COE 561 Digital System Design & Synthesis Sequential Logic Synthesis

Dr. Aiman H. El-Maleh

Computer Engineering Department

King Fahd University of Petroleum & Minerals

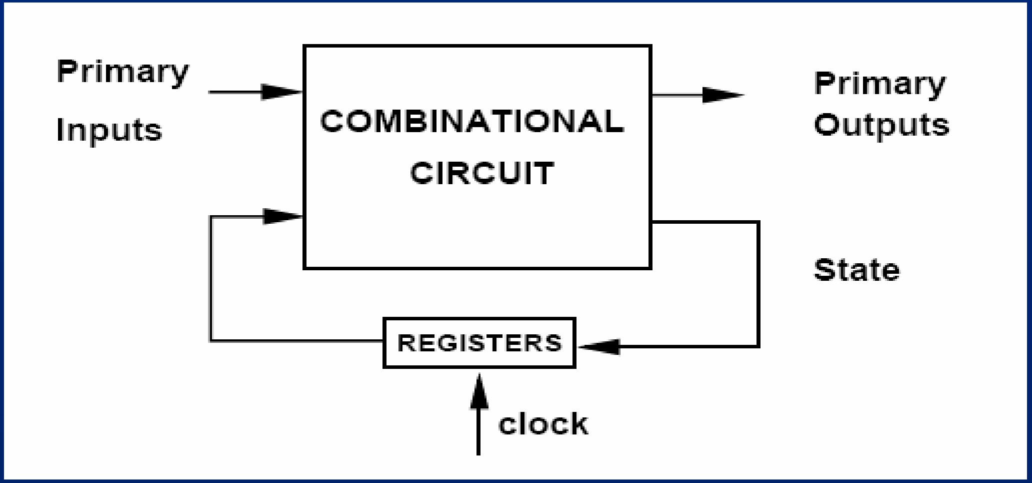
[Adapted from slides of Prof. G. De Micheli: Synthesis & Optimization of Digital Circuits]

Outline

- **Modeling synchronous circuits**
  - State-based models.
  - Structural models.
- **State-based optimization methods**
  - State minimization.
  - State encoding
    - State encoding for two-level logic
      - Input encoding
      - Output encoding
    - State encoding for multiple-level logic
- **Structural-based optimization methods**
  - Retiming

Synchronous Logic Circuits

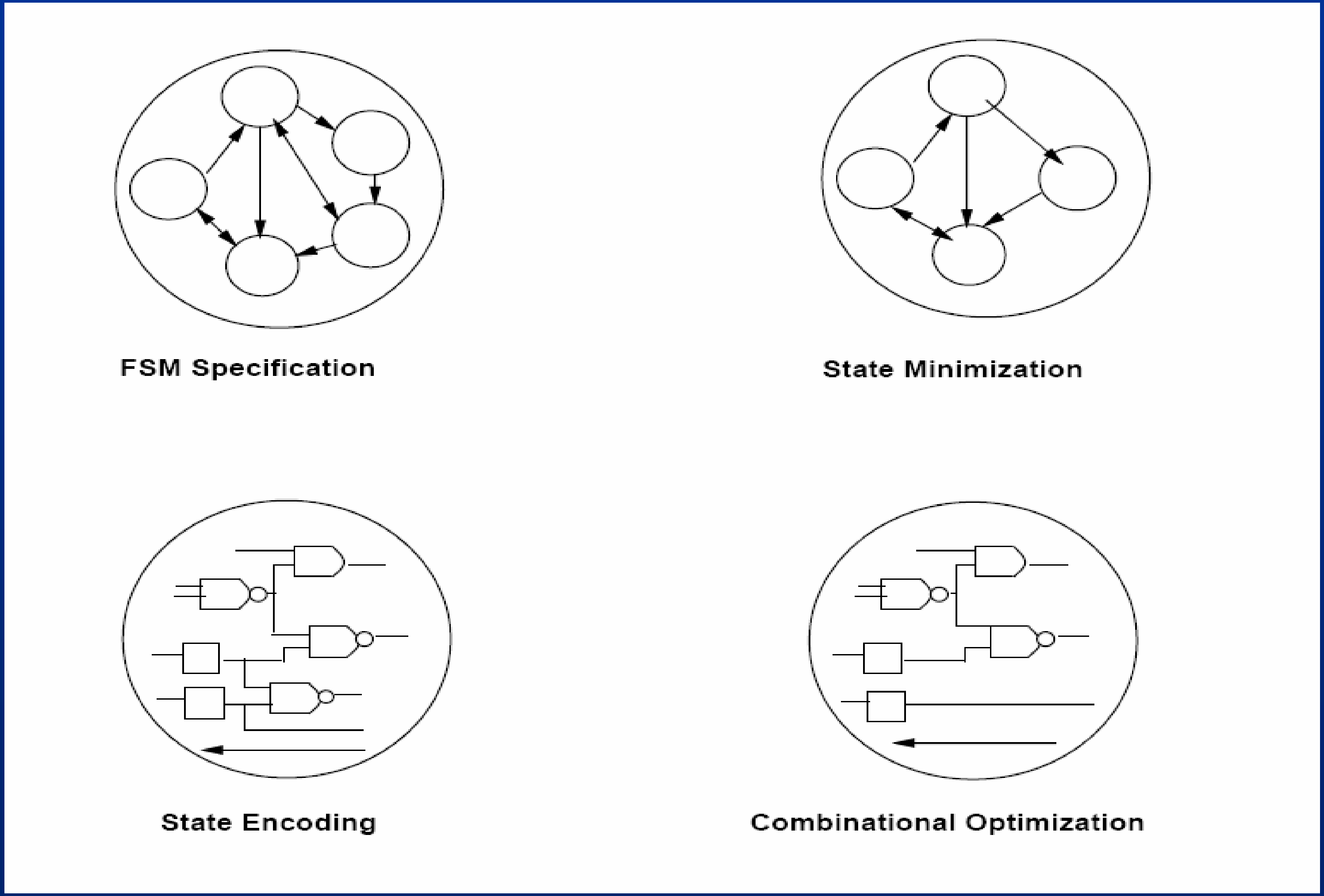
- **Interconnection of**
  - Combinational logic gates.
  - Synchronous delay elements
    - E-T or M-S registers.
- **Assumptions**
  - No direct combinational feedback.
  - Single-phase clocking.



Modeling Synchronous Circuits

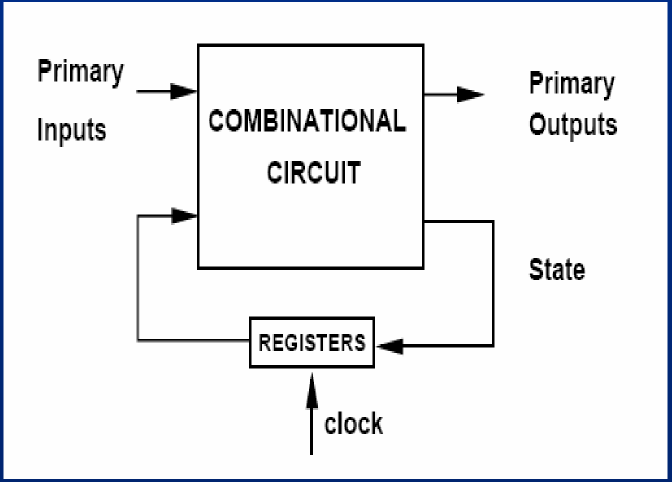
- **State-based model**
  - Model circuits as finite-state machines.
  - Represented by state tables/diagrams.
  - Lacks a direct relation between state manipulation and corresponding area and delay variations.
  - Apply exact/heuristic algorithms for
    - State minimization.
    - State encoding.
- **Structural models**
  - Represent circuit by synchronous logic network.
  - Apply
    - Retiming.
    - Logic transformations.

State-Based Optimization



Formal Finite-State Machine Model

- Defined by the quintuple $(X, Y, S, \delta, \lambda)$.
- A set of primary inputs patterns X.
- A set of primary outputs patterns Y.
- A set of states S.
- A state transition function
  - $\delta : X \times S \rightarrow S$.
- An output function
  - $\lambda: X \times S \rightarrow Y$ for Mealy models
  - $\lambda : S \rightarrow Y$ for Moore models.



State Minimization

- **Aims at reducing the number of machine states**
  - reduces the size of transition table.
- **State reduction may reduce**
  - the number of storage elements.
  - the combinational logic due to reduction in transitions
- **Completely specified finite-state machines**
  - No don't care conditions.
  - Easy to solve.
- **Incompletely specified finite-state machines**
  - Unspecified transitions and/or outputs.
  - Intractable problem.

State Minimization for Completely-Specified FSMs

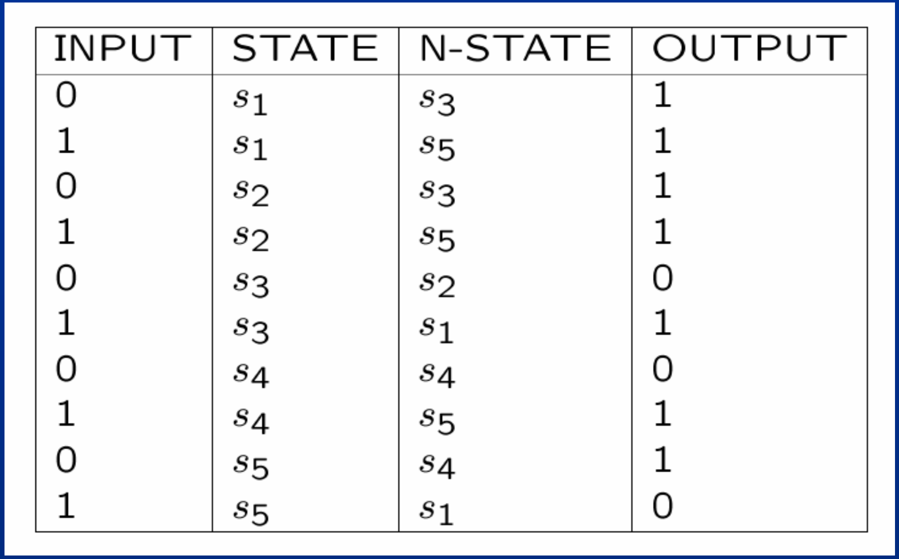
- **Equivalent states**
  - Given any input sequence the corresponding output sequences match.
- **Theorem: Two states are equivalent iff**
  - they lead to identical outputs and
  - their next-states are equivalent.
- **Equivalence is *transitive***
  - Partition states into *equivalence classes*.
  - Minimum finite-state machine is unique.

Algorithm

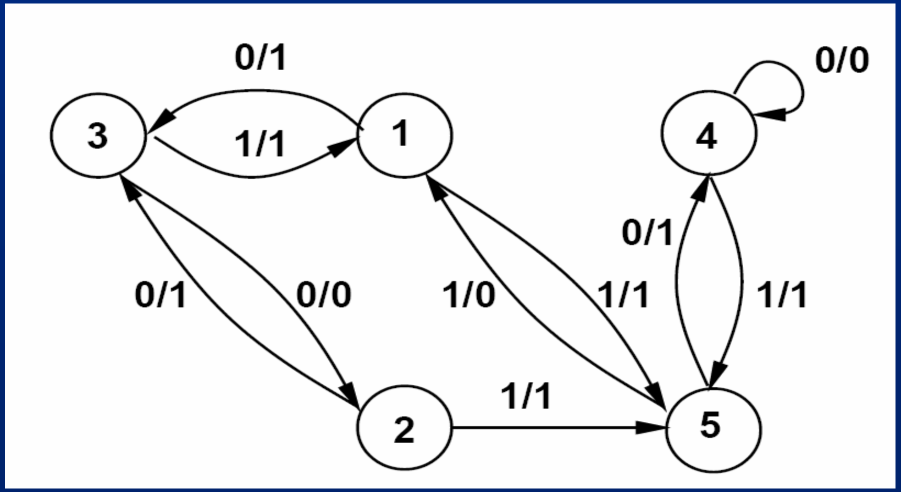
- **Stepwise partition refinement.**
- **Initially**
  - Π_1 = States belong to the same block when outputs are the same for any input.
- **Refine partition blocks: While further splitting is possible**
  - Π_{k+1} = States belong to the same block if they were previously in the same block and their next-states are in the same block of Π_k for any input.
- **At convergence**
  - Blocks identify equivalent states.

Example …

- Π_1 = {(s1, s2), (s3, s4), (s5)}.
- Π_2 = {(s1, s2), (s3), (s4), (s5)}.
- Π_2 = is a partition into equivalence classes
  - States (s1, s2) are equivalent.

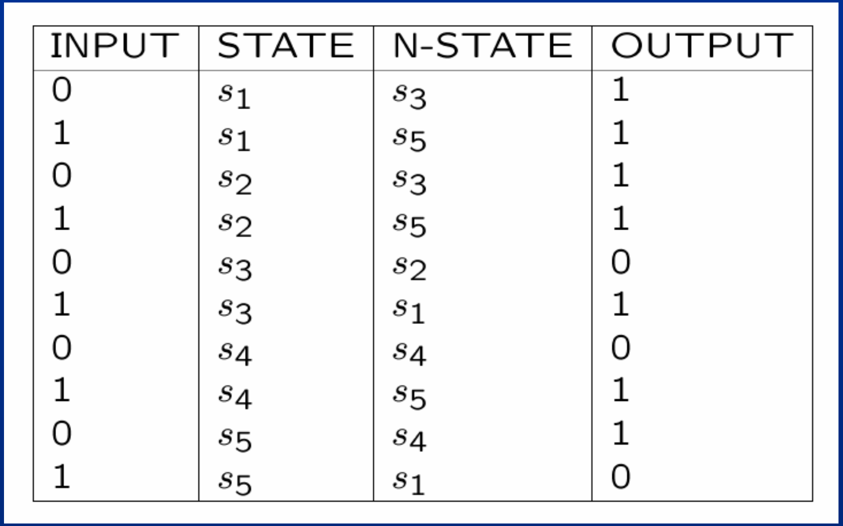


INPUT	STATE	N-STATE	OUTPUT
0	s_1	s_3	1
1	s_1	s_5	1
0	s_2	s_3	1
1	s_2	s_5	1
0	s_3	s_2	0
1	s_3	s_1	1
0	s_4	s_4	0
1	s_4	s_5	1
0	s_5	s_4	1
1	s_5	s_1	0



