6.8 Synthesis of Sequential Logic

Steps:

1. Given a description (usually in words), develop the state diagram.
2. Convert the state diagram to a next-state / output tables.
3. Minimize the number of states.
4. Encode inputs, states, and outputs. Assign different *n*-tuples of its flip-flop values to the states.
5. Generate the binary form of the next-state and output equations.
6. Choose memory elements (flip-flop types).
7. Derive excitation equations for each flip-flop input.
8. Optimize the logic implementation of the excitation and output equations.
9. Draw logic schematic that serves as a basis for the generation of a timing diagram.
10. Simulating the logic schematic.
11. Verify the functionality and timing.

6.9 FSM Model Capture

Step 1. Generate a state diagram from a description.

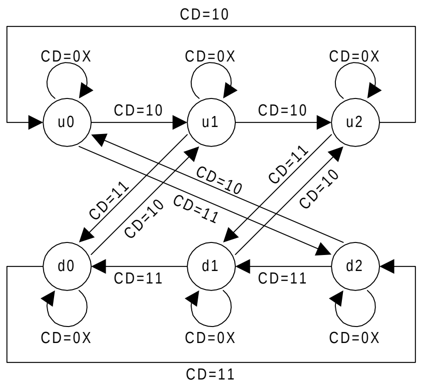
Example 6.4: Derive the state diagram for a modulo-3 up/down-counter. The counter has two inputs: count enable (C) and count direction (D). When C=1, the counter will count in the direction specified by D, and it will stop counting when C=0. The counter will count up when D=0 and down when D=1. The counter has one output Y, which will be asserted when the counter reaches 2 while counting up or when it reaches 0 while counting down.

Solution: Requires at least 2 ffs, since it must remember three digits: 0, 1, and 2.

Need two sequences: up sequence with three states (u0, u1, and u2) and down sequence with three states (d2, d1, d0). As long as CD = 10, the counter counts from u0, u1, u2 and returning to u0. When CD = 11, the counter proceeds from d0 to d2 to d1 and returns to d0.

Need to account for the possibility that the counter might change direction while it is counting.

Need to consider when the counter is disabled by a change of C from 1 to 0.



Although this state diagram is complete, it does not contain the minimal number of states.

6.10 State Minimization

The purpose of state minimization is to reduce the number of states in a sequential circuit so that the circuit requires fewer flip-flops.

Reducing a FSM from six states originally to five states will not reduce the number of flip-flops.

State minimization is based on the concept of the **behavioral equivalence** of FSMs.

We say that two FSMs are equivalent if they produce the same sequence of output symbols for every sequence of input symbols. They may have different number of states and may also transition through a different sequence of states for every input sequence.

We can reduce the number of states in the FSM by merging those states that are equivalent.

State equivalence. Two states, s_j and s_k, in an FSM are said to be equivalent, $s_j \equiv s_k$, iff the following two conditions are true.

1. Both states s_j and s_k produce the same output symbol for every input symbol i: that is, $h(s_j, i) = h(s_k, i)$.
2. Both states have equivalent next states for every input symbol i: that is, $f(s_j, i) = f(s_k, i)$.

State minimization procedure requires partitioning all the states in an FSM into equivalence classes and constructing the minimal-state FSM in which each state will represent one equivalence class.

Example 6.5: Derive the minimal-state FSM for the modulo-3 counter.
Solution:

Step 1. Start with the next-state / output table obtained from the state diagram of example 6.4.

Present State	Next State		
	CD = 0x	CD = 10	CD = 11
u0	u0 / 0	u1 / 0	d2 / 1
u1	u1 / 0	u2 / 0	d0 / 0
u2	u2 / 0	u0 / 1	d1 / 0
d0	d0 / 0	u1 / 0	d2 / 1
d1	d1 / 0	u2 / 0	d0 / 0
d2	d2 / 0	u0 / 1	d1 / 0

(a) Initial state / output table

Step 2. Determine the output values for each combination of input values and states.

Output values	u0	u1	u2	d0	d1	d2
CD=0x	0	0	0	0	0	0
10	0	0	1	0	0	1
11	1	0	0	1	0	0

(b) Output values

Step 3. Combine states into groups in such a way that all states in the same group generate the same output symbol for each input symbol.

001 000 010

Step 4. Determine the next state for each state in the groups and for every input symbol.

