# EEE205 – Digital Electronics (II)

# Lecture 19

A Revision Class

Dr. Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

## Reference Books

1. A. Marcovitz, *Introduction to Logic Design*, McGraw Hill, 2005. (M)
2. R. Tocci, N. Widmer and G. Moss, *Digital Systems: Principles and Applications*, 10th Edition, Pearson Education, 2007. (T)
3. T. Floyd, *Digital Fundamentals*, Pearson Education, 7th Edition, 2000. (F)
4. C. H. Roth, Fundamentals of Logic Design, 5th Edition, Thomson Learning, 2004. (R)
5. S. Brown and Z. Vranesic, Fundametals of Digital Logic with Verilog Design, 3rd Edition, McGraw-Hill Education, 2014. (B)

# Reference Books

| | | | | | |
|---|---|---|---|---|---|
| Lecture 1 | F | Lecture 11 | F | Statistics | |
| Lecture 2 | F | Lecture 12 | R | T | 6.5 |
| Lecture 3 | B | Lecture 13 | B, R | R (M) | 4.5 |
| Lecture 4 | F, T | Lecture 14 | UoL | F | 3.5 |
| Lecture 5 | T | Lecture 15 | T | B | 1.5 |
| Lecture 6 | T | Lecture 16 | T | | |
| Lecture 7 | T | Lecture 17 | T | | |
| Lecture 8 | R | | | | |
| Lecture 9 | R | | | | |
| Lecture 10 | R | | | | |

## This Module

**Contents**:

1. Programmable Logics Devices (PLDs)
2. Hardware Design Languages (HDLs)
3. Large Combinational and Sequential Circuits
4. Algorithmic State Machines (ASMs)
5. Processor Interface Circuits

## Programmable Logic Devices (PLDs)

Types

- SPLDs — Simple Programmable Logic Devices,
- CPLDs — Complex Programmable Logic Devices,
- FPGAs — Field Programmable Gate Arrays.

SPLDs and CPLDs are based on AND-OR *arrays*, which are ideally for SOP and POS expressions.

FPGAs are based on LUTs (Look-Up Tables), which are ideally for truth tables.

## SPLDs

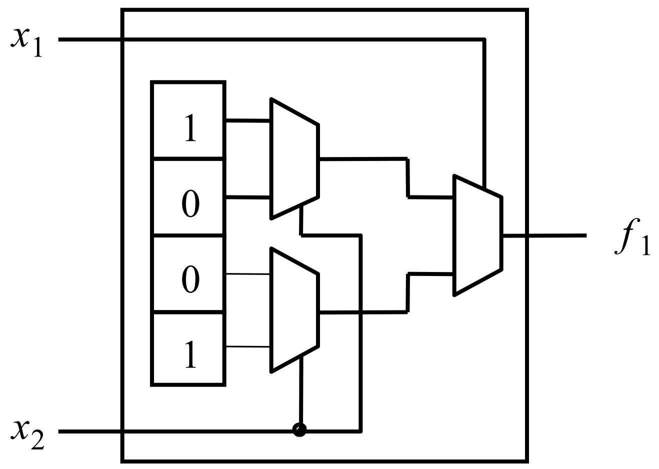
Types

- Programmable Read-Only Memory (**PROM**) consists of a fixed AND array and a programmable OR array.
- Programmable Logic Array (**PLA**) consists of a programmable AND array and a programmable OR array.
- Programmable Array Logic (**PAL**) consists of a one-time programmable AND array and a fixed OR array with output logic.
- Generic Array Logic (**GAL**) consists of a reprogrammable AND array and a fixed OR array with programmable output logic.

# Field Programmable Gate Arrays (FPGA)

| $x_1$ | $x_2$ | $f_1$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

$$f_1 = \bar{x}_1\bar{x}_2 + x_1x_2$$



- First decide the truth table.
- An LUT contains storage cells. The output values are copied to the storage cells in the same order.
- The outputs of such cells are multiplexed with the input variables as the data select. The MSB controls the last MUX. The LSB controls the first MUX.

