Intro to Digital Design
Karnaugh Maps

Instructor: Chris Thachuk

Teaching Assistants:

Jiuyang Lyu   Nandini Talukdar

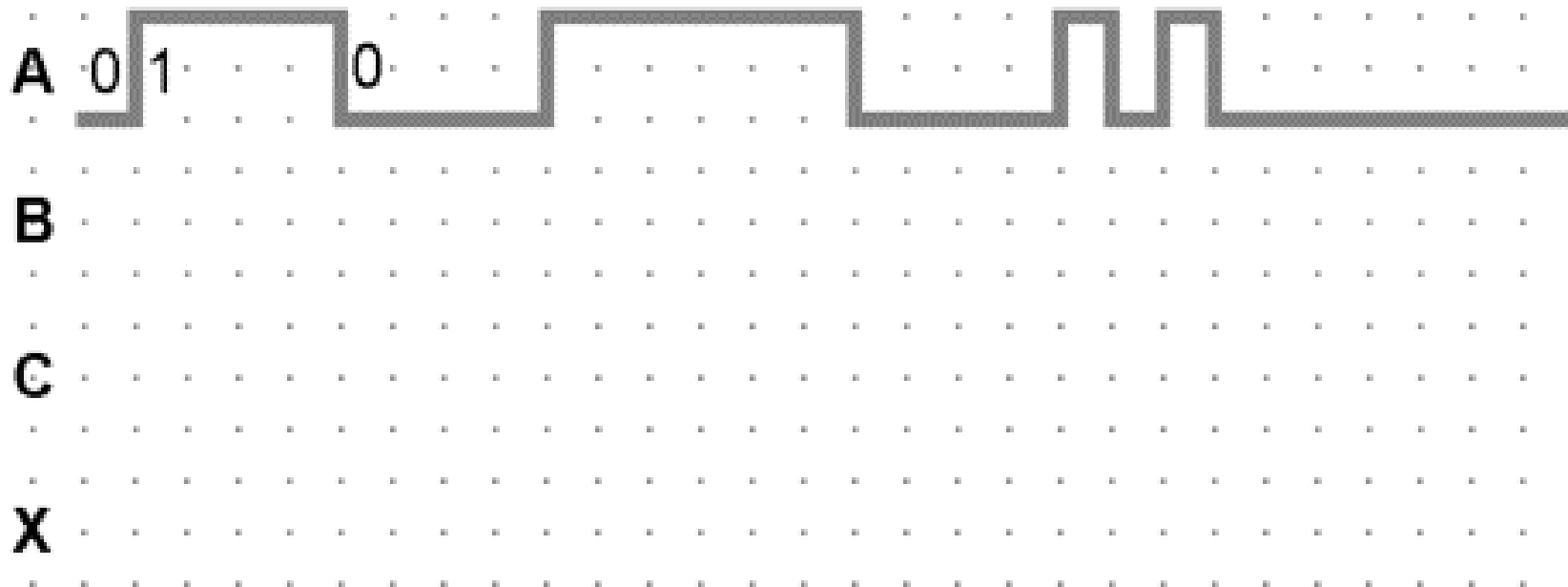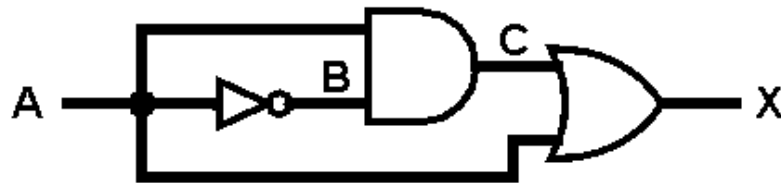Stephanie Osorio-Tristan   Wen Li

Question:

❖ Let the CL delays be 1 tick (NOT) and 3 ticks (AND, OR). How many ticks is the signal X high?

