CPE/EE 422/522

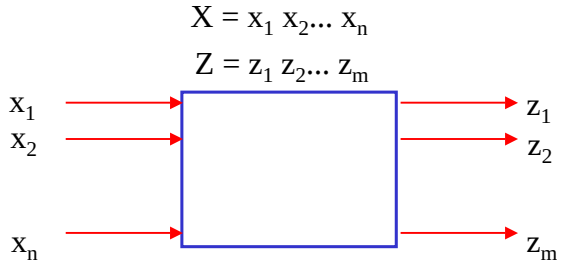
Chapter 1 - Review of Logic Design Fundamentals

Dr. Rhonda Kay Gaede

UAH

1.1 Combinational Logic

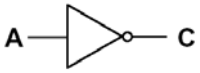
- Combinational Logic has no control inputs. When the inputs to a combinational network change, the output changes _____.



1.1 Combinational Logic - Basic Logic Gates

AND: C = A B

OR: C = A + B

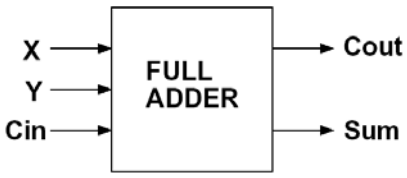


NOT: C = A'

EXCLUSIVE OR: C = A ⊕ B

1.1 Combinational Logic - Full Adder (Minterm Form)

Module



Truth table

X	Y	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

m-form

Sum =

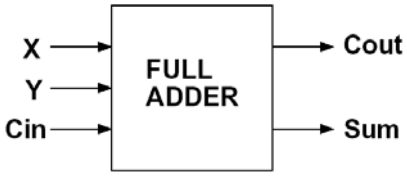
Cout =

Sum =

Cout =

1.1 Combinational Logic - Full Adder (Maxterm Form)

Module



Truth table

X	Y	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$\overline{Sum}$ =

Sum =

$\overline{Cout}$ =

Cout =

m-form

Sum =

Cout =

1.2 Boolean Algebra and Algebraic Simplification

Operations with 0 and 1:

$X + 0 = X$ (1-5) $X \cdot 1 = X$ (1-5D)

$X + 1 = 1$ (1-6) $X \cdot 0 = 0$ (1-6D)

Idempotent laws:

$X + X = X$ (1-7) $X \cdot X = X$ (1-7D)

Involution law:

$(X')' = X$ (1-8)

Laws of complementarity

$X + X' = 1$ (1-9) $X \cdot X' = 0$ (1-9D)

Commutative laws:

$X + Y = Y + X$ (1-10) $XY = YX$ (1-10D)

Associative laws:

$(X + Y) + Z = X + (Y + Z) = X + Y + Z$ (1-11) $(XY)Z = X(YZ) = XYZ$ (1-11D)

Distributive laws:

$X(Y + Z) = XY + XZ$ (1-12) $X + YZ = (X + Y)(X + Z)$ (1-12D)

1.3 More Boolean Algebra and Algebraic Simplification

Simplification theorems:

$XY + XY' = X$	(1-13)	$(X + Y)(X + Y') = X$	(1-13D)
$X + XY = X$	(1-14)	$X(X + Y) = X$	(1-14D)
$(X + Y')Y = XY$	(1-15)	$XY' + Y = X + Y$	(1-15D)

DeMorgan's laws:

$(X + Y + Z + \cdots)' = X'Y'Z'\cdots$	(1-16)	$(XYZ\cdots)' = X' + Y' + Z' + \cdots$	(1-16D)
$[f(X_1, X_2, \ldots, X_n, 0, 1, +, \cdot)]' = f(X_1', X_2', \ldots, X_n', 1, 0, \cdot, +)$			(1-17)

Duality:

$(X + Y + Z + \cdots)^D = XYZ\cdots$	(1-18)	$(XYZ\cdots)^D = X + Y + Z + \cdots$	(1-18D)
$[f(X_1, X_2, \ldots, X_n, 0, 1, +, \cdot)]^D = f(X_1, X_2, \ldots, X_n, 1, 0, \cdot, +)$			(1-19)

Theorem for multiplying out and factoring:

$(X + Y)(X' + Z) = XZ + X'Y$	(1-20)	$XY + X'Z = (X + Z)(X' + Y)$	(1-20D)

Consensus theorem:

$XY + YZ + X'Z = XY + X'Z$	(1-21)	$(X + Y)(Y + Z)(X' + Z) = (X + Y)(X' + Z)$	(1-21D)

1.2 Boolean Algebra with Exclusive OR

$X \oplus 1 = X'$

$X \oplus 0 = X$

$X \oplus X = 0$

$X \oplus X' = 1$

$X \oplus Y = Y \oplus X$ (Commutative law)

$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$

$X(Y \oplus Z) = XY \oplus XZ$ (Distributive law)

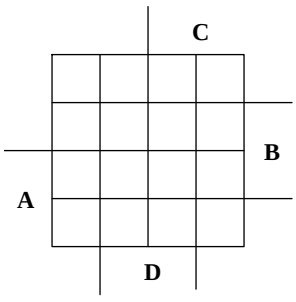
$(X \oplus Y)' = X \oplus Y' = X' \oplus Y = XY + X'Y'$

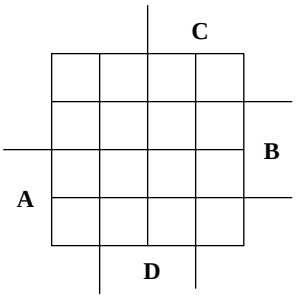
1.3 Karnaugh Maps

- Convenient way to simplify logic functions of 3, 4, 5, (6) variables
- Four-variable K-map
  - Each square
    - 1
    - 0
    - d
      - ---
      - ---
  - adjacent cells

CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

1.3 Karnaugh Maps - Examples





1.3 Karnaugh Maps - Sum of Products

- Function consists of a sum of _____
- Prime implicant

  _____
  __

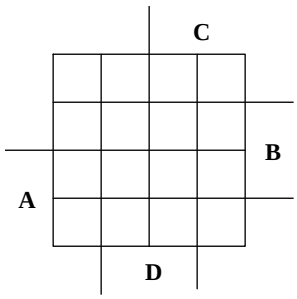
  _____
  __
- Prime implicant is _____ if it contains a 1 that is not contained in any other prime implicant

1.3 Karnaugh Maps - Prime Implicant Selection

- Two minimum forms

  f =

  f =

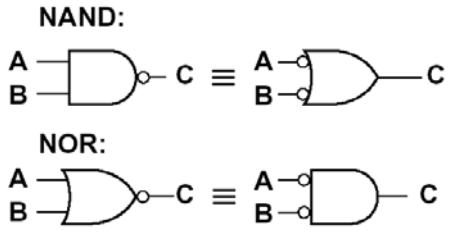


1.3 Karnaugh Maps - Selection Procedure

1. Choose a 1 (_____) that has not been covered yet
2. Find all _s and __s adjacent to that minterm. (Check the n adjacent squares on an n-variable map.)
3. If a single term covers the minterm and all the adjacent 1s and ds, then that term is an _____ prime implicant, so select that term.
4. Repeat steps 1, 2, 3 until all essential prime implicants have been chosen
5. Find a _____ set of prime implicants that cover the remaining 1s on the map. If there is more than one such set, choose a set with a _____ _____