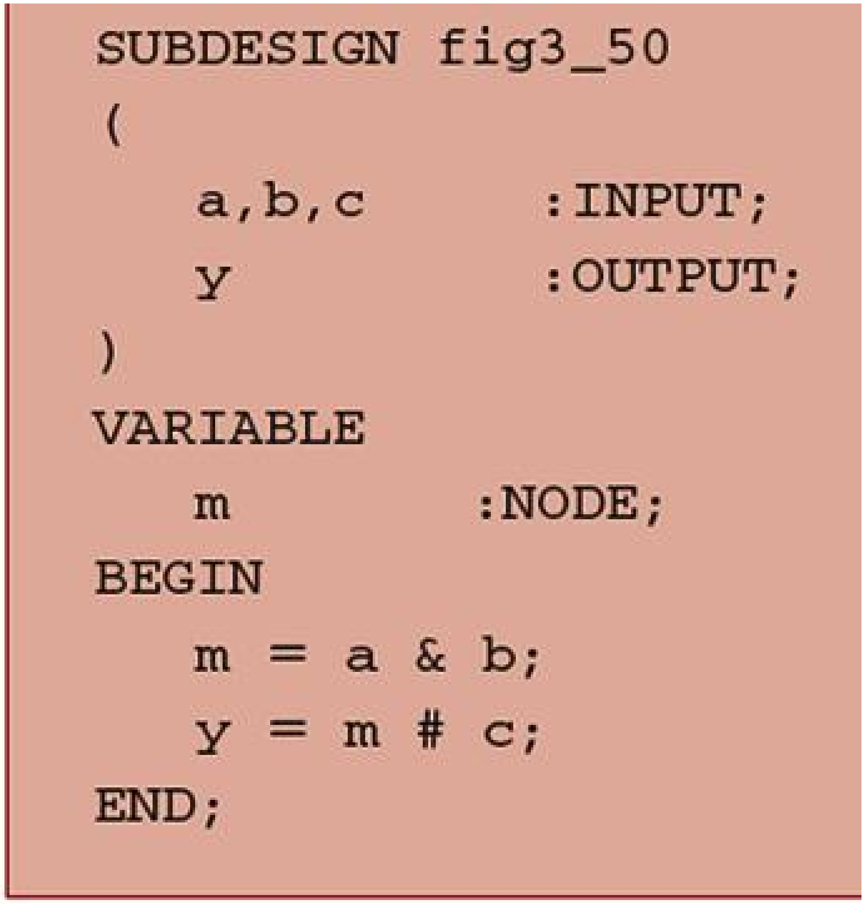
# AHDL Overview

AHDL vs. Programming Languages

- The statements are evaluated constantly and concurrently (at the same time). The order in which they are listed makes no difference.

- In the contrast, a computer programming language follows instructions in sequential order. Each instruction is processed once unless a loop is being used.

# AHDL Overview



```
SUBDESIGN fig3_50
(
    a,b,c         :INPUT;
    y             :OUTPUT;
)
VARIABLE
    m           :NODE;
BEGIN
    m = a & b;
    y = m # c;
END;
```

Format of AHDL files:

- Start with SUBDESIGN

1. I/O definitions
2. VARIABLE section used to define intermediate nodes and devices using primitives
3. Functional description: state description or behaviour description

## Synchronous Counters

### Behavioral Description – A Full Feature Counter

```
SUBDESIGN fig7_42
(
   clock, clear, load, cntenabl, down, din[3..0]      :INPUT;
   q[3..0], term_ct :OUTPUT;  -- declare 4-bit array of output bits
)
VARIABLE
   count[3..0]    :DFF;       -- declare a register of D flip flops

BEGIN
   count[].clk = clock;       -- connect all clocks to synch source
   count[].clrn= !clear;      -- connect for asynch active HIGH clear
   IF load THEN count[].d = din[]; -- synchronous load
      ELSIF !cntenabl THEN count[].d = count[].q; -- hold count
      ELSIF !down THEN count[].d = count[].q + 1; -- increment
      ELSE count[].d = count[].q - 1;            -- decrement
   END IF;
   IF ((count[].q == 0) & down # (count[].q == 15) & !down)& cntenabl
   THEN     term_ct = VCC;    -- synchronous cascade output signal
   ELSE term_ct = GND;
   END IF;
   q[] = count[];             -- transfer register contents to outputs
END;
```

# Synchronous Counters

- For behaviour description, always use DFFs for sequential circuits.
- “.q” is the present state, while “.d” is the next state.
- The order of the controls in the IF……ELSEIF structure reflects the priority order of these control signals. The first control to check has the highest priority.
- Don’t use non-plain text characters or C programming language.
- For falling-edge triggered FFs, the input is sometimes inverted with !.

## A Revision of Some Terminology

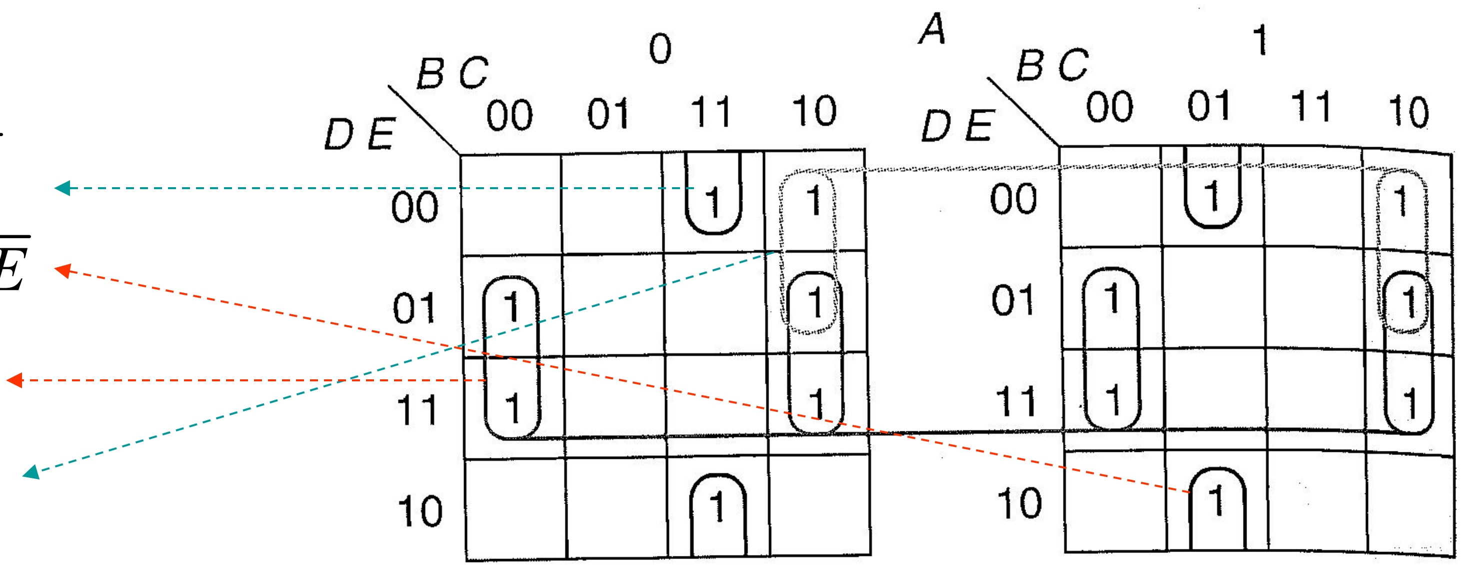
| | Boolean Algebra View | Karnaugh Map View |
|---|---|---|
| Minterm | a product term with all variables | a 1 cell |
| Implicant | a product term | a rectangle of $2^n$ 1's |
| Prime Implicant | a shortest product term | a group of 1's that is not fully contained in another group of 1's |
| Essential Prime Implicant | product terms that must appear in a minimized SOP | has at least a 1 not shared with others |