Relevant Course Information

- ❖ Lab 1 & 2 Demos due during your assigned demo slots
  - ▪ Don't forget to submit your lab materials *before* Wednesday at 2:30 pm, regardless of your demo time

- ❖ Lab 3 – Logic simplification in Verilog
  - ▪ Get practice with K-maps
  - ▪ Full credit for minimal logic

- ❖ Quiz #1 – 2/4 during lecture time

Lecture Outline

❖ **Karnaugh Maps (K-maps)**

❖ Design Examples

On and Off Sets

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

❖ *On Set* is the set of input patterns where the function is TRUE
  - Here on set = $\{\overline{A}\overline{B}C, \overline{A}BC, A\overline{B}\overline{C}, A\overline{B}C\}$

❖ *Off Set* is the set of input patterns where the function is FALSE
  - Here off set = $\{\overline{A}\overline{B}\overline{C}, \overline{A}B\overline{C}, AB\overline{C}, ABC\}$

❖ **Recall:** Use the On Set for *Sum of Products* (SoP) and the Off Set for *Product of Sums* (PoS)
 - Considered two-level Boolean expressions

Two-Level Simplification

❖ Using Sum of Products, "neighboring" input combinations simplify
- ▪ "Neighboring": inputs that differ by a single signal
- ▪ *The Uniting Theorem:* $A(\overline{B} + B) = A$
- ▪ *e.g.,* $AB + \overline{A}B = B$, $\overline{A}BC + \overline{A}B\overline{C} = \overline{A}B$

❖ **Goal:** Find neighboring subsets of the On Set to eliminate variables and simplify the expression

❖ **Idea:** Let's write out our Truth Table such that the neighbors become apparent!
- ▪ Need a Karnaugh map for *EACH* output

Karnaugh Maps

❖ A K-map is a method of representing a truth table that helps visualize adjacencies in $\leq$ 4 dimensions

- For more dimensions, computer-based methods are needed

1) Split inputs into 2 *evenly-sized* groups

- One group will have an extra if an odd # of inputs

2) Write out all combinations of each group on each axis Group of n inputs → 2^n combinations

- Successive combinations change only 1 input (Gray code)

2 Inputs:

	B	0	1
A			
0			
1			
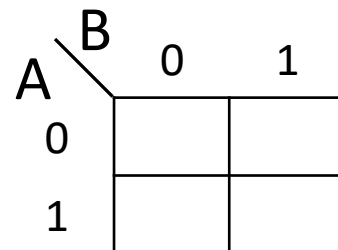
3 Inputs:
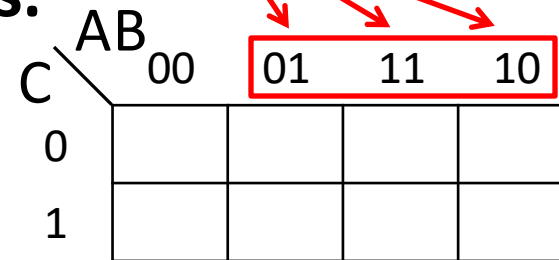
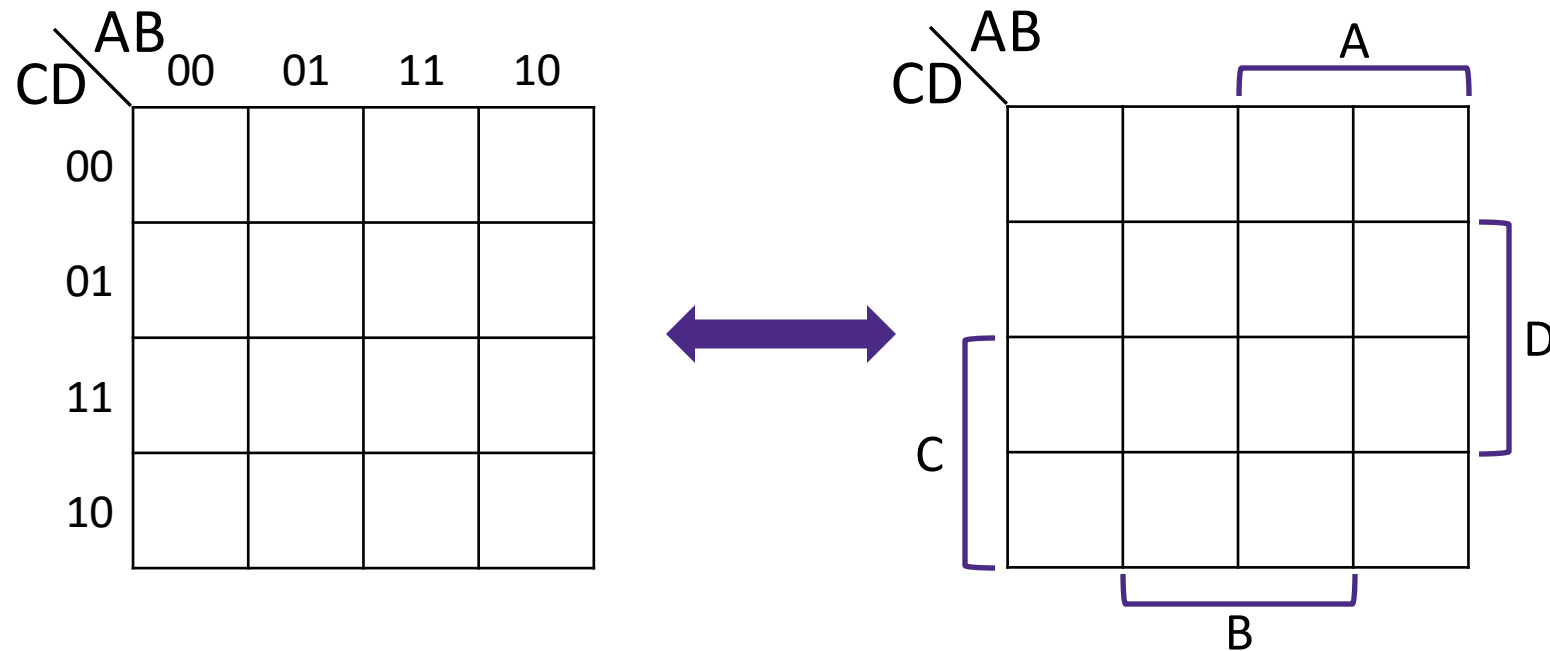| C \ AB	00	01	11	10
0				
1				