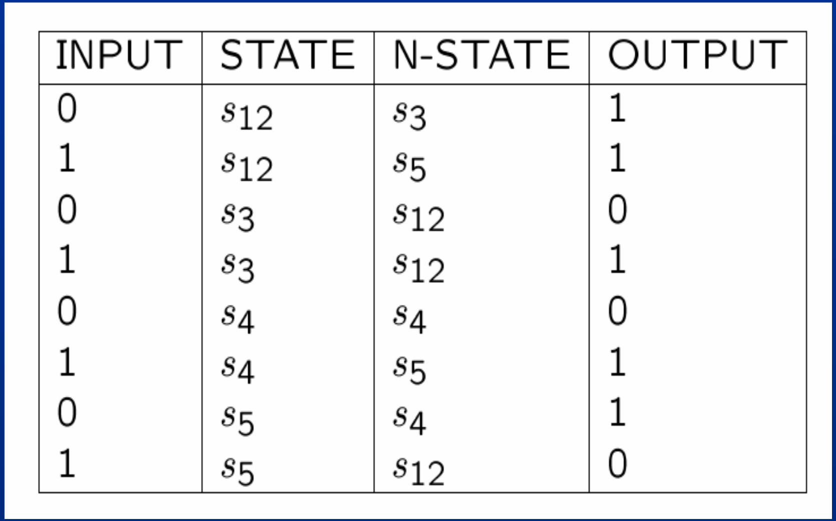
… Example …

Original FSM

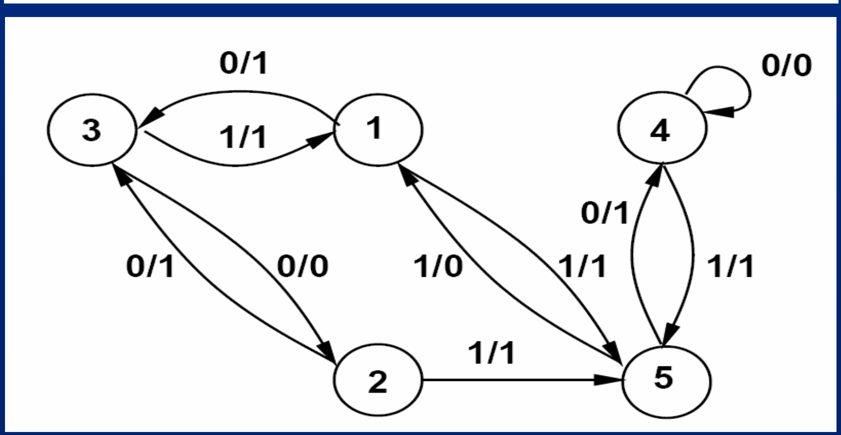


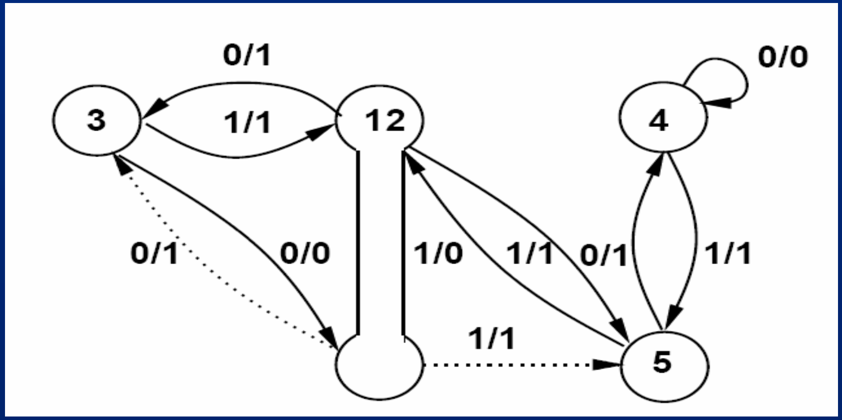
INPUT	STATE	N-STATE	OUTPUT
0	s_1	s_3	1
1	s_1	s_5	1
0	s_2	s_3	1
1	s_2	s_5	1
0	s_3	s_2	0
1	s_3	s_1	1
0	s_4	s_4	0
1	s_4	s_5	1
0	s_5	s_4	1
1	s_5	s_1	0

Minimal FSM



INPUT	STATE	N-STATE	OUTPUT
0	s_{12}	s_3	1
1	s_{12}	s_5	1
0	s_3	s_{12}	0
1	s_3	s_{12}	1
0	s_4	s_4	0
1	s_4	s_5	1
0	s_5	s_4	1
1	s_5	s_{12}	0





… Example

Original FSM

```
{OUT_0} = IN_0 LatchOut_v1' + IN_0 LatchOut_v3' + IN_0' LatchOut_v2'
v4.0 = IN_0 LatchOut_v1' + LatchOut_v1' LatchOut_v2'
v4.1 = IN_0' LatchOut_v2 LatchOut_v3 + IN_0' LatchOut_v2'
v4.2 = IN_0 LatchOut_v1' + IN_0' LatchOut_v1 + IN_0' LatchOut_v2 LatchOut_v3
sis> print_stats
 pi= 1   po= 1   nodes=  4       latches= 3
lits(sop)=  22  #states(STG)=   5
```

Minimal FSM

```
{OUT_0} = IN_0 LatchOut_v1' + IN_0 LatchOut_v2 + IN_0' LatchOut_v2'
v3.0 = IN_0 LatchOut_v1' + LatchOut_v1' LatchOut_v2‘
v3.1 = IN_0' LatchOut_v1' + IN_0' LatchOut_v2'
sis> print_stats
pi= 1   po= 1   nodes=  3       latches= 2
lits(sop)=  14  #states(STG)=   4
```

Computational Complexity

- **Polynomially-bound algorithm.**
- **There can be at most |S| partition refinements.**
- **Each refinement requires considering each state**
  - Complexity $O(|S|^2)$.
- **Actual time may depend upon**
  - Data-structures.
  - Implementation details.

State Minimization for Incompletely-Specified FSMs …

- **Applicable input sequences**
  - All transitions are specified.
  - Does not lead to any unspecified transition.
- **Compatible states**
  - Given any applicable input sequence the corresponding output sequences match.
- **Theorem: Two states are compatible iff**
  - they lead to identical outputs (when both are specified),
  - their next-states are compatible (when both are specified).
- Compatibility is not an equivalency relation (not transitive).

An Interesting Example

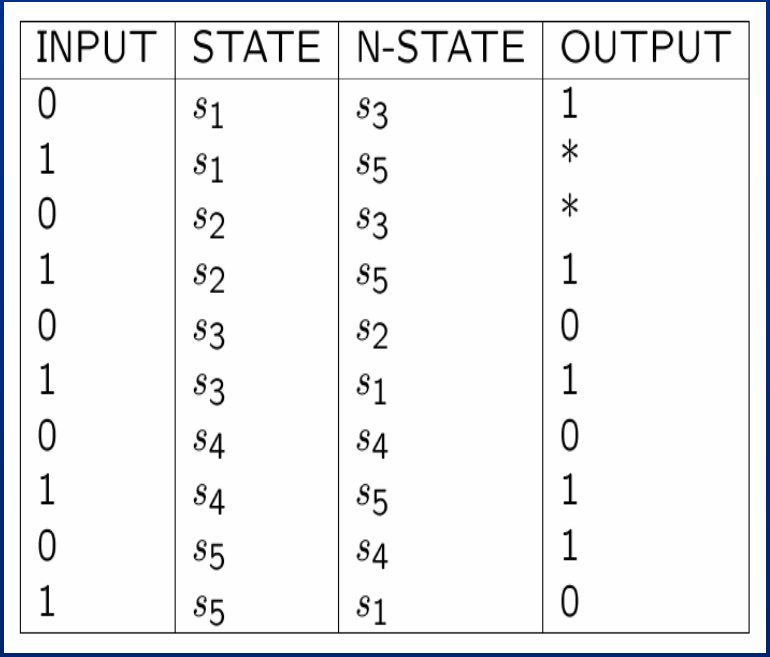
- Moore machine with 3-states.
- Replace don't care output of s1 by 0 → Can't be minimized.
- Replace don't care output of s1 by 1 → Can't be minimized.
- Replacing don't cares with all possible assignments does not guarantee a minimum solution.
- Maximal compatible classes
  - (s1, s2) ⇐ (s1, s3)
  - (s1, s3) ⇐ (s1, s2)
- Machine can be reduced to two states.

Input	State	N-State	Out
0	s1	s2	-
1	s1	s3	-
0	s2	-	0
1	s2	s1	0
0	s3	s1	1
1	s3	-	1

Input	State	N-State	Out
0	A	A	0
1	A	B	0
0	B	A	1
1	B	B	1

… State Minimization for Incompletely Specified FSMs

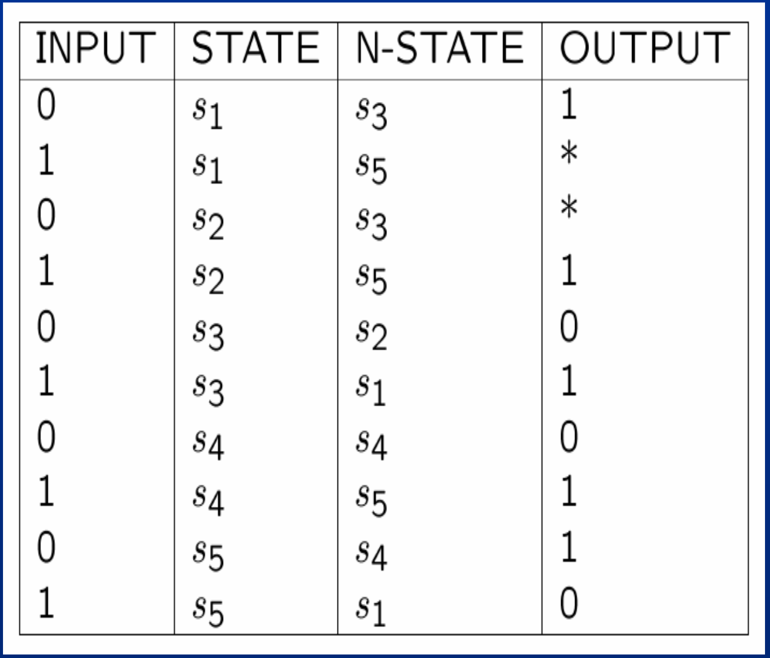
- **Minimum finite-state machine is not unique.**
- **Implication relations make problem intractable.**
- **Example**
  - Replace * by 1.
    - {(s1, s2), (s3), (s4), (s5)}.
  - Replace * by 0.
    - {(s1, s5), (s2, s3, s4)}.
  - Compatible states (s1, s2).
  - Incompatible states (s1, s3), (s1, s4), (s2, s5), (s3, s5), (s4, s5).
  - If (s3, s4) are compatible
    - then (s1, s5) are compatible.



INPUT	STATE	N-STATE	OUTPUT
0	s_1	s_3	1
1	s_1	s_5	*
0	s_2	s_3	*
1	s_2	s_5	1
0	s_3	s_2	0
1	s_3	s_1	1
0	s_4	s_4	0
1	s_4	s_5	1
0	s_5	s_4	1
1	s_5	s_1	0

Compatibility and Implications …

- **Compatible pairs**
  - (s1, s2)
  - (s1, s5) $\Leftarrow$ (s3, s4)
  - (s2, s4) $\Leftarrow$ (s3, s4)
  - (s2, s3) $\Leftarrow$ (s1, s5)
  - (s3, s4) $\Leftarrow$ (s2, s4) and (s1, s5)
- **Incompatible pairs**
  - (s2, s5), (s3, s5)
  - (s1, s4), (s4, s5)
  - (s1, s3)



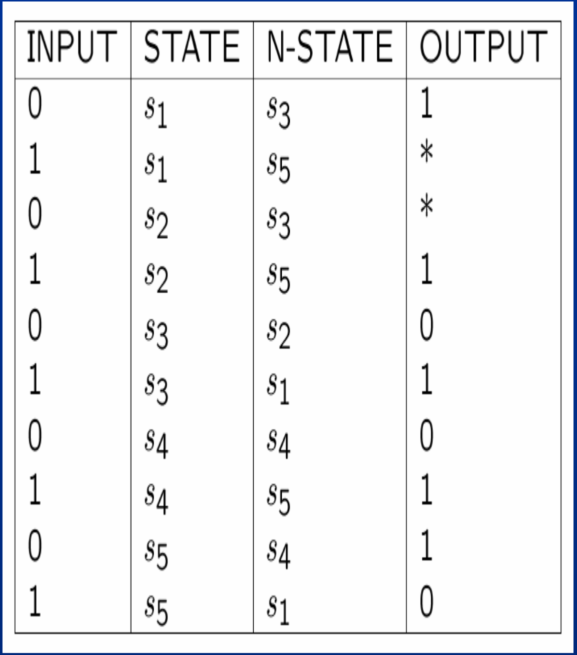
INPUT	STATE	N-STATE	OUTPUT
0	s_1	s_3	1
1	s_1	s_5	*
0	s_2	s_3	*
1	s_2	s_5	1
0	s_3	s_2	0
1	s_3	s_1	1
0	s_4	s_4	0
1	s_4	s_5	1
0	s_5	s_4	1
1	s_5	s_1	0

… Compatibility and Implications

- **A class of compatible states is such that all state pairs are compatible.**
- **A class is maximal**
  - If not subset of another class.
- **Closure property**
  - A set of classes such that all compatibility implications are satisfied.
- **The set of maximal compatibility classes**
  - Satisfies always the closure property.
  - May not provide a minimum solution.
- **Minimum covers may involve compatibility classes that are not necessarily maximal.**

Maximal Compatible Classes

- (s1, s2)
- (s1, s5) $\Leftarrow$ (s3, s4)
- (s2, s3, s4) $\Leftarrow$ (s1, s5)
- Cover with MCC has cardinality 3.
- Minimum cover: {(s1, s5) , (s2, s3, s4)}.
- Minimum cover has cardinality 2.



INPUT	STATE	N-STATE	OUTPUT
0	s_1	s_3	1
1	s_1	s_5	*
0	s_2	s_3	*
1	s_2	s_5	1
0	s_3	s_2	0
1	s_3	s_1	1
0	s_4	s_4	0
1	s_4	s_5	1
0	s_5	s_4	1
1	s_5	s_1	0

Reduced Table

Input	State	N-State	Output
0	A	B	1
1	A	A	0
0	B	B	0
1	B	A	1

Finding Maximal Compatible Classes …

- Knowing that s_i and s_j are two incompatible states implies that no maximal compatible class will contain both s_i and s_j.
- Every set of states that does not contain any incompatible pair is a compatible set.
- Associate a variable x_i to state s_i such that x_i =1 implies that s_i is an element of the set.
- Write a conjunction of two-literal clauses.
- Each clause corresponds to an incompatible pair.
- Convert the expression into a prime and irredundant sum-of-product form.
- Every term corresponds to a maximal compatible set and represents the sets that do no appear in the prime implicant.

… Finding Maximal Compatible Classes

- **Example**
  - Incompatible pairs
    - (s2, s5), (s3, s5)
    - (s1, s4), (s4, s5)
    - (s1, s3)
  - Two-literal Clauses
    - $(x_2'+x_5')\ (x_3'+x_5')\ (x_4'+x_5')\ (x_1'+x_4')\ (x_1'+x_3')$
  - Prime and Irredundant sum-of-product
    - $= (x_2'\ x_3'\ x_4'+x_5')\ (x_1'+ x_3'x_4')$
    - $= x_1'\ x_2'\ x_3'\ x_4'+ x_1'x_5'+ x_2'\ x_3'\ x_4'+ x_3'x_4'\ x_5'$
    - $= x_1'x_5'+ x_2'\ x_3'\ x_4'+ x_3'x_4'\ x_5'$
  - Maximal Compatible Classes
    - $x_1'x_5' \rightarrow (S_2,S_3,S_4)$
    - $x_2'\ x_3'\ x_4' \rightarrow (S_1,S_5)$
    - $x_3'x_4'\ x_5' \rightarrow (S_1,S_2)$

Formulation of the State Minimization Problem

- **A class is prime, if not subset of another class implying the same set or a subset of classes.**
- **Compute the prime compatibility classes.**
- **Select a minimum number of PCC such that**
  - all states are covered.
  - all implications are satisfied.
- **Binate covering problem.**
- **Upper bound on optimum solution given by**
  - Cardinality of set of maximal compatible classes and
  - Original state set cardinality.
- **Lower bound on optimum solution given by**
  - A unate cover solution that disregards implications.

Setting Up Covering Problem …

- **All states must be contained in at least one compatible class (covering).**
- **All compatible classes implied by any compatible class must be contained in a compatible class in the solution (closure).**
- **Associate a variable c_i to the i-th compatible class such that c_i =1 implies that class i is part of the solution.**
- **We write a Boolean formula (product of sum form) to express the conditions for a set of compatible classes to represent a closed cover.**
  - Terms representing covering constraints
  - Terms representing closure constraints.

… Setting Up Covering Problem …

- Maximal compatibles classes
  - c_1:(s1, s2)
  - c_2:(s1, s5) $\Leftarrow$ (s3, s4)
  - c_3:(s2, s3, s4) $\Leftarrow$ (s1, s5)
- Covering constraints
  - s1: $(c_1 + c_2)$
  - s2: $(c_1 + c_3)$
  - s3: (c_3)
  - s4: (c_3)
  - s5: (c_2)
- Closure constraints
  - $c_2 \rightarrow c_3 : (c_2' + c_3)$
  - $c_3 \rightarrow c_2 : (c_3' + c_2)$

… Setting Up Covering Problem

- **Binate covering problem**
  - $(c_1 + c_2) (c_1 + c_3) (c_2) (c_3) (c_2' + c_3) (c_3' + c_2)$
  - $=(c_1 + c_2 c_3) (c_2) (c_3) (c_2' + c_3) (c_3' + c_2)$
  - $= (c_2 c_3) (c_2' + c_3) (c_3' + c_2)$
  - $= (c_2 c_3)$
- Minimum cover: $\{c_2 c_3\}$.
- Minimum cover: {(s1, s5) , (s2, s3, s4)}.

Finding Prime Compatible Classes …

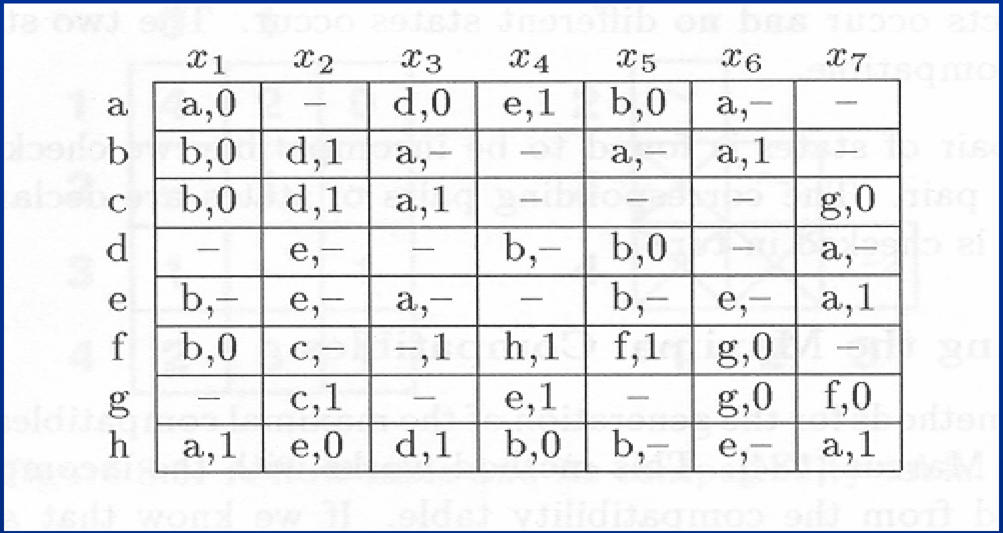
- **All maximal compatible classes are prime.**
- **Let C_1 and C_2 be two compatible classes such that $C_1 \supset C_2$.**
  - One tempted to favor C_1 over C_2 as it covers more states.
  - It might be better to choose C_2 in case that C_1 implies the selection of other compatible classes that would not be required if C_2 is selected.
- **The set of compatible classes implied by a compatible class is denoted as its *class set (CS)*.**
- **For each maximal compatible class p of size k with a *non-empty* class set, consider all its subsets S_p of compatible classes of size k-1.**
- **For each s $\in S_p$, and if there is a prime q of size > k-1, such that s $\subset$ q and $CS_s \supseteq CS_q$ then s is *not prime*. Otherwise, it is prime.**

… Finding Prime Compatible Classes

- Maximal compatibles classes
  - (s1, s2)
  - (s1, s5) $\Leftarrow$ (s3, s4)
  - (s2, s3, s4) $\Leftarrow$ (s1, s5)
- Prime compatibles classes
  - (s2, s3, s4)
    - (s2, s3) $\Leftarrow$ (s1, s5) Not prime
    - (s2, s4) $\Leftarrow$ (s3, s4) Prime
    - (s3, s4) $\Leftarrow$ (s2, s4) and (s1, s5) Not prime
  - (s1, s5)
    - (s1) Not prime
    - (s5) Prime
  - (s2, s4)
    - (s2) Not prime
    - (s4) Prime
- Set of prime compatible classes
  - (s1, s2), (s1, s5), (s2, s3, s4), (s2, s4), (s5), (s4)

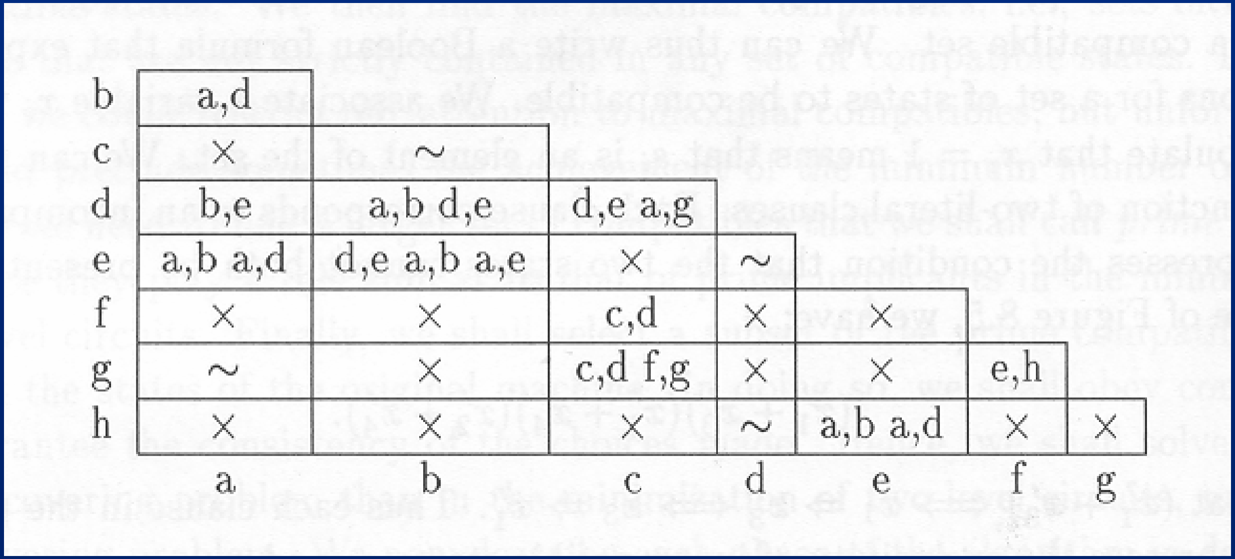
Additional State Minimization Example

Flow Table



	x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a,0	–	d,0	e,1	b,0	a,–	–
b	b,0	d,1	a,–	–	a,–	a,1	–
c	b,0	d,1	a,1	–	–	–	g,0
d	–	e,–	–	b,–	b,0	–	a,–
e	b,–	e,–	a,–	–	b,–	e,–	a,1
f	b,0	c,–	–,1	h,1	f,1	g,0	–
g	–	c,1	–	e,1	–	g,0	f,0
h	a,1	e,0	d,1	b,0	b,–	e,–	a,1

Compatibility Table



	a	b	c	d	e	f	g
b	a,d						
c	×	∼					
d	b,e	a,b d,e	d,e a,g				
e	a,b a,d	d,e a,b a,e	×	∼			
f	×	×	c,d	×	×		
g	∼	×	c,d f,g	×	×	e,h	
h	×	×	×	∼	a,b a,d	×	×

Maximal Compatibility Classes Computation

$$= \begin{gathered}(a'+c')(a'+f')(a'+h')(b'+f')(b'+g')(b'+h')(c'+e')\\(c'+h')(d'+f')(d'+g')(e'+f')(e'+g')(f'+h')(g'+h').\end{gathered}$$

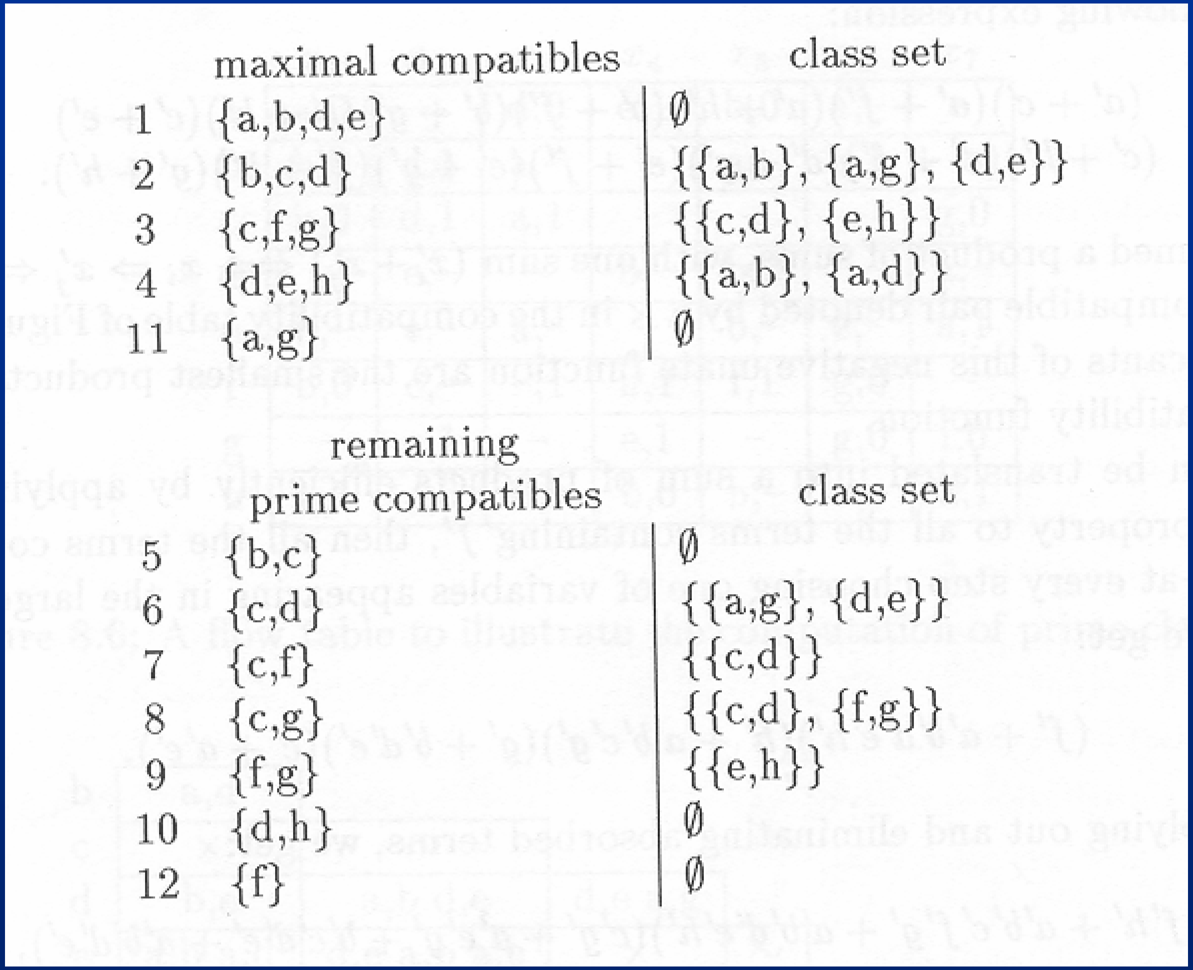
$$= (f'+a'b'd'e'h')(h'+a'b'c'g')(g'+b'd'e')(c'+a'e').$$

$$= (f'h'+a'b'c'f'g'+a'b'd'e'h')(c'g'+a'e'g'+b'c'd'e'+a'b'd'e'),$$

$$= c'f'g'h'+a'e'f'g'h'+b'c'd'e'f'h'+a'b'c'f'g'+a'b'd'e'h'.$$

$$= abde, bcd, ag, deh, cfg.$$

Prime Compatibility Classes Computation



	maximal compatibles	class set
1	{a,b,d,e}	∅
2	{b,c,d}	{{a,b}, {a,g}, {d,e}}
3	{c,f,g}	{{c,d}, {e,h}}
4	{d,e,h}	{{a,b}, {a,d}}
11	{a,g}	∅

	remaining prime compatibles	class set
5	{b,c}	∅
6	{c,d}	{{a,g}, {d,e}}
7	{c,f}	{{c,d}}
8	{c,g}	{{c,d}, {f,g}}
9	{f,g}	{{e,h}}
10	{d,h}	∅
12	{f}	∅

Setting Up the Covering Problem

- **Covering clauses: One clause for each state**
  - For state a, the only two compatibles that cover it are 1 and 11. $(c_1 + c_{11})$
- **Closure Constraints**
  - Prime 2 ({b, c, d}) requires {(a,b), (a, g), (d, e)}.