# Karnaugh Maps

- Boolean functions with five variables can be simplified using **two** 4-variable maps.
- Squares directly above or below each other are adjacent.

$$F(A,B,C,D,E) = \sum m(1,3,8,9,11,12,14,17,19,20,22,24,25,27)$$

$$F = \overline{A}BC\overline{E} + AB\overline{C}\overline{E} + \overline{C}E + B\overline{C}\overline{D}$$

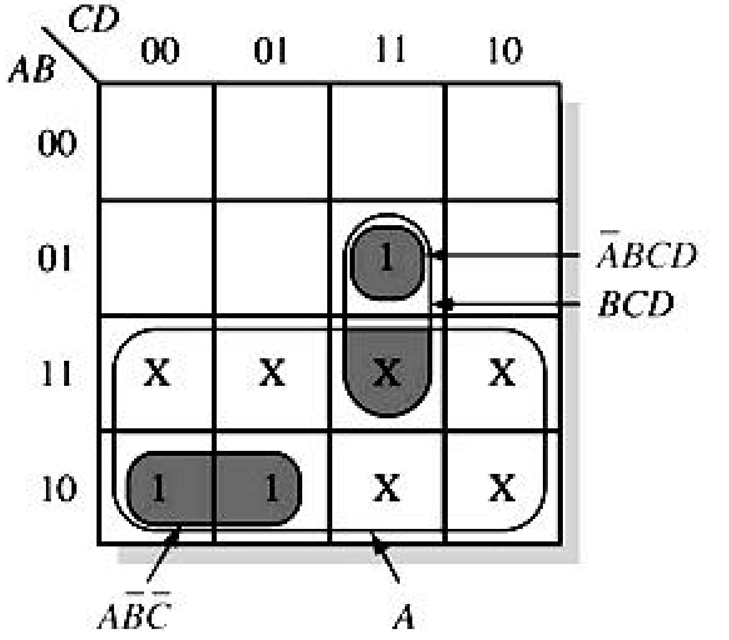


## Karnaugh Maps

Pitfalls

- Variable values should be sorted in 00, 01, **11**, 10.
- Variables are sorted according to alphabetical order or magnitude, e.g. AB/C, WX/YZ, $Q_2Q_1/Q_0$
- 1's groups are not fully maximized - forget warp-around adjacency.
- Don't know how to use don't cares.

## Karnaugh Maps



(b) Without "don't cares" $Y = A\bar{B}\bar{C} + \bar{A}BCD$
With "don't cares" $Y = A + BCD$

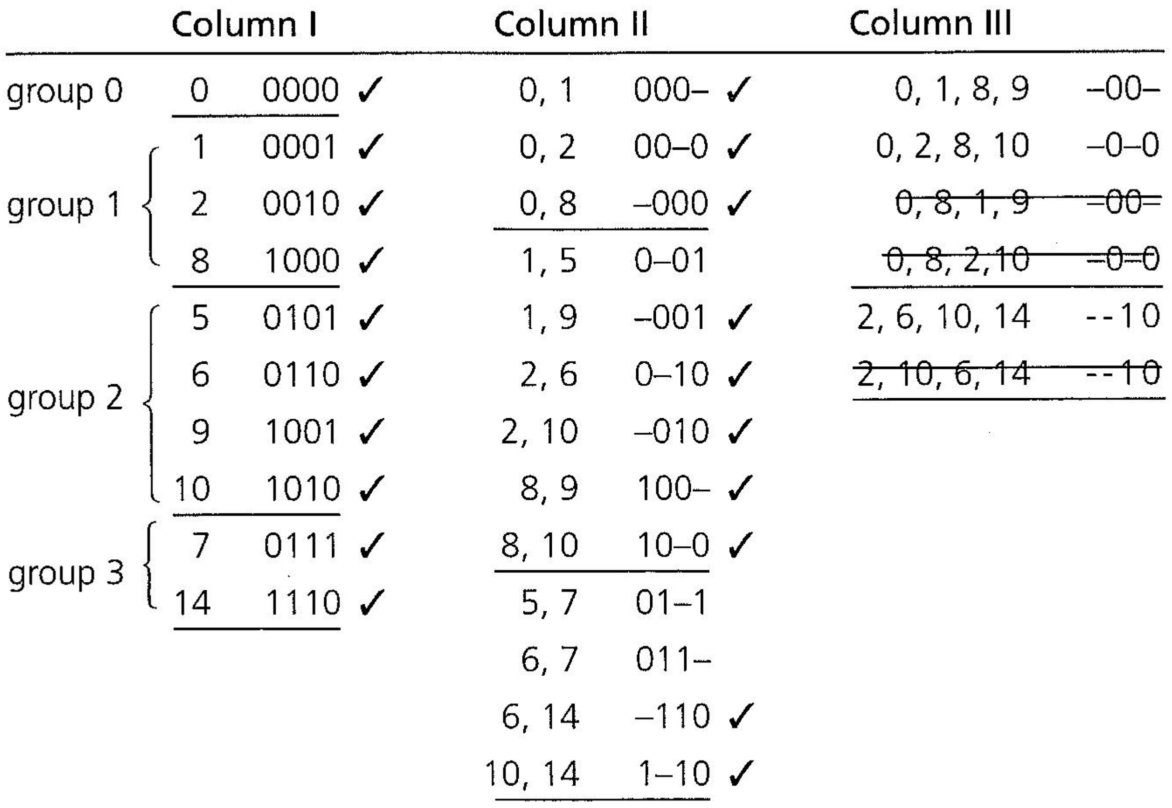
**"Don't Care"**

- Can be used to simplify Boolean expressions.
- When an X can be grouped with 1s, then it is thought as 1.
- Otherwise, it is thought as 0.

