Combinational Circuit:
MSI (Medium Scale Integrated)

CSIM601251

Instructor: Tim Dosen DDAK
Slide By : Erdefi Rakun
Fasilkom UI

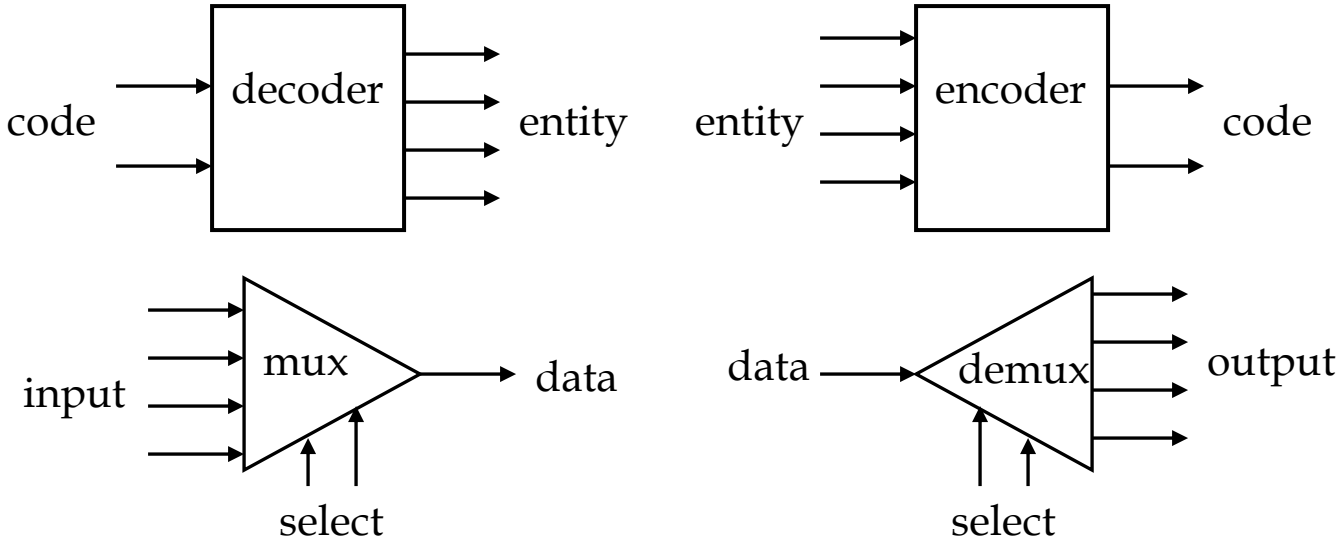
Outline

- Introduction
- Decoders
- Encoders
- Demultiplexers
- Multiplexers

Note: These slides are taken from Aaron Tan's slide

Introduction

- Four common and useful MSI circuits:
  - Decoder
  - Demultiplexer
  - Encoder
  - Multiplexer
- Block-level outlines of MSI circuits:



Outline

- Introduction
- Decoders
- Encoders
- Demultiplexers
- Multiplexers

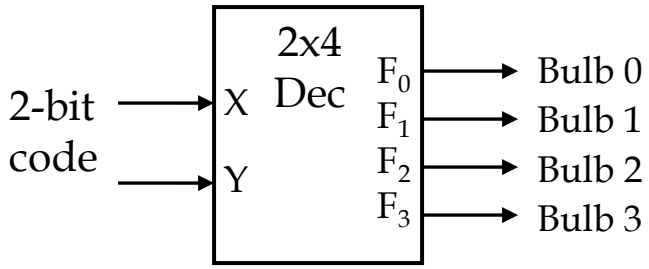
Note: These slides are taken from Aaron Tan's slide

Decoders (1/5)

- Codes are frequently used to represent entities, eg: your name is a code to denote yourself (an entity!).
- These codes can be identified (or decoded) using a decoder. Given a code, identify the entity.
- Convert binary information from n input lines to (maximum of) 2^n output lines.
- Known as n-to-m-line decoder, or simply n:m or $n \times m$ decoder ($m \leq 2^n$).
- May be used to generate 2^n minterms of n input variables.

Decoders (2/5)

- Example: If codes 00, 01, 10, 11 are used to identify four light bulbs, we may use a 2-bit decoder.



- This is a 2×4 decoder which selects an output line based on the 2-bit code supplied.
- Truth table:

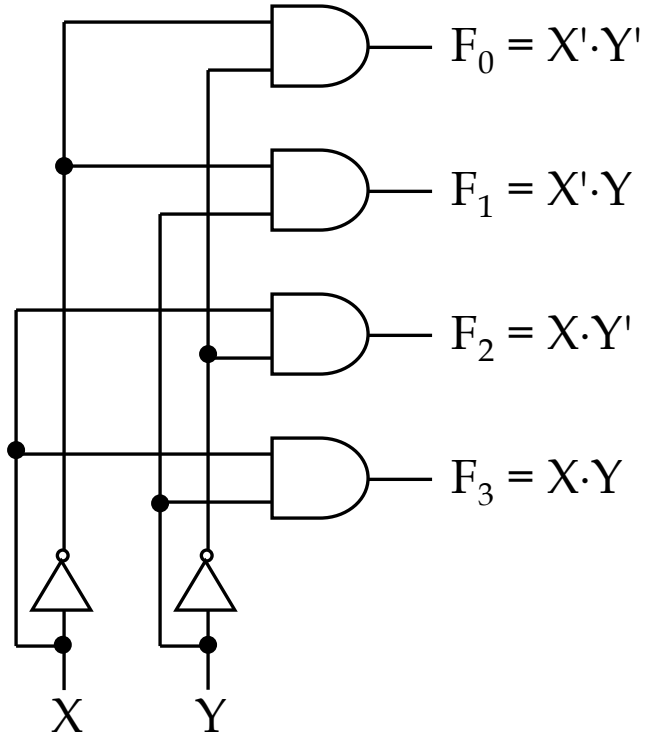
X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Decoders (3/5)

X	Y	F_0	F_1	F_2	F_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

- From truth table, circuit for 2×4 decoder is:

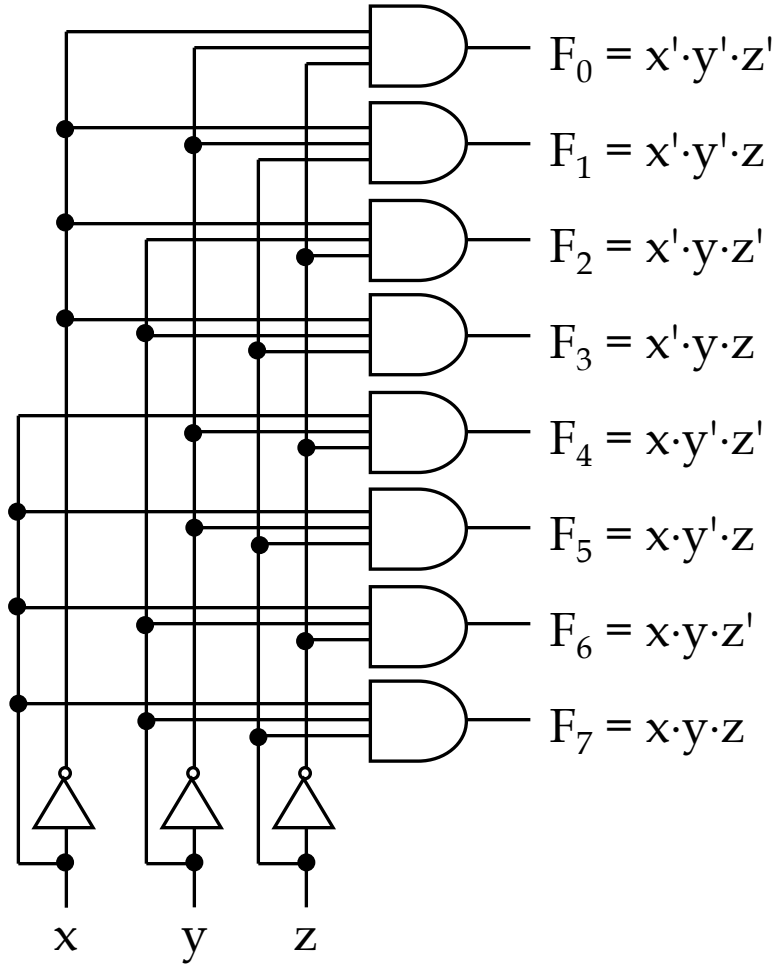
- Note: Each output is a 2-variable minterm (X'·Y', X'·Y, X·Y' or X·Y)