Karnaugh Maps

❖ Also see visualization with brackets for "asserted" simplifications:

4 Inputs:

K-map Example: Majority Circuit

❖ Filling in a Karnaugh map:

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

AB

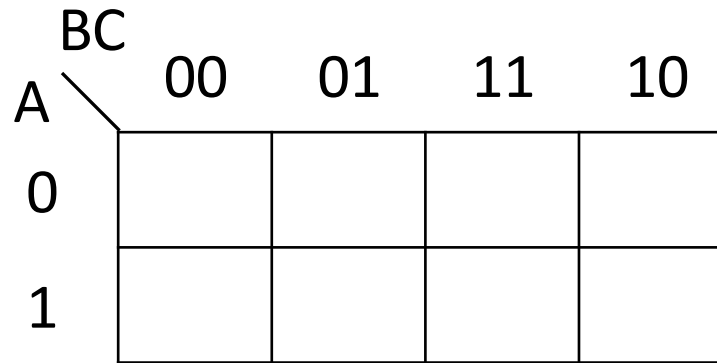| C	00	01	11	10
0				
1				

❖ Each row of truth table corresponds to ONE cell of Karnaugh map

❖ Note the jump when you go from input 011 to 100
(*most mistakes made here*)

K-map Example: Majority Circuit

❖ Filling in alternate Karnaugh map:

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

BC

A   00   01   11   10

0

1

❖ Each row of truth table corresponds to ONE cell of Karnaugh map

❖ Note the jump when you go from input 001 to 010 and 101 to 110 (*most mistakes made here*)

K-map Simplification

❖ Group neighboring 1's so all are accounted for:

- Each group of neighbors becomes a product term in the output

❖ $F = \textcolor{red}{BC + AB + AC}$

Single cell can be part of many groups

❖ Larger groups become smaller terms

- The single 1 in top row $\rightarrow AB\bar{C}$
- Vertical group of two 1's $\rightarrow AB$
- If entire lower row was 1's $\rightarrow C$