# Quine-McCluskey Method

## Step 1: Finding prime implicants

- Sort the product terms into groups according to the number of 1’s.
- Terms which have dashes in the same place and differ in only one variable can be combined.

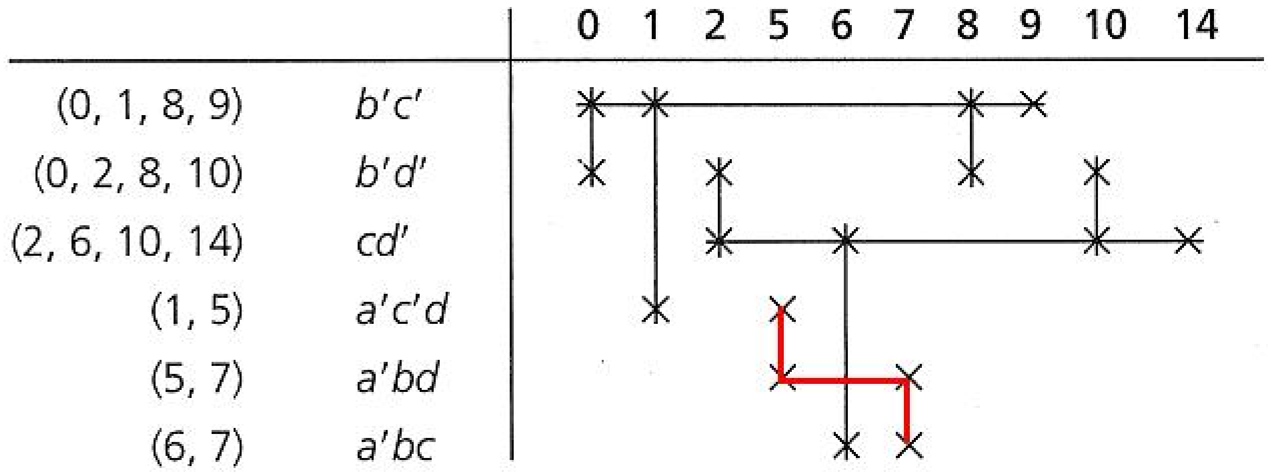


| | Column I | | Column II | | Column III | |
|---|---|---|---|---|---|---|
| group 0 | 0 | 0000 ✓ | 0, 1 | 000– ✓ | 0, 1, 8, 9 | –00– |
| group 1 | 1 | 0001 ✓ | 0, 2 | 00–0 ✓ | 0, 2, 8, 10 | –0–0 |
| | 2 | 0010 ✓ | 0, 8 | –000 ✓ | ~~0, 8, 1, 9~~ | ~~–00–~~ |
| | 8 | 1000 ✓ | 1, 5 | 0–01 | ~~0, 8, 2, 10~~ | ~~–0–0~~ |
| group 2 | 5 | 0101 ✓ | 1, 9 | –001 ✓ | 2, 6, 10, 14 | - -10 |
| | 6 | 0110 ✓ | 2, 6 | 0–10 ✓ | ~~2, 10, 6, 14~~ | ~~- -10~~ |
| | 9 | 1001 ✓ | 2, 10 | –010 ✓ | | |
| | 10 | 1010 ✓ | 8, 9 | 100– ✓ | | |
| group 3 | 7 | 0111 ✓ | 8, 10 | 10–0 ✓ | | |
| | 14 | 1110 ✓ | 5, 7 | 01–1 | | |
| | | | 6, 7 | 011– | | |
| | | | 6, 14 | –110 ✓ | | |
| | | | 10, 14 | 1–10 ✓ | | |

# Quine-McCluskey Method

Step 2: Prime implicant chart.

- The essential prime implicants are chosen first.
- Then additional non-essential prime implicants are selected by trial.
- They should cover as many minterms as possible.



| | | 0 | 1 | 2 | 5 | 6 | 7 | 8 | 9 | 10 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (0, 1, 8, 9) | $b'c'$ | × | × | | | | | × | × | | |
| (0, 2, 8, 10) | $b'd'$ | × | | × | | | | × | | × | |
| (2, 6, 10, 14) | $cd'$ | | | × | | × | | | | × | × |
| (1, 5) | $a'c'd$ | | × | | × | | | | | | |
| (5, 7) | $a'bd$ | | | | × | | × | | | | |
| (6, 7) | $a'bc$ | | | | | × | × | | | | |

