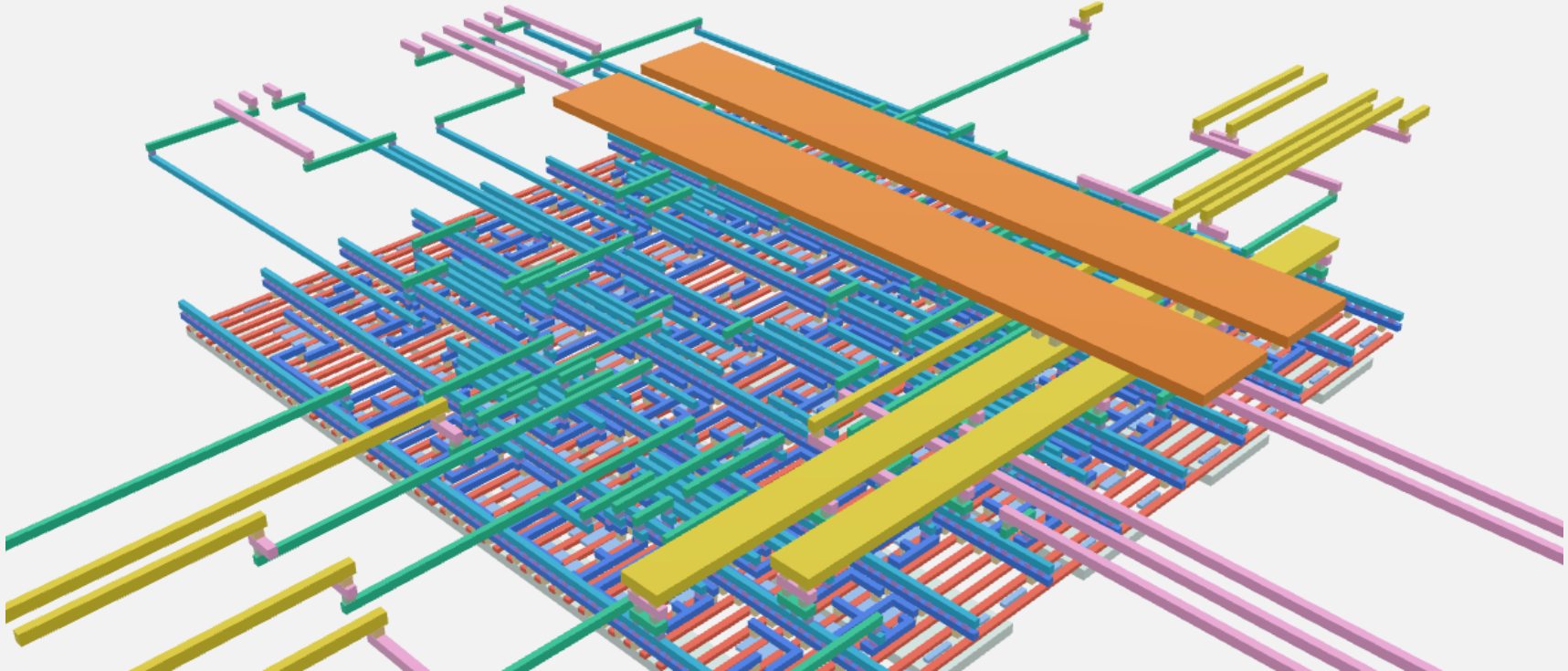
1.2
CSE140 – Digital Design

2026 Summer Session 1

Abarajithan G



First day info sheet

- **Career goals:** Buy a house

  Sell my soul to a Big Tech company

  I would like to find a healthy balance between stem and design.

- **Places grew up:** San Diego, San Francisco, Shanghai, Texas
- **Proud of:** Dancing, Piano, Design/Debug, Philosophy, Published, Baking
- **I appreciate when my instructor:**

  In addition to lecture materials, they also share their experiences and explain where in the industry/practice the concepts taught in class come in handy.

- **I dislike when my instructor:**

  Skip steps that might not be trivial

  rush through the content without pausing to let us process the information

  Are overly rigid in a close-minded way such as no late exceptions at all or no discussing topics with other students on homeworks because I feel like it makes us more like subjects than interactive students meant to learn from person-to-person.

Lecture 1: Feedback

- What people liked:

chill day

I liked the activities where we were to try to figure out the gates, as I found it helpful.

I liked the real-world chip examples.

video of guy making cpus at home

I like how examples and problems were worked out together on the board afterwards.

I like the in depth explanation of the topics and the pauses allowing us to really think about what is being shown.

The interactive slides are great, first time seeing this style!

- What could be improved:

I feel it's hard to catch up instructors talking speed, it's kind fast for me.

A bit more time during the activities would be helpful.

Less waiting time

The modeern ASIC flow was abstract and I did not feel like I understood each component.

I felt lost at the CMOS part.

maybe we can do more tracing examples with the transistors

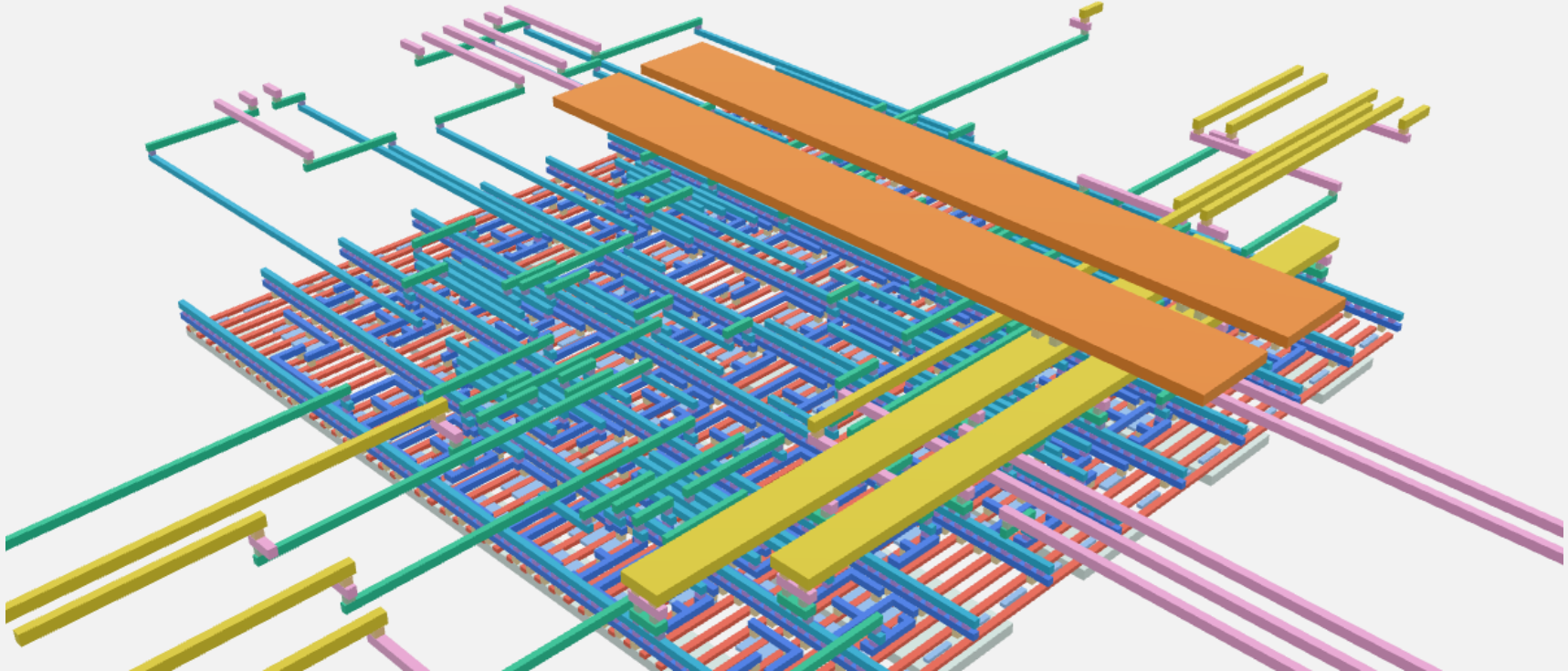
We discussed how in NAND and in NOR how were put either the AND and OR in series or parallel connections and how negative logic is more efficient I think a slide on that or an image on the slides could be helpful

There were a lot of acronyms that were being thrown around in the beginning and I was getting lost. I feel like it might be nice to have a reference to these somewhere.

Zybooks

- Digital Design by Frank Vahid, used by past courses has moved there.
- Interactive textbook with animations, randomized practice problems and instant feedback
- Gamifying learning experience
- Completely optional: if you would like to use ($69)
    1. Sign in or create an account at learn.zybooks.com
    2. Enter zyBook code: UCSDCSE140GnaneswaranSummer2026 (I don’t get anything from this)
    3. Subscribe
- I have also added two open-source books with examples and solutions to Canvas-Files-Books