(a,b)	is found in	{a,b,d,e} (c_1)
(a,g)	is found in	{a,g} (c_{11})
(d,e)	is found in	{a,b,d,e} (c_1) and {d,e,h} (c_4).

$$c_2 \Rightarrow c_1$$
$$c_2 \Rightarrow c_{11}$$
$$c_2 \Rightarrow (c_1 + c_4)$$

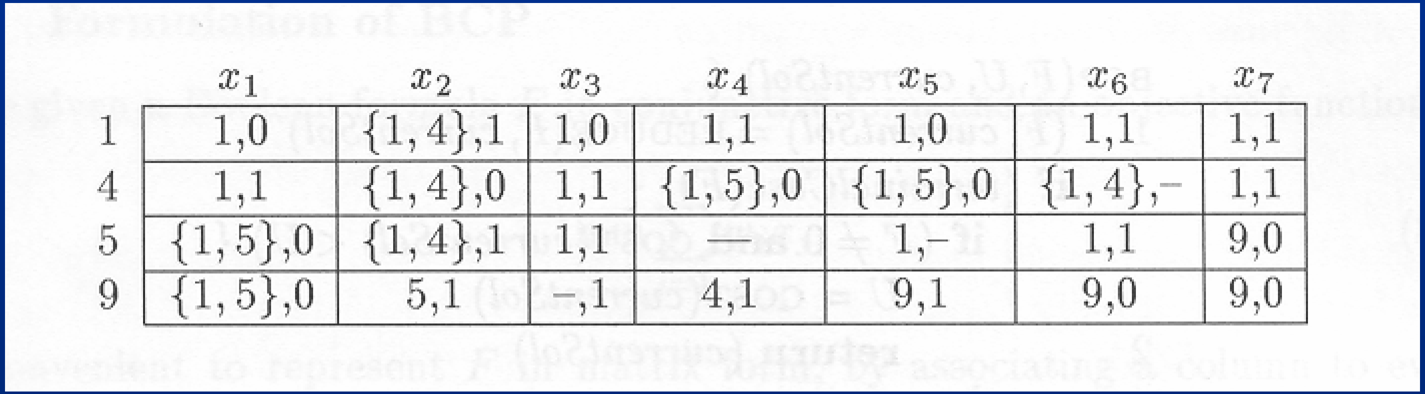
$$(c_2' + c_1)\ (c_2' + c_{11})\ (c_2' + c_1 + c_4)$$

- **Covering problem formulation**

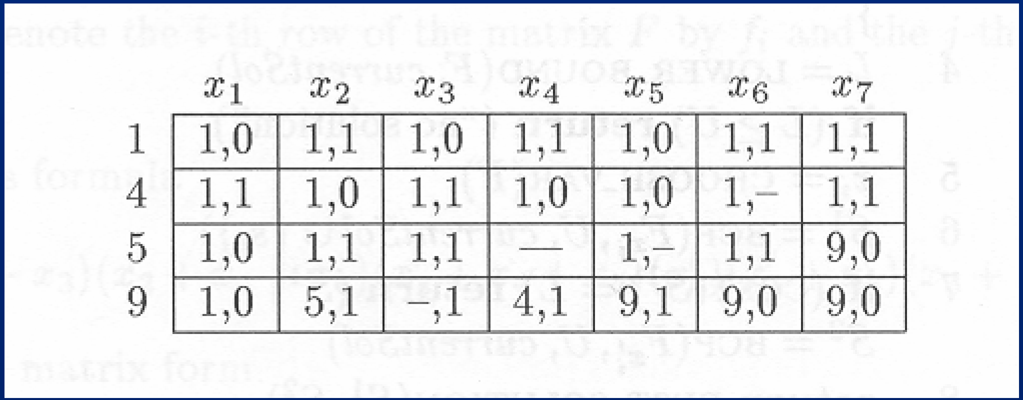
$$\begin{aligned}
&(c_1 + c_{11})(c_1 + c_2 + c_5)(c_2 + c_3 + c_5 + c_6 + c_7 + c_8)(c_1 + c_2 + c_4 + c_6 + c_{10})\\
&(c_1 + c_4)(c_3 + c_7 + c_9 + c_{12})(c_3 + c_8 + c_9 + c_{11})(c_4 + c_{10})\\
&(c_2' + c_1)(c_2' + c_{11})(c_2' + c_1 + c_4)(c_3' + c_2 + c_6)\\
&(c_3' + c_4)(c_4' + c_1)(c_4' + c_1)(c_6' + c_{11})(c_6' + c_1 + c_4)\\
&(c_7' + c_2 + c_6)(c_8' + c_2 + c_6)(c_8' + c_3 + c_9)(c_9' + c_4) = 1.
\end{aligned}$$

Reduced Table

- Minimum cost solution: $c_1=c_4=c_5=c_9=1$.
- Reduced table is not unique. It will lead to different implementations with different cost.



	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1,0	$\{1,4\}$,1	1,0	1,1	1,0	1,1	1,1
4	1,1	$\{1,4\}$,0	1,1	$\{1,5\}$,0	$\{1,5\}$,0	$\{1,4\}$,–	1,1
5	$\{1,5\}$,0	$\{1,4\}$,1	1,1	—	1,–	1,1	9,0
9	$\{1,5\}$,0	5,1	–,1	4,1	9,1	9,0	9,0



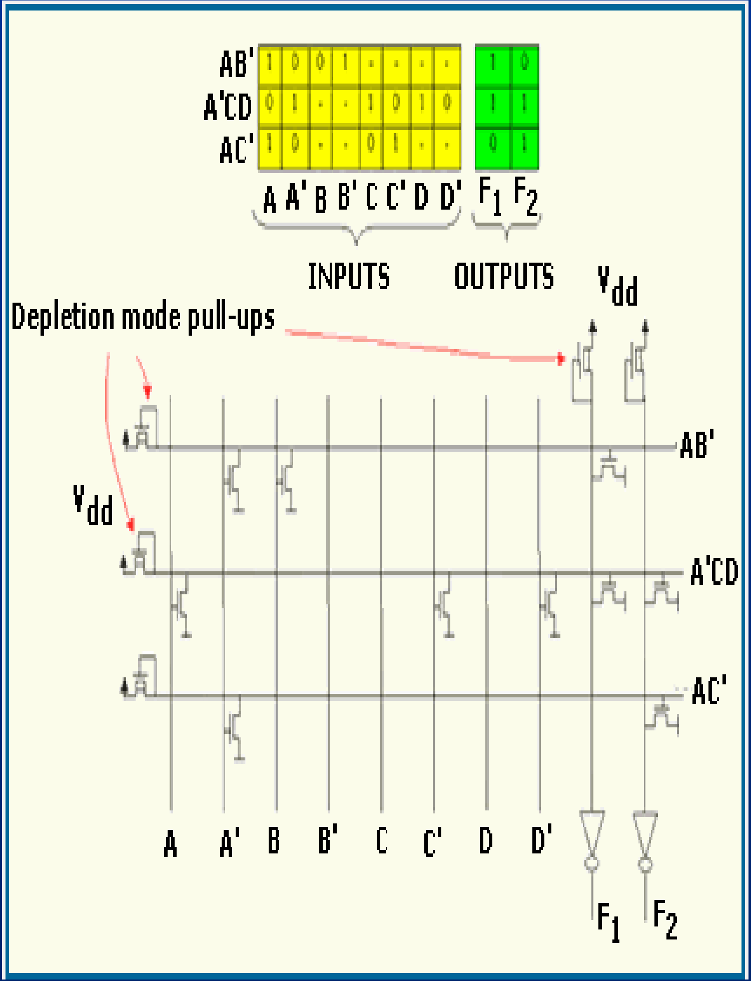
	x_1	x_2	x_3	x_4	x_5	x_6	x_7
1	1,0	1,1	1,0	1,1	1,0	1,1	1,1
4	1,1	1,0	1,1	1,0	1,0	1,–	1,1
5	1,0	1,1	1,1	—	1,–	1,1	9,0
9	1,0	5,1	–,1	4,1	9,1	9,0	9,0

State Encoding

- **Determine a binary encoding of the states ($|S|=n_s$) that optimize machine implementation**
  - Area
  - Cycle-time
  - Power dissipation
  - Testability
- **Assume D-type registers.**
- **Circuit complexity is related to**
  - Number of storage bits n_b used for state representation
  - Size of combinational component
- **There are** $2^{n_b}!/(2^{n_b}-n_s)!$ **possible encodings**
- **Implementation Modeling**
  - Two-level circuits.
  - Multiple-level circuits.

Two-Level Circuit Models

- **Sum of product representation.**
  - PLA implementation.
- **Area**
  - # of products (rows) × # I/Os (columns).
- **Delay**
  - Twice # of products (2 column length) plus # I/Os (1 row length).
- **Note**
  - # products considered as the size of a minimum implementation.
  - # I/Os depends on encoding length.



	A	A'	B	B'	C	C'	D	D'	F_1	F_2
AB'	1	0	0	1	·	·	·	·	1	0
A'CD	0	1	·	·	1	0	1	0	1	1
AC'	1	0	·	·	0	1	·	·	0	1

State Encoding for Two-Level Models

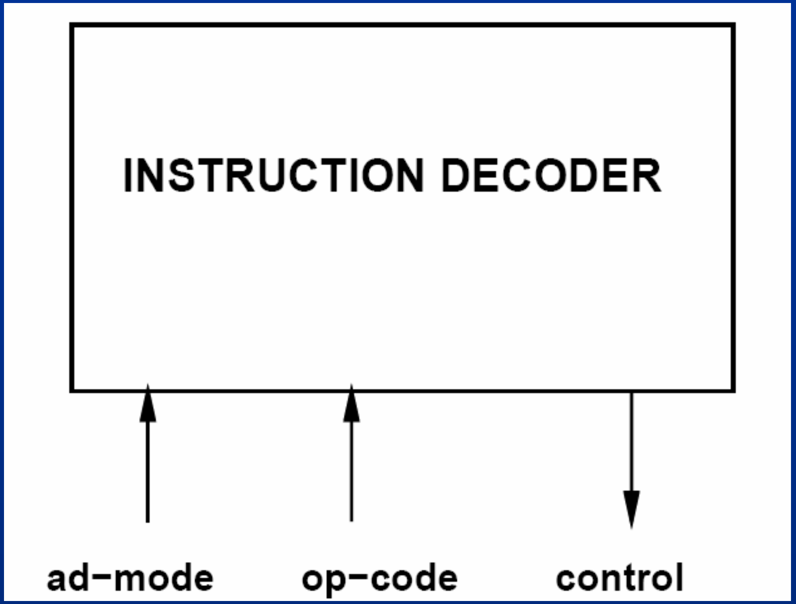
- **Early work focused on use of minimum-length codes i.e. using $n_b = \lceil \log_2 n_s \rceil$.**
- **Most classical heuristics are based on reduced dependency criterion**
  - Encode states so that state variables have least dependencies on those representing previous states.
  - Correlates weakly with minimality of sum-of-products representation.
- **Symbolic minimization of state table.**
  - Equivalent to minimizing the size of sum-of-products form related to all codes that satisfy the corresponding constraints.
- **Constrained encoding problems.**
  - Exact and heuristic methods.

Symbolic Minimization

- **Minimization of Boolean functions where codes of inputs and/or outputs are not specified.**
- **Minimize tables of symbols rather than binary tables.**
- **Extension to bvi and mvi function minimization.**
- **Reduce the number of rows of a table, that can have symbolic fields.**
- **Reduction exploits**
  - Combination of input symbols in the same field.
  - Covering of output symbols.
- **Applications**
  - Encoding of op-codes.
  - State encoding of finite-state machines.
- **Problems**
  - Input encoding.
  - Output encoding.
  - Mixed encoding.

Input Encoding Example

ad-mode	op-code	control
INDEX	AND	CNTA
INDEX	OR	CNTA
INDEX	JMP	CNTA
INDEX	ADD	CNTA
DIR	AND	CNTB
DIR	OR	CNTB
DIR	JMP	CNTC
DIR	ADD	CNTC
IND	AND	CNTB
IND	OR	CNTD
IND	JMP	CNTD
IND	ADD	CNTC



Replace symbols by binary strings to minimize corresponding covers

Definitions

- **Symbolic cover**
  - List of symbolic implicants.
  - List of rows of a table.
- **Symbolic implicant**
  - Conjunction of symbolic literals.
- **Symbolic literals**
  - Simple: one symbol.
  - Compound: the disjunction of some symbols.

Input Encoding Problem

- **Degrees of freedom in encoding the symbols.**
- **Goal**
  - Reduce size of the representation.
- **Approach**
  - Encode to minimize number of rows.
  - Encode to minimize number of bits.
- **Represent each string by 1-hot codes.**
- **Minimize table with mvi minimizer.**
- **Interpret minimized table**
  - Compound mvi-literals.
  - Groups of symbols.

Input Encoding Example …

Input Encoding Problem

ad-mode	op-code	control
INDEX	AND	CNTA
INDEX	OR	CNTA
INDEX	JMP	CNTA
INDEX	ADD	CNTA
DIR	AND	CNTB
DIR	OR	CNTB
DIR	JMP	CNTC
DIR	ADD	CNTC
IND	AND	CNTB
IND	OR	CNTD
IND	JMP	CNTD
IND	ADD	CNTC

Encoded Cover

100	1000	1000
100	0100	1000
100	0010	1000
100	0001	1000
010	1000	0100
010	0100	0100
010	0010	0010
010	0001	0010
001	1000	0100
001	0100	0001
001	0010	0001
001	0001	0010

Minimum Cover

100	1111	1000
010	1100	0100
001	1000	0100
010	0011	0010
001	**0 0 0 1**	0010
001	0110	0001

… Input Encoding Example

Minimum Symbolic Cover

INDEX	AND,OR,JMP,ADD	CNTA
DIR	AND,OR	CNTB
IND	AND	CNTB
DIR	JMP,ADD	CNTC
IND	ADD	CNTC
IND	OR,JMP	CNTD

Minimum Cover

100	1111	1000
010	1100	0100
001	1000	0100
010	0011	0010
001	**0001**	0010
001	0110	0001

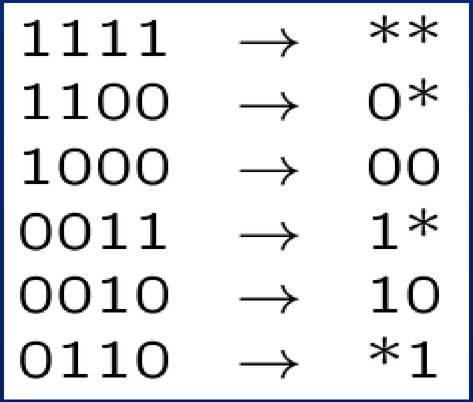
- **Examples of**
  - Simple literal: AND
  - Compound literal: AND,OR

Input Encoding Problem

- Transform minimum symbolic cover into minimum bv-cover.
- Map symbolic implicants into bv implicants (one to one).
- Compound literals
  - Encode corresponding symbols so that their supercube does not include other symbol codes.
- Replace encoded literals into cover.

