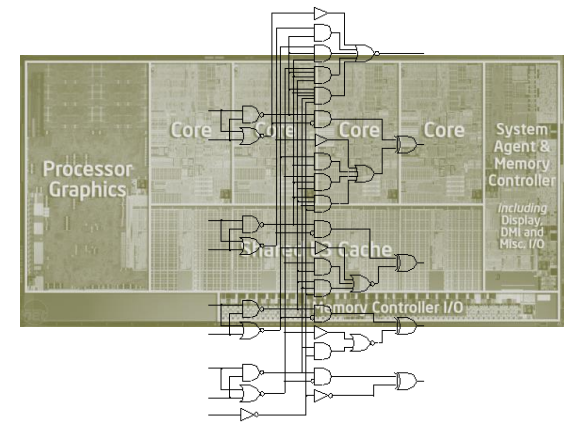




고려대학교
KOREA UNIVERSITY



COSE221, COMP211 Logic Design

Lecture 3. Combinational Logic #2

Prof. Taeweon Suh

Computer Science & Engineering

Korea University

Karnaugh Maps (K-Maps)

- When using Boolean algebra with axioms and theorems, you sometimes end up with a more complex equation instead of a simplified equation
- K-map is a graphical method of simplifying Boolean equations
 - It was invented by Maurice Karnaugh in 1953
 - K-map works well for problems up to 4 input variables

Gray Code

- Gray code is a binary numeral system where two successive values differ in only one bit
 - Patented by Frank Gray, a Bell Labs researcher in 1953
 - Adjacent entries differ only in a single variable**
 - Gray codes generalize to any number of bits

2-bit Gray code

A	B
0	0
0	1
1	0
1	1

00 → 01 → 11 → 10

3-bit Gray code

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

000 → 001 → 011 → 010 → 110 → 111 → 101 → 100

000 → 010 → 011 → 001 → 101 → 111 → 110 → 100

Karnaugh Maps (K-Maps)

- Arrange input combinations in **gray code**
- Circle 1's in adjacent squares
 - Each circle must span a power of 2 (i.e. 1, 2, 4) squares in each direction

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Y C	AB			
	00	01	11	10
0	1	0	0	0
1	1	0	0	0

$$Y = \overline{A}\overline{B}$$

Y C	AB			
	00	01	11	10
0	$\overline{A}\overline{B}\overline{C}$	$\overline{A}B\overline{C}$	$AB\overline{C}$	$A\overline{B}\overline{C}$
1	$\overline{A}\overline{B}C$	$\overline{A}BC$	ABC	$A\overline{B}C$

K-map Example

Truth Table

A	B	C	Y
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	0

K-Map

		AB			
		00	01	11	10
C	0	0	1	0	0
	1	0	1	0	1

$$Y = \bar{A}B + A\bar{B}C$$

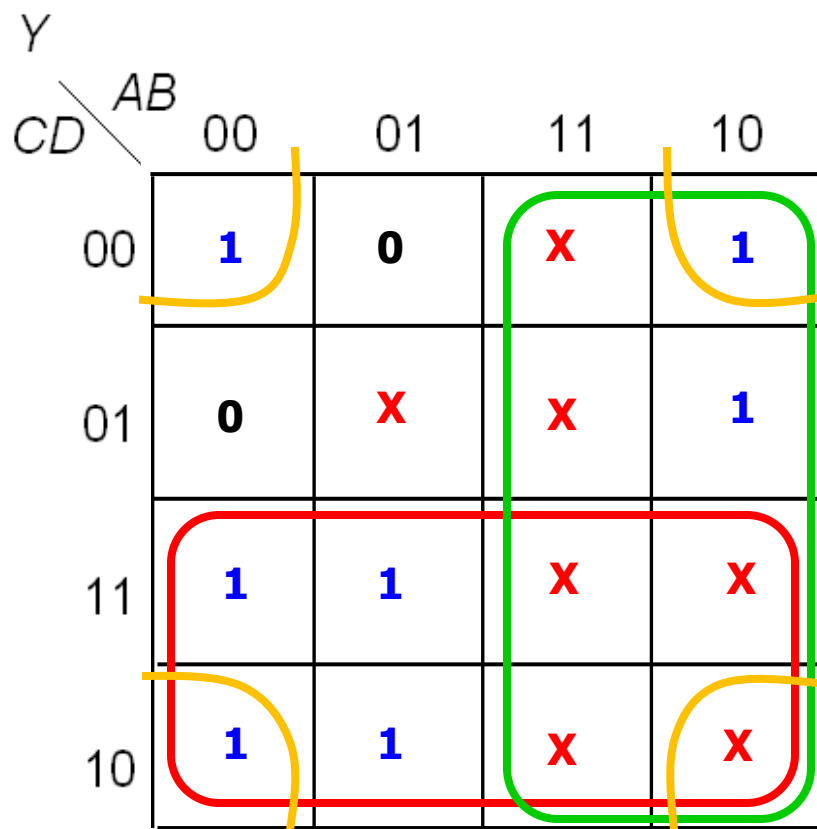
K-map Rules

- Each circle must span a power of 2 (i.e. 1, 2, 4) squares in each direction
 - Each circle must be as large as possible
- A circle may wrap around the edges of the K-map
- A one in a K-map may be circled multiple times
- A "don't care" (X) is circled only if it helps minimize the equation