Quick Recap

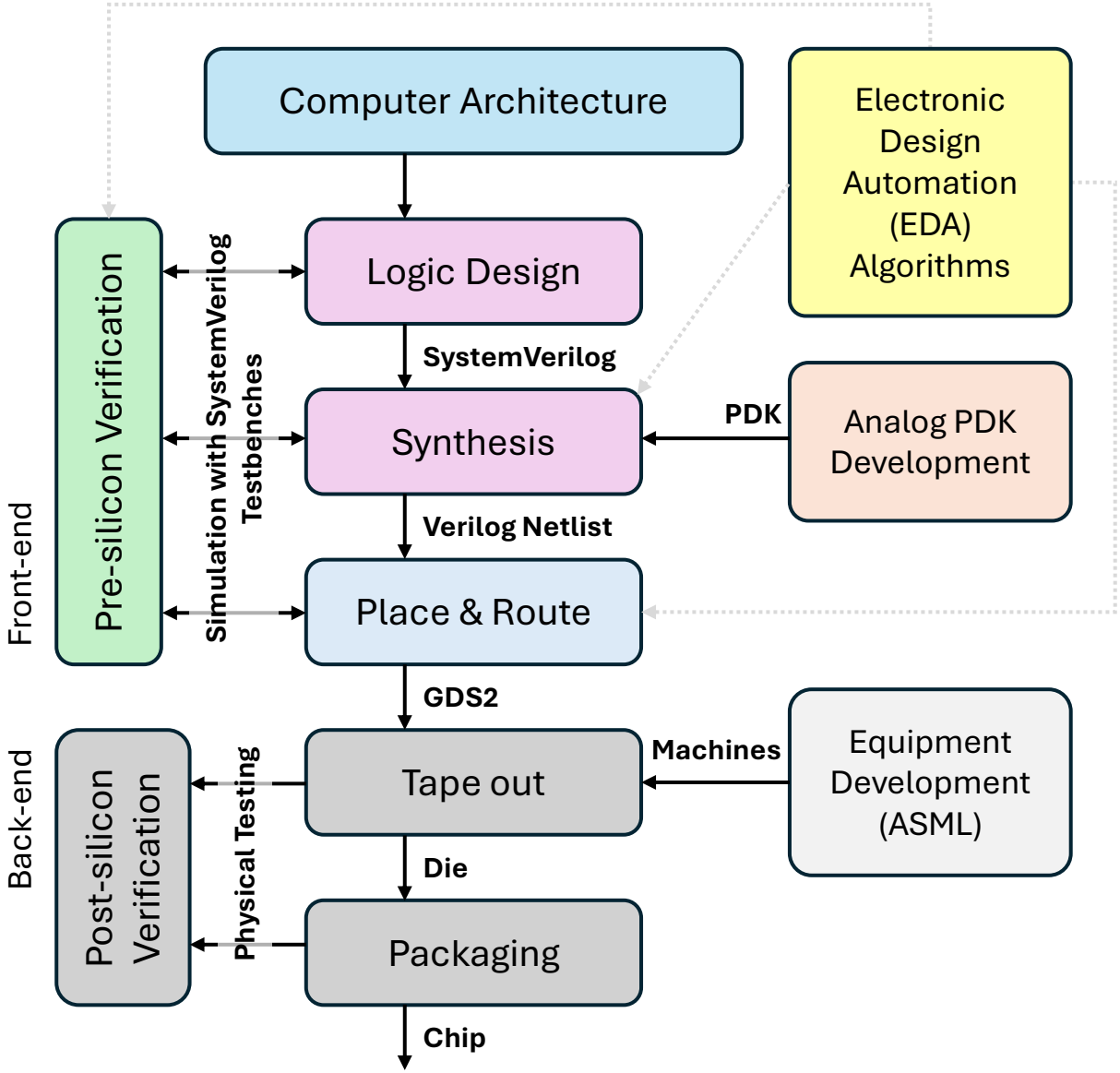


Daily reminder:

- Join Discord (QR code)
- Open FeedbackFruits
  Canvas → CSE140 → Assignments → Slides 1.2
- Attendance is mandatory
- Paper exams follow assignments. Don’t use AI
- Friday is the last day to drop



Modern ASIC Flow



Computer Architecture:
Full picture: “We are building a millennium”



What we learn in this course

Logic Design:
Designing subcomponent



Synthesis:
Selecting the tiny Lego blocks (standard cells = logic gates) needed to build it, and making a rough map



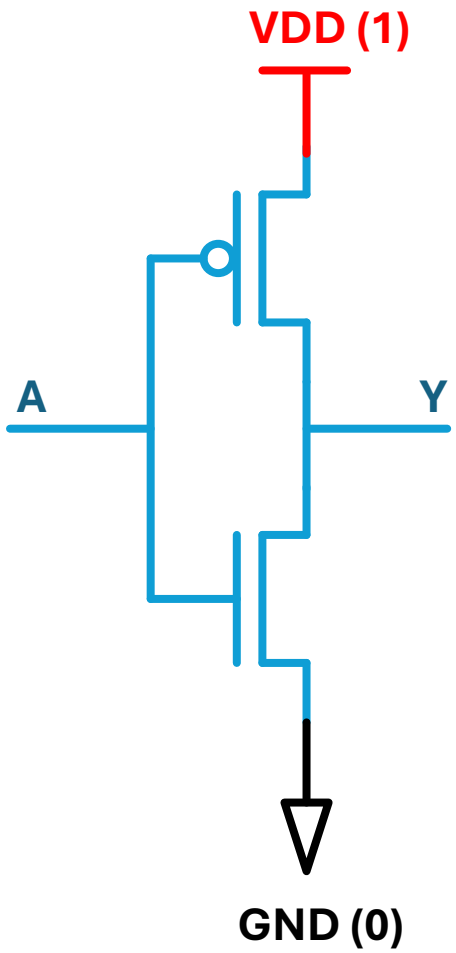
Place & Route:
Placing the blocks at the right spots, connecting with each other

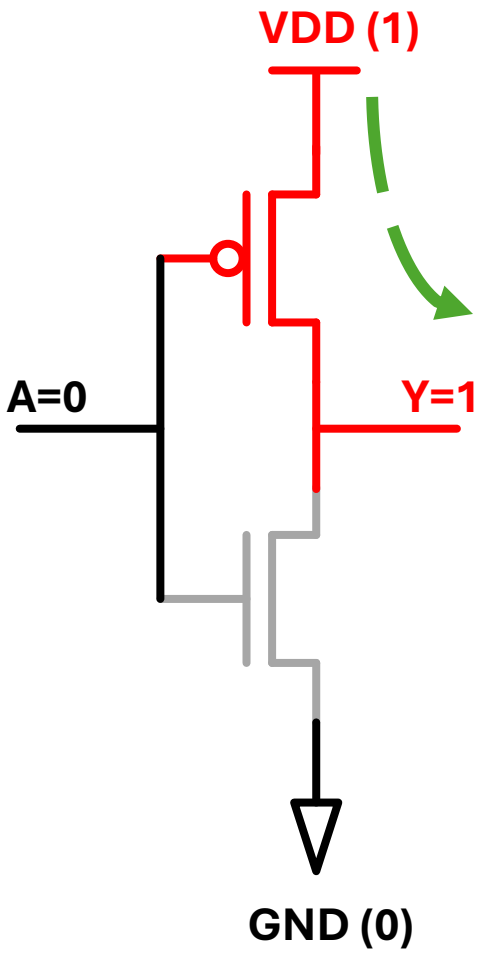


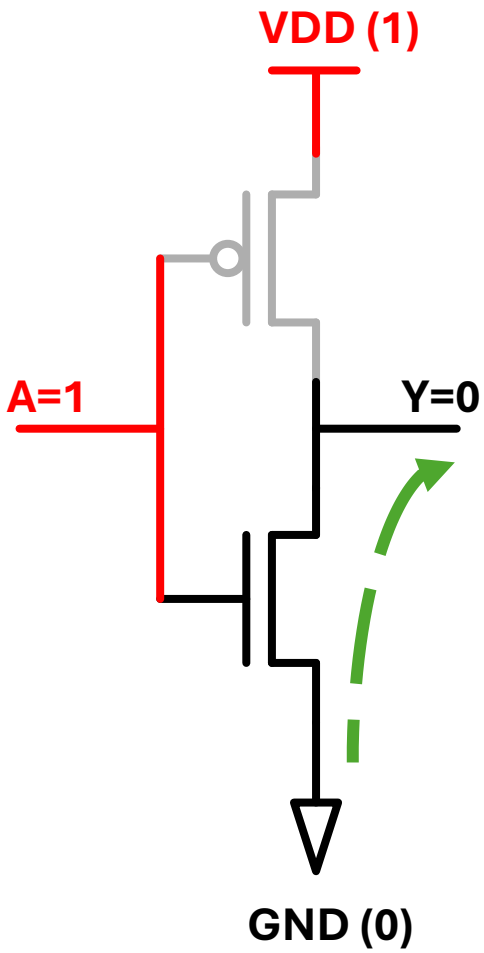
Done by EDA software

NOT Gate in CMOS

A	Y
0	1
1	0

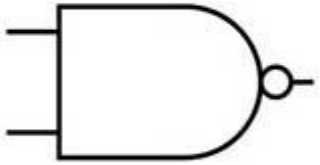






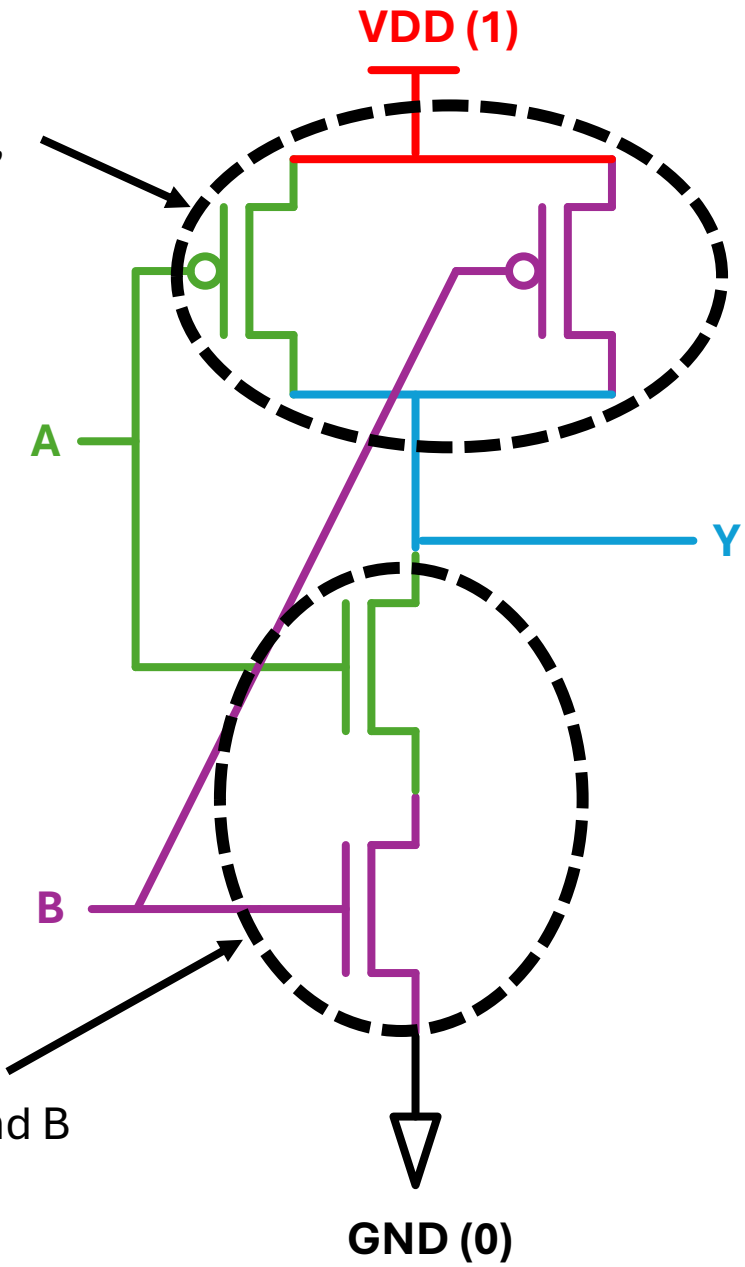
NAND Gate in CMOS

NAND



A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

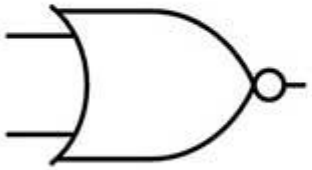
- PMOS in parallel
- Conducts if A’ or B’