1.4 Designing with NAND and NOR Gates

- Implementation of NAND and NOR gates is easier than that of AND and OR gates (e.g., CMOS)



1.4 Designing with NAND and NOR Gates (continued)

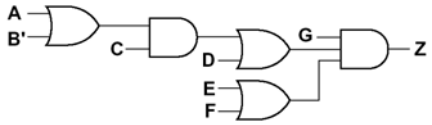
- Any logic function can be realized using only NAND or NOR gates -

  _____

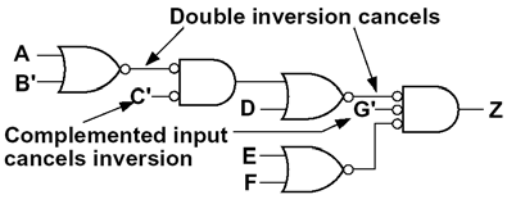
  – 1:
  – 0:
  – a':
  – ab:
  – a+b:

1.4 Designing with NAND and NOR Gates - Conversion to NOR Gates

- Start with POS (_____)
  – circle 0s in K-maps
- Find network of OR and AND gates



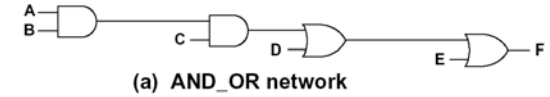
(a) AND-OR network



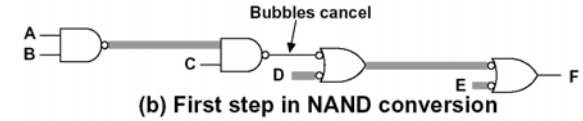
(b) Equivalent NOR-gate network

1.4 Designing with NANDand NOR Gates - Conversion to NAND Gates

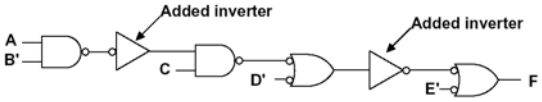
- Start with SOP (Sum of Products)
  - circle 1s in K-maps
- Find network of OR and AND gates



(a) AND_OR network



(b) First step in NAND conversion

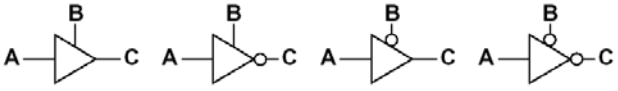


(c) Completed conversion

1.13 Tristate Logic and Busses

- Four kinds of tristate buffers

  B is a control input used to enable and disable the output



B	A	C
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

(a)

B	A	C
0	0	Hi-Z
0	1	Hi-Z
1	0	1
1	1	0

(b)

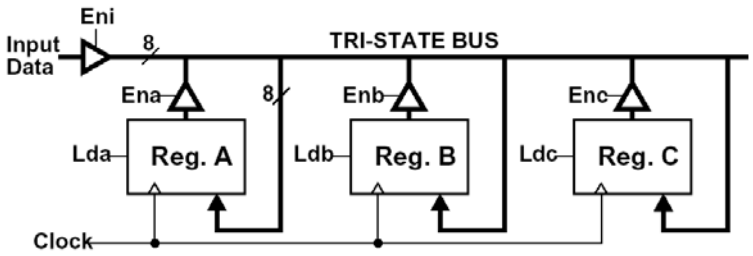
B	A	C
0	0	0
0	1	1
1	0	Hi-Z
1	1	Hi-Z

(c)

B	A	C
0	0	1
0	1	0
1	0	Hi-Z
1	1	Hi-Z

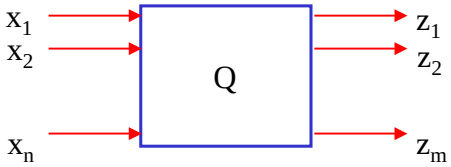
(d)

1.13 Tristate Logic and Busses - Data Transfer



1.6 Flip-Flops and Latches – Sequential Networks

- Have memory (state)
  - Present state depends not only on the _____, but also on all previous inputs (history)
  - Future state depends on the _____ and _____

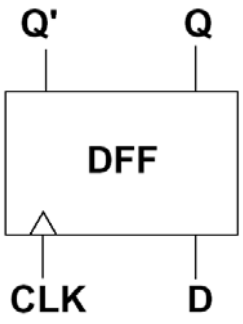


$$Z(t) = F(X(t), Q(t))$$

$$Q(t^+) = G(X(t), Q(t))$$

Flip-flops are commonly used as storage devices: D-FF, JK-FF, T-FF

1.6 Flip-Flops and Latches - Clocked D Flip-Flop with Rising-edge Trigger



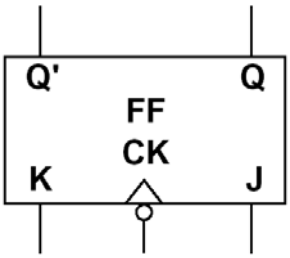
D	Q	Q^+
0	0	0
0	1	0
1	0	1
1	1	1

$Q^+ = D$

Next state

The next state in response to the _____ of the clock is equal to the _____ input before the rising edge

1.6 Flip-Flops and Latches – Clocked JK Flip-Flop

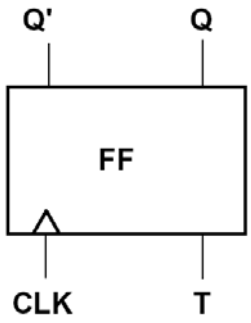


J	K	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Next state

JK = 00 =>
JK = 10 =>
JK = 01 =>
JK = 11 =>

1.6 Flip-Flops and Latches – Clocked T Flip-Flop



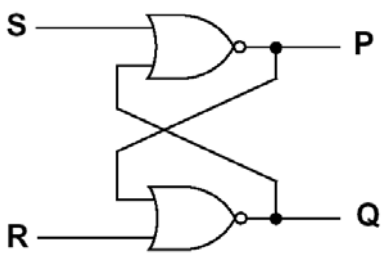
T	Q	Q^+
0	0	0
0	1	1
1	0	1
1	1	0

Next state

T = 1 =>

T = 0 =>

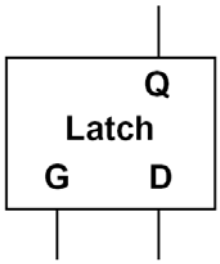
1.6 Flip-Flops and Latches – S-R Latch



S	R	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	–
1	1	1	–

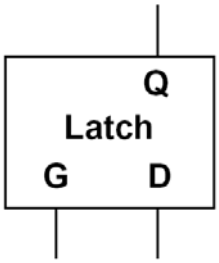
$Q^+ = S + R'Q$

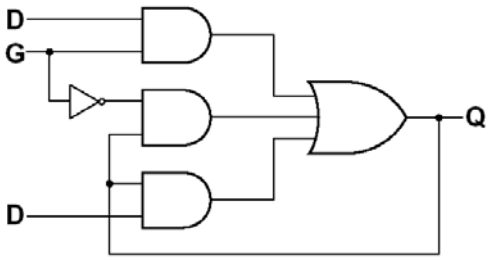
1.6 Flip-Flops and Latches – Transparent D Latch



