

ECE 215/S – Principles of Electronic Cyber Warfare

Block 2: Signal Processing

Lesson 24: Finite State Machines I

Department of Electrical and Computer Engineering
United States Air Force Academy

Fall 2026

Objective 2.8

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 23 closed on a question:** what happens when the output depends on *history*, not just current inputs? Today is the answer
- This objective spans **two lessons** — today the model, Lesson 25 the implementation
- It is also the whole of **Project 2** — diagram and table required, equations optional (check yours with **Test All Rows** in the FSM Explorer's **Student Logic** card), circuit step in software
- Assessed on **Quiz 7** along with Objective 2.7

Where We Are Going

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

- Distinguish **combinational** from **sequential** logic
- Explain what a **finite state machine** is and when you need one
- Distinguish a **Moore** machine from a **Mealy** machine
- Read and draw a **state transition diagram**, and convert it to a **transition table**

Combinational vs Sequential

Combinational (Lesson 23)

- Output depends **only** on the current inputs
- Same inputs always give the same output
- Fully described by a truth table
- No memory

Sequential (today)

- Output depends on current inputs **and on history**
- Same inputs can give different outputs
- Requires **memory** of where you have been
- Described by a state machine

If the answer depends on what happened before, you need state.

A garage door button is sequential: pressing it opens or closes depending on where the door already is.

What Is a Finite State Machine?

An **FSM** is a model with a finite number of **states**, and rules for moving between them.

- **States** — the finite set of situations the system can be in
- **Inputs** — what the system observes
- **Transitions** — the rule taking (current state, input) to a next state
- **Outputs** — what the system does
- **Initial state** — where the machine starts, usually forced by a reset

The state *is* the memory. It captures everything the machine needs to know about the past.

The machine advances **one transition per clock tick**, and holds its state in between.

FSMs in Engineering

- **Traffic light** — states are the light colors; input is a timer or a sensor
- **Vending machine** — states track how much money has been inserted
- **Elevator controller** — state is the current floor and direction
- **Communication protocols** — idle, transmitting, waiting for acknowledgment, retrying
- **Weapon system modes** — safe, searching, locked, engaged. The transitions between them are safety-critical

FSMs appear wherever behavior is modal — and being explicit about the states is what makes that behavior verifiable.

Moore vs Mealy

Moore machine

- Output depends **only on the state**
- Output is written **inside** the state bubble
- Output changes only when the state changes
- Simpler to reason about and to verify

Mealy machine

- Output depends on **state and current input**
- Output is written **on the transition arrow**
- Output can change without a state change
- Often needs fewer states

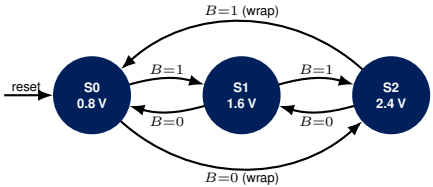
Output in the bubble means Moore. Output on the arrow means Mealy.

Project 2 carries **one output of each kind** — a telemetry voltage fixed by the state alone (**Moore**), and a FIRE command that also needs the operator's input (**Mealy**). You will build both.

State Transition Diagrams

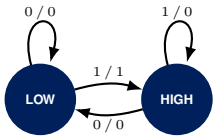
Here B is a **direction** input: $B = 1$ counts up, $B = 0$ counts down.

Moore — output inside the bubble



The same telemetry voltages you will use in Project 2 — but a different machine.

Mealy — output on the arrow



A rising-edge detector. Each arc reads **input / output** (X/Z): $Z = 1$ fires on the arrow, not in a state.

- Each **bubble** is a state, with a Moore output inside it; the arrow from nowhere is the **initial state**
- Each **arrow** is a transition, labeled with the input that causes it — and, on a Mealy machine, the output
- Every state needs an arrow for **every input value** — an undefined transition is a bug

State Transition Tables

The diagram is for humans. The **table** is what you compute with.

Current state	Input B	Next state	Output
S0	0	S2	0.8 V
S0	1	S1	0.8 V
S1	0	S0	1.6 V
S1	1	S2	1.6 V
S2	0	S1	2.4 V
S2	1	S0	2.4 V

- One row for **every (state, input) pair** — 3 states and 2 input values gives 6 rows. (Lesson 25 encodes the states in bits and counts over all 2^{bits} codes instead: 8 rows here, two of them for the unused code)
- Note the output column repeats per state — that is the signature of a **Moore** machine

Method: From Description to Transition Table

1. **Name the states.** One per situation the system must tell apart — no more
2. **Name the inputs, and list every value** each one can take
3. **Say what the output is**, and whether it depends on the input: state alone is **Moore**, state and input is **Mealy**
4. **Write one row per (state, input) pair** — count the rows before you start, then fill every one
5. **Check:** every state has a rule for **every** input value, and every table row appears as an arrow on the diagram

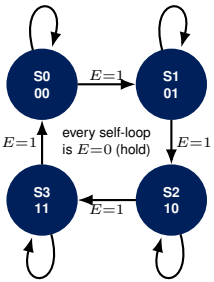
Draw the diagram and build the table from the same description — then check them against each other, not against the description twice.

A missing row is a missing arrow. That is where a state machine hangs.

Worked Example: A 2-Bit Counter

A counter counts $0 \to 1 \to 2 \to 3 \to 0$ when input $E = 1$, and holds when $E = 0$. Output is the count.

Diagram



Table

State	E	Next	Out
S0	0	S0	00
S0	1	S1	00
S1	0	S1	01
S1	1	S2	01
S2	0	S2	10
S2	1	S3	10
S3	0	S3	11
S3	1	S0	11

- 4 states $\times$ 2 input values $= 8$ rows. Count them before you start
- $E = 0$ is the **self-loop** — it returns to the same state, which is what “hold” means
- S3 with $E = 1$ **wraps** to S0. Forgetting the wrap is the classic error
- Output depends only on state $\Rightarrow$ **Moore**

Your Turn

Design an FSM for a **turnstile**. It has two states, **Locked** and **Unlocked**, and a single input with two values: **coin** and **push** — the turnstile sees one event per tick.

- Inserting a coin unlocks it. Pushing through a locked turnstile does nothing. Pushing through an unlocked one passes you and re-locks it

1. Draw the state transition diagram
2. Build the transition table — how many rows should it have?
3. Is this Moore or Mealy? Justify it

Wrap-Up

Key ideas:

- **Sequential** logic depends on history; **combinational** logic does not
- An FSM has **states**, **inputs**, **transitions**, **outputs**, and an **initial state** — and the state is the memory
- **Moore**: output in the bubble, depends only on state. **Mealy**: output on the arrow
- The transition table has one row per **(state, input) pair**. Count them first, and check every row against an arrow

Next lesson:

- **Lesson 25 — FSM implementation.** Turning the table into logic equations and real hardware — flip-flops and a clock do the holding
- **Quiz 6** (Objectives 2.5–2.6) is due tonight