- NMOS in series
- Conducts if A and B

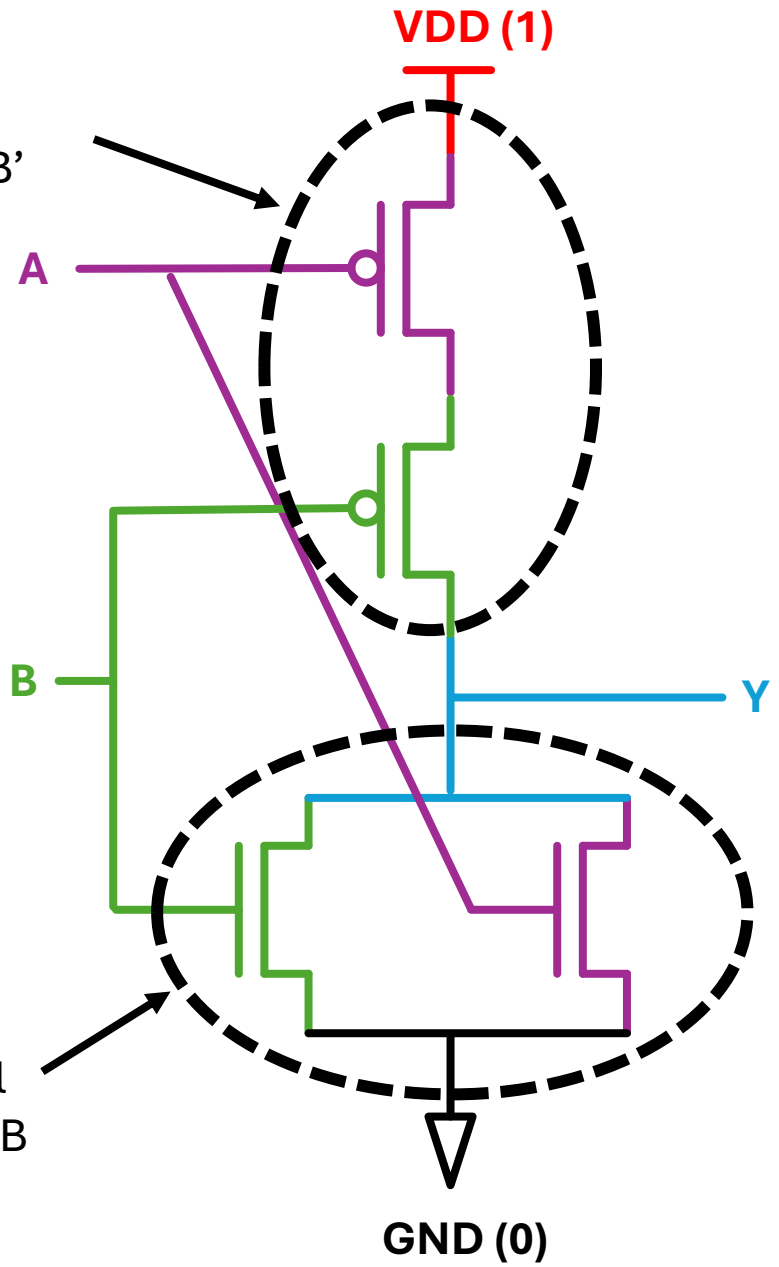
NOR Gate in CMOS

NOR



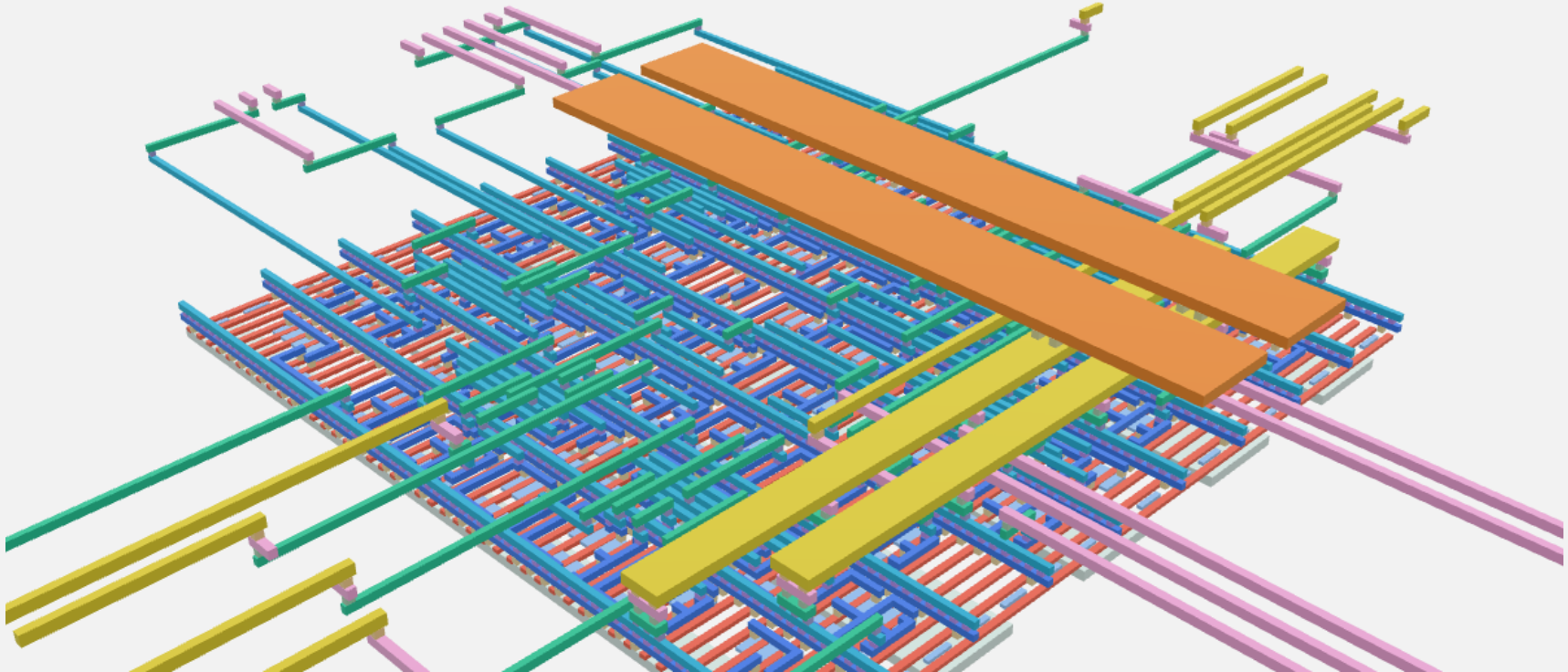
A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

- PMOS in series
- Conducts if A’ and B’



- NMOS in parallel
- Conducts if A or B

Basics of Boolean Algebra



Truth Tables

- Input/output are binary. Each can either be 0 or 1.
- Enumerate all possible combinations of inputs
  - N inputs → 2^N combinations
- For each combination of inputs, there's an output
- Standard way to write truth tables
  - Alternate between 0,1 less frequently from right to left
  - **i**th row is the binary representation of **i**



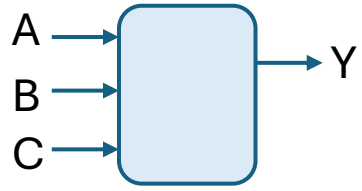
1 input

i	A	Y
0	0	
1	1	



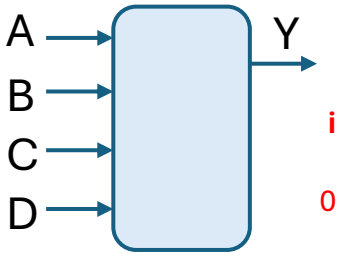
2 inputs

i	A	B	Y
0	0	0	
1	0	1	
2	1	0	
3	1	1	



3 inputs

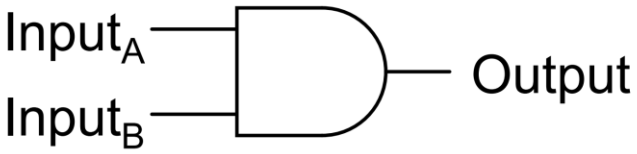
i	A	B	C	Y
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	



4 inputs

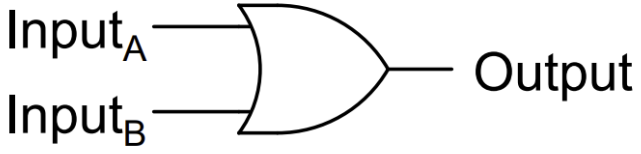
i	A	B	C	D	Y
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	

AND, OR, NOT Gates



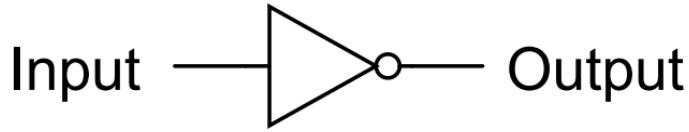
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Logical : **Y = A . B**
SystemVerilog : **Y = A && B**



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Logical : **Y = A + B**
SystemVerilog : **Y = A || B**



Input	Output
0	1
1	0

Logical : **Y = A'**
SystemVerilog : **Y = !A**

XOR Gate



- eXclusive OR gate
- **OR gate**
  - output is 1 if any input is 1
  - Includes the case where multiple inputs are 1
- **XOR gate**
  - output is 1, if any input is exclusively 1
  - Excludes case where two inputs are 1

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

$$A \oplus B = Q$$

build-electronic-circuits.com/xor-gate

XNOR Gate



- NOT of XOR
- Equivalence/similarity gate
- Q = (A==B)
- 1 if both inputs are same, otherwise 0

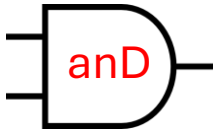
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

