

Common Combinational Logic Circuits

- Adders
 - Subtraction typically via 2s complement addition
- Multiplexers
 - N control signals select 1 of up to 2^N inputs as output
- Demultiplexers
 - N control signals select input to go to 1 of up to 2^N outputs
- Decoders
 - N inputs produce M outputs (typically $M > N$)
- Encoders
 - N inputs produce M outputs (typically $N > M$)
- Converter (same as decoder or encoder)
 - N inputs produce M outputs (typically $N = M$)

More Common Circuits

- Comparators
 - Compare two N -bit binary values
 - Equal-to or Not-equal-to
 - Easiest to design
 - Greater-than, Less-than, Greater-than-or-equal-to, etc.
 - Require adders
- Parity check/generate circuit
 - Calculates even or odd parity over N bits of data
 - Checks for good/bad parity (parity errors) on incoming data

Adders

- Consider i^{th} column addition of 2 binary numbers (A and B)
 - $A_i + B_i + \text{Cin}_i = \text{Cout}_i + \text{Sum}_i$
 - Derive truth table
 - Populate K-maps
 - Obtain minimized SOPs
 - Draw logic diagram
 - Optimize with P&Ts

Truth Table

A	B	C	Co	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

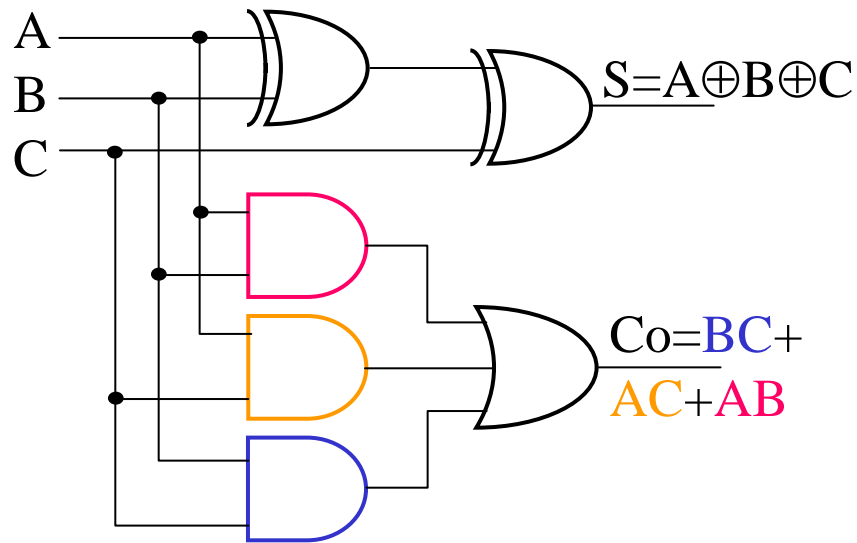
		BC			
		00	01	11	10
A	0	0	1	0	1
	1	1	0	1	0

$$\begin{aligned}
 S &= A'B'C + A'BC' \\
 &\quad + AB'C' + ABC \\
 &= A'(B \oplus C) + A(B \oplus C) \\
 &= A \oplus B \oplus C
 \end{aligned}$$

