EEE205 – Digital Electronics (II)

Lecture 7

Xiaoyang Chen, Jiangmin Gu, Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

In This Session

Sequential Circuits in AHDL

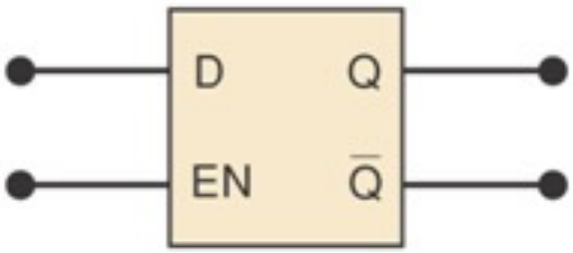
- The Latches and Flip-Flops
- Counters
- Shift Registers

Register Primitives

- Several types of register primitives are available in AHDL.
- Registers that use these primitives are declared in the VARIABLE section.

Primitive	Description
LATCH	Data (D) Latch
DFF/DFFE	Data (D) Flip-Flop
JKFF/JKFFE	JK Flip-Flop
SRFF/SRFFE	Set/Reset (SR) Flip-Flop
TFF/TFFE	Toggle (T) Flip-Flop

The D Latch



Inputs		Output
EN	D	Q
0	X	Q_0 (no change)
1	0	0
1	1	1

- Primitive: **LATCH**
- Inputs
  - Data Input (**d**)
  - Enable (**ena**)
- Outputs
  - Data Output (**q**)

The D Latch

- `dlatch` is an instance of LATCH.
- `enable`, `din` and `q` are the ports of the circuit block `dlatch_ahdl`.

```
SUBDESIGN dlatch_ahdl
(
     enable, din            :INPUT;
     q                      :OUTPUT;
)
VARIABLE
     dlatch                 :LATCH;
BEGIN
     dlatch.ena = enable;
     dlatch.d = din;
     q = dlatch.q;
END;
```

The D Latch

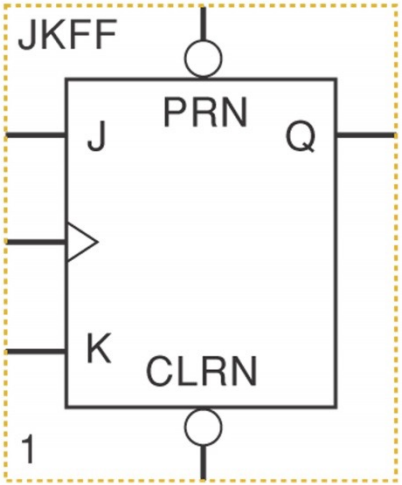
- If the instance name q is the same as the name of the circuit block output, it means the output of the latch is connected to the output port.

```
SUBDESIGN dlatch_ahdl
(
     enable, din            :INPUT;
     q                      :OUTPUT;
)
VARIABLE
     q                      :LATCH;
BEGIN
     q.ena = enable;
     q.d = din;
END;
```

The D Flip Flop

- Primitive: **DFF** or **DFFE**
- Inputs
  - Data Input (**d**)
  - Clock (**clk**)
  - Asynchronous Clear (**clrn**) and Preset (**prn**), both active-LOW.
  - Enable (**ena**), **DFFE** only
- Outputs
  - Data Output (**q**)

The J-K Flip Flop



- Primitive: **JKFF** or **JKFFE**
- Inputs
  - Set (**j**) and Reset (**k**)
  - Clock (**clk**)
  - Asynchronous Clear (**clrn**) and Preset (**prn**), both active-LOW.
  - Enable (**ena**), **JKFFE** only
- Outputs
  - Data Output (**q**)

The S-R Flip Flop

- Primitive: **SRFF** or **SRFFE**
- Inputs
  - Set (**s**) and Reset (**r**)
  - Clock (**clk**)
  - Asynchronous Clear (**clrn**) and Preset (**prn**), both active-LOW.
  - Enable (**ena**), **SRFFE** only
- Outputs
  - Data Output (**q**)

The T Flip Flop

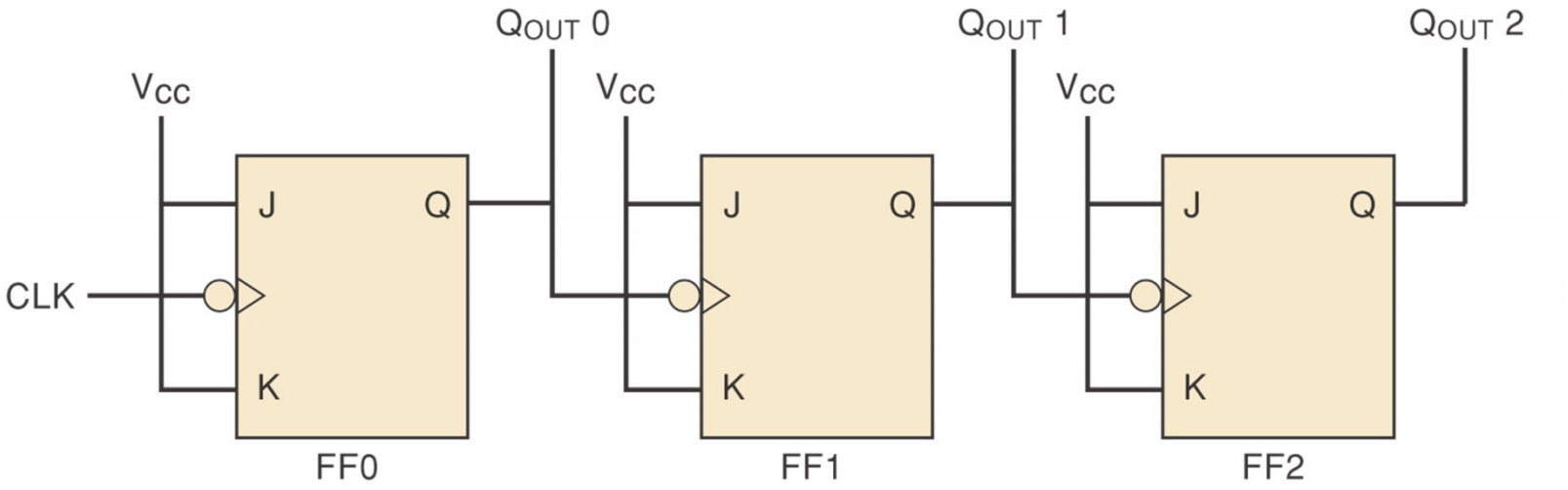
- Primitive: **TFF** or **TFFE**
- Inputs
  - Toggle (**t**)
  - Clock (**clk**)
  - Asynchronous Clear (**clrn**) and Preset (**prn**), both active-LOW.
  - Enable (**ena**), **TFFE** only
- Outputs
  - Data Output (**q**)