Decoders (4/5)

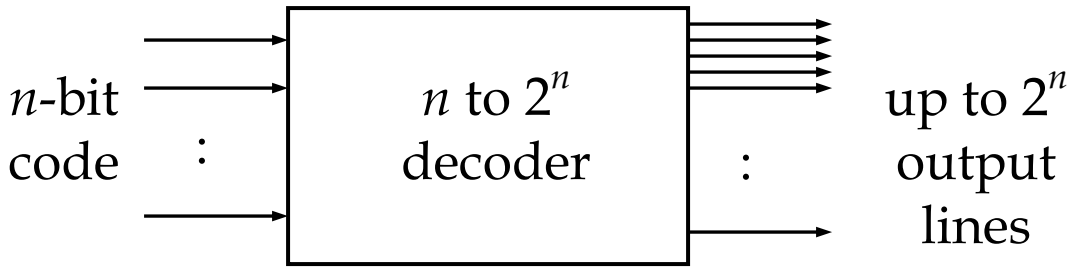
- Design a 3×8 decoder.

x	y	z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Decoders (5/5)

- In general, for an n-bit code, a decoder could select up to 2^n lines:



Decoders: Implementing Functions (1/5)

- A Boolean function, in sum-of-minterms form a decoder to generate the minterms, and an OR gate to form the sum.
- Any combinational circuit with n inputs and m outputs can be implemented with an n:2^n decoder with m OR gates.
- Good when circuit has many outputs, and each function is expressed with few minterms.

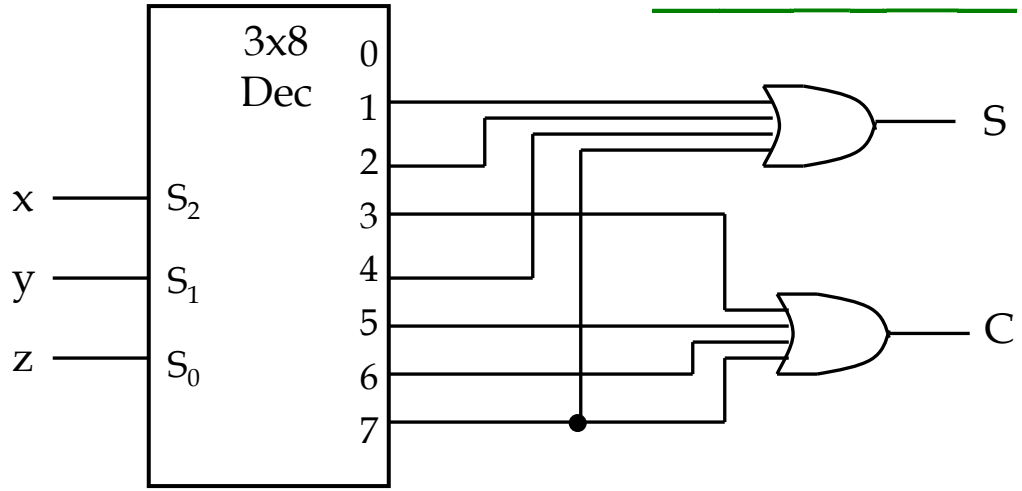
Decoders: Implementing Functions (2/5)

- Example: Full adder

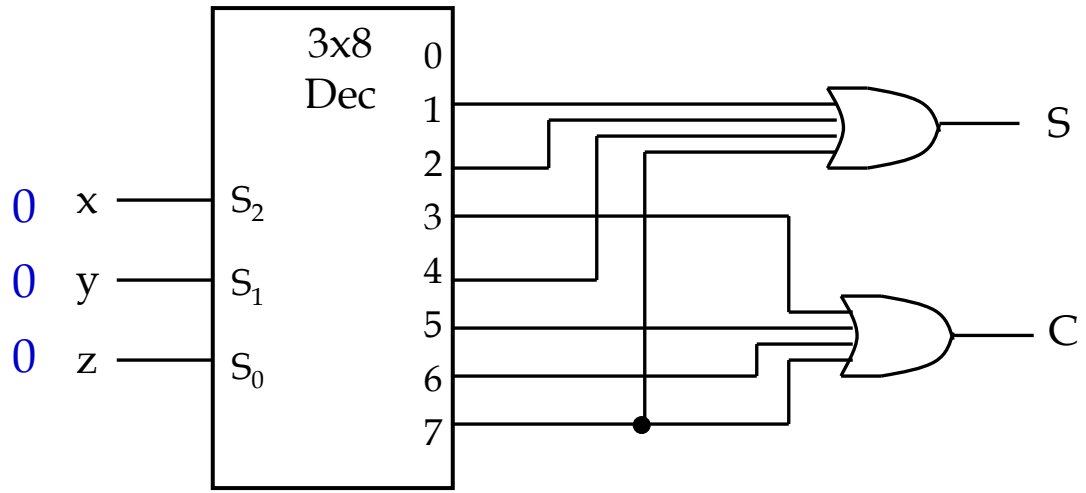
  $S(x, y, z) = \sum m(1,2,4,7)$

  $C(x, y, z) = \sum m(3,5,6,7)$

x	y	z	C	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

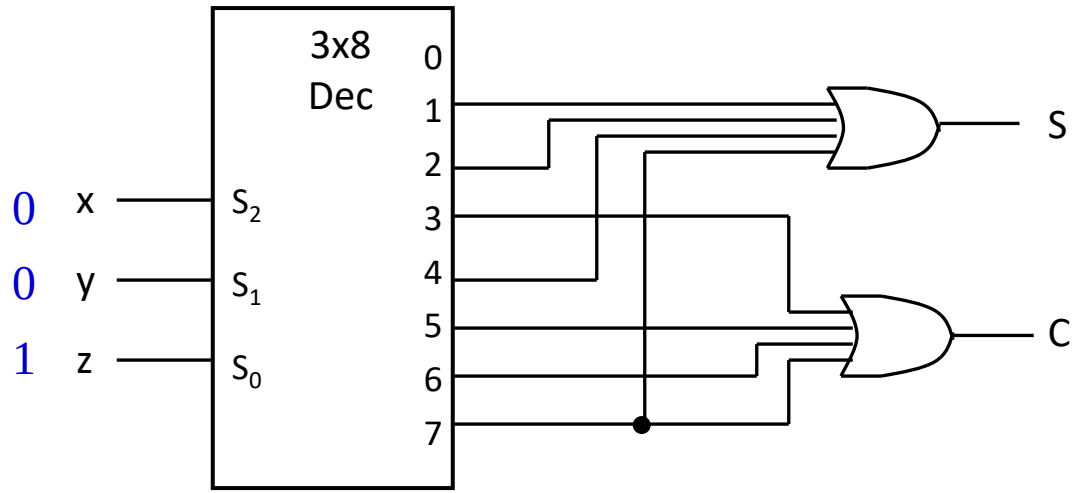


Decoders: Implementing Functions (3/5)



