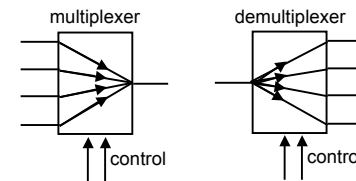


Overview

- ◆ Last lecture
 - Timing diagrams
 - Multilevel logic
 - ↳ Multilevel NAND/NOR conversion
 - ↳ AOI and OAI gates
 - Hazards
- ◆ Today
 - "Switching-network" logic blocks
 - ↳ Multiplexers/selectors
 - ↳ Demultiplexers/decoders
 - Programmable logic devices (PLDs)
 - ↳ Regular structures for 2-level logic

Switching-network logic blocks

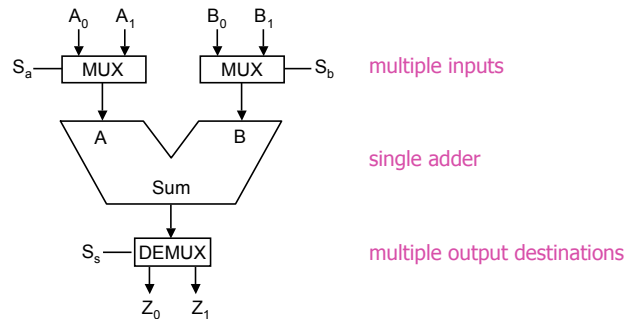
- ◆ Multiplexer
 - Routes one of many inputs to a single output
 - Also called a *selector*
- ◆ Demultiplexer
 - Routes a single input to one of many outputs
 - Also called a *decoder*



We construct these devices from:
 (1) logic gates
 (2) networks of transistor switches

Rationale: Sharing complex logic functions

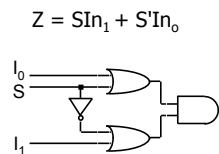
- ◆ Share an adder: Select inputs; route sum



Multiplexers

- ◆ Basic concept
 - 2^n data inputs; n control inputs ("selects"); 1 output
 - Connects one of 2^n inputs to the output
 - "Selects" decide which input connects to output
 - Two alternative truth-tables: **Functional** and **Logical**

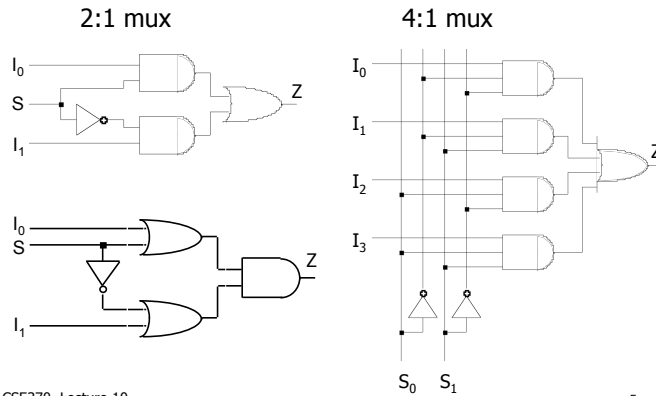
Example: A 2:1 Mux **Functional** truth table **Logical** truth table



S	Z
0	I_{n_0}
1	I_{n_1}

I_{n_1}	I_{n_0}	S	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Logic-gate implementation of multiplexers

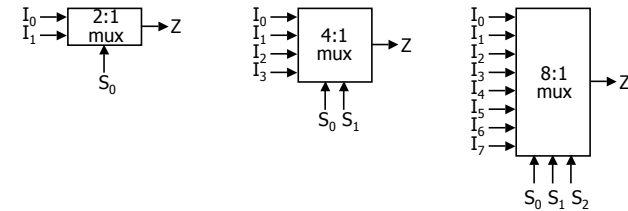


CSE370, Lecture 10

5

Multiplexers (con't)

- ◆ 2:1 mux: $Z = S'I_0 + SI_1$
- ◆ 4:1 mux: $Z = S_0'S_1'I_0 + S_0'S_1I_1 + S_0S_1'I_2 + S_0S_1I_3$
- ◆ 8:1 mux: $Z = S_0'S_1'S_2'I_0 + S_0'S_1S_2I_1 + \dots$

