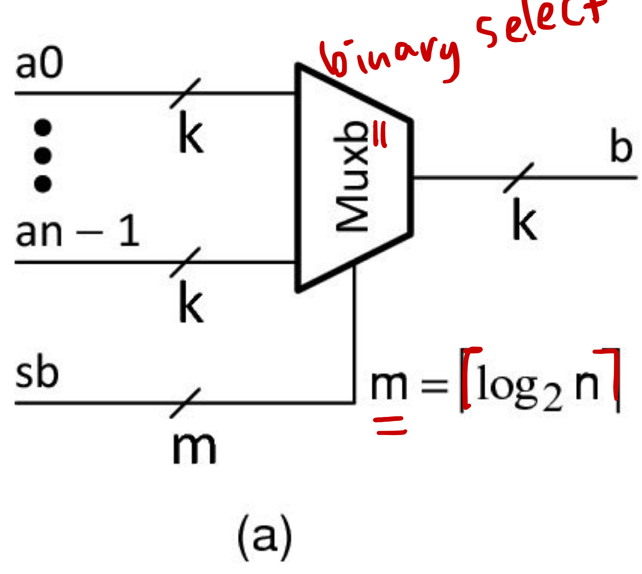
3-state buffer implementation

Quadruple 2:1 MUX (4-Bit 2:1 MUX)

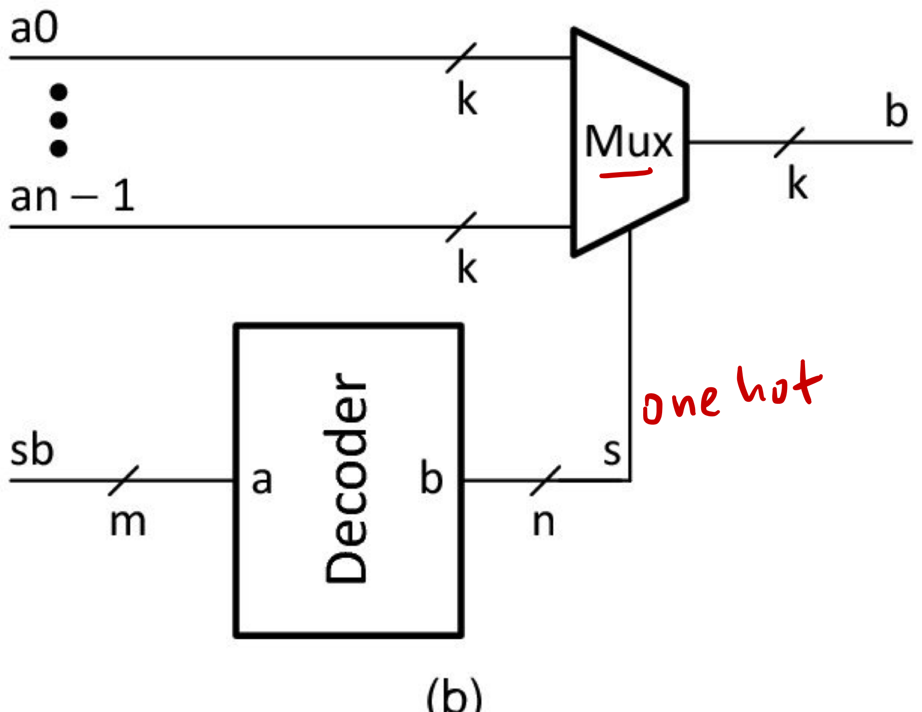


E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

K-Bit n:1 MUX



(a)