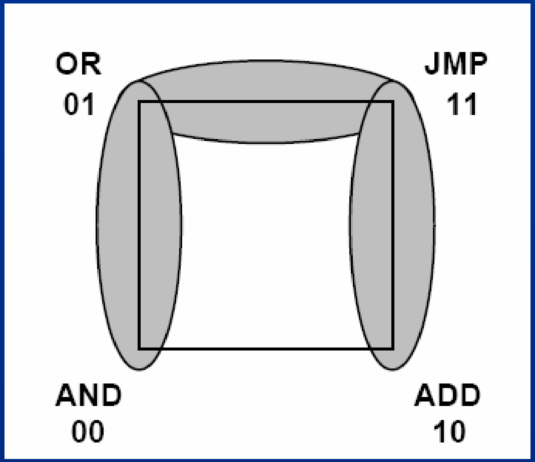
Example

- **Compound literals**
  - AND,OR,JMP,ADD
  - AND,OR
  - JMP,ADD
  - OR,JMP
- **Valid code**
  - {AND 00, OR 01, JMP 11, ADD 10}
- **Replacement in cover**

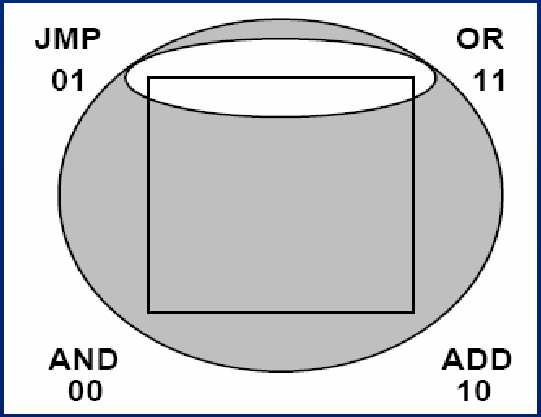


1111	$\rightarrow$	**
1100	$\rightarrow$	0*
1000	$\rightarrow$	00
0011	$\rightarrow$	1*
0010	$\rightarrow$	10
0110	$\rightarrow$	*1

Valid code



Invalid code



Input Encoding Algorithms

- **Problem specification**
  - Constraint matrix **A**
    - **One row for each constraint (literal) and one column for each symbol**
    - $a_{ij} = 1$ iff symbol j belongs to literal i.
- **Solution sought for**
  - *Encoding matrix* **E**
    - As many rows as the symbols.
    - Encoding length n_b.
- Constraint matrix

$$\mathbf{A} = \begin{array}{cccc|l} \text{AND} & \text{OR} & \text{JMP} & \text{ADD} & \\ 1 & 1 & 0 & 0 & \text{AND,OR} \\ 0 & 0 & 1 & 1 & \text{JMP,ADD} \\ 0 & 1 & 1 & 0 & \text{OR,JMP} \end{array}$$

- Encoding matrix

$$\mathbf{E} = \begin{array}{cc|l} 0 & 0 & \text{AND} \\ 0 & 1 & \text{OR} \\ 1 & 1 & \text{JMP} \\ 1 & 0 & \text{ADD} \end{array}$$

Input Encoding Problem

- **Given *constraint matrix* A**
  - Find *encoding matrix* E
    - satisfying all input encoding constraints (due to compound literals)
    - With minimum number of columns (bits).
- **An encoding matrix E satisfies the encoding constraints by A if for each row a^T of A**
  - The supercube of rows of E corresponding to 1's in a^T does not intersect any of the rows of E corresponding to 0's in a^T.
- **Identity matrix is always a valid encoding.**
  - 1-hot encoding
- **Theorem: Given a constraint matrix A, the encoding matrix $E=A^T$ satisfies the constrains of A.**

Dichotomy Theory

- **Dichotomy**
  - Two sets (L, R).
  - Bipartition of a subset of the symbol set.
- **Encoding**
  - Set of columns of E.
  - Each column corresponds to a bipartition of symbol set.
- **Rationale**
  - Encoding matrix is equivalent to a set of bipartitions.
  - Each row of the constraint matrix implies some choice on the codes i.e. induces a bipartition.

Dichotomies

- **Dichotomy associated with row a^T of A**
  - A set pair (L, R)
    - L has the symbols with the 1s in a^T
    - R has the symbols with the 0s in a^T
- **Seed dichotomy associated with row a^T of A**
  - A set pair (L, R)
    - L has the symbols with the 1s in a^T
    - R has one symbol with a 0 in a^T
- **Dichotomy associated with constraint a^T = 1100**
  - ({AND, OR}; {JMP, ADD}).
- **The corresponding seed dichotomies are**
  - ({AND, OR}; {JMP})
  - ({AND, OR}; {ADD}).

Definitions

- **Compatibility**
  - (L_1, R_1) and (L_2, R_2) are compatible if
    - $L_1 \cap L_2 = \emptyset$ and $R_1 \cap R_2 = \emptyset$ or
    - $L_1 \cap R_2 = \emptyset$ and $R_1 \cap L_2 = \emptyset$.
- **Covering**
  - Dichotomy (L_1, R_1) covers (L_2, R_2) if
    - $L_1 \supseteq L_2$ and $R_1 \supseteq R_2$ or
    - $L_1 \supseteq R_2$ and $R_1 \supseteq L_2$.
- **Prime dichotomy**
  - Dichotomy that is not covered by any compatible dichotomy of a given set.
- **Union of two compatible dichotomies is a dichotomy covering both with smallest left and right blocks.**

Exact Input Encoding

- Compute all prime dichotomies.
- Form a prime/seed dichotomy table.
- Find minimum cover of seeds by primes.
- Prime dichotomies can be computed based on Compatibility graphs of seed dichotomies by
  - Finding largest clique covering each edge
- An encoding can be found based on
  - Compatibility graphs of seed dichotomies by a clique covering
  - Conflict graphs of seed dichotomies by a graph coloring

Example …

- **Encoding matrix**

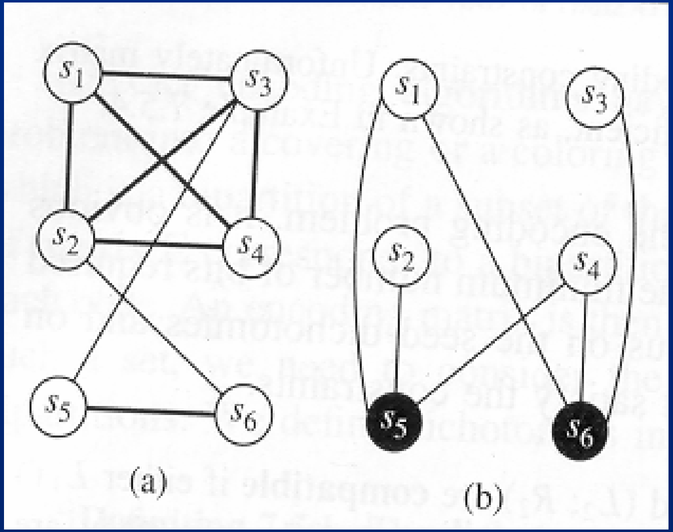
$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

- **Seed dichotomies**

s_1	({ AND,OR}	;	{ JMP })
s_2	({ AND,OR}	;	{ ADD })
s_3	({ JMP,ADD}	;	{ AND })
s_4	({ JMP,ADD}	;	{ OR })
s_5	({ OR,JMP}	;	{ AND })
s_6	({ OR,JMP}	;	{ ADD })

Prime Dichotomies

p_1	({ AND,OR}	;	{ JMP,ADD })
p_2	({ OR,JMP}	;	{ AND,ADD })
p_3	({ OR,JMP,ADD}	;	{ AND })
p_4	({ AND,OR,JMP}	;	{ ADD })



… Example

- Table

	s_1	s_2	s_3	s_4	s_5	s_6
p_1	1	1	1	1	0	0
p_2	0	0	0	0	1	1
p_3	0	0	1	0	1	0
p_4	0	1	0	0	0	1

- Minimum cover: p_1 and p_2
- Encoding

p_1 | ({ AND,OR} ; { JMP,ADD })
p_2 | ({ OR,JMP} ; { AND,ADD })

$$\mathbf{E} = \begin{array}{c} \\ \\ \\ \\ \end{array}\begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{array}{l} \text{AND} \\ \text{OR} \\ \text{JMP} \\ \text{ADD} \end{array}$$

Heuristic Encoding

- **Determine dichotomies of rows of A.**
- **Column-based encoding**
  - Construct E column by column.
- **Iterate**
  - Determine maximum compatible set.
  - Find a compatible encoding.
  - Use it as column of E.
- **Dichotomies**

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

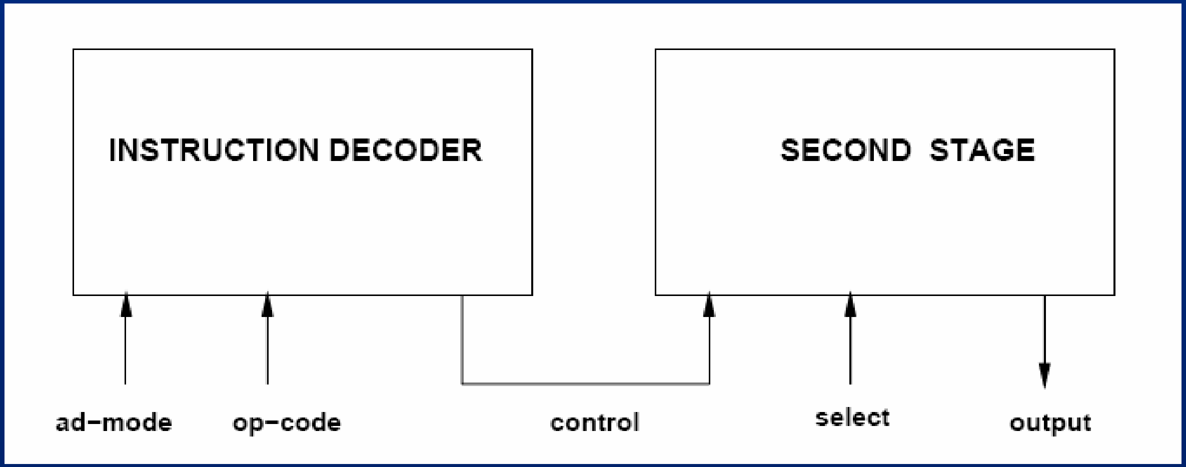
d_1	({ AND,OR }	;	{ JMP,ADD })
d_2	({ JMP,ADD }	;	{ AND,OR })
d_3	({ OR,JMP }	;	{ AND,ADD })

$$\mathbf{E} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

- First two dichotomies are compatible.
- Encoding column $[1100]^T$ satisfies d_1 , d_2.
- Need to satisfy d_3. Second encoding column $[0110]^T$.

Output and Mixed Encoding

- **Output encoding**
  - Determine encoding of output symbols.
- **Mixed encoding**
  - Determine both input and output encoding
  - Examples
    - Interconnected circuits.
    - Circuits with feedback.



Symbolic Minimization

- **Extension to mvi-minimization.**
- **Accounts for**
  - Covering relations.
  - Disjunctive relations.
- **Exact and heuristic minimizers.**
- **Minimum symbolic cover computed before**

INDEX	AND,OR,JMP,ADD	CNTA
DIR	AND,OR	CNTB
IND	AND	CNTB
DIR	JMP,ADD	CNTC
IND	ADD	CNTC
IND	OR,JMP	CNTD

- **Can we use fewer implicants?**
- **Can we merge implicants?**

Covering Relations Example

- Assume the code of CNTD covers the codes of CNTB and CNTC.

INDEX	AND,OR,JMP,ADD	CNTA
DIR	AND,OR	CNTB
IND	AND	CNTB
DIR	JMP,ADD	CNTC
IND	ADD	CNTC
IND	OR,JMP	CNTD

100	1111	1000
010	1100	0100
001	1000	0100
010	0011	0010
001	0010	0010
001	0110	0001

- Possible codes
  - CNTA = 00, CNTB = 01,
  - CNTC =10 and CNTD = 11.

100	1111	CNTA
011	1100	CNTB
011	0011	CNTC
001	0110	CNTD

Disjunctive Relations Example

- Assume the code of CNTD is the OR of the codes of CNTB and CNTC.

INDEX	AND,OR,JMP,ADD	CNTA
DIR	AND,OR	CNTB
IND	AND	CNTB
DIR	JMP,ADD	CNTC
IND	ADD	CNTC
IND	OR,JMP	CNTD

100	1111	1000
010	1100	0100
001	1000	0100
010	0011	0010
001	0010	0010
001	0110	0001

100	1111	CNTA
010	1100	CNTB
010	0011	CNTC
001	1110	CNTB
001	0111	CNTC

- **Possible codes**
  - CNTA = 00, CNTB = 01,
  - CNTC =10 and CNTD = 11.

Output Encoding Algorithms

- **Often solved in conjunction with input encoding.**
- **Exact algorithms**
  - Prime dichotomies compatible with output constraints.
  - Construct prime/seed table.
  - Solve covering problem.
- **Heuristic algorithms**
  - Construct E column by column.
- **Compatibility**
  - (L_1, R_1) and (L_2, R_2) are compatible if $L_1 \cap R_2 = \emptyset$ and $R_1 \cap L_2 = \emptyset$.
- **Covering**
  - Dichotomy (L_1, R_1) covers (L_2, R_2) if $L_1 \supseteq L_2$ and $R_1 \supseteq R_2$.
- **Prime dichotomies need to cover at least one element of all seed pairs.**

Example …

- Input constraint matrix of second stage

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

- Output constraint matrix of first stage

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

- Assume the code of CNTD covers the codes of CNTB and CNTC.

… Example …

- Seed dichotomies associated with A

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

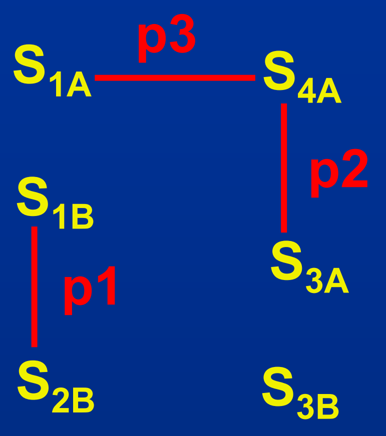
S_{1A}:	({CNTA, CNTB} ; {CNTC})
S_{1B}:	({CNTC} ; {CNTA, CNTB})
S_{2A}:	({CNTA, CNTB} ; {CNTD})
S_{2B}:	({CNTD} ; {CNTA, CNTB})
S_{3A}:	({CNTB, CNTD} ; {CNTA})
S_{3B}:	({CNTA} ; {CNTB, CNTD})
S_{4A}:	({CNTB, CNTD} ; {CNTC})
S_{4B}:	({CNTC} ; {CNTB, CNTD})

- Seed dichotomies s_{2A} and s_{4B} are not compatible with B.
- Note that only one of the seed dichotomies S_{iA} or S_{iB} needs to be covered and not both.