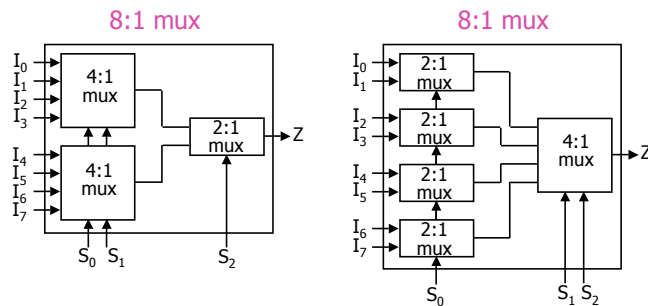


CSE370, Lecture 10

6

Cascading multiplexers

- ◆ Can form large multiplexers from smaller ones
 - Many implementation options



CSE370, Lecture 10

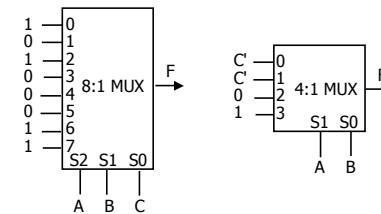
7

Multiplexers as general-purpose logic

- ◆ A $2^n:1$ mux can implement any function of n variables
 - A lookup table
 - A $2^{n-1}:1$ mux also can implement any function of n variables

- ◆ Example: $F(A,B,C) = m_0 + m_2 + m_6 + m_7$
 $= A'B'C' + A'BC' + ABC' + ABC$
 $= A'B'(C') + A'B(C') + AB(0) + AB(1)$

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

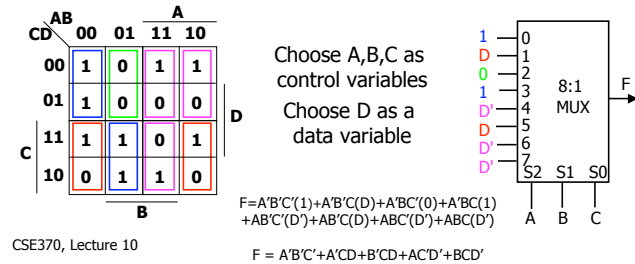


CSE370, Lecture 10

8

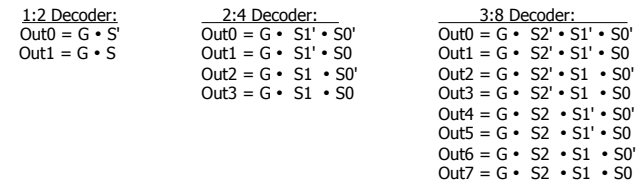
Multiplexers as general-purpose logic

- Implementing a $2^n:1$ mux as a function of $n-1$ variables
 - ($n-1$) mux control variables $S_0 - S_{n-1}$
 - One data variable S_n
 - Four possible values for each data input: 0, 1, S_n , S_n'
 - Example: $F(A,B,C,D)$ implemented using an 8:1 mux



Demultiplexers

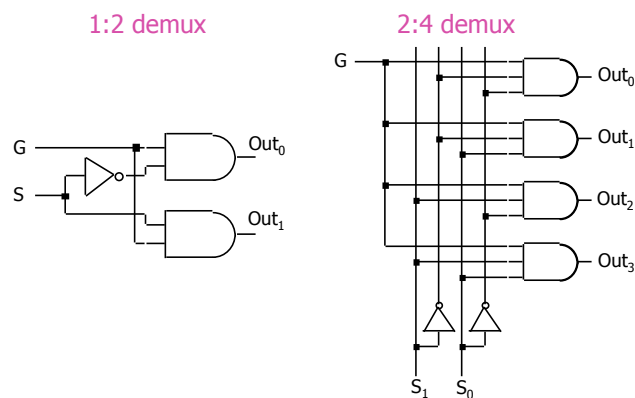
- Basic concept
 - Single data input; n control inputs ("selects"); 2^n outputs
 - Single input connects to one of 2^n outputs
 - "Selects" decide which output is connected to the input
 - When used as a decoder, the input is called an "enable" (G)



CSE370, Lecture 10

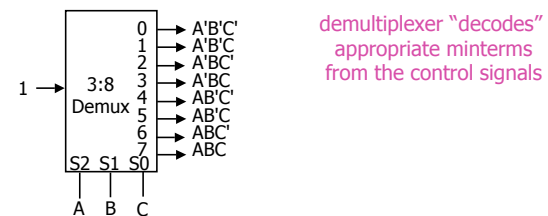
10

Logic-gate implementation of demultiplexers



Demultiplexers as general-purpose logic

- A $n:2^n$ demux can implement any function of n variables
 - Use variables as select inputs
 - Tie enable input to logic 1
 - Sum the appropriate minterms (extra OR gate)