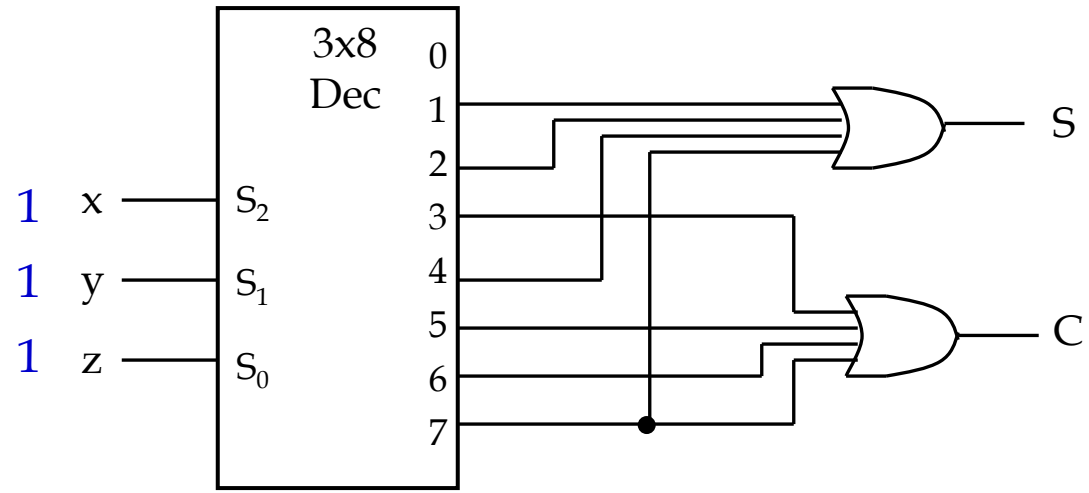
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Decoders: Implementing Functions (4/5)



x	y	z	C	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

Decoders: Implementing Functions (5/5)



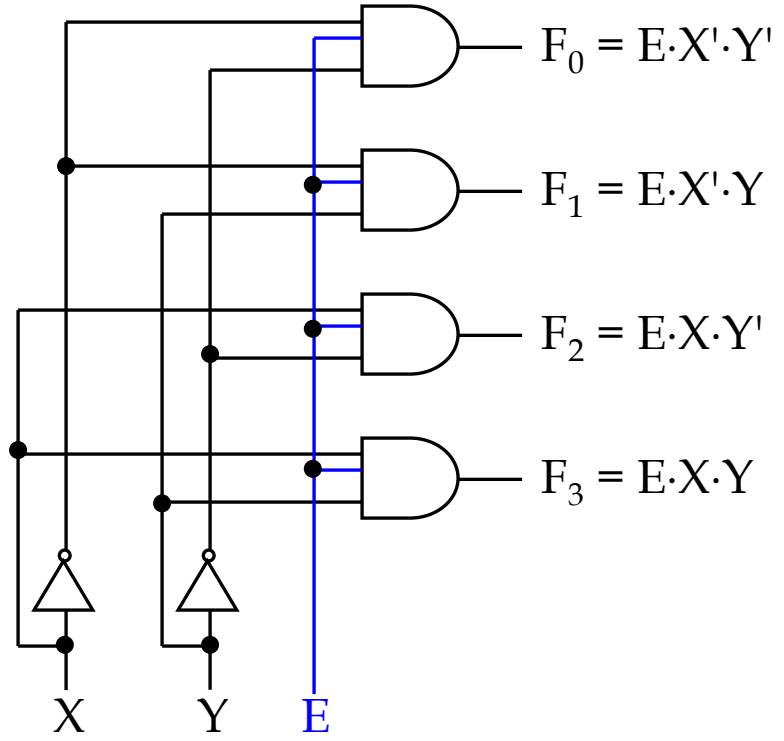
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Decoders with Enable (1/2)

- Decoders often come with an *enable* control signal, so that the device is only activated when the enable, E = 1.
- Truth table:

E	X	Y	F_0	F_1	F_2	F_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	X	X	0	0	0	0

- Circuit of a 2×4 decoder with enable:



Decoders with Enable (2/2)

- In the previous slide, the decoder has a one-enable control signal, i.e. the decoder is enabled with E=1.
- In most MSI decoders, enable signal is zero-enable, usually denoted by E' or $\bar{E}$. The decoder is enabled when the signal is zero (low).

E	X	Y	F_0	F_1	F_2	F_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	X	X	0	0	0	0

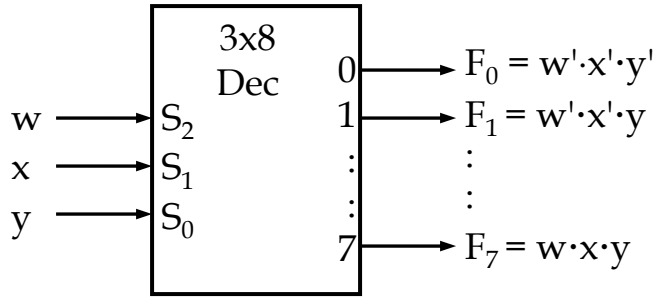
Decoder with 1-enable

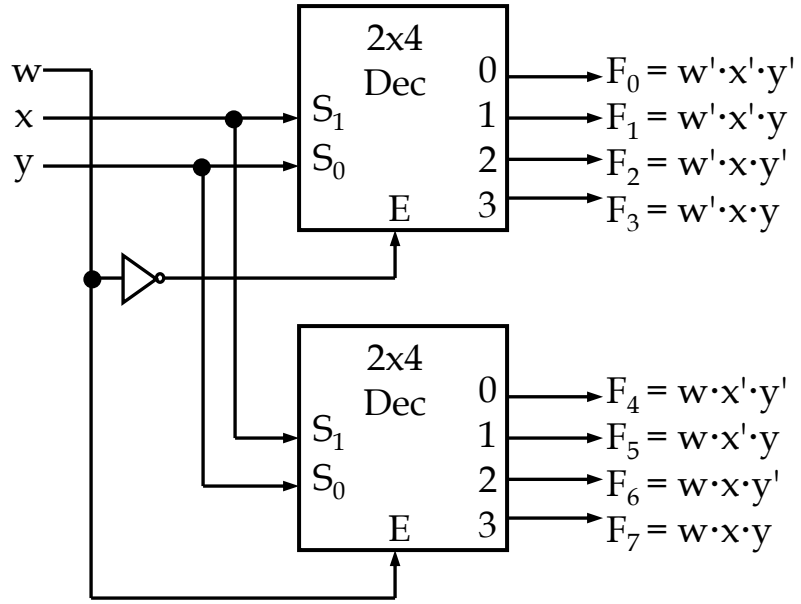
E'	X	Y	F_0	F_1	F_2	F_3
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1
1	X	X	0	0	0	0

Decoder with 0-enable

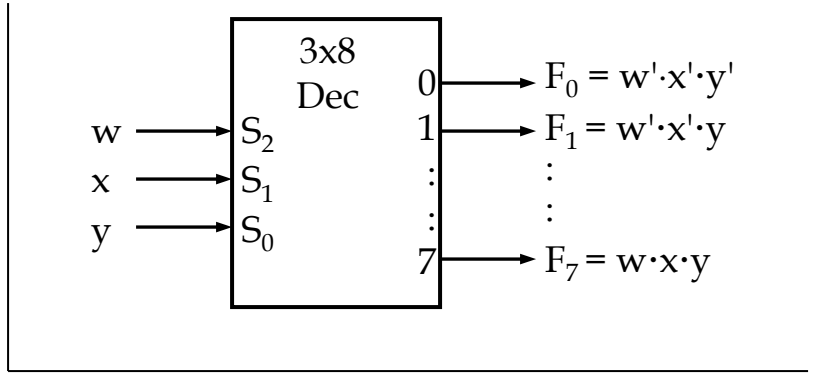
Larger Decoders (1/3)