… Example …

- Prime dichotomies compatible with B

P1: ({CNTC, CNTD}; {CNTA, CNTB})
P2: ({CNTB, CNTD}; {CNTA, CNTC})
P3: ({CNTA, CNTB, CNTD}; {CNTC})
P4: ({CNTA}; {CNTB, CNTD})



- Cover: p_1 and p_2
- Encoding matrix

$$E = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

S_{1A}: ({CNTA, CNTB} ; {CNTC})
S_{1B}: ({CNTC} ; {CNTA, CNTB})
~~S_{2A}: ({CNTA, CNTB} ; {CNTD})~~
S_{2B}: ({CNTD} ; {CNTA, CNTB})
S_{3A}: ({CNTB, CNTD} ; {CNTA})
S_{3B}: ({CNTA} ; {CNTB, CNTD})
S_{4A}: ({CNTB, CNTD} ; {CNTC})
~~S_{4B}: ({CNTC} ; {CNTB, CNTD})~~

… Example

- Heuristic encoding considers the dichotomy pairs associated with each row

d_{1A}: ({CNTC, CNTD} ; {CNTA, CNTB})
d_{1B}: (~~{CNTA, CNTB} ; {CNTC, CNTD}~~)
d_{2A}: ({CNTB, CNTD} ; {CNTA, CNTC})
d_{2B}: (~~{CNTA, CNTC} ; {CNTB, CNTD}~~)

- Dichotomies d_{1B} and d_{2B} do not satisfy output constraints.
- Dichotomies d_{1A} and d_{2A} are not compatible and considered one at a time yielding two-column encoding.

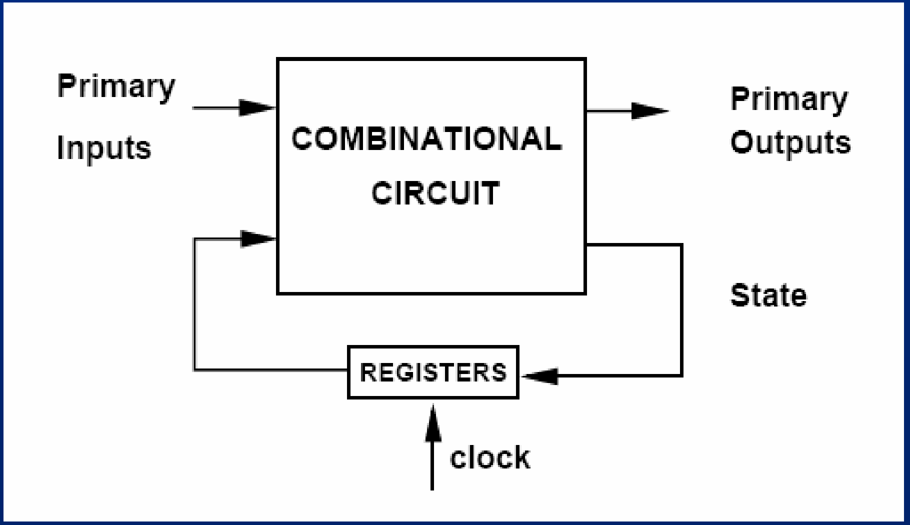
$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$E = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

State Encoding of Finite-State Machines

- Given a (minimum) state table of a finite-state machine
  - **Find a consistent encoding of the states**
    - **that preserves the cover minimality**
    - **with minimum number of bits.**
- **The state set must be encoded while satisfying simultaneously both input and output constraints.**



Example …

INPUT	P-STATE	N-STATE	OUTPUT
0	s_1	s_3	0
1	s_1	s_3	0
0	s_2	s_3	0
1	s_2	s_1	1
0	s_3	s_5	0
1	s_3	s_4	1
0	s_4	s_2	1
1	s_4	s_3	0
0	s_5	s_2	1
1	s_5	s_5	0

Minimum Symbolic Cover

*	$s_1 s_2 s_4$	s_3	0
1	s_2	s_1	1
0	$s_4 s_5$	s_2	1
1	s_3	s_4	1

Covering Constraints

- s1 and s2 cover s3
- s5 is covered by all other states.

Encoding Constraint Matrices

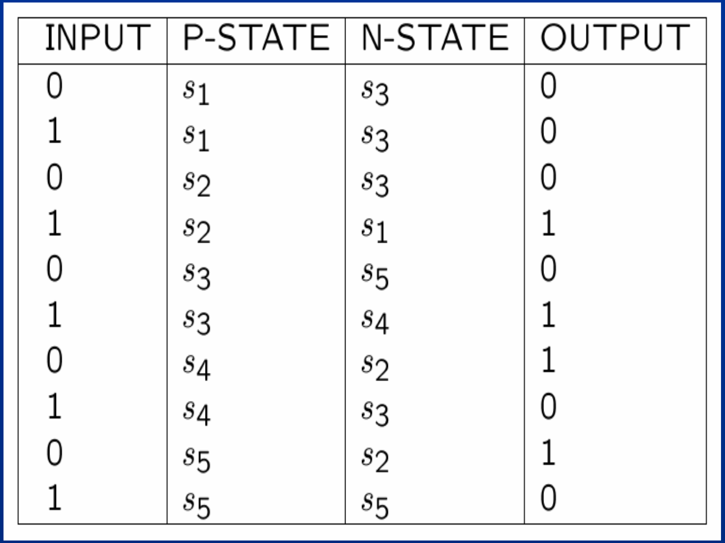
$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

… Example …

After Implicant Merging:

0	*s1, s2*	*s3*	*0*
1	*s2*	*s1*	*1*
1	*s3*	*s4*	*1*
0	*s4, s5*	*s2*	*1*
1	*s1, s4*	*s3*	*0*
0	*s3*	*s5*	*0*
1	*s5*	*s5*	*0*



INPUT	P-STATE	N-STATE	OUTPUT
0	s_1	s_3	0
1	s_1	s_3	0
0	s_2	s_3	0
1	s_2	s_1	1
0	s_3	s_5	0
1	s_3	s_4	1
0	s_4	s_2	1
1	s_4	s_3	0
0	s_5	s_2	1
1	s_5	s_5	0

If s1 $\supseteq$ s3 and s2 $\supseteq$ s3:

*	*s1, s2, s4*	*s3*	*0*
1	*s2*	*s1*	*1*
1	*s3*	*s4*	*1*
0	*s4, s5*	*s2*	*1*
0	*s3*	*s5*	*0*
1	*s5*	*s5*	*0*

Last two rows, output is 0 => no impact on output equation. If s5 is covered by all other states, a minimal code will have s5 assigned the 0 code, i.e. no impact on next state equations => last two rows can be eliminated.

*	*s1, s2, s4*	*s3*	*0*
1	*s2*	*s1*	*1*
1	*s3*	*s4*	*1*
0	*s4, s5*	*s2*	*1*

… Example

- Encoding matrix (one row per state)

$$\mathbf{E} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Covering Constraints

•s1 and s2 cover s3

•s5 is covered by all other states.

- Encoded cover of combinational component

*	1**	001	0
1	101	111	1
0	*00	101	1
1	001	100	1

*	$s_1 s_2 s_4$	s_3	0
1	s_2	s_1	1
0	$s_4 s_5$	s_2	1
1	s_3	s_4	1

Limitation of Symbolic Minimization and Constrained Encoding

- **The minimum-length solution compatible with a set of constraints may require bits larger than $\lceil \log_2 n_s \rceil$.**
- **Example: Consider an FSM whose minimal symbolic cover is:**

00	s1, s2	s3	100
01	s2, s3	s1	010
10	s1, s3	s2	001

- **Satisfaction of input constraints requires at least 3 bits.**
  - e.g. s1=100, s2=010, s3=001
  - PLA size is 3 rows and 11 columns (2 PI+3 PS+3 NS+3 PO)
- **Assume we do not satisfy first input constraint**
  - 2 bits are sufficient
  - PLA size is 4 rows and 9 columns (2 PI+2 PS+2 NS+3 PO)

State Encoding for Multiple-Level Models

- Logic network representation.
- Area: # of literals.
- Delay: Critical path length.
- Encoding based on cube-extraction heuristics [Mustang-Devadas].
- Rationale
  - When two (or more) states have a transition to the same next-state
    - Keep the distance of their encoding short.
    - Extract a large common cube.
- Exploit first stage of logic.
- Works fine because most FSM logic is shallow.

Example

- **5-state FSM (3-bits).**
  - s1 $\rightarrow$ s3 with input *i*.
  - s2 $\rightarrow$ s3 with input *i'*.
- **Encoding**
  - s1 $\rightarrow$ 000 = *a'b'c'*.
  - s2 $\rightarrow$ 001 = *a'b'c*.
- Transition
  - ***i a'b'c' +i' a'b'c = a'b' (ic+i'c')***
  - **6 literals instead of 8.**

Algorithm

- **Examine all state pairs**
  - Complete graph with |V| = |S|.
- **Add weight on edges**
  - Model desired code proximity.
  - The higher the weight the lower the distance.
- **Embed graph in the Boolean space.**
  - Objective is to minimize the cost function

$$\sum_{i=1}^{N_s} \sum_{j=i+1}^{N_s} Weight\ (v_i, v_j) * Dist\ (v_i, v_j)$$

- **Difficulties**
  - The number of occurrences of common factors depends on the next-state encoding.
  - The extraction of common cubes interact with each other.

Mustang Algorithm Implementation …

- **Fanout-oriented algorithm**
  - Consider *state fanout* i.e. next states and outputs.
  - Assign closer codes to pair of states that have same next state transition or same output.
  - *Maximize the size of the most frequent common cubes*.

$$M^P_{k,l} = \sum_{i=1}^{m_0}(P^o_{k,i} * P^o_{l,i}) + \frac{n_E}{2}\sum_{i=1}^{n_s}(P^s_{k,i} * P^s_{l,i})$$

$P^o_{k,i}$ is the number of times state k is represented in output i,

$P^s_{k,i}$ is number of times state k is represented in state i,

m_o is the number of outputs,

n_s is the number of states,

n_E is the number of encoding bits.

… Mustang Algorithm Implementation …

- **Fanin-oriented algorithm**
  - Consider *state fan-in* i.e. present states and inputs.
  - Assign closer codes to pair of states that have transition from same present state or same input.
  - *Maximize the frequency of the largest common cubes*.

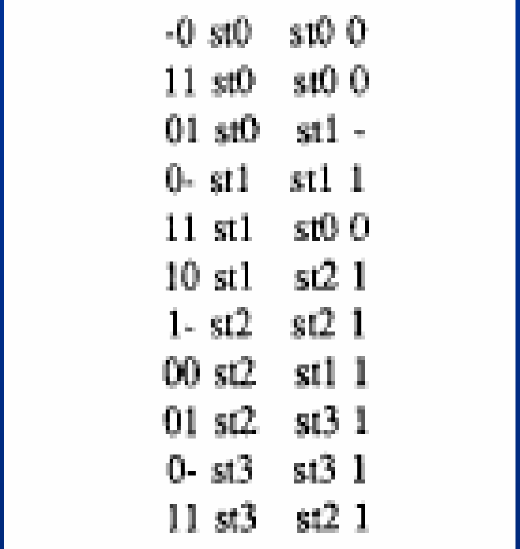
$$M_{k,l} = \sum_{i=1}^{ni} (P_{k,i}^{ON} * P_{l,i}^{ON}) + (P_{k,i}^{OFF} * P_{l,i}^{OFF}) + n_b * \sum_{i=1}^{ns} (P_{k,i}^{s} * P_{l,i}^{s})$$

$P_{k,i}^{ON}$: number of ocurrences of input i in equation of state k

$P_{k,i}^{OFF}$: number of ocurrences of input $\bar{i}$ in equation of state k

$P_{k,i}^{s}$: number of ocurrences of state i in equation of state k

… Mustang Algorithm Implementation



```
-0 st0   st0 0
11 st0   st0 0
01 st0   st1 -
0- st1   st1 1
11 st1   st0 0
10 st1   st2 1
1- st2   st2 1
00 st2   st1 1
01 st2   st3 1
0- st3   st3 1
11 st3   st2 1
```

Fig. 1. Example FSM.

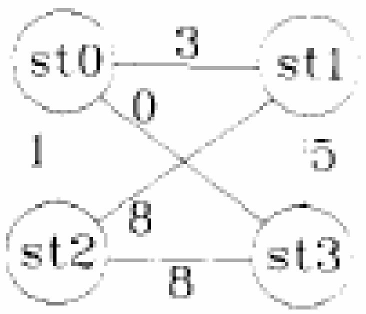


Fig. 5. Graph generated by fanout-oriented algorithm.

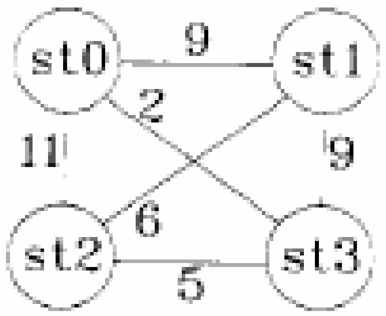
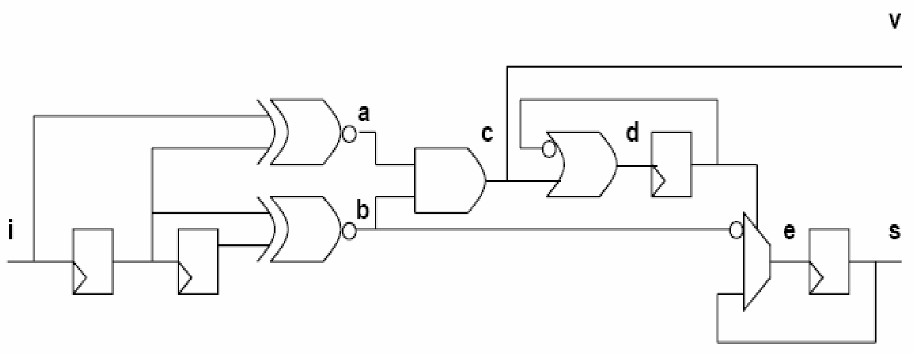


Fig. 6. Graph generated by fanin-oriented algorithm.

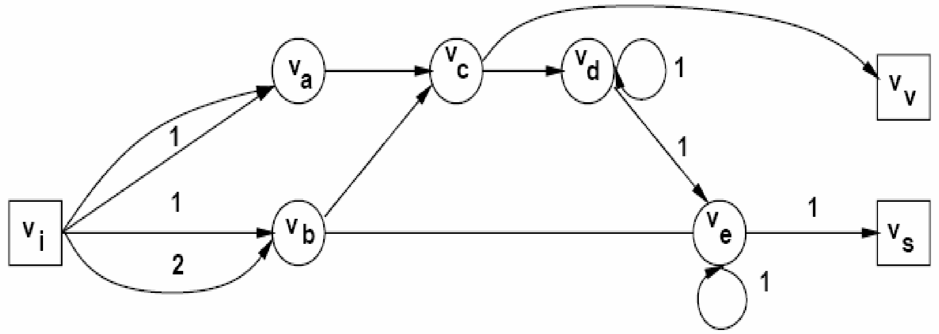
Synchronous Logic Network …

- **Synchronous Logic Network**
  - Variables.
  - Boolean equations.
  - Synchronous delay annotation.
- **Synchronous network graph**
  - Vertices $\leftrightarrow$ equations $\leftrightarrow$ I/O , gates.
  - Edges $\leftrightarrow$ dependencies $\leftrightarrow$ nets.
  - Weights $\leftrightarrow$ synch. delays $\leftrightarrow$ registers.