G	D	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

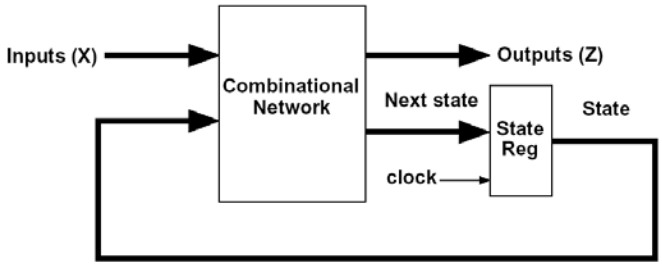
1.6 Flip-Flops and Latches – Transparent D Latch





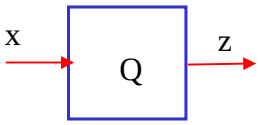
$Q^+ = DG + G'Q + (DQ)$

1.7 Mealy Sequential Network Design – General Model of Mealy Sequential Machine



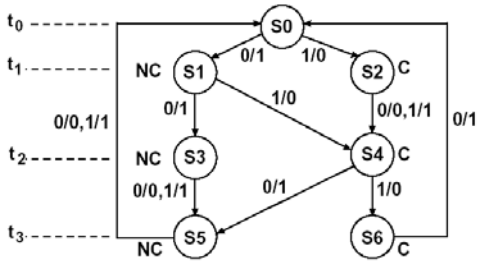
(1) X inputs are changed to a new value
(2) After a delay, the Z outputs and next state appear at the output of CN
(3) The next state is clocked into the state register and the state changes

1.7 Mealy Sequential Network Design - 8421 BCD to Excess3 BCD Code Converter Example



X (inputs)				Z (outputs)			
t3	t2	t1	t0	t3	t2	t1	t0
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

1.7 Mealy Sequential Network Design – State Graph and Table for Code Converter



1.9 Equivalent States and Reduction of State Tables – Code Converter Example

I. States which have the same next state (NS) for a given input should be given adjacent assignments (look at the columns of the state table).

II. States which are the next states of the same state should be given adjacent assignments (look at the rows).

III. States which have the same output for a given input should be given adjacent assignments.

I. (1,2) (3,4) (5,6) (in the X=1 column, S_1 and S_2 both have NS S_4; in the X=0 column, S_3 & S_4 have NS S_5, and S_5 & S_6 have NS S_0)

II. (1,2) (3,4) (5,6) (S_1 & S_2 are NS of S_0; S_3 & S_4 are NS of S_1 and S_5 & S_6 are NS of S_4)

III. (0,1,4,6) (2,3,5)

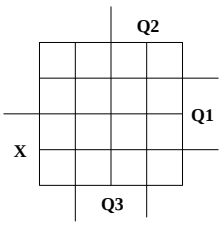
Q2 Q3 \ Q1	0	1
00	S0	S1
01		S2
11	S5	S3
10	S6	S4

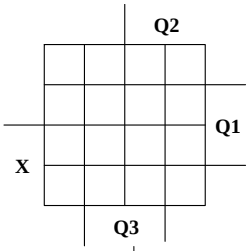
1.9 Equivalent States and Reduction of State Tables – Code Converter Transition Table

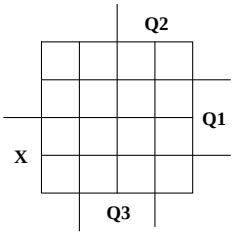
PS	NS X = 0	NS X = 1	Z X= 0	Z X = 1
S0				
S1				
S2				
S3				
S4				
S5				
S6				

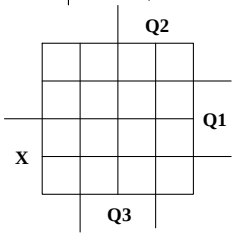
Q1Q2Q3	Q1+Q2+Q3+ X = 0	Q1+Q2+Q3+ X = 1	Z X= 0	Z X = 1

1.9 Equivalent States and Reduction of State Tables – Code Converter K-maps

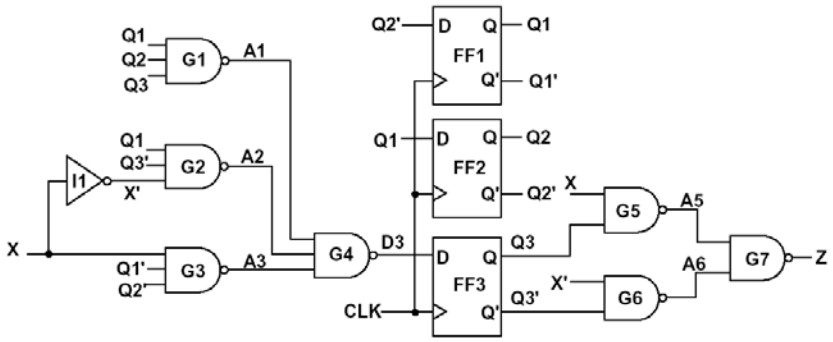






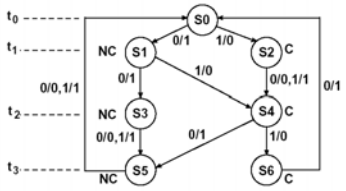


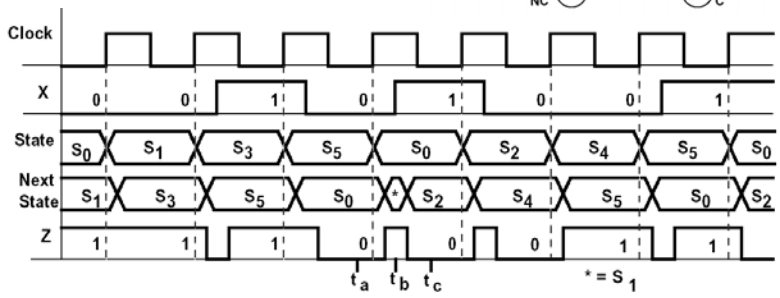
1.9 Equivalent States and Reduction of State Tables – Code Converter Realization



