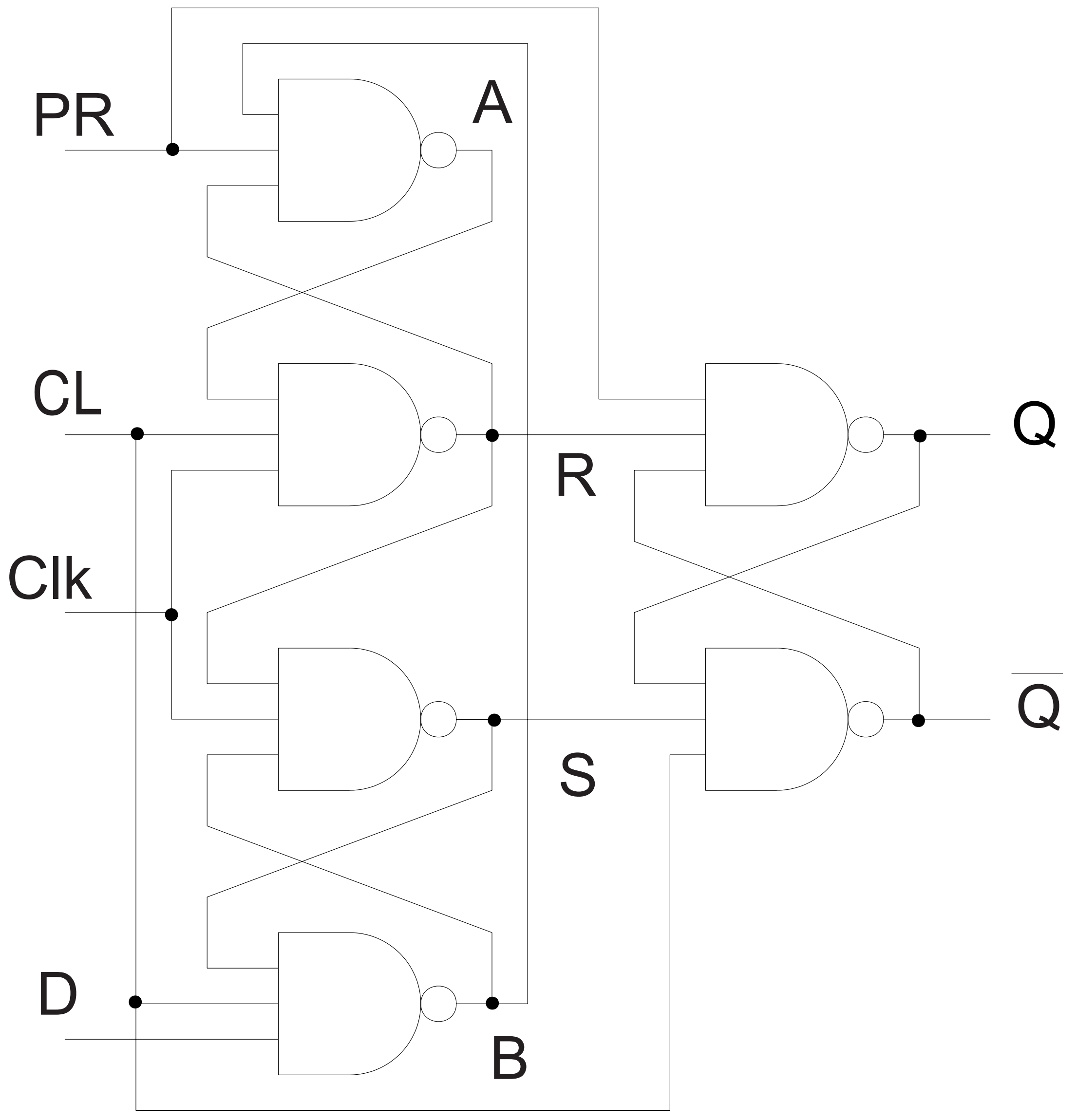
Fundamental-Mode Sequential Circuits

- How do flip-flops work?
- How to analyse behaviour of flip-flops?
- How to design fundamental-mode circuits?
- Fundamental mode restriction - only one input can change at a time; circuit must be stable before next input can change.

Edge-Triggered D Flip-Flop



How does it work?

How was it designed?

Edge-Triggered D Flip-flop

- Intuitive explanation:

- clock=0 => R=S=1, CL=PR=1, Q and -Q are held.
  either D=1 => A=1, B=0
  or D=0 => A=0, B=1
  D can change but Q and -Q are held.

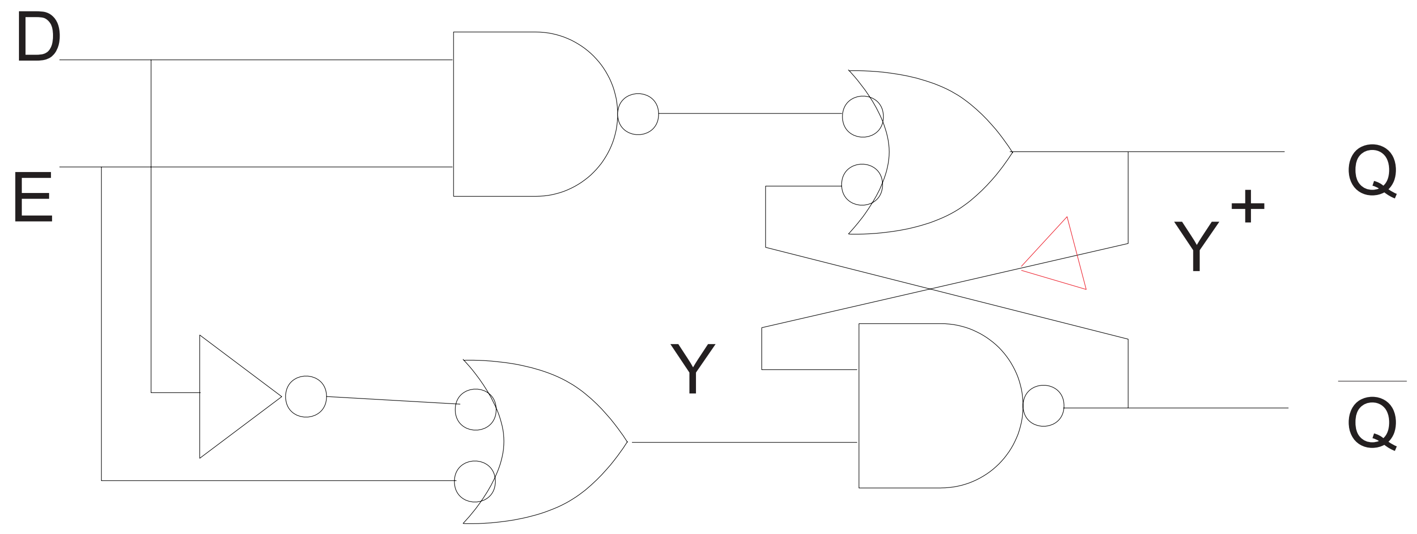
- Let clock change 0->1
  Either B=1, A=0 => R=1, S=0 => Q=0, -Q=1
  or B=0, A=1 => R=0, S=1 => Q=1, -Q=0

