Chapter #9: Finite State Machine Optimization

Contemporary Logic Design

Randy H. Katz
University of California, Berkeley

July 1993

Chapter Outline

- ***Procedures for optimizing implementation of an FSM***
  - **State Reduction**
  - **State Assignment**
- ***Computer Tools for State Assignment: Nova, Mustang, Jedi***
- ***Choice of Flipflops***
- ***FSM Partitioning***

Motivation

Basic FSM Design Procedure:

(1) Understand the problem

(2) Obtain a formal description

(3) Minimize number of states

(4) Encode the states

(5) Choose FFs to implement state register

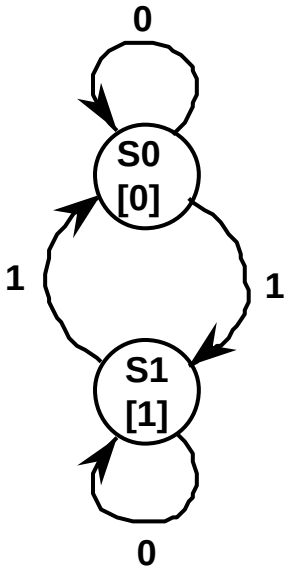
(6) Implement the FSM

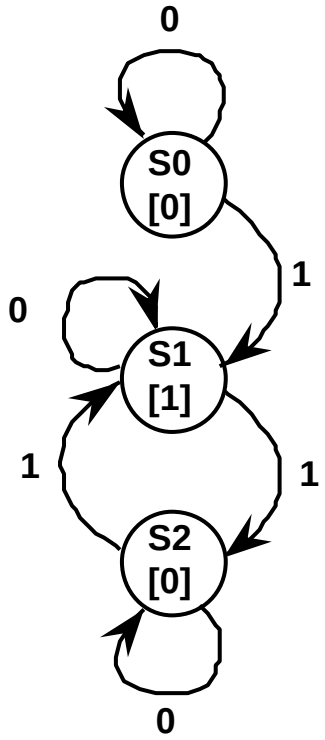
This Chapter!

Next Chapter

Motivation

State Reduction





Odd Parity Checker: two alternative state diagrams

- **Identical output behavior on all input strings**
- **FSMs are *equivalent*, but require different implementations**
- **Design state diagram without concern for # of states, Reduce later**

Motivation

State Reduction (continued)

Implement FSM with fewest possible states

- **Least number of flipflops**
- **Boundaries are power of two number of states**
- **Fewest states usually leads to more opportunities for don't cares**
- **Reduce the number of gates needed for implementation**

State Reduction

Goal

Identify and combine states that have equivalent behavior

***Equivalent States*: for all input combinations, states transition to the same or equivalent states**

***Odd Parity Checker*: S0, S2 are equivalent states**
Both output a 0
Both transition to S1 on a 1 and self-loop on a 0

Algorithmic Approach

- **Start with state transition table**
- **Identify states with same output behavior**
- **If such states transition to the same next state, they are equivalent**
- **Combine into a single new renamed state**
- **Repeat until no new states are combined**

State Reduction

Row Matching Method

Example FSM Specification:

Single input X, output Z
Taking inputs grouped four at a time, output 1 if last four inputs were the string 1010 or 0110

Example I/O Behavior:

X = 0010 0110 1100 1010 0011 …
Z = 0000 0001 0000 0001 0000 …

Upper bound on FSM complexity:

Fifteen states (1 + 2 + 4 + 8)