1.10 Sequential Network Timing – Code Converter

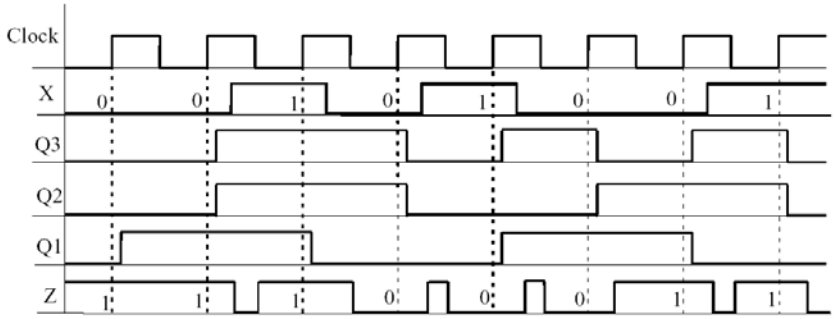
- Code converter
  - X = 0010_1001 =>
    Z = 1110_0011





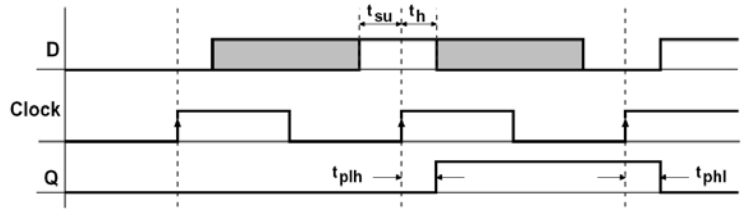
1.10 Sequential Network Timing – Timing Diagram for Code Converter

Timing diagram assuming a propagation delay
of 10 ns for each flip-flop and gate
(State has been replaced with the state of three flip-flops)



1.11 Setup and Hold Times – D Flip-Flop

- For a real D-FF
  - D input must be stable for a certain amount of time before the active edge of clock cycle => _____
  - D input must be stable for a certain amount of time after the active edge of the clock => _____
- Propagation time: from the time the clock changes to the time the output changes



Manufacturers provide minimum t_{su}, t_h, and maximum t_{plh}, t_{phl}

1.11 Setup and Hold Times – Maximum Clock Frequency Calculation

$t_{c\ max}$ - Max propagation delay through the combinational network

$t_{p\ max}$ - Max propagation delay from the time the clock changes to the flip-flop output changes { = max(t_{plh}, t_{phl})}

t_{ck} - Clock period

$$t_{c\ max} + t_{p\ max} \leq t_{ck} - t_{su}$$

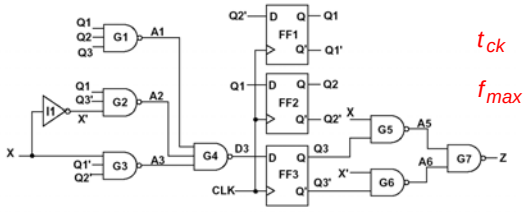
$$t_{ck} \geq t_{c\ max} + t_{p\ max} + t_{su}$$

Example:

$t_{p\ max} = 15\ ns$, $t_{su} = 5\ ns$, $t_{gate} = 15\ ns$

$$t_{ck} = 2 * 15 + 15 + 5 = 50\ ns$$

$$f_{max} = \frac{1}{50\ ns} = 20\ MHz$$



1.11 Setup and Hold Times – Hold Time Violations (from Q)

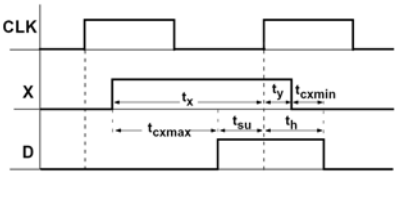
- Occurs if the change in Q that is fed back through the combinational network causes input D to change too soon after the clock edge

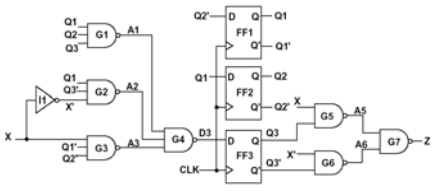
Hold time is satisfied if:

$$t_{p\ min} + t_{c\ min} \geq t_h$$

$t_{c\ min}$ -Min propagation delay through - the combinational network

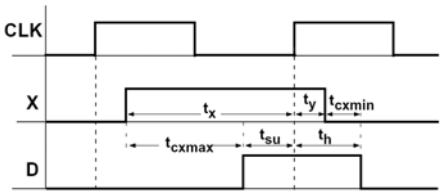
$t_{p\ min}$ - Min propagation delay from - the time the clock changes to - the flip-flop output changes - { = min(t_{plh}, t_{phl})}





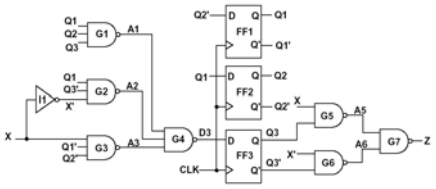
1.11 Setup and Hold Times – Setup Time Violations (from X)

- Occurs if the change in X that is fed back through the combinational network causes input D to change too soon after the clock edge



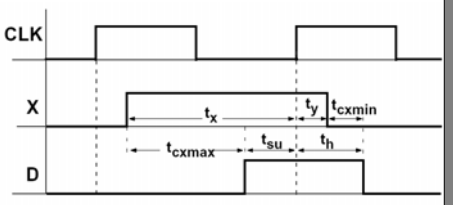
$$t_x \geq t_{cx\ max} + t_{su}$$

$t_{cx\ max}$ -Max propagation delay through -the combinational network from - input X (or any other input) to -the flip-flop input D



1.11 Setup and Hold Times – Hold Time Violations (from X)

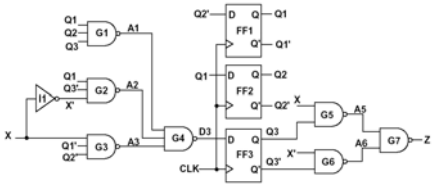
- Occurs if the change in X that is fed back through the combinational network causes input D to change too soon after the clock edge



Make sure that X does not change too soon after the clock.
If X changes at time t_y after the active edge, hold time is satisfied if

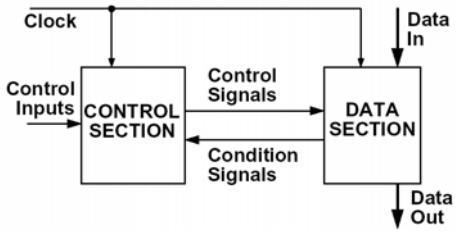
$$t_y \geq t_h - t_{cx\ min}$$

$t_{cx\ min}$ -Min propagation delay -through the combinational -network from input X (or any -other input) to the flip-flop input D