11
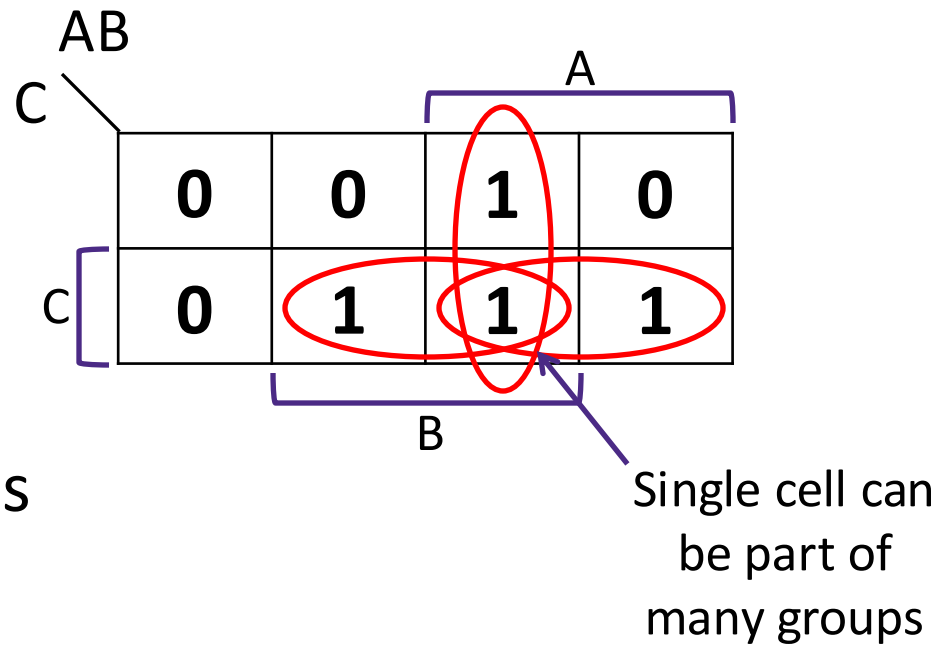
General K-map Rules

- ❖ Only group in powers of 2
  - ■ Grouping should be of size $2^i \times 2^j$
  - ■ Applies for both directions
- ❖ Wraps around in all directions
  - ■ "Corners" case is extreme example
- ❖ Always choose largest groupings possible
  - ■ Avoid single cells whenever possible

- ❖ $F = BD + \overline{B}\,\overline{D} + ACD$

1) NOT a valid group
2) IS a valid group
3) IS a valid group
4) "Corners" case
5) 1 of 2 good choices here

K-Map Example

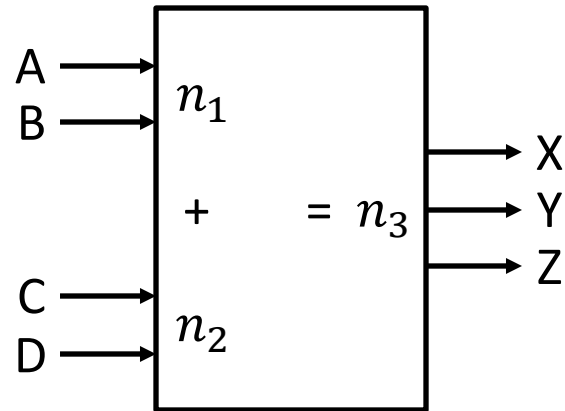❖ $F = \overline{A}D + BD + \overline{B}C + A\overline{B}D$
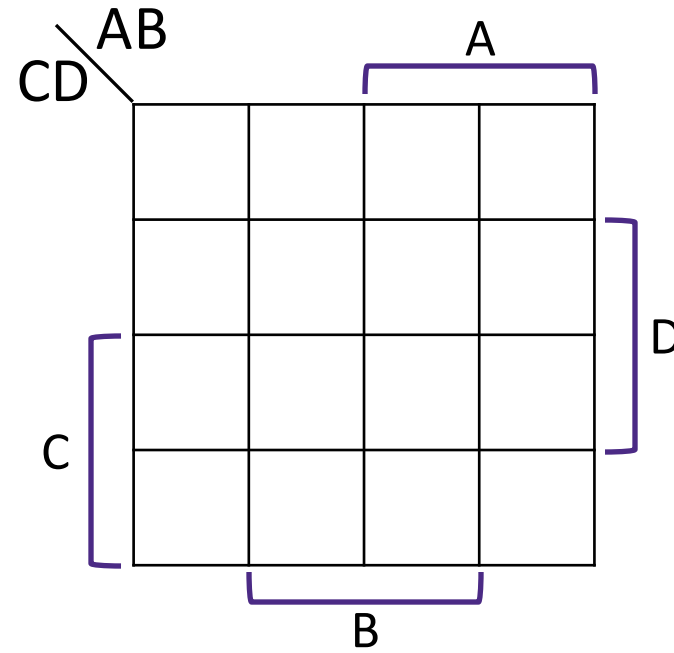
Lecture Outline

