Analysis of Asynchronous Sequential networks

- Issues in design of machines (hardware and software) always shift in response to evolution intechnology
- VLSI / ULSI promises high performance as scaling down the digital IC process... but benefits of faster devices / gates can't be fully exploited due to other fundamental limitations
- already system clock speeds lag behind logic speeds
  - gate delays <1ns in advanced CMOS processes where clock rates of >50MHz, in GHz, are very hard to achieve
- Asynchronous design (which does not require an external c clock signal is expetec to give better performance.

Limitations of clock speed

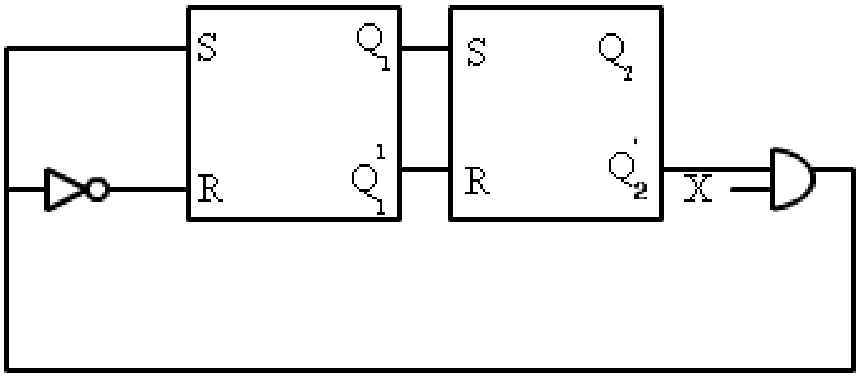
- clock skew: the phase difference of a global synchronization signal (clock) at different locations in the system
- to reduce clockskew, there exist special cad tools just of rproducing proper clock signal distribution/ routing on the chip
- asynchronous design has not been extensively used

Asynchronous sequential networks

- when an input changes, the state of the network changes without waiting for a clock pulse (in fact the input changed may even cause a sequence of state changes)
- design of this asynchronous network is more difficult than synchronous networks, because of **timing problems**
- in synchronous systems, we wait long enough until all input changes for all flip-flops reach a steady state value before applying the clock pulse
- to simplify we assume that the asynchronous network operates in fundamental mode.
  - which assumes that input signals will be changed only when the circuit is in a stable condition (i.e.. no internal signals are changing)
  - all input signals are considered to be levels
- in a different type of asynchronous network the inputs are pulses, not levels
- pulse-mode networks (similar to synchronous sequential instead of clock pulses,they use input pulses to trigger state changes)
  - restrictions below are used
  - input pulses are all of proper duration
  - separation between input pulses are sufficient so network can react to one pulse before 2nd pulse occurs.
- we'll first analyze some asynchronous circuits by first building transition tables

- these transition tables are like the state tables for a clocked synchronous circuit except for the synchronous case the internal state changes when the clock pulse occurs, while for the asynchronous case the internal state change occurs immediately following the input change as fast as the circuit response will permit.
- METHOD :given a circuit based on flip-flops , determine if there are any critical races.
  - from diagram, write down equations
  - form a transition table using the equations, starting in a stable state.
  - whenever two or more flipflops must change state in response to a single change in the input the result is a ***RACE***
  - if the resulting stable state is the same no matter in what order the flipflops changed, we call it a **NonCritical race**
  - if it's possible to end up in 2 or more different stable states depending upon in which order the flip flops changed then it's a **Critical Race**
- EXAMPLE:

Given the following circuit below, determine if there are any critical races.



Note: SR-latch on left is Q1, on right is Q2.