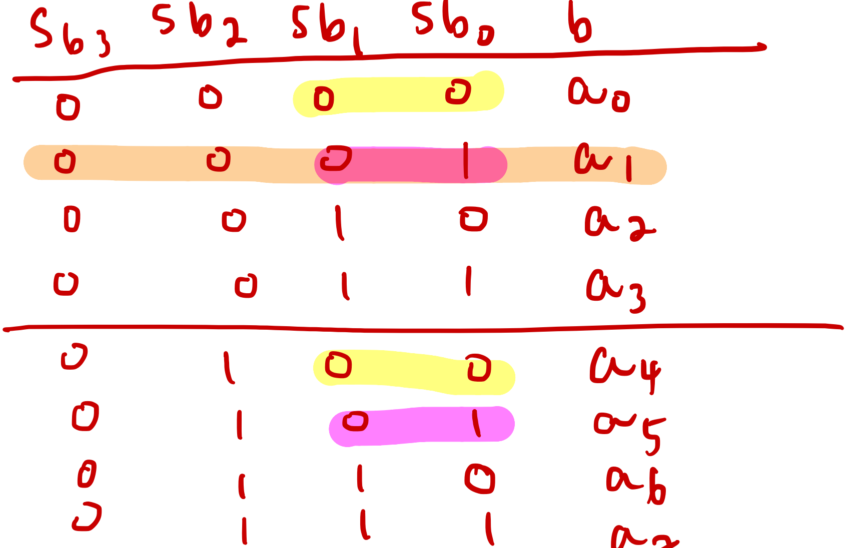


(b)

16-to-1

Large Binary Select MUX

- Larger binary select MUX can be constructed from smaller binary select MUXes.
- 16:1 MUX can be constructed from five 4:1 MUXes.



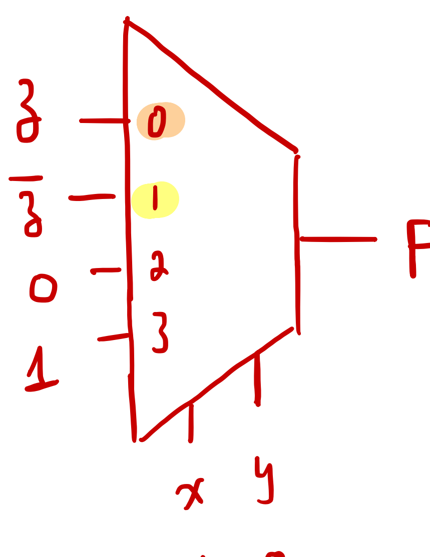
a0 a1 a2 a3 Muxb
a4 a5 a6 a7 Muxb
a8 a9 a10 a11 Muxb
a12 a13 a14 a15 Muxb
x0 x1 x2 x3 Muxb b
$sb_{3:2}$
$sb_{1:0}$

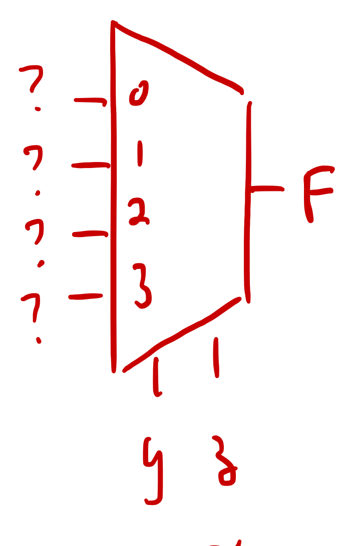
Boolean Function Implementation with a MUX (1/2)

$$F(x, y, z) = \sum(1, 2, 6, 7)$$

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

- x, y : select inputs
- z : data input





Boolean Function Implementation with a MUX (2/2)

- Assign an ordering sequence of the n-1 input variables (x,y) to the selection input of MUX.
- The last variable (z) will be used for the input lines.
- Construct the truth table.
- Consider a pair of consecutive minterms starting from m_0.
- Determine the input lines according to the last variable (z) and output signals (F) in the truth table.

Arbiters and Priority Encoders

Arbiters

- Arbiter handles requests from multiple devices to use a single resource.
  - Also called find-first-one (FF1) unit.
  - Accepts an arbitrary input signal r and outputs one-hot signal g to indicate the least significant 1 (or the most significant 1) of the input.
- Example: input 01011100
  - Output: 00000100 (least significant 1)
  - Output: 01000000 (most significant 1)

inputs: $r[i]$, $w[i]$ — outputs: $g[i]$, $w[i+1]$

$r[i]$	$w[i]$	$g[i]$	$w[i+1]$
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

$w[i]$: no one yet

when $w[i]=1$, it means there is no "1" yet.