Y CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	1

$$Y = \bar{A}\bar{C} + \bar{A}BD + A\bar{B}\bar{C} + \bar{B}\bar{D}$$

K-Maps with Don't Cares

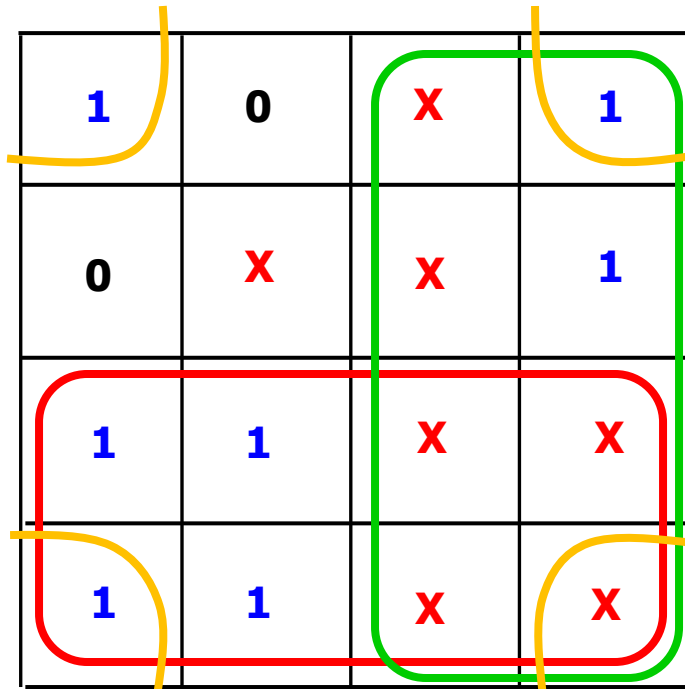


$$Y = C + A + \bar{B}\bar{D}$$

Prime Implicants

- **Prime implicant**

- Prime implicant is an implicant corresponding to the largest circle in a K-map
- It can not be combined with any other implicants to form a new implicant with fewer literals

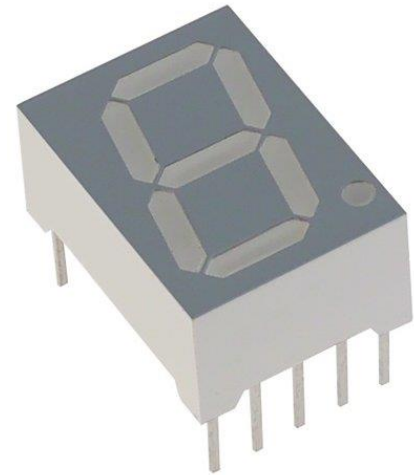
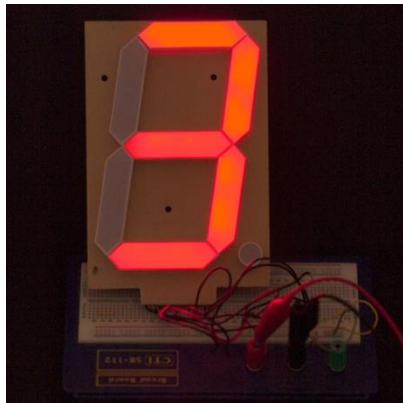


Prime Implicants

$$Y = C + A + \overline{B}\overline{D}$$

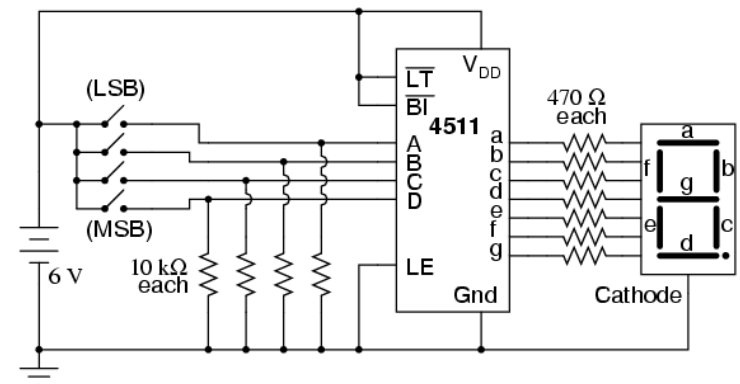
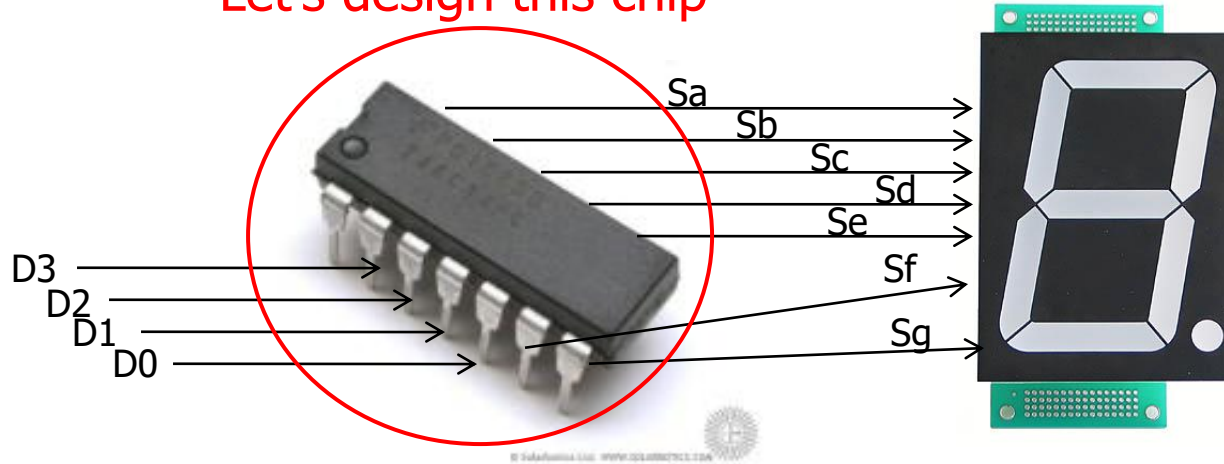
7 Segments

- Have you seen this?

