State Reduction

States may carry information that is found in other states. Such states are **redundant** and will be eliminated by **state reduction** procedure. This is of interest since it may reduce the number of memory devices and simplify the combinational logic implementation.

Reduction of Completely Specified State Table

Definition: Two states S_i and S_j are **equivalent**, $S_i = S_j$, if any sequence of inputs starting at S_i results in a sequence of outputs identical to the sequence of outputs obtained by starting at S_j.

States $S_1, S_2, \ldots, S_n$ are equivalent iff they they result identical output sequences for any

input sequence

State equivalence is equivalance relation

- $S_i = S_i$ for any state - reflexivity
- $S_i = S_j \iff S_j = S_i$ - symmetry
- $S_i = S_j$, $S_j = S_k \Rightarrow S_i = S_k$ - transitivity

State equivalence partition state set into equivalent classes, of which one state per class suffice and all the rest can be dropped.

There's no need to check all sequences of inputs!

Theorem: Two states S_i and S_j of a completely specified state table are equivalent iff for every input
1- the outputs produced by S_i and S_j are identical
2- next states are equivalent

Proof: Homework

State equivalence implication

Let S_i and S_j have **identical outputs** for all inputs and **identical next state** for all inputs, **except one input** where their next-states are S_m and S_n, respectively.

It follows that $S_i = S_j$ if $S_m = S_n$. We say that (S_i, S_j) **implies** (S_m, S_n). If also (S_m, S_n) implies (S_i, S_j) then $S_i = S_j$ and $S_m = S_n$

Example:

	PS	NS / output	
		$x=0$	$x=1$
*	a	c/0	b/1
*	b	d/0	b/1
**	c	a/1	d/0
**	d	b/1	c/0

trivially true

* $a=b$ if $c=d$ and $b=b$

(a,b) implies (c,d)

true by symmetry

** (c=d) if a=b and (d=c)

(c,d) implies (a,b)

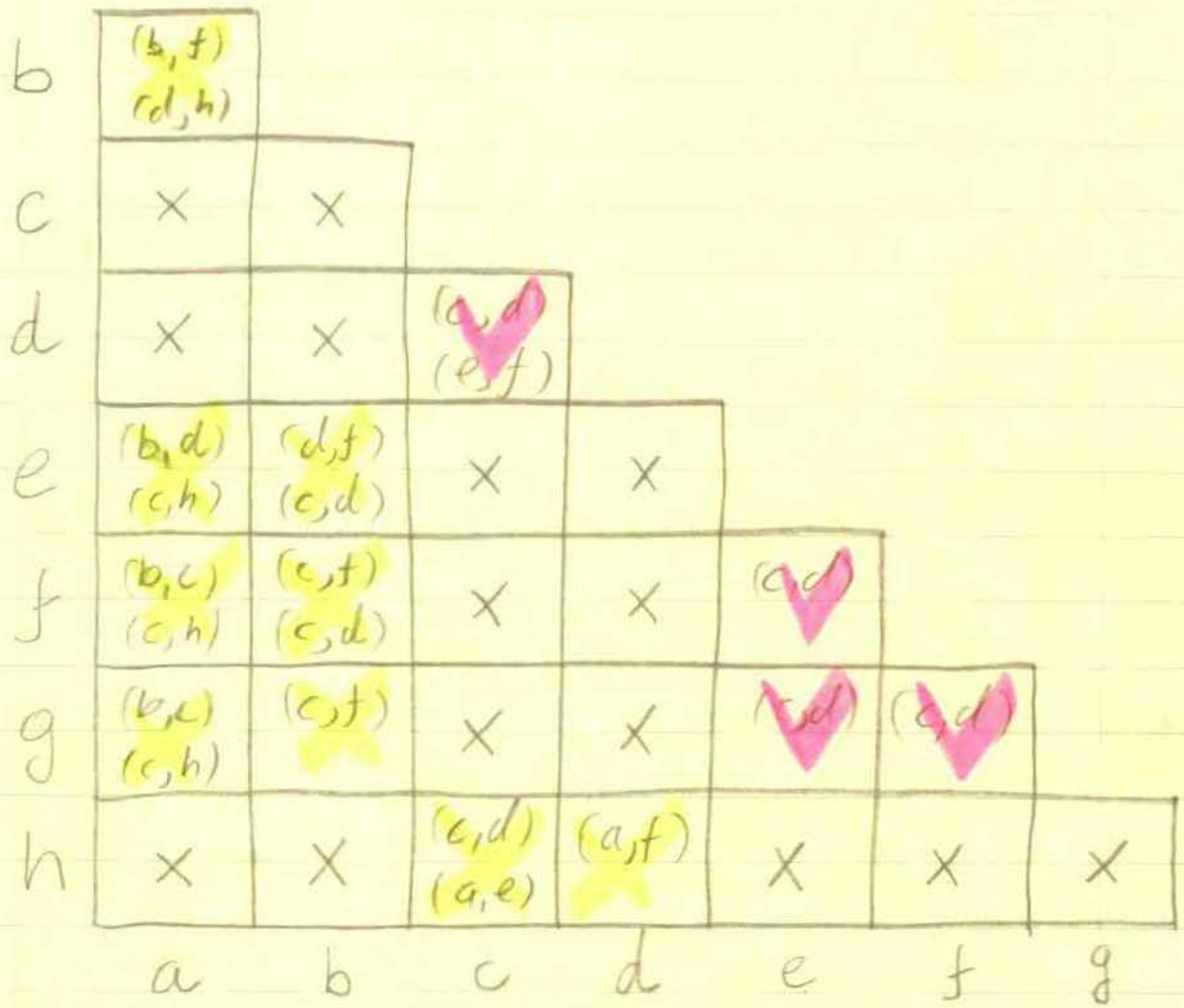
Consequently a=b and c=d. We choose representative state for every equivalent class, delete the rest rows and replace states by their representative.

PS	NS/output x=0	x=1
a	d/0	a/1
d	a/1	d/0

Implication table

PS	NS/output x=0	x=1
a	b/1	h/1
b	f/1	d/1
c	d/0	e/1
d	c/0	f/1
e	d/1	c/1
f	c/1	c/1
g	c/1	d/1
h	c/0	a/1

Implication table is a cross reference of states (lower diagonal triangle of a matrix)



(e,f) implies (c,d) and (c,d) implies (e,f), therefore e=f and c=d.

(f,g) implies (c,d) but c=d, Hence f=g.

(e,g) -"- -"- ; -"- -"- , -"- e=g.

c≠f, hence f≠b and g≠b and e≠b

But then $b \neq a$

$b \neq d$, hence $e \neq a$

$b \neq c$, hence $f \neq a$ and $g \neq a$

And so on, until all entries are either ✓ or ×.

In conclusion we obtained the following equivalent classes

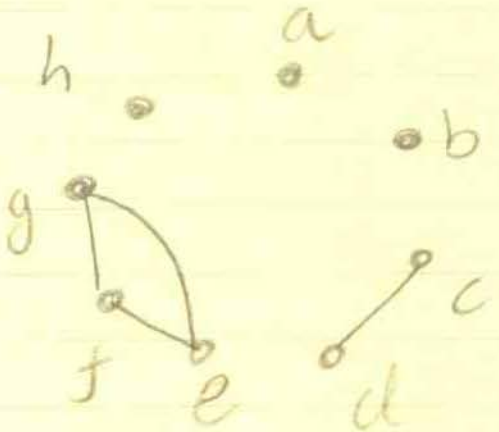
$$\underset{A}{\{a\}}, \underset{B}{\{b\}}, \underset{C}{\{c,d\}}, \underset{D}{\{e,f,g\}}, \underset{E}{\{h\}}$$

Assigning new symbols, we obtained reduced state table

PS	NS/output	
	$x=0$	$x=1$
A	B/1	E/1
B	D/1	C/1
C	C/0	D/1
D	C/1	C/1
E	C/0	A/1

Implication graph

Vertices are states, equivalence are edges
Equivalent classes are cliques.



Reduction of Flow Tables (Primitive)

NS/Output ($z_1 z_2$)

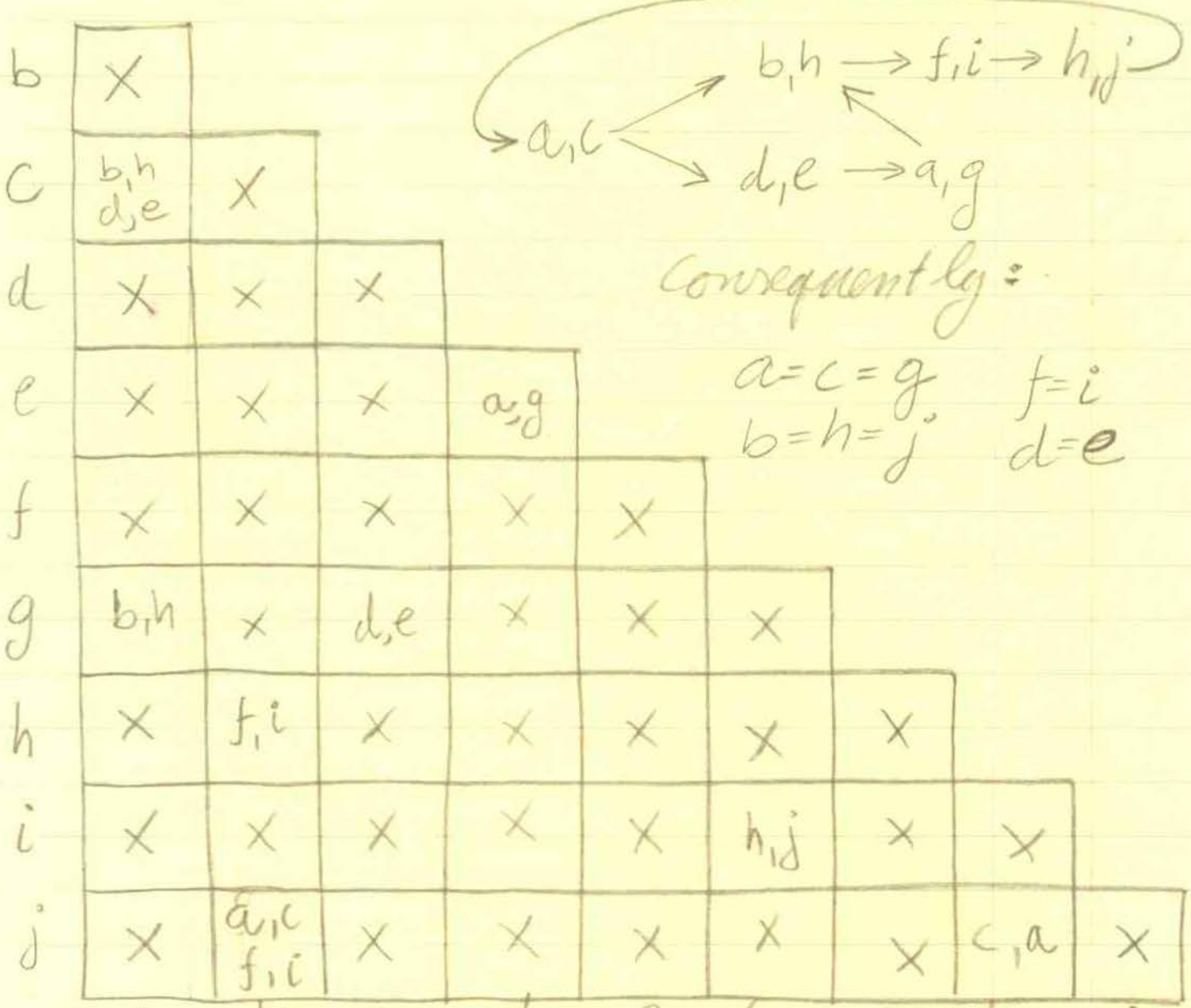
PS	Input ($x_1 x_2$) 00	01	11	10
a	(a/00)	b/–	–/–	e/–
b	c/–	(b/10)	f/–	–/–
c	(c/00)	h/–	–/–	d/–
d	a/–	–/–	f/–	(d/11)
e	g/–	–/–	f/–	(e/11)
f	–/–	h/–	(f/01)	e/–
g	(g/00)	h/–	–/–	e/–
h	c/–	(h/10)	i/–	–/–
i	–/–	j/–	(i/01)	e/–
j	a/–	(j/–)	i/–	–/–

fundamental mode: transitions are defined only for change of one input variable

Two steps:

1- Redundant state removal as for completely specified tables

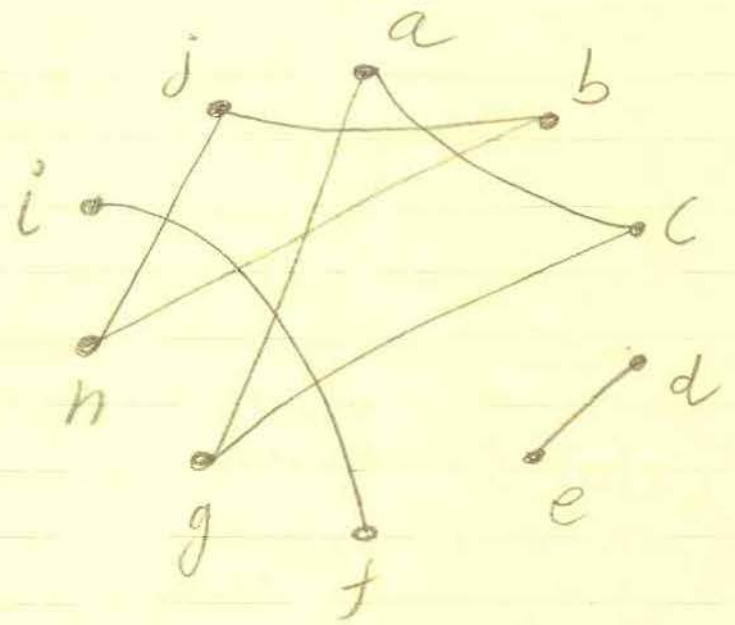
2- Row merging where few rows are captured in a single row (this is not redundancy removal



Consequently:

$a = c = g$ $\quad f = i$

$b = h = j$ $\quad d = e$



$\{a, c, g\}, \{b, h, i\}, \{d, e\}, \{f, i\}$
$\quad A \qquad\quad B \qquad\quad C \qquad\quad D$

	NS/output ($z_1 z_2$)			
	Input ($x_1 x_2$)			
PS	00	01	11	10
A	(A/00)	B/–	–/–	C/–
B	A/–	(B/10)	D/–	–/–
C	A/–	–/–	D/–	(C/11)
D	–/–	B/–	(D/01)	C/–

Merger Process

PS	00	01	11	10
A	(A/00)	B/–	–/–	C/–
B	A/–	(B/10)	D/–	–/–

⇓

A,B	(A/00)	(B/10)	D/–	C/–

merging C and D yields:

C,D	A/–	B/–	(D/01)	(C/11)

But then A,B can be merged with CD, yielding

A,B,C,D	(A/00)	(B/10)	(D/01)	(C/11)

But the above has one state, thus a combinational logic!

Merging is valid only in fundamental-mode (only one input change at a time)

Definition: Any two rows r_i and r_j in a flow table can be **merged** into a single row (r_i, r_j) iff there are no conflict in any column.

Merger is **not** transitive

Example:

PS	00	01	11	10
a	(a)	b	-	c
b	a	(b)	d	-
c	a	-	e	(c)

(a,b) mergable, (a,c) mergeable but (c,b) does not!

Definition: Rows $r_1, \ldots, r_j$ can be merged into a single row $(r_1, \ldots, r_j)$ if any pair can be merged

Example:

PS	NS Input ($x_1 x_2$) 00	01	11	10	output (z)
a	(a)	b	–	f	0
b	a	(b)	d	–	0
c	(c)	b	–	–	0
d	–	b	(d)	e	1
e	c	–	d	(e)	1
f	a	–	d	(f)	0

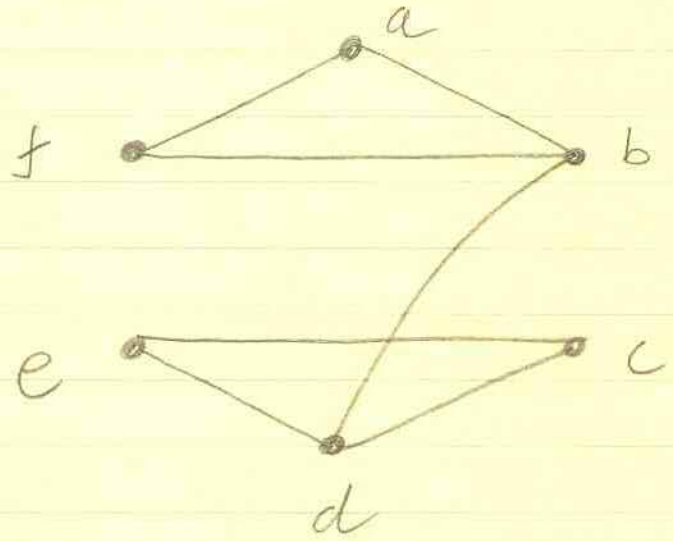
Assume Moore model. Then output is associated with the entire NS of a row. Hence it must not conflict for rows subjected to merging. Hence merging results in

(a, b, f), (c), (de)
A B C

2 memory elements

PS	00	01	11	10	output (z)
A	(A)	(A)	C	(A)	0
B	(B)	A	–	–	0
C	B	A	(C)	(C)	1

In Mealy models outputs are associated to stable states only. Merger therefore may yield smaller table.



Minimal cardinality cover is (a, b, f), (c, d, e)
A B

PS	00	01	10	11
A	(A/0)	(A/0)	B/–	(A/0)
B	(B/0)	A/–	(B/1)	(B/1)

Notice the different outputs in a row. 1 memory element

State Assignment

Assign unique binary code to states.

n states require at least N state variables $N \geq \lceil \log_2 n \rceil$. For synchronous machines this is sufficient. Asynchronous however are more delicate due to races, and more memory devices may be required, depending on state transitions.

There are $P = 2^N! / (2^N - n)!$

$n=4, N=2 \Rightarrow P=24$, $n=5, N=3 \Rightarrow P=6720$

Two different assignments may result significant hardware difference.

Two assignments are equivalent if they can be obtained from each other by complement or state variable swapping

$(2^N)!/2^N$

There are therefore $R=(2^N-1)!/[(2^N-n)!N!]$ different assignments

N rows, n columns

$n=4, N=2 \Rightarrow R=3$, $n=5, N=3 \Rightarrow R=140$

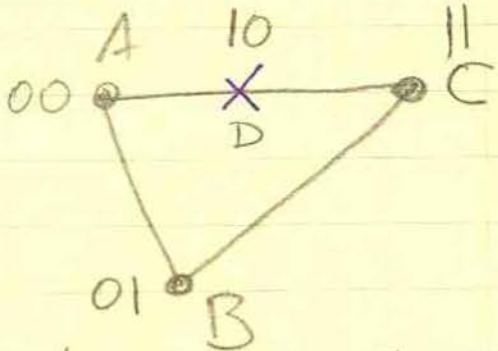
No efficient and simple method for optimal assignment. We'll focus on **robustness** (race-free), although assignments resulting **non-critical races** are **valid**.

Avoid state transitions that result change in more than one state variable

Example

PS	x_1x_2 00	01	11	10
A	(A)	B	C	(A)
B	A	(B)	(B)	C
C	B	(C)	(C)	(C)

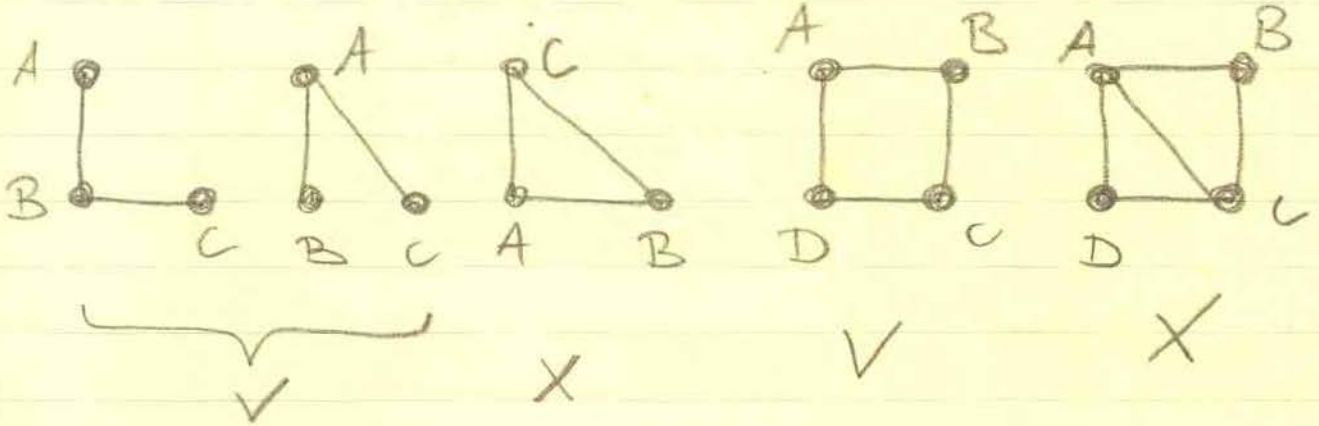
Flow-table



State-assignment graph

State assignment graph: vertices are states, edges are transitions.

adjacent vertices must be assigned with 1-bit different codes. Not always possible.