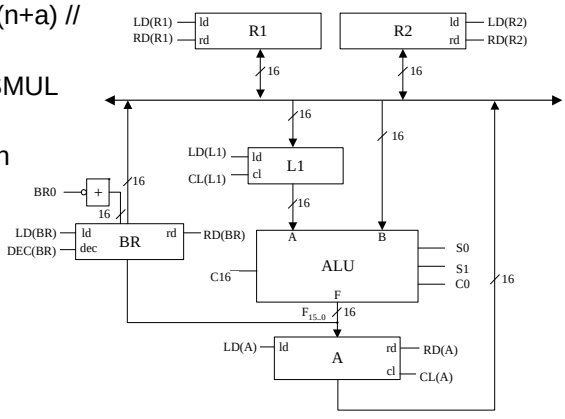
Synchronous Design

- Use a clock to synchronize the operation of all flip-flops, registers, and counters in the system
  - all changes occur immediately following the active clock edge
  - clock period must be long enough so that all changes flip-flops, registers, counters will have time to stabilize before the next active clock edge
- Typical design: Control section + Data Section



Synchronous Design Example

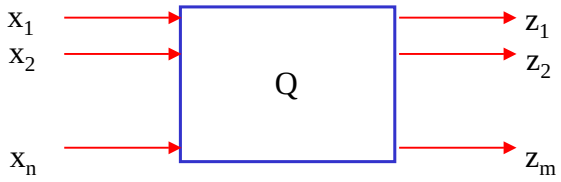
- Data section // s= n*(n+a) // R1=n, R2=a // R1=s
- Design flowchart for SMUL operation
- Design Control section
- S0 S1 F
  0 0 B
  0 1 B – C0
  1 0 B + C0
  1 1 A + B



Moore Sequential Networks

Outputs depend only on present state!

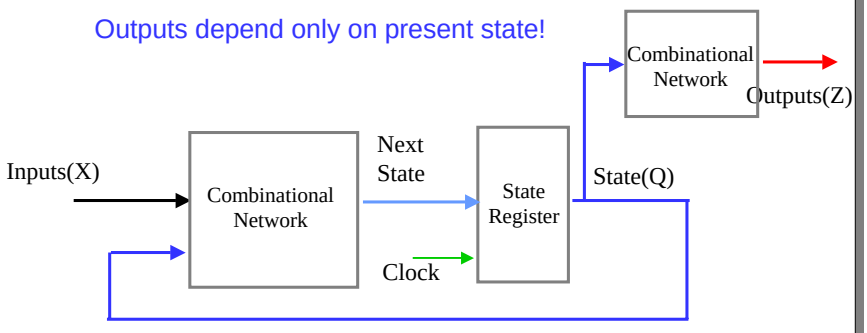
$X = x_1\ x_2 \ldots\ x_n$
$Q = Q_1\ Q_2 \ldots\ Q_k$
$Z = z_1\ z_2 \ldots\ z_m$



$$Z(t) = F(Q(t))$$
$$Q(t^+) = G(X(t), Q(t))$$

General Model of Moore Sequential Machine

Outputs depend only on present state!



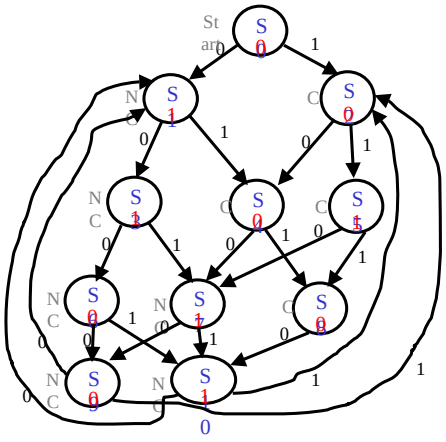
$X = x_1\ x_2 \ldots\ x_n$
$Q = Q_1\ Q_2 \ldots\ Q_k$
$Z = z_1\ z_2 \ldots\ z_m$

$$Q(t^+) = G(X(t), Q(t))$$
$$Z(t) = F(Q(t))$$

Code Converter: Moore Machine

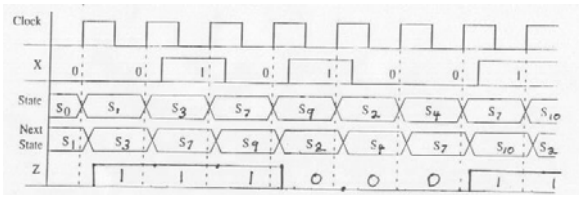
Moore Machine: State Table

PS	NS		Z
	X=0	X=1	
S0	S1	S2	0
S1	S3	S4	1
S2	S4	S5	0
S3	S6	S7	1
S4	S7	S8	0
S5	S7	S8	1
S6	S9	S10	0
S7	S9	S10	1
S8	S10	-	0
S9	S1	S2	0
S10	S1	S2	1

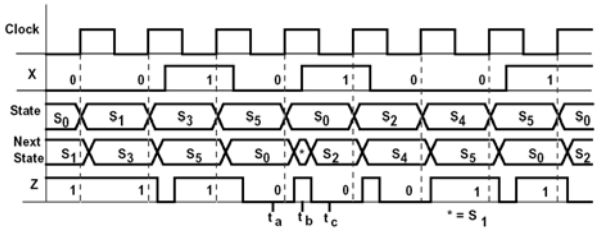


Moore Machine Timing

- X = 0010_1001 => Z = 1110_0011



Moore



Mealy

State Assignments

I. States which have the same next state (NS) for a given input should be given adjacent assignments (look at the columns of the state table).

II. States which are the next states of the same state should be given adjacent assignments (look at the rows).

III. States which have the same output for a given input should be given adjacent assignments.

Rule I: (S0, S9, S10), (S4, S5), (S6, S7)
Rule II: (S1, S2), (S3, S4), (S4, S5), (S6, S7), (S7, S8), (S9, S10)
Rule III: (S0, S2, S4, S6, S8, S9) (S1, S3, S5, S7, S10)

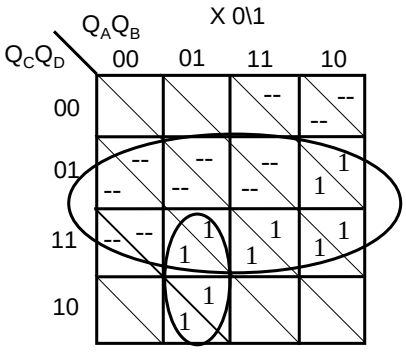
S0 – 0010
S1 - 0111
….
S10 - 0100

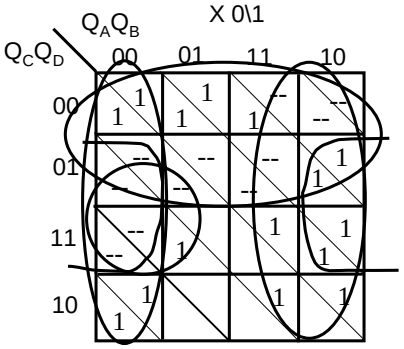
PS	NS		Z
	X=0	X=1	
S0	S1	S2	0
S1	S3	S4	1
S2	S4	S5	0
S3	S6	S7	1
S4	S7	S8	0
S5	S7	S8	1
S6	S9	S10	0
S7	S9	S10	1
S8	S10	-	0
S9	S1	S2	0
S10	S1	S2	1