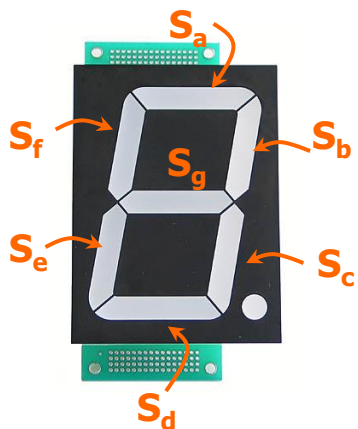


Digital Logic for 7 Segment

Let's design this chip



Truth Table for 7 Segment Logic



0 1 2 3 4 5 6 7 8 9

D3 D2 D1 D0	S _a	S _b	S _c	S _d	S _e	S _f	S _g
0000 (0)	1	1	1	1	1	1	0
0001 (1)	0	1	1	0	0	0	0
0010 (2)	1	1	0	1	1	0	1
0011 (3)	1	1	1	1	0	0	1
0100 (4)	0	1	1	0	0	1	1
0101 (5)	1	0	1	1	0	1	1
0110 (6)	1	0	1	1	1	1	1
0111 (7)	1	1	1	0	0	0	0
1000 (8)	1	1	1	1	1	1	1
1001 (9)	1	1	1	0	0	1	1
others	0	0	0	0	0	0	0

S_a Logic

D3 D2 D1 D0	S_a	S_b	S_c	S_d	S_e	S_f	S_g
0000 (0)	1	1	1	1	1	1	0
0001 (1)	0	1	1	0	0	0	0
0010 (2)	1	1	0	1	1	0	1
0011 (3)	1	1	1	1	0	0	1
0100 (4)	0	1	1	0	0	1	1
0101 (5)	1	0	1	1	0	1	1
0110 (6)	1	0	1	1	1	1	1
0111 (7)	1	1	1	0	0	0	0
1000 (8)	1	1	1	1	1	1	1
1001 (9)	1	1	1	0	0	1	1
others	0	0	0	0	0	0	0

S_a

D3D2 \ D1D0	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	0

$$S_a = \bar{D}3D1 + \bar{D}3D2D0 + D3\bar{D}2\bar{D}1 + \bar{D}2\bar{D}1\bar{D}0$$

S_b Logic

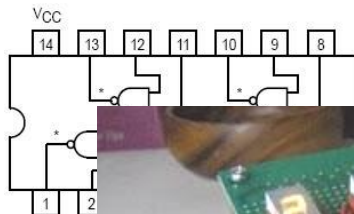
D3 D2 D1 D0	S_a	S_b	S_c	S_d	S_e	S_f	S_g
0000 (0)	1	1	1	1	1	1	0
0001 (1)	0	1	1	0	0	0	0
0010 (2)	1	1	0	1	1	0	1
0011 (3)	1	1	1	1	0	0	1
0100 (4)	0	1	1	0	0	1	1
0101 (5)	1	0	1	1	0	1	1
0110 (6)	1	0	1	1	1	1	1
0111 (7)	1	1	1	0	0	0	0
1000 (8)	1	1	1	1	1	1	1
1001 (9)	1	1	1	0	0	1	1
others	0	0	0	0	0	0	0

S_b

D3D2 \ D1D0	00	01	11	10
00	1	1	0	1
01	1	0	0	1
11	1	1	0	0
10	1	0	0	0

$$S_b = \bar{D}3\bar{D}2 + \bar{D}3D1D0 + \bar{D}3\bar{D}1\bar{D}0 + \bar{D}2\bar{D}1$$

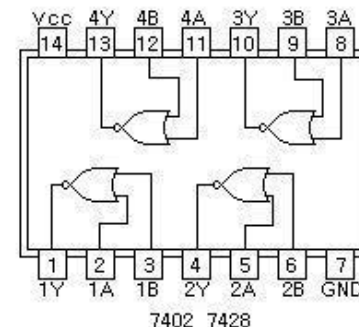
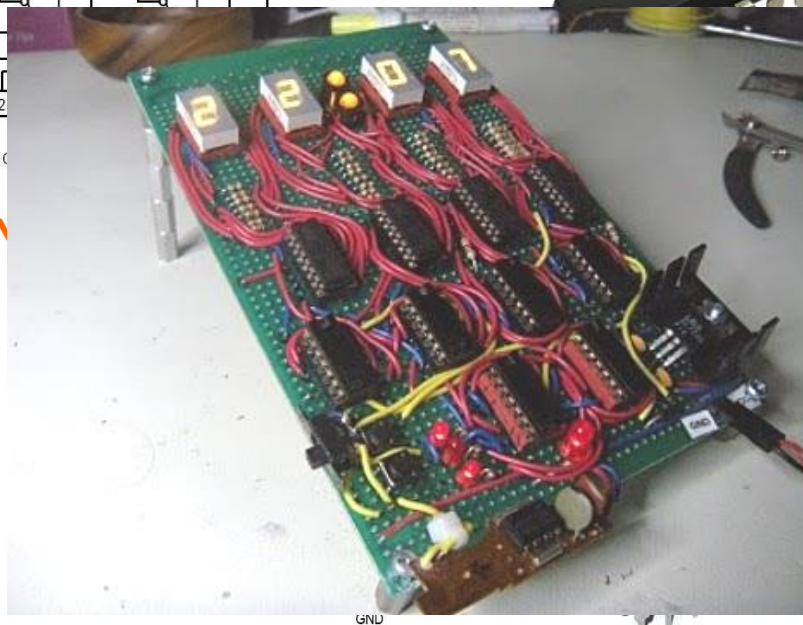
Real Logic Gates



