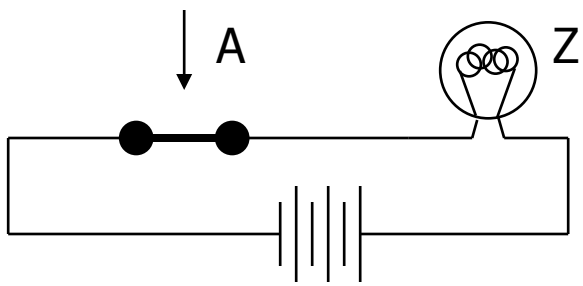
Combinational logic

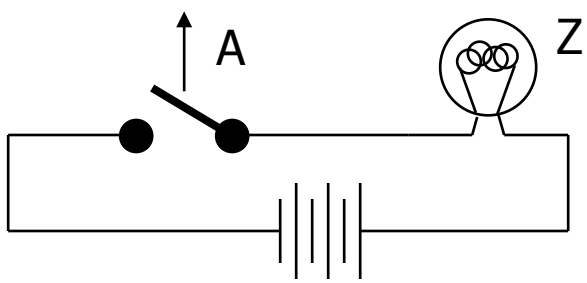
- Switches, basic logic and truth tables, logic functions
- Algebraic expressions to gates
- Mapping to different gates
- Discrete logic gate components (used in labs 1 and 2)
- Canonical forms
- Regular logic: multiplexers, decoders, LUTs and FPGAs

Switches: basic element of physical implementations

- Implementing a simple circuit:



close switch (if A is "1" or asserted) and turn on light bulb (Z)



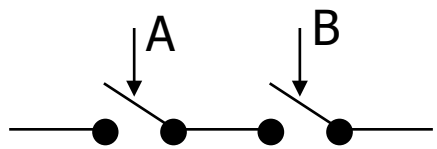
open switch (if A is "0" or unasserted) and turn off light bulb (Z)

$$Z \equiv A$$

Switches (cont'd)

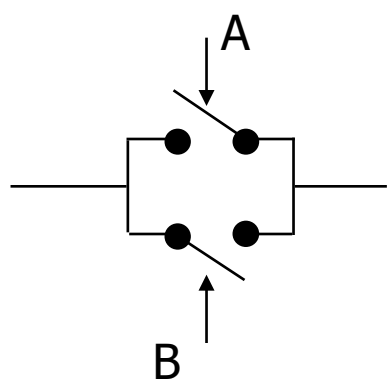
- Compose switches into more complex ones (Boolean functions):

AND



Z = A <u>and</u> B = A * B = AB

OR

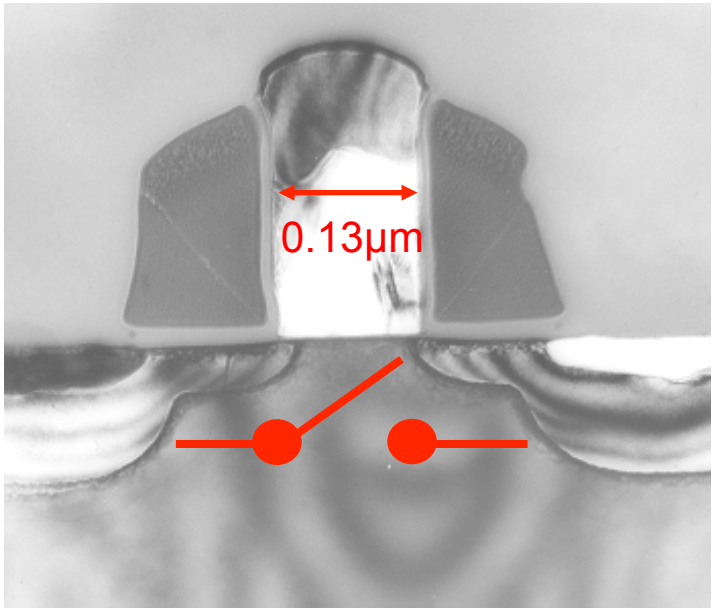


Z = A <u>or</u> B = A + B

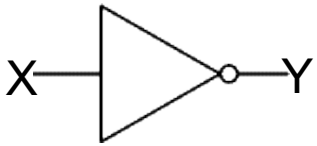
Transistor networks

- Modern digital systems are designed in CMOS technology
  - MOS stands for Metal-Oxide on Semiconductor
  - C is for complementary because there are both normally-open and
    normally-closed switches
- MOS transistors act as voltage-controlled switches
  - similar, though easier to work with than relays.