State Assignments

PS	NS		Z
	X=0	X=1	
$Q_AQ_BQ_CQ_D$	$Q_AQ_BQ_CQ_D$	$Q_AQ_BQ_CQ_D$	
0 0 0 0	0 1 1 1	0 1 1 0	0
0 0 0 1	- - - -	- - - -	-
0 0 1 0	0 1 1 1	0 1 1 0	0
0 0 1 1	- - - -	- - - -	-
0 1 0 0	0 1 1 1	0 1 1 0	1
0 1 0 1	- - - -	- - - -	-
0 1 1 0	1 0 1 1	1 0 0 1	0
0 1 1 1	1 1 1 1	1 0 1 1	1
1 0 0 0	- - - -	- - - -	-
1 0 0 1	1 1 1 0	1 1 0 0	1
1 0 1 0	0 0 0 0	0 1 0 0	0
1 0 1 1	1 1 1 0	1 1 0 0	0
1 1 0 0	0 1 0 0	- - - -	0
1 1 0 1	- - - -	- - - -	-
1 1 1 0	0 0 0 0	0 1 0 0	1
1 1 1 1	1 0 1 0	1 1 1 0	1

Logic Reduction

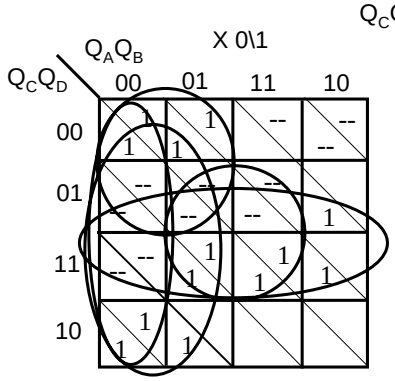


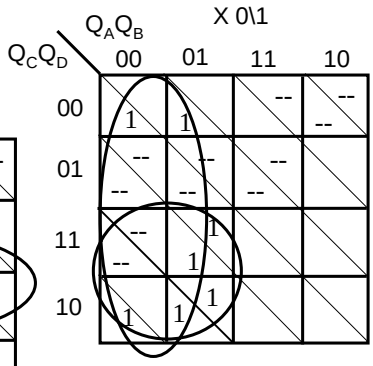


$$Q_A^+ = Q_D + Q_A'Q_BQ_C$$

$$Q_B^+ = Q_C' + Q_B'Q_D + X'Q_A' Q_D + Q_A'Q_B' + XQ_A$$

Logic Reduction

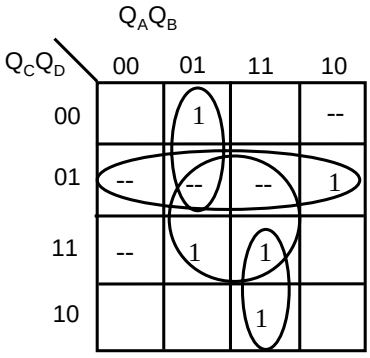




$Q_D^+ = X'Q_A' + Q_A'Q_BQ_C$

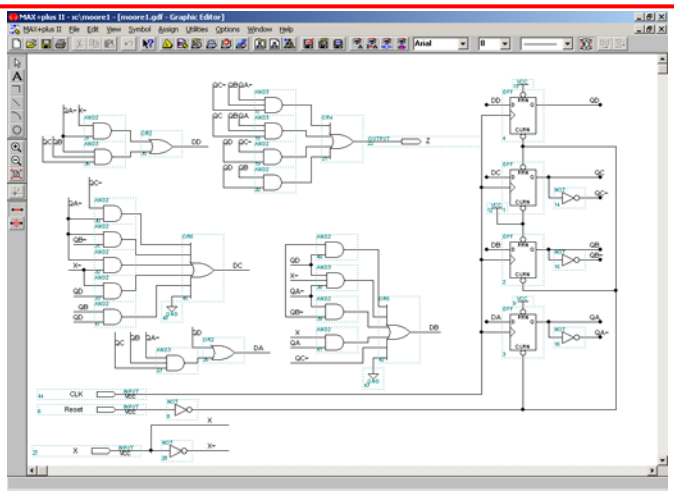
$Q_C^+ = X'Q_D + X'Q_A' + Q_A'Q_C' + Q_A'Q_B' + Q_BQ_D$

Logic Reduction

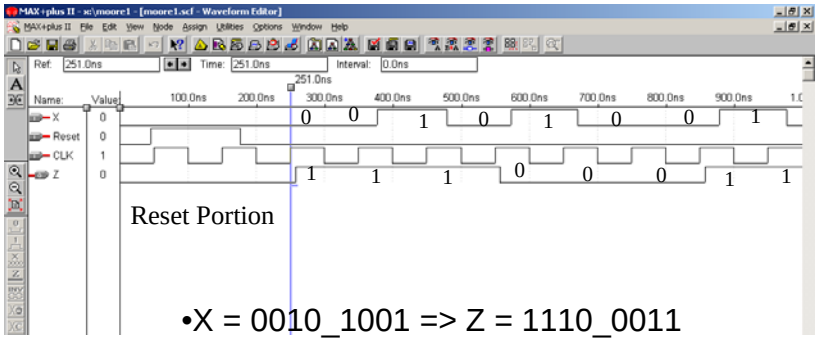


$Z = Q_A'Q_BQ_C' + Q_AQ_BQ_C + Q_C'Q_D + Q_BQ_D$

Altera Logic Implementation



Altera Simulation

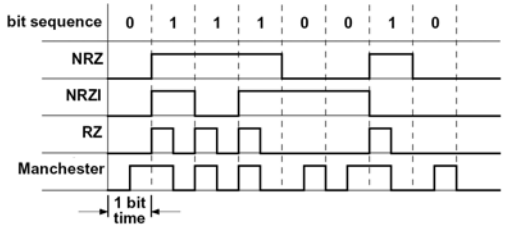


Reset Portion

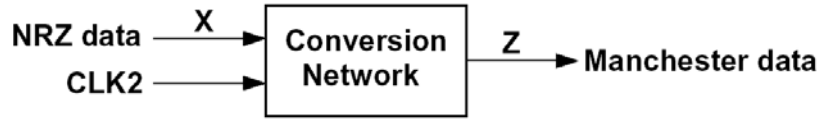
- X = 0010_1001 => Z = 1110_0011

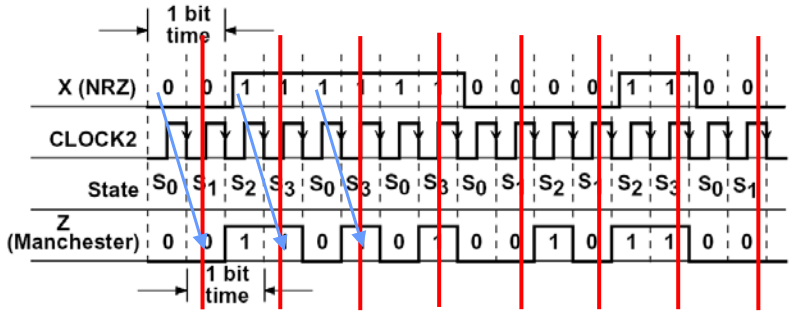
Moore Machine: Another Example – NRZ to Manchester Converter

