UNITED STATES AIR FORCE ACADEMY

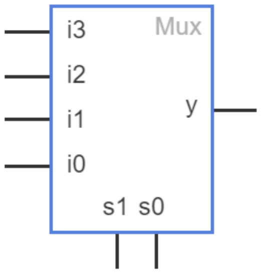
# ECE 281

# Lesson 9 Notes

**Objectives:**

- Given a circuit with multiplexers or decoders, describe its logic using a truth table or Boolean equation(s)
- Demonstrate the ability to use multiplexers and decoders as building blocks to design more complex combinational circuit diagrams
- Understand and recognize the VHDL idioms required to implement muxes and decoders
- Understand the causes and impacts of delays in combinational circuitry
- Understand how to read a basic timing diagram
- Be able to identify the short path and critical path in a combinational circuit
- Be able to correctly identify glitches in a circuit output

**Multiplexer:** a combinational circuit that passes one of multiple data inputs to a single output, selecting which one base on additional control inputs



| s1 s0 | y |
|---|---|

**You may also see muxes drawn using the following symbology:**

**Example #1:** In your zyBooks reading, they always provided fairly straight forward examples, where the select lines simply controlled the traffic flow based on the combination of select lines. Additionally, all of the zyBooks examples limited the number of inputs to the number of select lines. In other words a 4:1 mux had 2 select lines, an 8:1 mux had 3 select lines**,** etc... However, you could also do more complex designs that essentially treat the inputs to the mux as additional inputs to the logic circuit. For instance, represent the following logic equation using both an 8:1 mux and a 4:1 mux:

**Y=BC + A'B'C' + BC'**

**8:1 Mux**

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | |

**4:1 Mux**

| B | C | Y |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | |
| 1 | 0 | |
| 1 | 1 | |

## VHDL Code to Implement a Mux:

```
library IEEE; -- These lines are similar to a #include in C
use IEEE.STD_LOGIC_1164.ALL;

entity mux_4to1 is
  Port ( SEL : in STD_LOGIC_VECTOR (1 downto 0);-- select input
           D : in STD_LOGIC_VECTOR (3 downto 0); -- inputs
           Y : out STD_LOGIC);
end mux_4to1;

architecture Behavioral of mux_4to1 is

begin
  -- These lines are similar to a switch statement in C
  Y <= D(0) when (SEL = "00") else
        D(1) when (SEL = "01") else
        D(2) when (SEL = "10") else
        D(3) when (SEL = "11") else A(0);
end Behavioral;
```

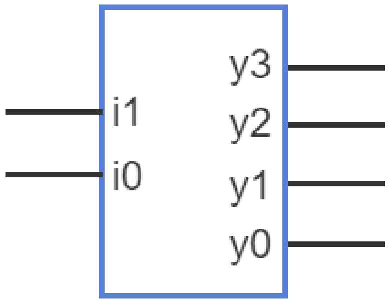
Only one of the cases will get executed based on the combination of select lines.

This is similar to the simple switch example from C below

```
   switch(grade) {
      case 'A' :
         printf("Excellent!\n" );
         break;
      case 'B' :
      case 'C' :
         printf("Well done\n" );
         break;
      case 'D' :
         printf("You passed\n" );
         break;
      case 'F' :
         printf("Better try again\n" );
         break;
      default :
         printf("Invalid grade\n" );
```

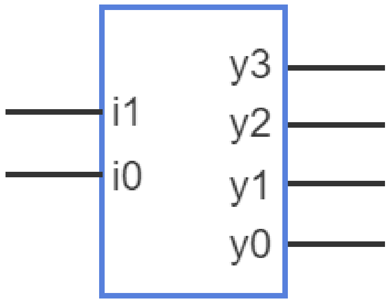
**Decoder:** is a combinational circuit that converts N inputs to a 1 on one of the $2^N$ outputs.