Most digital logic is CMOS

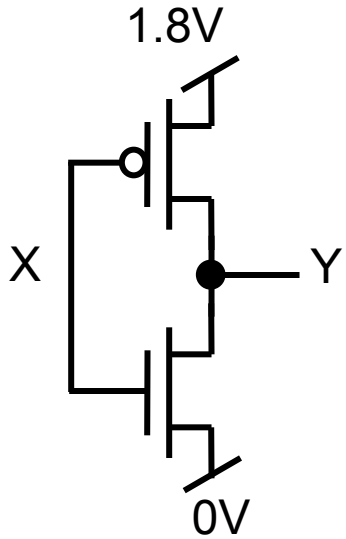


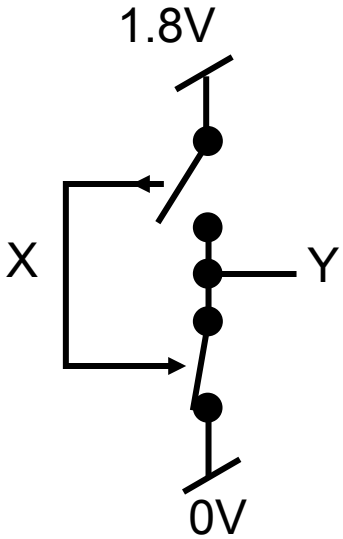
Mark Bohr Intel



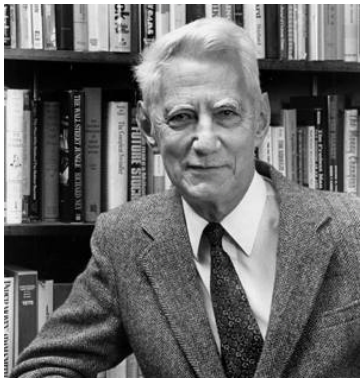
0V ≡ Logic 0
1.8V ≡ Logic 1

X	Y
0V	1.8V
1.8V	0V



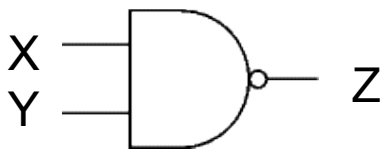


Multi-input logic gates

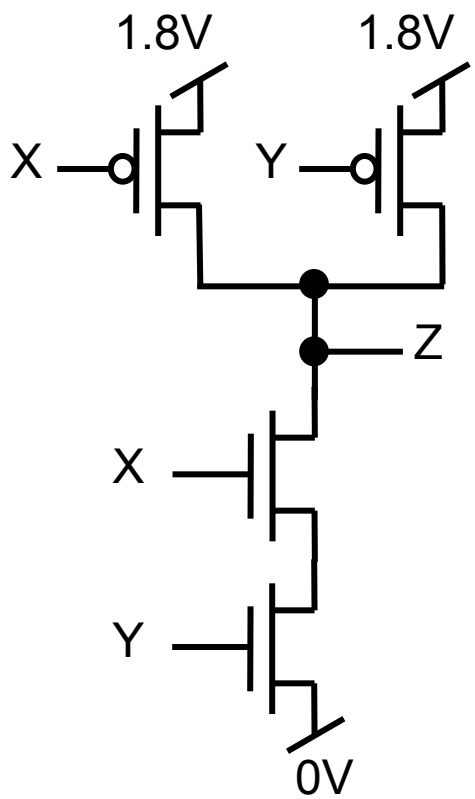


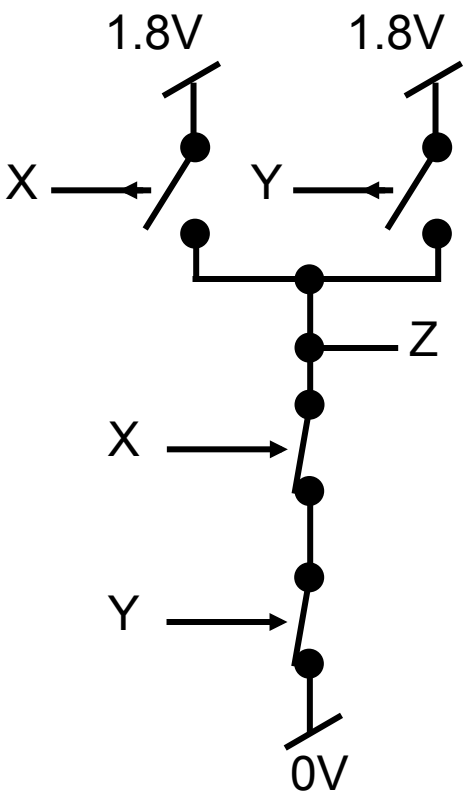
Claude Shannon – 1938

- CMOS logic gates are inverting
  - Easy to implement NAND, NOR, NOT while AND, OR, and Buffer are harder



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0





Possible logic functions of two variables

- There are 16 possible functions of 2 input variables:
  - in general, there are 2**(2**n) functions of n inputs

X	Y	16 possible functions (F_0–F_{15})															
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1	0	1
		0	X and Y	X	Y		X xor Y		X or Y	X nor Y not (X or Y)	X = Y	not Y		not X		X nand Y not (X and Y)	1

Proving theorems (perfect induction)

- Using perfect induction (complete truth table):
  - e.g., de Morgan's:

$(X + Y)' = X' \bullet Y'$

NOR is equivalent to AND with inputs complemented

X	Y	X′	Y′	(X + Y)′	X′ • Y′
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

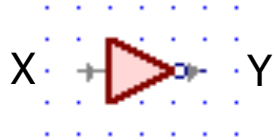
$(X \bullet Y)' = X' + Y'$

NAND is equivalent to OR with inputs complemented

X	Y	X′	Y′	(X • Y)′	X′ + Y′
0	0	1	1		
0	1	1	0		
1	0	0	1		
1	1	0	0		

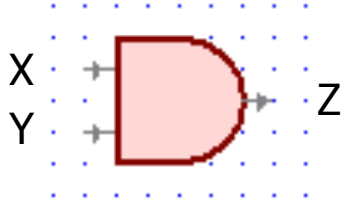
From Boolean expressions to logic gates

- NOT X' $\overline{X}$ $\sim X$ $X/$



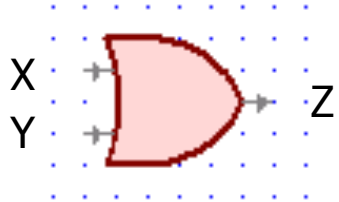
X	Y
0	1
1	0

- AND $X \bullet Y$ XY $X \wedge Y$



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

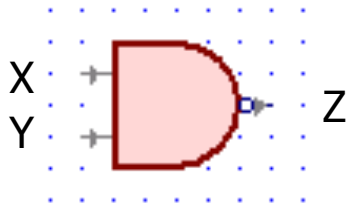
- OR $X + Y$ $X \vee Y$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

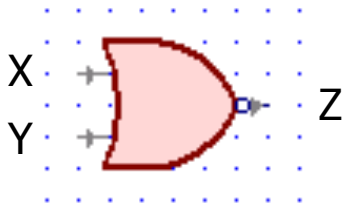
From Boolean expressions to logic gates (cont'd)

- NAND



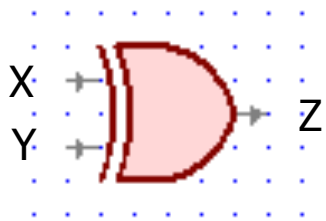
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

- NOR



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

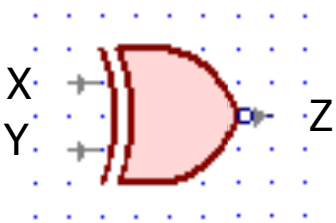
- XOR
  $X \oplus Y$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

X <u>xor</u> Y = X Y' + X' Y
X or Y but not both
("inequality", "difference")

- XNOR
  X = Y



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

X <u>xnor</u> Y = X Y + X' Y'
X and Y are the same
("equality", "coincidence")

Canonical forms

- Truth table is the unique signature of a Boolean function
- The same truth table can have many gate realizations
  - we've seen this already
  - depends on how good we are at Boolean simplification
- Canonical forms
  - standard forms for a Boolean expression
  - we all come up with the same expression

Sum-of-products canonical forms

- Also known as disjunctive normal form
- Also known as minterm expansion

F = *001* *011* *101* *110* *111*

F = A′B′C + A′BC + AB′C + ABC′ + ABC

A	B	C	F	F′
0	0	0	0	1
0	0	1	1	0
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	0

