Lab Location: GETC C043 & 45.

→ Using Karnaugh-Maps to minize equations graphically.

# Karnaugh Maps (K-Maps)

- Boolean expressions can be minimized by combining terms
- K-maps minimize equations graphically
- $PA + P\bar{A} = P$

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

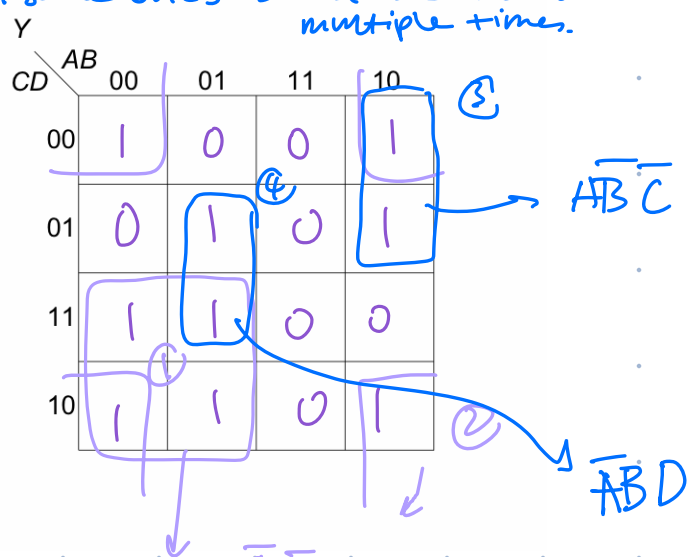
| Y | AB 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| C 0 | $\bar{A}\bar{B}\bar{C}$ | $\bar{A}B\bar{C}$ | $AB\bar{C}$ | $A\bar{B}\bar{C}$ |
| 1 | $\bar{A}\bar{B}C$ | $\bar{A}BC$ | $ABC$ | $A\bar{B}C$ |



# 4-Input K-Map

* circles may wrap around the edges.
* each circle must span a pow of 2
* some one's could be circled multiple times.

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



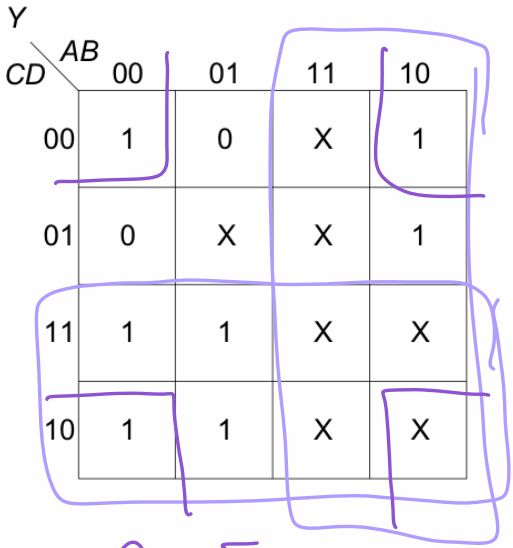

$Y = \bar{A}C + \bar{B}\bar{D} + A\bar{B}\bar{C} + \bar{A}\bar{B}D$

↑ final expression

# K-Maps with Don't Cares

# of blocks in circles?

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




① C ② A ③ $\bar{B}\bar{D}$

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



ELSEVIER

Digression:

Does $\bar{B}\bar{D} = \overline{BD}$ ?

| $\bar{B}$ | $\bar{D}$ | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

$\neq$

| B | D | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$\overline{BD} = \bar{B} + \bar{D}$

but this should be right?

1. 1+1

C ?

$\overline{BD} \neq \bar{B} + \bar{D}$

1 1 1 1

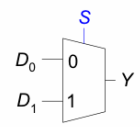
0 0 0 0

Multiplexer.

