CS 225 Switching Theory

Dr. Somanath Tripathy
Indian Institute of Technology Patna

Previous Class

Combinational Circuit logic design

This Class

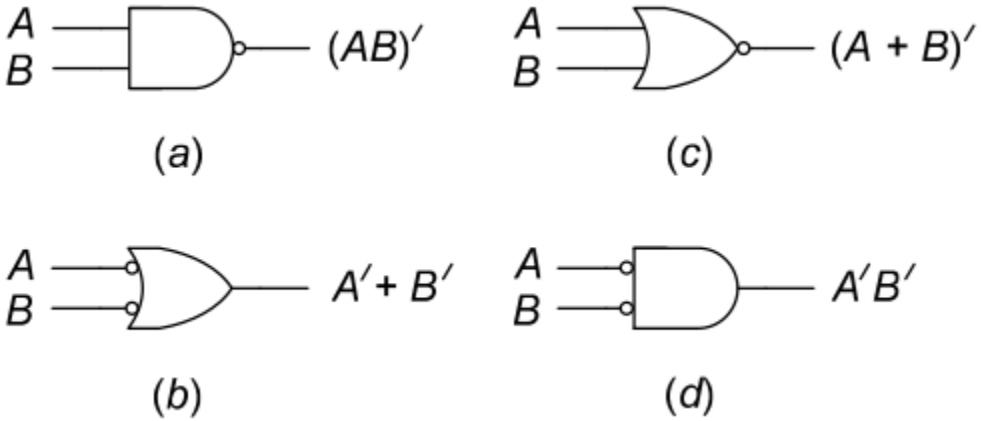
Combinational Circuit logic design

Combinational Logic Design Process

Step		Description
Step 1: Capture behavior	Capture the function	Create a truth table or equations, ***whichever is most natural for the given problem***, to describe the desired behavior of each output of the combinational logic.
Step 2: Convert to circuit	2A: Create equations	This substep is only necessary if you captured the function using a truth table instead of equations. Create an equation for each output by ORing all the minterms for that output. Simplify the equations if desired.
	2B: Implement as a gate-based circuit	For each output, create a circuit corresponding to the output's equation. (Sharing gates among multiple outputs is OK optionally.)

Warmup (NAND/NOR) Circuits

Switching algebra: not directly applicable to NAND/NOR logic



NAND and NOR gate symbols

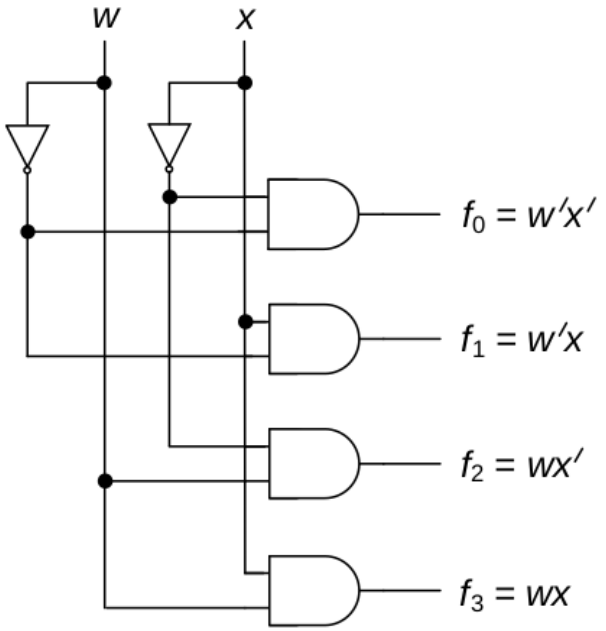
Warm-up (Decoders)

Decoders with n inputs and 2^n outputs: for any input combination, only one output is 1

Useful for:

- Routing input data to a specified output line, e.g., in addressing memory
- Basic building blocks for implementing arbitrary switching functions
- Code conversion
- Data distribution

Example: 2-to-4- decoder

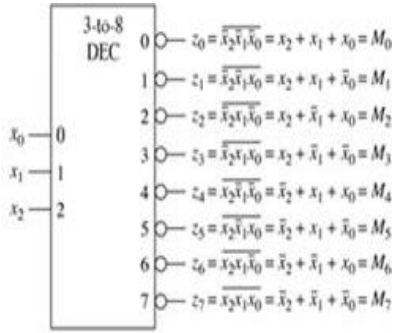


Warm-up (Decoders)