An Example

```
1   %     JK flip-flop circuit     %
2   SUBDESIGN fig5_65
3   (
4         jin, kin, clkin, preset, clear    :INPUT;
5         qout                              :OUTPUT;
6   )
7   VARIABLE
8   ff1          :JKFF;       -- define this flip-flop as a JKFF type
9   BEGIN
10        ff1.prn = preset; -- these are optional and default to vcc
11        ff1.clrn = clear;
12        ff1.j = jin;      -- connect primitive to the input signal
13        ff1.k = kin;
14        ff1.clk = clkin;
15        qout = ff1.q;     -- connect the output pin to the primitive
16  END;
```

Asynchronous Counters

MOD-8 Counter



- The name of this register is q, like the name of the output port.
- So the output of each flip-flop is connected to the output port.

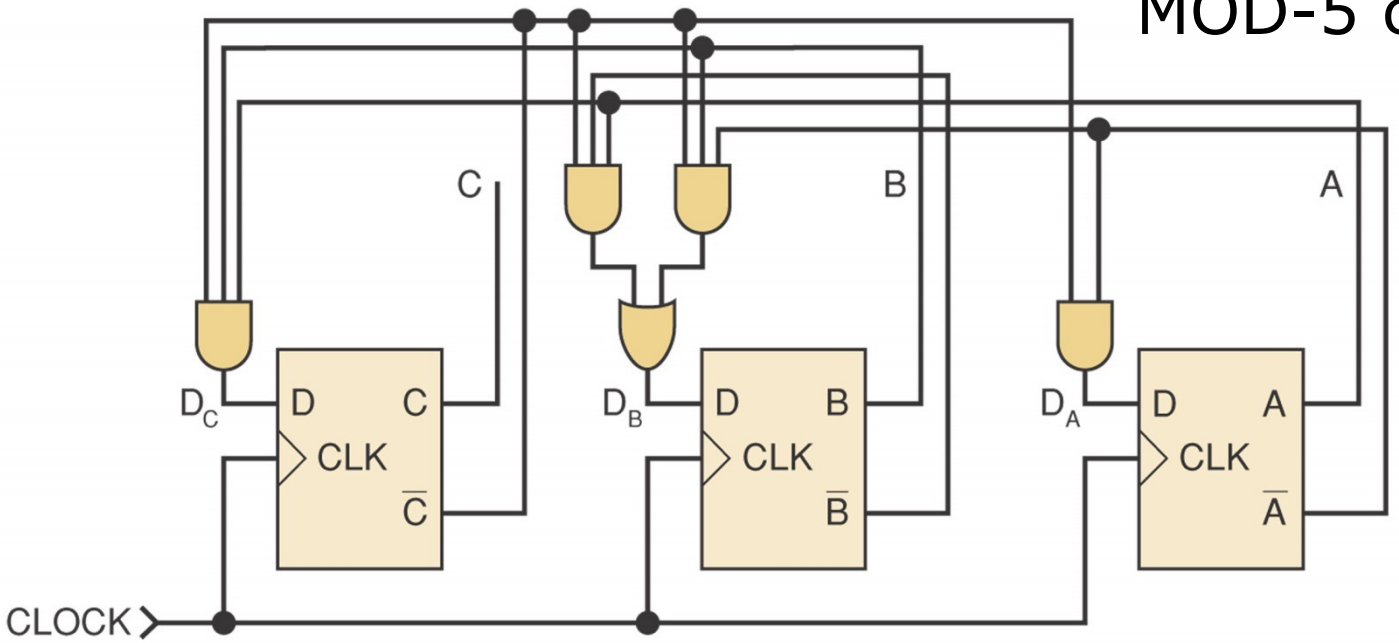
Asynchronous Counters

```
1   %  MOD 8 ripple up counter. %
2   SUBDESIGN fig5_71
3   (
4         clock                    :INPUT;
5         q[2..0]                  :OUTPUT;
6   )
7   VARIABLE
8         q[2..0] :JKFF;           -- defines three JK FFs
9   BEGIN
10                                 -- note: prn, clrn default to vcc!
11        q[2..0].j = VCC;         -- toggle mode J=K=1 for all FFs
12        q[2..0].k = VCC;
13        q[0].clk = !clock;
14        q[1].clk = !q[0].q;
15        q[2].clk = !q[1].q;      -- connect clocks in ripple form
16  END;
```

Synchronous Counters

- When using AHDL to implement a synchronous counter, we are no longer focusing on *wiring issues* but rather on describing the *circuit operation* concisely.

MOD-5 counter



Synchronous Counters

State Description – An MOD-5 Counter

```
1   SUBDESIGN fig7_35
2   (
3         clock          :INPUT;
4         q[2..0]        :OUTPUT;
5   )
6   VARIABLE
7         count[2..0]    :DFF;          --create a 3-bit register
8   BEGIN
9         count[].clk = clock;          --connect all clocks in parallel
10
11              CASE count[] IS
12  --                  Present              Next
13  ------------------------------------------------------------
14                      WHEN  0      =>     count[].d = 1;
15                      WHEN  1      =>     count[].d = 2;
16                      WHEN  2      =>     count[].d = 3;
17                      WHEN  3      =>     count[].d = 4;
18                      WHEN  4      =>     count[].d = 0;
19                      WHEN OTHERS  =>     count[].d = 0;
20                END CASE;
21          q[] = count[];              -- assign register to output pins
22  END;
```

Synchronous Counters

Behavioral Description – An MOD-5 Counter

```
1   SUBDESIGN fig7_39
2   (
3      clock    :INPUT;
4      q[2..0]  :OUTPUT;  -- declare 3-bit array of output bits
5   )
6   VARIABLE
7      count[2..0]  :DFF; -- declare a register of D flip flops.
8
9   BEGIN
10     count[].clk = clock;  -- connect all clocks to synchronous source
11     IF count[].q < 4 THEN    -- note; count[] is the same as count[].q
12        count[].d = count[].q + 1; -- increment current value by one
13     ELSE count[].d = 0;         -- recycle to zero: force unused states to 0
14     END IF;
15     q[] = count[];            -- transfer register contents to outputs
16  END;
```