		BC			
		00	01	11	10
A	0	0	0	1	0
	1	0	1	1	1

$$Co = BC + AC + AB$$

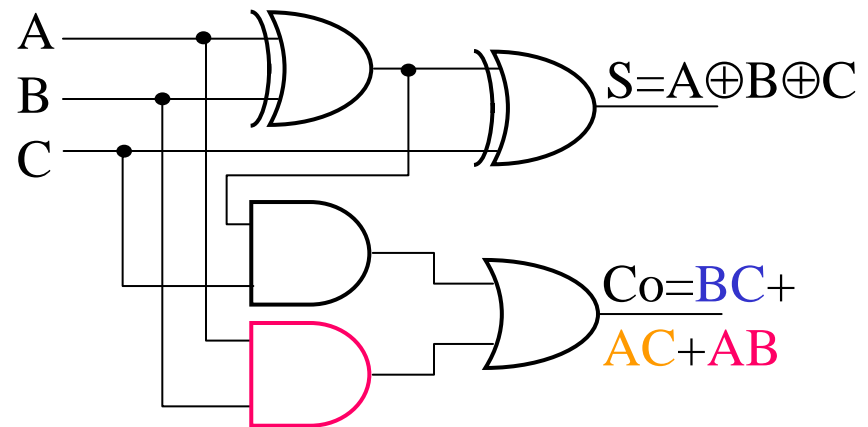
Adders



		BC			
		00	01	11	10
A	0	0	0	1	0
	1	0	1	1	1

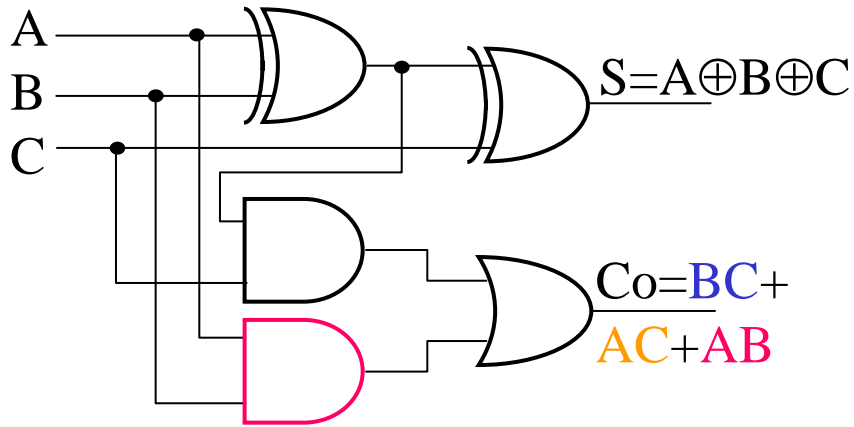
$$\begin{aligned}
 Co &= A'BC + AB'C + AB \\
 &= C(A'B + AB') + AB \\
 &= C(A \oplus B) + AB
 \end{aligned}$$

Taking advantage of common product terms between S and Co we see that we can use the XOR gate for $A \oplus B$ to reduce the gate count