- ❖ Karnaugh Maps (K-maps)
- ❖ **Design Examples**

Design Example: 2-bit Adder

❖ Block Diagram and Truth Table:

A
B
n_1

+   = n_3 → X
→ Y
→ Z

C
D
n_2

A	B	C	D	X	Y	Z
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Design Example: 2-bit Adder

K-map for X

CD \ AB		A	
0	0	0	0
0	0	1	0
0	1	1	1
0	0	1	1

(D on right, C on left, B on bottom)

K-map for Y

CD \ AB		A	
0	0	1	1
0	1	0	1
1	0	1	0
1	1	0	0

(D on right, C on left, B on bottom)

K-map for Z

CD \ AB		A	
0	1	1	0
1	0	0	1
1	0	0	1
0	1	1	0

(D on right, C on left, B on bottom)
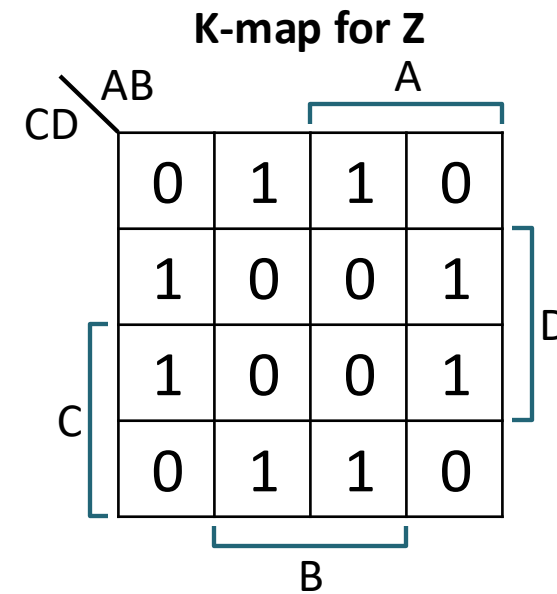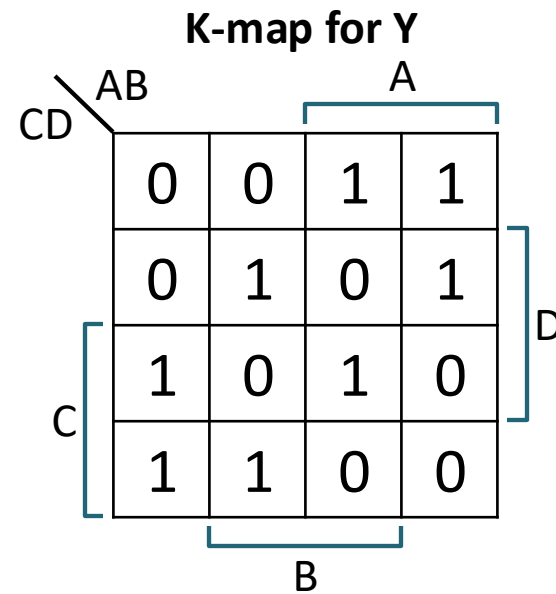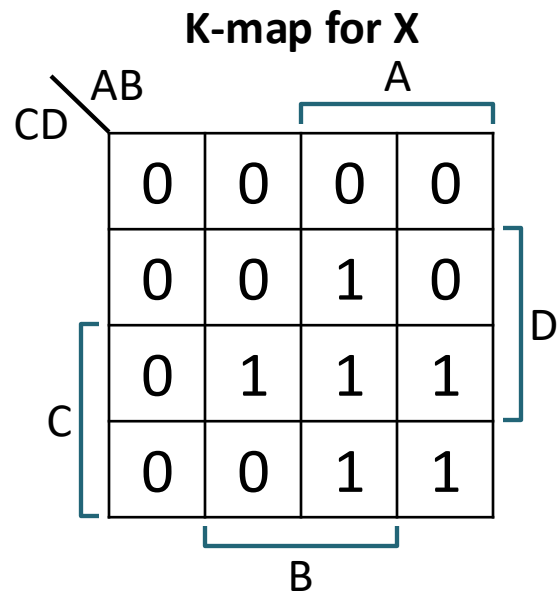
$X =$