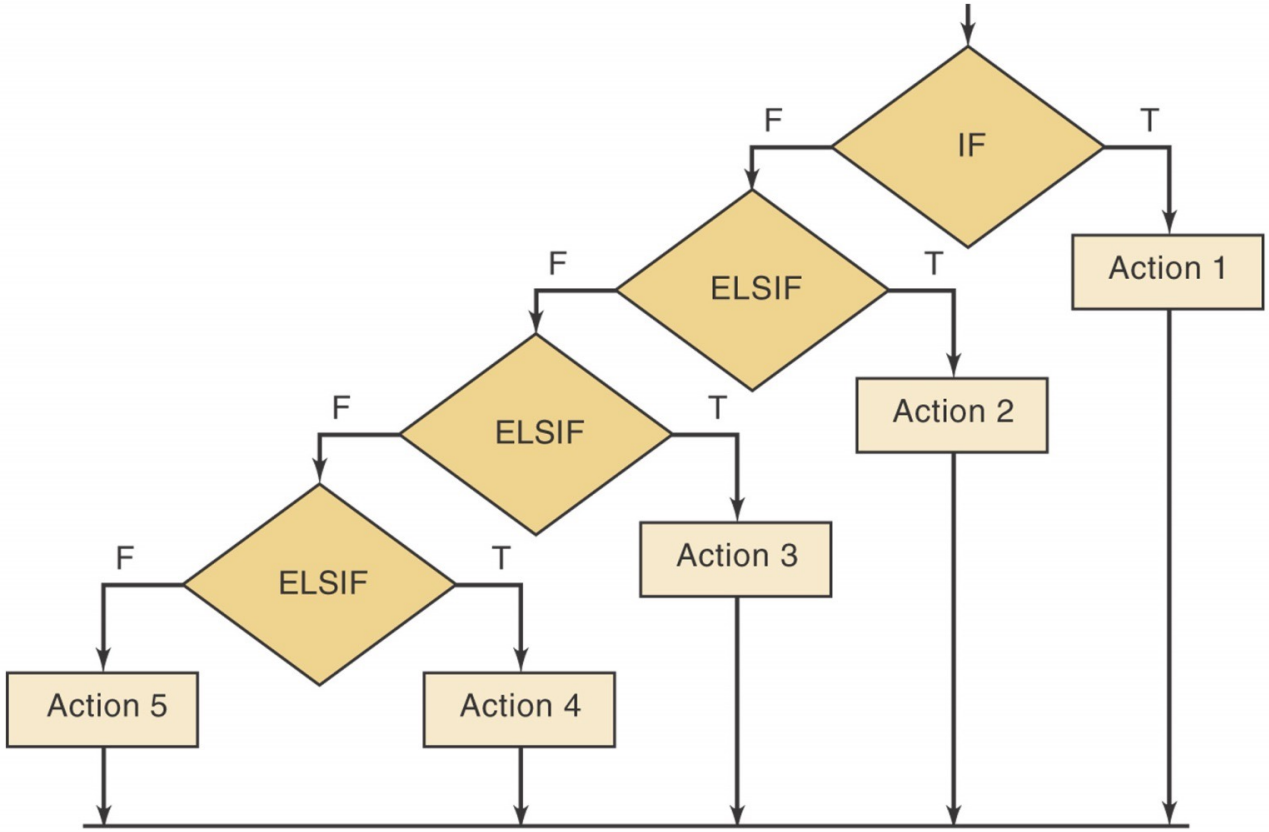
Synchronous Counters

Behavioral Description – A Full Feature Counter

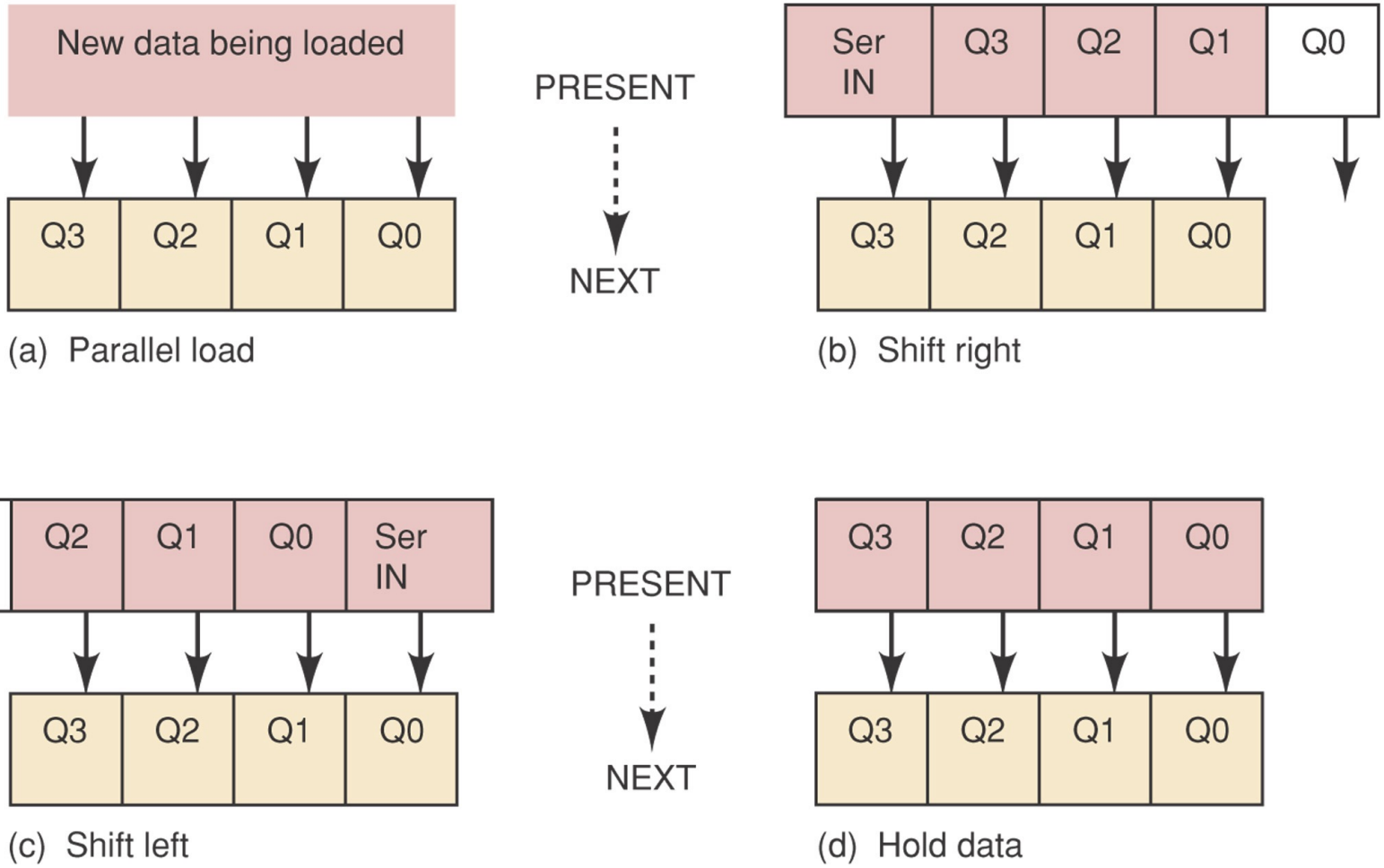
```
1   SUBDESIGN fig7_42
2   (
3      clock, clear, load, cntenabl, down, din[3..0]      :INPUT;
4      q[3..0], term_ct :OUTPUT;  -- declare 4-bit array of output bits
5   )
6   VARIABLE
7      count[3..0]    :DFF;       -- declare a register of D flip flops
8
9   BEGIN
10     count[].clk = clock;       -- connect all clocks to synch source
11     count[].clrn= !clear;      -- connect for asynch active HIGH clear
12     IF load THEN count[].d = din[]; -- synchronous load
13        ELSIF !cntenabl THEN count[].d = count[].q; -- hold count
14        ELSIF !down THEN count[].d = count[].q + 1; -- increment
15        ELSE count[].d = count[].q - 1;             -- decrement
16     END IF;
17     IF ((count[].q == 0) & down # (count[].q == 15) & !down)& cntenabl
18     THEN     term_ct = VCC;    -- synchronous cascade output signal
19     ELSE term_ct = GND;
20     END IF;
21     q[] = count[];             -- transfer register contents to outputs
22  END;
```