- What if D changes while clock=1?
  a) D was 0 and changes to 1
    A=0, B=1 => R=1, S=0
    as S=0, change in D does not change B.
  b) D was 1 and changes to 0
    A=1, B=0 => R=0, S=1
    B changes from 0 to 1, but A does not change.

- Falling clock edge has no effect. Fundamental mode restriction applies.

Formal Analysis

- D latch:



To analyse this circuit, break the feedback loop.
It is convenient to pretend that all the gates have 0 delays and that there is a finite but small delay in the feedback loop.

Analysis of D latch

- Y^+ can be thought of as the next state.
- $Y^+ = D.E + Y.(D + \overline{E})$

Transition Table

Y	DE 00	01	11	10
0	0	0	1	0
1	1	0	1	1

Y^+

State Table

	DE			
S	00	01	11	10
S0	S0	S0	S1	S0
S1	S1	S0	S1	S1

S*

Stable states - next state is the same as present state (circled).

State & Output Table

S	DE 00	01	11	10
S0	$\boxed{S0}$,01	$\boxed{S0}$,01	S1 ,11	$\boxed{S0}$,01
S1	$\boxed{S1}$,10	S0 ,01	$\boxed{S1}$,10	$\boxed{S1}$,10

S*, Q $\overline{Q}$

$$Q = D.E + Y.\overline{E} + Y.D$$

$$\overline{Q} = \overline{D}.E + \overline{Y}$$

State Table

D changes to 1

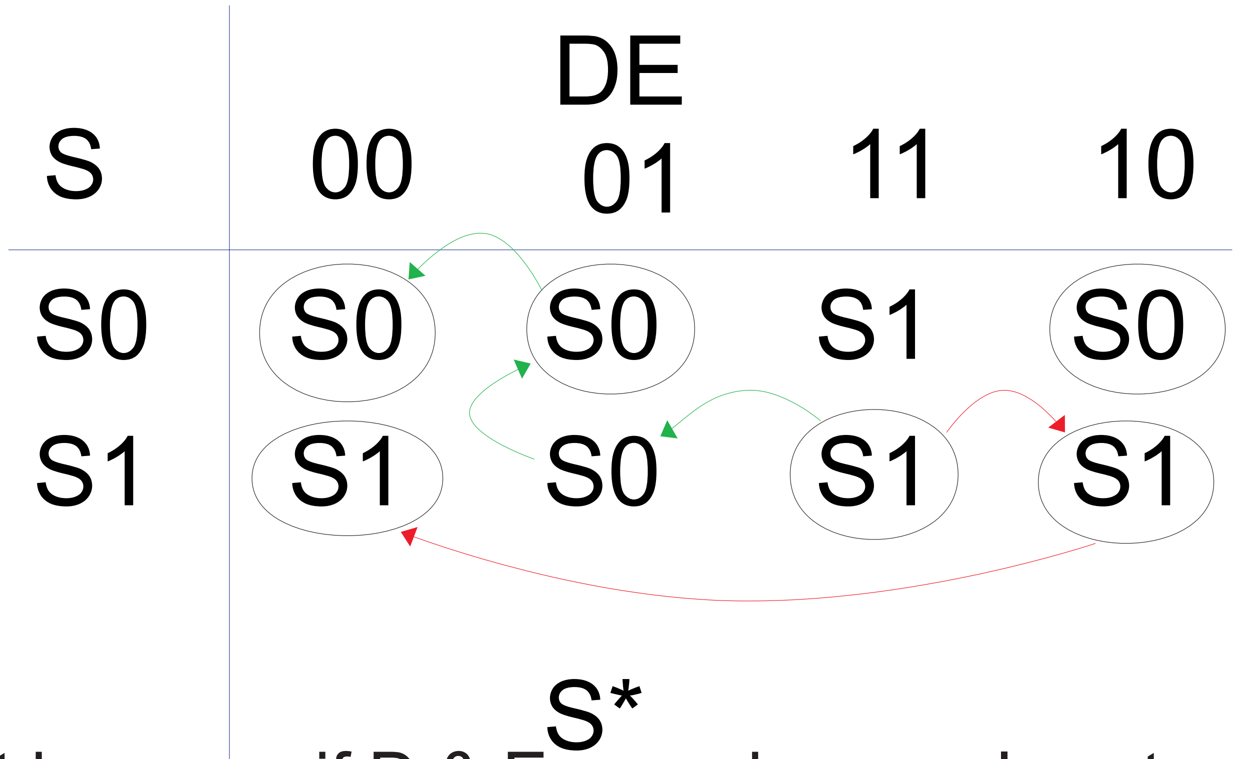
	DE			
S	00	01	11	10
S0	S0	S0	S1	S0
S1	S1	S0	S1	S1

E changes to 1

S*

Can trace what happens when inputs change from DE = 00, S= S0. D goes to 1 followed by E to 1.