## One-hot



| i1 | i0 | y3 | y2 | y1 | y0 |
|---|---|---|---|---|---|

## One-Cold



| i1 | i0 | y3 | y2 | y1 | y0 |
|---|---|---|---|---|---|

## VHDL Code to Implement a Decoder:

```
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
entity decode_2to4 is
  Port ( A : in STD_LOGIC_VECTOR (1 downto 0); -- 2-bit input
          EN : in STD_LOGIC; -- enable input
           Y : out STD_LOGIC_VECTOR (3 downto 0)); -- 4-bit output
end decode_2to4;
architecture Behavioral of decode_2to4 is
begin
        Y(0) <= '1' when A = "00" else
                   '0';
        Y(1) <= '1' when A = "01" else
                   '0';
        Y(2) <= '1' when A = "10" else
                   '0';
        Y(3) <= '1' when A = "11" else
                   '0';
end Behavioral;
```

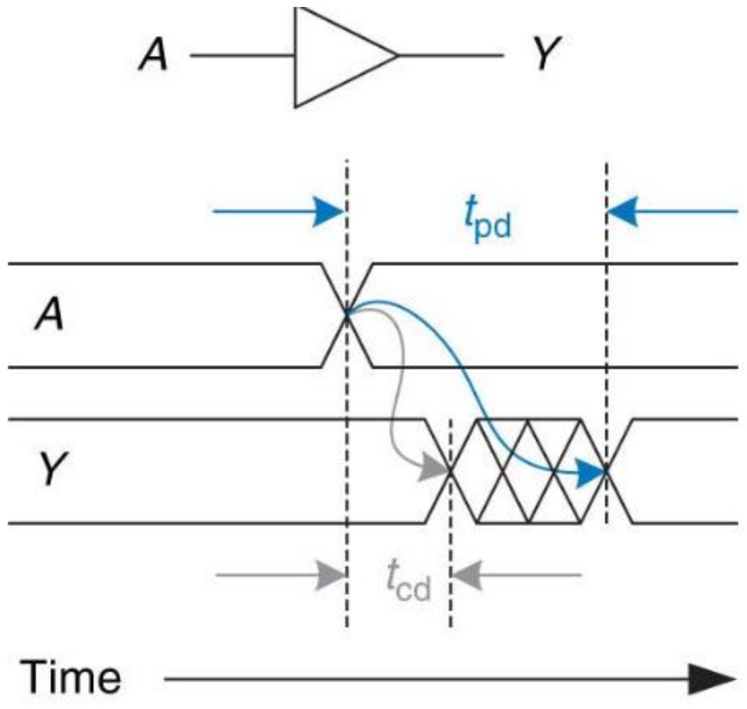
**Timing:** can be very important to the operation of your system. While we have treated everything to happen instantaneously up to this point, in actuality, components do have some amount of delay, and it varies from component to component. Many hardware vulnerabilities can actually be introduced due to timing delays or associated issues.