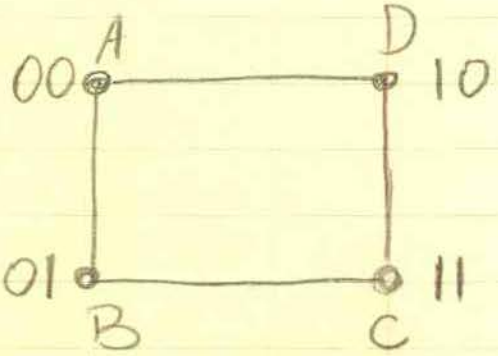


Resolution by cycles: a sequence of unstable states that terminate in a stable state. (or new states)

We can use unspecified next states without increasing # state variables (preferred) or using more state variable

PS	x_1x_2 00	01	11 (was C)	10
A	(A)	B	D	(A)
B	A	(B)	(B)	C
C	B	(C)	(C)	(C)
→ D (new)	—	—	C	—

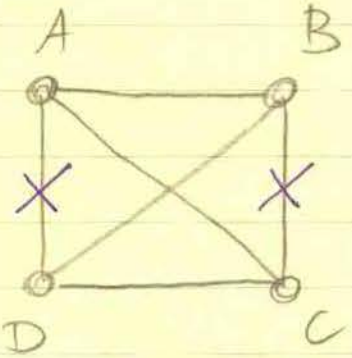
watch up not to assign D



We broke the transition (edge in graph) between A and C by introducing new unstable state D

Example: use unspecified next states

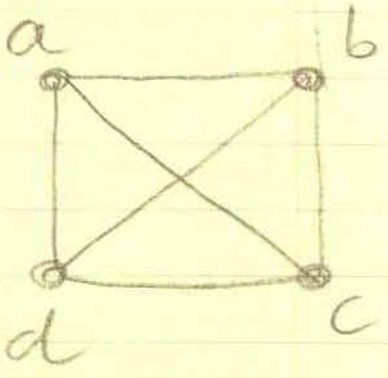
	00	01	11	10
A	(A)	B	C—	C
B	(B)	(B)	AC	D
C	A—	D	(C)	(C)
D	CA	(D)	(D)	(D)



Break A–D edge (due to 00 input) by replacing unstable A by C and setting unspecified to A. Similar with B–C

Adding state variables

	00	01	11	10
a	b	(a)	(a)	b
b	(b)	d	c	(b)
c	(c)	a	(c)	d
d	c	(d)	a	(d)

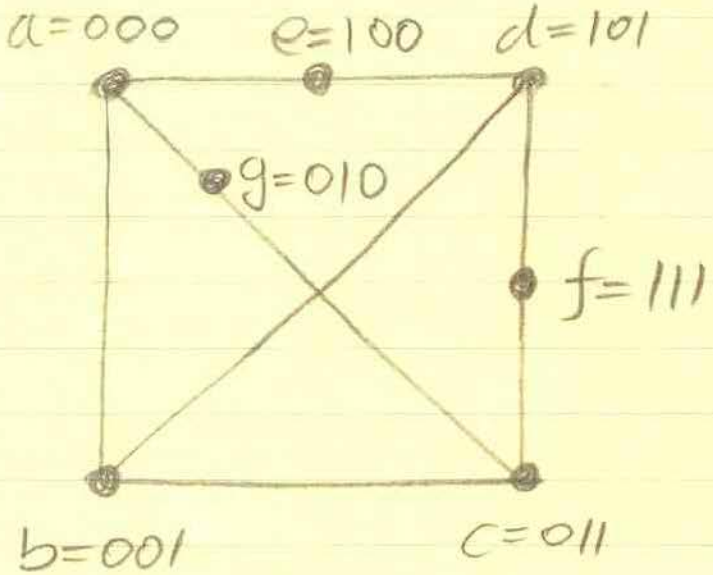


addition of one state variable is a must to assign adjacent codes to adjacent states We'll have then 3 state variables

