

UNIT 5

Sequential Logic Design

(6 Hours) — Digital Logic

This unit formalises the design of sequential circuits: how a circuit's behaviour is modelled using the concept of state, how state diagrams capture that behaviour, how a state table can be reduced to its minimum number of states, and how the reduced design is finally realised in hardware using flip-flops — including the general procedure for counter design.

5.1 Basic Models of Sequential Machines

Unlike combinational circuits, whose outputs depend only on the present inputs, a sequential circuit's outputs depend on both the present inputs and the circuit's history — captured through the idea of “state”. A sequential machine is therefore formally described by five elements: a finite set of input symbols, a finite set of output symbols, a finite set of internal states, a next-state function, and an output function.

Concept of State

The state of a sequential circuit is a summary of all the relevant past input history that is needed, together with the present input, to determine the circuit's future behaviour. Physically, the state is stored in the circuit's flip-flops: an n -flip-flop circuit can represent up to 2^n distinct states.

Sequential machines are commonly classified into two standard models, based on how the output is generated:

- Moore machine: the output depends only on the present state, and is therefore associated with each state itself. Outputs change only at clock edges (when the state changes).
- Mealy machine: the output depends on both the present state and the current input, and is therefore associated with each state transition (arc). Outputs can change as soon as the input changes, even between clock edges.

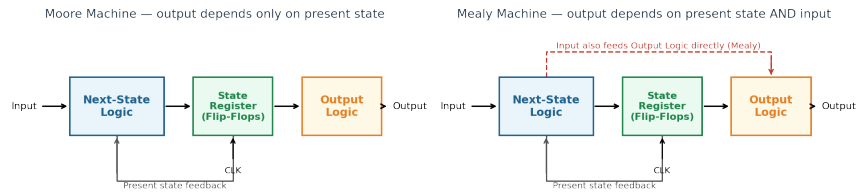


Fig. 5.1 — General block models of Moore and Mealy sequential machines

Feature	Moore Machine	Mealy Machine
Output depends on	Present state only	Present state + input
Output associated with	States (nodes)	Transitions (arcs)
Output timing	Changes only at clock edge	Can change between clock edges
Number of states	Often more	Often fewer for the same task
Circuit complexity	Slightly simpler output logic	Output logic depends on more variables

Note: Any Mealy machine can be converted to an equivalent Moore machine and vice versa; the choice between them is usually a design trade-off between state count and output timing behaviour.

State Diagram

A state diagram (state transition graph) is the standard graphical representation of a sequential machine. Each state is drawn as a circle (node), and each transition is drawn as a directed arc labelled with the input that causes it (and, for a Mealy machine, the output produced during that transition). For a Moore machine, the output associated with each state is written inside or beside its circle.

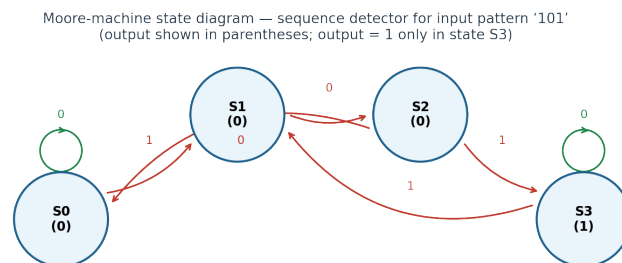


Fig. 5.2 — Example Moore-machine state diagram: a sequence detector that outputs 1 whenever the input pattern '101' has just been received

Example 5.1 — Reading the state diagram

Starting in state S_0 , input sequence $1 \rightarrow 0 \rightarrow 1$ drives the machine $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3$. State S_3 has output 1, so the detector correctly signals that the pattern '101' has just been seen. Any additional '0' input from S_3 returns to S_2 to check for an overlapping occurrence such as '10101'.

A state diagram is usually converted into an equivalent tabular form called the state table (or state transition table), which lists, for every present state and every possible input, the resulting next state and output. The state table is the starting point for all further analysis and design steps described in this unit.