Synchronous Counters

- The **IF/ELSIF** indicates the priority of functions.
- In this example, `load` > `cntenabl` > `down`



Shift Registers



Shift Registers

Serial In/ Serial Out

```
1   SUBDESIGN  fig7_77
2   (
3         clk, shift, serial_in          :INPUT;
4         serial_out                     :OUTPUT;
5   )
6   VARIABLE
7         q[3..0]                        :DFF;
8   BEGIN
9         q[].clk = clk;
10        serial_out = q[0].q;                          -- output last register bit
11        IF (shift == VCC)  THEN
12              q[3..0].d = (serial_in, q[3..1].q);     -- concatenates for shift
13        ELSE
14              q[3..0].d = (q[3..0].q);                -- hold data
15        END IF;
16  END;
```

Shift Registers

Universal Shift Register

```
 1  SUBDESIGN fig7_83
 2  (
 3     clock        :INPUT;
 4     din[3..0]    :INPUT;  -- parallel data in
 5     ser_in       :INPUT;  -- serial data in from Left or Right
 6     mode [1..0]  :INPUT;  -- MODE Select: 0=hold, 1=right, 2=left, 3=load
 7     q[3..0]      :OUTPUT;
 8  )
 9  VARIABLE
10     ff[3..0] :DFF;        -- define register set
11  BEGIN
12     ff[].clk = clock;     -- synchronous clock
13     CASE mode[] IS
14        WHEN 0  => ff[].d     = ff[].q;         -- hold shift
15        WHEN 1  => ff[2..0].d = ff[3..1].q);    -- shift right
16                   ff[3].d    = ser_in;         -- new data from left
17        WHEN 2  => ff[3..1].d = ff[2..0].q;     -- shift left
18                   ff[0].d    = ser_in;         -- new data bit from right
19        WHEN 3  => ff[].d     = din[];          -- parallel load
20     END CASE;
21     q[] = ff[];               -- update outputs
22  END;
```

EEE205 – Digital Electronics (II)

Lecture 8

Xiaoyang Chen, Jiangmin Gu, Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

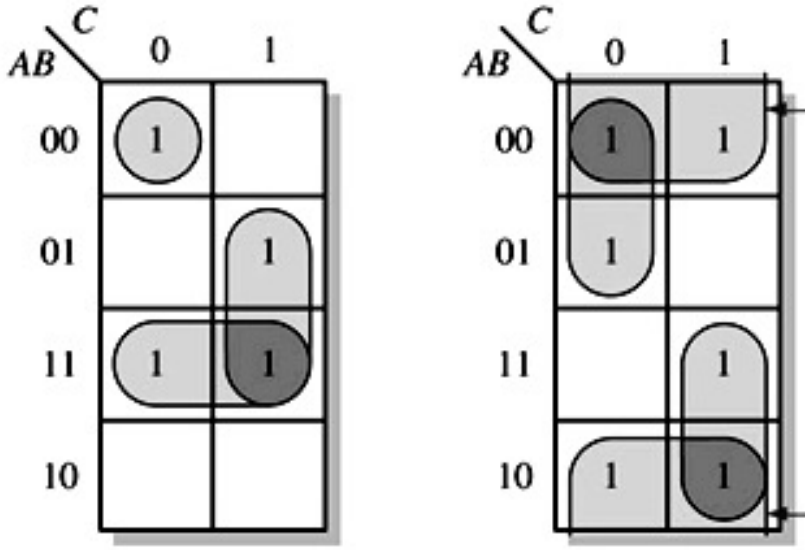
In This Session

- A Revision of Karnaugh Maps
- The Terminology
- Five-Variable Karnaugh Maps

A Revision of Karnaugh Maps

Grouping the 1s

- The goal is to **maximize the size** of the groups (shorter product terms) and to **minimize the number** of groups (less product terms).
- A group may contain 1, 2, 4, 8, or 16 adjacent cells.
- Each 1 must be included in one or **more** groups.