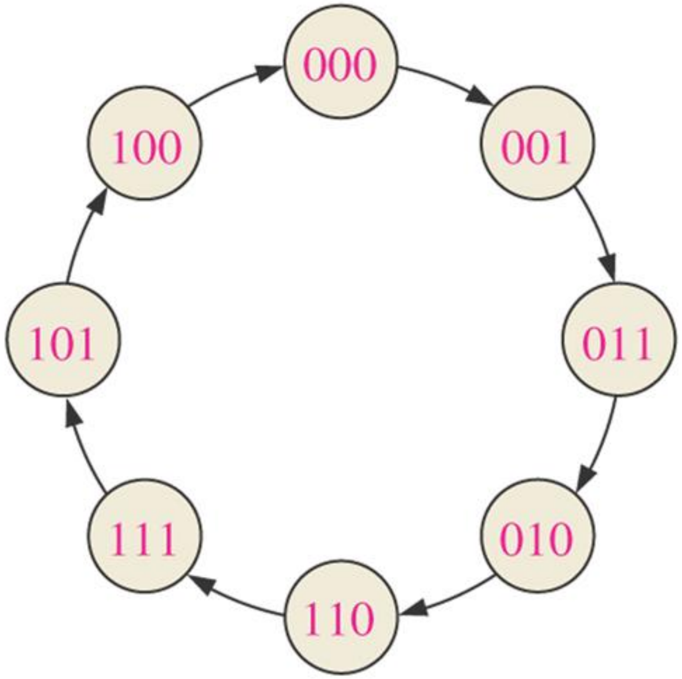
## Quine-McCluskey Method

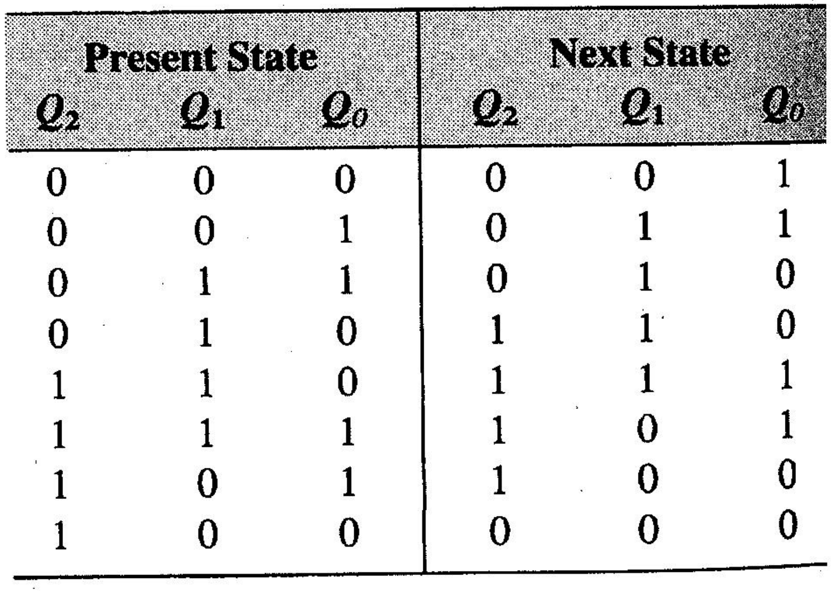
Pitfalls:

- In finding the prime implicants, the don’t cares are treated as minterms.
- When forming the prime implicant chart, the don’t cares should not be listed at the top.
- All alternative solutions must be provided.
- Frequently miss some combinations or forget column three.

## Design of Synchronous Counters

Step 1: State diagram Step 2: Next-state table



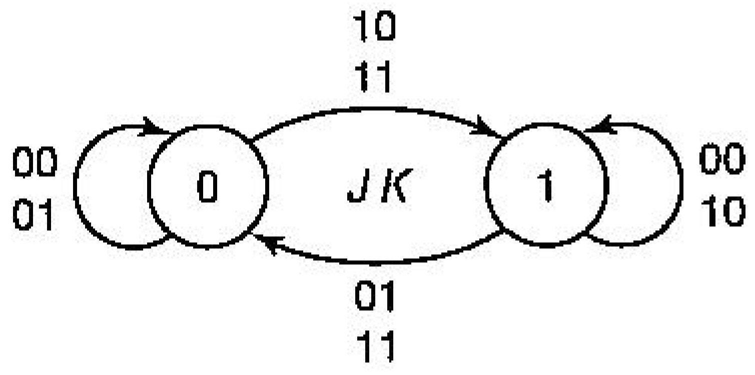


| Present State | | | Next State | | |
|---|---|---|---|---|---|
| $Q_2$ | $Q_1$ | $Q_0$ | $Q_2$ | $Q_1$ | $Q_0$ |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |

# Design of Synchronous Counters

## Step 3: Flip-flop transition tables and equations

*JK* flip flop state diagram.

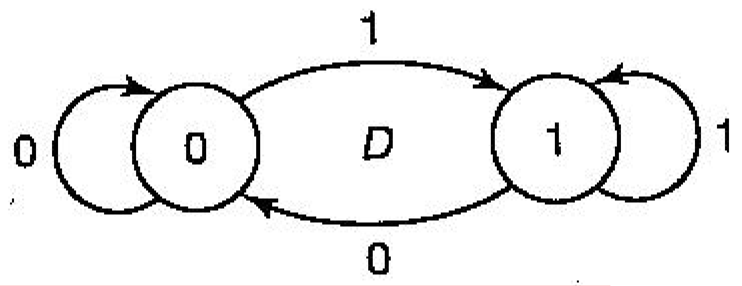


J-K flip-flop $Q^+ = JQ' + K'Q$

*JK* flip flop design table.

| $q$ | $q^*$ | $J$ | $K$ |
|---|---|---|---|
| 0 | 0 | 0 | X |
| 0 | 1 | 1 | X |
| 1 | 0 | X | 1 |
| 1 | 1 | X | 0 |