- from diagram, write down equations

Q1+ = S1 + R1' Q1 = X Q2' + (X Q2')'' Q1 = X Q2'

Q2+ = S2 + R2' Q2 = Q1 + Q1 Q2 = Q1

- form a transition table using the equations, starting in a stable state.
  - Here we start in state X Q1 Q2 = 0 0 0 .
  - Then we let X=0 -> 1, and move from Q1 Q2 = 00 => 10 =>11 =>01=>00=>10=>11=> etc... we continue in this cycle until X goes back to 0. We put this information into ~~state~~ transition table now...
  - (we can show arrows in state table to illustrate the transitions)

Q1+ Q2+		
Q1 Q2 \ X	0	1

00	00	10
01		00
11		01
10		11

- now we must complete the table: ie. Q1 Q2 = 10 and X changes from 1 back to 0
- X=1=>0 Q1 Q2=10=>01 =>00, so we can fill in table as :

Q1+ Q2+		
Q1 Q2 \ X	0	1
00	00	10
01	**00**	00
11		01
10	**01**	11

- and continue , say Q1 Q2=11 and X=1=>0, Q1 Q2 = 11=>01=>00

Q1+ Q2+		
Q1 Q2 \ X	0	1
00	00	10
01	**00**	00
11	**01**	01
10	**01**	11

- ,now it's a complete transition table. But notice , X=0 Q1 Q2 = 10 => Q1+ Q2+ = 01 , a RACE !
- In this example, we have a NONCRITICAL race, since we end up in stable state 00 whether we go through transitions Q1 Q2 = 10=>11=>01=>00 (where Q2 changes first) OR 10=>00 (where Q1 changes first).

Example of Critical Race: can be found in transition table below, describing a *different* circuit