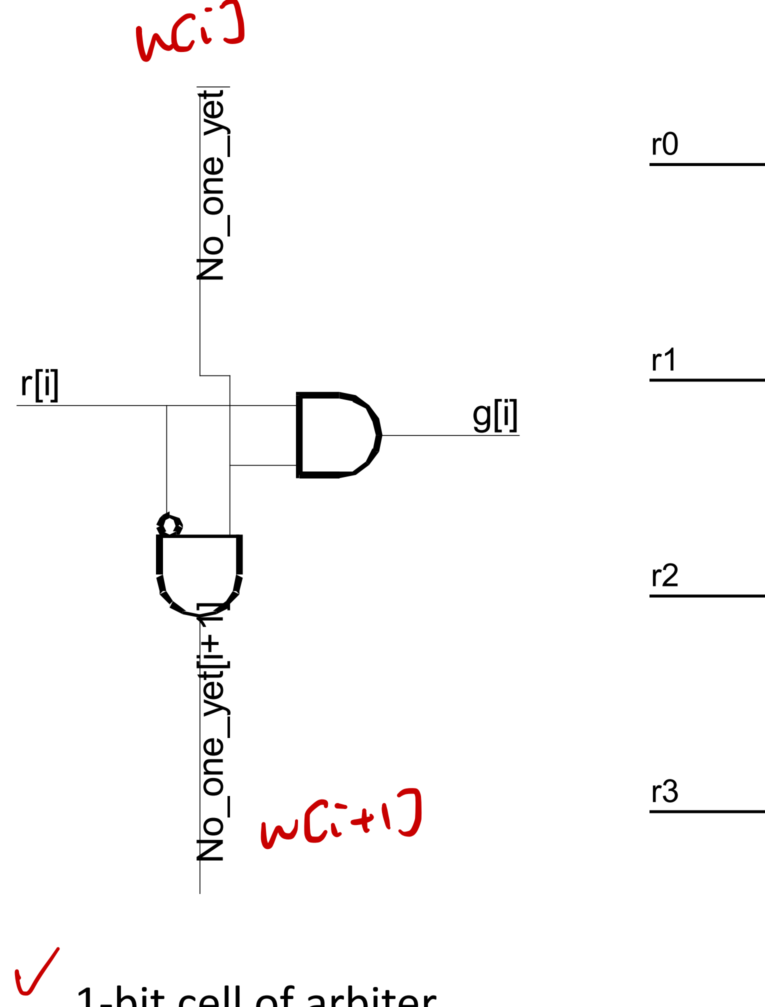
$r[i]$ → [ ] → $g[i]$

$w[i]$ → [ ] → $w[i+1]$

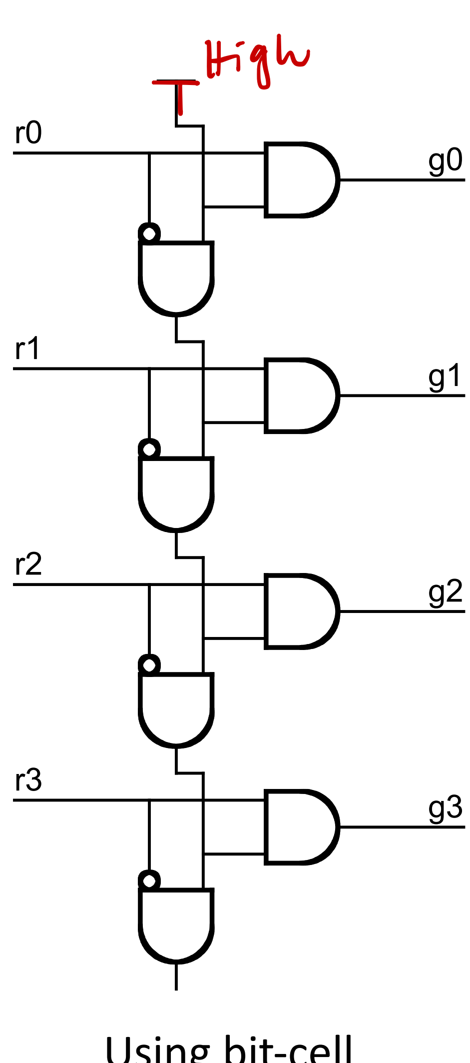
✓ $g[i] = r[i] \cdot w[i]$

✓ $w[i+1] = \overline{r[i]} \cdot w[i]$

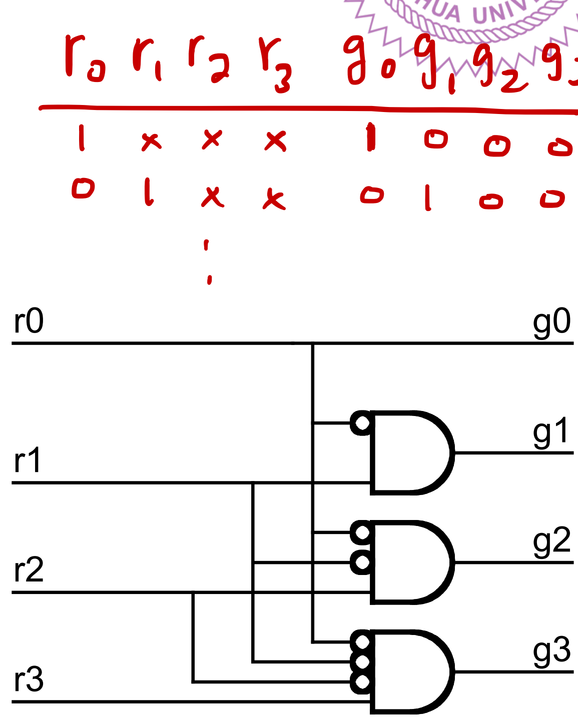
Arbiters Implementation



1-bit cell of arbiter



Using bit-cell



Using lookahead

$g_0 = f(r_0, r_1, r_2, r_3)$

g_1

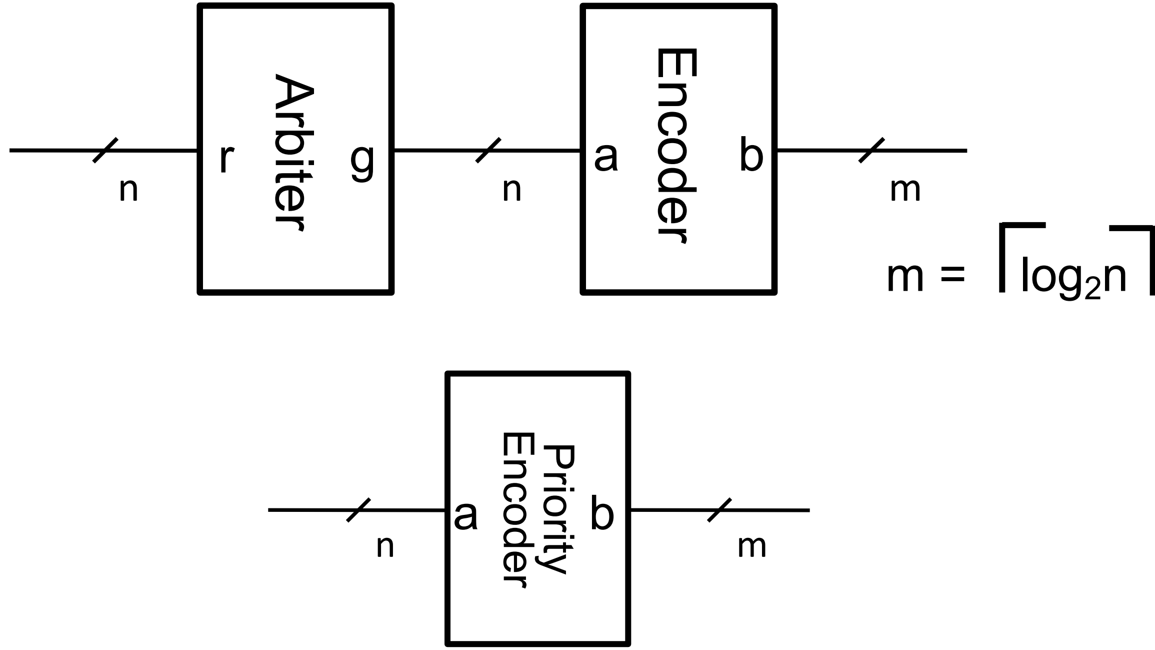
g_2

g_3

Priority Encoder (1/2)

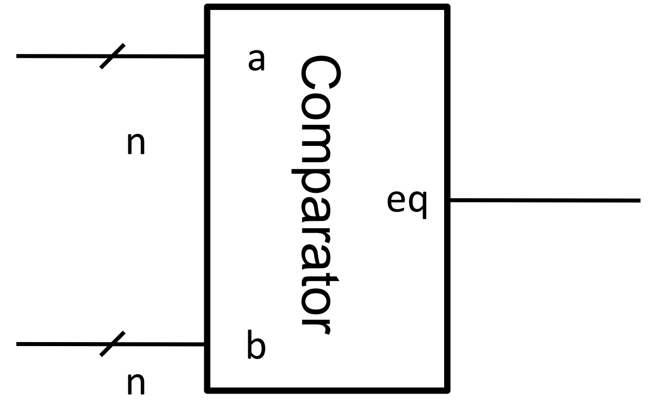
- If more than one input value is 1, then the encoder just designed does not work.
- One encoder that can accept all possible combinations of input values and produce a meaningful result is a *priority encoder*.
- Among the 1s that appear, it selects the most significant input position (highest priority) containing a 1 and responds with the corresponding binary code for that position.