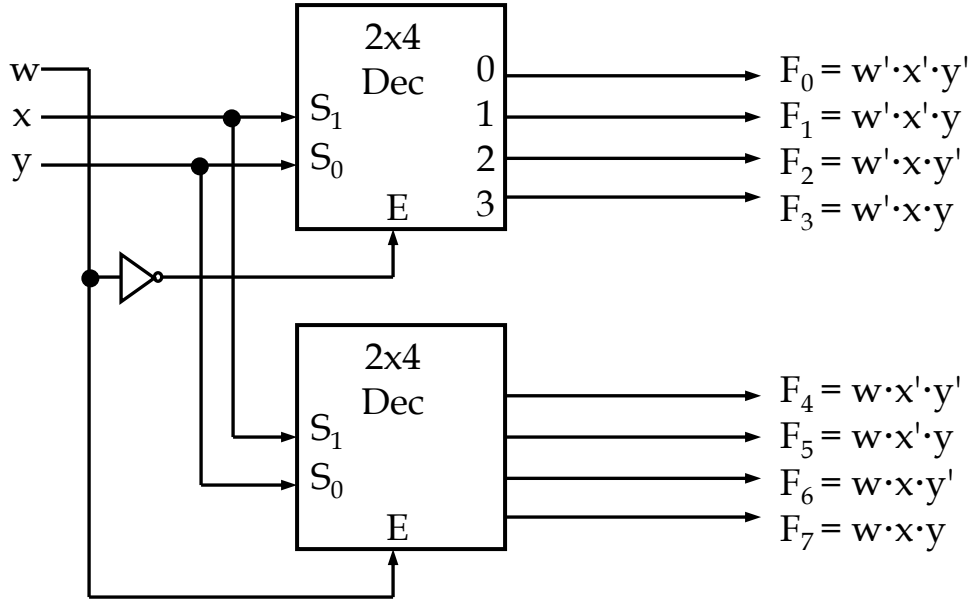
- Larger decoders can be constructed from smaller ones.
- Example: A 3×8 decoder can be built from two 2×4 decoders (with one-enable) and an inverter.





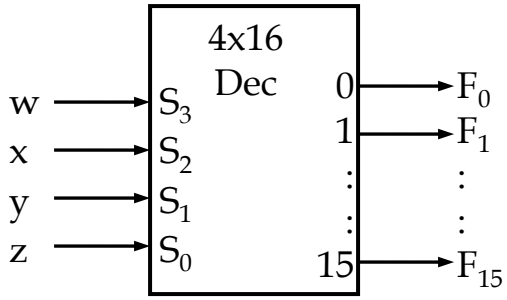
Larger Decoders (2/3)

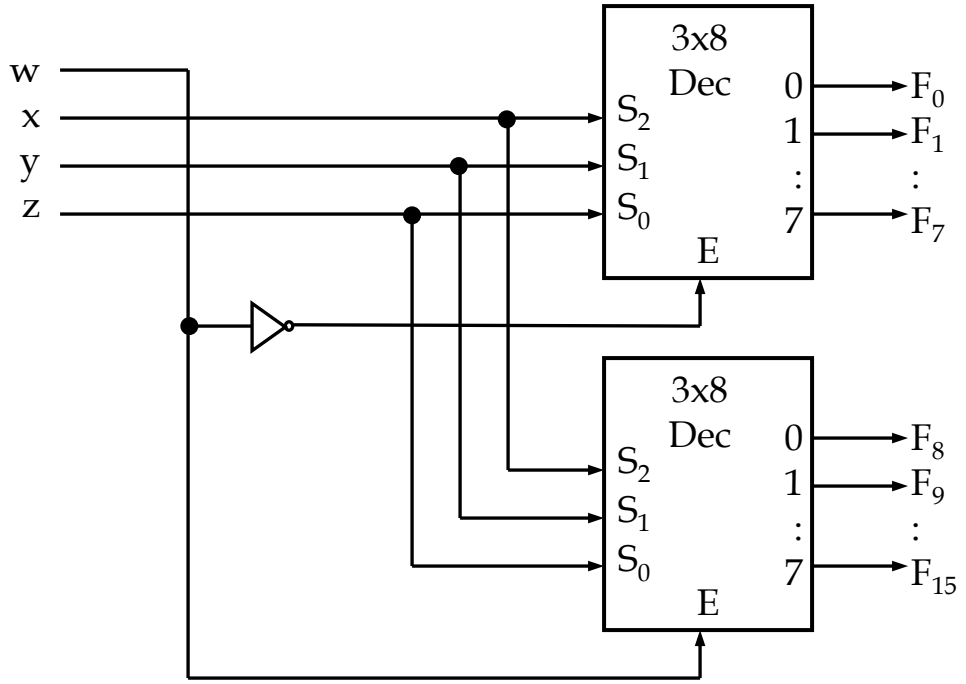




Larger Decoders (3/3)

- Construct a 4×16 decoder from two 3×8 decoders with one-enable.





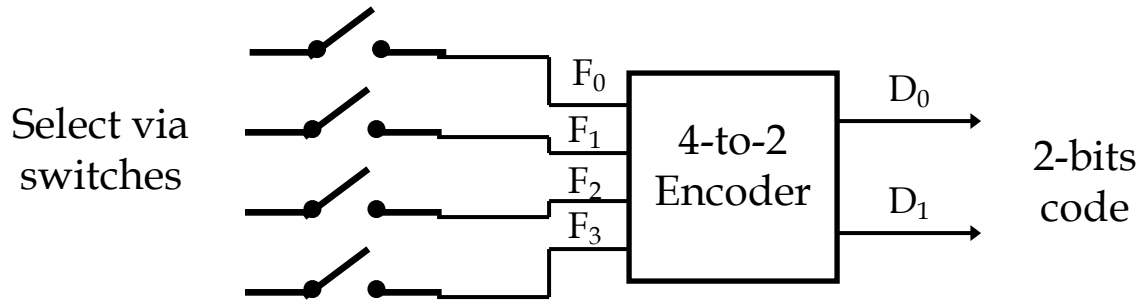
Outline

- Introduction
- Decoders
- Encoders
- Demultiplexers
- Multiplexers

Note: These slides are taken from Aaron Tan's slide

Encoders (1/4)

- Encoding is the converse of decoding.
- Given a set of input lines, of which exactly one is high, the encoder provides a code that corresponds to that input line.
- Contains 2^n (or fewer) input lines and n output lines.
- Implemented with OR gates.
- Example:



Encoders (2/4)

- Truth table:

- With K-map, we obtain:

  $D_0 = F1 + F3$
  $D_1 = F2 + F3$

- Circuit:

F_0	F_1	F_2	F_3	D_1	D_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1
0	0	0	0	X	X
0	0	1	1	X	X
0	1	0	1	X	X
0	1	1	0	X	X
0	1	1	1	X	X
1	0	0	1	X	X
1	0	1	0	X	X
1	0	1	1	X	X
1	1	0	0	X	X
1	1	0	1	X	X
1	1	1	0	X	X
1	1	1	1	X	X

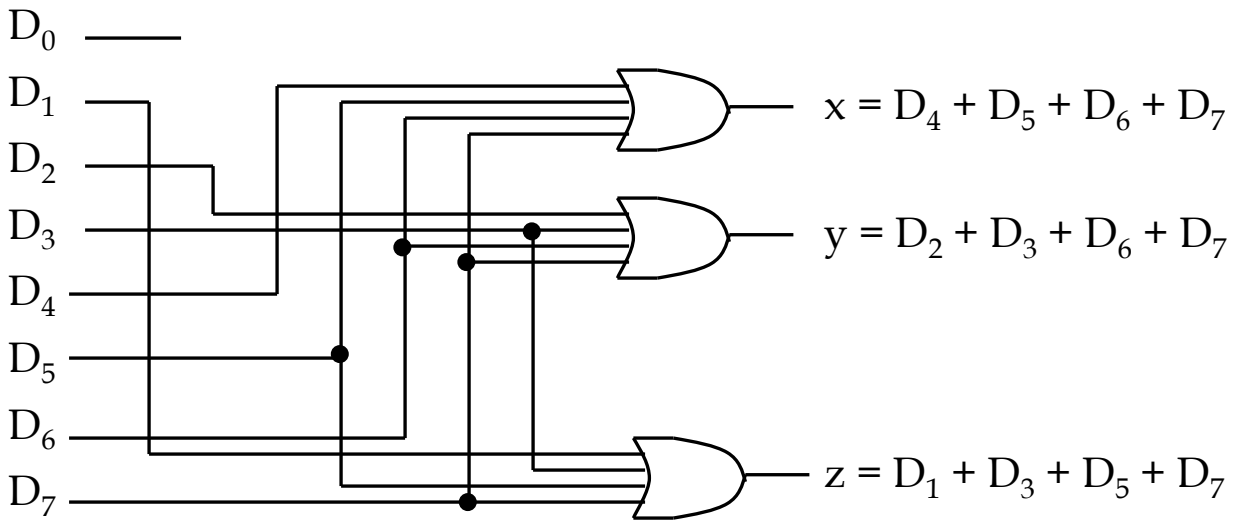
Encoders (3/4)

- Example: Octal-to-binary encoder.
  - At any one time, only one input line has a value of 1.
  - Otherwise, we need priority encoder.