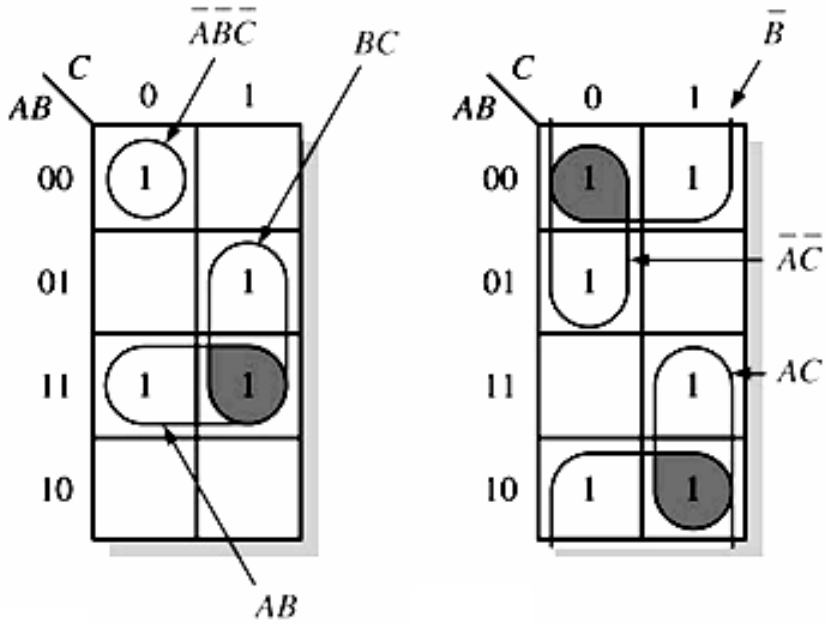


A Revision of Karnaugh Maps

Determine the Minimum SOP

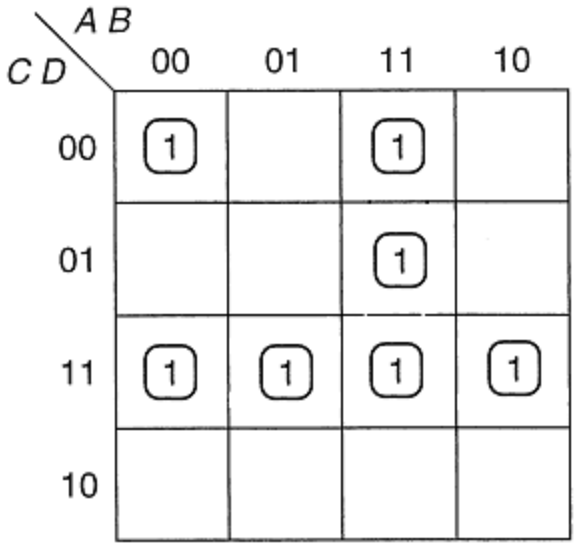
1. When a variable appears in both complemented and uncomplemented form in a group, that variable is eliminated.

2. Variables that are the same for all cells of the group must appear — 1 for uncomplemented form and 0 for complemented form.



The Terminology

- A **minterm** is a product term that includes all the variables in a Boolean function.
- It corresponds a 1 cell in a K-map.
- Here 7 minterms in total.



The Terminology

- A minterm is often referred to by the decimal number converted from its binary value.

ABC	Minterm	Number
000	A′B′C′	0
001	A′B′C	1
010	A′BC′	2
011	A′BC	3
100	AB′C′	4
101	AB′C	5
110	ABC′	6
111	ABC	7

$$m_1 = A'B'C$$

$$m_6 = ABC'$$

The Terminology

- Minterms are often used to express a specific Boolean function or a truth table in a compact way.

ABC	f
000	0
001	1
010	1
011	1
100	1
101	1
110	0
111	0

$$f(A,B,C) = A'B'C + A'BC' + A'BC + AB'C' + AB'C$$

$$f(A,B,C) = m_1 + m_2 + m_3 + m_4 + m_5 = \Sigma m(1,2,3,4,5)$$

The Terminology

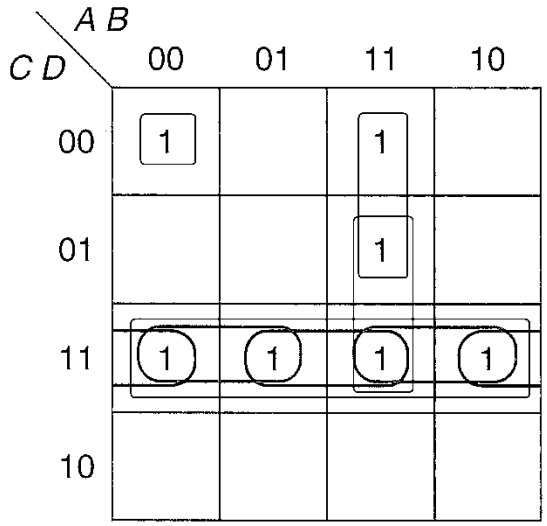
- If the function includes **don't cares**, those terms are included in a separate sum.

abc	g
000	X
001	1
010	1
011	X
100	0
101	1
110	0
111	0

$$g(a,b,c) = \Sigma m(1,2,5) + \Sigma d(0,3)$$

The Terminology

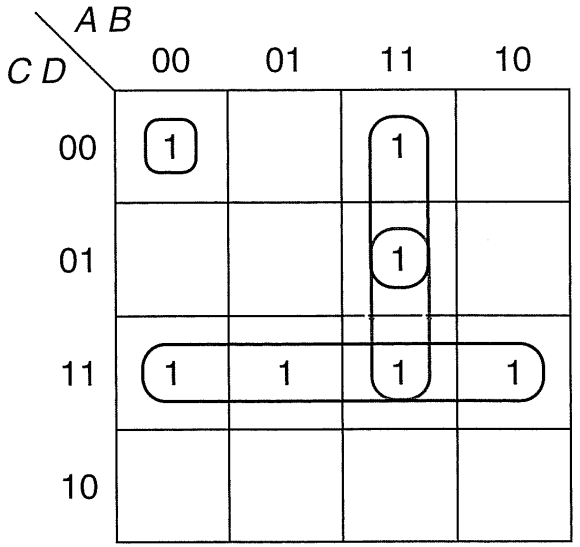
- An **implicant** is a product form that can be used in the sum-of-products expression for a function.
- An implicant corresponds to a rectangle of 1, 2, 4, 8, …(any power of 2) 1's in a K-map.



- Here 14 implicants: seven 1's, six groups of 2 and one group of 4.