**Rising Edge** –

**Falling Edge** –

**Propagation Delay ($t_{pd}$)** –

**Contamination Delay ($t_{cd}$)** –

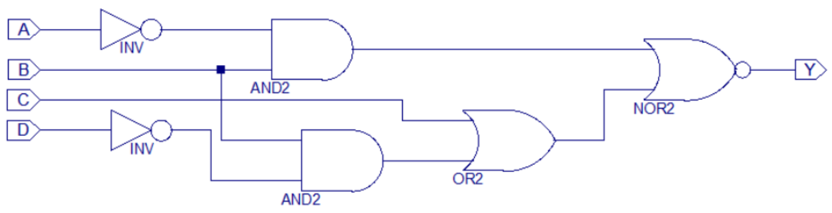


**Critical Path** –

**Short Path** –

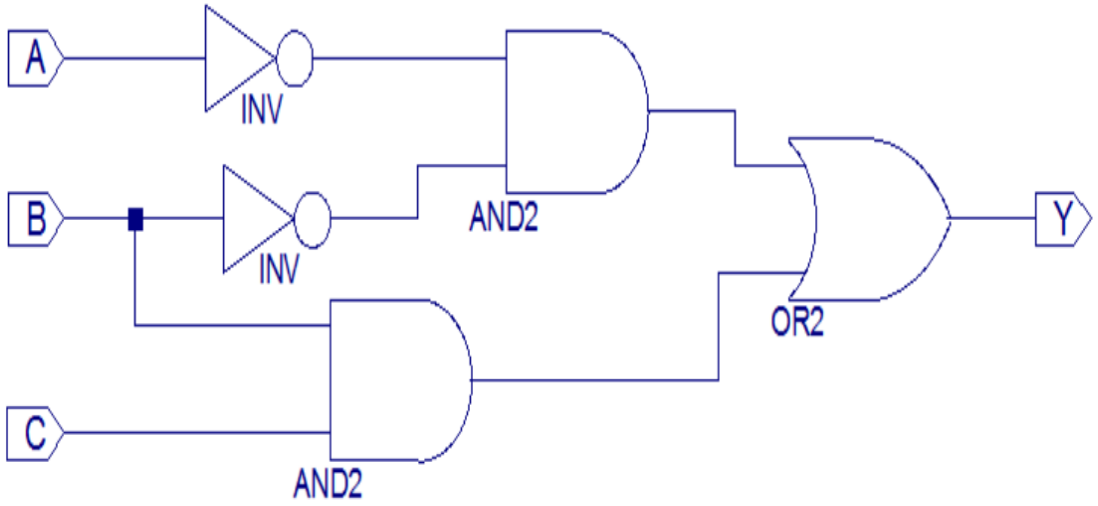
**Glitch** –

**Example 2)** Estimate the critical and short path for the following circuit.



| | $t_{pd}$ (ps) | $t_{cd}$ (ps) |
|---|---|---|
| NOT | 15 | 10 |
| AND | 30 | 25 |
| OR | 40 | 35 |
| NOR | 30 | 25 |

## Glitch Example:



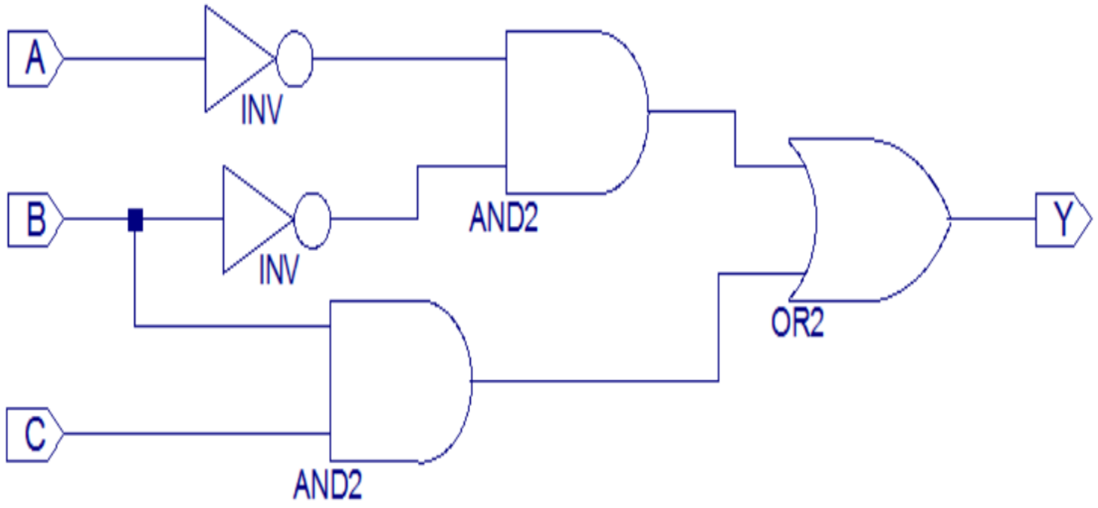
**B**

**BC**

$\bar{A}\bar{B}$

**Y**

## NAND-gate-only implementation



**Practice Problem –** Implement the following truth table using:

a.) 4:1 mux and inverters
b.) Only NOT, AND and OR gates
c.) Only NAND gates

| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |