

ECE 215/S – Principles of Electronic Cyber Warfare

Block 2: Signal Processing

Lesson 25: Finite State Machines II

Department of Electrical and Computer Engineering
United States Air Force Academy

Fall 2026

Objective 2.8 (completing)

I can model Moore and Mealy Finite State Machines (FSMs) using tools such as state transition diagrams, transition tables, and combinational and sequential logic circuits.

- Lesson 24 gave you the **model**. Today: the **implementation**
- That last phrase — “combinational and sequential logic circuits” — is what we build today
- Assessed on **Quiz 7**, and it is the whole of **Project 2** — which requires the diagram and table, with the circuit step in software. The project’s equations are optional: check yours with **Test All Rows** in the FSM Explorer’s **Student Logic** card

Where We Are Going

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

- Work the **seven-step FSM design process** end to end
- **Encode** states, inputs, and outputs in binary
- Extract **logic equations** from a transition table
- Explain how **flip-flops** provide the memory, and sketch the resulting circuit

Method: The Seven-Step FSM Design Process

1. **Define and encode the states**
2. **Define and encode the inputs**
3. **Define and encode the outputs**
4. **Create the state transition diagram**
5. **Create the state transition table**
6. **Express the logic equations**
7. **Implement with flip-flops and logic gates**

Steps 1–5 are Lesson 24's model, now in **binary**; 6–7 turn the table into hardware. **Check** at step 5: every state code has a row for every input value, and every row has a next state.

Do them in order. Skipping to the circuit is how you get a machine that almost works.

Steps 1–3: Encoding

Hardware only understands bits, so everything gets a binary code.

States

- n states need $\lceil \log_2 n \rceil$ bits
- 3 or 4 states $\rightarrow$ 2 bits ($Q_1 Q_0$)
- Each bit needs its own **flip-flop** (the memory element — frame 8)

Inputs and outputs

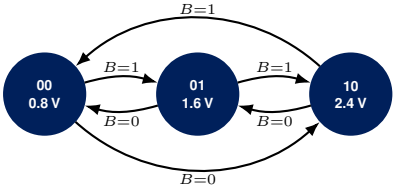
- A two-valued input is 1 bit
- A three-valued input needs 2 bits
- Outputs likewise

2 bits gives 4 states. If you only use 3, decide what the unused code does.

An unhandled state code is a real failure mode — a glitch can land the machine there and it will never come back.

Steps 4–5: Diagram, Then Table

- Redraw the diagram with the **binary codes** in the bubbles, and rebuild the table with Q_1Q_0, the input bits, and $Q_1^+Q_0^+$ — the $^+$ means *next*
- The exhaustive table has $\mathbf{2^{k+m}}$ rows for k state bits and m input bits — fill **every** one, unused codes included. Lesson 24 counted 6 over this machine's three *states*; 2 bits hold **four** codes, so in binary it is 8 rows: the 6 legal ones plus 2 for the unused 11
- Many inputs? One row per **(state, condition)** pair in **priority order** instead — the Project 2 form





Q_1	Q_0	B	Q_1^+	Q_0^+	**Out**
0	0	1	0	1	0.8 V
0	1	1	1	0	1.6 V
1	0	1	0	0	2.4 V

The same telemetry voltages you will use in Project 2 — but a different machine. (Other rows omitted for space.)

Step 6: Logic Equations from the Table

Each **next-state bit** is just a combinational function of the current state bits and the inputs — which is Objective 2.7.

1. Treat the Q_1^+ column as the output of a truth table whose inputs are Q_1, Q_0, and B
2. Write its **SOP** expression — one product term per row where $Q_1^+ = 1$
3. Repeat for Q_0^+
4. Repeat for each **output** bit

An FSM is combinational logic wrapped around memory. You already know the hard half.

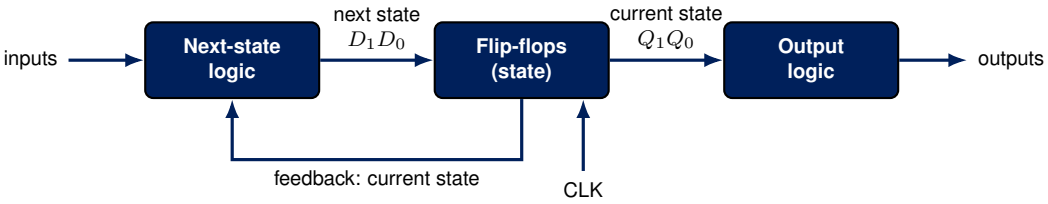
Flip-Flops: Memory in Hardware

A **flip-flop** stores one bit and updates it on a clock edge.

- A **D flip-flop** copies its input D to its output Q at each clock edge, and holds it in between
- That hold is the memory — it is what makes the circuit sequential
- You need **one flip-flop per state bit**. Two bits, two flip-flops
- The **clock** sets the pace: the machine advances one transition per clock edge

Combinational logic computes the next state.
The flip-flops remember it until the next clock.