… Synchronous Logic Network



(a)



(b)

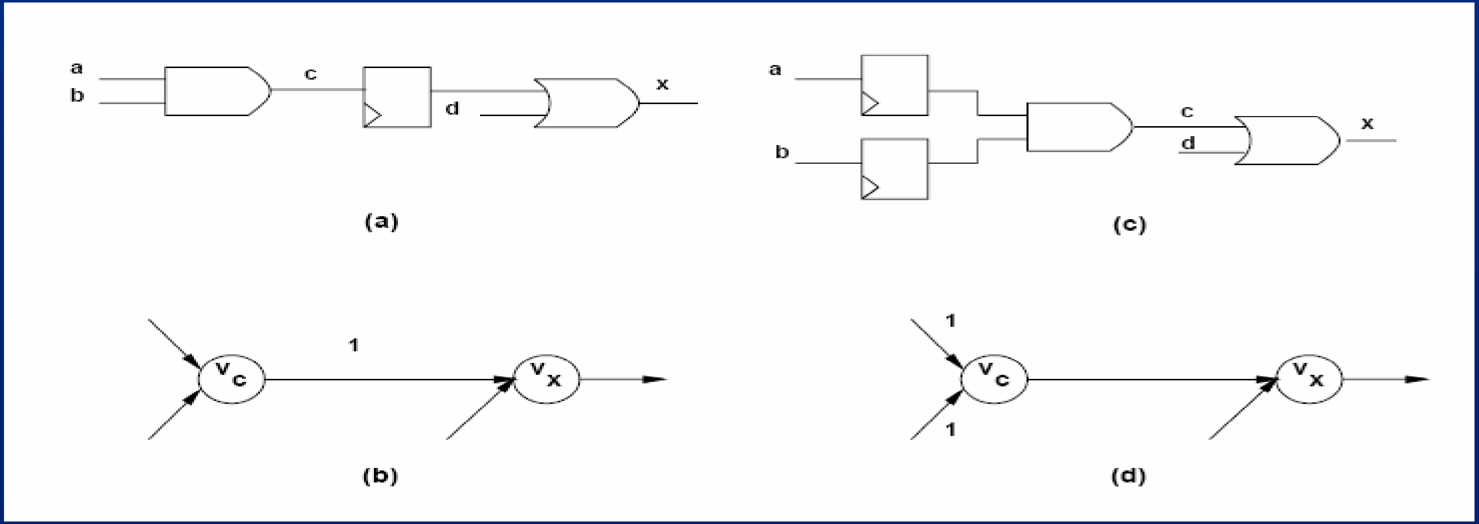
$$
\begin{aligned}
a^{(n)} &= i^{(n)} \overline{\oplus}\, i^{(n-1)} \\
b^{(n)} &= i^{(n-1)} \overline{\oplus}\, i^{(n-2)} \\
c^{(n)} &= a^{(n)} b^{(n)} \\
d^{(n)} &= c^{(n)} + d'^{(n-1)} \\
e^{(n)} &= d^{(n)} e^{(n-1)} + d'^{(n)} b'^{(n)} \\
v^{(n)} &= c^{(n)} \\
s^{(n)} &= e^{(n-1)}
\end{aligned}
$$

Approaches to Synchronous Logic Optimization

- **Optimize combinational logic only.**
- **Optimize register position only**
  - Retiming.
- **Optimize overall circuit**
  - Peripheral retiming.
  - Synchronous transformations
    - Algebraic.
    - Boolean.

Retiming …

- Minimize cycle-time or area by changing register positions.
- Do not modify combinational logic.
- Preserve network structure
  - Modify weights.
  - Do not modify graph structure.



… Retiming …

- **Global optimization technique [Leiserson].**
- **Changes register positions**
  - affects area
    - changes register count.
  - affects cycle-time
    - changes path delays between register pairs.
- **Solvable in polynomial time.**
- **Assumptions**
  - Vertex delay is constant: No fanout delay dependency.
  - Graph topology is invariant: No logic transformations.
  - Synchronous implementation
    - Cycles have positive weights.
    - Edges have non-negative weights.

… Retiming …

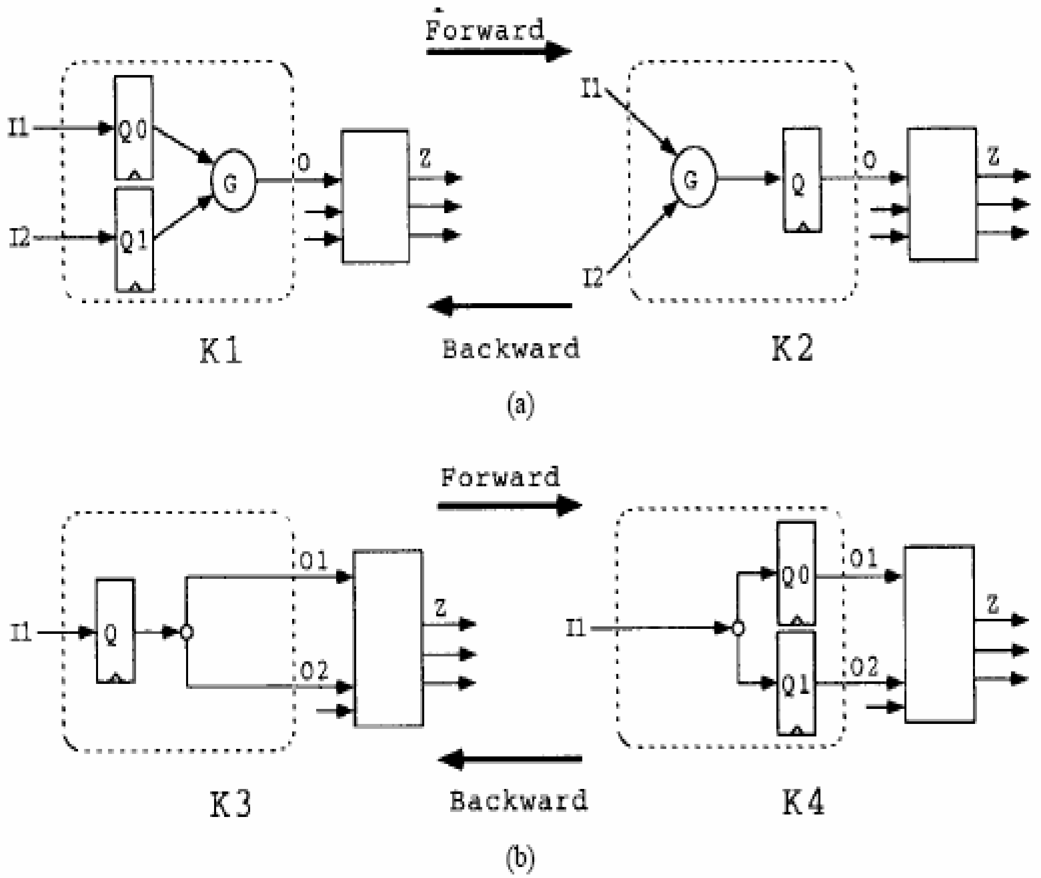


Fig. 1. (a) Retiming forward/backward across a single-output combinational gate and (b) retiming forward/backward across a fanout stem.

… Retiming …

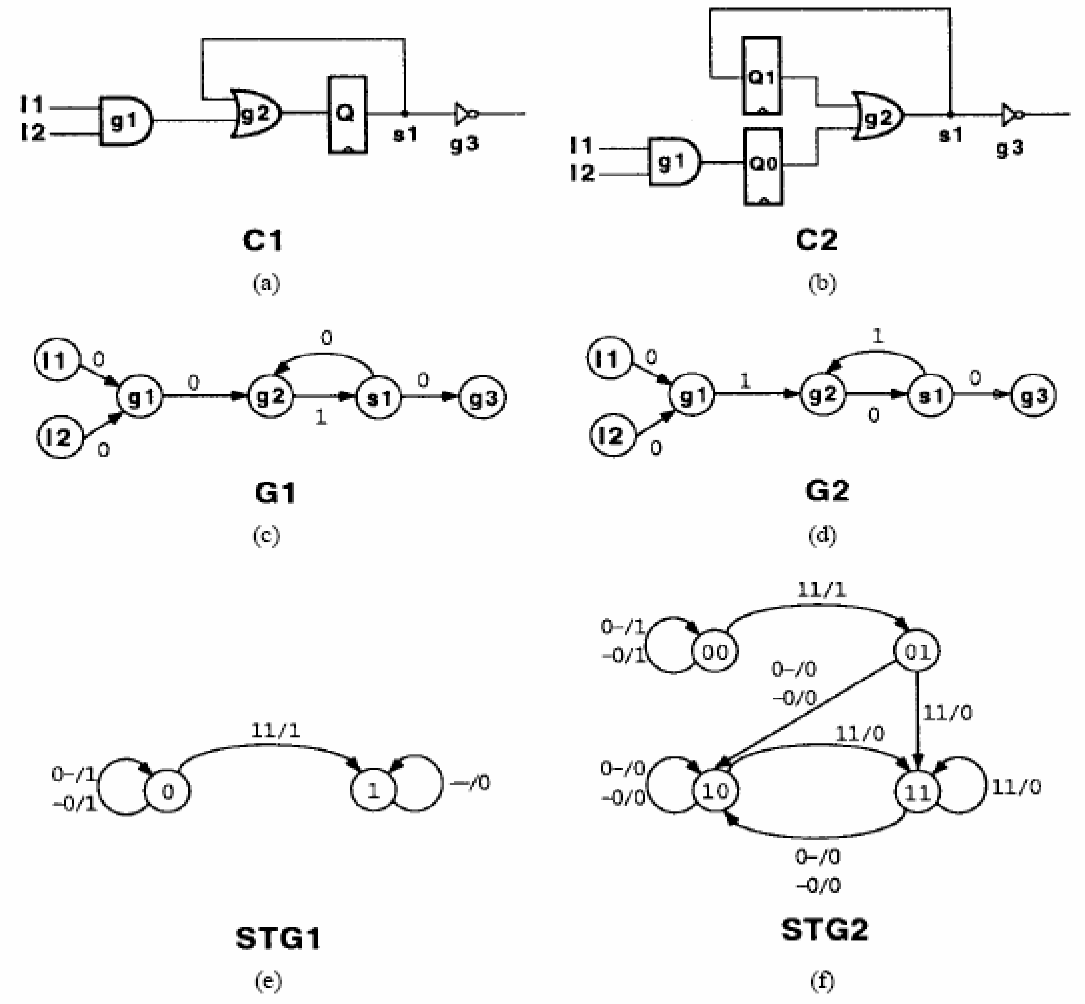


Fig. 2. An example of a backward retiming move across a single-output combinational gate.

… Retiming

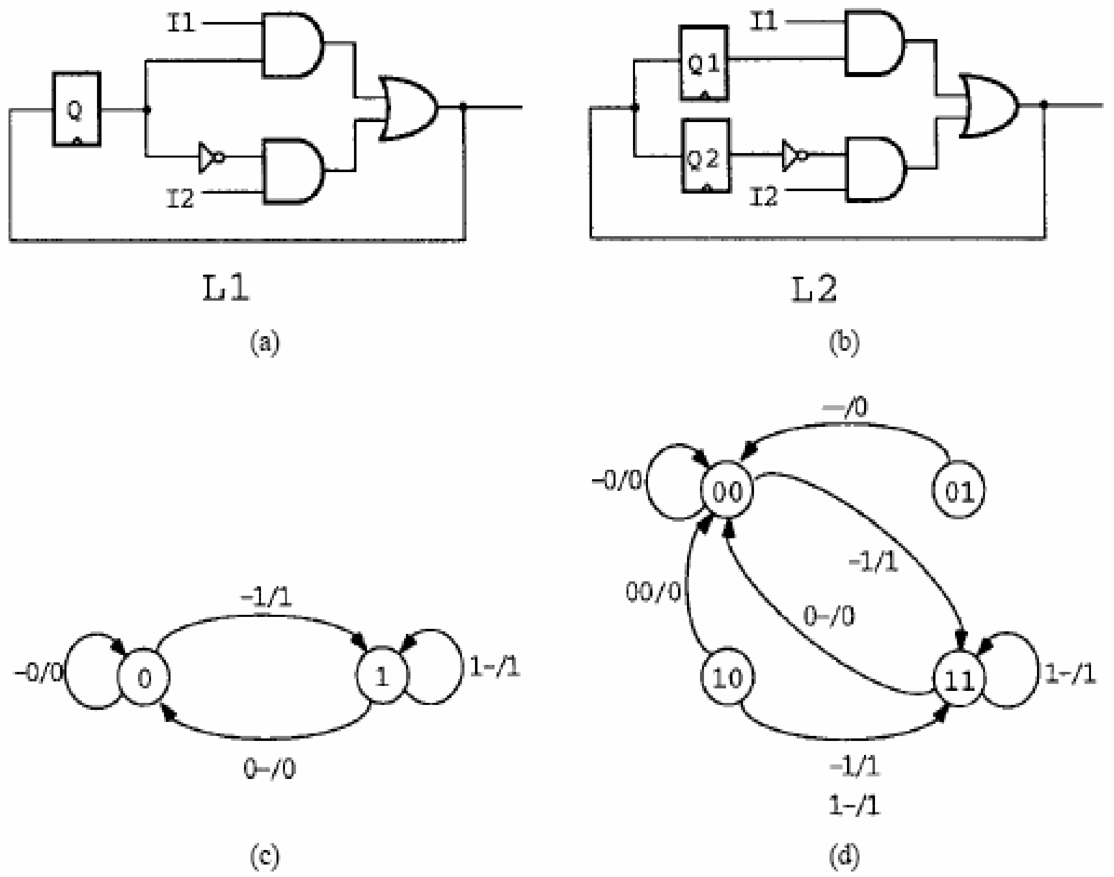
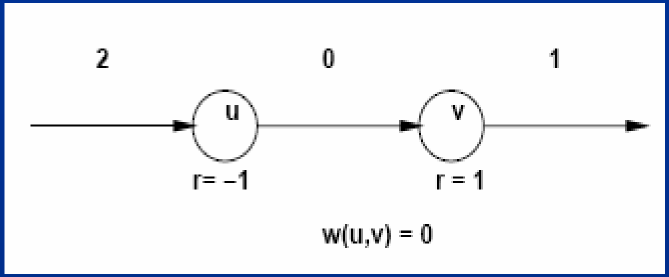
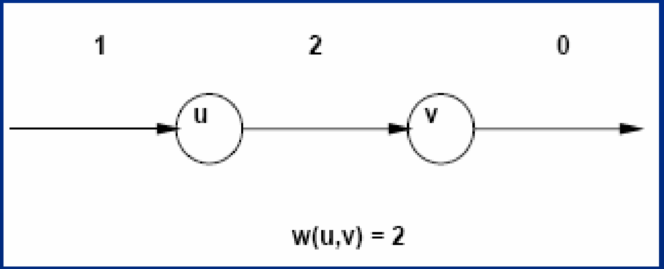


Fig. 3. An example of a forward retiming move across a fanout stem.