$Y =$

$Z =$

Technology Break

Don't Cares

❖ Use symbol 'X' to mean it can be either a 0 or 1
  ▪ Make choice to simplify final expression

		A	
0	0	X	0
1	1	X	1
1	1	0	0
0	X	0	0

Let all $X = 0$:

$F =$

		A	
0	0	X	0
1	1	X	1
1	1	0	0
0	X	0	0

Let all $X = 1$:

$F =$

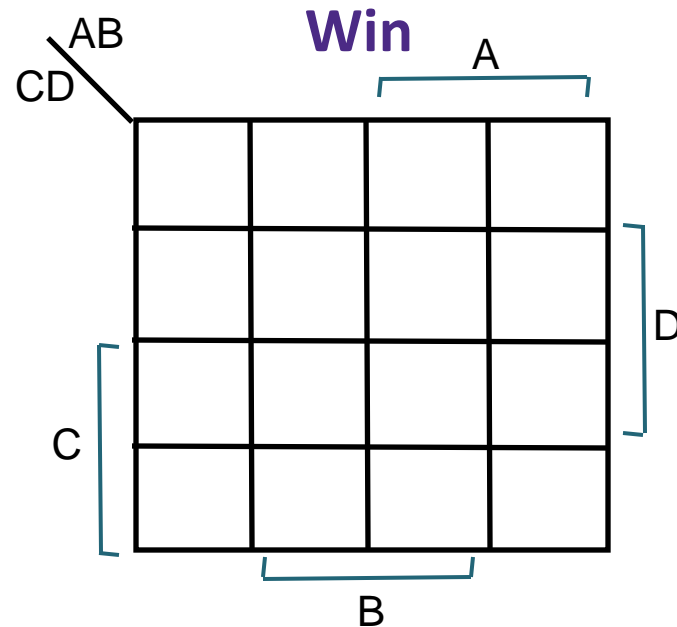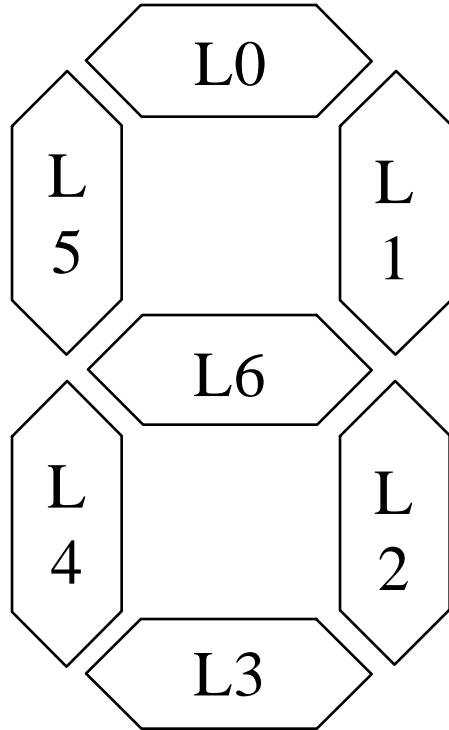|		A	
0	0	X	0
1	1	X	1
1	1	0	0
0	X	0	0

Choose wisely:

$F =$

Design Example: Rock-Paper-Scissors

❖ Rock (00), Paper (01), Scissors (10)
for two players P0 and P1

❖ **Output:** Win = Winner's ID (0/1)

Tie = 1 if Tie, 0 else

P1		P0			
A	B	C	D	Win	Tie
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Win