State Table

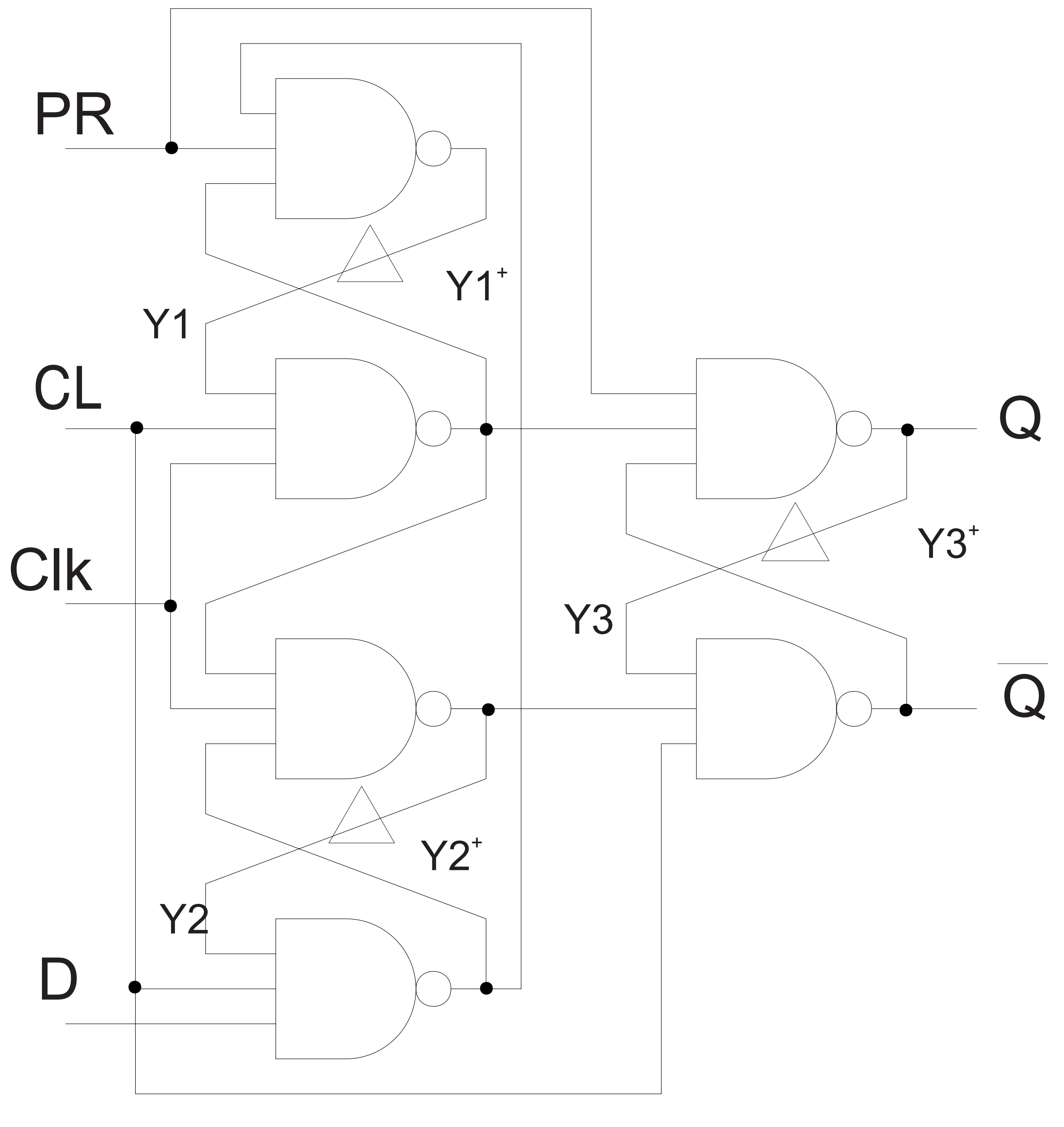


S	DE 00	01	11	10
S0	S0	S0	S1	S0
S1	S1	S0	S1	S1

S*

What happens if D & E now change almost simultaneously? Exact sequence is unpredictable. Final result is unpredictable.

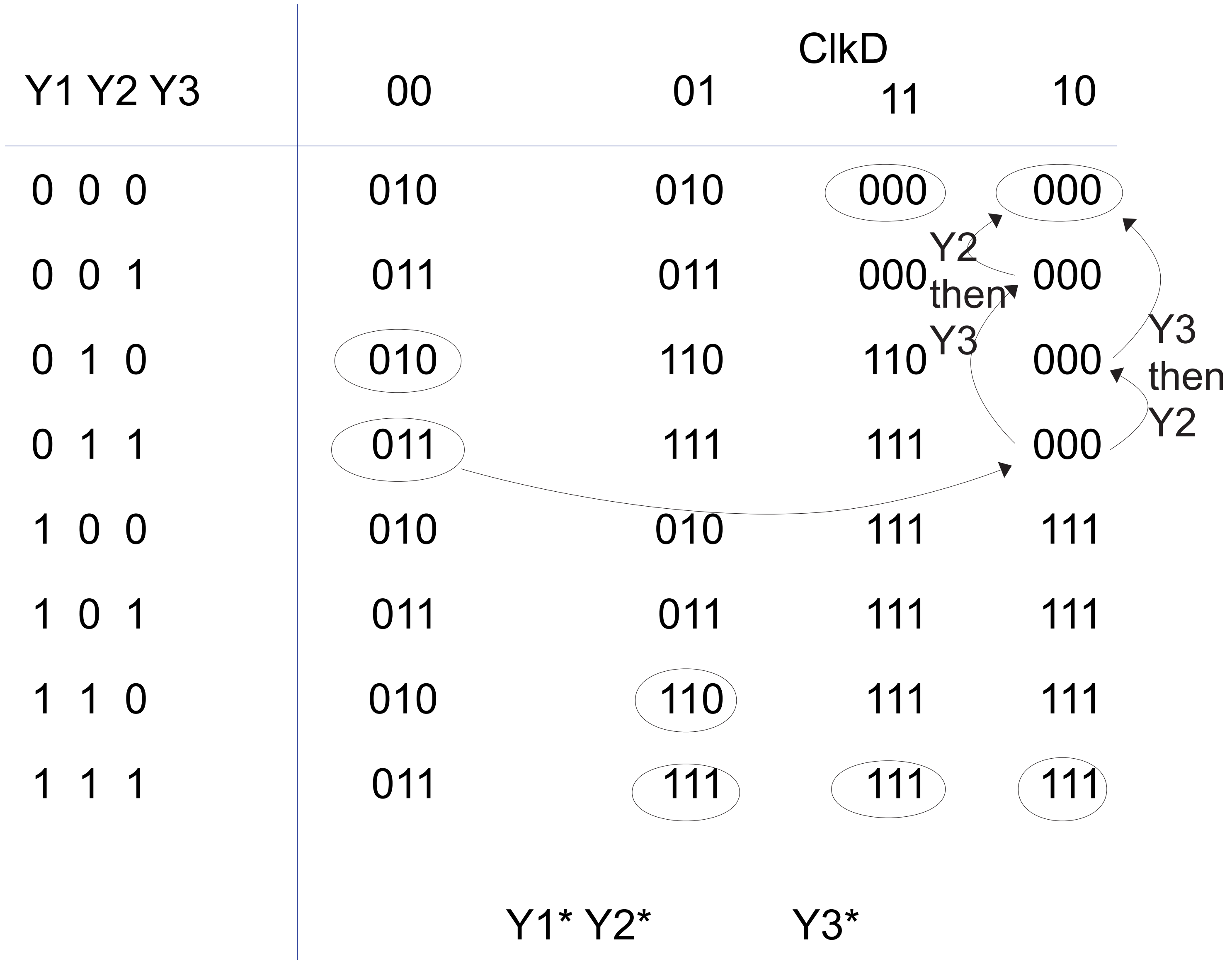
Edge-Triggered D Flip-Flop



Multiple Feedback Loops

- Assume PR and CL are at 1 - hence ignore.
- $Y1^{+} = Y2.D + Y1.Clk$
  $Y2^{+} = Y2.D + Y1.Clk + \overline{Clk}$
  $Y3^{+} = Y1.Clk + Y3.(Y2.D + Y1.Clk + \overline{Clk})$
- $Q = Y1.Clk + Y3.(Y2.D + Y1.Clk + \overline{Clk})$
  $\overline{Q} = \overline{Y3} + \overline{Y1}.\overline{Y2}.Clk + \overline{Y1}.Clk.\overline{D}$

Transition Table


Y2
then
Y3
Y3
then
Y2

Y1 Y2 Y3	ClkD 00	01	11	10
0 0 0	010	010	000	000
0 0 1	011	011	000	000
0 1 0	010	110	110	000
0 1 1	011	111	111	000
1 0 0	010	010	111	111
1 0 1	011	011	111	111
1 1 0	010	110	111	111
1 1 1	011	111	111	111

Y1* Y2* Y3*

Races

- Suppose starting state is Y1,Y2,Y3 =011, Clk,D=00 and Clk changes to 1
  Exact order of state changes depends on the order of change of internal variables
- Race - Multiple internal variables change state as a result of ONE input changing state
- If final state does not depend on order of internal state changes: non-critical race
- Suppose the transition table looked like:

Transition Table

	ClkD			
Y1 Y2 Y3	00	01	11	10
0 0 0	010	010	000	000
0 0 1	011	011	000	000
0 1 0	010	110	110	110
0 1 1	011	111	111	000
1 0 0	010	010	111	111
1 0 1	011	011	111	111
1 1 0	010	110	111	111
1 1 1	011	111	111	111

Y1* Y2* Y3*

Y2 then Y3

Critical Race
to be avoided!

State & Output Table

	ClkD			
S	00	01	11	10
S0	S2,01	S2,01	S0,01	S0,01
S1	S3,10	S3,10	S0,01	S0,01
S2	S2,01	S6,01	S6,01	S0,01
S3	S3,10	S7,10	S7,10	S0,01
S4	S2,01	S2,01	S7,11	S7,11
S5	S3,10	S3,10	S7,10	S7,10
S6	S2,01	S6,01	S7,11	S7,11
S7	S3,10	S7,10	S7,10	S7,10

S*,Q -Q

S1,S4,S5 are unstable
S2,11, S3,11, S6,10 cannot be reached

Flow Table

	ClkD			
S	00	01	11	10
S0	S2,01	**S6,01**	S0,01	S0,01
S2	S2,01	S6,01	-,-	S0,01
S3	S3,10	S7,10	-,-	S0,01
S6	S2,01	S6,01	S7,11	-,-
S7	S3,10	S7,10	S7,10	S7,10

S*,Q -Q

Eliminate unstable states, multiple hops and unreachable states

Flow Table

S	ClkD 00	01	11	10
SM	SM,01	S6,01	SM,01	SM,01
S3	S3,10	S7,10	-,-	SM,01
S6	SM,01	S6,01	S7,11	-,-
S7	S3,10	S7,10	S7,10	S7,10

S*,Q -Q

S0 and S2 are compatible and may be merged

Fundamental Mode Design

1. State Design Specifications
2. Derive a primitive flow table
3. Reduce the flow table
4. Make a race-free state assignment
5. Obtain the transition table and output map
6. Obtain hazard-free state equations

State Minimisation

- Two states can be considered equivalent if their outputs and next states are the same (for all combinations of inputs). i.e. they are indistinguishable from outside.
- States c & k are equivalent (the two rows are the same).
- Replace all instances of k by c (only applies to row j).
- Now states b & j are equivalent.