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Encoders (4/4)

- Example: Octal-to-binary encoder.



An 8-to-3 encoder

- Exercise: Can you design a 2^n-to-n encoder without using K-map?

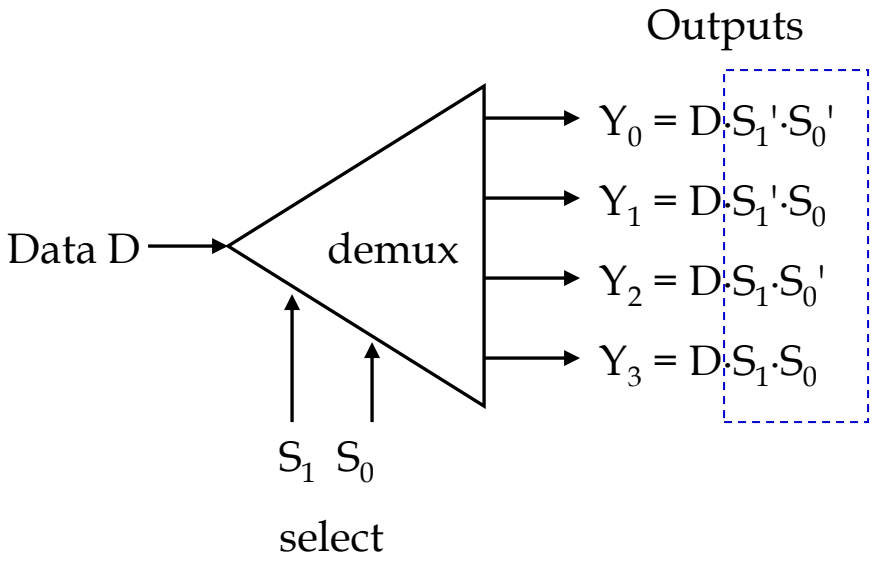
Outline

- Introduction
- Decoders
- Encoders
- Demultiplexers
- Multiplexers

Note: These slides are taken from Aaron Tan's slide

Demultiplexers (1/2)

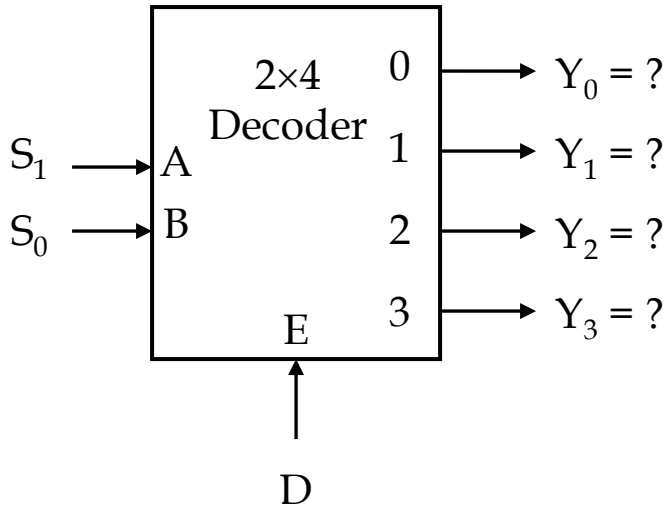
- Given an input line and a set of selection lines, a demultiplexer directs data from the input to one selected output line.
- Example: 1-to-4 demultiplexer.



S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	D	0	0	0
0	1	0	D	0	0
1	0	0	0	D	0
1	1	0	0	0	D

Demultiplexers (2/2)

- It turns out that the demultiplexer circuit is actually identical to a decoder with enable.



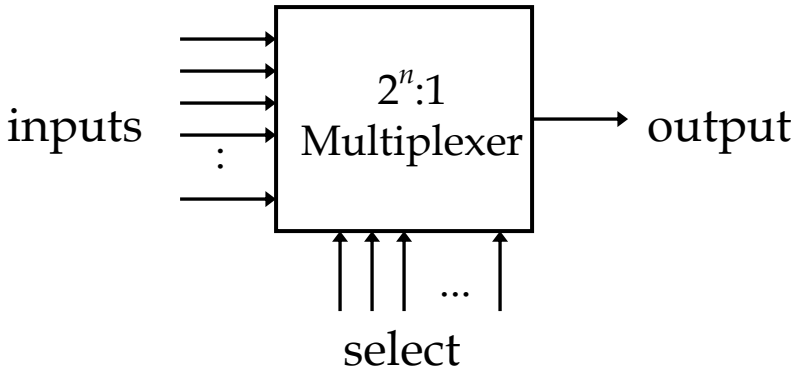
Outline

- Introduction
- Decoders
- Encoders
- Demultiplexers
- Multiplexers

Note: These slides are taken from Aaron Tan's slide

Multiplexers (1/5)

- A multiplexer is a device which has
  - A number of input lines
  - A number of selection lines
  - One output line
- It steers one of 2^n inputs to a single output line, using n selection lines. Also known as a *data selector*.

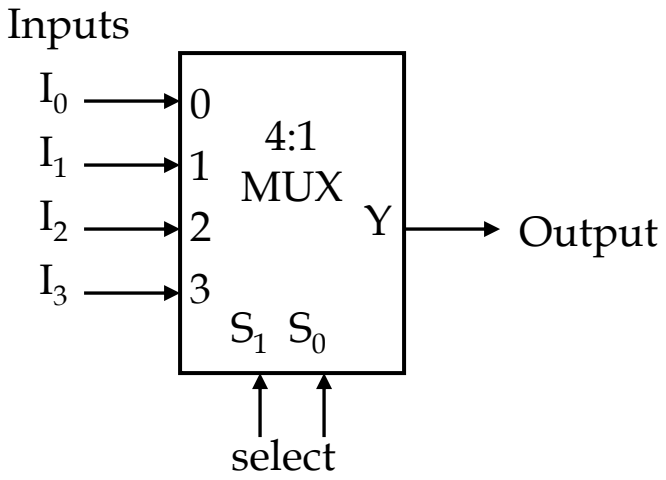


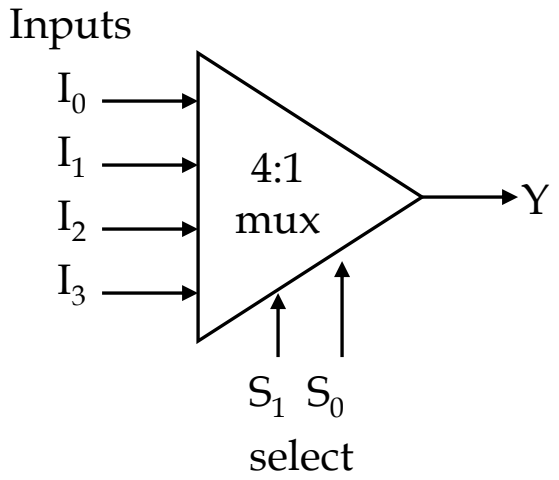
Multiplexers (2/5)

- Truth table for a 4-to-1 multiplexer:

I_0	I_1	I_2	I_3	S_1	S_0	Y
d_0	d_1	d_2	d_3	0	0	d_0
d_0	d_1	d_2	d_3	0	1	d_1
d_0	d_1	d_2	d_3	1	0	d_2
d_0	d_1	d_2	d_3	1	1	d_3

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3





Multiplexers (3/5)

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- Output of multiplexer is

  "sum of the (product of *data lines* and *selection lines*)"