*D* flip flop state diagram



D flip-flop $Q^+ = D$

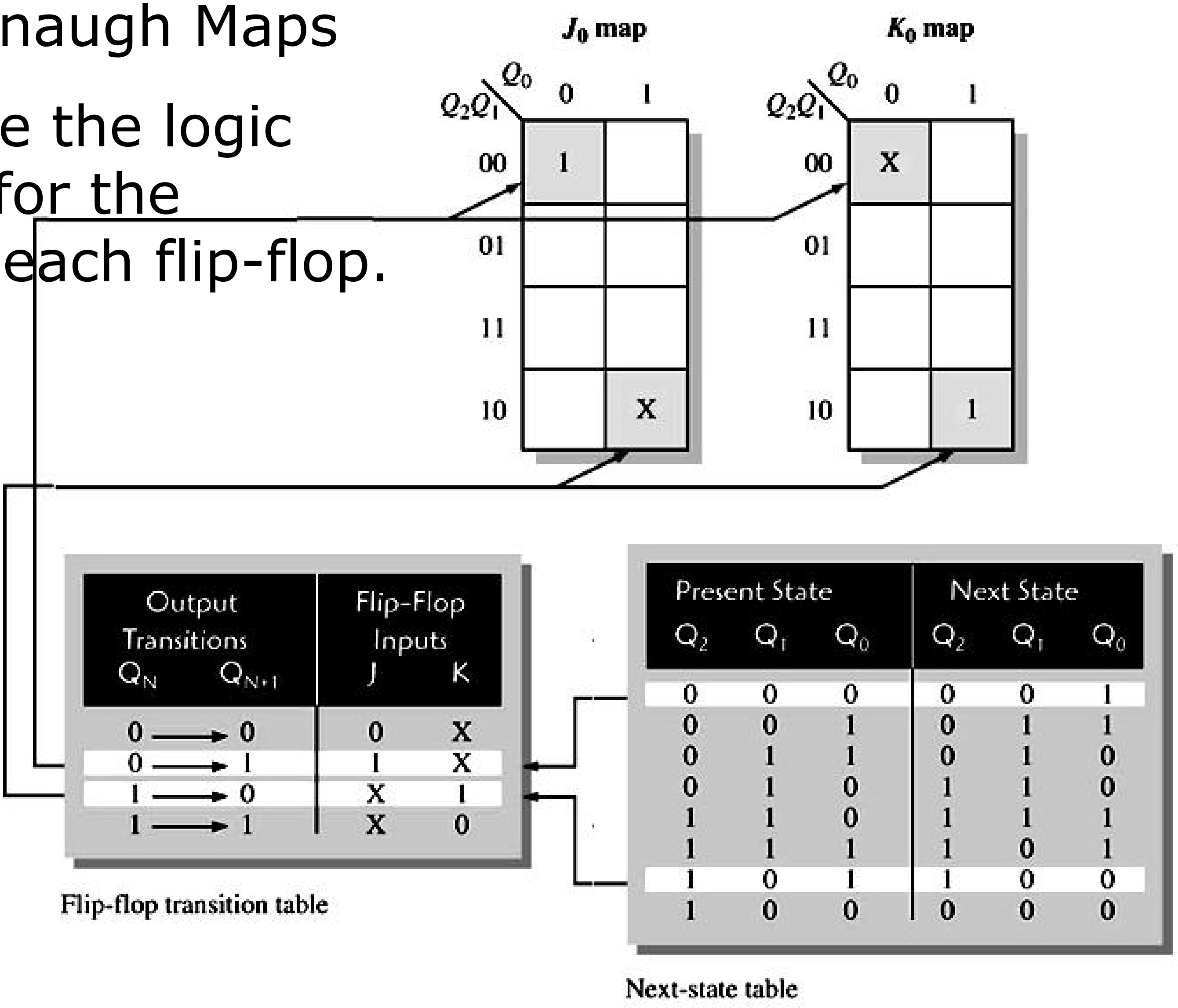
*D* flip flop design table.

| $q$ | $q^*$ | $D$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

# Design of Synchronous Counters

## Step 4: Karnaugh Maps

- Determine the logic required for the inputs of each flip-flop.



$J_0$ map

| $Q_2Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 00 | 1 | |
| 01 | | |
| 11 | | |
| 10 | | X |

$K_0$ map

| $Q_2Q_1$ \ $Q_0$ | 0 | 1 |
|---|---|---|
| 00 | X | |
| 01 | | |
| 11 | | |
| 10 | | 1 |

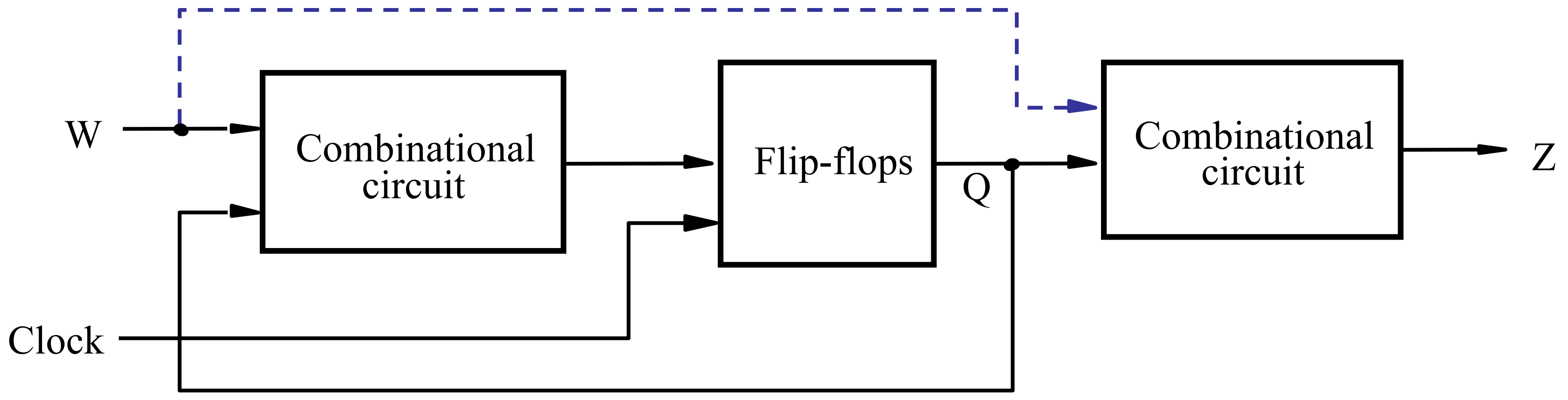
| Output Transitions $Q_N$ | Output Transitions $Q_{N+1}$ | Flip-Flop Inputs J | Flip-Flop Inputs K |
|---|---|---|---|
| 0 | 0 | 0 | X |
| 0 | 1 | 1 | X |
| 1 | 0 | X | 1 |
| 1 | 1 | X | 0 |