F′ = A′B′C′ + A′BC′ + AB′C′

Product-of-sums canonical form

- Also known as conjunctive normal form
- Also known as maxterm expansion

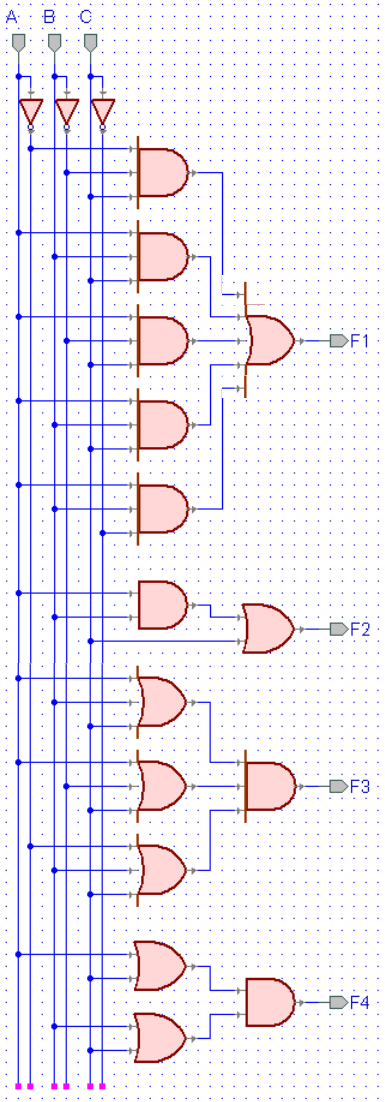
F = *000* *010* *100*

F = (A + B + C) (A + B′ + C) (A′ + B + C)

A	B	C	F	F′
0	0	0	0	1
0	0	1	1	0
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	0

F′ = (A + B + C′) (A + B′ + C′) (A′ + B + C′) (A′ + B′ + C) (A′ + B′ + C′)

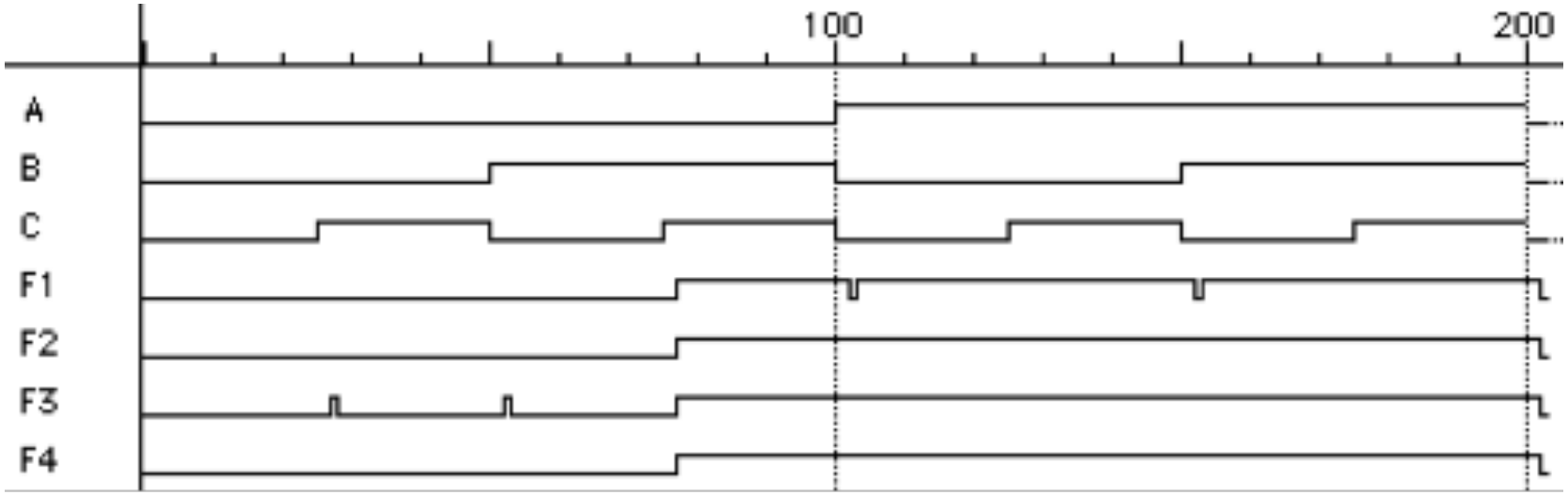
Four alternative two-level implementations of F = AB + C



	Transistors (NOT = 2)	Delay (approx) (NOT = 1)	Hazards
F1	*5 3-input NANDs* *1 5-input NAND* 5*6 + 1*10 = 40	*2 levels* 3^2 + 5^2 = 34	yes
F2	*2 2-input NANDs* 2*4 = 8	*2 levels* 2^2 + 2^2 = 8	no
F3	*4 3-input NANDs* 4*6 = 24	*2 levels* 3^2 + 3^2 = 18	yes
F4	*3 2-input NANDs* 3*4 = 12	*2 levels* 2^2 + 2^2 = 8	no

Waveforms for the four alternatives

- Waveform: just a sideways truth table
  - but note how edges don't line up exactly
  - it takes time for a gate to switch its output!
- Waveforms are essentially identical
  - except for timing hazards (glitches)
  - delays almost identical (modeled as a delay per level, not type of gate or number of inputs to gate)



Mapping truth tables to logic gates

- Given a truth table:
  1. Write the Boolean expression
  2. Minimize the Boolean expression
  3. Draw as gates
  4. Map to available gates

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

1

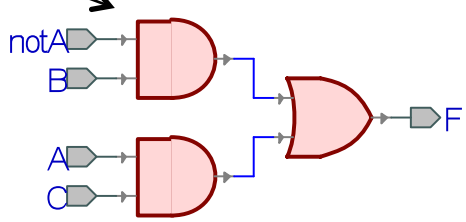
$$F = A'BC'+A'BC+AB'C+ABC$$

2

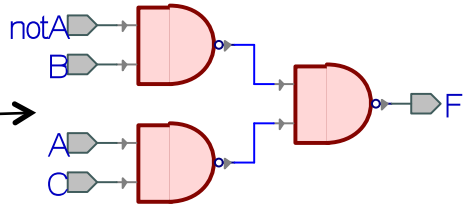
$$= A'B(C'+C)+AC(B'+B)$$

$$= A'B+AC$$

3



4



Which realization is best?

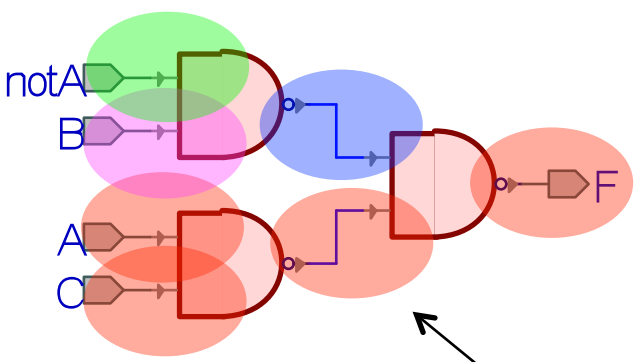
- Reduce number of inputs
  - fewer literals (input variables) means less transistors -> smaller circuits
  - fewer inputs implies faster gates -> gates are smaller and thus also faster
  - fan-ins (# of gate inputs) are limited in some technologies
- Reduce number of gates
  - fewer gates (and the packages they come in) means smaller circuits
- Reduce number of levels of gates
  - fewer level of gates implies reduced signal propagation delays
- How do we explore tradeoffs?
  - automated tools to generate synthesize solutions -> mostly good