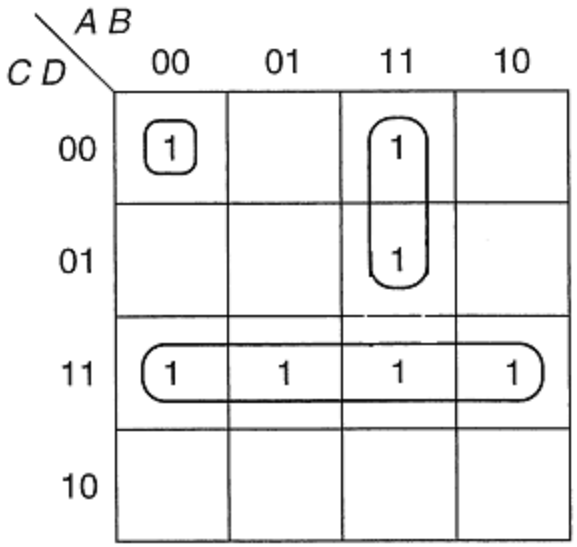
The Terminology

- A **prime implicant** is an implicant which can not be merged with another implicant to remove a variable.
- It corresponds to a 1's group which is not fully contained in another group.
- Here 4 prime implicants: A'B'C'D', ABC', ABD and CD.



The Terminology

- An **essential prime implicant** is a prime implicant that includes at least one 1 that is not included in any other prime implicant.
- Essential prime implicants must appear in a fully minimized SOP expression.
- Here 3 essential prime implicants: A'B'C'D', ABC' and CD.

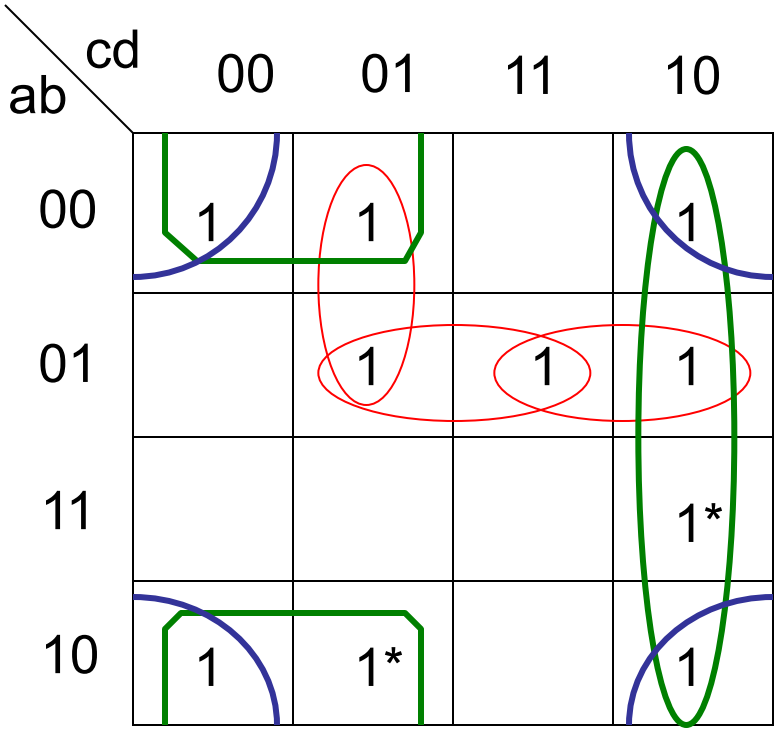


Karnaugh Maps

Find prime implicants

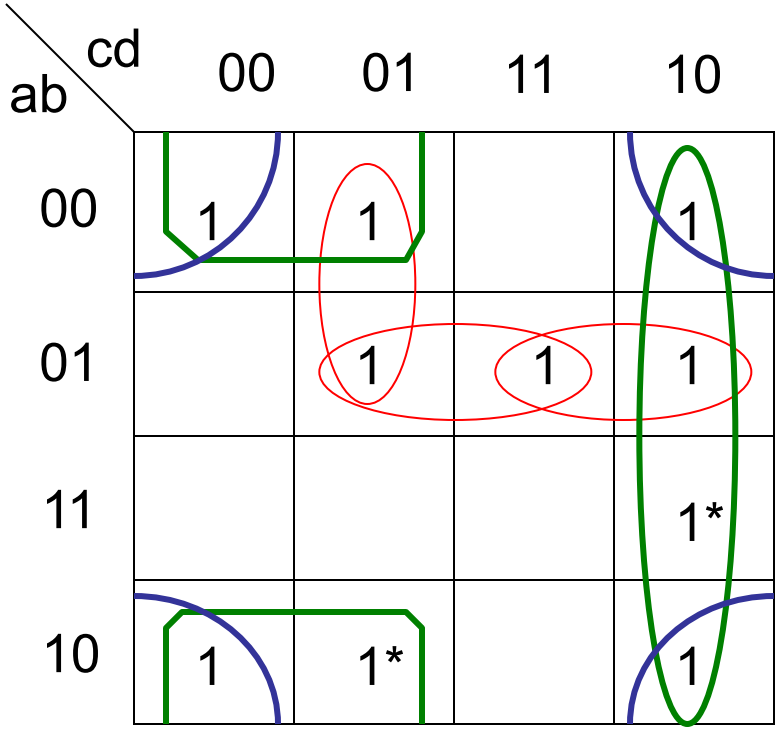
$$f(a, b, c, d) =$$
$$\Sigma\, m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)$$

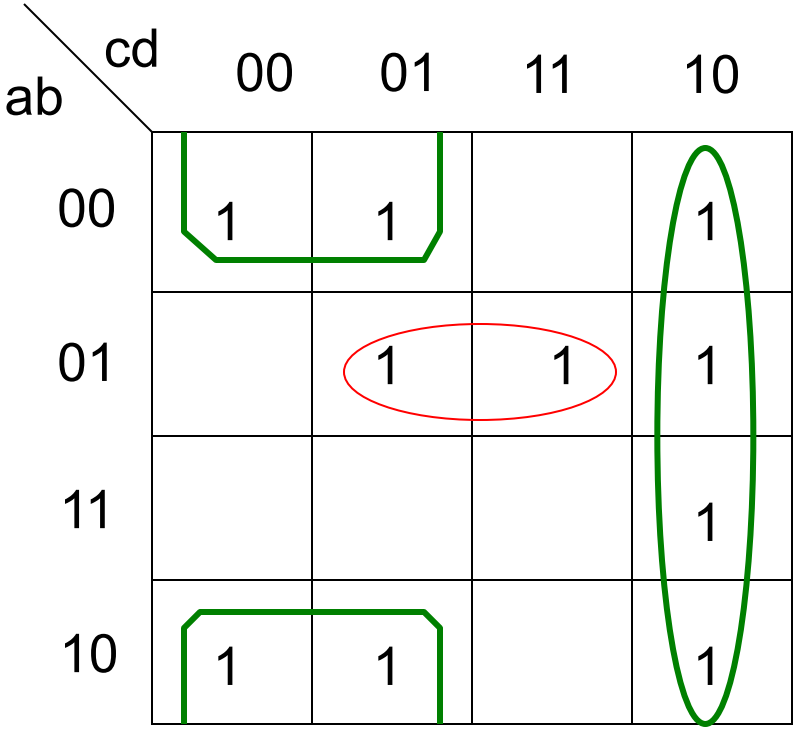
ab \ cd	00	01	11	10
00	1	1		1
01		1	1	1
11				1
10	1	1		1