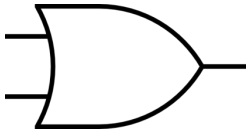
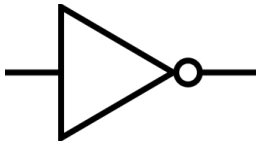
- **In Boolean algebra & logic minimization, we will only use AND, OR, NOT**

$$\overline{A \oplus B} = Q$$

build-electronic-circuits.com/xnor-gate/

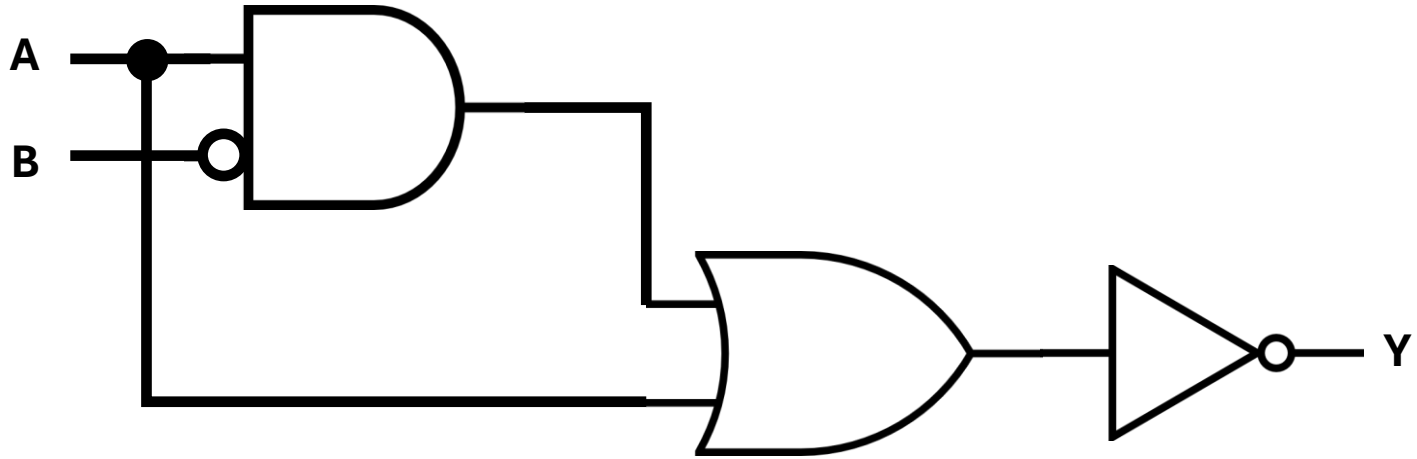
Logic circuits

- Basic gates: AND 

OR 
NOT 

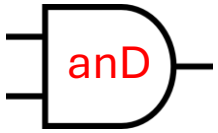
- White dot : Shorthand for not gate
- Black dot : Electrically connected
- Crossing without dot : Wire just passes over

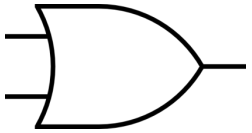
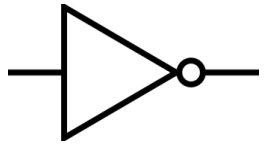
A = 1
B = 0
Y = ?



What is Y?

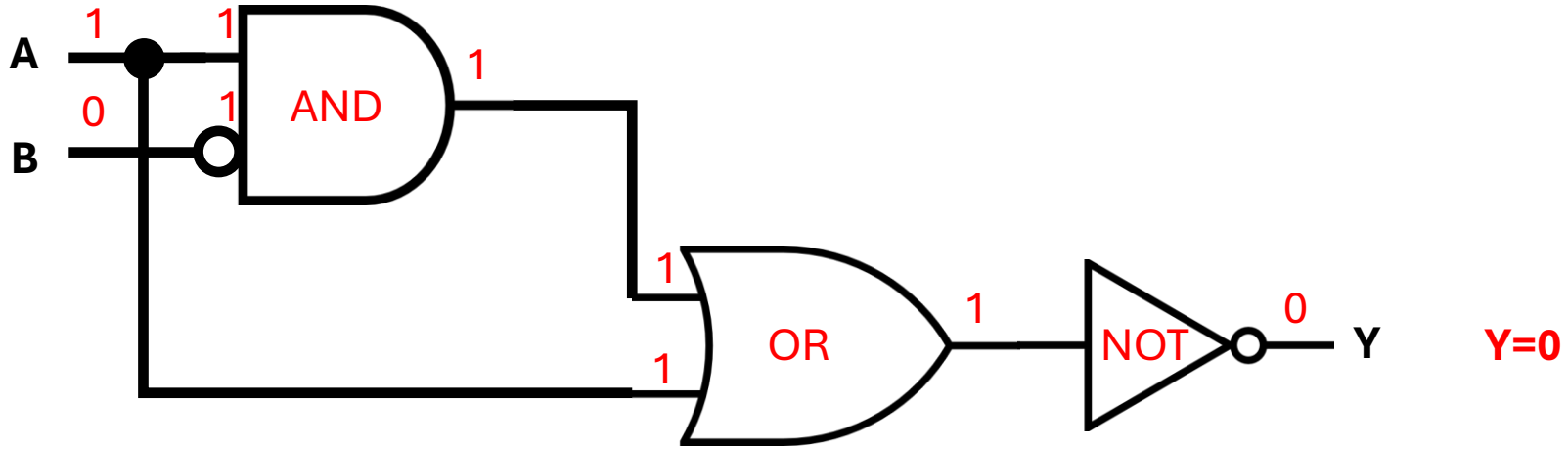
Logic circuits

- Basic gates: AND 

OR 
NOT 

- White dot : Shorthand for not gate
- Black dot : Electrically connected
- Crossing without dot : Wire just passes over

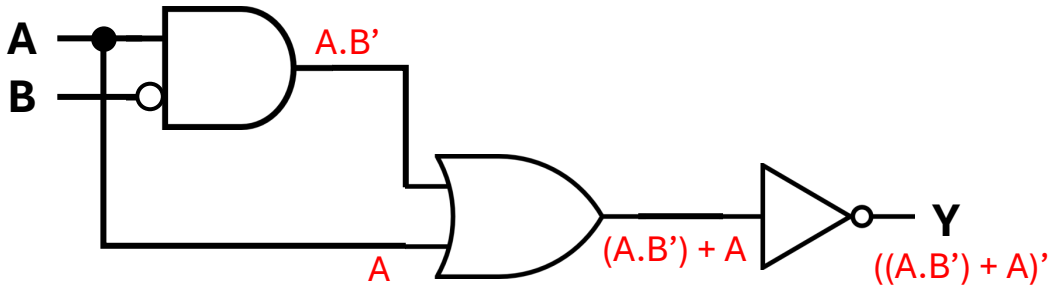
A = 1
B = 0
Y = ?



Y=0

Boolean Functions

- Function: **input** → **output** mapping
- All combinational circuits are Boolean functions
- Three equivalent ways to represent Boolean functions



Logical circuit diagram

$$Y = ((A.B') + A)'$$

Algebraic expression



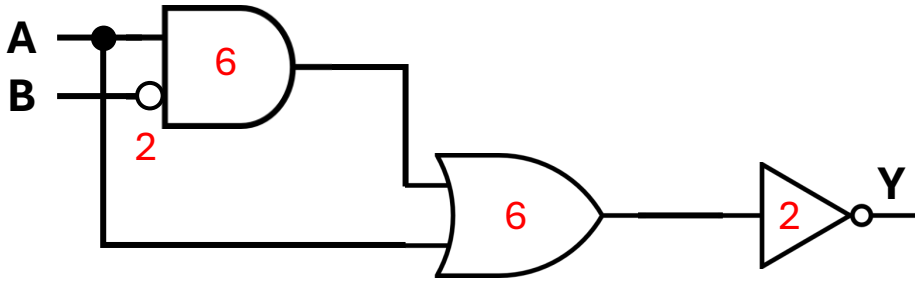
2 inputs

i	A	B	Y
0	0	0	1
1	0	1	1
2	1	0	0
3	1	1	0

Truth Table

How many transistors?

- NAND/NOR/NOT gates have 2 transistors per input
- 2 inputs → 4 transistors
- NOT has 2 transistors
- AND/OR = NAND+NOT / NOR+NOT = 6



Logical circuit diagram

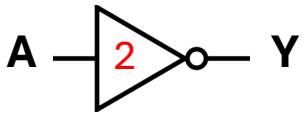
$$Y = ((A.B') + A)'$$

Algebraic expression

i	A	B	Y
0	0	0	1
1	0	1	1
2	1	0	0
3	1	1	0

Truth Table

16 Transistors



$$Y = A'$$

i	A	Y
0	0	1
1	1	0

2 Transistors

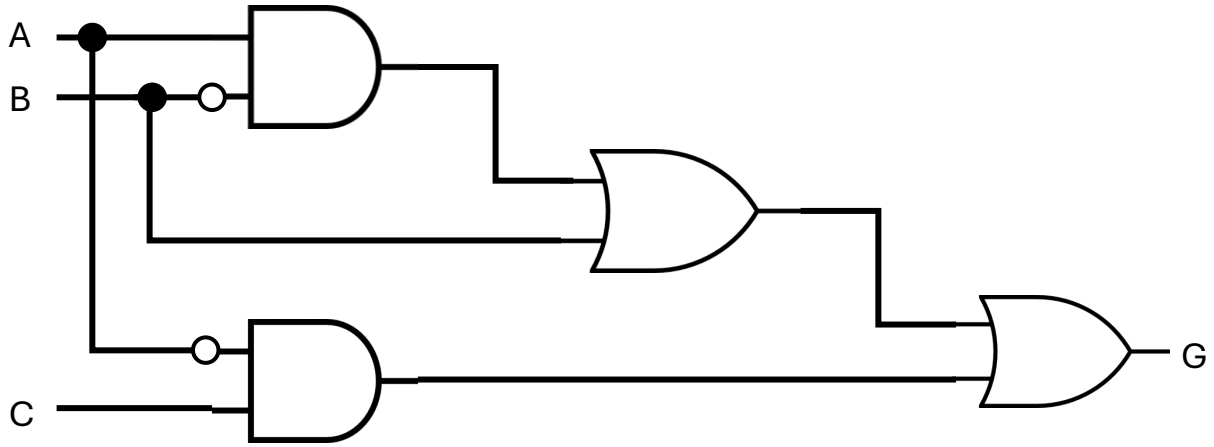
Logic Minimization: 88% smaller

Activity

Derive

- The number of transistors needed
- Algebraic expression
- Truth table

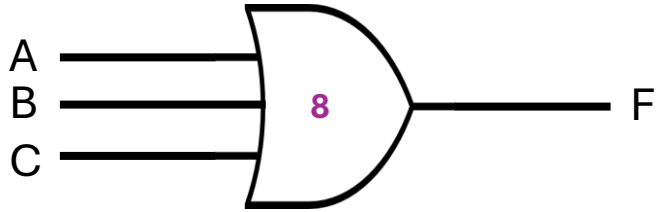
See if you can find a simpler expression, and count the transistors



DONE?

How many transistors?