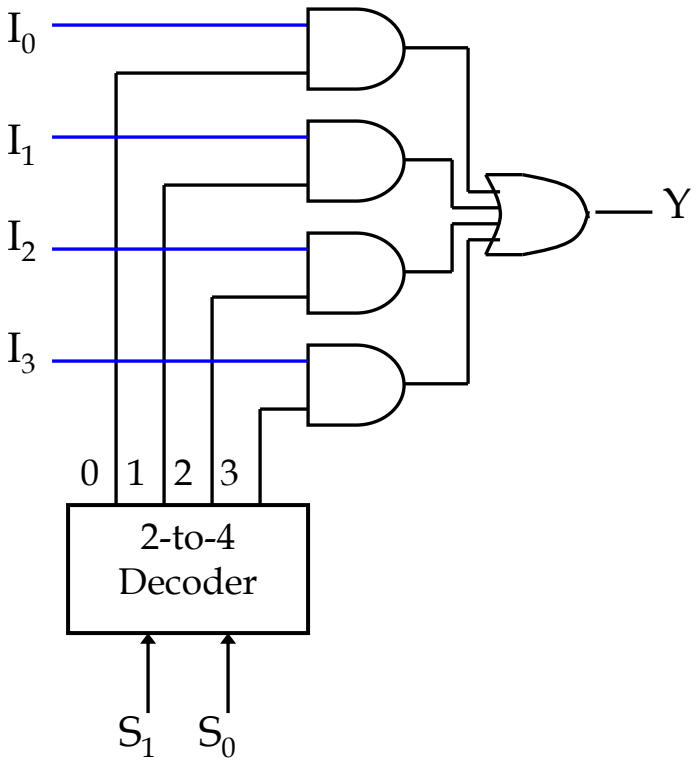
- Example: Output of a 4-to-1 multiplexer is:

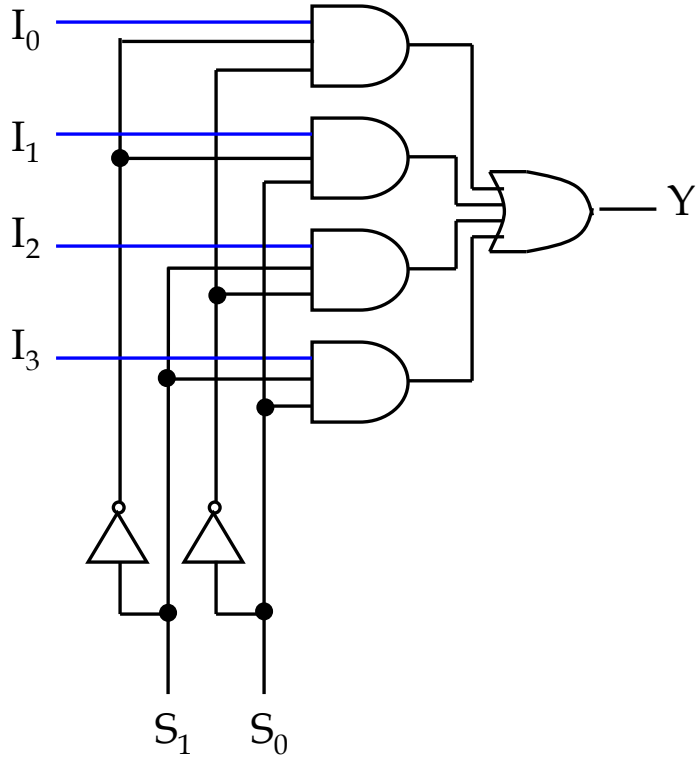
  $$Y = I_0 \bullet (S_1' \bullet S_0') + I_1 \bullet (S_1' \bullet S_0) + I_2 \bullet (S_1 \bullet S_0') + I_3 \bullet (S_1 \bullet S_0)$$

- A 2^n-to-1-line multiplexer, or simply 2^n:1 MUX, is made from an n:2^n decoder by adding to it 2^n input lines, one to each AND gate.

Multiplexers (4/5)

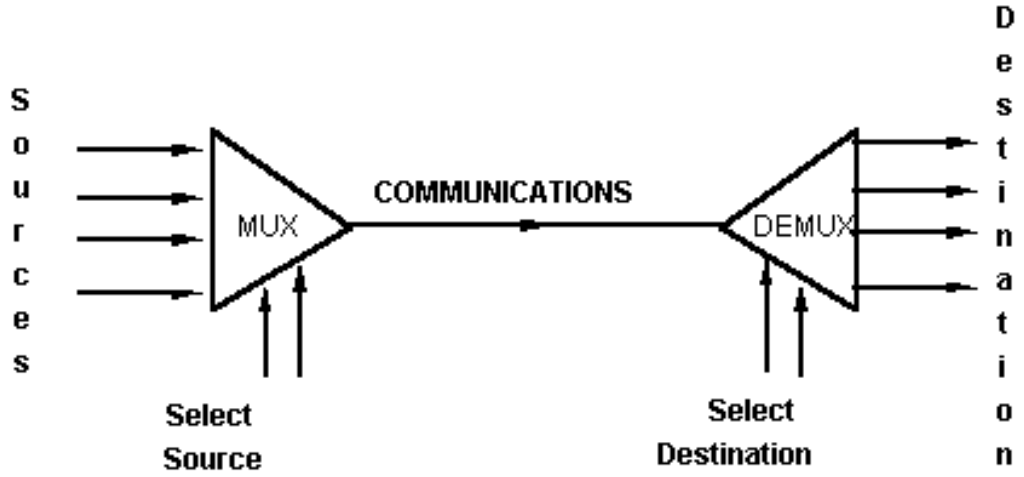
- A 4:1 multiplexer circuit:





Multiplexers (5/5)

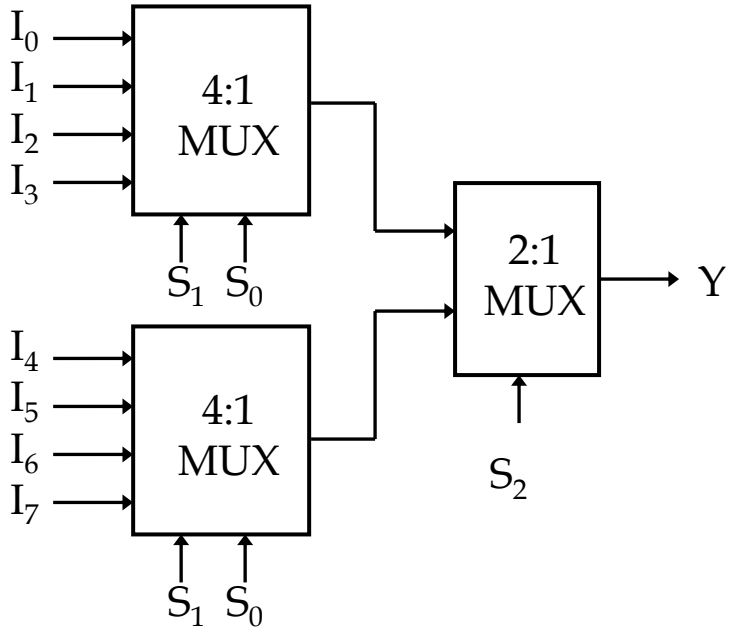
- An application:



- Helps share a *single communication line* among a number of devices.
- At any time, only one source and one destination can use the communication line.