Karnaugh Maps

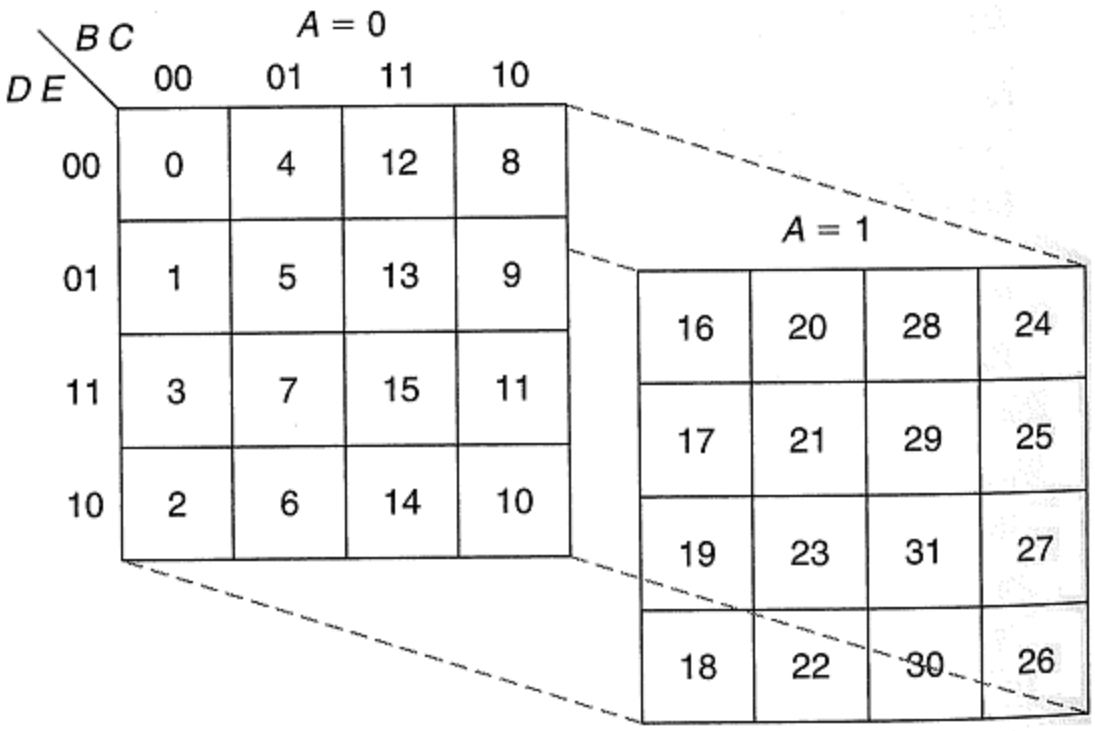
Find essential prime implicants





Five-Variable Karnaugh Maps

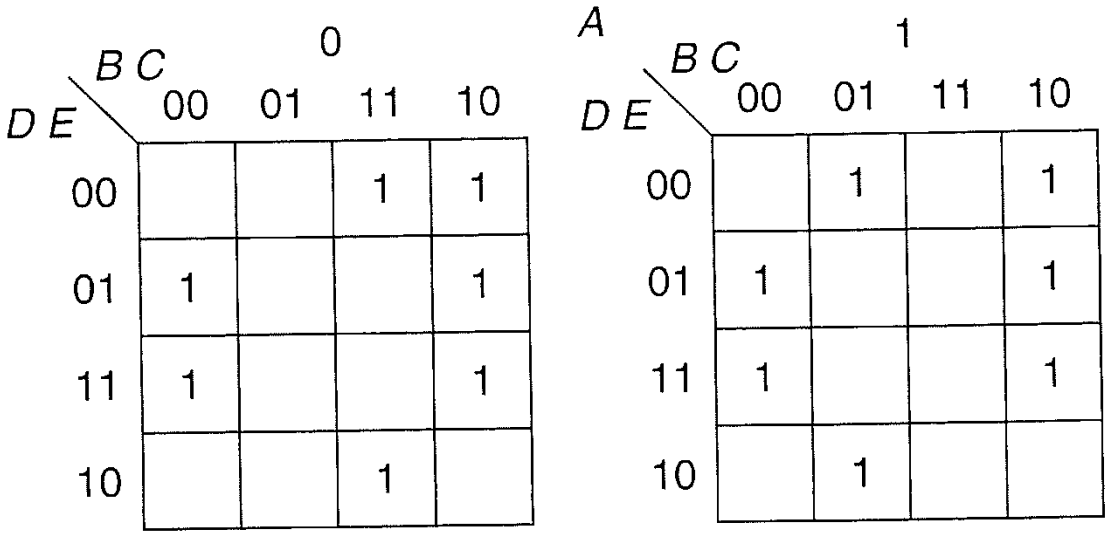
- Boolean functions with five variables can be simplified using **two** 4-variable maps.
- Squares directly above or below each other are adjacent.