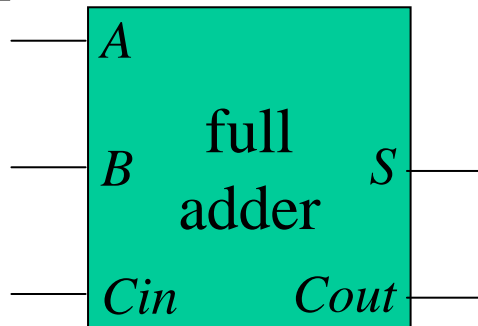


Adders

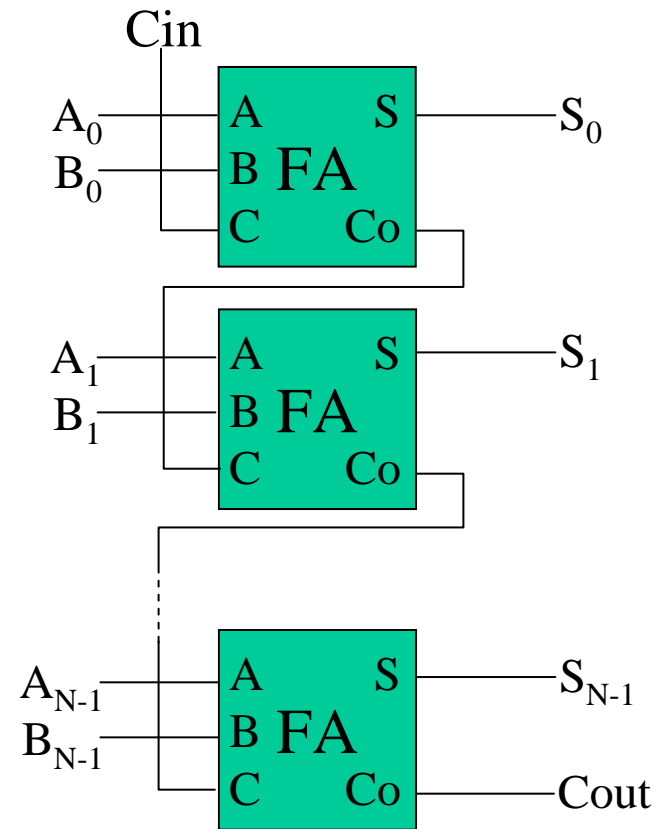
referred to as a *full adder*



we can let a block represent the full adder



now we can build an N -bit adder from N full adders



Subtractors

Build an N -bit subtractor from an N -bit adder using 2's complement

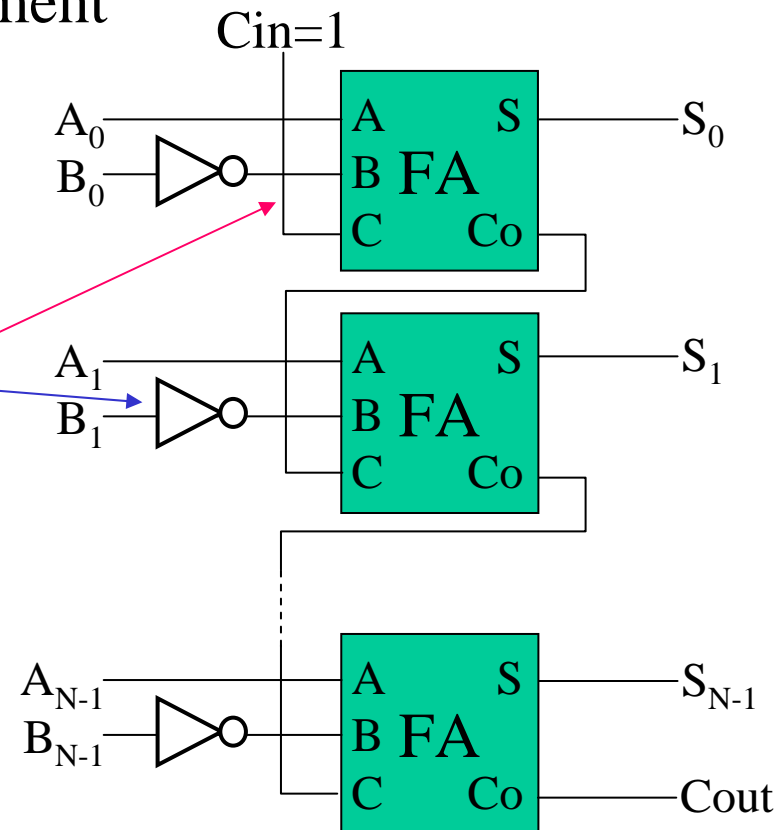
- Recall the 2's complement transformation for a negative number:

1) invert

2) then add 1

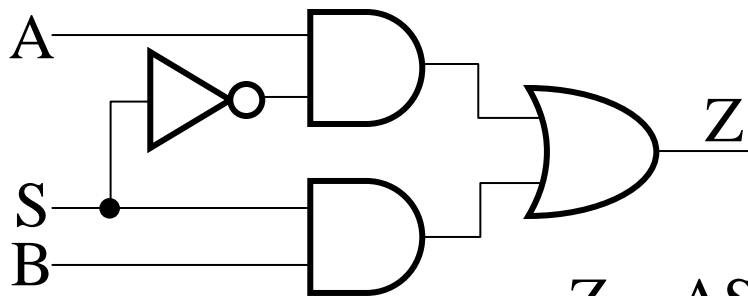
*here we use C_{in}
to add a 1*

- Therefore, $S=A-B$
- Note that this includes a sign bit (S_{N-1})