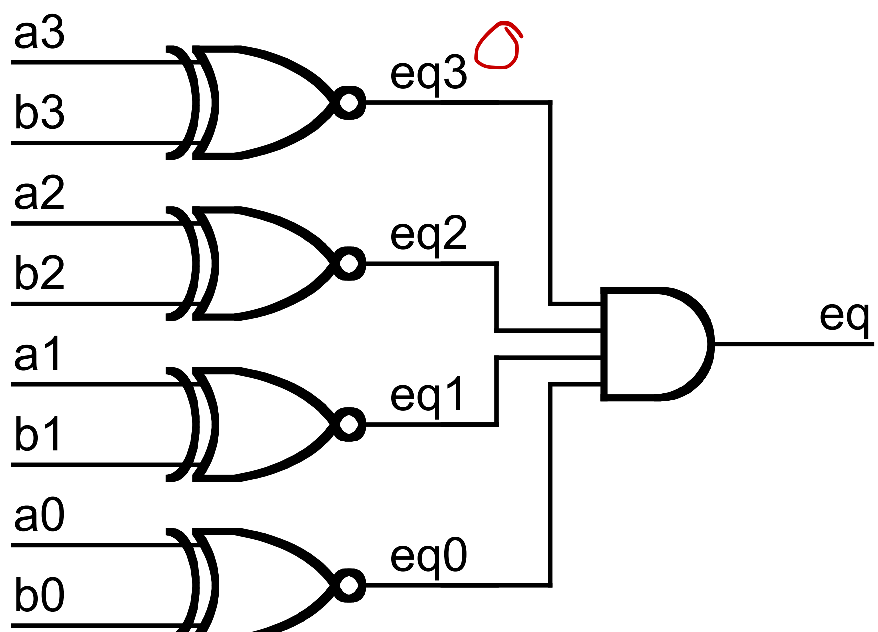
Priority Encoder (2/2)



Comparators

Equality Comparator





Magnitude Comparator (1/2)

- Compare two numbers A and B
  - Three possible results (A > B, A = B, A < B)
- Design approach for n-bit numbers
  - By truth table (need 2^{2n} rows, not practical)
  - By algorithm to build a regular circuit
    - $A = A_3A_2A_1A_0$, $B = B_3B_2B_1B_0$
    - $A = B$ if $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, $A_0 = B_0$
      - Equality $x_i = A_iB_i + A_i'B_i'$, $(A = B) = x_3x_2x_1x_0$
    - $(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$
    - $(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$

outputs

	A>B	A=B	A<B
when A>B	1	0	0
when A=B	0	1	0
when A<B	0	0	1

$A = A_3A_2A_1A_0$, $B = B_3B_2B_1B_0$

$A > B$

① when $A_3 > B_3$, $A_3 = 1$, $B_3 = 0$

② when $A_3 = B_3$ ($\overline{A_3 \oplus B_3} = 1$) , $A_2 = 1$, $B_2 = 0$

③ when $A_3 = B_3$, $A_2 = B_2$, $A_1 = 1$, $B_1 = 0$

④ when $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$, $A_0 = 1$, $B_0 = 0$

Magnitude Comparator (2/2)

A_3, B_3, x_3
A_2, B_2, x_2
A_1, B_1, x_1
A_0, B_0, x_0
$(A < B)$
$(A > B)$
$(A = B)$

Shifters

Shifters

- Shifter: shifts one bit to the left or right at a time.
- Logical shifter: shift the number to the left or right and fills empty spots with 0's.
  - Example: 1101

    LSR 1 = 0110
- Arithmetic shifter: same as logical shifter but on right shift fills empty MSBs with the sign bit (sign extension).
  - Example: 1101

    LSR 1 = 1110
- Barrel shifter: rotate numbers in a circle such that empty spots are filled with the bits shifted off the other end.
  - Example: 1101

    LSR 1 = 1110