Random logic gates

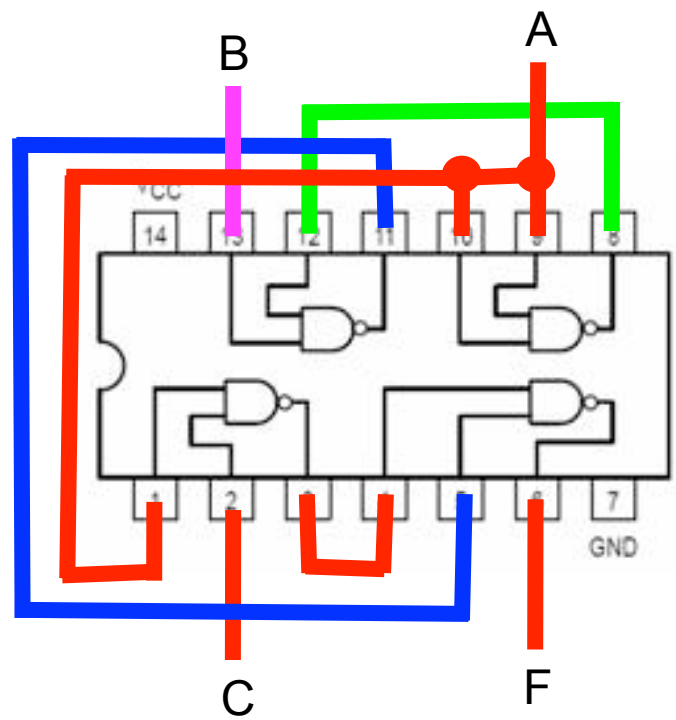
- Transistors quickly integrated into logic gates (1960s)
- Catalog of common gates (1970s)
  - Texas Instruments Logic Data Book – the yellow “bible”
  - all common packages listed and characterized (delays, power)
  - typical packages:
    - in 14-pin IC: 6-inverters, 4 NAND gates, 4 XOR gates
- Today, very few of these parts are still in use
- However, parts libraries exist for chip design
  - designers reuse already characterized logic gates on chips
  - same reasons as before
  - difference is that the parts don’t exist in physical inventory – created as needed

Mapping truth tables to logic gates

- Given a truth table:
  - Write the Boolean expression
  - Minimize the Boolean expression
  - Draw as gates
  - Map to available gates
  - Determine number of packages and their connections