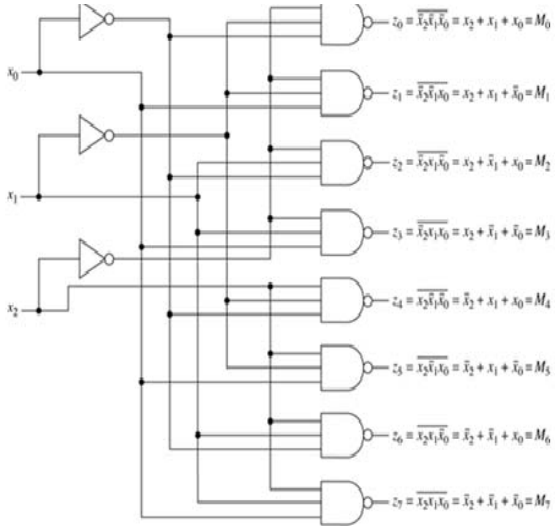
3-8 linde Decoder Truth table

Inputs			Outputs							
x_2	x_1	x_0	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

3-8 linde Bubbled output Decoder Symbol



3-8 linde Decoder using NanD gates Logic Diagram



Generates a single output (desired) low

Warmup (Implementing Boolean expression)

Using Decoder Ex.: Realize using 3x8 decoder

$f1 = \sum m(1,2,4,5)$ and $f2 = \sum m(1,5,7)$

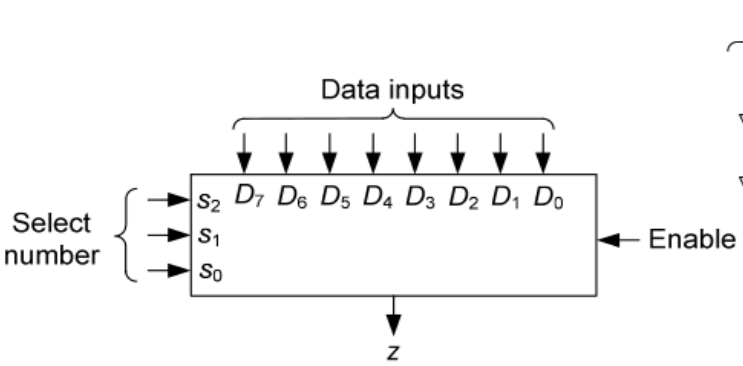
Realize the above function with given bubbled output decoder (Nand gate decoder)

Warmup (Multiplexer)

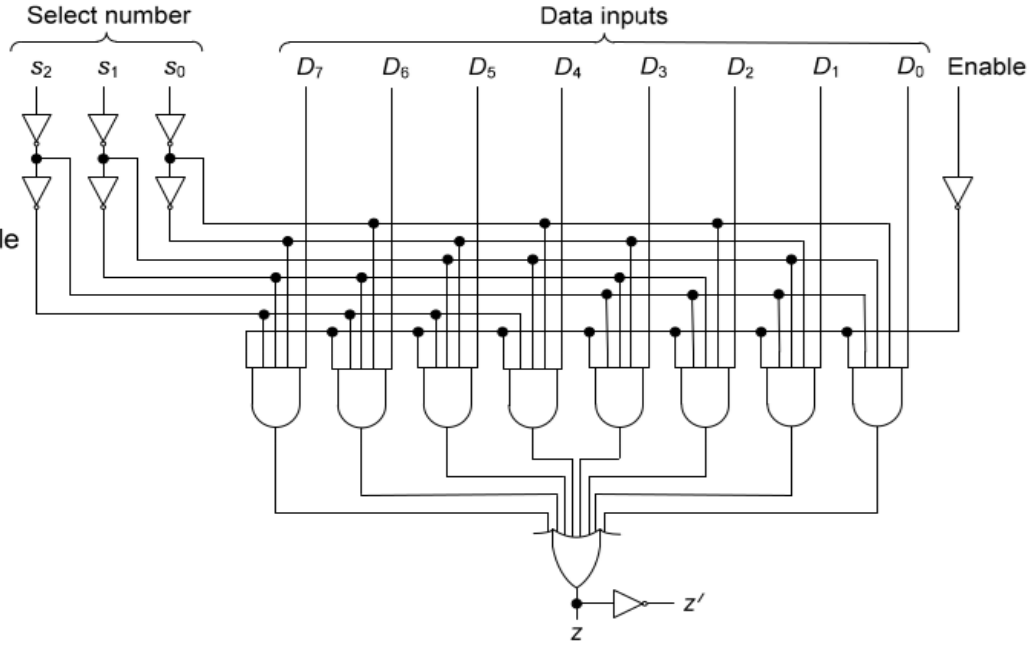
Multiplexer: electronic switch that connects one of n inputs to the output

Data selector: application of multiplexer

- n data input lines, D_0 , D_1 , ..., D_{n-1}
- m select digit inputs S_0 , S_1 , ..., s_{m-1}
- 1 output



(*a*) Block diagram.



(*b*) Logic diagram.

Implementing Boolean expression using

Using Multiplexer Ex.: Realize F (x,y,z) = ∑m(1,2,6,7) using 4x1 Mux

Design of High-speed Adders

Full adder: performs binary addition of three binary digits

- Inputs: arguments A and B and carry-in C
- Outputs: sum S and carry-out C_0

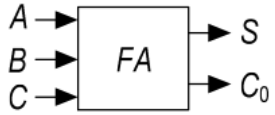
Example:

```
0  1  1     =        carry-in
           1  0  0  1  =      augend
           0  0  1  1  =   addend
          ------------
           1  1  1  0  =      sum
```

Truth table, block diagram and expressions:

A	B	C	S	C0
0	0	0	0	0
0	0	1	1	0
0	1	1	0	1
0	1	0	1	0
1	1	0	0	1
1	1	1	1	1
1	1	1	0	1
1	0	0	1	0

$$S = A'B'C + A'BC' + AB'C' + ABC$$
$$= A \oplus B \oplus C$$
$$C_0 = A'BC + ABC' + AB'C + ABC$$
$$= AB + AC + BC$$

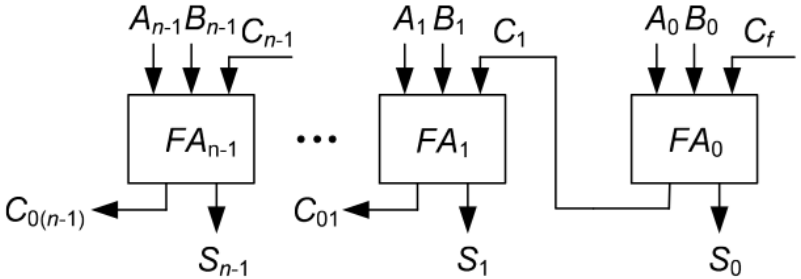


(*b*) Block diagram.

Ripple-carry Adder

Ripple-carry adder: Stages of full adders

- C_f : forced carry
- $C_0(n-1)$: overflow carry



$$S_i = A_i \oplus B_i \oplus C_i$$

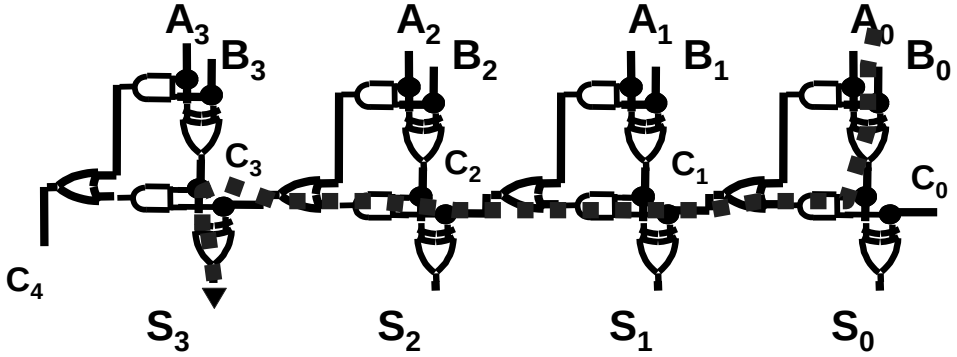
$$C_{0i} = A_iB_i + A_iC_i + B_iC_i$$

Time required:

- Time per full adder: 2 units
- Time for ripple-carry adder: 2n units

Carry Propagation & Delay

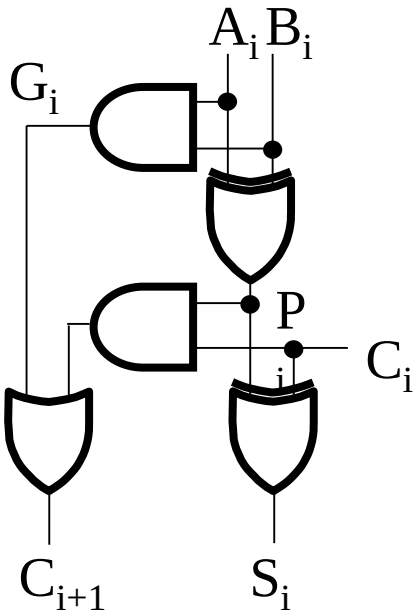
- One problem with the addition of binary numbers is the length of time to propagate the ripple carry from the least significant bit to the most significant bit.
- The gate-level propagation path for a 4-bit ripple carry adder of the last example:



- Note: The "long path" is from A_0 or B_0 though the circuit to S_3.