State Minimisation

Present state s	xy 00	01	11	10
a	f,0	b,0	a,0	a,0
b	f,0	c,0	a,0	a,0
c	f,0	i,0	a,0	d,1
d	e,1	f,0	h,0	g,1
e	e,1	i,0	a,0	g,1
f	f,0	f,0	a,0	a,0
g	e,1	j,0	a,0	g,1
h	f,0	b,0	h,0	a,0
i	i,0	i,0	a,0	a,0
j	f,0	k,0	a,0	a,0
k	f,0	i,0	a,0	d,1
	s*,z			

State Minimisation

Present state s	xy			
	00	01	11	10
a	f,0	b,0	a,0	a,0
b	f,0	c,0	a,0	a,0
c	f,0	i,0	a,0	d,1
d	e,1	f,0	h,0	g,1
e	e,1	i,0	a,0	g,1
f	f,0	f,0	a,0	a,0
g	e,1	b,0	a,0	g,1
h	f,0	b,0	h,0	a,0
i	i,0	i,0	a,0	a,0
		s*,z		

State Minimisation

- Note that this is an informal approach.
  We have reduced the number of states from 11 to 9.
- We can reduce the number of states to 6.
- This can be done by constructing equivalence classes.
- Can also be done using an implication table.
- Implication table can also be used for incompletely specified systems.
- Implication table contains all possible combinations of states:

Implication Table

If outputs are different, put X.
If outputs are same, list next states that must be equivalent.
(If next states are the same as present states, put a tick.)

b	b-c							
c	X	X						
d	X	X	X					
e	X	X	X	f-i a-h				
f	b-f	c-f	X	X	X			
g	X	X	X	b-f a-h	i-b	X		
h	√	b-c h-a	X	X	X	f-b a-h	X	
i	i-f b-i	f-i c-i	X	X	X	√	X	f-i a-h b-i
	a	b	c	d	e	f	g	h

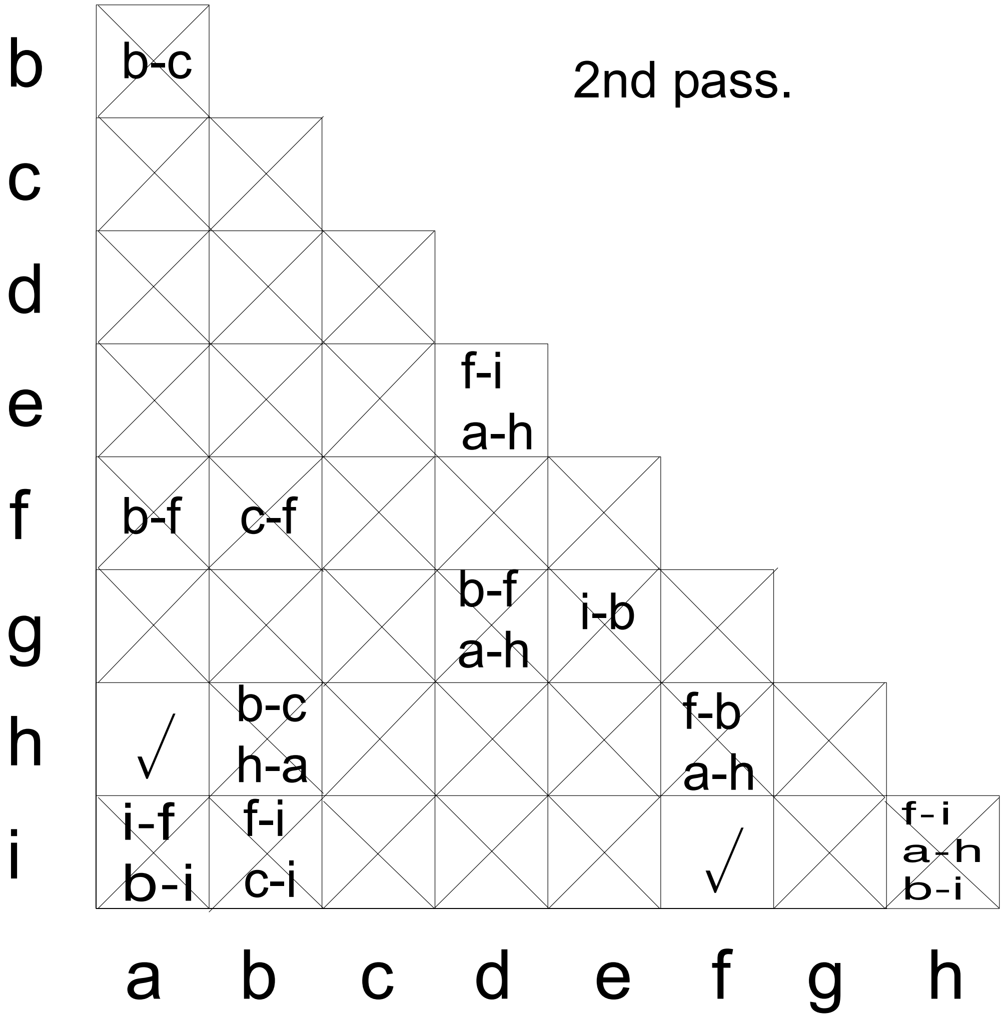
Implication Table

Now do right to left pass through the table, putting in Xs where implic-ations have already been discounted.
e.g. b & i cannot be equivalent as c-i has an X.
1st pass shown.

b	~~b-c~~							
c	X	X						
d	X	X	X					
e	X	X	X	f-i a-h				
f	~~b-f~~	~~c-f~~	X	X	X			
g	X	X	X	b-f a-h	i-b	X		
h	√	~~b-c~~ ~~h-a~~	X	X	X	f-b a-h	X	
i	~~i-f~~ ~~b-i~~	~~f-i~~ ~~c-i~~	X	X	X	√	X	f-i a-h b-i
	a	b	c	d	e	f	g	h

Implication Table

2nd pass.