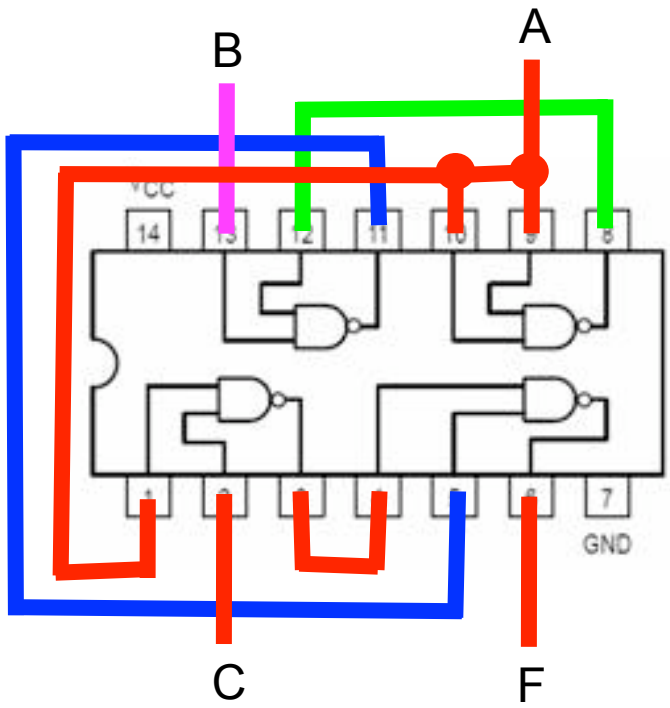


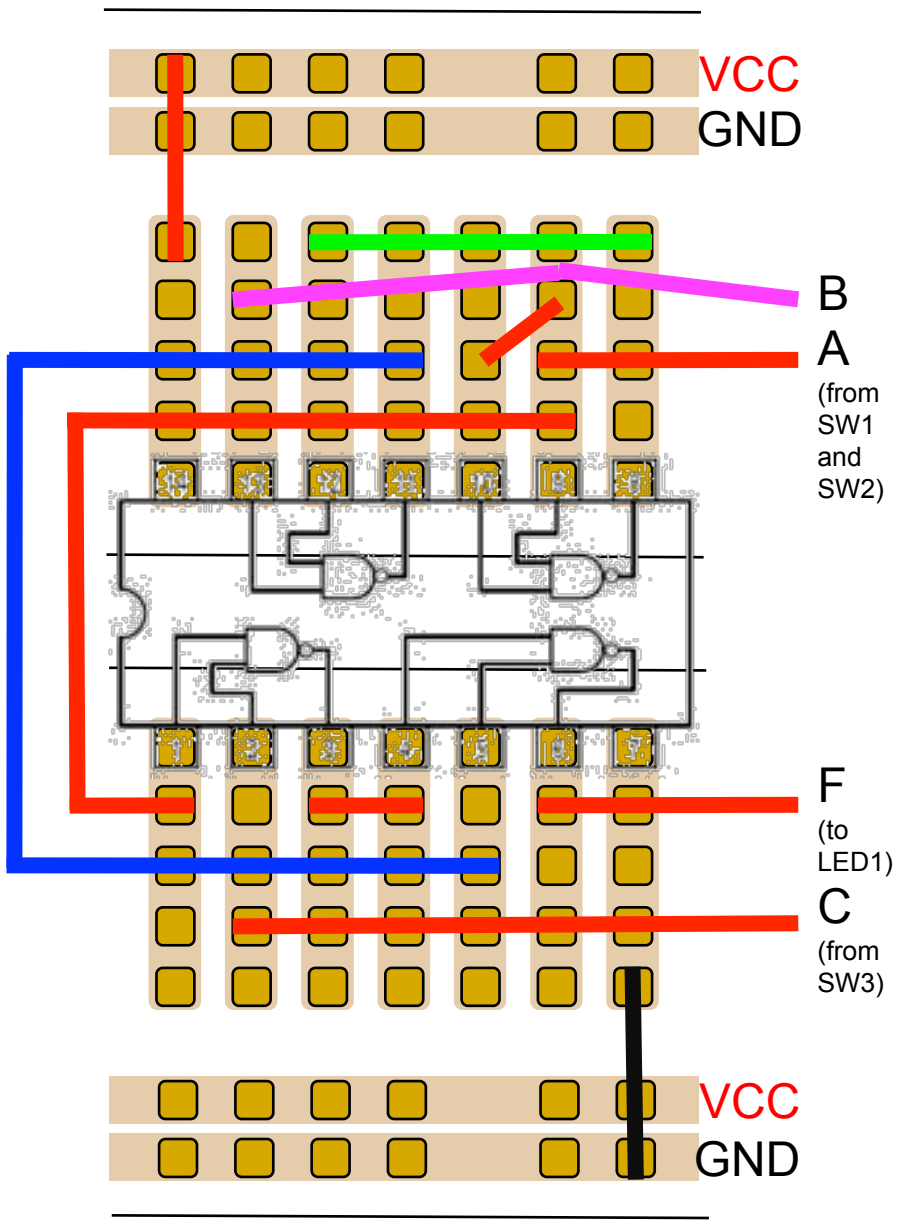
4



7 nets (wires) in this design

Breadboarding circuits





Random logic

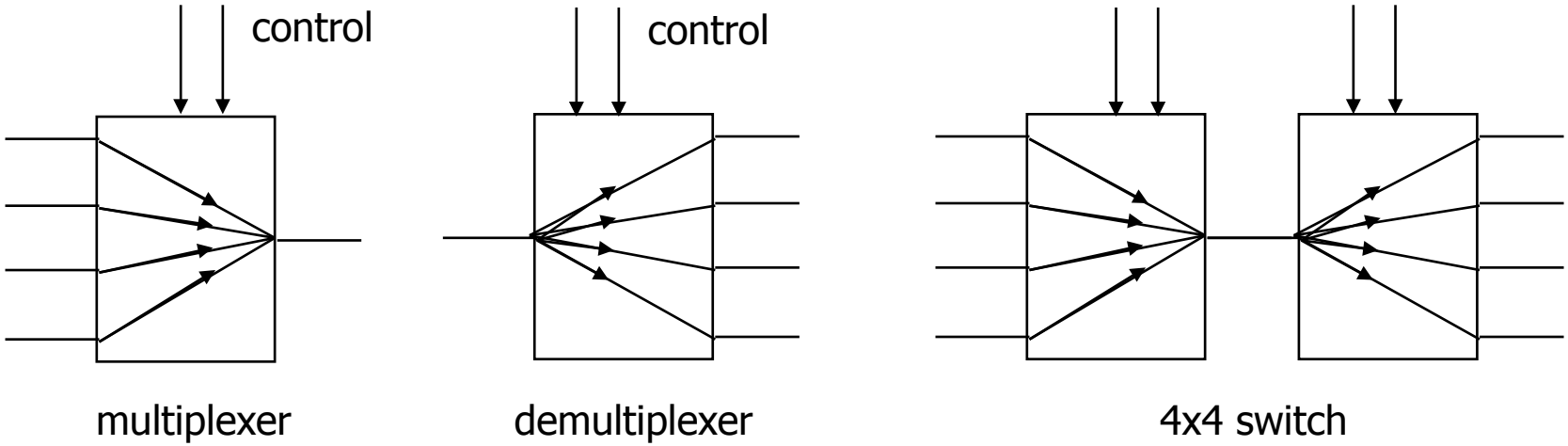
- Too hard to figure out exactly what gates to use
  - map from logic to NAND/NOR networks
  - determine minimum number of packages
    - slight changes to logic function could decrease cost
- Changes too difficult to realize
  - need to rewire parts
  - may need new parts
  - design with spares (few extra inverters and gates on every board)
- Need higher levels of integration to keep costs down
  - cost directly related to number of devices and their pins

Regular logic

- Need to make design faster
- Need to make engineering changes easier to make
- Simpler for designers to understand and map to functionality
  - harder to think in terms of specific gates
  - easier to think in terms of larger multi-purpose blocks

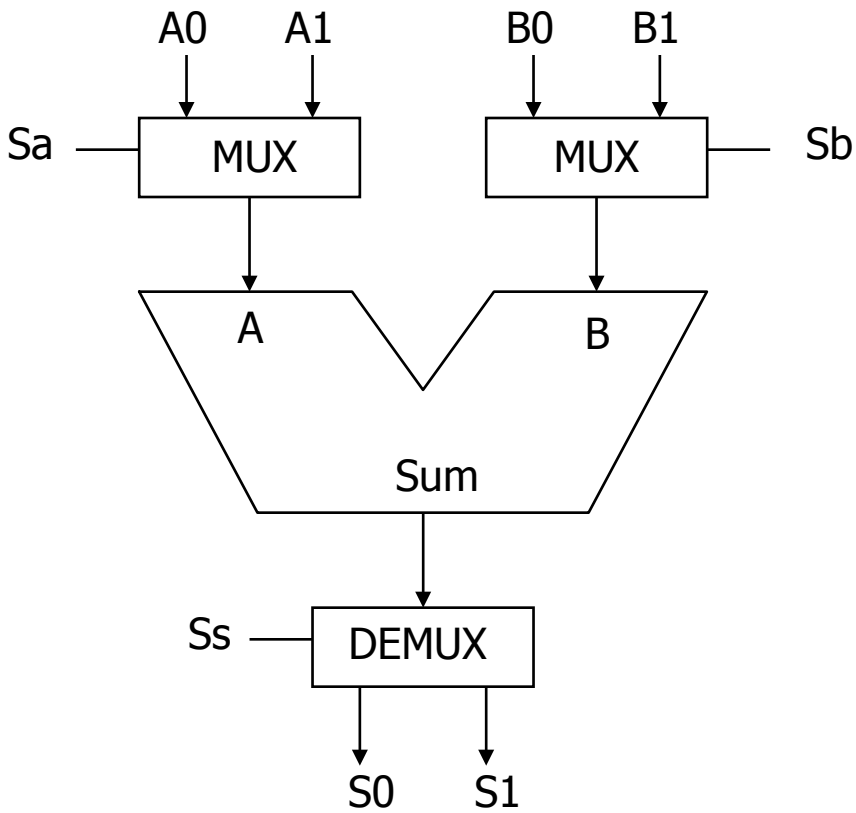
Making connections

- Direct point-to-point connections using wires
- Route one of many inputs to a single output --- multiplexer
- Route a single input to one of many outputs --- demultiplexer



Mux and demux (cont'd)

- Uses of multiplexers/demultiplexers in multi-point connections



multiple input sources

multiple output destinations

Multiplexers/selectors

- Multiplexers/selectors: general concept
  - 2^n data inputs, n control inputs (called "selects"), 1 output
  - used to connect 2^n points to a single point
  - control signal pattern forms binary index of input connected to output

$Z = A' I_0 + A I_1$

A	Z
0	I_0
1	I_1

functional form

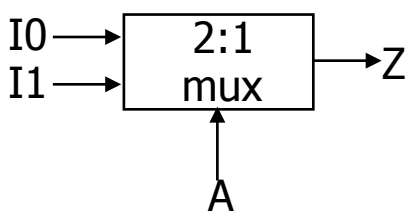
logical form

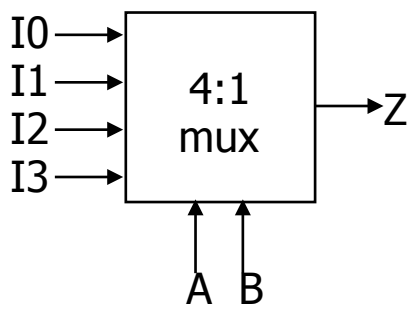
I_1	I_0	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