Key Takeaways — Basic Models

- ✓ State summarizes the relevant past history needed to predict future behaviour.
- ✓ Moore machine: output depends on present state only (associated with nodes).
- ✓ Mealy machine: output depends on present state and input (associated with arcs).
- ✓ A state diagram graphically shows states, transitions, inputs and outputs.
- ✓ The state table is the equivalent tabular form used for design and reduction.

5.2 State Reduction through Partitioning and Implementation of Synchronous Sequential Circuits

A state table derived directly from a word description of a problem often contains more states than are strictly necessary. Since the number of flip-flops required grows with $\log_2(\text{number of states})$, removing redundant states reduces hardware cost. State reduction identifies and merges states that are equivalent.

Equivalent States

Two states S_i and S_j are said to be equivalent if, for every possible input sequence, they produce exactly the same output sequence and end up in equivalent next states. Formally, S_i and S_j are equivalent if, for every input symbol, (a) their outputs are identical, and (b) their next states are themselves equivalent (or identical).

The Partitioning Method

The partitioning method (also called row-matching or implication-table reduction) systematically finds all equivalent states by successive refinement:

- Step 1 — Initial partition P_0 : group all states that have identical outputs for every input into the same partition block. States with different outputs can never be equivalent, so they start in different blocks.
- Step 2 — Refine to P_1 : within each block of P_0 , further split states whose next states (for the same input) do not fall in the same block of P_0 . Two states stay together only if their next states for every input lie in the same P_0 block.
- Step 3 — Repeat: continue refining $P_1 \rightarrow P_2 \rightarrow \dots$ using the same rule, checking membership against the previous partition, until two successive partitions are identical ($P_k = P_{k+1}$).
- Step 4 — Merge: all states remaining together in the final partition are equivalent and can be merged into a single state; redraw the state diagram/table with the reduced state set.

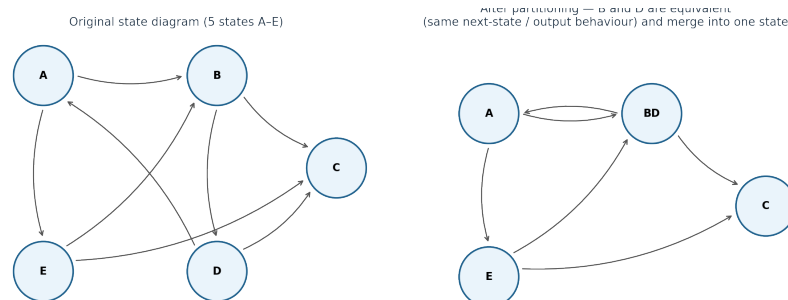


Fig. 5.3 — State reduction by partitioning: states B and D are found equivalent and merged, reducing 5 states to 4

Example 5.2 — Applying the partitioning method

States B and D in Fig. 5.3 both transition to C on input 0 and to A on input 1, and both are non-output states. Since their next states and outputs match for every input, B and D belong to the same equivalence class in every partition refinement and are therefore merged into a single state, reducing the design from 5 states to 4.

Implementation of Synchronous Sequential Circuits

Once the state table has been reduced to its minimum form, the sequential circuit is implemented using the following standard procedure:

- 1. State assignment: assign a unique binary code to each reduced state (using $m = \lceil \log_2(\text{number of states}) \rceil$ flip-flops). The choice of assignment affects circuit complexity and is often guided by rules such as keeping states that share next states 'close' in code.
- 2. Derive the transition/excitation table: for each present state and input, list the required flip-flop input values (from the flip-flop's excitation table) that will cause the desired next-state transition.
- 3. Simplify with Karnaugh maps: derive minimal Boolean expressions for each flip-flop's excitation inputs and for the output logic, in terms of the present-state variables and the external input.
- 4. Draw the logic circuit: implement the simplified equations with combinational gates feeding the flip-flop inputs, and connect the flip-flop outputs back as the present-state feedback.

Note: A poor state assignment can significantly increase the complexity of the next-state and output logic even though the number of states (and hence flip-flops) stays the same — state assignment is therefore an important design decision in its own right.