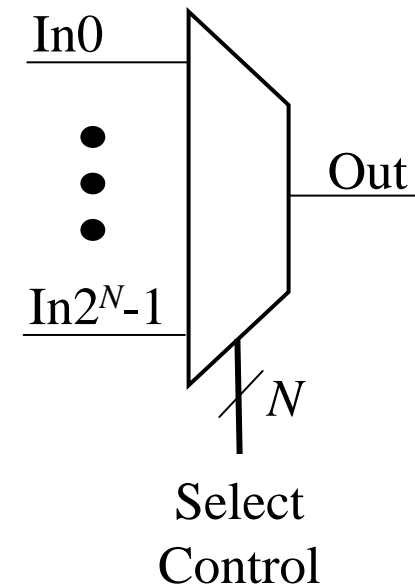
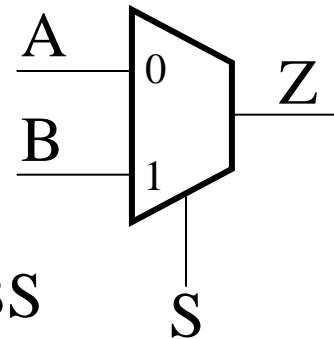
Multiplexers

- N control signals select 1 of up to 2^N inputs as output
 - Sometimes called selectors
 - We looked at a 2-to-1 MUX



$$Z = AS' + BS$$

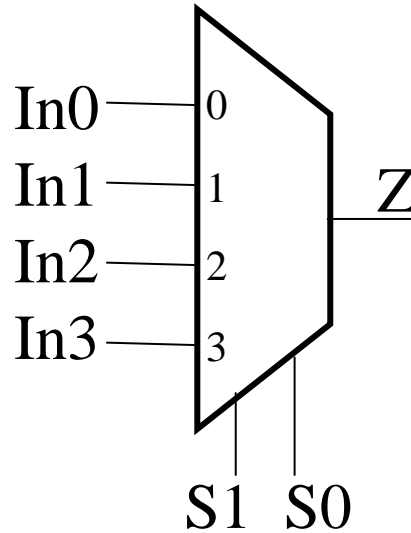
if $S=0$, then $Z=A$
else if $S=1$, then $Z=B$