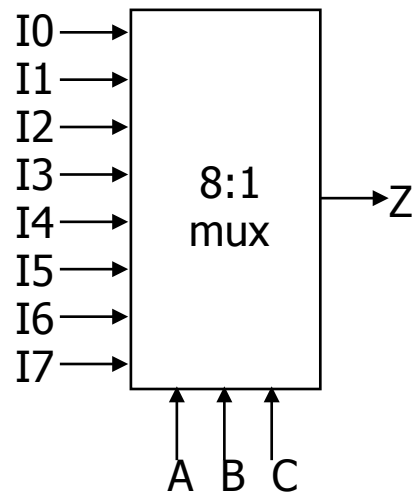
two alternative forms
for a 2:1 Mux truth table

Multiplexers/selectors (cont'd)

- 2:1 mux: $Z = A'I_0 + AI_1$
- 4:1 mux: $Z = A'B'I_0 + A'BI_1 + AB'I_2 + ABI_3$
- 8:1 mux: $Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7$
- In general: $Z = \Sigma_{k=0}^{2^n-1}(m_k I_k)$
  - in minterm shorthand form for a 2^n:1 Mux

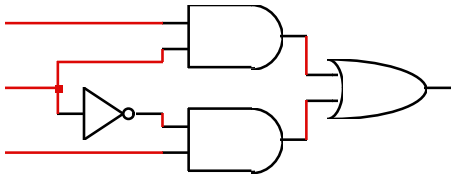


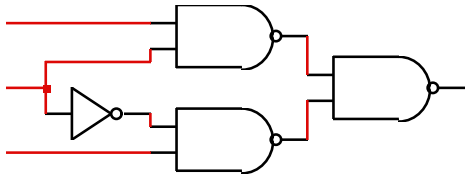




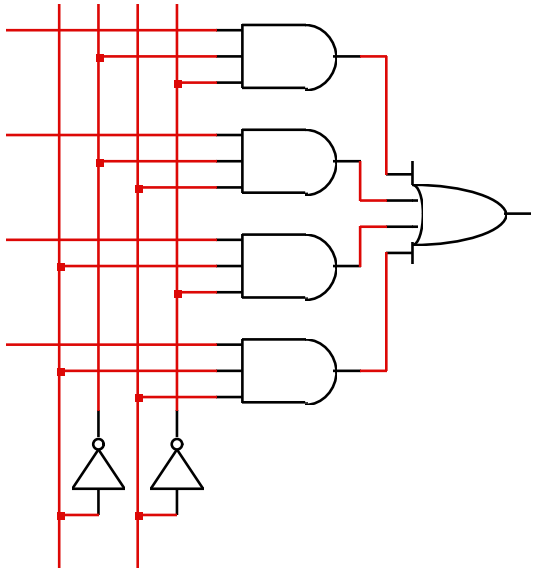
Gate level implementation of muxes

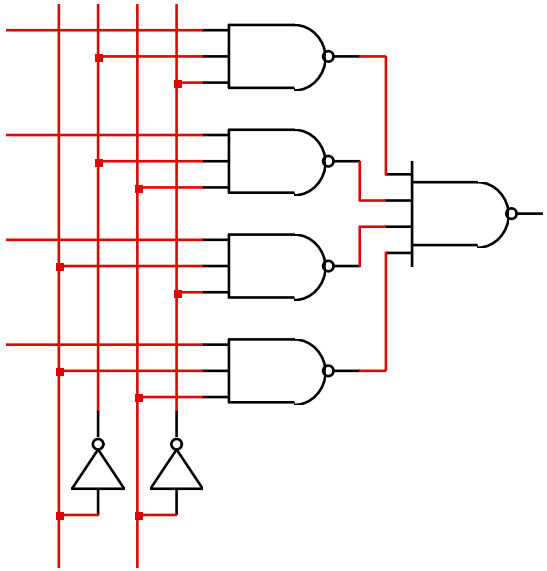
- 2:1 mux





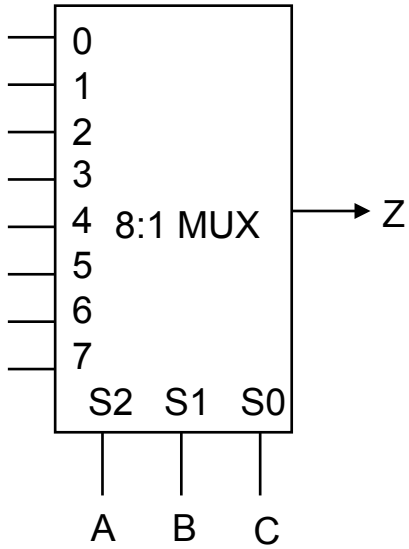
- 4:1 mux





Multiplexers as general-purpose logic

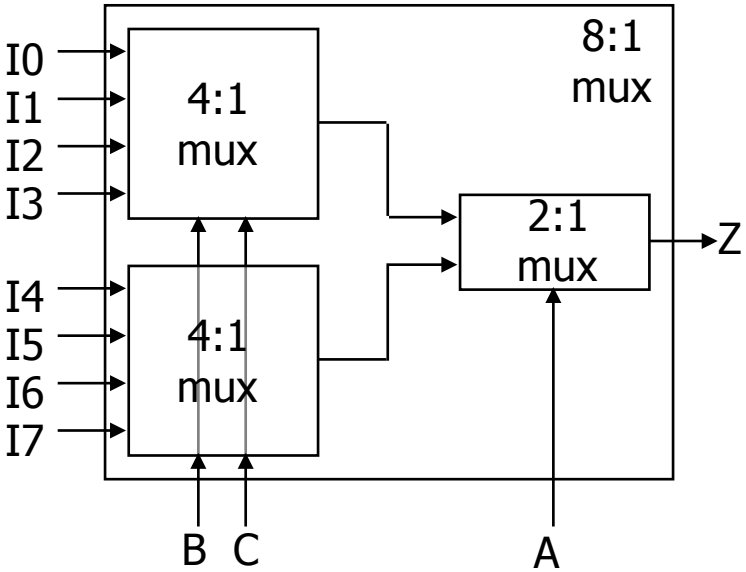
- A 2^n:1 multiplexer can implement any function of n variables
  - with the variables used as control inputs and
  - the data inputs tied to 0 or 1
  - in essence, a lookup table (LUT), basis of FPGAs
- Example:
  - F(A,B,C) = m0 + m2 + m6 + m7
    = A'B'C' + A'BC' + ABC' + ABC