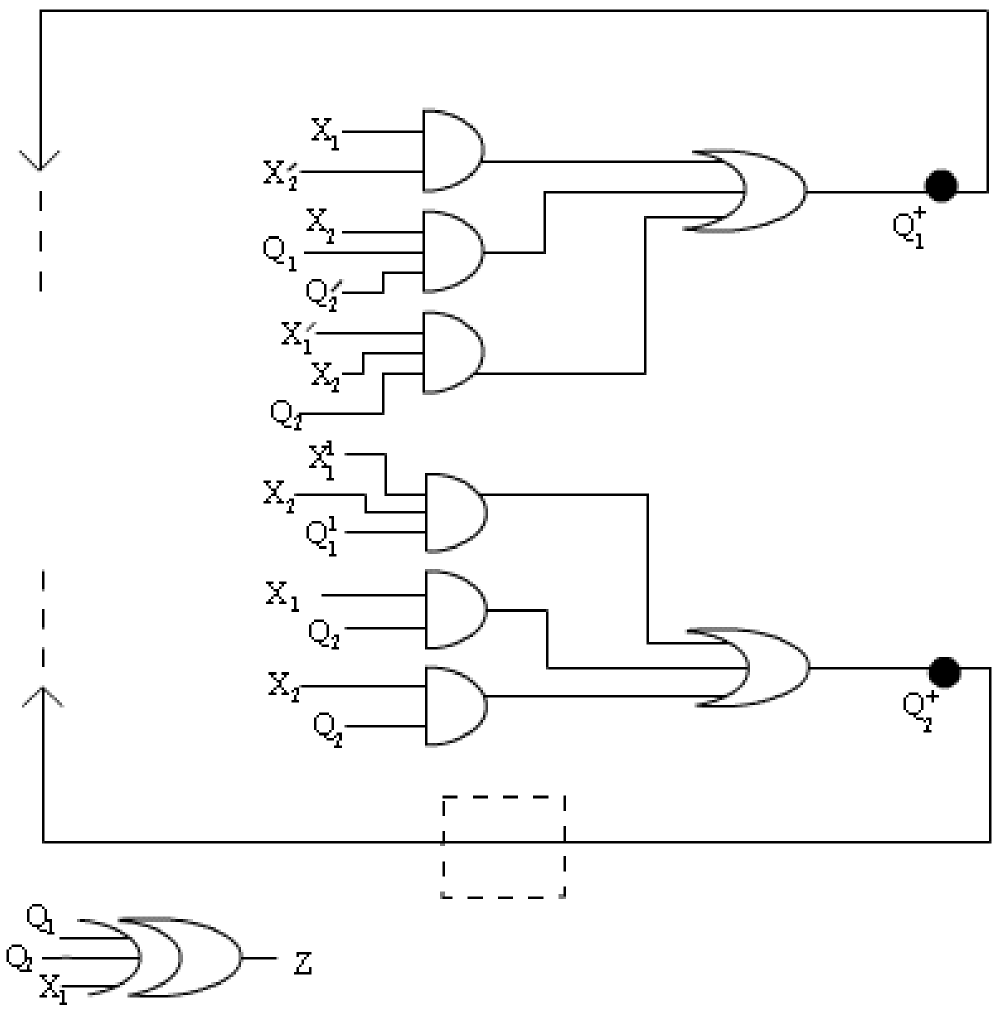
• Q1+ Q2+		
Q1 Q2 \ X	0	1

00	**00**	**11**
01	00	***01***
11	01	***11***
10	10	***10***

Starting in Q1 Q2 = 0 0 and X=0=>1 we have a RACE since Q1 Q2= **00 => 11**

- If Q2 changes first: 00=> 01 we end up in ***01***.
- If Q1 changes first 00=>10 we end up in ***10.***
- If Q1 and Q2 change simultaneously we end up in state ***11.***
- THEREFORE we have a critical race!

Another Example: Given a circuit based on logic gates only:



we assume there's a **delay** on these wires and represent our state variables Q1, Q1+, and Q2, Q2 +.

- Given wires representing state variables, Q1, Q2.

- construct " Excitation " equations,

  $Q1+ = X1\ X2' + X1'\ X2\ Q2 + X2\ Q1\ Q2'$

  $Q2+ = X1'\ X2\ Q1' + X1\ Q2 + X2\ Q2$

  $Z = X1 \oplus Q1 \oplus Q2$

- transition table

Q1 Q2\X1 X2	Q1 + Q2 +			
	00	01	11	10
00	**00**	01	**00**	10
01	00	11	**01**	11
11	00	**11**	01	**11**
10	00	**10**	**10**	**10**

- circle stable states (next internal state equals present stable state, ie. Q1+ = Q1, Q2+ = Q2)
- a change of input state corresponds to change between columns of table not rows
- if new total state is unstable a row change (internal state changes) takes place to the row with the corresponding next internal state.
  - i.e. start in X1 X2 Q1 Q2 = 0000, if input changes to X1 X2 = 01 then internal state changes to 01 and then to 11 .
  - NOTE: this single input change caused two internal state changes before a stable total state was reached.
- can convert transition table to a state table, where each internal state or each stable total state is labeled with arbitrary designation
- note state table for clocked synchronous is like asynchronous except internal state change occurs for clock pulse whereas asynchronous changes state for input change
- label internal states in "flow" table: (Q1 Q2 or Q1+ Q2+ = 00 : 1, 01:2, 11:3, 10:4)

Q1 Q2 \X1 X2	Q1 + Q2 +			
	00	01	11	10
1	**1**	2	**1**	4
2	1	3	**2**	3
3	1	**3**	2	**3**
4	1	**4**	**4**	**4**

we can also add output signal designation with ***state, output signal*** in each box of table, or put output signal in separate table, as shown below.

Q1 Q2 \X1 X2	Z			
	00	01	11	10
1	0	0	1	1
2	1	1	0	0
3	0	0	1	1
4	1	1	0	0

Synthesis of Fundamental Mode Asynchronous Sequential networks

- To synthesize:
  - we derive primitive flow table
  - reduce flow table to minimum number of rows
- Primitive flow table has exactly 1 stable total state per row
- Assume:
  - only 1 input variable changes at a time
  - and reach stable state before next input changes

Example: a network X1, X2 inputs, Z output. input sequence 00=> 01 => 11 causes output to => 1 , next input change causes output to go to 0. No other sequences causes output to go to 1.