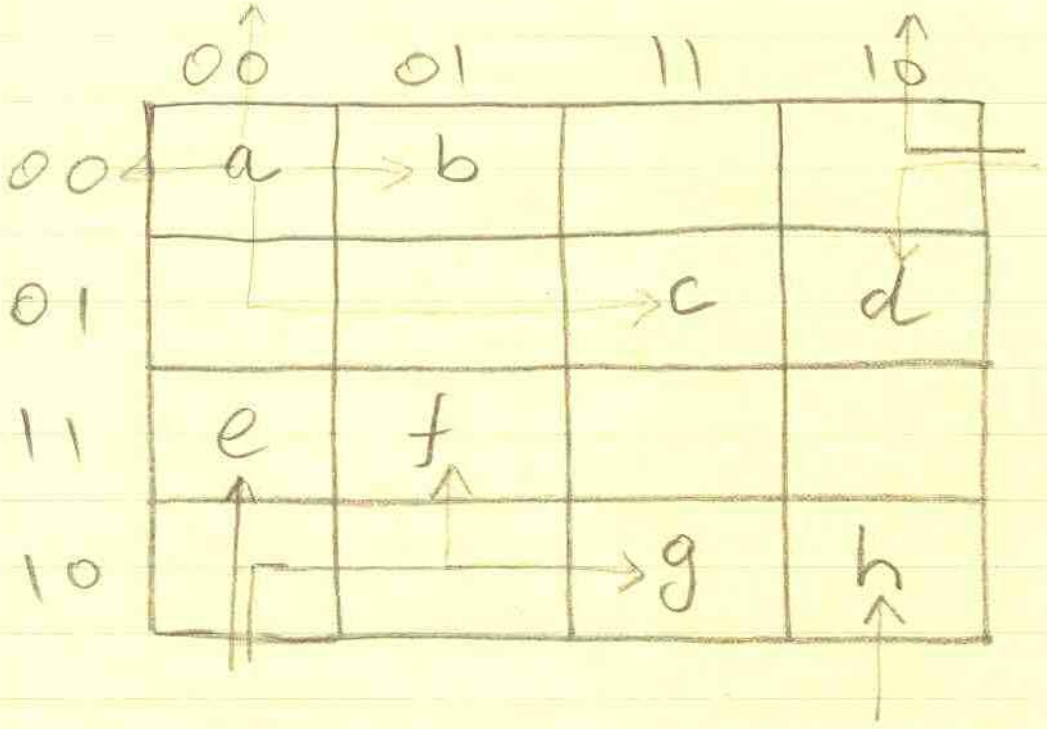
y_1 \ y_2y_3	00	01	11	10
0	a	b	c	(g)
1	(e)	d	(f)	

The idea is to create "path" between any states through new unstable state. This creates three new states e, f and g

	00	01	11	10
a	b	ⓐ	ⓐ	b
b	ⓑ	d	c	ⓑ
c	ⓒ	~~g~~	ⓒ	~~d f~~
d	~~d f~~	ⓓ	~~d e~~	ⓓ
g	—	a	—	—
e	—	—	a	—
f	c	—	—	d



Any 8-row flow table that needs 4 state variables can use the following template

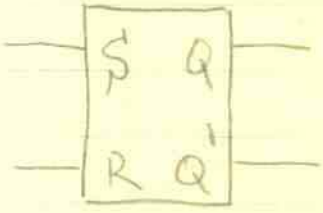


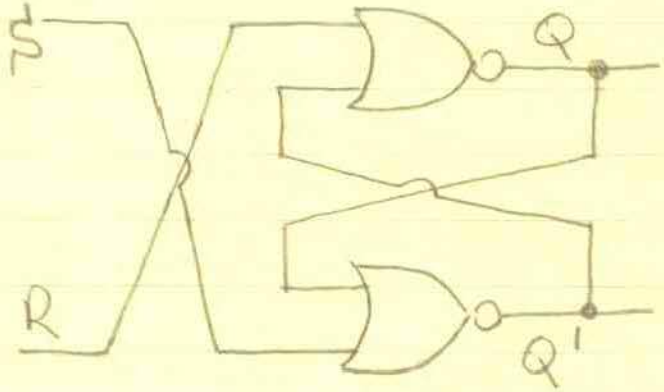
Memory Devices

A latch is a bistable device. It has output Q and its complement Q' and can be in either of Q=1, Q'=0 called set state, or Q=0, Q'=1 called reset state. A latch changes value when its inputs change value. Output is changed after some internal propagation delay, a property called transparency, common to all latche

Flip-Flop state changes are controlled by clock input, and can change state only in response to a transition of clock signal.