Selects one of the N inputs to connect to output

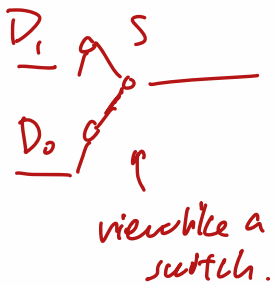
## Multiplexer (Mux)

- Selects between one of *N* inputs to connect to output
- $\log_2 N$-bit select input – control input
- **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 |

| S | Y |
|---|---|
| 0 | $D_0$ |
| 1 | $D_1$ |

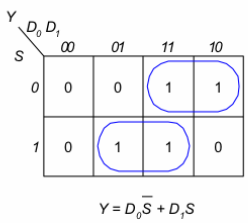




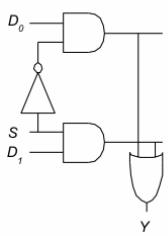

ELSEVIER

## Multiplexer Implementations

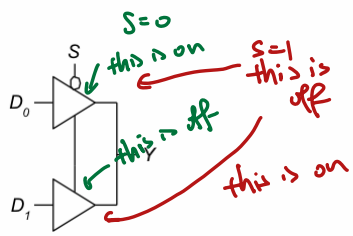
- **Logic gates**
  - Sum-of-products form




$Y = D_0\bar{S} + D_1S$




- **Tristates**
  - For an N-input mux, use N tristates
  - Turn on exactly one to select the appropriate input

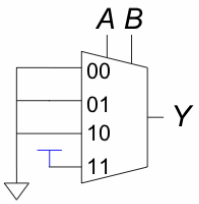


# Logic using Multiplexers

## Using the mux as a lookup table

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

$Y = AB$






ELSEVIER

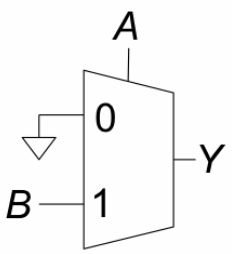
# Logic using Multiplexers

## Reducing the size of the mux

$Y = AB$

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

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




that's smart!

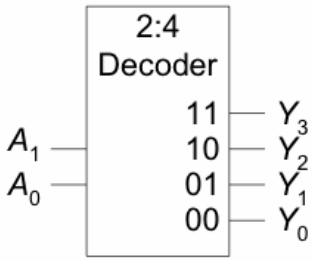


ELSEVIER

Decoder

# Decoders

- $N$ inputs, $2^N$ outputs
- **One-hot** outputs: only **one output HIGH** at once

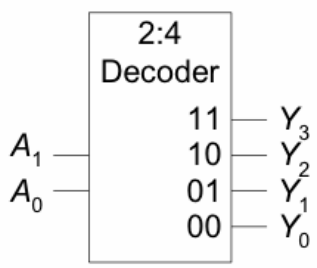



| $A_1$ | $A_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

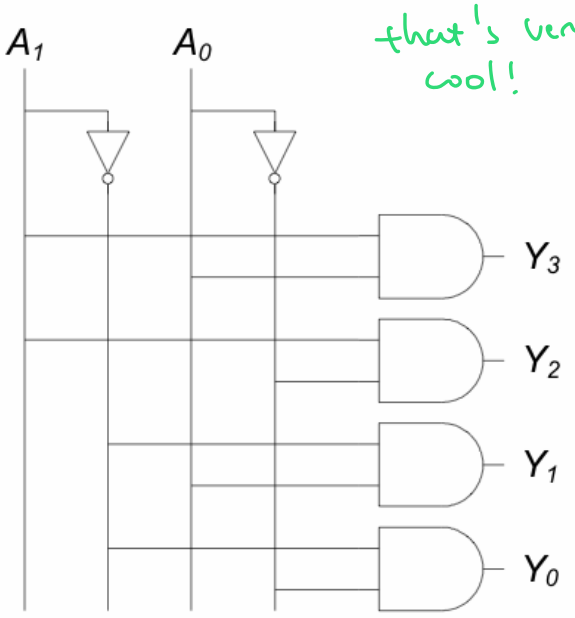




# Decoder Implementation




| $A_1$ | $A_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

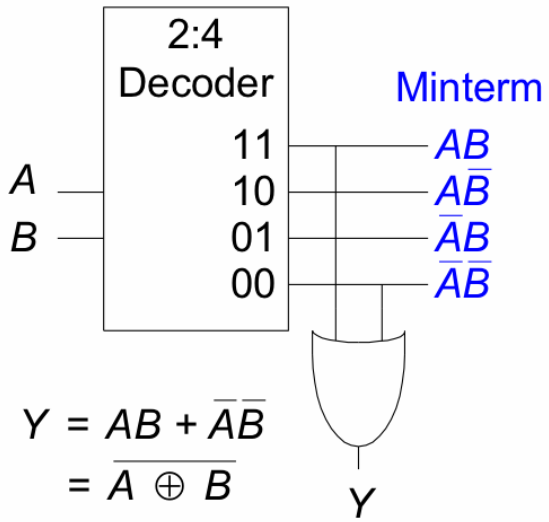



that's very cool!