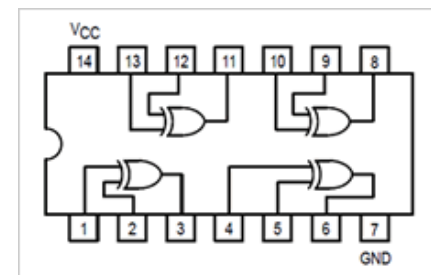
Quad 2-input **NAND** Gate (74ALS01P)



Quad 2-input **AND** Gate (74ALS08)



Quad 2-input **NOR** Gate (74LS02)



Quad 2-input **XOR** Gate (74LS136)

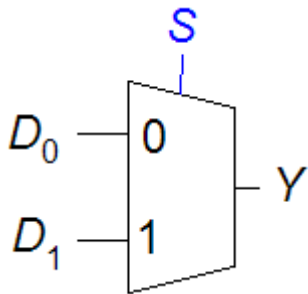
Combinational Building Blocks

- Combinational logic is often grouped into larger building blocks to build more complex systems
- We already studied some of building blocks
 - Priority logic, full adder (?), 7 segment display decoder
- 2 other very commonly used digital components
 - **Multiplexers**
 - **Decoders**

Multiplexer (Mux)

- Multiplexer selects an output from inputs based on the value of a **select** signal
- Multiplexer is called **mux** in short

Example: 2:1 Mux



S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Y S	D1D0			
	00	01	11	10
0	0	1	1	0
1	0	0	1	1

S	Y
0	D_0
1	D_1

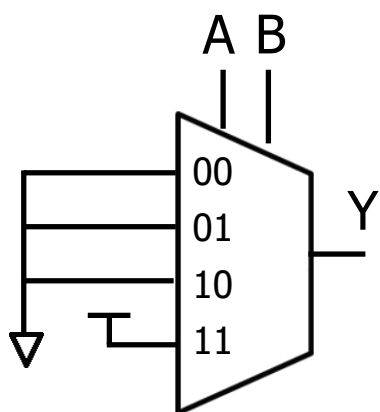
$$Y = \bar{S} D_0 + S D_1$$

Wider Muxes

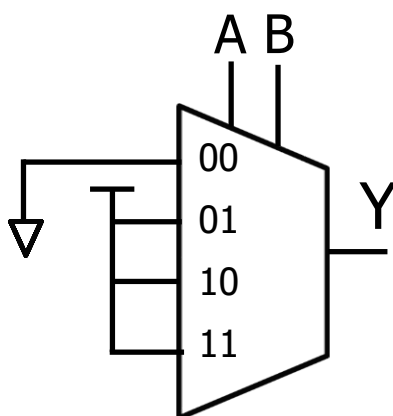
- 4:1 Mux
 - 4 inputs, 1 output, and 2 select signals
- 8:1 Mux
 - 8 inputs, 1 output, and 3 select signals
- 16:1 Mux
 - 16 inputs, 1 output, and 4 select signal
- N:1 Mux
 - N inputs, 1 output, and $\log_2 N$ select signals

Logic using Multiplexers

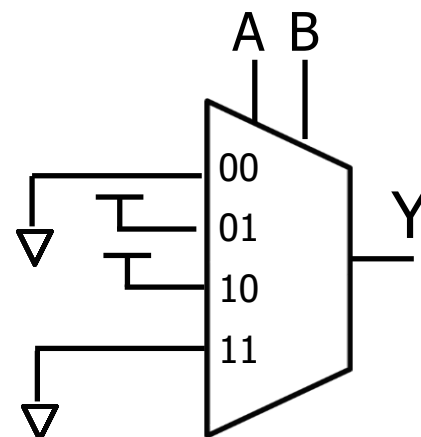
- Using the mux as a lookup table to perform logic functions



$$Y = AB$$



$$Y = A + B$$

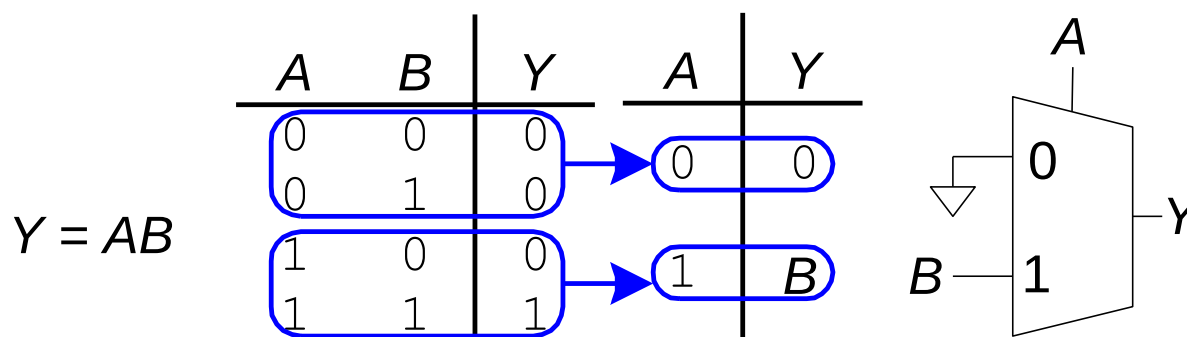


$$Y = \bar{A}B + A\bar{B} = A \oplus B$$

- 2^N -input multiplexer** can be programmed to perform **any N-input logic function** by applying 0's and 1's to the appropriate data inputs

Logic using Multiplexers

- With a little cleverness, we can cut the multiplexer size in half, using only a 2^{N-1} input multiplexer to perform any N-input logic function



- How to implement 2-input OR or XOR gates?

A Real Multiplexer Chip



MOTOROLA

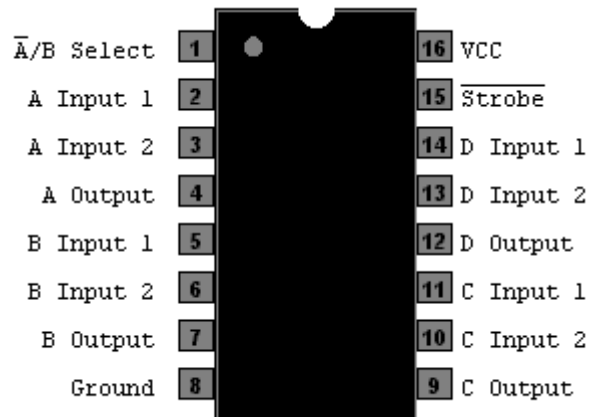
QUAD 2-INPUT MULTIPLEXER

The LSTTL/MSI SN54/74LS157 is a high speed Quad 2-Input Multiplexer. Four bits of data from two sources can be selected using the common Select and Enable inputs. The four buffered outputs present the selected data in the true (non-inverted) form. The LS157 can also be used to generate any four of the 16 different functions of two variables. The LS157 is fabricated with the Schottky barrier diode process for high speed and is completely compatible with all Motorola TTL families.

- Schottky Process for High Speed
- Multifunction Capability
- Non-Inverting Outputs
- Input Clamp Diodes Limit High Speed Termination Effects
- Special Circuitry Ensures Glitch-Free Multiplexing

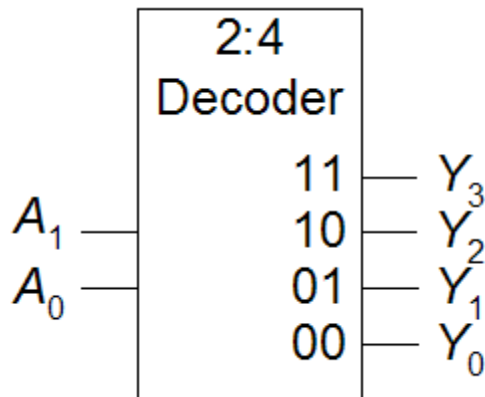
SN54/74LS157

QUAD 2-INPUT MULTIPLEXER
LOW POWER SCHOTTKY



Decoders

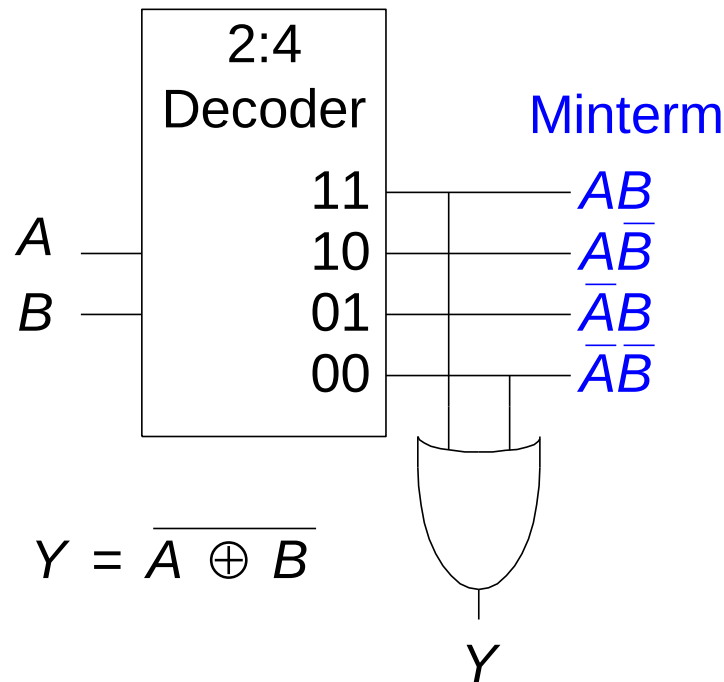
- Decoder asserts only one of outputs depending on the input combination
 - N inputs, 2^N outputs
 - **One-hot** because only one output is "hot" (HIGH) at a given time



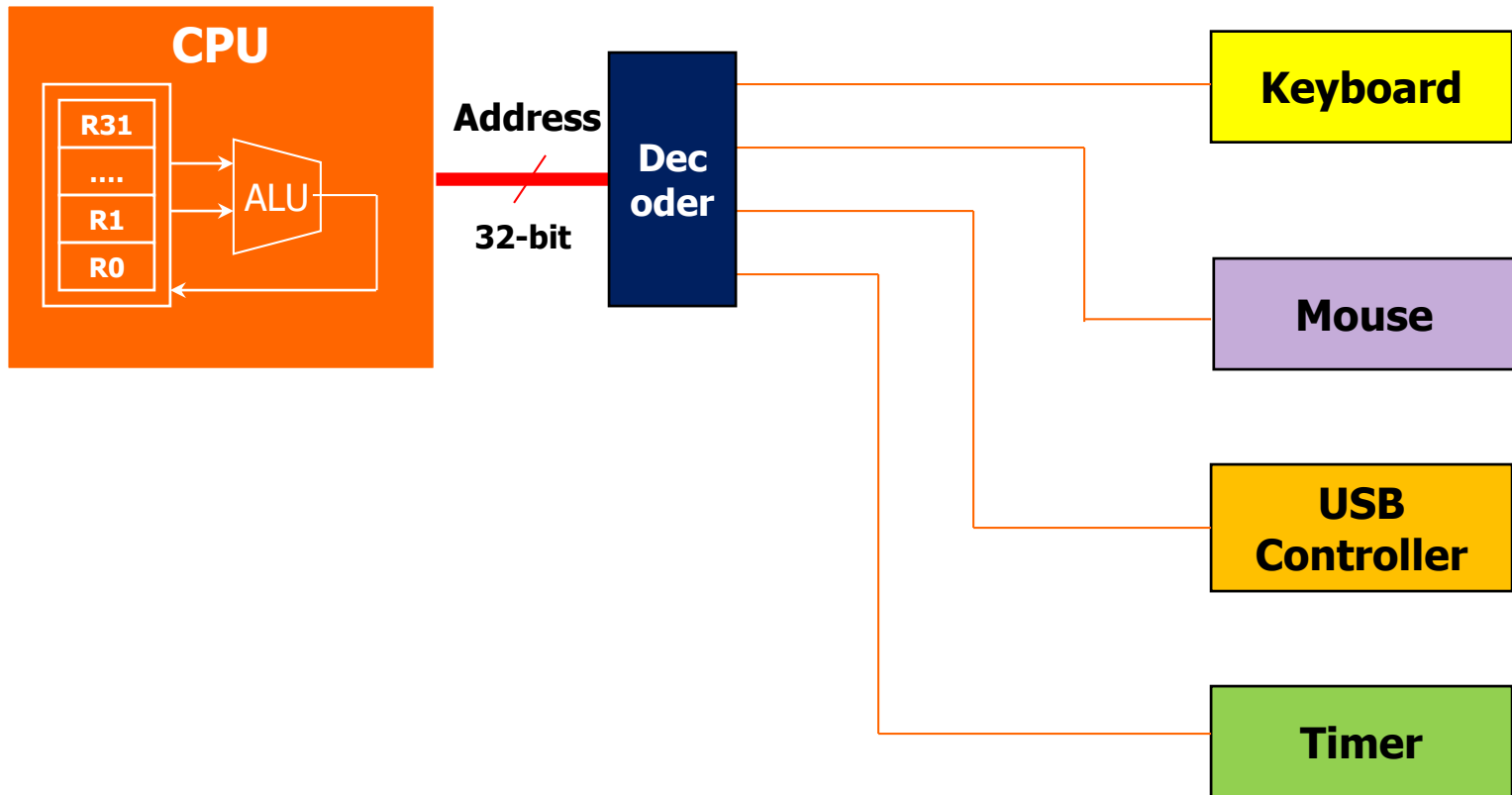
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Logic using Decoders

- Decoders can be combined with OR gates to build logic functions
 - SOP form (ORing minterms)