- Coding schemes for serial data transmission
  - NRZ: _____
  - NRZI: _____
    - 0 in input sequence – _____ _____
    - 1 in input sequence – _____
  - RZ: return-to-zero
    - 0 – _____; 1 – _____
  - Manchester

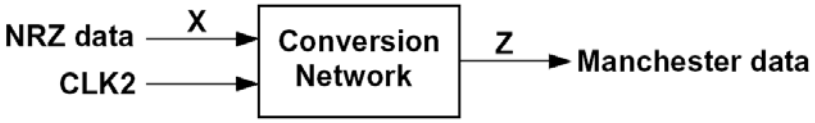


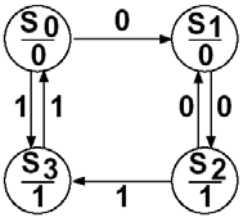
NRZ-to-Manchester Converter: Timing





NRZ-to-Manchester: State Diagram and Table



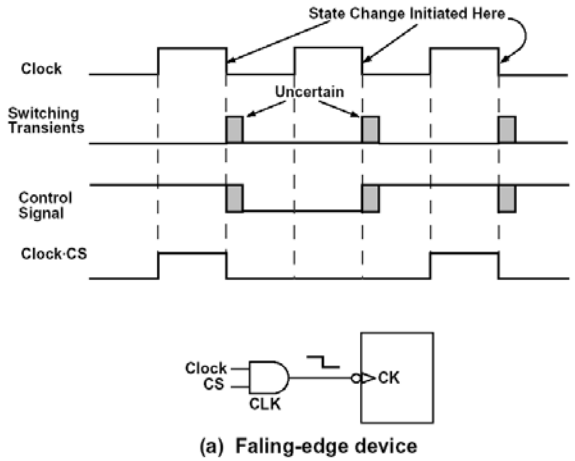


Present State	Next State X = 0	Next State X = 1	Present Output (Z)
S_0	S_1	S_3	0
S_1	S_2	–	0
S_2	S_1	S_3	1
S_3	–	S_0	1

1.12 Synchronous Design: Control Signal Timing Issues

- Change in state of the flip-flops in control section determined by the propagation delay
- Time control signals change depend upon this FF propagation delay and combinational network delay
- Glitches and spikes may occur in the control signals due to hazards in the network
- Noise may be introduced on the control signals by changing signals in another part of the circuit
- THIS MEANS THAT THERE IS A TIME INTERVAL AFTER THE ACTIVE EDGE OF THE CLOCK WHERE THE STATE OF THE CONTROL SIGNAL IS NOT KNOWN AND MAY NOT BE STABLE

1.12 Synchronous Design: Timing Chart for System with Falling-edge Devices



(a) Faling-edge device

1.12 Synchronous Design: Timing Chart for System with Rising-edge Devices

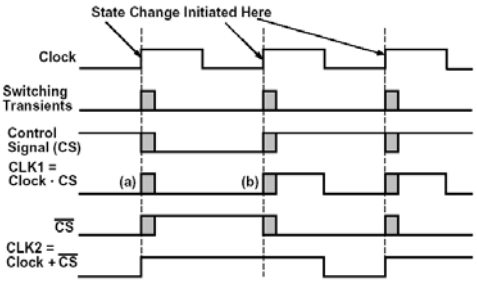


Figure 1-35 Incorrect Design

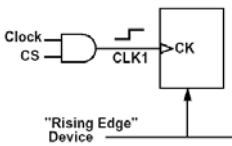
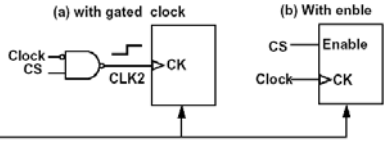


Figure 1-36 Correct Design



Principles of Synchronous Design

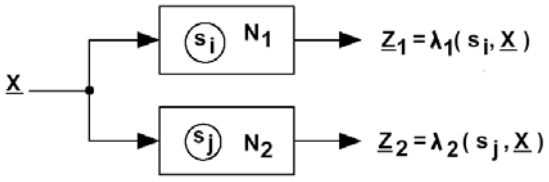
- Method
  - All clock inputs to flip-flops, registers, counters, etc., are driven directly from the system clock or from the clock ANDed with a control signal
- Result
  - All state changes occur immediately following the active edge of the clock signal
- Advantage
  - All switching transients, switching noise, etc., occur between the clock pulses and have no effect on system performance

Asynchronous Design

- Disadvantage - More difficult
  - Problems
    - Race conditions: _____
    - Hazards
  - Special design techniques are needed to cope with races and hazards
- Advantages = Disadvantages of Synchronous Design
  - In high-speed synchronous design propagation delay in wiring is significant => clock signal must be carefully routed so that it reaches all devices at essentially same time
  - Inputs are not synchronous with the clock – need for synchronizers
  - Clock cycle is determined by the worst-case delay

Equivalent States

- Two state are equivalent if we cannot tell them apart by observing input and output sequences



$s_i \equiv s_j$ iff $\underline{Z}_1 = \underline{Z}_2$ for every input sequence $\underline{X}$

Definition: Two states are equivalent $s_i == s_j$ if and only if, for every input sequence X, the output sequences Z1 and Z2 are the same.

Equivalent States

State Equivalence Theorem

- Two state are equivalent $S_i == S_j$ if and only if for every single input X, the outputs are the same and the next states are equivalent

Implication Table Method

1. Construct a chart that contains a square for each pair of states.
2. Compare each pair in the state table. If the outputs associated with states i and j are different, place an X in square i-j to indicate that i!=j.
   If outputs are the same, place the implied pairs in square i-j.
   If outputs and next states are the same (or i-j implies only itself), i==j.
3. Go through the implication table square by square.
   If square i-j contains the implied pair m-n, and square m-n contains X, then i!=j, and place X in square i-j.
4. If any Xs were added in step 3, repeat step 3 until no more Xs are added.
5. For each square i-j that does not contain an X, i==j.