Carry Lookahead

- Given Stage i from a Full Adder, we know that there will be a carry generated when $A_i = B_i = "1"$, whether or not there is a carry-in.
- Alternately, there will be a carry propagated if the "half-sum" is "1" and a carry-in, C_i occurs.
- These two signal conditions are called *generate*, denoted as G_i, and *propagate*, denoted as P_i respectively and are identified in the circuit:



Carry Lookahead **(continued)**

- In the ripple carry adder:
  - Gi, Pi, and Si are local to each cell of the adder
  - Ci is also local each cell
- In the carry lookahead adder, in order to reduce the length of the carry chain, C_i is changed to a more global function spanning multiple cells
- Defining the equations for the Full Adder in term of the P_i and G_i:

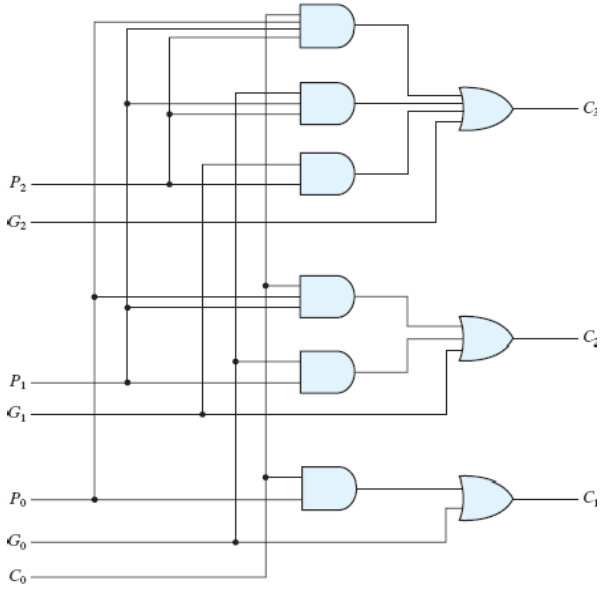
$$S_i = P_i \oplus C_i \qquad C_{i+1} = G_i + P_i C_i$$

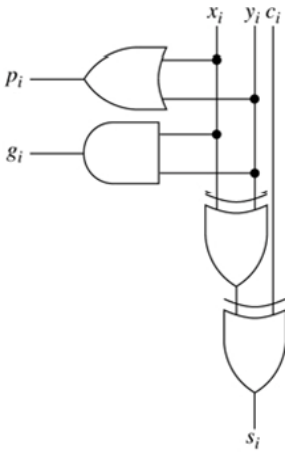
$$P_i = A_i \oplus B_i \qquad G_i = A_i B_i$$

Example - 4-bit Adder

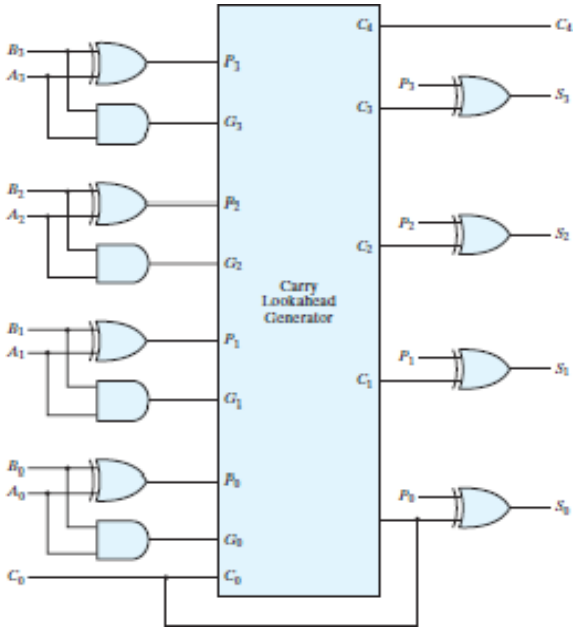
$$
\begin{aligned}
c_1 &= G_0 + c_0 P_0, \\
c_2 &= G_1 + G_0 P_1 + c_0 P_0 P_1, \\
c_3 &= G_2 + G_1 P_2 + G_0 P_1 P_2 + c_0 P_0 P_1 P_2, \\
c_4 &= G_3 + G_2 P_3 + G_1 P_2 P_3 + G_0 P_1 P_2 P_3 + c_0 P_0 P_1 P_2 P_3
\end{aligned}
$$

A carry lookahead generator.





A carry lookahead Adder



Thanks

Wish you all have a Good Exam