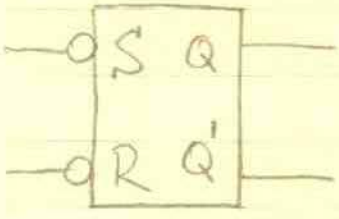
SR (Set-Reset) Latch

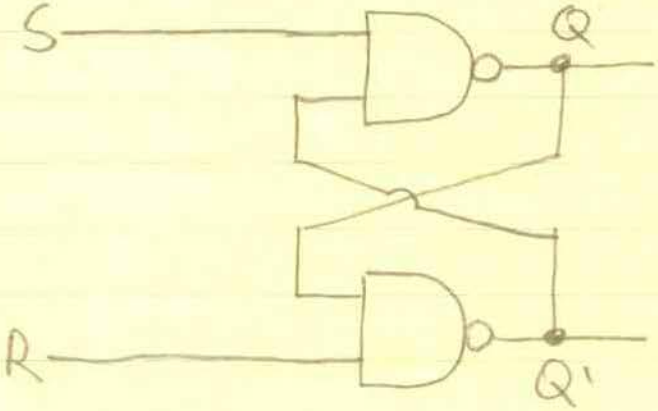




S=0 R=0 in normal operation one input changes to 1.

Active-High





S=1 R=1 in normal operation one input change to 0.

Active-Low

↑

called S'R' ($\bar{S}$-$\bar{R}$) latch

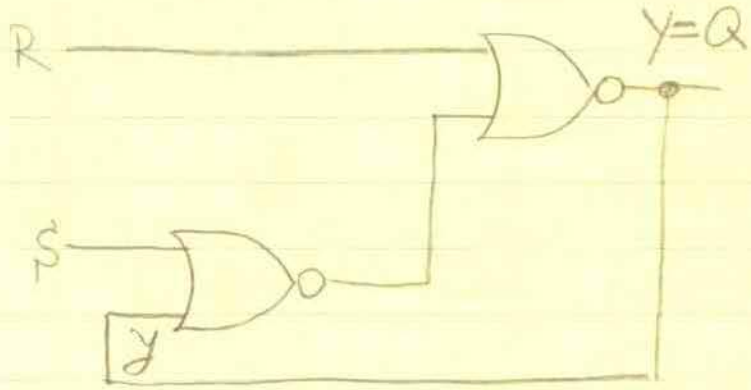
S	R	Q(PS)	Q^+(NS)	
0	0	0	0	not change
0	0	1	1	not change
0	1	0	0	Reset
0	1	1	0	Reset
1	0	0	1	Set
1	0	1	1	Set
1	1	0		not determinate
1	1	1		not determinate

characteristic table

$\Rightarrow$ SR=0

S	R	Q^+ (NS)
0	0	Q
0	1	0
1	0	1
1	1	nd determinate

Equivalent characteristic table



SR=0

$Q^+ = R'Q + R'S =$

$= R'Q + S$

Active High

Derive excitation table: What should be S and R in order to obtain Q and Q^+. It is derived from the characteristic table.

Q	Q^+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

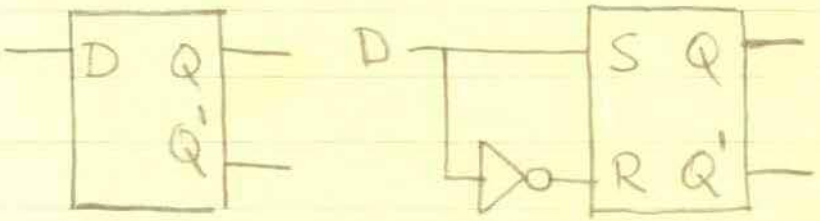
Excitation table

Excitation table is useful for deriving the logic for setting S and R

D Latch one input

Obtained from SR latch by enforcing

$S = R'$



D	Q^+
0	0
1	1

equivalent characteristic table

$Q^+ = D$

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

Excitation table

Gated Latches

Using enable input to control operation. Latch can change state only at E=1

S'R' latch