Flip-flop transition table

| Present State $Q_2$ | $Q_1$ | $Q_0$ | Next State $Q_2$ | $Q_1$ | $Q_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |

Next-state table

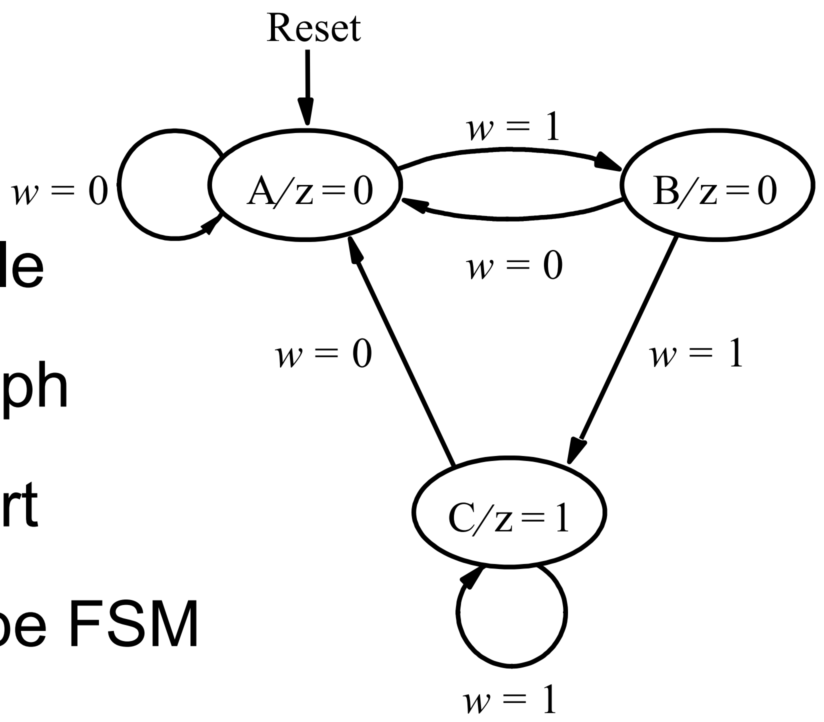
# Moore- and Mealy-Type FSMs

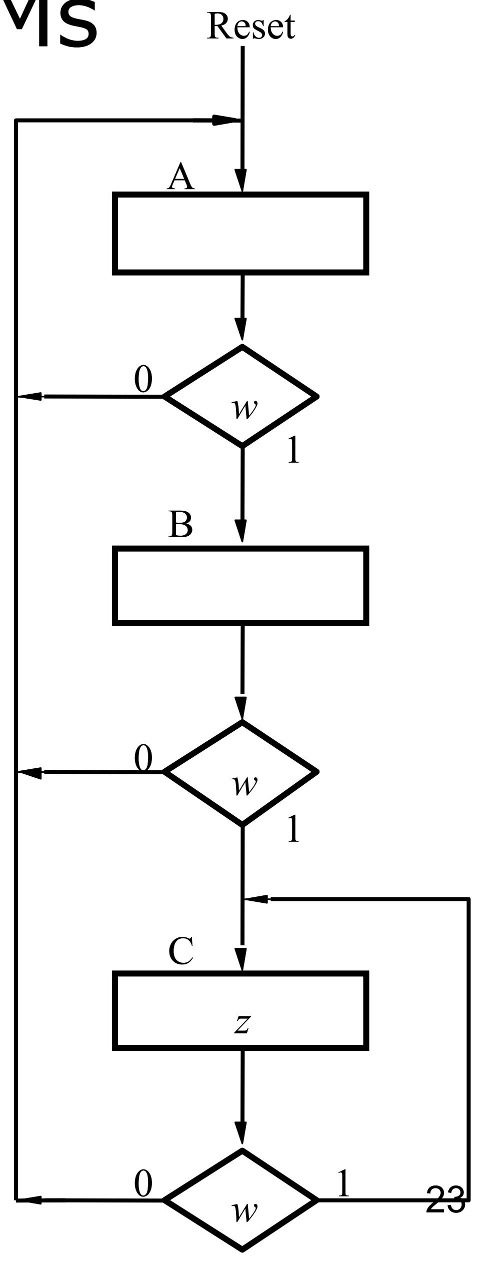


- Sequential circuits are called **finite state machines (FSM)**.
- Moore-type FSMs are those whose outputs depend only on the state of the circuits.
- Mealy-type FSMs are those whose outputs depend on both the state and the inputs.