Short-hand Truth Table

Multiplexers

- 4-to-1 MUX
 - 4 inputs
 - In0-3
 - 2 controls
 - S1, S0 (LSB)
 - 1 output
 - Z

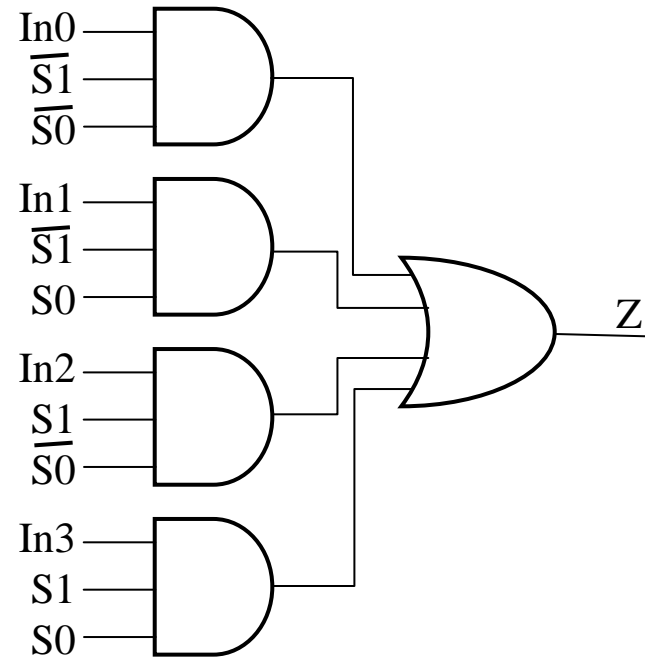


S1	S0	Z
0	0	In0
0	1	In1
1	0	In2
1	1	In3

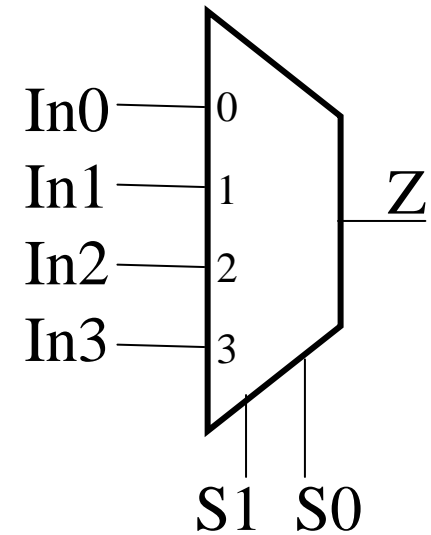
- Can generated any size MUX

$$Z = \text{In0 } S1' S0' + \text{In1 } S1' S0 + \text{In2 } S1 S0' + \text{In3 } S1 S0$$

SOP obtained directly from short-hand Truth Table

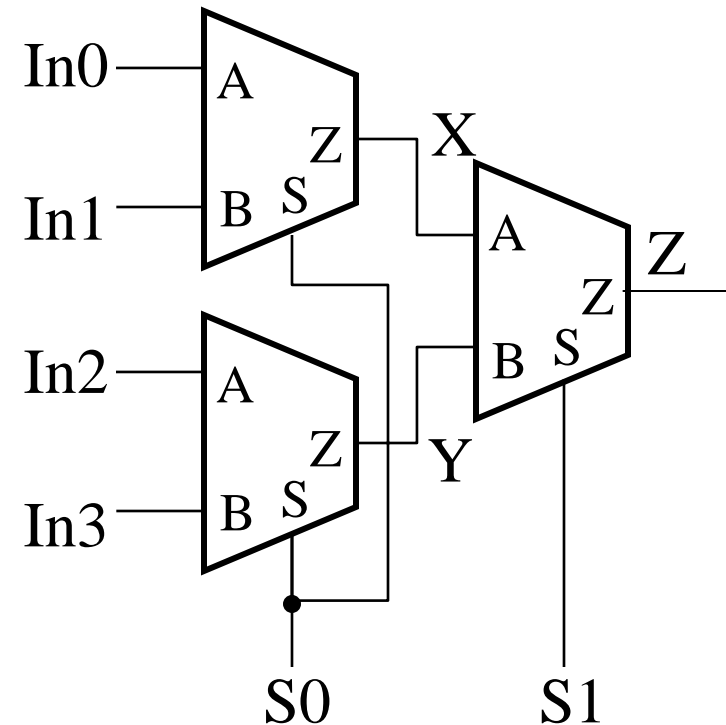


Multiplexers



- 4-to-1 MUX
 - Built from 2-to-1 MUXs
- Can generate any size MUX

$$\begin{aligned}
 Z &= X S1' + Y S1 \\
 &= (\text{In0 } S0' + \text{In1 } S0) S1' \\
 &\quad + (\text{In2 } S0' + \text{In3 } S0) S1 \\
 &= \text{In0 } S1' S0' + \text{In1 } S1' S0 \\
 &\quad + \text{In2 } S1 S0' + \text{In3 } S1 S0
 \end{aligned}$$



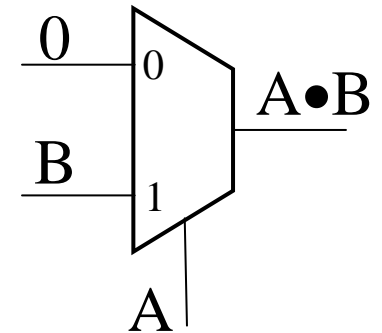
Multiplexers

- Recall Shannon's Expansion Theorem
 - T10a: $f(X_1, X_2, \dots, X_{n-1}, X_n) = (X_1' \cdot f(0, X_2, \dots, X_{n-1}, X_n)) + (X_1 \cdot f(1, X_2, \dots, X_{n-1}, X_n))$
 - A 2-to-1 MUX is X_1 as select input
 - Can be taken further:
 - $f(X_1, X_2, \dots, X_{n-1}, X_n) = (X_1' \cdot X_2' \cdot f(0, 0, \dots, X_{n-1}, X_n)) + (X_1 \cdot X_2' \cdot f(1, 0, \dots, X_{n-1}, X_n)) + (X_1' \cdot X_2 \cdot f(0, 1, \dots, X_{n-1}, X_n)) + (X_1 \cdot X_2 \cdot f(1, 1, \dots, X_{n-1}, X_n))$
 - A 4-to-1 MUX with $X_1 X_2$ and X_1 as select inputs
- This says a MUX is functionally complete