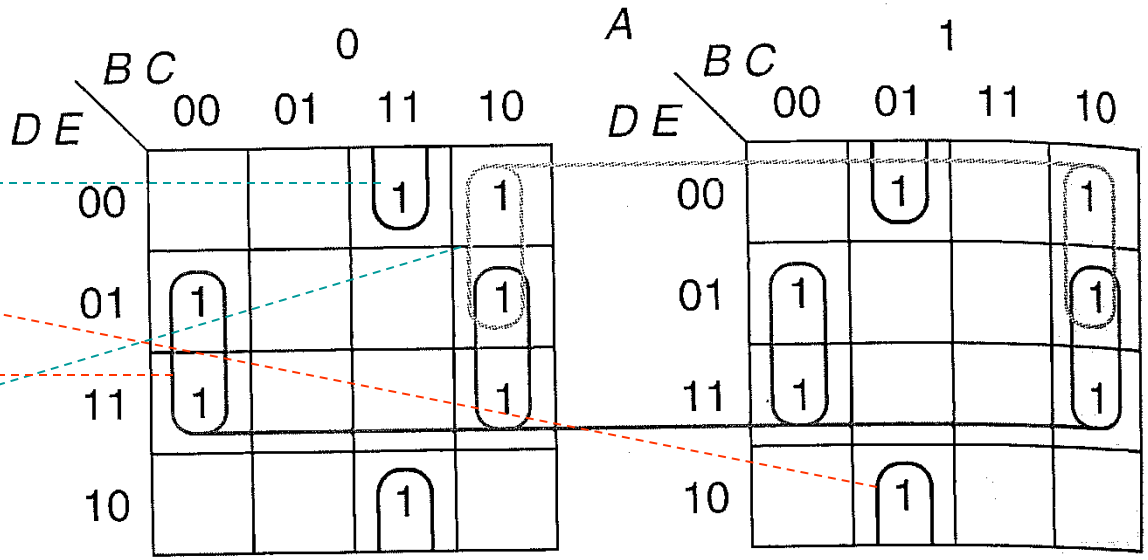
Five-Variable Karnaugh Maps

Example 1:

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

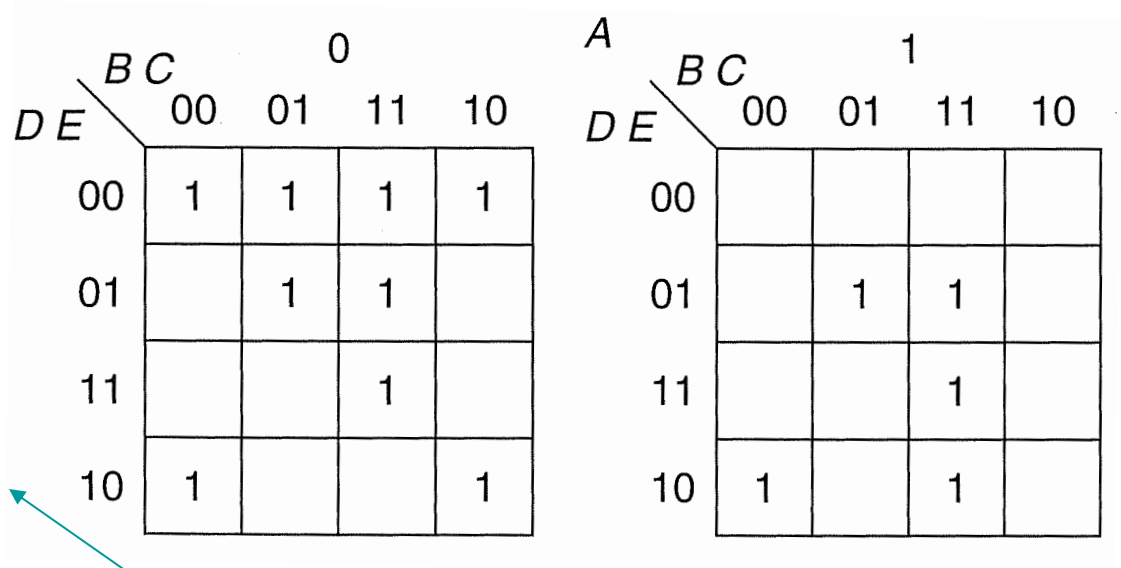


$$F = \bar{A}BC\bar{E} + A\bar{B}C\bar{E} + \bar{C}E + B\bar{C}\bar{D}$$

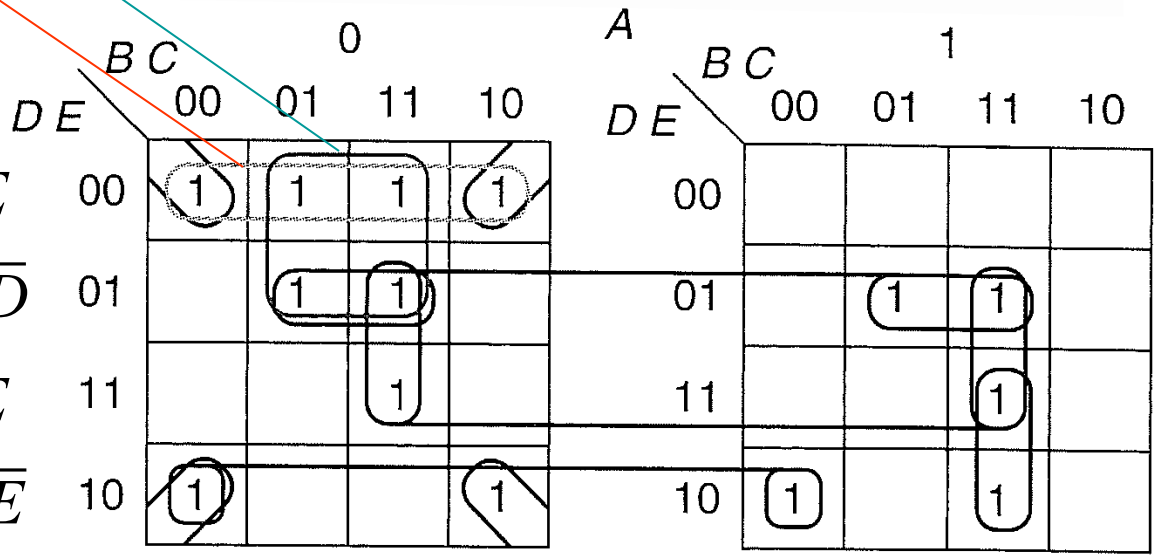


Five-Variable Karnaugh Maps

Example 2:



Alternative Solution !



$$F = \bar{A}\bar{C}\bar{E} + ABCD + C\bar{D}E + BCE + \bar{B}\bar{C}D\bar{E} + \bar{A}C\bar{D}$$

$$F = \bar{A}\bar{C}\bar{E} + ABCD + C\bar{D}E + BCE + \bar{B}\bar{C}D\bar{E} + \bar{A}\bar{D}\bar{E}$$