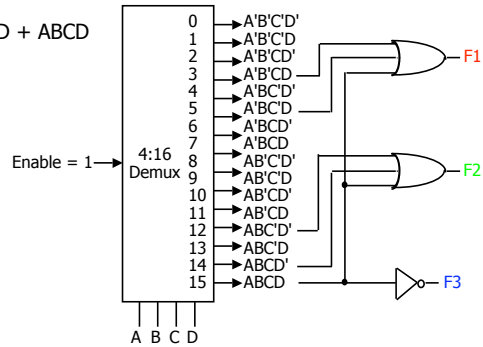
Demultiplexers as general-purpose logic

Example

$$F1 = A'BC'D + A'B'CD + ABCD$$

$$F2 = ABC'D + ABC$$

$$F3 = (A'+B'+C'+D')$$

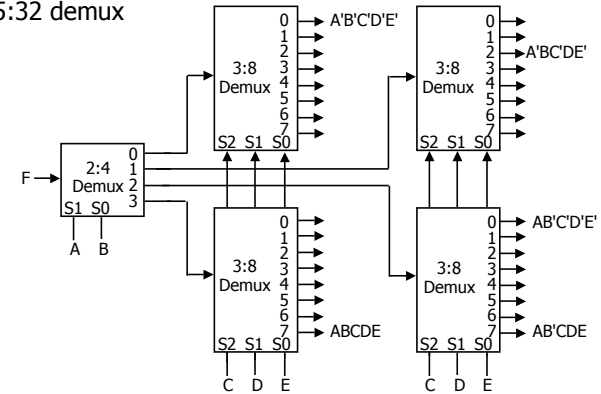


CSE370, Lecture 10

13

Cascading demultiplexers

◆ 5:32 demux

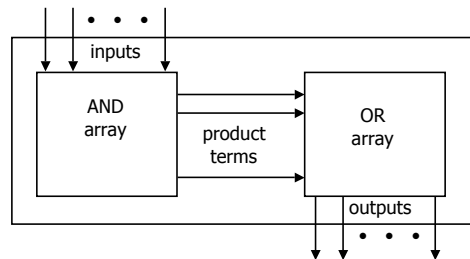


CSE370, Lecture 10

14

Programmable logic (PLAs & PALs)

- ◆ Concept: Large array of uncommitted AND/OR gates
 - Actually NAND/NOR gates
 - You program the array by making or breaking connections
 - ↳ Programmable block for sum-of-products logic

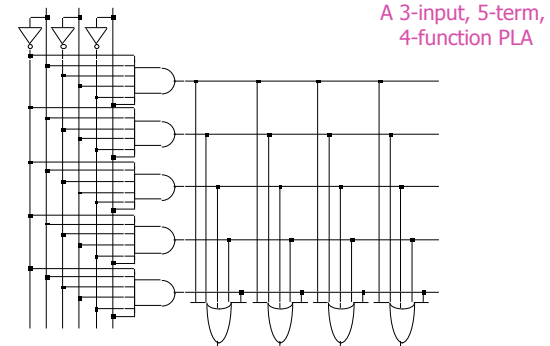


CSE370, Lecture 10

15

All two-level logic functions are available

- ◆ You "program" the wire connections



CSE370, Lecture 10

16

Sharing product terms

- ◆ Example: $F_0 = A + B'C'$
 $F_1 = AC' + AB$
 $F_2 = B'C' + AB$
 $F_3 = B'C + A$

inputs
 1 = asserted in term
 0 = negated in term
 - = does not participate

- ◆ Personality matrix:

outputs
 1 = term connected to output
 0 = no connection to output

product term	inputs			outputs			
	A	B	C	F0	F1	F2	F3
AB	1	1	-	0	1	1	0
B'C	-	0	1	0	0	0	1
AC'	1	-	0	0	1	0	0
B'C'	-	0	0	1	0	1	0
A	1	-	-	1	0	0	1

Reuse terms

Programming the wire connections

- Fuse: Comes connected; break unwanted connections
- Anti-fuse: Comes disconnected; make wanted connections

$F_0 = A + B'C'$
 $F_1 = AC' + AB$
 $F_2 = B'C' + AB$
 $F_3 = B'C + A$