Activity



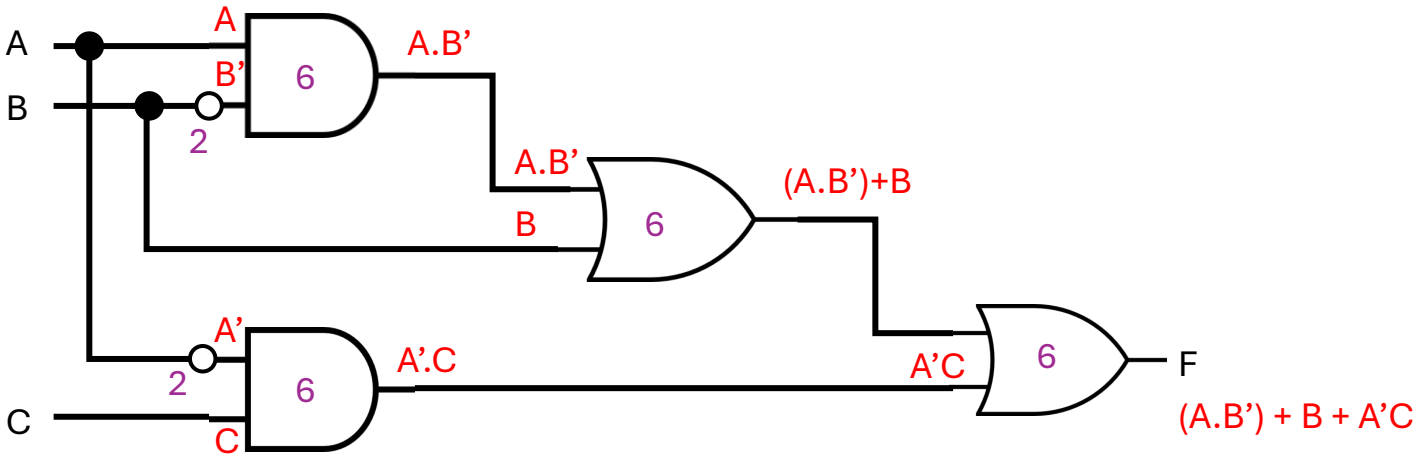
F = A + B + C

8 Transistors

Derive

- The number of transistors needed
- Algebraic expression
- Truth table

See if you can find a simpler expression, and count the transistors



28 Transistors

A	B	C	F
0	0	0	0
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	1

Key Takeaways

Boolean (2-state) functions can be represented in 3 equivalent forms:

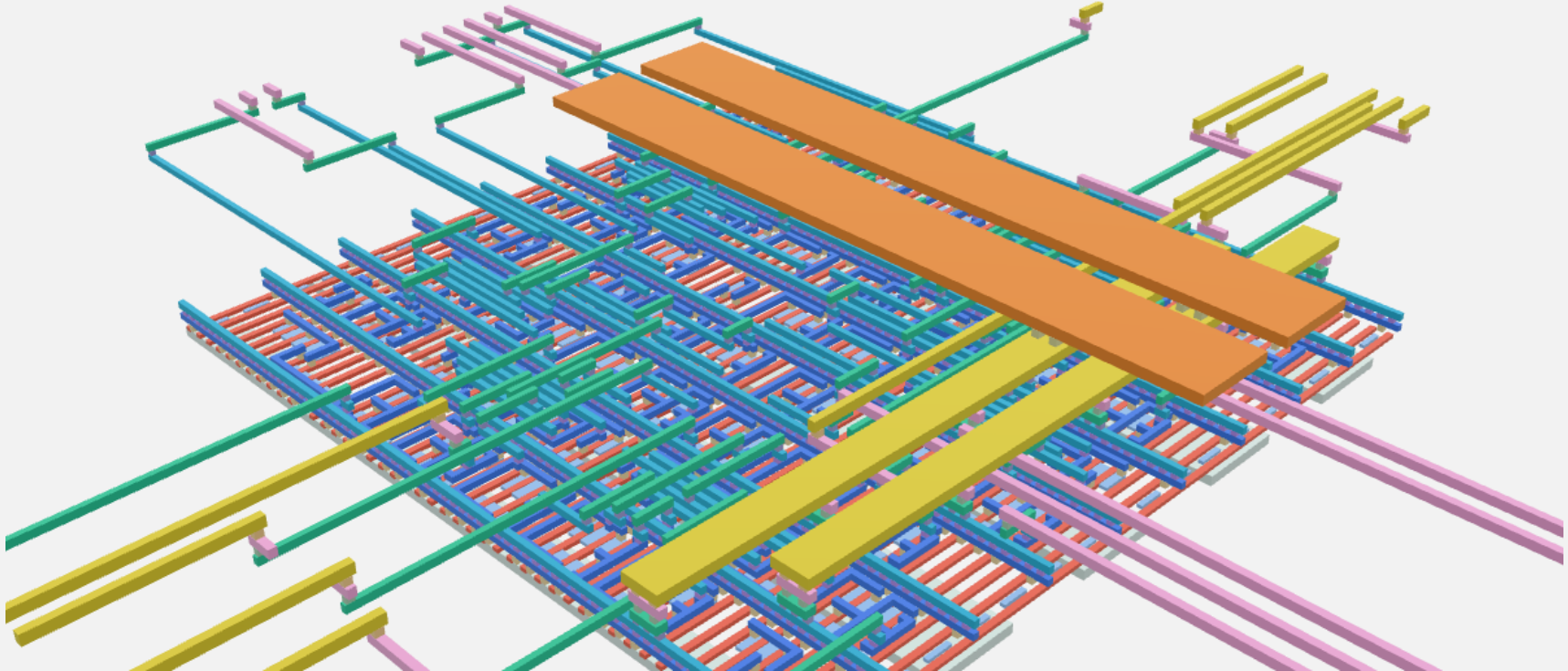
1. Logic circuits
2. Algebraic expressions
3. Truth tables

- (1), (2) are not unique.
- The same function can be simplified / made complicated into a different circuit / expression.
- Truth table is unique – only one truth table for a function – **A Canonical Representation**

How to find the simplest circuit (using AND, OR, NOT) for a function?

- Two ways:
    1. Simplify it algebraically using **Boolean Identities**
    2. Simplify it visually using **Karnaugh Maps**

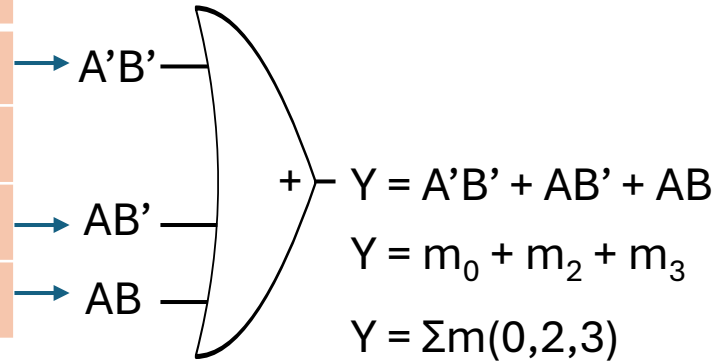
Logic Minimization



Canonical Forms: Sum of Minterms

- Truth table → expression?
- Truth Table is a canonical representation – because it's unique
- Can we create a canonical algebraic expression?

minterm	i	A	B	Y
A'B'	0	0	0	1
A'B	1	0	1	0
AB'	2	1	0	1
AB	3	1	1	1



- **Minterm**
  - product that includes all input variables
  - N inputs → 2^N minterms.
  - $\mathbf{m_i}$: minterm of the $\mathbf{i^{th}}$ row of truth table (i in binary).
  - If A=0, B=1, minterm is A'B. If A=1,B=1, minterm is AB
- Truth table → expression: Expression is OR of all minterms where output is 1 in truth table

Activity 1 – Sum of Minterms

- Derive the expression (as a sum of minterms) from the given truth table
- Steps:
  - Write all possible minterms
  - Select the minterms where output = 1
  - OR them to make the expression
  - Write the compact form: Σm(,,,)

i	A	B	Y
0	0	0	1
1	0	1	1
2	1	0	0
3	1	1	1

DONE?

Activity 1 – Sum of Minterms

minterm	i	A	B	Y
A’B’	0	0	0	1
A’B	1	0	1	1
AB’	2	1	0	0
AB	3	1	1	1

$Y = A'B' + A'B + AB$

$Y = m_0 + m_1 + m3$

$Y = \Sigma m(0,1,3)$

Activity 2 – Sum of Minterms

- Derive the expression (as a sum of minterms) from the given truth table
- Steps:
  - Write all possible minterms
  - Select the minterms where output = 1
  - OR them to make the expression
  - Write the compact form: Σm(,,,)

i	A	B	C	Y
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

Activity 2 – Sum of Minterms

minterm	i	A	B	C	Y	
A’B’C’	0	0	0	0	1	→ A’B’C’
A’B’C	1	0	0	1	0	
A’BC’	2	0	1	0	0	
A’BC	3	0	1	1	1	→ A’BC
AB’C’	4	1	0	0	1	→ AB’C’
AB’C	5	1	0	1	0	
ABC’	6	1	1	0	1	→ ABC’
ABC	7	1	1	1	0	

+

$Y = A'B'C' + A'BC + AB'C' + AB'C'$

$Y = m_0 + m_3 + m_4 + m_6$

$Y = \Sigma m(0,3,4,6)$

Canonical Forms: Product of Maxterms

- 2nd way to write an expression from a truth table
- Why? When truth table has many 1s, sum of minterms gives too many terms.

minterm	Maxterm	i	A	B	Y
A'B'	A+B	0	0	0	1
A'B	A+B'	1	0	1	0
AB'	A'+B	2	1	0	1
AB	A'+B'	3	1	1	1

Y = A'B' + AB' + AB = Σm(0,2,3)

Y = (A+B') = ∏M(1)

- **Maxterm**
  - Sum that includes all input variables
  - N inputs → 2^N Maxterms
  - $\mathbf{M_i}$: Maxterm of the $\mathbf{i^{th}}$ row of truth table (i in binary).
  - If input is 1, complement it (opposite to minterm)
    If A=0, B=1, Maxterm is (A + B'). If A=1,B=1, Maxterm is (A'+B')
- Truth Table to expression: AND all Maxterms where output is 0 (opposite to minterm)
- Compact form: ∏M(,,,)

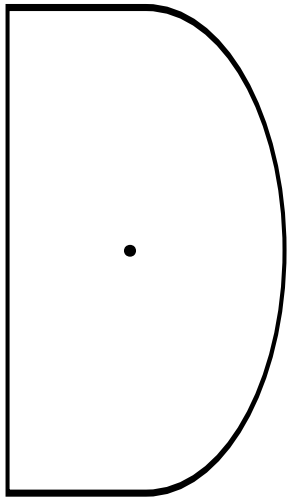
Activity – Product of Maxterms

- Derive the expression (as a product of maxterms) from the given truth table
- Steps:
  - Write all possible maxterms
  - Select the maxterms where output = 0
  - AND them to make the expression
  - Write the compact form: ∏ M(,,,)

i	A	B	C	Y
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

Activity – Product of Maxterms

Maxterm	i	A	B	C	Y	
A+B+C	0	0	0	0	1	
A+B+C’	1	0	0	1	0	→ A+B+C’
A+B’+C	2	0	1	0	0	→ A+B’+C
A+B’+C’	3	0	1	1	1	
A’+B+C	4	1	0	0	1	
A’+B+C’	5	1	0	1	0	→ A’+B+C’
A’+B’+C	6	1	1	0	1	
A’+B’+C’	7	1	1	1	0	→ A’+B’+C’



