

ECE 215/S – Principles of Electronic Cyber Warfare

Block 2: Signal Processing
Lesson 23: Digital Logic

Department of Electrical and Computer Engineering
United States Air Force Academy

Fall 2026

Objective 2.7

I can relate digital inputs to digital outputs using tools such as fundamental digital logic gates (AND, OR, NOT), truth tables, and Sum of Products (SOP) and Product of Sums (POS) Boolean expressions.

- Objective 2.5 turned a voltage into bits, and at **Lesson 22** you watched your own ADC do it — a calibrated distance, in code
- Today is the logic that decides what to *do* with that number
- Assessed on **Quiz 7** along with Objective 2.8 — due at **Lesson 27**

Where We Are Going

By the end of this lesson, you will be able to:

- Explain why digital representation is **robust** in a way analog is not
- Name and use the gates: **NOT, AND, OR, NAND, NOR, XOR, XNOR**
- Build a **truth table** for a compound logic circuit
- Write that truth table as a **Sum of Products** or **Product of Sums** expression

Why Digital?

An analog voltage carries infinite precision — and every bit of noise corrupts it permanently.

- A digital signal has only **two** valid states: **0** and **1**, represented by voltage ranges
- Noise that does not push a value across the decision threshold is **completely removed** when the signal is regenerated
- That is why a digital signal can cross a continent, or be copied a thousand times, without degrading
- The cost is that you must **quantize** first — and quantization error, unlike noise, never goes away

Digital trades a one-time, known error for immunity to every error after it.

Binary

- Each **bit** is one binary digit — 0 or 1
- n bits represent 2^n distinct values
- Place values are powers of 2, right to left: $1, 2, 4, 8, 16, \ldots$

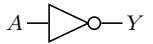
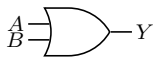
Example: $1011_2 = 8 + 0 + 2 + 1 = 11_{10}$

Where you have already seen this

- The ADC's output in Objective 2.5 is a binary number
- A 10-bit ADC gives $2^{10} = 1024$ levels — the same 2^n

Logic gates operate on these bits one at a time. Everything a computer does is built from what follows.

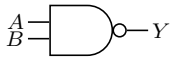
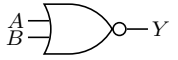
The Three Fundamental Gates

Gate	Symbol	Notation	Output is 1 when…
NOT		$\overline{A}$ or A'	the input is 0 — "the opposite of"
AND		$A \cdot B$, or just AB	*all* inputs are 1 — "both"
OR		$A + B$	*any* input is 1 — "either or both"

A	B	$A \cdot B$	$A + B$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Every other gate can be built from these three.

Derived Gates

Gate	Symbol	Built from AND, OR, NOT	Output is 1 when…
NAND		$\overline{A \cdot B}$	*not* all inputs are 1
NOR		$\overline{A + B}$	*no* input is 1
XOR		$A \oplus B = \overline{A}B + A\overline{B}$	the inputs **differ**
XNOR		$\overline{A \oplus B} = AB + \overline{A}\,\overline{B}$	the inputs **match**

A	B	NAND	NOR	XOR	XNOR
0	0	1	1	0	1
0	1	1	0	1	0
1	0	1	0	1	0
1	1	0	0	0	1

XOR is the “are these different?” gate — everywhere in arithmetic and error checking.

Note the **bubble**: on the symbol it means “invert the output”. XOR’s second back arc is what separates it from a plain OR.

Truth Tables

A **truth table** lists the output for *every possible* combination of inputs.

- With n inputs there are 2^n rows — 2 inputs give 4 rows, 3 inputs give 8
- List the input combinations in **binary counting order**. That guarantees you miss none
- The truth table is the **complete specification** of a **combinational** circuit — one whose output depends only on the inputs *right now*, with no memory of what came before. Two circuits with the same truth table are functionally identical, however differently they are built

The truth table is the ground truth. Gates are just one way to realize it.