Key Takeaways — State Reduction & Implementation

- ✓ Equivalent states produce identical outputs and equivalent next states for every input.
- ✓ Partitioning refines state groups ($P_0 \rightarrow P_1 \rightarrow \dots$) until no further split occurs, revealing equivalence classes.
- ✓ Equivalent states are merged, reducing the state count and hence flip-flop count.
- ✓ Implementation: state assignment $\rightarrow$ excitation table $\rightarrow$ K-map simplification $\rightarrow$ logic circuit.
- ✓ State assignment choice affects circuit complexity even for the same number of states.

5.3 Use of Flip-Flops in Realizing the Models

Flip-flops are the physical memory elements that store the state of a Mealy or Moore machine between clock pulses. Realizing a sequential machine model in hardware means translating the abstract state table into a working circuit built from flip-flops plus combinational next-state and output logic.

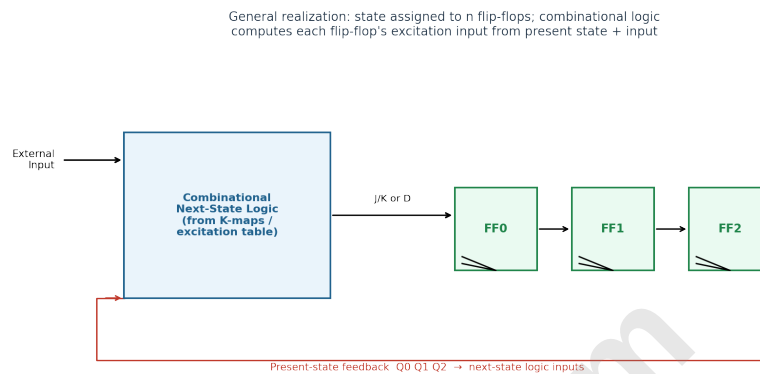


Fig. 5.4 — General flip-flop realization: combinational next-state logic drives the flip-flops, whose outputs feed back as present state

Choosing a Flip-Flop Type

Any of the standard flip-flop types (RS, JK, D, T) can be used to realize a given state table, but the choice affects the complexity of the resulting next-state logic:

- D flip-flops give the simplest design procedure — the required D input is always identical to the desired next-state bit, so $D = Q(\text{next})$ directly, with no separate excitation table lookup needed.
- JK flip-flops offer the most don't-care terms in their excitation table (see Unit 4, Table 4.1.2), which often leads to the most heavily simplified — and therefore cheapest — next-state logic after K-map minimisation.
- T flip-flops are convenient when a bit must simply toggle under certain conditions, such as in binary counters, since the required T input directly encodes “change state?” as a single bit.
- RS flip-flops are rarely chosen for new designs because of the invalid $S = R = 1$ condition, but the analysis method is identical to the other types.

Design Procedure Using Flip-Flops

Given a (reduced) state table and a chosen flip-flop type, the realization proceeds as follows:

- 1. Assign binary codes to the states and rewrite the state table in terms of present-state flip-flop values Q and next-state values $Q(\text{next})$.

- 2. For every row of the table, use the flip-flop's excitation table to determine the required flip-flop input values that force each $Q \rightarrow Q(\text{next})$ transition.
- 3. Treat each flip-flop input (e.g. $J_0, K_0, J_1, K_1, \dots$) as a separate Boolean function of the present-state variables and external input; plot each on a Karnaugh map and simplify.
- 4. Similarly derive and simplify the output logic — a function of present state only for Moore machines, or of present state and input for Mealy machines.
- 5. Draw the final circuit: flip-flops for the state register, combinational gates for the next-state and output logic, with outputs of the state register fed back as present-state inputs to the next-state logic.

Example 5.3 — Deriving a JK excitation equation

Suppose a 2-bit state register ($Q_1 Q_0$) must transition $00 \rightarrow 01 \rightarrow 10 \rightarrow 00$ on every clock pulse (a simple Mod-3 sequence, input-independent). Tabulating $Q_1 Q_0 \rightarrow Q_1(\text{next}) Q_0(\text{next})$ and reading off the JK excitation table for each bit position gives $J_0 = \bar{Q}_1, K_0 = 1, J_1 = Q_0, K_1 = 1$ after K-map simplification — a compact two-flip-flop realization.