$Y = (A+B+C')(A+B'+C)(A'+B+C')(A'+B'+C')$

$Y = M_1 M_2 M_5 M_7$

$Y = \prod M(1,2,5,7)$

Boolean Identities for Algebraic Simplification

IDENTITY	EXPRESSION	
Logical Inverse	$\overline{0} = 1;\ \overline{1} = 0$	
Involution	$\overline{\overline{A}} = A$	
Dominance	$A + 1 = 1$	$A \cdot 0 = 0$
Identity	$A + 0 = A$	$A \cdot 1 = A$
Idempotence	$A + A = A$	$A \cdot A = A$
Complementarity	$A + \overline{A} = 1$	$A \cdot \overline{A} = 0$
Commutativity	$A + B = B + A$	$A \cdot B = B \cdot A$
Associativity	$(A + B) + C = A + (B + C)$	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$

allaboutcircuits.com/technical-articles/boolean-identities/

Example:

F = A(B+B') + C.0 + D''+ D

F = A(B+B') + C.0 + D+ D [D''=D]

F = A(B+B') + D+ D [C.0=0]

F = A(B+B') + D [D+D=D]

F = A.1 + D [B+B' = 1]

F = A+D

Distributivity

	OR	AND
Distributivity	$A + (B \cdot C) = (A + B) \cdot (A + C)$	$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
	May not be intuitive	Intuitive

Example:

F = A+A'B

F = (A+A')(A+B) [Distributivity]

F = 1. (A+B) [A+A' = 1]

F = A+B [1.A=A]

Example:

F = A + A'(B + C)

F = A + A'B + A'C [Distribute A' – but doesn't lead to anywhere]

F = (A + A')(A + (B+C)) [Distribute A: A+YZ = (A+Y)(A+Z)]

F = 1.(A + B + C) [A+A' = 1]

F = A + B + C

allaboutcircuits.com/technical-articles/boolean-identities/

DeMorgan’s Theorem

	OR	AND
DeMorgan's	$A + B = \overline{\overline{A} \cdot \overline{B}}$	$A \cdot B = \overline{\overline{A} + \overline{B}}$

Example:

F = (A + B)' + A’.B

Example:

F = A.(A.B)’

DeMorgan’s Theorem

	OR	AND
DeMorgan's	$A + B = \overline{\overline{A} \cdot \overline{B}}$	$A \cdot B = \overline{\overline{A} + \overline{B}}$

Example:

F = (A + B)' + A’.B

F = A’.B’ + A’.B **demorgan’s**

F = A’.(B’ + B)

F = A’.1

F = A’

Example:

F = A.(A.B)’

F = A.(A’+B’) **demorgan’s**

F = A.A’ + A.B’

F = 0 + A.B’

F = A.B’

NAND: Universal Gate

- Fundamental gate that can be used to construct any Boolean function
- NAND(A,A) = (A.A)' = A' = NOT(A) – can be used to create not gate

- Steps to make any function using NAND gates:
  1. Write the truth table
  2. Write it as sum of minterms: $m_x + m_y + m_z$
  3. Invert the sum twice and distribute the complement to make it NAND of NANDs $(m_x + m_y + m_z)'' = (m_x' m_y' m_z)'$
  4. Implement the NOT gates inside each minterm using NAND

i	A	B	C	Y
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

- Example: Implement using NAND gates

  Y = A'B'C' + A'BC + AB'C' + ABC' [sum of minterms]

Sum of minterms to NAND-only circuit

- Example: Implement using NAND gates

Y = A’B’C’ + A’BC + AB’C’ + ABC’ [sum of minterms]

Y = (A’B’C’ + A’BC + AB’C’ + ABC’)’’

Y = ((A’B’C’)’ .(A’BC)’ .(AB’C’)’ .(ABC’)’)’

NOR: Universal Gate

- Fundamental gate that can be used to construct any Boolean function
- NOR(A,A) = (A+A)’ = A’ = NOT(A) – can be used to create not gate
- Steps to make any function using NOR gates:
  1. Write the truth table
  2. Write it as product of Maxterms: $M_x .M_y .M_z$
  3. Invert the product twice and distribute the complement to make it NOR of NORs $(M_x .M_y .M_z)'' = (M_x' + M_y' + M_z')'$
  4. Implement the NOT gates inside each Maxterm using NOR

Absorption

OR	AND
$A+AB=A$	$A(A+B)=A$

Proof:

A.(A+B) = A.A + A.B

= A + A.B // OR form of absorption

= A.1 + A.B

= A.(1+B)

= A

Example:

F = A+A.B(C+D)

F = A + A. [B(C+D)]

F = A

Example:

F = B(B+C)(D+E)

F = [B(B+C)] (D+E)

F = B(D+E)

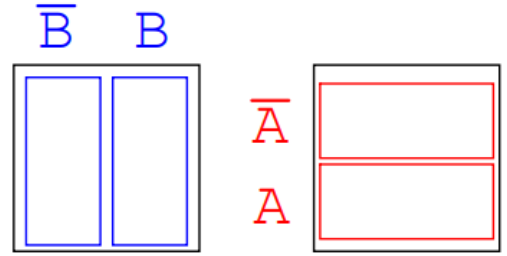
The Uniting Theorem – Key to simplification

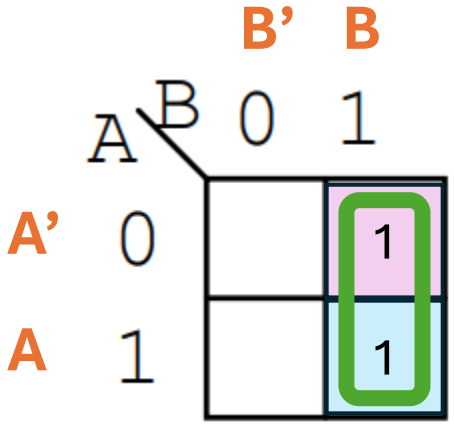
- A’ + A = 1
- B(A’+A) = B
- Essence of two-level logic simplification
  - Find two product terms where one variable changes value
  - Eliminate that variable

Karnaugh maps

- A visual way to simplify circuits using AND,OR,NOT
- **Steps:**
  - Draw the map & Put 1s where output has ones
  - Make the biggest rectangular/square groups of 1s (must be power of 2)
  - Starting from the biggest, for each rectangle:
    - write the terms whose values don't change across the rectangle
    - sum the products, until all 1s are covered

A	B	Output
0	0	0
0	1	1
1	0	0
1	1	1

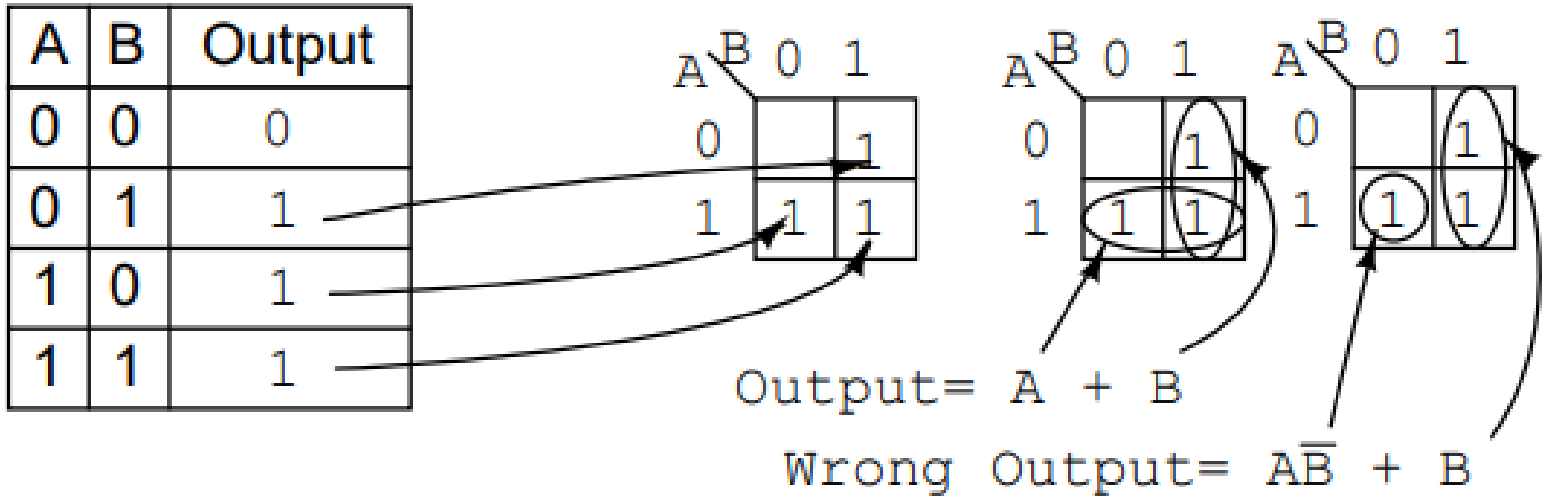




- B=1 across the rectangle of size 2
- Term is B
- F = B

K-map example 2

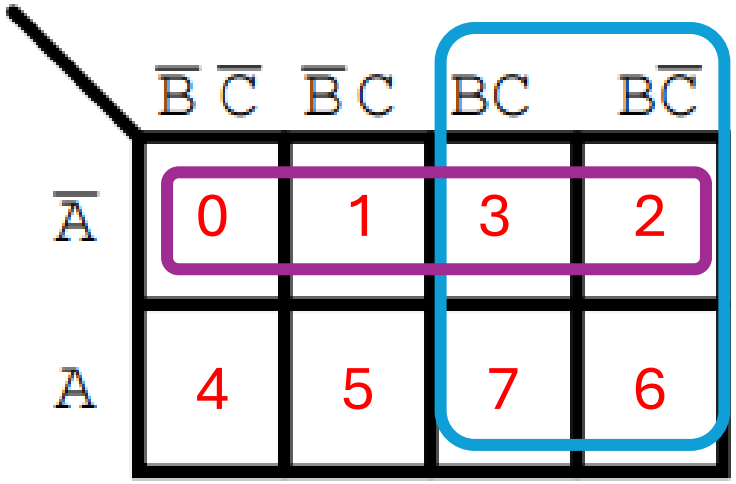
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