$$Z = A'B'C'I_0 + A'B'CI_1 + A'BC'I_2 + A'BCI_3 + AB'C'I_4 + AB'CI_5 + ABC'I_6 + ABCI_7$$

Cascading multiplexers

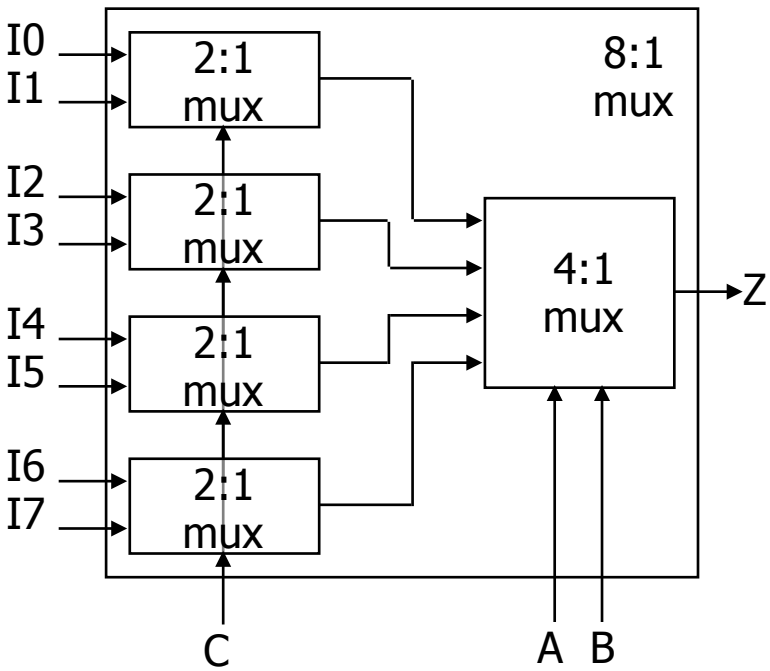
- Large multiplexers can be made by cascading smaller ones



control signals B and C simultaneously choose one of I0, I1, I2, I3 and one of I4, I5, I6, I7

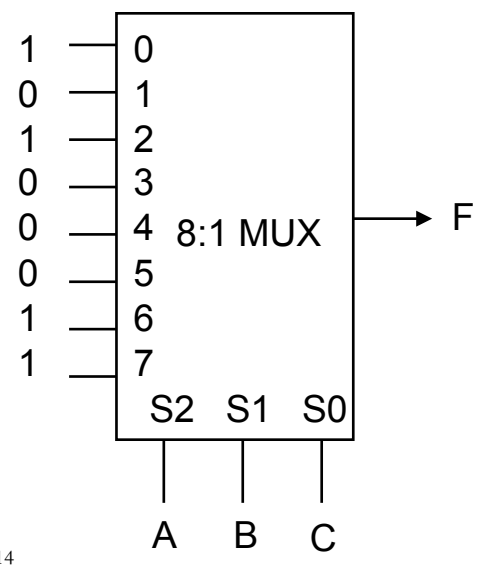
control signal A chooses which of the upper or lower mux's output to gate to Z

alternative implementation

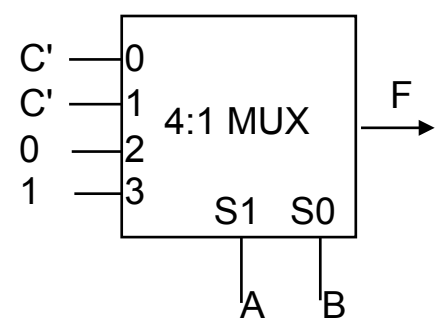


Multiplexers as general-purpose logic (cont'd)

- A 2^{n-1}:1 multiplexer can implement any function of n variables
  - with n-1 variables used as control inputs and
  - the data inputs tied to the last variable or its complement
- Example:
  - F(A,B,C) = m0 + m2 + m6 + m7
    = 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	C'
0	0	1	0	
0	1	0	1	C'
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	



Demultiplexers/decoders

- Decoders/demultiplexers: general concept
  - single data input, n control inputs, 2^n outputs
  - control inputs (called "selects" (S)) represent binary index of output to which the input is connected
  - data input usually called "enable" (G)

1:2 Decoder:

$O0 = G \bullet S'$
$O1 = G \bullet S$

2:4 Decoder:

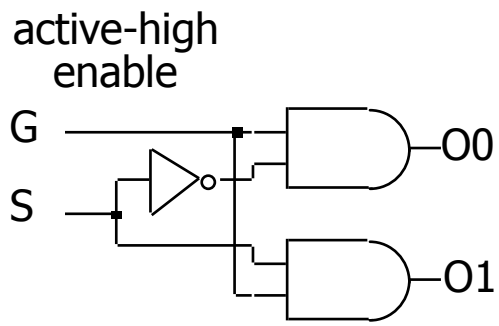
$O0 = G \bullet S1' \bullet S0'$
$O1 = G \bullet S1' \bullet S0$
$O2 = G \bullet S1 \bullet S0'$
$O3 = G \bullet S1 \bullet S0$

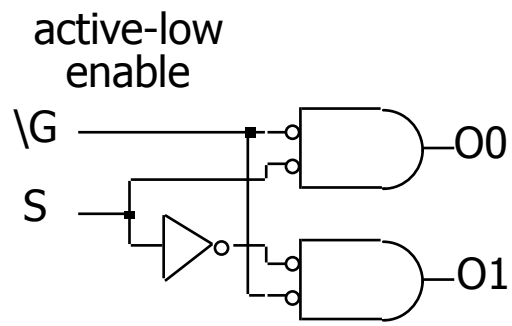
3:8 Decoder:

$O0 = G \bullet S2' \bullet S1' \bullet S0'$
$O1 = G \bullet S2' \bullet S1' \bullet S0$
$O2 = G \bullet S2' \bullet S1 \bullet S0'$
$O3 = G \bullet S2' \bullet S1 \bullet S0$
$O4 = G \bullet S2 \bullet S1' \bullet S0'$
$O5 = G \bullet S2 \bullet S1' \bullet S0$
$O6 = G \bullet S2 \bullet S1 \bullet S0'$
$O7 = G \bullet S2 \bullet S1 \bullet S0$

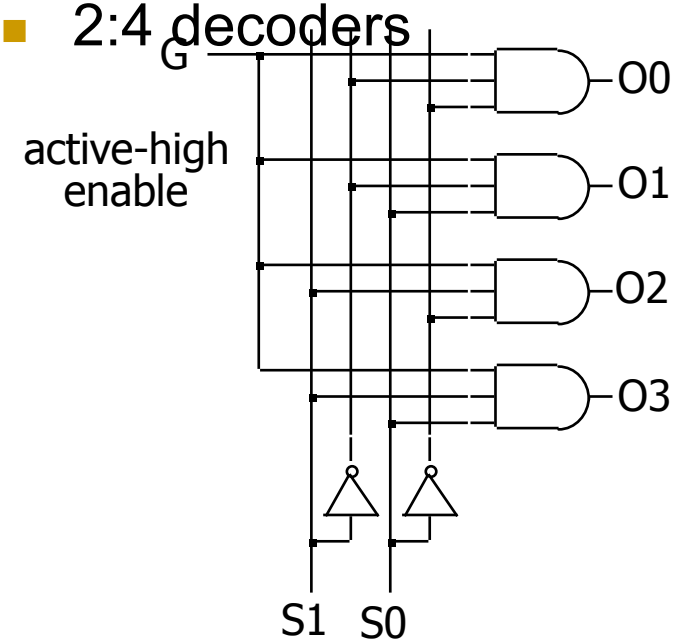
Gate level implementation of demultiplexers