	a	b	c	d	e	f	g	h
b	~~b-c~~							
c	✗	✗						
d	✗	✗	✗					
e	✗	✗	✗	f-i a-h				
f	~~b-f~~	~~c-f~~	✗	✗	✗			
g	✗	✗	✗	~~b-f a-h~~	~~i-b~~	✗		
h	√	~~b-c h-a~~	✗	✗	✗	~~f-b a-h~~	✗	
i	~~i-f b-i~~	~~f-i c-i~~	✗	✗	✗	√	✗	~~f-i a-h b-i~~

Implication Table

- There are now three squares left uncrossed.
  In this case, we can deduce that states
  (f,i), (a,h), (d,e) are equivalent.
- Thus the states needed to implement the system are (f,i), (a,h), (d,e), (b), (c), (g).
- Can have more complex cases:

Implication Table

After all passes.

b	~~b-d~~ ~~c-e~~							
c	b-f c-e	~~d-f~~						
d	✗	✗	✗					
e	b-h c-i	~~d-h~~ ~~e-i~~	f-h e-i	✗				
f	~~b-d~~ ~~c-i~~	e-i	~~d-f~~ ~~e-i~~	✗	~~d-h~~			
g	~~b-g~~ ~~a-c~~	~~d-g~~ ~~a-e~~	~~f-g~~ ~~a-e~~	✗	~~g-h~~ ~~a-i~~	~~d-g~~ ~~a-i~~		
h	~~b-d~~ ~~a-c~~	a-e	~~d-f~~ ~~a-e~~	✗	~~d-h~~ ~~a-i~~	a-i	~~d-g~~	
i	a-c	~~b-d~~ ~~a-e~~	b-f a-e	✗	b-h a-i	~~b-d~~ ~~a-i~~	~~b-g~~	~~b-d~~
	a	b	c	d	e	f	g	h

Equivalence Classes

- Start at Right of implication table. In column f, the square (f,h) is not crossed out so we list the pair (f,h). In column e, we find (e,i). In column c, we find both (c,i) and (c,e), so we can add c to (e,i). Similarly, we can add b to (f,h) and a to (c,e,i).

-

i	-
h	-
g	-
f	(f,h)
e	(e,i) (f,h)
d	(e,i) (f,h)
c	(c,e,i) (f,h)
b	(c,e,i) (b,f,h)
a	(a,c,e,i) (b,f,h)

- Equivalence Classes: (a,c,e,i) (b,f,h) (d) (g)

Incompletely Specified System

- Suppose that we do not care what the output is in certain states.
- Further, certain next state transitions may not be defined.
- This can be exploited to minimise states.
  We don't care about internal states, only about outputs!
- We now talk about *compatible* states. Two states are compatible if (and only if) their outputs, if specified, are the same after the same sequence of inputs.
- e.g. consider a system with 4 inputs and 2 outputs specified by the following state and output table. (N.B. we have not considered all combinations of inputs - only one input is true at any time.)

Incompletely specified system

s	Input K	L	M	N
a	a,-	a,-	a,-	b,00
b	c,-	d,-	e,-	-,-
c	-,-	f,-	h,-	-,-
d	h,-	-,-	h,-	-,-
e	h,-	g,-	-,-	-,-
f	-,-	-,-	f,-	a,01
g	g,-	-,-	-,-	a,10
h	h,-	h,-	h,-	a,00
		s*,YZ		

Implication Table

b	~~a-c a-d a-e~~						
c	~~a-f a-h~~	~~d-f e-h~~					
d	~~a-h~~	~~c-h e-h~~	√				
e	~~a-g a-h~~	~~c-h d-g~~	~~f-g~~	√			
f	✗	e-f	~~f-h~~	~~f-h~~	√		
g	✗	c-g	√	~~g-h~~	~~g-h~~	✗	
h	~~a-c~~	~~c-h d-h e-h~~	~~f-h~~	√	~~g-h~~	✗	✗
	a	b	c	d	e	f	g

Determination of maximum compatibles

- Same process as for finding equivalence classes. Start at right of table and list compatibles.

- g -
  f -
  e (e,f)
  d (e,f) (d,e) (d,h)
  c (e,f) (d,e) (d,h) (c,d) (c,g)
  b (e,f) (d,e) (d,h) (c,d) (c,g) (b,f) (b,g)
  a (e,f) (d,e) (d,h) (c,d) (c,g) (b,f) (b,g) (a)

- Note that, unlike a fully specified system, (e,f) are compatible, (d,e) are compatible, but (d,f) are not compatible. Thus we have two compatible pairs, instead of one class.

Maximal Compatibles

- Note that we now have 8 compatible sets. We started with 8 states, so this doesn't appear to have gained us anything! But we do not need all the sets. We are trying to find compatible states, thus do not need to use a set. We need the following sets: (a) (d,h) but all the other states occur in two sets. This is analogous to redundancy in K-maps.

- Therefore, we select sufficient sets to cover all the states e.g.
    (a) (d,h) (c,g) (e,f) (b,f)
  or (a) (d,h) (c,g) (e,f) (b,g)

- Note from the implication table, that (b,f) implies (e,f) and (b,g) implies (c,g).