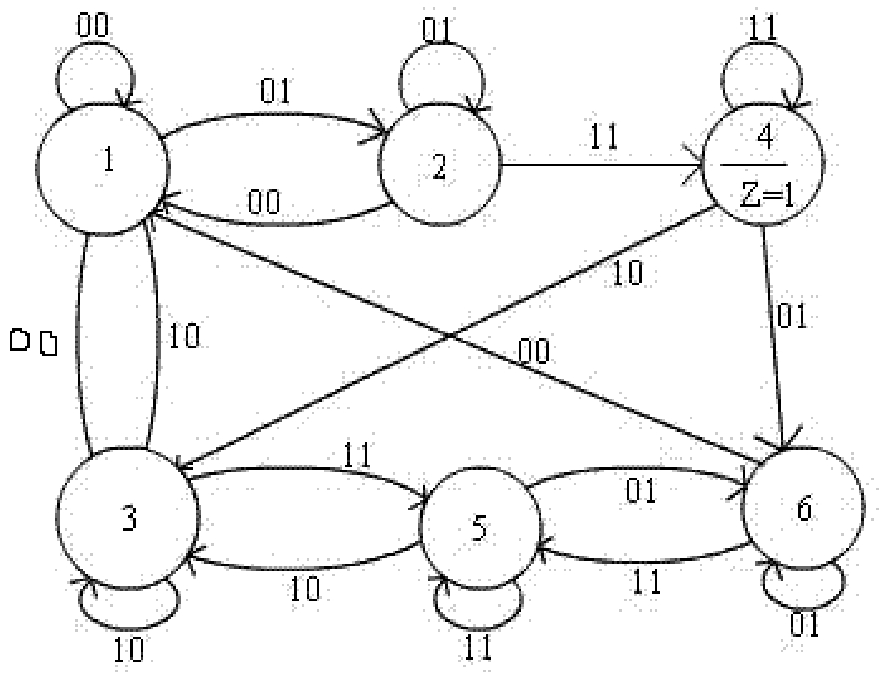
- First we create a state table:
  - then primitive flow table (plot equations in flow table) shaded boxes in table indicate a stable total state., otherwise state is a unstable total state. (note: 00 => 11 is not allowed).

	X1 X2				
	00	01	11	10	Z
reset 1	1	2	-	3	0
(00,01) 2	1	2	4	-	0
* 3	1	-	5	3	0
(00,01,11) 4	-	6	4	3	1
* 5	-	6	5	3	0

*6	1	6	5	-	0

* : states cannot lead to a 1 output without first resetting.

- circle stable states (next internal state equals present stable state, ie. Q1+ = Q1, Q2+ = Q2)
- (PRIMITIVE: state must change each time input changes.)
- Note all arrows leading into a state must have the same inputs.



- so now we have the primitive flow table, next we proceed to "Reduce the flow table" or obtain a minimum number of states (rows in table).
- Two steps are required:
  - find minimum row primitive flow table
  - further reduce table by merging rows

How to do this? eliminate redundant stable total states: we can do this if

(i) inputs are the same, (ii) outputs are the same (iii) next states are the same for each possible next input. As an ***example*** consider table below (which is ***representing a* Different *circuit***):

	00 01 11 10	z1 z2
1	1 7 - 4	11
2	2 5 - 4	01
3	- 7 3 11	10

4	2 - 3 4	00
5	6 5 9 -	11
6	6 7 - 11	01
7	1 7 14 -	10
8	8 12 - 4	01
9	- 7 9 13	01
10	- 7 10 11	10
11	8 - 10 11	00
12	6 12 9 -	11
13	8 - 14 13	11
14	- 12 14 11	00

colors indicate rows (potentially equivalent stable total states (2,6,8),(5,12),(3,10),(4,11), have same outputs, next you check the next states (iii)-criteria: here 6 cannot merge with 2 and 8, -'s must line up also) so set of states which can merge are shown with colors above. These can be merged so table becomes, as below, where equivalent states are 2=8, 4=11, 5=12, 3=10 :

	00 01 11 10	z1 z2
1	1 7 - 4	11
2	2 5 - 4	01
3	- 7 3 11	10
4	2 - 3 4	00
5	6 5 9 -	11
6	6 7 - 11	01
7	1 7 14 -	10
~~8~~	~~8 12 - 4~~	~~01~~
9	- 7 9 13	01
~~10~~	~~- 7 10 11~~	~~10~~
~~11~~	~~8 - 10 11~~	~~00~~
~~12~~	~~6 12 9 -~~	~~11~~
13	8 - 14 13	11
14	- 12 14 11	00