Step 7: The Complete Circuit

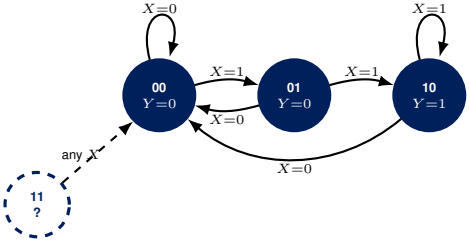


- **Next-state logic** (combinational) reads the current state and inputs
- **Flip-flops** latch the result on the clock edge
- **Output logic** produces the outputs — from state alone (Moore) or state and inputs (Mealy)
- The **feedback path** is what makes it a machine rather than a function

Worked Example: Sequence Recognizer

Design a Moore FSM that outputs 1 when it has seen the input sequence **1 then 1**.

Steps 1–4: encode, then draw



S0 $= 00$ (nothing yet), S1 $= 01$ (one 1), S2 $= 10$ (two 1s, $Y = 1$); 11 unused. Input X and output Y are 1 bit each.

Step 5: the table

Q_1	Q_0	X	Q_1^+	Q_0^+	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	1	0	1
1	1	0	0	0	0
1	1	1	0	0	0

Any $X = 0$ breaks the run and sends you back to S0. A third 1 *holds* S2 — the last two bits are still 11, so the match still stands (**overlapping** detection). The last two rows are the unused code, forced to 00: a corrupted state must fail safe.

Worked Example — Logic Equations

Step 6. Read SOP off each next-state column, then off the output column.

Q_1^+ is 1 in two rows: $(0, 1, 1)$ and $(1, 0, 1)$.

$$Q_1^+ = \overline{Q_1}\, Q_0\, X + Q_1\, \overline{Q_0}\, X$$

Q_0^+ is 1 in one row: $(0, 0, 1)$.

$$Q_0^+ = \overline{Q_1}\, \overline{Q_0}\, X$$

Y is 1 in two rows, $(1, 0, 0)$ and $(1, 0, 1)$:

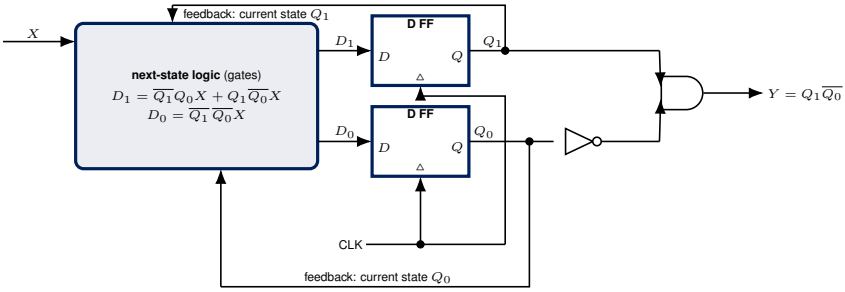
$$Y = Q_1\, \overline{Q_0}\, \overline{X} + Q_1\, \overline{Q_0}\, X$$

The two terms differ only in X, so X does not matter: Y is 1 exactly in state S2 $= 10$.

$$Y = Q_1\, \overline{Q_0}$$

No input variable in the output is the **Moore** signature. Both forms are the same circuit; we do not simplify algebraically here. **Step 7** (next frame): two D flip-flops, these three expressions as gates.

Worked Example — The Circuit



- Two **D flip-flops** hold Q_1 and Q_0 — one per state bit — on **one shared clock**
- The step-6 expressions are the gates inside the next-state block; the clock edge latches D_1 and D_0
- Y is an **AND** of Q_1 and $\overline{Q_0}$ — state only, so **Moore**. A filled dot joins wires; a plain crossing does not

Your Turn

Design a Moore FSM that counts $0 \to 1 \to 2 \to 0$ (a circular 3-count), with a direction input B: $B = 1$ counts **up**, $B = 0$ counts **down**. You already have this table from Lesson 24 — today you put it in binary and turn it into equations.

1. Encode the three states. How many bits and flip-flops?
2. Build the full transition table. How many rows?
3. Write the SOP equations for Q_1^+ and Q_0^+
4. What happens if the machine powers up in the unused fourth state code?

Project 2 runs this same process on a different machine — the fire-control engagement FSM, three states in two bits, same leftover code. Its table lists one row per **(state, condition)** pair in priority order, and **Project 2 requires the diagram and table** (steps 1–5) and makes step 6, the logic equations, optional but encouraged: check yours with **Test All Rows** in the FSM Explorer's **Student Logic** card. Step 7 is in **software**: a `switch` on the state variable replaces the gates, and the variable replaces the flip-flops.

Wrap-Up

Key ideas:

- Seven steps: **encode** states/inputs/outputs, **diagram**, **table**, **equations**, **circuit**
- n states need $\lceil \log_2 n \rceil$ bits, and **one flip-flop per bit**, all on **one clock**
- Next-state logic is **combinational** — SOP off each next-state column, as in Objective 2.7
- **Always define the unused state codes** — an undefined code can lock the machine up

Next lesson:

- **Lesson 26 — Project 2 Workday 1.** The SEARCH / DETECT / LOCK engagement FSM, on the Arduino Due
- **Before class:** step the **Fire Control Radar** tab of the FSM Explorer (`usafa-ece.org/apps/fsm-demo/`); bring your **Lesson 22 sketch** (`CAL_K`, `CAL_B`), Due, sensor, and a DMM
- **Quiz 7** (Objectives 2.7–2.8) is due at **Lesson 27 (23/26 Oct)** — the **last new material before it**; Lessons 26 and 27 are workdays