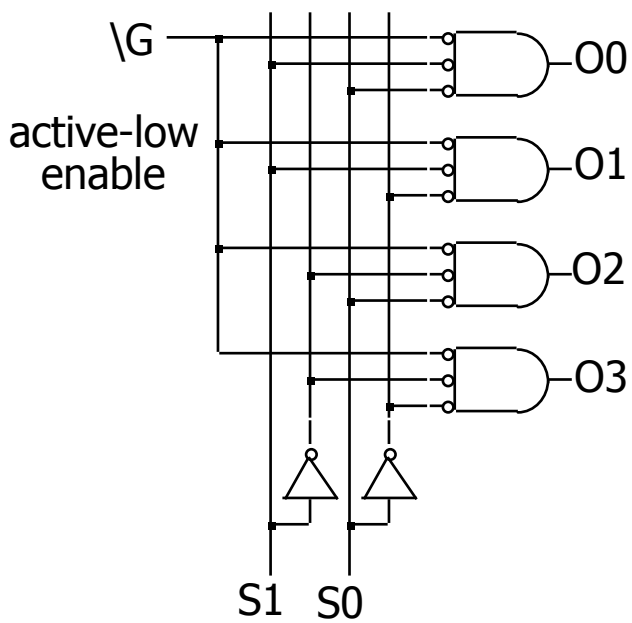
- 1:2 decoders





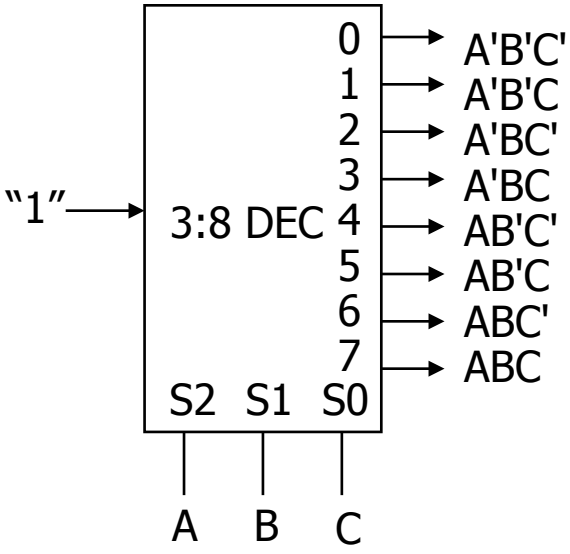
- 2:4 decoders





Demultiplexers as general-purpose logic

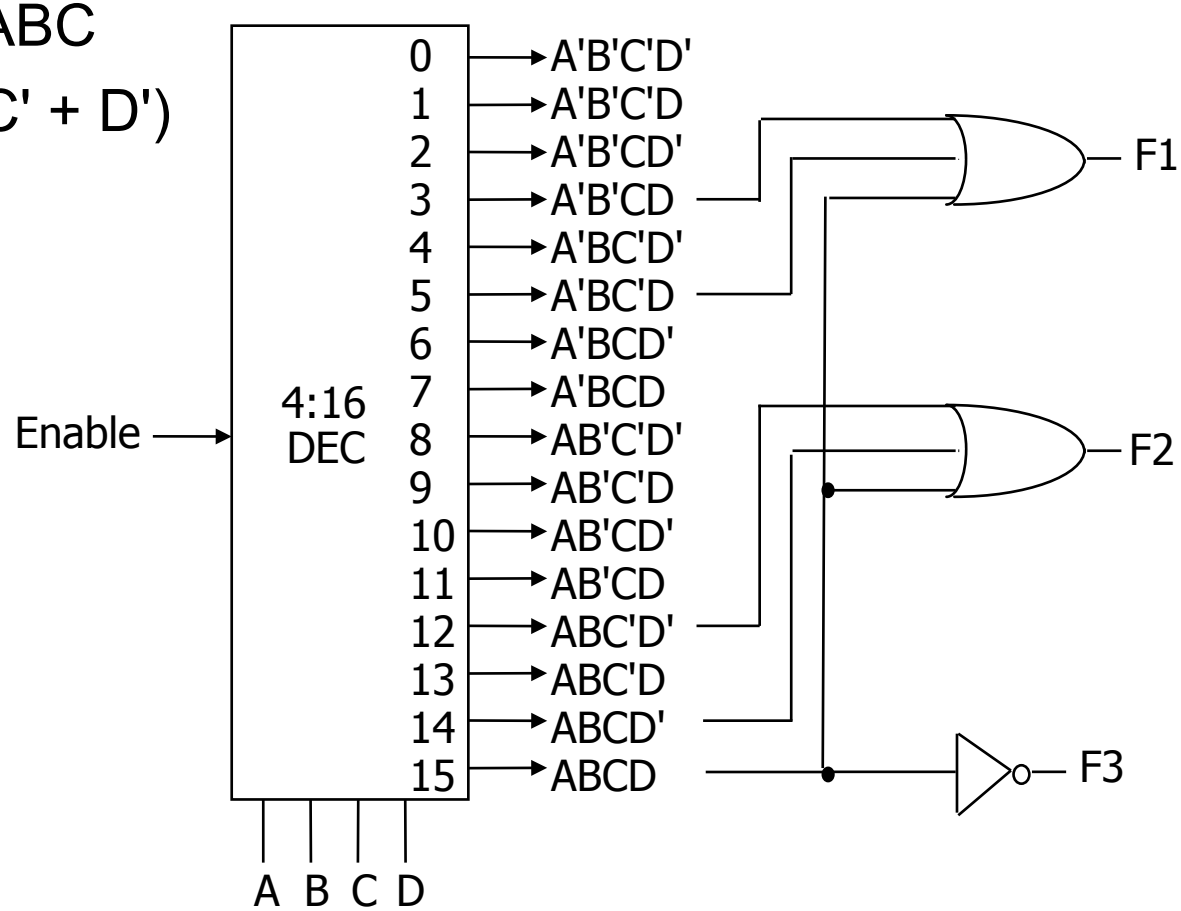
- A n:2^n decoder can implement any function of n variables
  - with the variables used as control inputs
  - the enable inputs tied to 1 and
  - the appropriate minterms summed to form the function



demultiplexer generates appropriate minterm based on control signals (it "decodes" control signals)

Demultiplexers as general-purpose logic (cont'd)

- F1 = A'BC'D + A'B'CD + ABCD
- F2 = ABC'D' + ABC
- F3 = (A' + B' + C' + D')



Cascading decoders

- 5:32 decoder
  - 1x2:4 decoder
  - 4x3:8 decoders