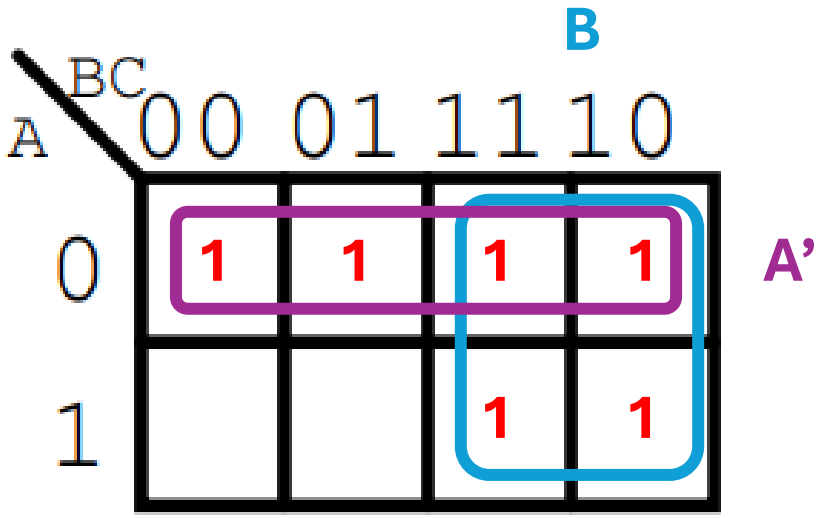
3-variable K-map

- **Important:** Columns are enumerated as gray code: 00, 01, 11, 10
- Rows & columns wrap around – need to draw rectangles accordingly

i	A	B	C	Y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1



Each cell corresponds to a minterm



1 means that minterm exists in the sum

F = A' + B

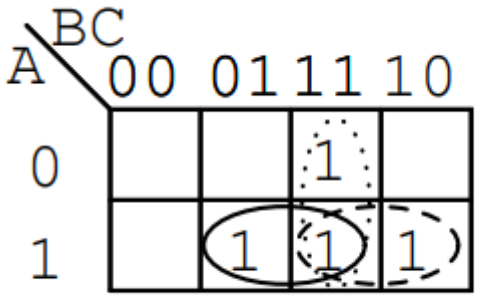
Activity – Derive the expression through a K-map

i	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

DONE?

Activity – Derive the expression through a K-map

i	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



Output = AB + BC + AC

Activity 2 – Derive the expression through a K-map

- **Important:** Edges of the K-map wraps around (check the minterms)

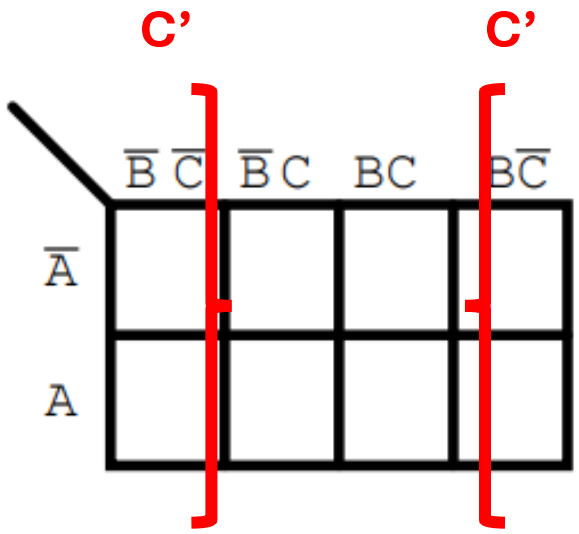
i	A	B	C	Y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

DONE?

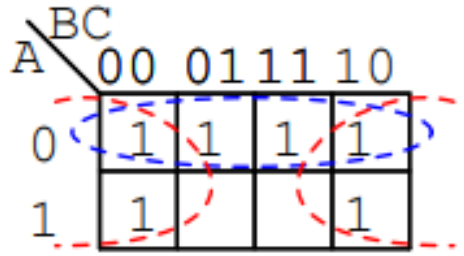
Activity 2 – Derive the expression through a K-map

- **Important:** Edges of the K-map wraps around (check the minterms)

i	A	B	C	Y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0



$$Out = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + AB\overline{C}$$

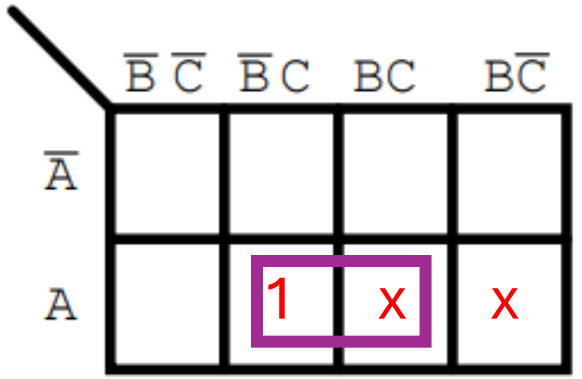


$$Out = \overline{A} + \overline{C}$$

Don’t care Conditions

- A counter counts 0,1,2,3,4,5,0,1,2… indefinitely.
- We need a circuit to detect whenever it reaches 5.

i	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	x
7	1	1	1	x



Y =AB’C

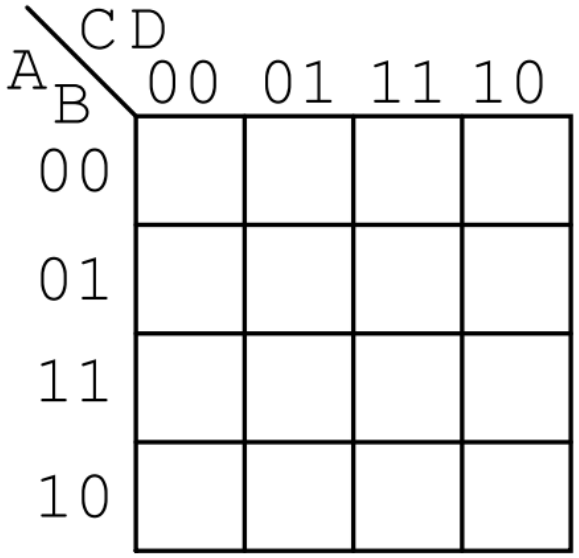
Utilizing don’t cares:

Y =AC

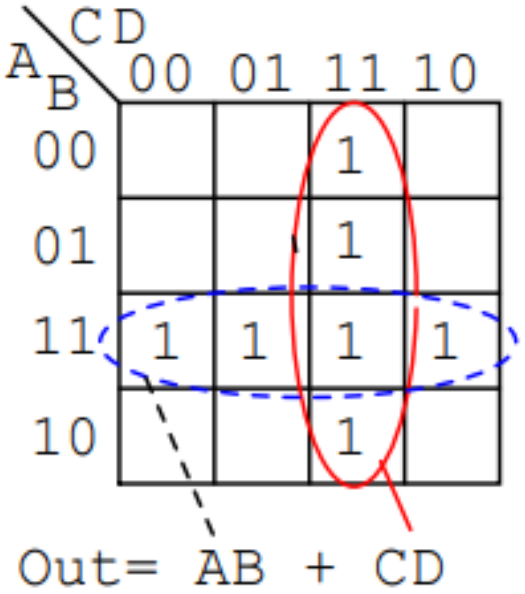
The counter will never be 6 or 7.
So, our circuit can ignore those input cases, if it makes the circuit simpler

4-variable K-map

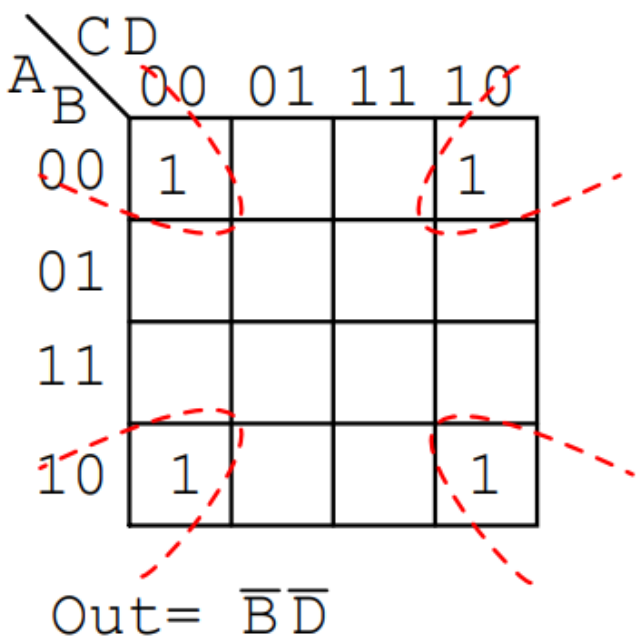
- Both rows and columns are enumerated as gray code: 00, 01, 11, 10, and they wrap

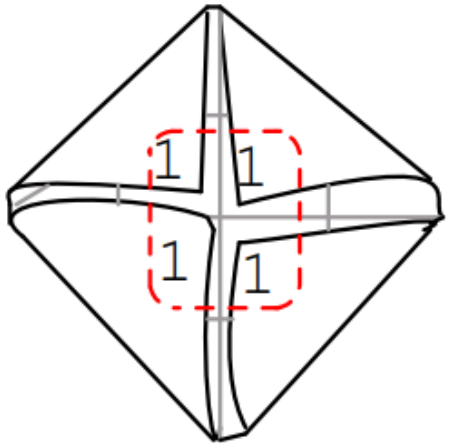


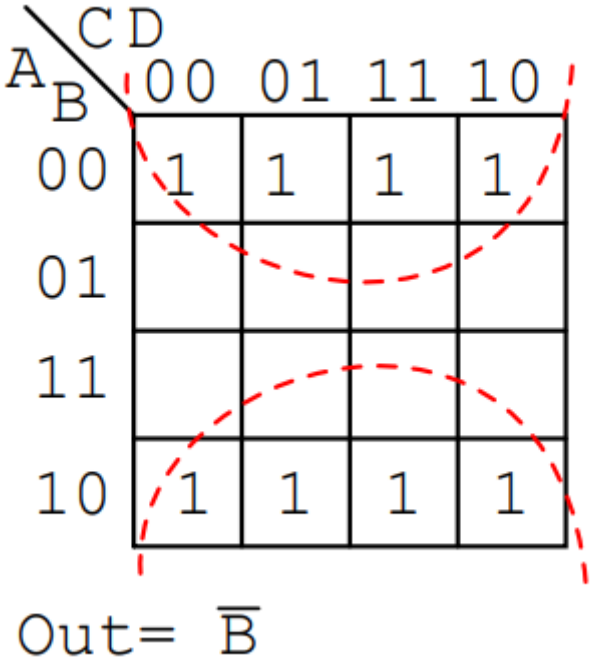
$$Out = \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}D + ABCD + A\overline{B}CD + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABC\overline{D}$$