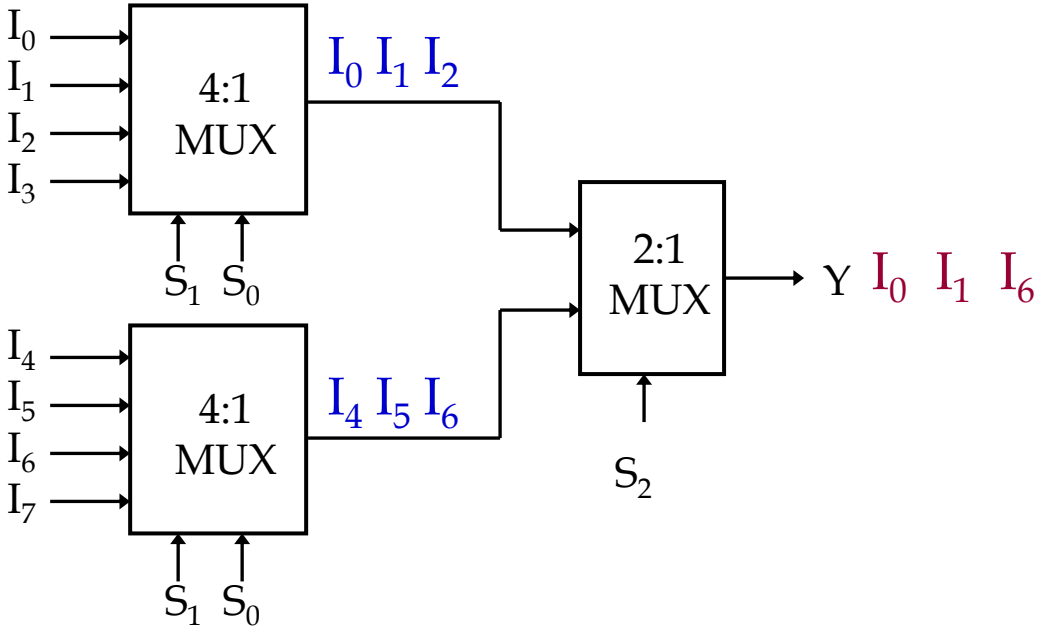
Larger Multiplexers (1/4)

- Larger multiplexers can be constructed from smaller ones.
- An 8-to-1 multiplexer can be constructed from smaller multiplexers like this (note placement of selector lines):



S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Larger Multiplexers (2/4)

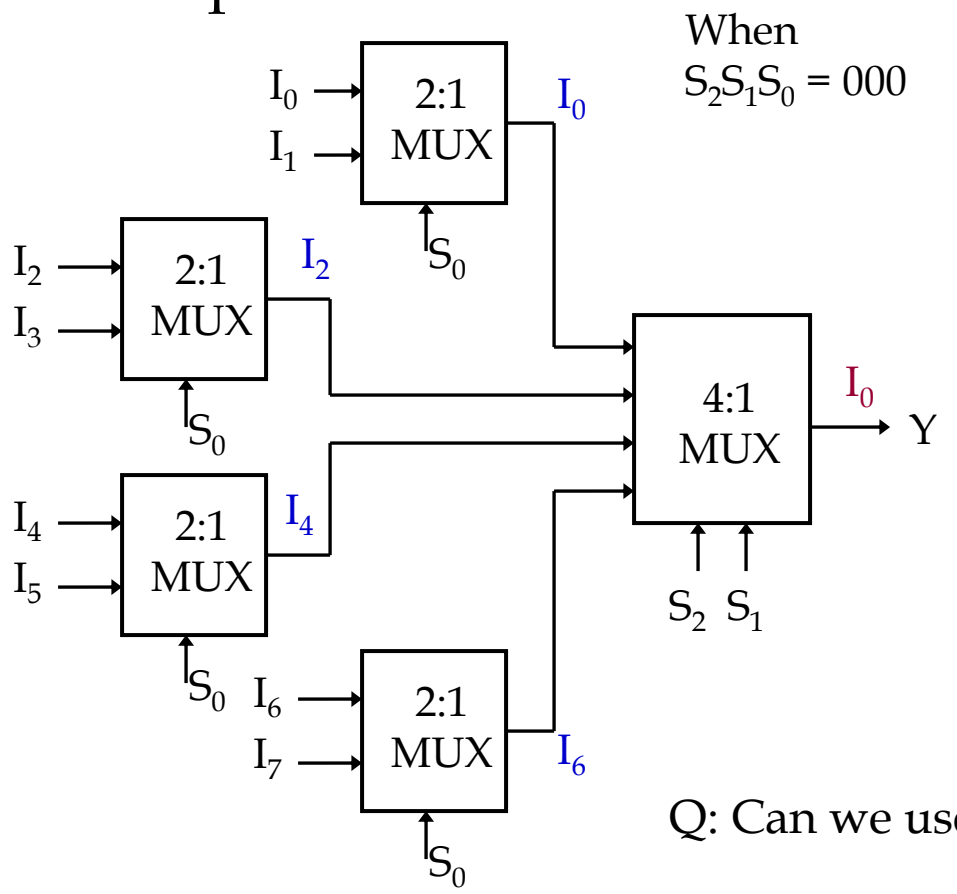


S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

- When $S_2S_1S_0 = 000$
- When $S_2S_1S_0 = 001$
- When $S_2S_1S_0 = 110$

Larger Multiplexers (3/4)

- Another implementation of an 8-to-1 multiplexer using smaller multiplexers:

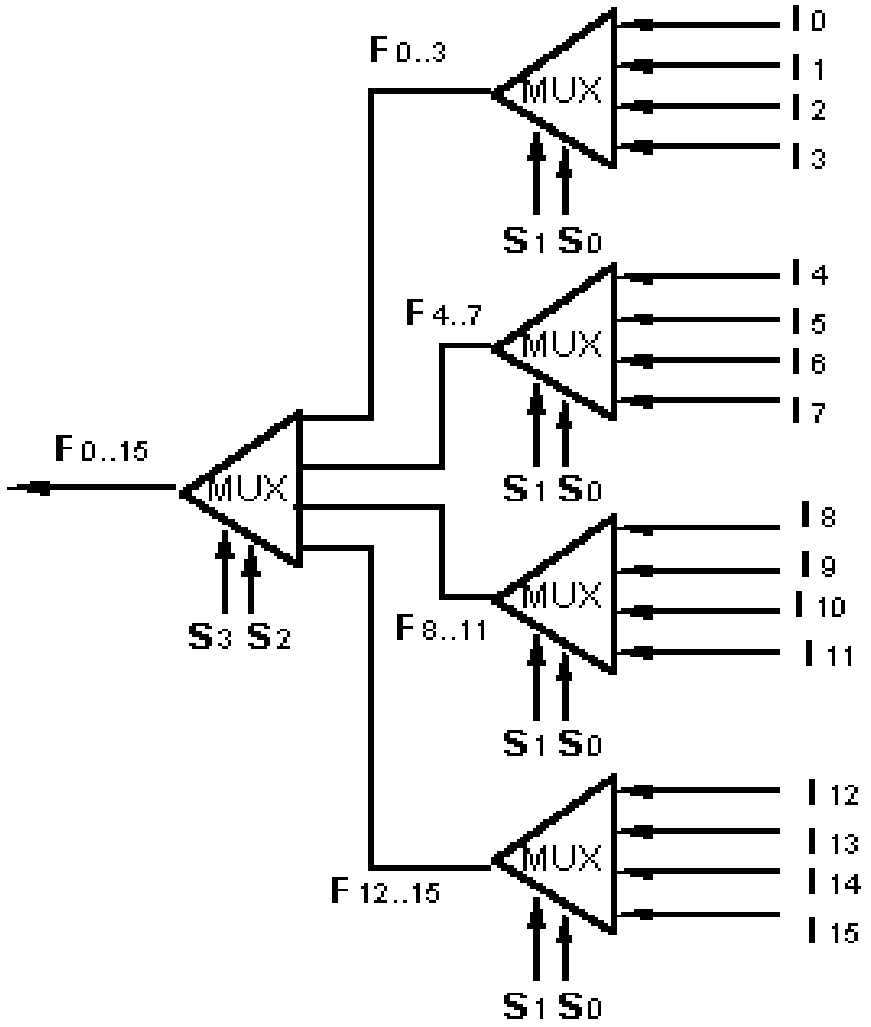


S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Q: Can we use only 2:1 multiplexers?