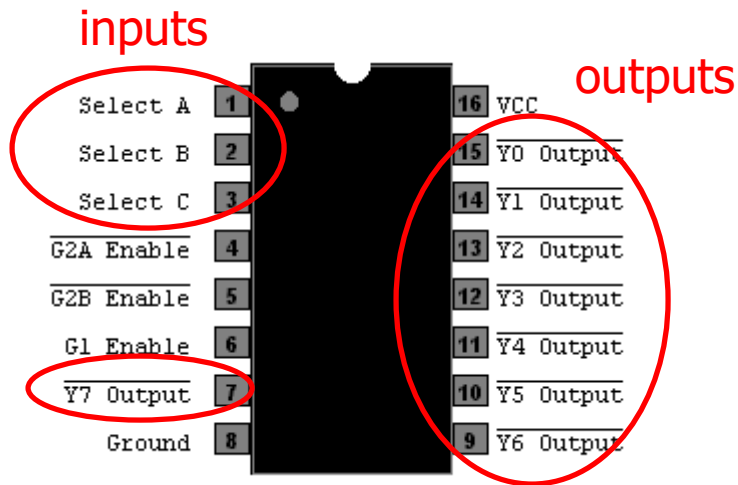


Decoder in Computer Systems



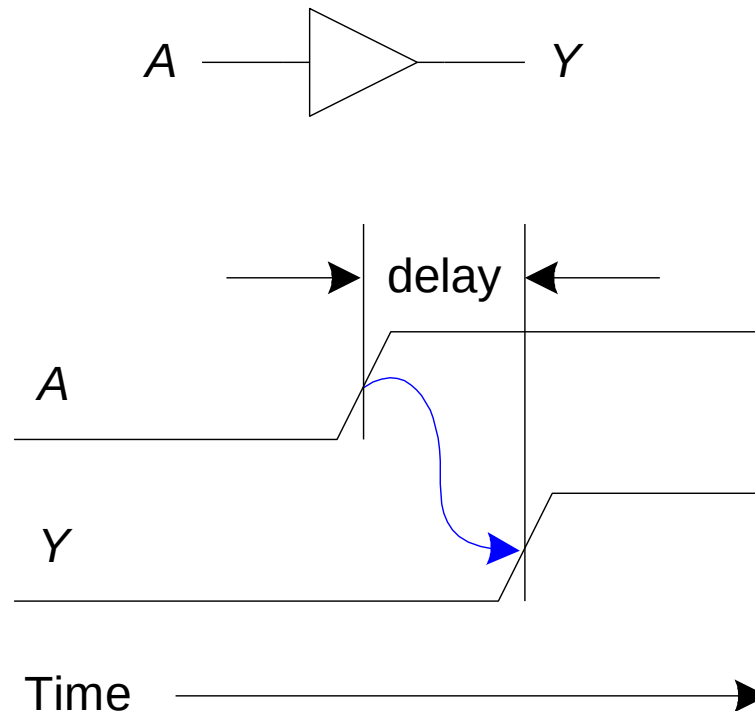
A Real Decoder Chip

- 74LS138
 - 3 inputs, 8 outputs



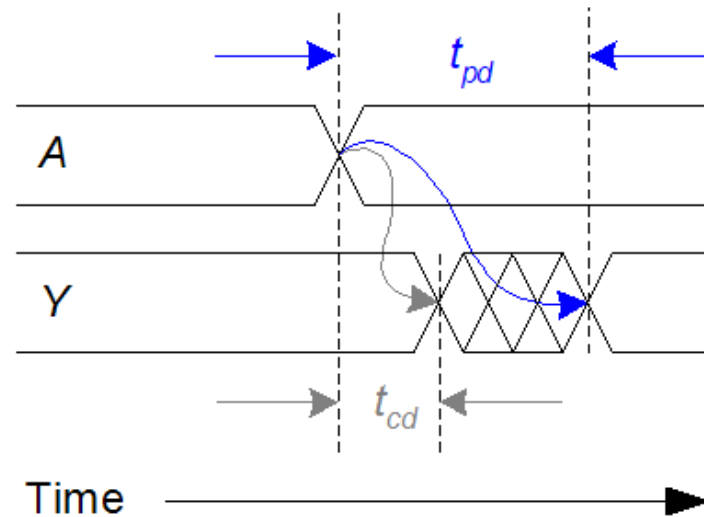
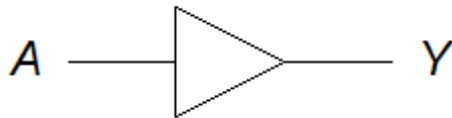
Timing

- There is always delay from input change to output change in real world
- One of the biggest challenges in circuit design is to **make the circuit fast**



Propagation & Contamination Delay

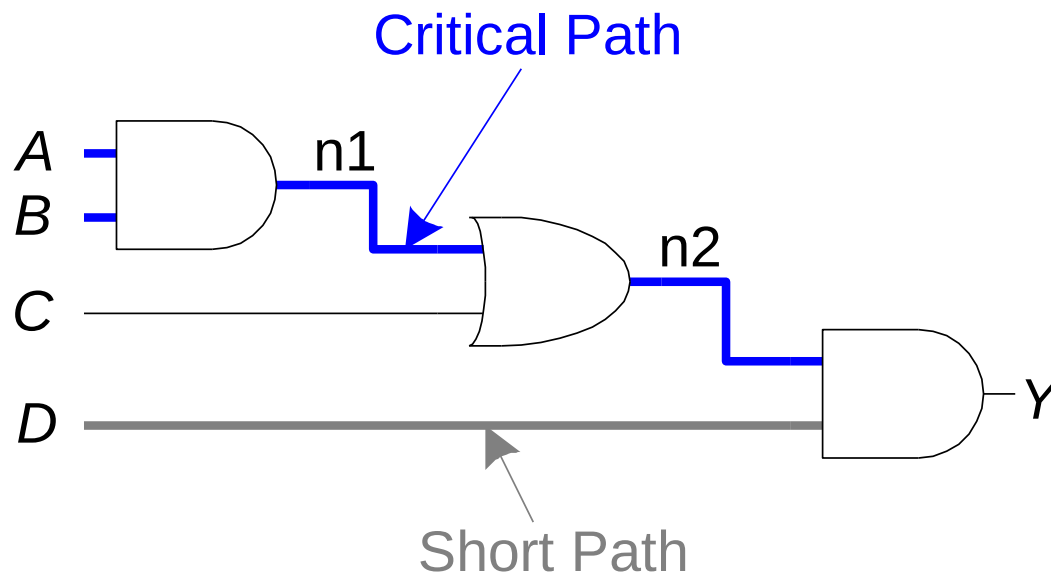
- Propagation delay
 - $t_{pd} = \text{max}$ delay from input to output
- Contamination delay
 - $t_{cd} = \text{min}$ delay from input to output



Propagation & Contamination Delay

- Delay is caused by
 - Transistor capacitance and resistance in a circuit
 - Interconnection capacitance and resistance
- Reasons why t_{pd} and t_{cd} may be different
 - Different rising and falling delays
 - Multiple inputs and outputs, some of which are faster than others
 - Circuits speeds are different depending on temperature
 - Circuit slows down when hot
 - Circuit speeds up when cold