Next State	001 G_0={u0,d0}	000 G_1={u1,d1}	010 G_2={u2,d2}
CD=0x	G_0 G_0	G_1 G_1	G_2 G_2
10	G_1 G_1	G_2 G_2	G_0 G_0
11	G_2 G_2	G_0 G_0	G_1 G_1

$\leftarrow$ Step 3. The column labels.

$\leftarrow$ Step 4. The table entries.

(c) Partitioning into equivalence classes.

Step 5. Since each group represents a class of equivalent states, we can rename the groups (G_0, G_1, G_2) as states (s_0, s_1, s_2).

Present State	Next State		
	CD = 0x	CD = 10	CD = 11
s0	s0 / 0	s1 / 0	s2 / 1
s1	s1 / 0	s2 / 0	s0 / 0
s2	s2 / 0	s0 / 1	s1 / 0

(d) Final next-state / output table

6.11 State Encoding

To determine how many binary variables are required to represent the states in the state table, and to assign a specific combination to each named state.

The total number of states in a machine with n flip-flops is 2^n, so the number of flip-flops needed to code s states is $\lceil \log_2 s \rceil$, the smallest integer greater than or equal to $\log_2 s$.

For example, a four-state FSM with states s0, s1, s2, and s3 could be implemented with two flip-flops that contain values 00, 01, 10, or 11. In this case, there would be 4! = 24 possible encodings of the four states to flip-flop values.

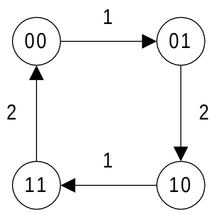
#	s0	s1	s2	s3
1	00	01	10	11
2	00	01	11	10
3	00	10	01	11
…				
24	11	10	01	00

The simplest assignment of s coded states to 2^n possible states is to use the first s binary integers in binary counting order. However, this does not always lead to the simplest excitation equations, output equations, and resulting logic circuit.

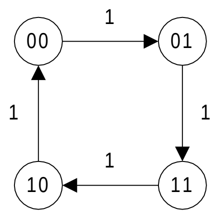
Three most popular heuristics: **minimum bit change**, **prioritized adjacency**, and **one-hot** encoding.

Minimum bit change – assigns Boolean values to the states in such a way that the total number of bit changes for all state transitions is minimized. In other words, if every arc in the state diagram has a weight that is equal to the number of bits by which the source and destination encodings differ, this strategy would select the one that minimizes the sum of all these weights.

Example:

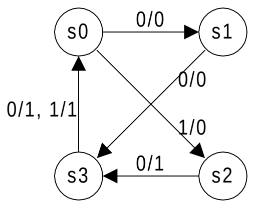


Encoding with 6 bit changes.



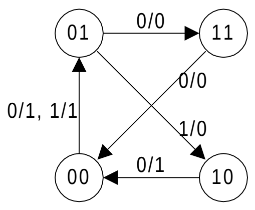
Minimum-bit change encoding with only 4 bit changes.

Prioritized adjacency – assigns adjacent encodings, i.e. encodings which differ in one bit only, to all states that have (in the following order of priority):
priority 1: a common destination – i.e. states that have the same next state for a given input value.
priority 2: a common source – i.e. next states of the same state.
priority 3: a common output – i.e. states that have the same output value for the same input values.



Initial state diagram

Priority 1: (s1, s2) – the input value of 0 will move both states into the same state s3.
Priority 2: (s1, s2) – they are both next states of the state s0.
Priority 3: (s0, s1), and (s2, s3) – states s0 and s1 have the same output value 0 for the same input value 0.



One possible encoding

One-hot encoding – uses redundant encoding in which one flip-flop is assigned to each state. i.e. each state is distinguishable by its own flip-flop having a value of 1 while all others have a value of 0.

Best encoding strategy. To determine the encoding with the minimum cost and delay, we need to:

1) generate Karnaugh maps for next-state and output functions.
2) derive excitation equations from the next-state map.
3) derive output equations from the output functions map.
4) implement above equations using two-level NAND gates, ignoring a variable's true and complemented values.
5) calculate cost and delay.