… Retiming …

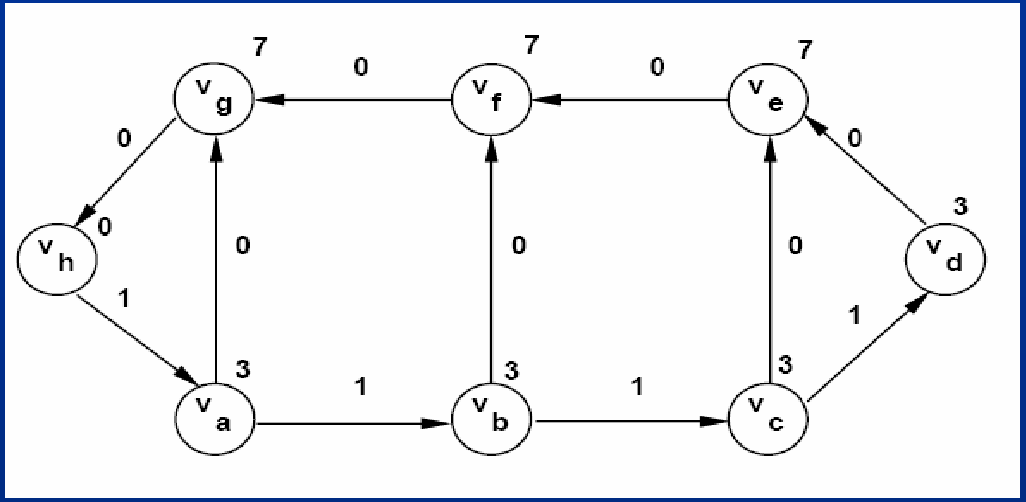
- **Retiming of a vertex**
  - Moving registers from output to inputs or vice versa
  - Integer
    - Positive value: from outputs to inputs
    - Negative value: from inputs to outputs
- **Retiming of a network**
  - Vector of vertex retiming.
- Definitions
  - $w(v_i, v_j)$ – weight of edge (v_i, v_j).
  - Retiming of an edge (v_i, v_j): $\tilde{w}_{ij} = w_{ij} + r_j - r_i$.

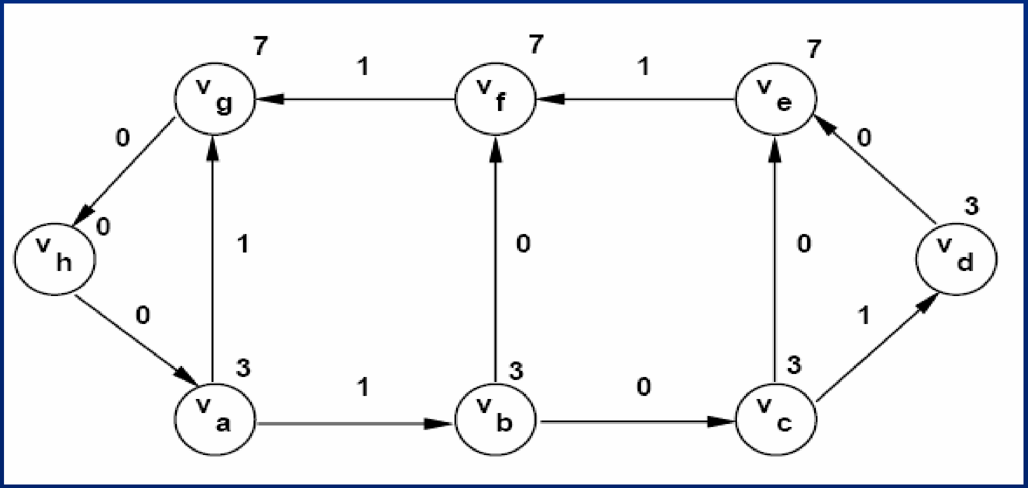




Example

- Node values indicate delay.
- Retiming vector
  - [a b c d e f g h]= [0 0 -1 -1 -1 0 1 1]
  - [a b c d e f g h]= - [1 1 2 2 2 1 0 0]=
  - [1 1 0 0 0 1 2 2]
- Original critical path is (v_d, v_e, v_f, v_g, v_h)= 24 units
- Retimed critical path is (v_b, v_c, v_e)= 13 units

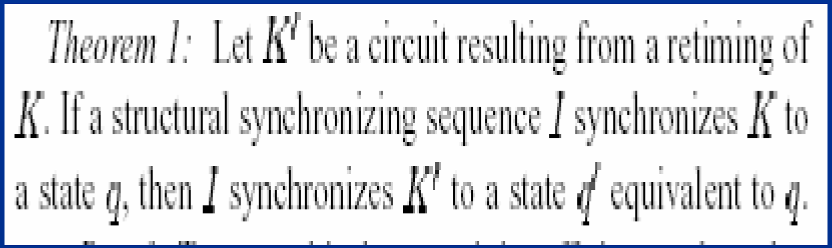




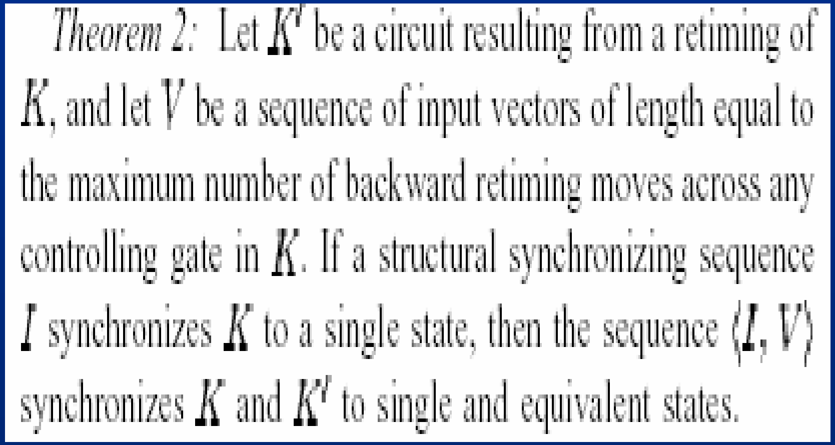
Behavior and Testability Preservation under Retiming …

- A **synchronizing sequence** for a machine is an input sequence that brings the machine to a known and unique state determined without knowledge of output response or initial state of the machine.
- An interesting class of synchronizing sequences allows the state reached after applying the synchronizing sequence to be either a single state or a set of equivalent states.
- A synchronizing sequence (or a test) derived based on pessimistic 3-valued simulation is called **structural**; otherwise, it is called **functional**.
  - functional synchronizing sequences (or tests) correspond to those derived based on the state transition graph of a circuit.

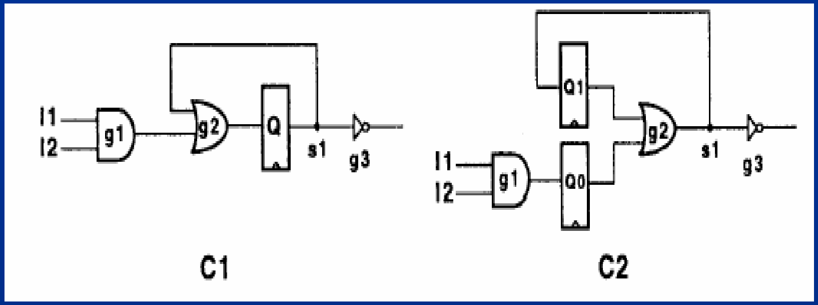
… Behavior and Testability Preservation under Retiming …

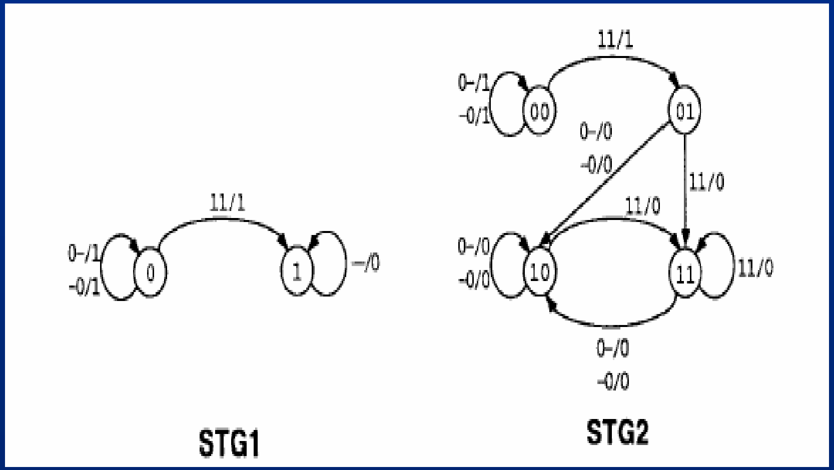


Theorem 1: Let K' be a circuit resulting from a retiming of K. If a structural synchronizing sequence I synchronizes K to a state q, then I synchronizes K' to a state q' equivalent to q.



Theorem 2: Let K' be a circuit resulting from a retiming of K, and let V be a sequence of input vectors of length equal to the maximum number of backward retiming moves across any controlling gate in K. If a structural synchronizing sequence I synchronizes K to a single state, then the sequence (I, V) synchronizes K and K' to single and equivalent states.

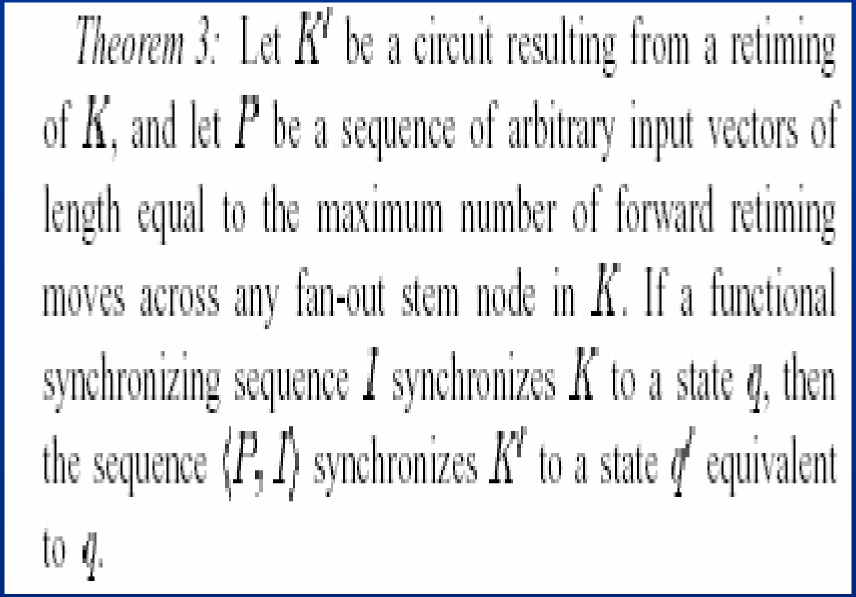




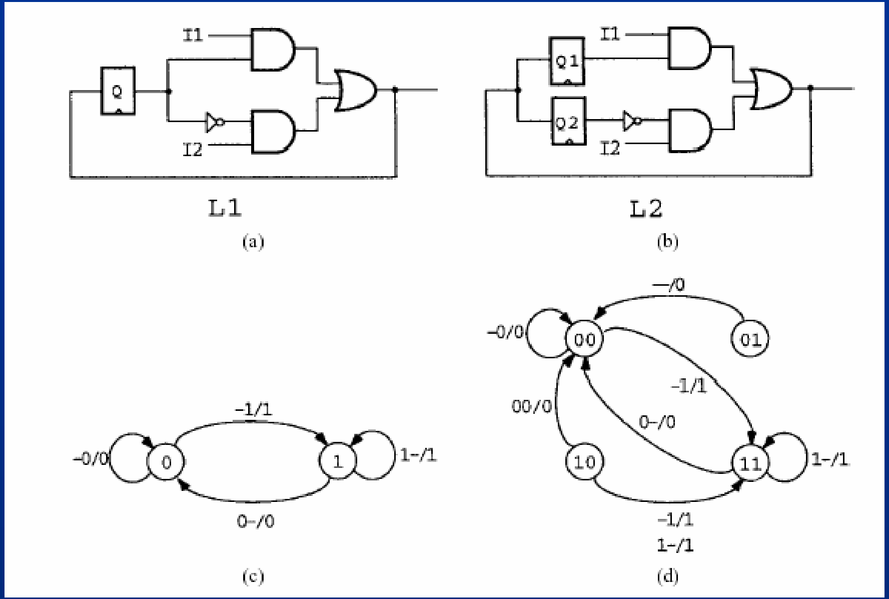
•*Vector 11 synchronizes C1 to a single state and C2 to an equivalent set of states.*

•*Any of the vectors (11, 00), (11, 01), (11, 10), (11, 11) synchronizes C2 to a single equivalent state.*

… Behavior and Testability Preservation under Retiming …



Theorem 3: Let K' be a circuit resulting from a retiming of K, and let P be a sequence of arbitrary input vectors of length equal to the maximum number of forward retiming moves across any fan-out stem node in K. If a functional synchronizing sequence I synchronizes K to a state q, then the sequence (P, I) synchronizes K' to a state q' equivalent to q.



•*Vector 11 is a functional synchronizing sequence for L1 but not for L2.*
•*Any of the vectors (00, 11), (01, 11), (10, 11), (11, 11) is a synchronizing sequence for L2.*

… Behavior and Testability Preservation under Retiming

Theorem 5: Let K' be a circuit resulting from a retiming of K, and let P be a sequence of arbitrary vectors of length equal to the maximum number of forward retiming moves across any node in K. For every fault f' in K', there exists a corresponding fault f in K such that if T is a test for f in K, then the sequence $\langle P, T\rangle$ is a test for f' in K'.

Corollary 1: Retiming preserves single stuck-at fault testability.

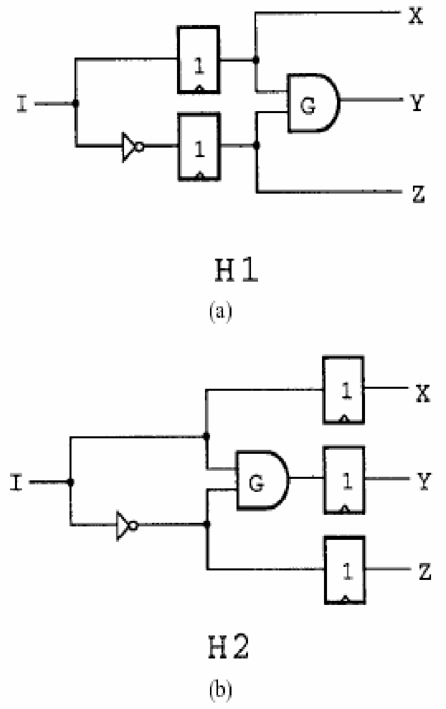


Fig. 6. An example illustrating that retiming may not preserve single stuck-at fault testability under a hardware reset or a global reset state assumption.