Note: This is exactly the same excitation-table-driven method introduced for synchronous counter design in Unit 4 — a sequential machine realization is simply the general case, of which a counter is one specific (input-independent) example.

Key Takeaways — Flip-Flop Realization

- ✓ Flip-flops store the machine's state; combinational logic computes next-state and output values.
- ✓ D flip-flops give the simplest design ($D = \text{desired next state directly}$).
- ✓ JK flip-flops often give the most simplified logic due to abundant don't-care terms.
- ✓ T flip-flops suit toggle-based designs such as counters.
- ✓ Procedure: state assignment $\rightarrow$ excitation table $\rightarrow$ K-map simplification $\rightarrow$ circuit realization.

5.4 Counter Design

A counter is simply a special case of a synchronous sequential machine whose state diagram is a single cycle of states with no external input (or with a simple count/hold control input), and whose output is usually just the present state itself (a Moore machine with identity output). The general sequential-design procedure of Sections 5.2 and 5.3 therefore applies directly to counter design.

General Counter Design Procedure

- 1. State diagram: draw the required counting sequence as a cycle of states (e.g. $000 \rightarrow 001 \rightarrow 010 \rightarrow \dots \rightarrow \text{back to } 000$).
- 2. State table: tabulate each present state and its required next state (there is normally no external data input, only the clock, though an optional Count/Hold or Up/Down control input may be included).
- 3. Excitation table: using the chosen flip-flop type's excitation table, determine the required flip-flop inputs for every present-state $\rightarrow$ next-state transition.
- 4. K-map simplification: simplify each flip-flop's excitation equation as a function of the present-state bits (and control input, if any).
- 5. Circuit realization: implement the simplified equations with gates driving the flip-flop inputs, exactly as in Fig. 5.4.

Example 5.4 — Designing a Mod-6 synchronous up-counter (JK)

Required sequence: $000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 101 \rightarrow (\text{back to}) 000$.

Step 1–2: 6 states need 3 flip-flops ($Q_2 Q_1 Q_0$); state 110 and 111 are unused (don't-care) states.

Step 3: build the JK excitation table for all 6 valid transitions, using 'X' for the two unused states.

Step 4 (result after K-map simplification): $J_0=K_0=1$; $J_1=Q_0$, $K_1=Q_0$; $J_2=Q_1 \cdot Q_0$, $K_2=1$.

Step 5: realize with three JK flip-flops and the two small AND gates required for J_1 and J_2 — a compact 3-flip-flop, 6-state counter.

Note: Unused states (110, 111 in Example 5.4) should ideally also be checked for self-correction — i.e. verifying that if the counter is accidentally powered up in an unused state, it eventually returns to the valid counting cycle rather than getting stuck in an unused loop.

Relationship to Modulus-N Counter Design

The reset-based Mod-N design technique introduced in Unit 4 (Section 4.2.4) — using $\lceil \log_2 N \rceil$ flip-flops plus an asynchronous reset gate that detects state N — is a simpler, lower-cost alternative to the full synchronous excitation-table method. The reset method needs less design effort and fewer gates, but produces a brief invalid glitch state and is asynchronous in its reset

action; the excitation-table method of this section produces a fully synchronous circuit with guaranteed-clean transitions and optional self-correction on unused states, at the cost of extra design steps and typically extra gates.

Design method	Design effort	Timing	Self-correcting?
Reset-based Mod-N (Unit 4)	Low	Reset is asynchronous	No (needs manual check)
Excitation-table method (Unit 5)	Higher (K-maps per flip-flop)	Fully synchronous	Can be designed in

Key Takeaways — Counter Design

- ✓ A counter is a sequential machine whose state diagram is a single counting cycle.
- ✓ Design procedure: state diagram → state table → excitation table → K-map → circuit.
- ✓ Unused states (when $2^n > \text{modulus}$) should be checked for self-correction.
- ✓ The reset-based method (Unit 4) trades design effort for a simpler, asynchronous circuit.
- ✓ The excitation-table method (Unit 5) gives a fully synchronous, self-correcting design.

— *End of Unit 5* —