- next we convert our moore table to a mealy table. We do this for asynchronous networks only by associating output for each row with the corresponding stable total state in that row.
- Now , back to our example we're trying to synthesize:

	X1 X2				
	00	01	11	10	Z
reset 1	1	2	-	3	0
(00,01) 2	1	2	4	-	0
* 3	1	-	5	3	0
(00,01,11) 4	-	6	4	3	1
* 5	-	6	5	3	0
*6	1	6	5	-	0

so we proceed to rewrite table in mealy form:

•

	X1 X2				Z			
	00	01	11	10	00	01	11	10
1	1	2	-	3	0	-	-	-
2	1	2	4	-	-	0	-	-
3	1	-	5	3	-	-	-	0
4	-	6	4	3	-	-	1	-
5	-	6	5	3	-	-	0	-
6	1	6	5	-	-	0	-	-

(note when networks are stable outputs from mealy and moore tables will be the same.)

- Now, 2 rows of a reduced primitive mealy flow table are compatible and can be merged if and only if there are no state conflicts in any column
  - i.e. merge rows 1 and 2 above, rows 2 and 3 cannot be merged, 4 does not equal 5 (since we have already removed all redundant stable total states).
- to help out in determining how one can merge a maximum number of rows we can form a graph, where
  - nodes=states
  - edges=states that can be merged
  - ie. we have (1,2),(1,3),(6,3),(5,6),(5,3) are 5 edges of graph and 4 is a node with no edges attached to it.
  - now problem is to partition graph into a minimum number of cliques (for each pair of nodes in subgraph, there is always an edge, and it has a maximum number of edges from the graph)
  - Solution of this graph :
    - we could merge (5,6) and (1,3) to get 4 rows
    - or **(3,5,6)** and **(1,2)** to get only 3 rows, a better solution!
- So we get

	00 01 11 10	00 01 11 10
a (1,2)	***1 2*** 4 3	0 0 - -
b (3,5,6)	1 ***6 5 3***	- 0 0 0
c (4)	- 6 ***4*** 3	- - 1 -

- so now we have shown how to form a primitive flow table and how to reduce it to a minimum-row mealy table
- next we must make a state assignment and then realize the circuit with gates and flipflops or just gates.
- change table to list just states a,b,c
- Perform state assignment
  - we need a RACE-FREE state assignment
  - nodes=state, state to state transition is an edge of graph
  - assign binary state to each node such that edges represent only at most one state variable being changed (ie. at most one state variable changes during each state transition, so we don't have any races)
  - Often we must add a node to make this possible, such as in this example (a,b,c) forms a triangle graph, which cannot be labeled with this characteristic (ie. a: 00, b:01, c:11, is not good for state transition a-c)
  - So we add a node (corresponding to a row in table) a-00 , b-01 , d-11 , c-10

	00 01 11 10	00 01 11 10
00 a	***00 00*** 10 01	**0** 0 - -
01 b	00 ***01 01 01***	- **0** 0 0
11 d	- 01 - -	- - - -
10 c	- 11 ***10*** 00	- - 1 -

where we redirect transitions from row c to row b through row d.