State Table Reduction

Present State	Next State X = 0	Next State X = 1	Present Output X = 0	Present Output X = 1
a	c	f	0	0
b	d	e	0	0
c	~~h~~ a	g	0	0
d	b	g	0	0
e	e	b	0	1
f	f	a	0	1
g	c	g	0	1
~~h~~	~~e~~	~~f~~	~~0~~	~~0~~

1) States a and h have the same next states and outputs (when X=0 and X=1)
2) Eliminate h from the table and replace with a
3) States a and b have the same output => they are same iff c==d and f==e.
   We say c-d and e-f are implied pairs for a-b.
   To keep track of the implied pairs we make an implication chart.

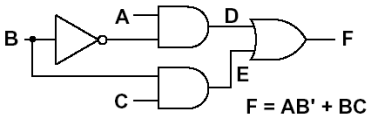
State Table Reduction

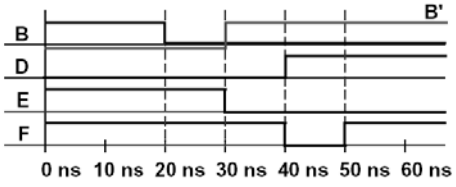
Present State	Next State X = 0	1	Present Output X = 0	1
a	c	f	0	0
b	d	e	0	0
c	~~h~~ a	g	0	0
d	b	g	0	0
e	e	b	0	1
f	f	a	0	1
g	c	g	0	1
~~h~~	~~e~~	~~f~~	~~0~~	~~0~~

4) Make another pass through the chart.
E-g cell contains c-e and b-g;
since c-e cell contains x, c!=e => e!=g (put X).
5) Repeat the step 4 until no additional squares are X-ed. {Put X in f-g, a-c, a-d, b-c, b-d squares}.
6) The remaining squares indicate equivalent state pairs => a==b, c==d, e==f.

Hazards in Combinational Networks

- What are hazards in Combinational Networks?
  - Unwanted switching transients at the output (glitches)
- Example
  - ABC = 111, B changes to 0
  - Assume each gate has propagation delay of 10ns





Hazards in Combinational Networks

- Occur when different paths from input to output have different propagation delays
- Static 1-hazard
  - a network output momentarily go to 0 when it should remain a constant 1
- Static 0-hazard
  - a network output momentarily go to 1 when it should remain a constant 0
- Dynamic hazard
  - if an output change three or more times, when the output is supposed to change from 0 to 1 (1 to 0)

Hazards in Combinational Circuits

Why do we care about hazards?

- Combinational networks
  - don't care – the network will function correctly
- Synchronous sequential networks
  - don't care - the input signals must be stable within setup and hold time of flip-flops
- Asynchronous sequential networks
  - hazards can cause the network to enter an incorrect state
  - circuitry that generates the next-state variables must be hazard-free
- Power consumption is proportional to the number of transitions

Detection of Static 1 and 0 Hazards

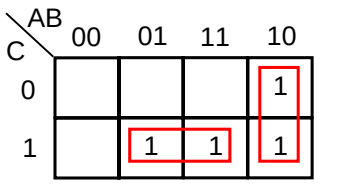
- We only consider hazards which occur when a single input variable changes
- Analysis begins by determining the *Transient Output Function*, F^t, which represents the behavior of the network under transient conditions.

  The *Transient Output Function*, F^t, is determined in the same way as ordinary (steady-state) output Functions except each variable and its complement are treated as independent variables because under transient conditions these variables may assume the same values.

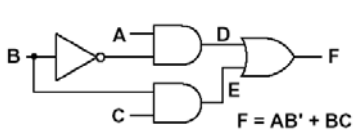
- Boolean manipulation of F^t involves the use of theorems that do not involve a variable and its complement on the same side of the expression.

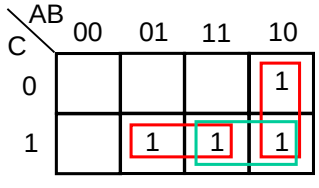
  Allowed Theorems: Associate Law, Distributive Laws, DeMorgan's Law, XX=X, X+XY = X, etc.

Hazards in Combinational Circuits

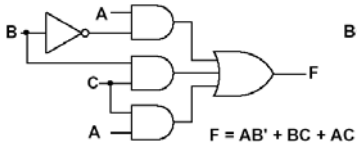


$f = AB' + BC$





$f = AB' + BC + AC$



To avoid hazards:
every pair of adjacent 1s should be covered by a 1-term

Detection of Static 1 Hazards

Each product term of F^t is called a 1-term.

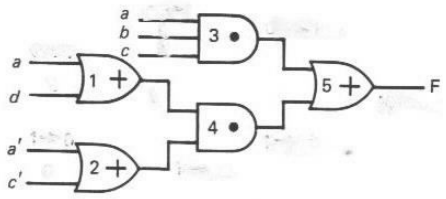
The 1-terms of F^t are plotted on a Karnaugh map

If two 1's in adjacent squares on the map of F^t are covered by the same 1-term, changing the input to the network between the corresponding two input states cannot cause a hazard.

If two 1's in adjacent squares on the map are not covered by a single 1-term, a hazard is present.

Detection of Static 1 Hazards

$$F^t = abc + (a+d)(a' + c') = abc + aa' + ac' + a'd + c'd$$

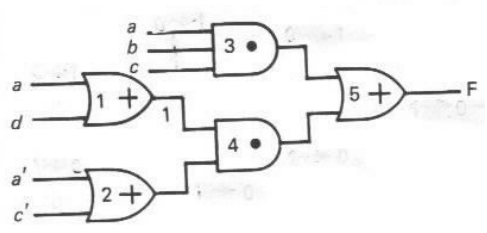


Detection of Static 0 Hazards

- Each sum term of F^t is called a 0-term.
- The 0-terms of F^t are plotted on a Karnaugh map
- If two 0's in adjacent squares on the map of F^t are covered by the same 0-term, changing the input to the network between the corresponding two input states cannot cause a hazard.
- If two 0's in adjacent squares on the map are not covered by a single 0-term, a hazard is present.

Detection of Static 0 Hazards

$F^t = abc + (a+d)(a' + c') = (a+d)(a+a'+c')(b+a'+c')(c+a'+c')$



Design of Hazard Free Networks

To design a network that is free of static and dynamic hazards, the following procedure may be used:

1. Find a sum-of-products expression (F^t) for the output in which every pair of adjacent 1s is covered by a 1-term. (The sum of all prime implicants will always satisfy this condition.) A two-level AND-OR network based on this F^t will be free of 1-, 0-, and dynamic hazards.
2. If a different form of network is desired, manipulate F^t to the desired form by simple factoring, DeMorgan's laws, etc. Treat each x_i and x_i' as independent variables to prevent introduction of hazards.

Alternatively, you can start with a product-of-sums expression in which every pair of adjacent 0s is covered by a 0-term.