Multiplexers

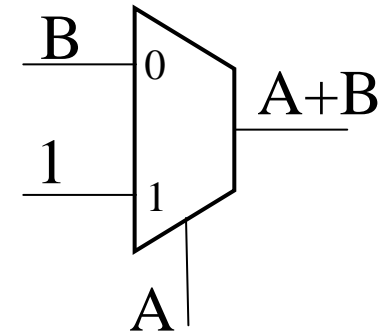
- AND gate

$$\begin{aligned}
 - f(A,B) &= AB = A' \bullet f(0,B) + A \bullet f(1,B) \\
 &= A'(0B) + A(1B) = A'0 + AB
 \end{aligned}$$



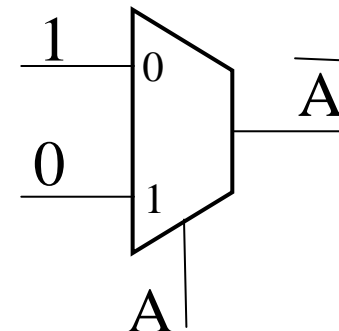
- OR gate

$$\begin{aligned}
 - f(A,B) &= A+B = A' \bullet f(0,B) + A \bullet f(1,B) \\
 &= A'(0+B) + A(1+B) = A'B + A1
 \end{aligned}$$



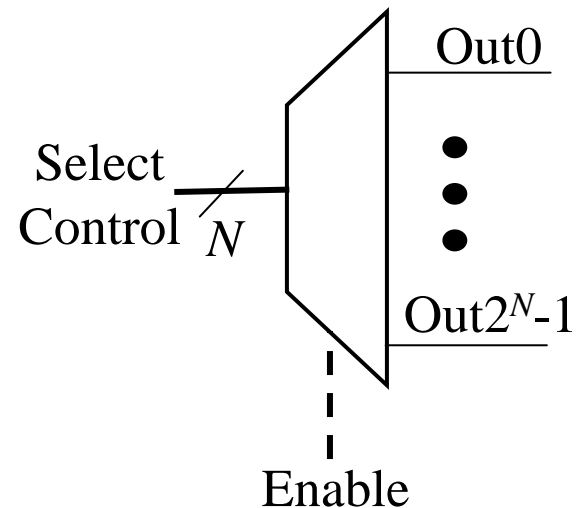
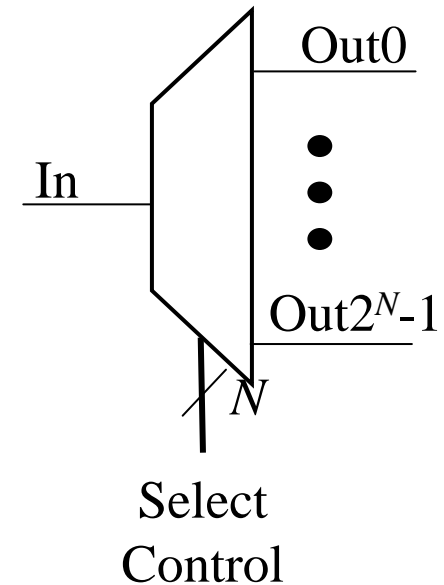
- NOT gate

$$\begin{aligned}
 - f(A) &= A' = A' \bullet f(0) + A \bullet f(1) \\
 &= A'(1) + A(0)
 \end{aligned}$$