Method: Building a Truth Table for a Compound Circuit

1. Write all 2^n input combinations in binary counting order
2. Add **one column per intermediate signal** inside the circuit, in the order the signals are produced. If the drawing does not name a wire, **name it yourself** — P, Q, R — and use that as the column head
3. Fill each intermediate column using only the columns to its left
4. The last column is the output
5. **Check**: n inputs give exactly 2^n rows, every intermediate column depends only on columns to its left, and every wire in the drawing has a column of its own

Reading the drawing. A **filled dot** where wires meet —  — is one signal feeding two places (*fan-out*): the same value arrives at both gates, so it gets *one* column. Wires that merely cross with no dot are unrelated.

Give every wire its own column. Holding them in your head is where errors come from.

Sum of Products (SOP)

Read the expression straight off the rows where the output is **1**.

1. Find every row with output $= 1$
2. For each such row, write a **product** (AND) of all inputs, complementing any input that is 0 in that row
3. **Sum** (OR) those products together

Example. Output is 1 for rows $(A, B) = (0, 1)$ and $(1, 1)$:

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

Each product term is a "this exact row" detector. OR them together and you cover every 1.

Product of Sums (POS)

The mirror image — read off the rows where the output is **0**.

1. Find every row with output $= 0$
2. For each such row, write a **sum** (OR) of all inputs, complementing any input that is **1** in that row
3. **Multiply** (AND) those sums together

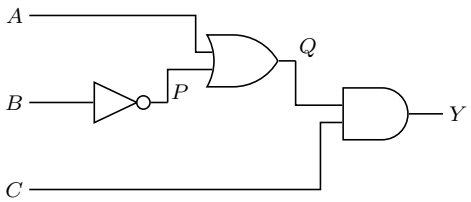
Example — the same circuit as the last frame, whose output is 0 on rows $(A, B) = (0, 0)$ and $(1, 0)$. Complement the **1s**: row $(0, 0)$ has none, giving $(A + B)$; row $(1, 0)$ has A, giving $(\overline{A} + B)$:

$$Y = (A + B)\,(\overline{A} + B)$$

Note the double inversion. In SOP you complement the 0s; in POS you complement the 1s. Getting that backward is the most common error on this objective.

Few 1s in the table? Use SOP. Few 0s? Use POS. Both describe the same circuit.

Worked Example: Circuit → Truth Table → SOP



A	B	C	P	Q	Y
0	0	0	1	1	0
0	0	1	1	1	**1**
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	1	0
1	0	1	1	1	**1**
1	1	0	0	1	0
1	1	1	0	1	**1**

Run the method. Three inputs, so $2^3 = 8$ rows. Two wires inside the circuit have no name, so name them — $P = \overline{B}$ out of the NOT gate, $Q = A + \overline{B}$ out of the OR gate, and the output is $Y = Q \cdot C$. Each column is filled from the columns to its left only.

SOP — one product per 1-row, complementing the inputs that are 0 in that row:

$$Y = \overline{A}\,\overline{B}C + A\overline{B}C + ABC$$

Check the count. Three 1-rows, three product terms; three inputs, so three literals in every term. Five 0-rows against three 1-rows, so SOP is the shorter form here.

Read straight off the drawing, the circuit is $Y = (A + \overline{B}) \cdot C$ — **the same circuit, two descriptions**, which is the point of the lesson.

Your Turn

A circuit has three inputs A, B, and C, and outputs 1 when **exactly one** of them is 1.

1. Build the truth table — how many rows?
2. Write the SOP expression
3. Count the 0-rows: which form is shorter here, SOP or POS? Write the POS only if you have time
4. Does a single standard gate do this? (Careful: what does a three-input XOR output for 111?)

Wrap-Up

Key ideas:

- Digital is **robust**: only two states are valid, so noise below the threshold disappears on regeneration
- **NOT, AND, OR** are fundamental; NAND, NOR, XOR, XNOR are built from them — each with its own drawn symbol
- A **truth table** with 2^n rows completely specifies a combinational circuit
- **SOP** reads the 1-rows (complement the 0s); **POS** the 0-rows (complement the 1s)
- **Where this goes: Project 2**'s decision logic is built exactly this way — a table of what the machine should do, read out as SOP

Gate fabrication (TTL, CMOS) and the Boolean simplification laws are in the reading; neither is on the quiz.

- **Lesson 24 — Objective 2.8: Finite State Machines.** What happens when the output depends on *history*, not just current inputs
- **Quiz 6** (Objectives 2.5–2.6) is due at Lesson 24