Critical and Short Paths



Critical (Longest) Path: $t_{pd} = 2t_{pd_AND} + t_{pd_OR}$

Short Path: $t_{cd} = t_{cd_AND}$

Glitches

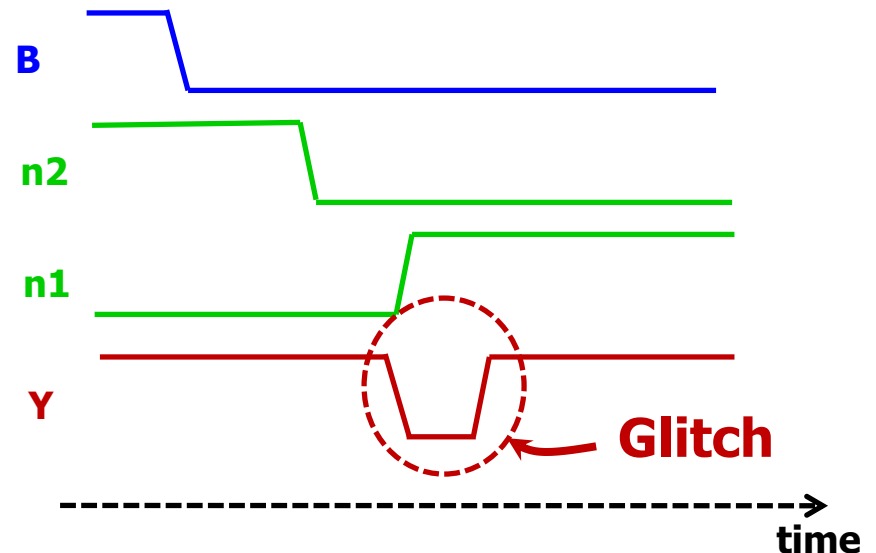
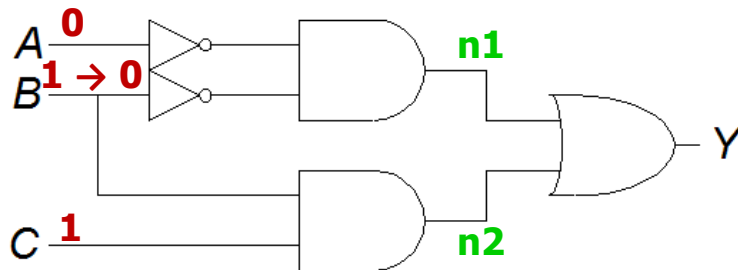
- So far, we have discussed the case where a single input transition causes a single output transition
- However, it is possible that a **single** input transition can cause **multiple** output transitions
 - These are called **glitches**

Glitch Example

- Initially, $(A, B, C) = (0, 1, 1)$
- What happens when B changes from 1 to 0?

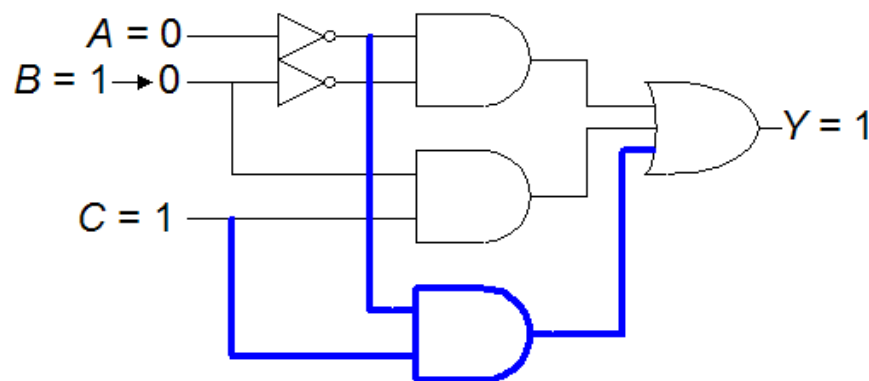
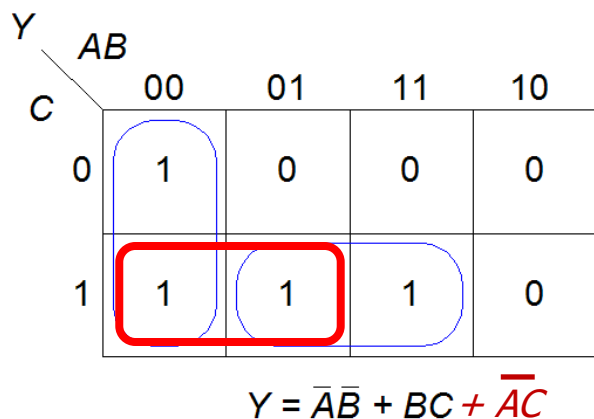
Y C	AB			
	00	01	11	10
0	1	0	0	0
1	1	1	1	0

$$Y = \bar{A}\bar{B} + BC$$



How to Eliminate Glitch?

- A glitch can occur when a change in a single variable crosses the boundary between 2 prime implicants in a K-map
 - We can remove the glitch by adding redundant implicants to cover these boundaries
 - Is it the consensus theorem?



Glitches

- Glitch removal comes at the cost of extra hardware
- Simultaneous transitions on multiple variables can also cause glitches
 - These glitches can not be fixed by adding extra hardware
 - The vast majority of interesting systems have simultaneous (or near-simultaneous) transitions on multiple variables
 - So, glitches are a fact of life in most circuits
- The point of discussing glitches is not to eliminate them all, but to **be aware that they exist**
 - It is especially important when looking at timing diagrams