State Assignment

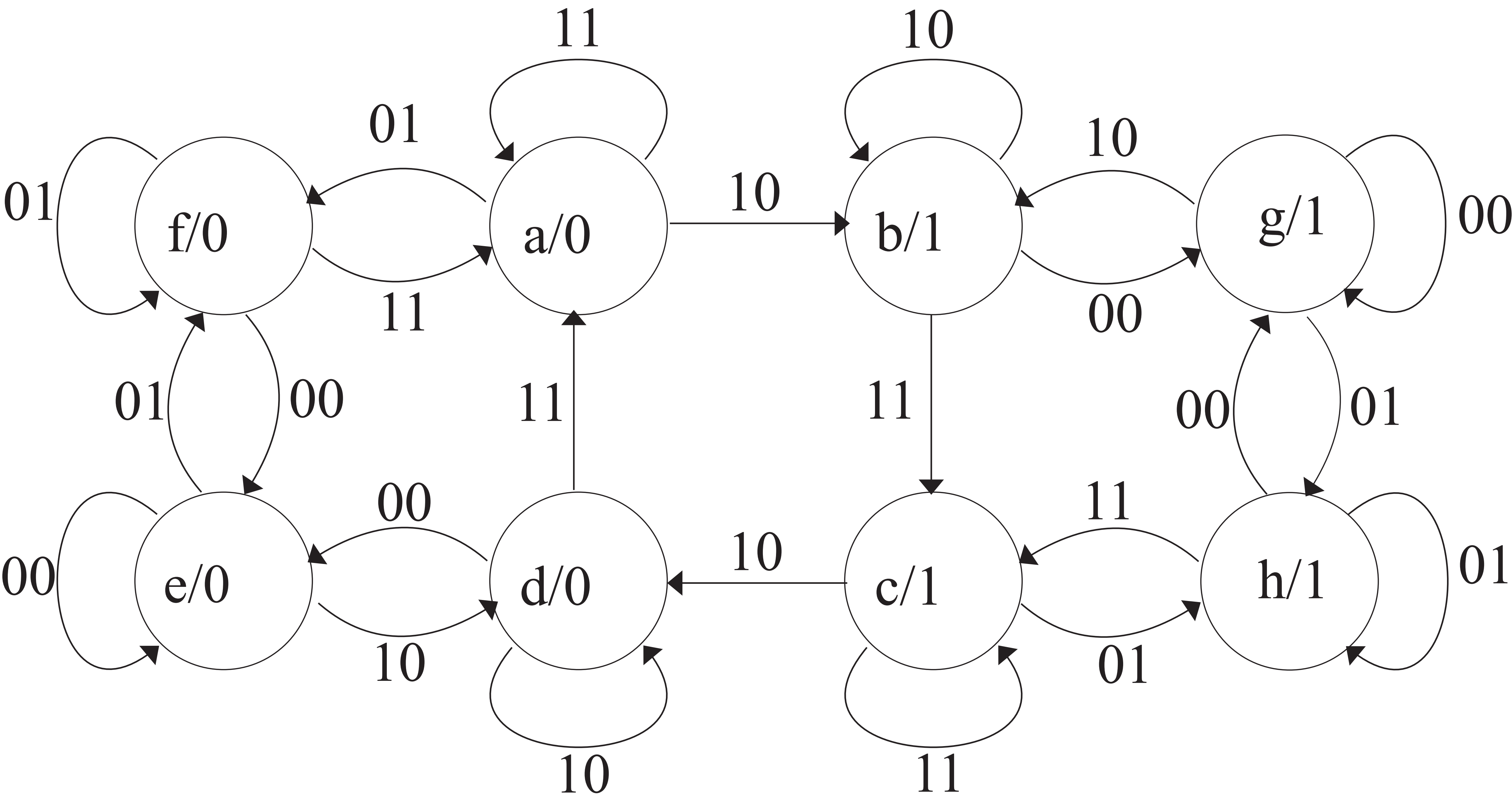
- m states and r state variables gives $2^r! / (2^r - m)!$ possible assignments.
- No way of determining which is best.
- Can assign to make state variables meaningful.
- Can assign to minimise number of bits changing between states.

Design Example

Negative edge-triggered T flip-flop

State	Inputs T	C	Output Q	Comments
a	1	1	0	Initial state
b	1	0	1	After a
c	1	1	1	Initial output
d	1	0	0	After c
e	0	0	0	After d or f
f	0	1	0	After a or e
g	0	0	1	After b or h
h	0	1	1	After g or c

Negative edge-triggered T Flip-flop



Primitive Flow Table

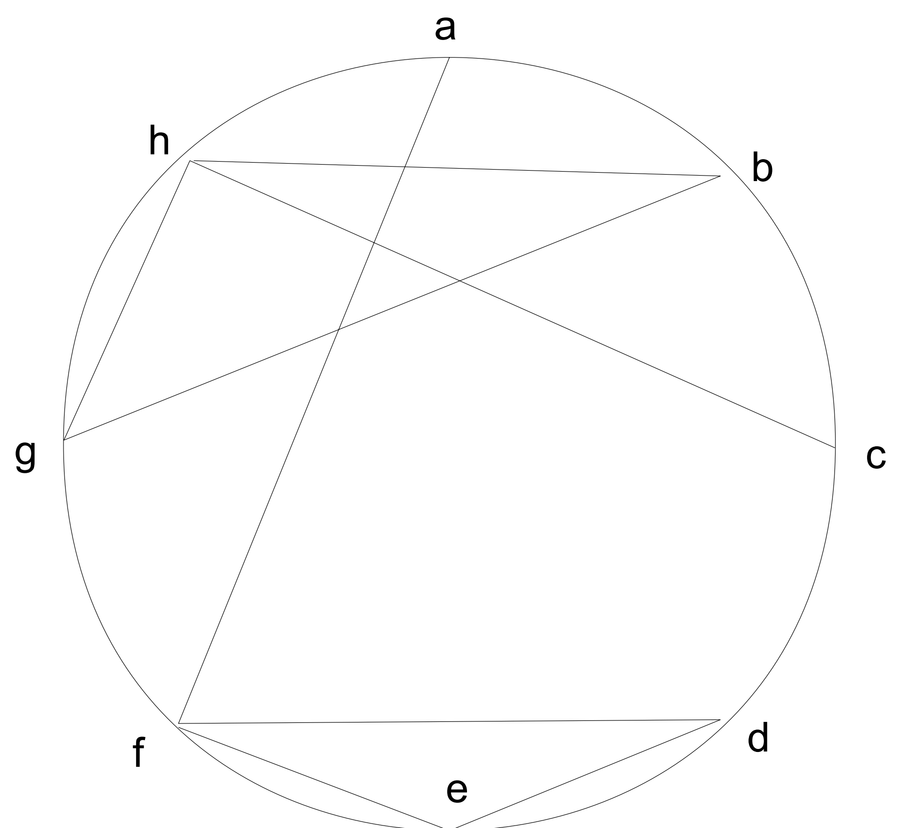
State	TC				Output
	00	01	11	10	
a	-	f	ⓐ	b	0
b	g	-	c	ⓑ	1
c	-	h	ⓒ	d	1
d	e	-	a	ⓓ	0
e	ⓔ	f	-	d	0
f	e	ⓕ	a	-	0
g	ⓖ	h	-	b	1
h	g	ⓗ	c	-	1