Example 6.7: Using the following next-state / output table for the modulo-3 counter and the three encoding schemes, find the best encoding scheme:

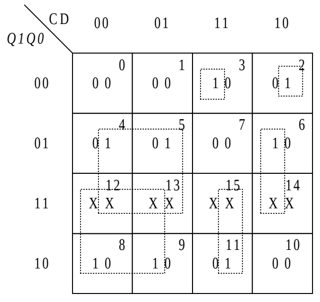
Present State	Next State		
	CD = 0x	CD = 10	CD = 11
s0	s0 / 0	s1 / 0	s2 / 1
s1	s1 / 0	s2 / 0	s0 / 0
s2	s2 / 0	s0 / 1	s1 / 0

Next-state / output table

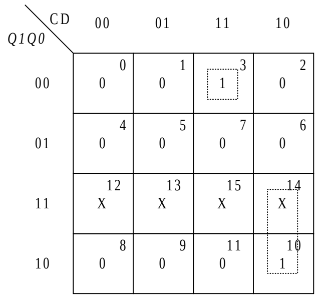
State	Min bit change	Prioritized Adj	One-hot
	$Q_1\ Q_0$	$Q_1\ Q_0$	$Q_1\ Q_0$
s0	0 0	0 1	0 0 1
s1	0 1	0 0	0 1 0
s2	1 0	1 0	1 0 0

Possible state encodings

1) Next-state map and output function map for the minimum bit change encoding strategy.



$Q_{1(\text{next})}$, $Q_{0(\text{next})}$



Y

Ex. If current state is s0=00 and the input CD=11, then the next state is 10=s2. Thus, for the minimum-bit change encoding, s0=00 and s2=10. So we have the entry 10 in row 00 and column 11.

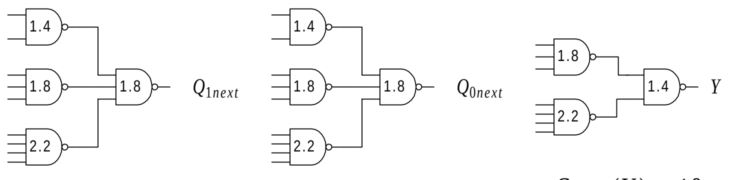
2&3) Excitation and output equations.

$$Q_{1(\text{next})} = Q_1C' + Q_0CD' + Q_1'Q_0'CD$$

$$Q_{0(\text{next})} = Q_0C' + Q_1CD + Q_1'Q_0'CD'$$

$$Y = Q_1CD' + Q_1'Q_0'CD$$

4) Implementation using NAND gates.



Cost (Q1) = 24
Delay (Q1) = 4.0

Cost (Q0) = 24
Delay (Q0) = 4.0

Cost (Y) = 18
Delay (Y) = 3.6

5) Cost and delay

	Encoding A	Encoding B	Encoding C
Total cost	24+24+18 = 66	24+24+16 = 64	22+22+22+16 = 82
Maximum input delay	max(4.0,4.0) = 4.0ns	max(4.0,4.0) = 4.0ns	max(3.6, 3.6, 3.6) = 3.6
Output delay	3.6ns	3.2ns	3.2ns
Comment	easy to encode	least expensive	fastest, most expensive

6.11 Choice of Memory Elements

We now choose the proper type of flip-flop for implementation of the state encodings.
As we know, there are four types of flip-flops:

Flip-Flop	Usage	Advantage	Disadvantage
T	counter-type circuits in which the flip-flops must flip from 0 to 1 and back with great frequency	- requires fewer connections than SR & JK - better suited for VLSI implementation	
D	applications where input data must be stored for some time and then used later	- requires fewer connections than SR & JK - better suited for VLSI implementation	
SR	situations where different signals set and reset the flip-flops	- most useful - reduce the cost of the input logic	- requires twice as many connections as T and D ffs
JK	whenever we need to combine the behavior of a T and an SR flip-flop	- most useful - reduce the cost of the input logic	- requires twice as many connections as T and D ffs

Example 6.8: Given the modulo-3 counter with encoding A, select the type of flip-flop that will minimize the cost and/or delay of the input logic.

Step 1. Write down the excitation tables for the various flip-flops. Following example is for the JK flip-flop.

Start with the truth table.

J	K	Q_{next}
0	0	Q
0	1	0
1	0	1
1	1	Q’

Expand to the characteristic table.

Q	J	K	Q_{next}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

JK characteristic table

Switch columns, combine the don’t cares, and delete duplicate rows to form excitation table.

Q	Q_{next}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK excitation table

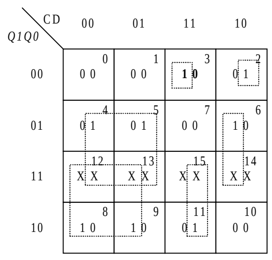
The excitation tables for the SR, JK, T, & D flip-flops are summarized below:

Q	Q_{next}	S	R	J	K	T	D
0	0	0	X	0	X	0	0
0	1	1	0	1	X	1	1
1	0	0	1	X	1	1	0
1	1	X	0	X	0	0	1

Flip-Flop excitation table

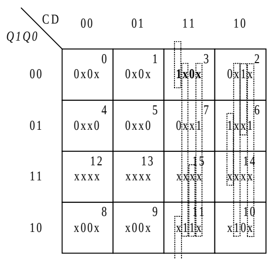
Step 2. Generate the **implementation map**. Take each pair of present and next states from the next-state map and replace their next-state values with the required input values from the excitation table.