Case Study: Seven-Segment Display

❖ Chip to drive digital display

B3	B2	B1	B0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0

Case Study: Seven-Segment Display
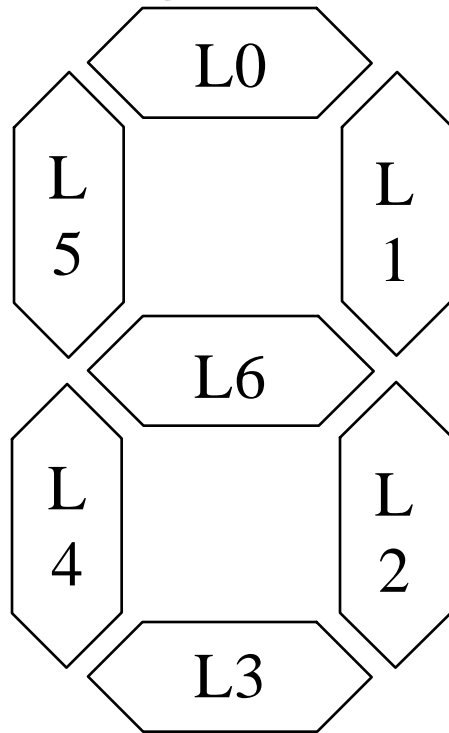
B3	B2	B1	B0	Val	L0	L1	L2	L3	L4	L5	L6
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0	0	0	0
0	0	1	0	2	1	1	0	1	1	0	1
0	0	1	1	3	1	1	1	1	0	0	1
0	1	0	0	4	0	1	1	0	0	1	1
0	1	0	1	5	1	0	1	1	0	1	1
0	1	1	0	6	1	0	1	1	1	1	1
0	1	1	1	7	1	1	1	0	0	0	0
1	0	0	0	8	1	1	1	1	1	1	1
1	0	0	1	9	1	1	1	1	0	1	1
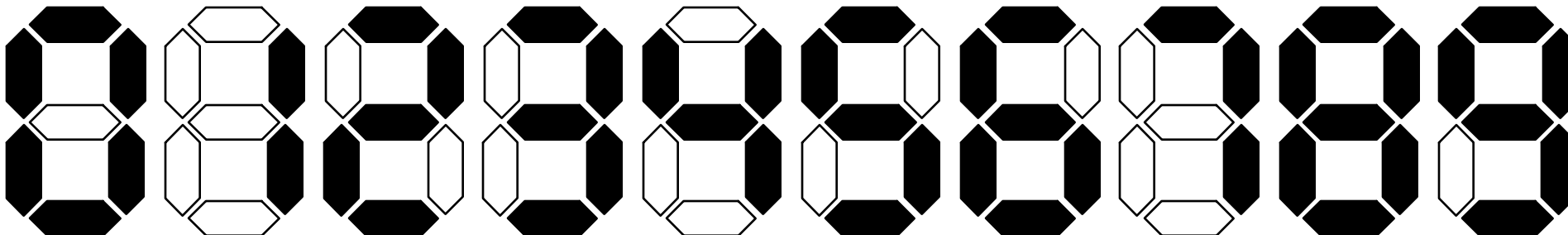
Case Study: Seven-Segment Display

❖ Implement L5:

B3	B2	B1	B0	L5
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
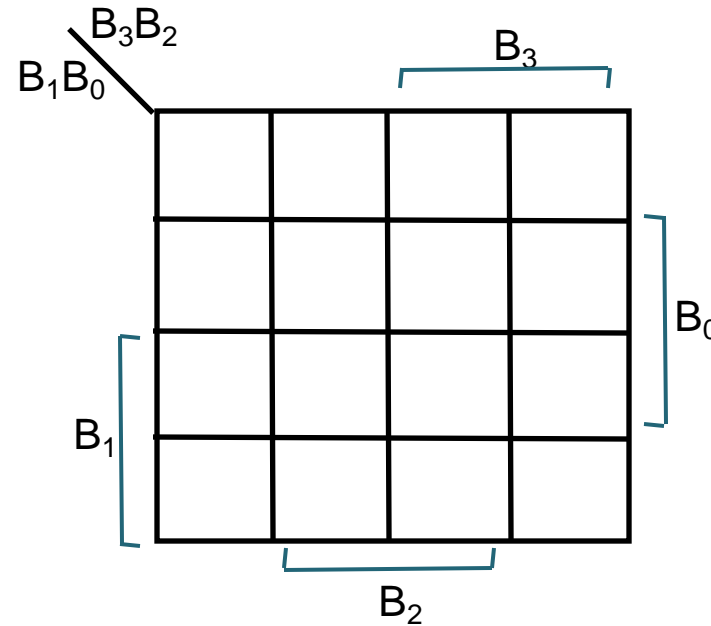
7-Seg Display in Verilog

```verilog
module seg7 (bcd, leds);
    input  logic [3:0] bcd;
    output logic [6:0] leds;

    always_comb
        case (bcd)
        // 3210                6543210
            4'b0000: leds = 7'b0111111;
            4'b0001: leds = 7'b0000110;
            4'b0010: leds = 7'b1011011;
            4'b0011: leds = 7'b1001111;
            4'b0100: leds = 7'b1100110;
            4'b0101: leds = 7'b1101101;
            4'b0110: leds = 7'b1111101;
            4'b0111: leds = 7'b0000111;
            4'b1000: leds = 7'b1111111;
            4'b1001: leds = 7'b1101111;
            default: leds = 7'bX;
        endcase
endmodule
```