# Representation of FSMs

| Present state | Next state $w=0$ | Next state $w=1$ | Output $z$ |
|---|---|---|---|
| A | A | B | 0 |
| B | A | C | 0 |
| C | A | C | 1 |





1. State table

2. State graph

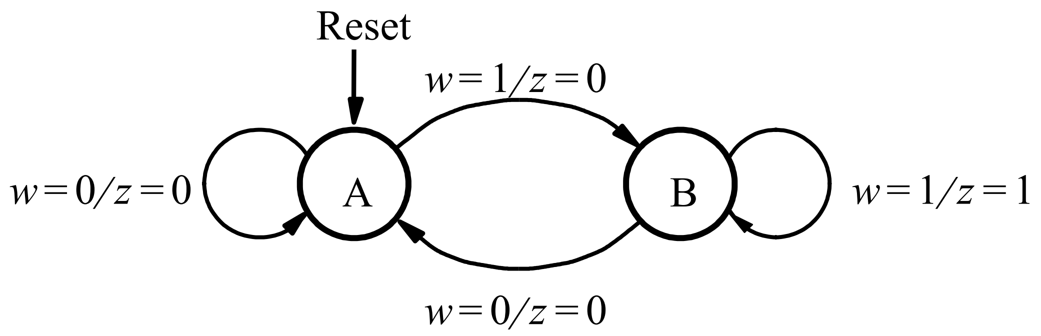
3. ASM chart

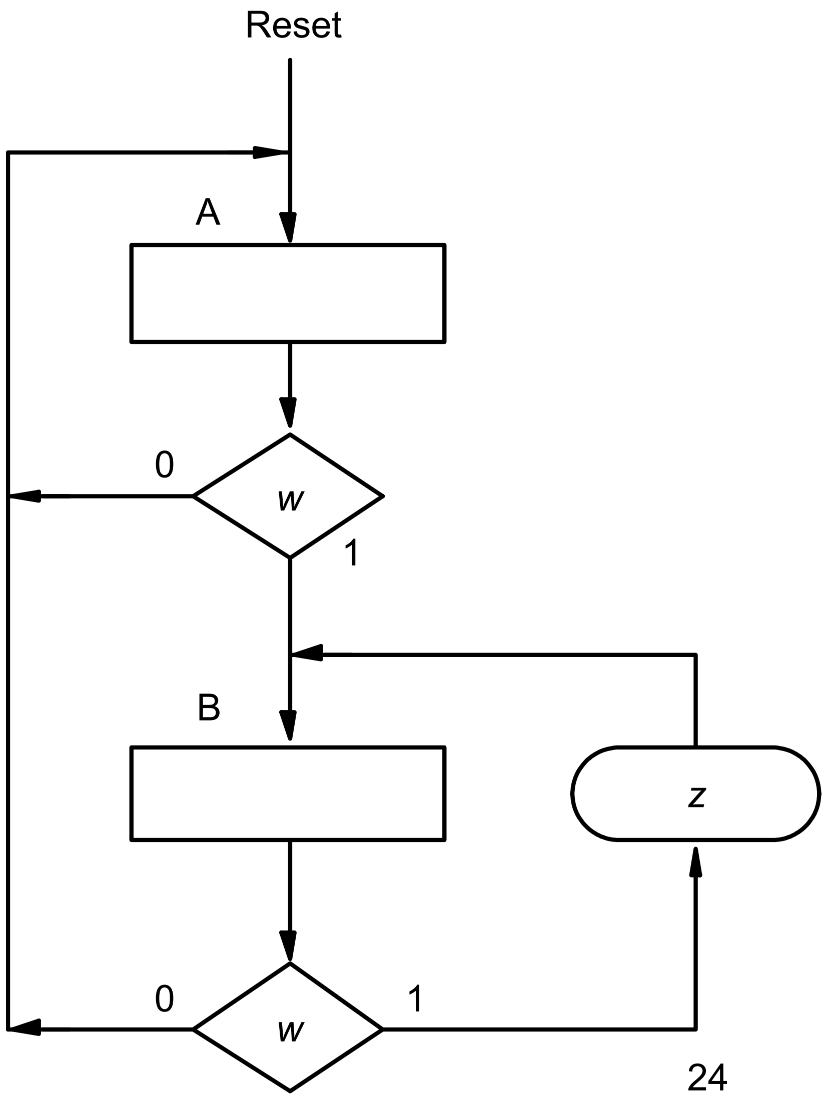
A Moore-type FSM

# Representation of FSMs

A Mealy-type FSM

| Present state | Next state | | Output $z$ | |
|---|---|---|---|---|
| | $w = 0$ | $w = 1$ | $w = 0$ | $w = 1$ |
| A | A | B | 0 | 0 |
| B | A | B | 0 | 1 |





# Design and Analysis of FSMs

Analysis of FSMs

Given a sequential circuit, sketch the state graph

1. Determine FF input (excitations) and output equations
2. Decide the next-state equation
3. The state table

# Design and Analysis of FSMs

Design of FSMs

Given a state graph, design the sequential circuit

1. State table
2. State assignment
3. Use present state, next state and FF design table to deduce FF inputs.
4. K-map to find the input (excitation) and output equations

# Design and Analysis of FSMs

$$\begin{tikzcd}
\text{J, K (D)} \arrow[r] & \text{Q+} \\
\text{Analysis} & \\
& \text{Q} \arrow[u]
\end{tikzcd}
\qquad
\begin{tikzcd}
\text{J, K (D)} & \text{Q+} \arrow[l] \\
& \text{Design} \\
\text{Q} \arrow[uu] &
\end{tikzcd}$$

Pitfalls

1. Don’t have a clear picture about the deduction process.
2. Forget FF next-state equations, e.g. $Q^+=JQ'+K'Q$
3. Forget FF design tables
4. Careless in the state table
5. Have no idea to cope with a Karnaugh map with don’t cares.