Larger Multiplexers (4/4)

- A 16-to-1 multiplexer can be constructed from five 4-to-1 multiplexers:

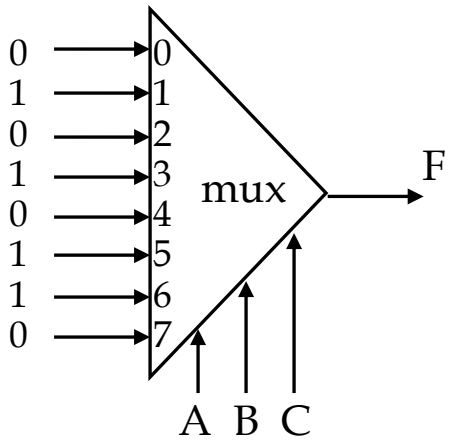


Multiplexers: Implementing Functions (1/3)

- Boolean functions can be implemented using multiplexers.
- A 2^n-to-1 multiplexer can implement a Boolean function of n input variables, as follows:
  1. Express in sum-of-minterms form. Example:
     $F(A,B,C) = A'\cdot B'\cdot C + A'\cdot B\cdot C + A\cdot B'\cdot C + A\cdot B\cdot C'$
     $= \sum m(1,3,5,6)$
  2. Connect n variables to the n selection lines.
  3. Put a '1' on a data line if it is a minterm of the function, or '0' otherwise.

Multiplexers: Implementing Functions (2/3)

- $F(A,B,C) = \sum m(1,3,5,6)$



This method works because:

$$\text{Output} = m_0 \cdot I_0 + m_1 \cdot I_1 + m_2 \cdot I_2 + m_3 \cdot I_3 + m_4 \cdot I_4 + m_5 \cdot I_5 + m_6 \cdot I_6 + m_7 \cdot I_7$$

Supplying '1' to I_1, I_3, I_5, I_6 , and '0' to the rest:

$$\text{Output} = m_1 + m_3 + m_5 + m_6$$

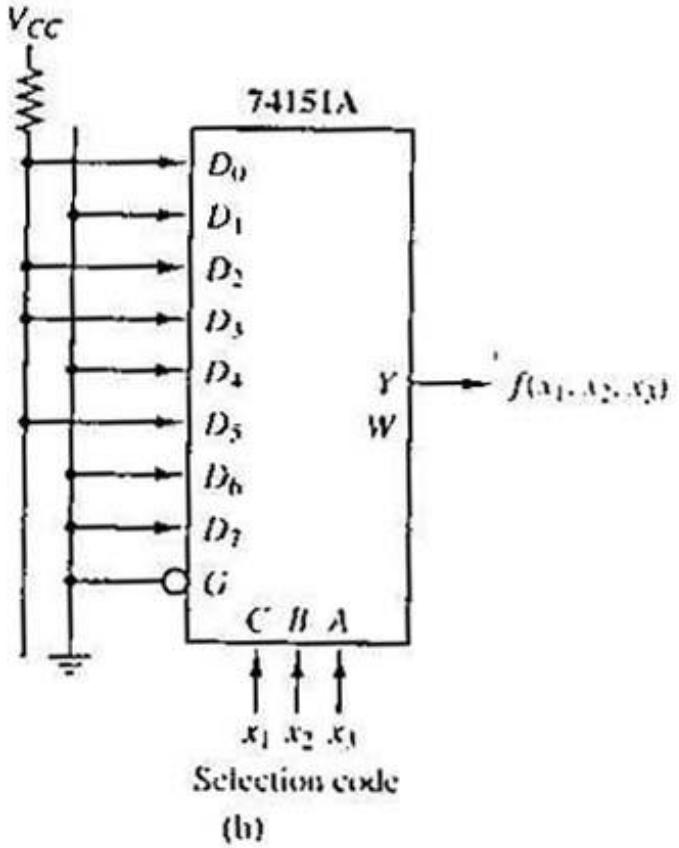
Multiplexers: Implementing Functions (3/3)

- Example: Use a 74151A to implement

$$f(x_1,x_2,x_3) = \sum m(0,2,3,5)$$

i	C	B	A	Y	
	x_1	x_2	x_3	f	
0	0	0	0	1	$D_0 = 1$
1	0	0	1	0	$D_1 = 0$
2	0	1	0	1	$D_2 = 1$
3	0	1	1	1	$D_3 = 1$
4	1	0	0	0	$D_4 = 0$
5	1	0	1	1	$D_5 = 1$
6	1	1	0	0	$D_6 = 0$
7	1	1	1	0	$D_7 = 0$

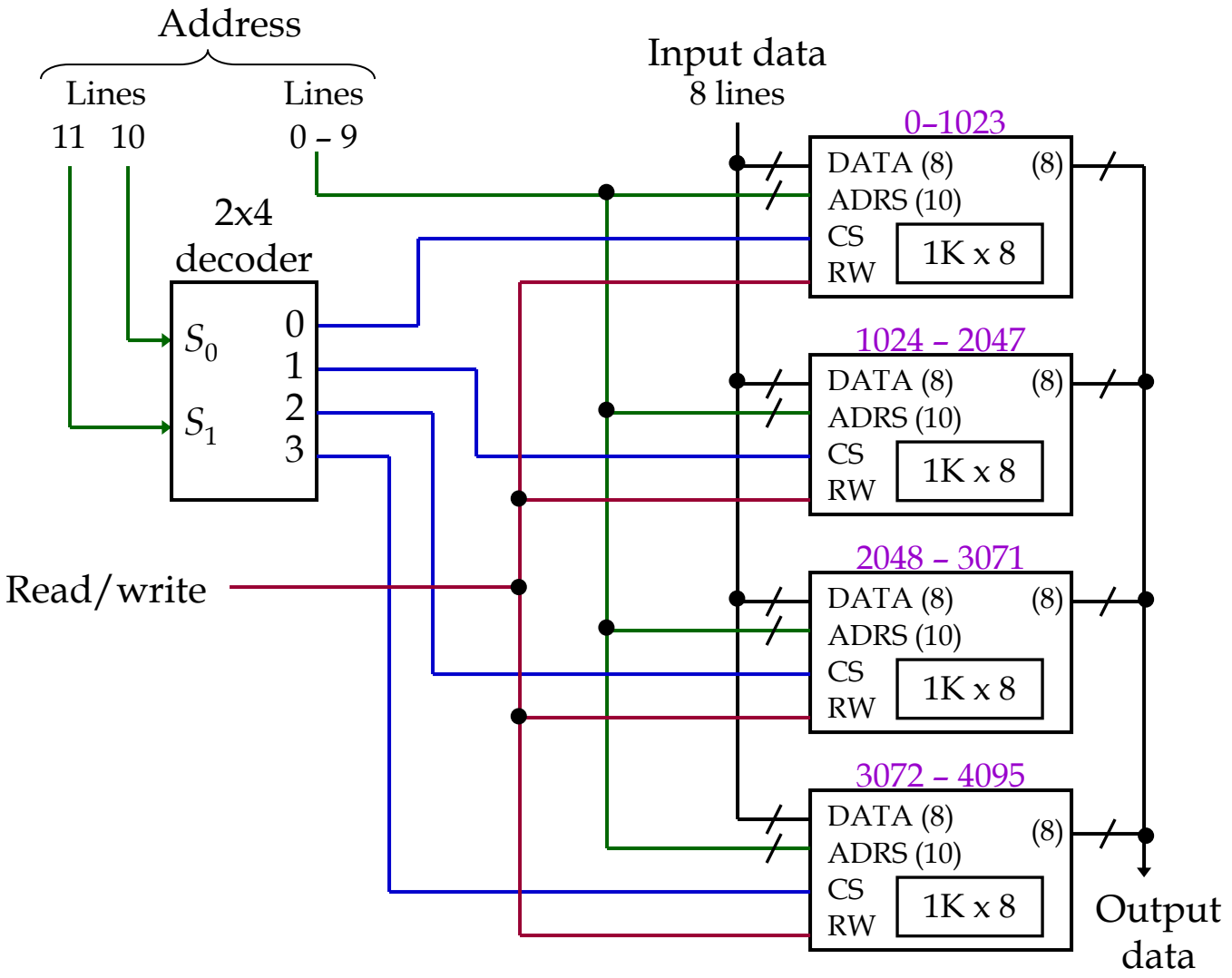
(a)



(b)

Realization of $f(x_1,x_2,x_3) = \sum m(0,2,3,5)$.
(a) Truth table.
(b) Implementation with 74151A.

Peeking Ahead (1/2)



Peeking Ahead (2/2)