Demultiplexers

- N control signals select input to go to 1 of up to 2^N outputs
- Opposite of MUXs
 - Sometimes called *de-selectors*
- Alternate view is a *decoder*
 - N inputs produce a logic 1 on 1 of up to 2^N outputs
 - An enable input can be added to “enable” the logic 1 on the output
 - Otherwise all outputs are 0
 - Now it’s same as the DMUX



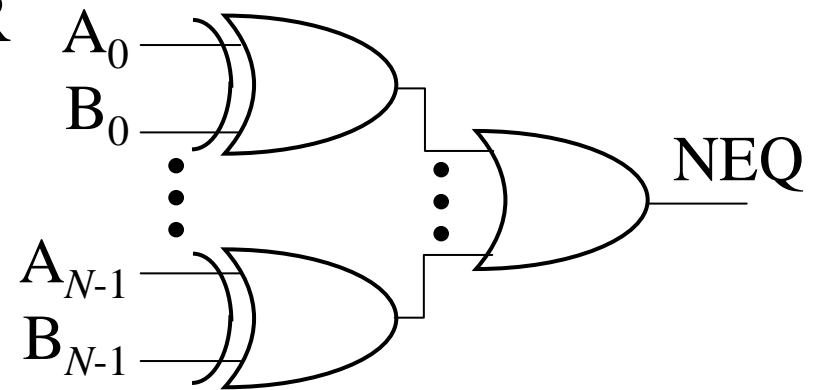
Decoders/Encoders

- Decoders
 - N inputs produce M outputs (typically $M > N$)
 - BCD to 7-segment *decoder* is a good example
- Encoders
 - N inputs produce M outputs (typically $N > M$)
 - We could just as easily have taken the 7-segment values (A-G) and encode to BCD or HEX
 - That would be a 7-variable K-map
- Converters
 - N inputs produce M outputs (typically $N = M$)
 - Excess-3 code to BCD *converter* is a good example

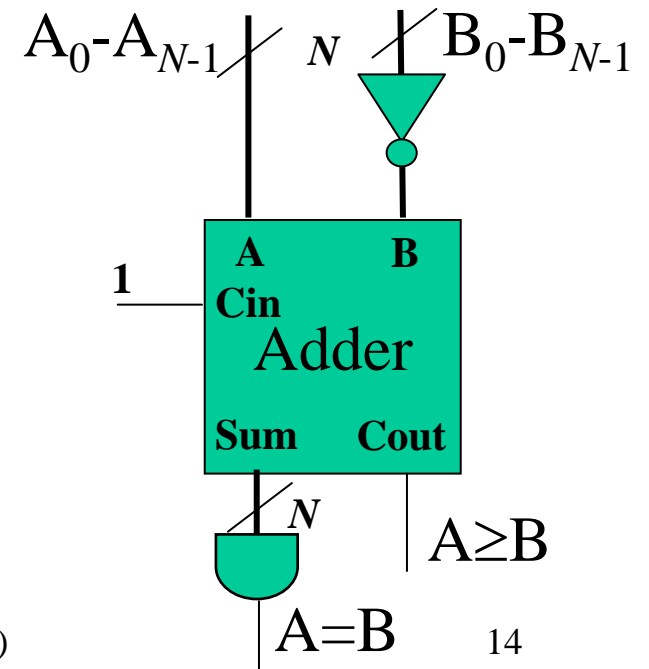
Comparators

- Equal-to comparators use XOR function

- XOR produces 1 when inputs differ
 - Do bit-wise compare of N pairs of bits
 - N XOR gates
- OR the XOR outputs to produce a 1 when the input values differ
 - An N -input OR gate
 - Invert the output (NOR) to produce a 1 when the input values are the same



- Greater/Less-than use 2s-comp subtraction



Parity Circuits

- Parity makes use of the XOR function
 - XOR produces a 1 for an odd # of 1s on input
 - This produces even parity over data + parity bit
 - For odd parity, we can invert the output (an XNOR)
 - Check for correct parity by comparing P_{gen} with incoming parity bit P_{in}
 - Another XOR gate for comparison