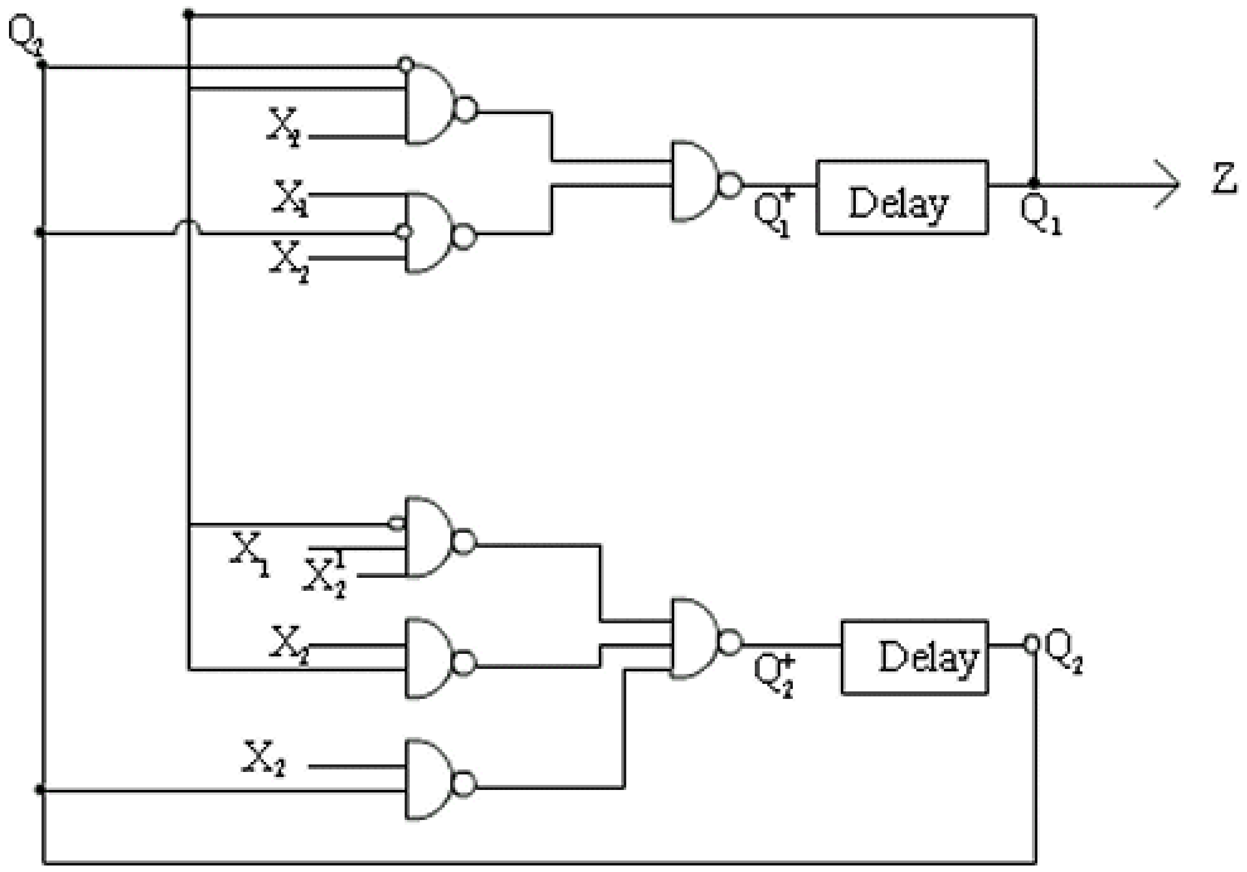
- now how do we complete the output table? (note we have an output for each total stable state. now we have to assign '-' don't care or 0 or 1 to other outputs)
- we want to prevent intermediate output transitions 0=>1=>0 (or false outputs).
  - ie. in table when X1 X2 Q1 Q2 = 0101 =>0000, the outputs go from 0 => - => 0. we wish to make sure it does not make transition 0=>1=>0, so we must change the don't care '-' to a '0'. So table is modified

	00 01 11 10	00 01 11 10
00 a	***00*** ***00*** 10 01	**0** 0 - 0
01 b	00 ***01*** ***01 01***	**0 0** 0 0
11 d	- 01 - -	- - - -
10 c	- 11 ***10*** 00	- 1 1 -