The next-state map from example 6.7 is duplicated here on the left:



$Q_{1(\text{next})}$, $Q_{0(\text{next})}$
Next-state map.

e.g. From the next-state map, the entry for CD=11 and Q_1=0 is $Q_{1(\text{next})}$=1. Therefore, we look up $Q_1Q_{1(\text{next})}$=01 in the excitation table to get JK=1X. Thus, the implementation with JK ff for that entry is 1X.



J_1, K_1 J_0, K_0
JK implementation map.

Step 3. From the implementation map, we can derive minimal expressions for that flip-flop type.

$$
\begin{aligned}
J_1 &= Q_0CD' + Q_0'CD \\
&= Q_0CD' + Q_0'CD + Q_0'Q_0C + CD'D \\
&= C(Q_0' + D')(Q_0 + D) \quad \text{// DeMorgan} \\
&= C(Q_0D)'(Q_0'D')' \quad \text{// DeMorgan} \\
&= (C' + Q_0D + Q_0'D')' \quad \text{// DeMorgan} \\
K_1 &= C \\
J_0 &= Q_1CD + Q_1'CD' \\
&= Q_1CD + Q_1'CD' + Q_1'Q_1C + CD'D \\
&= C(Q_1 + D')(Q_1' + D) \quad \text{// DeMorgan} \\
&= C(Q_1'D)'(Q_1D')' \quad \text{// DeMorgan} \\
&= (C' + Q_1'D + Q_1D')' \quad \text{// DeMorgan} \\
K_0 &= C
\end{aligned}
$$

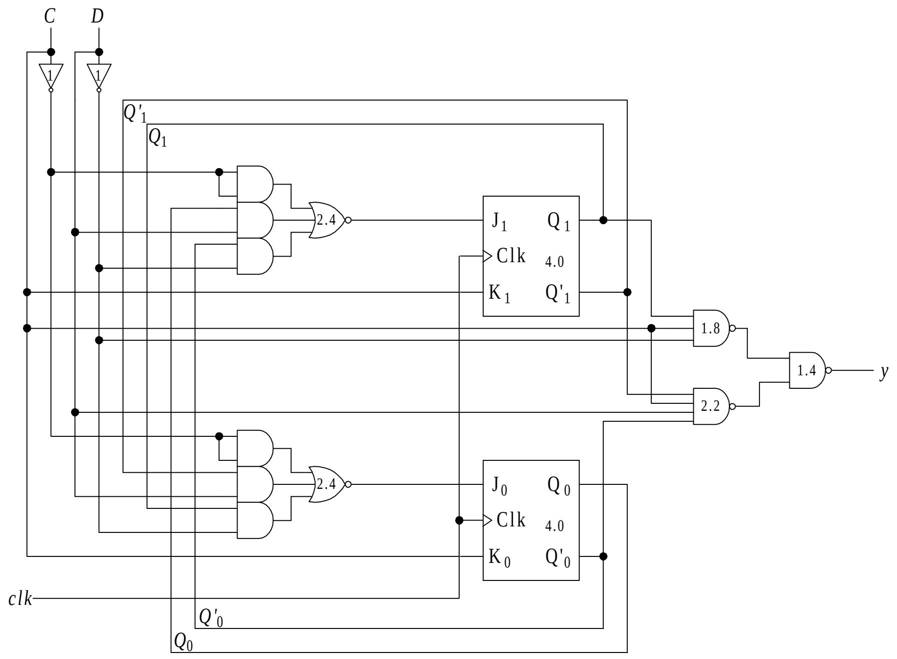
Step 4. Finally, cost and delay can be calculated from the equations obtained from step 3 above.

Mapping

The next step in a sequential logic synthesis would consist of mapping the input and output logic to the components in the given library.

Schematic Drawing

After mapping, we can draw the schematic to visualize all the counter's gates and connections. The following is the logic schematic for the JK implementation.



6.13 Optimization and Timing

The final step in the process of sequential synthesis consists of deriving a timing diagram from the schematic and the given gate and flip-flop delays.