Implication Table

b	~~a-c~~						
c	✗	~~b-d~~					
d	~~b-d~~	✗	✗				
e	~~b-d~~	✗	✗	✓			
f	✓	✗	✗	✓	✓		
g	✗	✓	~~b-d~~	✗	✗	~~e-g f-h~~	
h	✗	✓	✓	✗	✗	✗	✓
	a	b	c	d	e	f	g

Compatible states

- From Implication table, compatible states are: (a,f) (b,g) (b,h) (c,h) (d,e) (d,f) (e,f) (g,h)
- These can be merged - one technique is:



Reduced Flow Table

State		TC				
		00	01	11	10	Output
(a,f)	A	D	Ⓐ	Ⓐ	B	0
(b,g,h)	B	Ⓑ	Ⓑ	C	Ⓑ	1
(c,h)	C	B	Ⓒ	Ⓒ	D	1
(d,e,f)	D	Ⓓ	Ⓓ	A	Ⓓ	0

State Assignment

$$\begin{tikzcd}
A=00 \arrow[r] \arrow[d, leftrightarrow] & B=01 \arrow[d, leftrightarrow] \\
D=10 & C=11 \arrow[l]
\end{tikzcd}$$

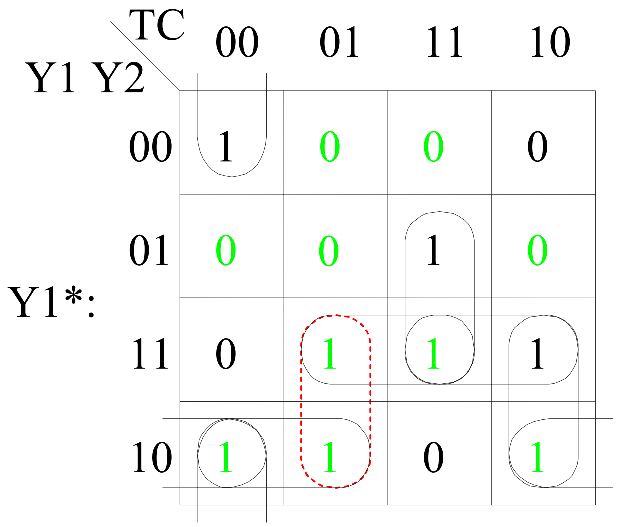
Reduced Flow Table

Y1 Y2	TC				Output
	00	01	11	10	
00	10	$\textcircled{00}$	$\textcircled{00}$	01	0
01	$\textcircled{01}$	$\textcircled{01}$	11	$\textcircled{01}$	1
11	01	$\textcircled{11}$	$\textcircled{11}$	10	1
10	$\textcircled{10}$	$\textcircled{10}$	00	$\textcircled{10}$	0

Y1* Y2*

-ve edge-triggered T flip-flop

K-maps (Entries corresponding to stable states in green):

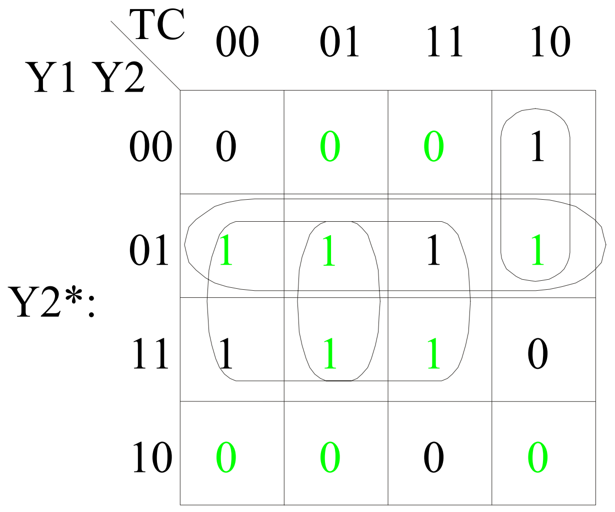


Y1*:

Y1 Y2 \ TC	00	01	11	10
00	1	0	0	0
01	0	0	1	0
11	0	1	1	1
10	1	1	0	1

$$Y1^* = \overline{T}.\overline{C}.\overline{Y2} + \overline{C}.Y1.\overline{Y2} + \overline{T}.Y1.\overline{Y2} + C.Y1.Y2 + T.C.Y2 + T.Y1.Y2 + T.\overline{C}.Y1$$

There are 8 product terms in Y1*. Note that a 1 to 0 change in T or C can cause a hazard, *even between stable states*. Therefore *all* product terms are needed, *except* $\overline{T}.C.Y1$ because that covers two stable states, with the same inputs. Although Y2 is 1 in one state and 0 in the other, the transition can't happen.



Y2*:

Y1 Y2 \ TC	00	01	11	10
00	0	0	0	1
01	1	1	1	1
11	1	1	1	0
10	0	0	0	0

$$Y2^* = \overline{T}.Y2 + C.Y2 + \overline{Y1}.Y2 + T.\overline{C}.\overline{Y1}$$

We need need $\overline{Y1}.Y2$ for Y2* because it covers a transition between stable states when T changes from 1 to 0.

The rule is: include all product terms except those that cover stable pairs in the same column (i.e. with the same inputs) and those where a transition from a stable state to an unstable state (in the same row) can only occur as a result of a 0 to 1 transition.