similarly we fixed output specification for case X1 X2 Q1 Q2 =1110 =>0101,

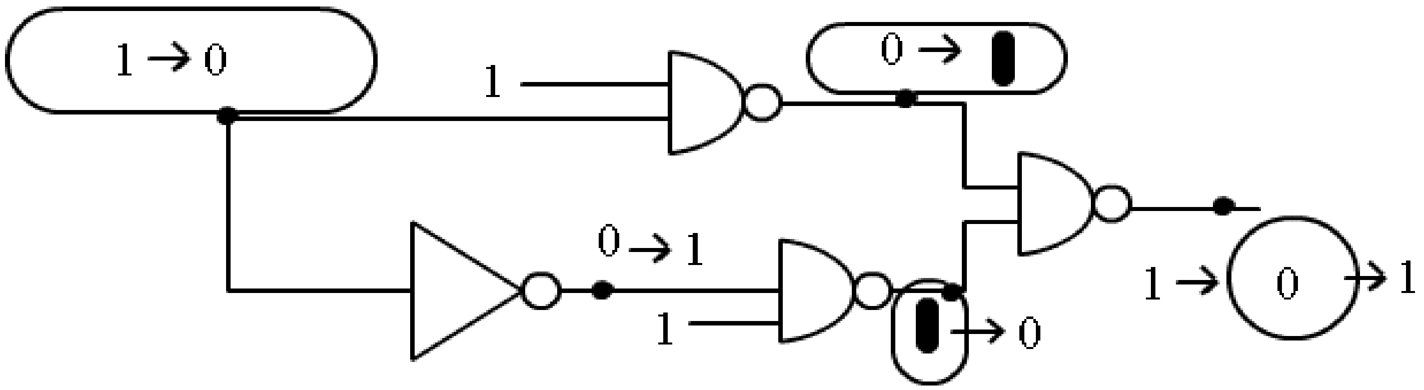
- now we can proceed: We have a RACE free state assignment, ready to be implemented:
- create next state Kmaps and design circuit: to get
  - Q1+ = X1 X2 Q2' + X2 Q1 Q2'
  - Q2+ = X1 X2' Q1' + X1' Q1 + X2 Q2

- thus equations define circuit with Q1,Q2 feedback wires into gates, and Z = Q1 .



Hazards

- when input to combinational network changes, unwanted switching transitions may occur
- why? because paths through network have different propagation delays



- if for some input changes and combination of propagation delays an output momentarily goes to zero when it should remain a constant 1, we say that the network has a static -1 hazard... vica versa for static - 0 hazard
- if output changes 3 or more times we say that the network has a **dynamic hazard**
- note hazards can cause a combinational part of a sequential circuit to malfunction

Detection of static -0 and -1 hazards

- first we determine the transient output function Ft
  - we get this the same way we did before except we treat X and X' as separate variables
  - because in transient, X and X' may have the same value
  - note therefore we cannot use several theorems of boolean algebra
  - we cannot use xx' = 0, x + x' = 1, x + x' y = x+y,etc...
  - but we can use associative, distributive, deMorgan's, and xx=x,x+xy=x,etc...
  - if Ft is in sum of products form then each product term is called a ***1-term of F***
- by plotting k-map, if there is no 1-term which includes both input states (for which the input states are adjacent and for which Ft is 1) of the pair , a 1-hazard is present
- if Ft sum of products form, does not contain the product of a variable and its complement **no 0-hazards are present**, else a 0-hazard may be present
  1. obtain product-of-sums form for Ft
  2. examine each pair of adjacent input states for which Ft is 0
  - if there is no 0-term which includes both input states of the pair, a 0-hazard is present (plot 0-

terms) on k-map

- Example: Ft = abc + a a' + a c' + a' d + c' d = (a+d)(a+a'+c')(b+c'+a')(c+a'+c')

cd/ab	00	01	11	10
00			1	1
01	1	1	1	1
11	1	***1***	***1***	
10			1	

,0111=>1111 not covered by a single term therefore can cause a 1-hazard: **(a'd)=>1=>0 and (abc)=>0=>1: BOTH at 0.**

cd/ab	00	01	11	10
00	0	0		
01				
11				0
10	***0***	0		***0***

,0010=>1010 a 0-hazard is present : **(a+d)=>0=>1 and (a'+b+c')=>1=>0 : BOTH at 1.**

Designing hazard-free combinational networks

- based on two theorems
- Theorem 1: if the 1-terms of Ft satisfy the following conditions, the network will not contain static/ dynamic hazards
  - for each pair of adjacent input states that produce 1 output, there is at least one 1-term that includes both input states of the pair
  - there are no 1-terms that contain exactly one pair of complementary literals
- Theorem 2: if the 0-terms of Ft satisfy the following conditions, the network will not contain static/ dynamic hazards:
  - for each pair of adjacent input states that produce 0 output, there is at least 1 0-term that includes both input states of the pair
  - there are no 0-terms that contain exactly 1 pair of complementary literals.