# Logic Using Decoders

OR the minterms

XNOR




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

# HDL Introduction

↳ Hardware description language.

↳ HDL Design.

- Simulation: verify functionality of HDL
- Synthesis: transforms HDL code into a netlist describing the hardware.

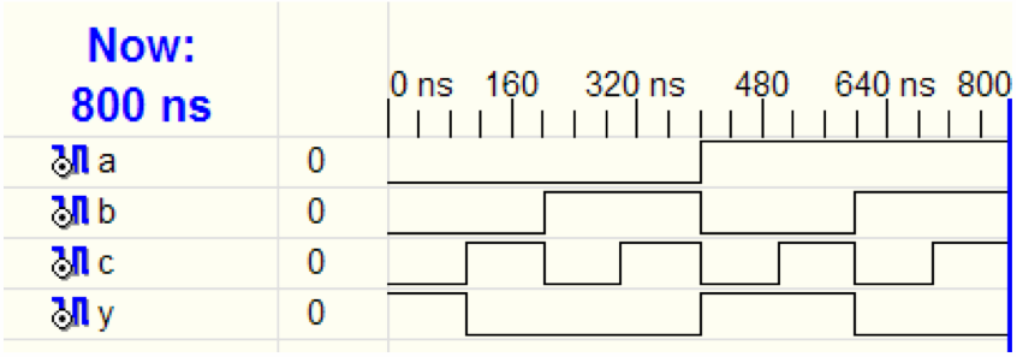
↳ Modules

- Behavior: describe what a module does
- Structural: describe how a module is built from larger modules.

## Behavioral Verilog Simulation

### Verilog:

```
module example(input  a, b, c,
               output y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```





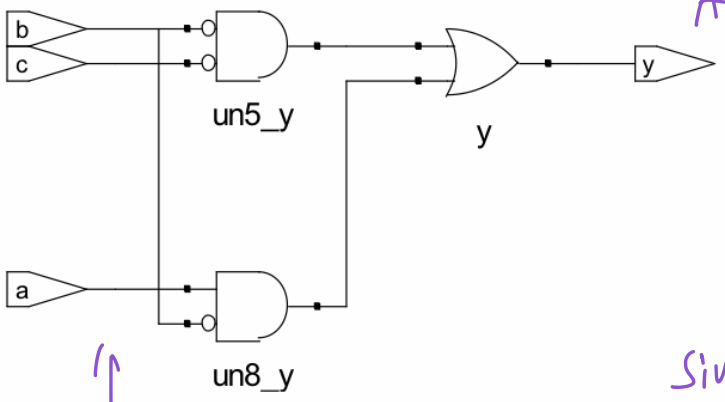

ELSEVIER

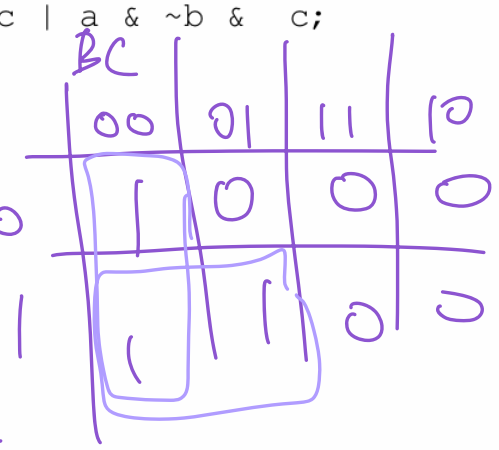
# Behavioral Verilog Synthesis

## Verilog:

```
module example(input  a, b, c,
               output y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```

## Synthesis:




simplified equation:

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

it automatically does optimization

# Verilog Syntax

- Case sensitive
  - `Example: reset` and `Reset` are not the same signal.
- No names that start with numbers
  - Example: `2mux` is an invalid name.
- Whitespace ignored
- Comments:
  - // single line comment
  - /* multiline
    comment */




# Structural Modeling - Hierarchy

```
module and3(input  a, b, c,
            output y);
  assign y = a & b & c;
endmodule


module inv(input  a,
           output y);
  assign y = ~a;
endmodule


module nand3(input  a, b, c
             output y);
  wire n1;                      // internal signal

  and3 andgate(a, b, c, n1);  // instance of and3
  inv  inverter(n1, y);      // instance of inverter
endmodule
```