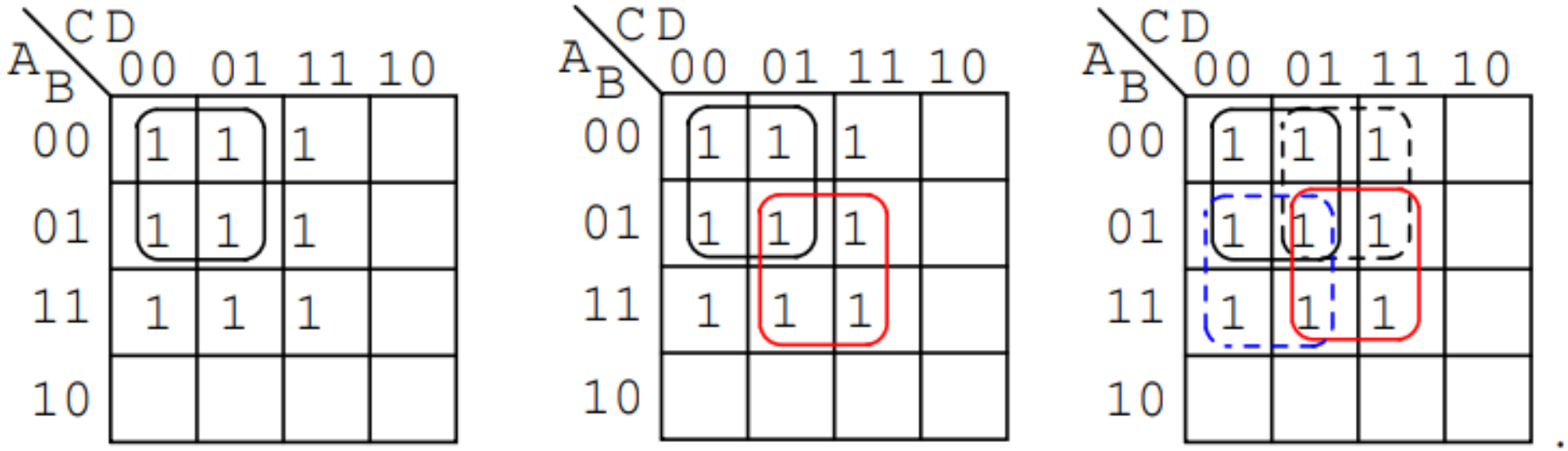
Wrap arounds







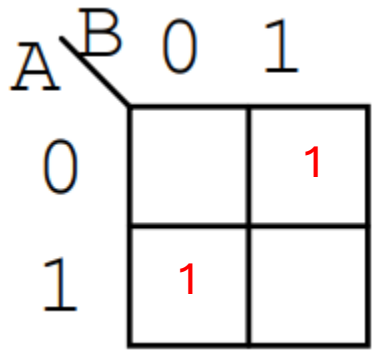
Another Example



$$\text{Out} = \overline{A}\,\overline{C} + \overline{A}D + B\overline{C} + BD$$

Limits of K-maps

i	A	B	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0



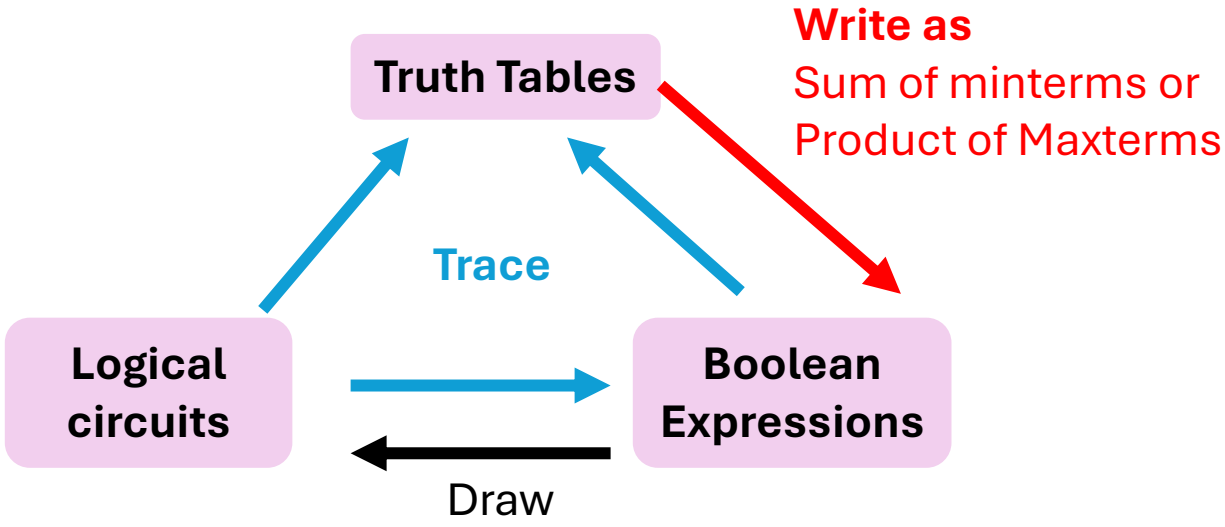
Y = AB’ + A’B

Y = A xor B is simpler

- K-maps only help to boil down a circuit to AND, OR, NOT gates
- But modern PDKs have standard cells (lego blocks) of compound gates
- Synthesis tools use advanced algorithms to perform minimization

Summary

- Boolean functions can be represented in 3 ways: Truth tables, circuits, expressions
- Canonical forms (unique representations)
  - Truth Table
  - Sum of Minterms
  - Product of Maxterms



- **Finding the smallest circuit**
  - Simplify an expression using identities
  - K-maps
- But even these are limited to AND,OR,NOT

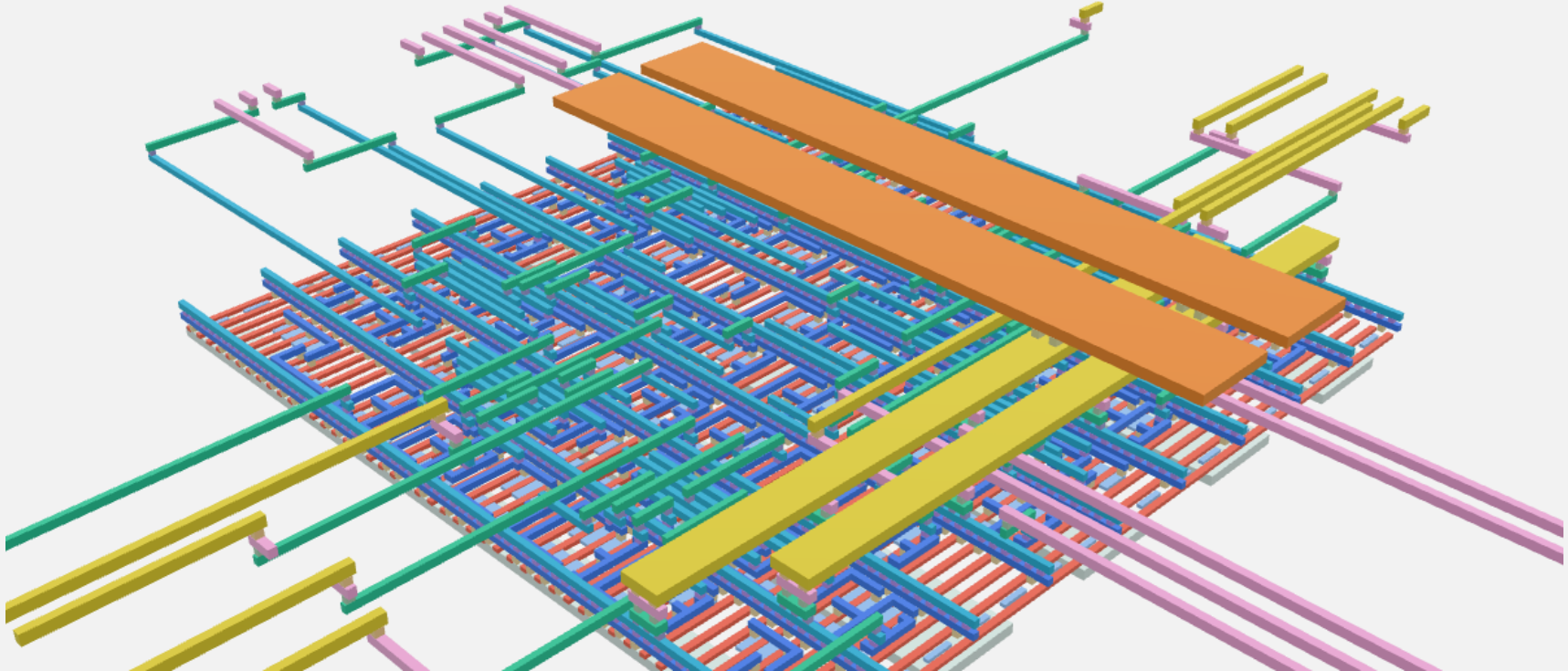
Day 1 Feedback

- One thing I liked about today’s class:
- One thing that could be improved:
- One place where I felt lost (if any):

Practice Problems

- **Practice Problems**
  - Trace a logic circuit and construct its truth table
  - Write a function as a **sum of minterms**
  - Write a function as a **product of maxterms**
  - Simplify an expression using **Boolean identities**
  - Minimize a function using a **3-variable K-map**
  - Minimize a function using a **4-variable K-map**
- **References**
  - Feher: Chapters 2–4, Appendix F
  - *Lessons in Electric Circuits*: Chapters 7–8
  - *Karnaugh Mapping* worksheet: All questions

Introduction to SystemVerilog



Hello SystemVerilog!

```
// hello_tb.sv

module hello_tb;

  initial begin
    $display("Hello, SystemVerilog!");
    $finish;
  end

endmodule
```

```
verilator --binary hello_tb1.sv
```

Compile using verilator

```
./obj_dir/Vhello_tb
```

Run compiled simulation

- This is software code
- SystemVerilog is used to describe hardware (Synthesizable RTL), and also to write software code (non synthesizable testbench) to test it
- Verilator – open source, industry-standard simulator

Simulating signals

```sv
// and_tb.sv
`timescale 1ns/1ps

module and_tb;
  bit A, B, C;
  always_comb C = A && B;


  initial begin
    $dumpfile("waveform.vcd"); $dumpvars(0, and_tb);

        A = 0; B = 0; #1; $display("A=%b B=%b C=%b", A, B, C);
    #9  A = 0; B = 1; #1; $display("A=%b B=%b C=%b", A, B, C);
    #9  A = 1; B = 0; #1; $display("A=%b B=%b C=%b", A, B, C);
    #9  A = 1; B = 1; #1; $display("A=%b B=%b C=%b", A, B, C);

    #9 $finish;
  end
endmodule
```

```
verilator --binary --timing --trace-vcd and_tb.sv
```

Compile using verilator

```
./obj_dir/Vand_tb
```

Run compiled simulation

```
gtkwave waveform.vcd
```

View waveform

- Always_comb – create combinational (stateless) circuits
- Dumpfile, dumpvars – to dump waveform
- Display – print statement, like C
- #9 – wait for 9ns