Procedural Blocks

❖ `assign`: continuous assignment

- Used with wires

- *e.g.,* `assign F = ~((A & B) | (C & D));`

❖ `initial`: executes once at time zero

- Set initial values (generally simulation only!!!)

- Define testbench waveforms (and `monitor`)

- *e.g.,* `initial`
```
        for(i = 0; i < 8; i = i+1) begin
            {SEL, I, J} = i;   #10;
        end
```

Procedural Blocks

❖ `always`: loop to execute over and over again

- Block gets triggered by a *sensitivity list*

- Any object that is assigned a value in an `always` statement must be declared as a variable (`reg`/`logic`).

- Examples:
  - `always @ (a or b or c)` ↔ `always @ (a, b, c)`
  - `always @ (*)` implicitly contains all read signals within the block

❖ `always_comb`: special SystemVerilog for CL

- Similar to `always @(*)`, but generally more robust

- *Only for use with combinational logic!!!*

Verilog: Extend 7-Seg to Hex

❖ Show "A" on 0b1010 (ten) to "F" on 0b1111 (fifteen)
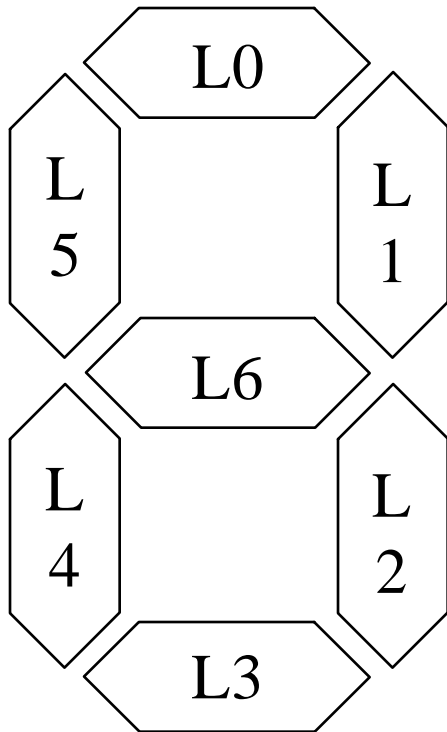
```verilog
module seg7 (bcd, leds);
    input  logic [3:0] bcd;
    output logic [6:0] leds;

    always_comb
        case (bcd)
        // 3210              6543210
        4'b0000: leds = 7'b0111111;
        4'b0001: leds = 7'b0000110;
        4'b0010: leds = 7'b1011011;
        4'b0011: leds = 7'b1001111;
        4'b0100: leds = 7'b1100110;
        4'b0101: leds = 7'b1101101;
        4'b0110: leds = 7'b1111101;
        4'b0111: leds = 7'b0000111;
        4'b1000: leds = 7'b1111111;
        4'b1001: leds = 7'b1101111;
        default: leds = 7'bX;
        endcase
endmodule
```

Circuit Implementation Techniques

❖ **Truth Tables** – "Black box" circuit description

❖ **Boolean Algebra** – Math form for optimization
  ■ *K-Maps* – Alternate simplification technique

❖ **Circuit Diagrams** – TTL Implementations

❖ **Verilog** – Simulation & mapping to FPGAs