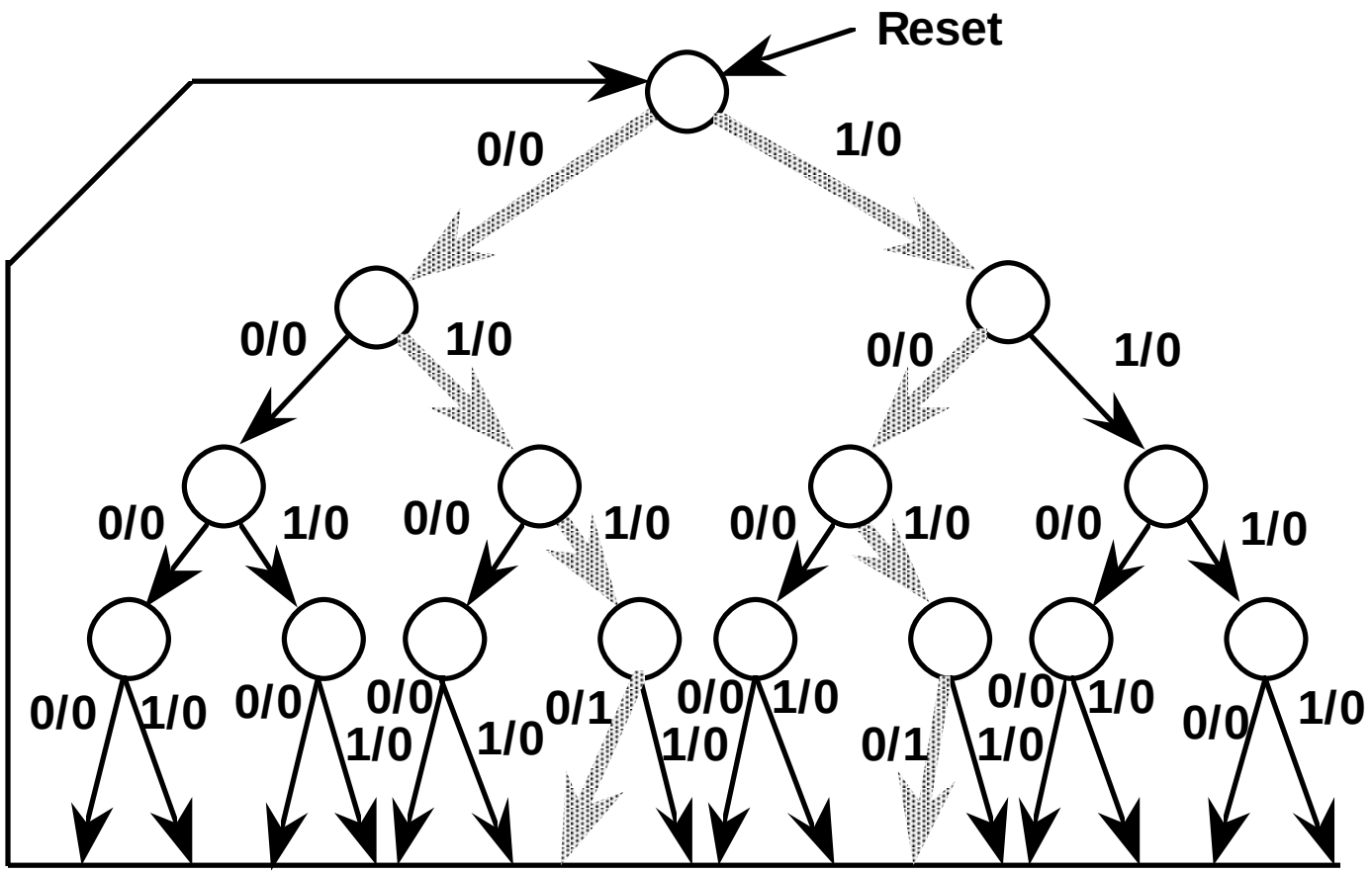
Thirty transitions (2 + 4 + 8 + 16)

sufficient to recognize any binary string of length four!

State Reduction

Row Matching Method

State Diagram for Example FSM:



State Reduction

Row Matching Method

Initial State Transition Table:

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S_{10}	0	0
10	S_5	S_{11}	S_{12}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011	S_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
101	S_{12}	S_0	S_0	1	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

State Reduction

Row Matching Method

Initial State Transition Table:

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S_{10}	0	0
10	S_5	S_{11}	S_{12}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011	S_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
101	S_{12}	S_0	S_0	1	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

State Reduction

Row Matching Method

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S'_{10}	0	0
10	S_5	S_{11}	S'_{10}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

State Reduction

Row Matching Method

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S'_{10}	0	0
10	S_5	S_{11}	S'_{10}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

State Reduction

Row Matching Method

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S'_7	S'_7	0	0
01	S_4	S'_7	S'_{10}	0	0
10	S_5	S'_7	S'_{10}	0	0
11	S_6	S'_7	S'_7	0	0
not (011 or 101)	S'_7	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0

State Reduction

Row Matching Method

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S'_7	S'_7	0	0
01	S_4	S'_7	S'_{10}	0	0
10	S_5	S'_7	S'_{10}	0	0
11	S_6	S'_7	S'_7	0	0
not (011 or 101)	S'_7	S_0	S_0	0	0
011 or 101	S'_{10}	S_0	S_0	1	0

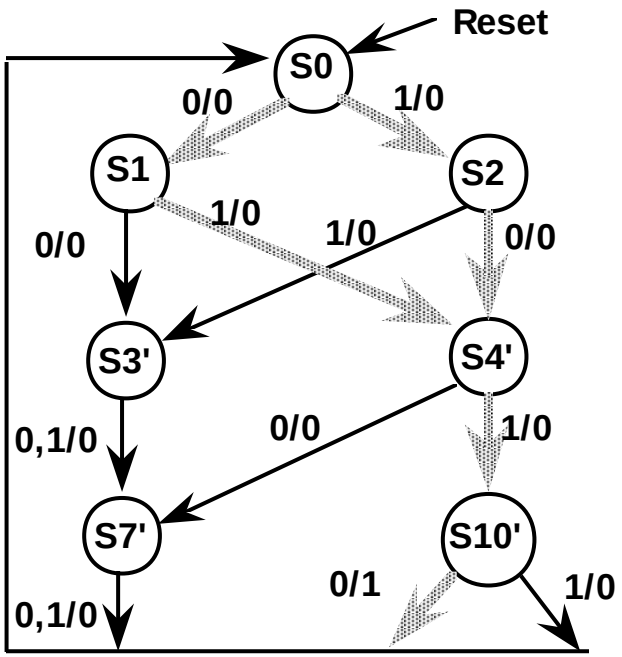
State Reduction

Row Matching Method

Final Reduced State Transition Table

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3'	S4'	0	0
1	S2	S4'	S3'	0	0
00 or 11	S3'	S7'	S7'	0	0
01 or 10	S4'	S7'	S10'	0	0
not (011 or 101)	S7'	S0	S0	0	0
011 or 101	S10'	S0	S0	1	0

Corresponding State Diagram



State Reduction

Row Matching Method

- **Straightforward to understand and easy to implement**
- **Problem: does not allows yield the most reduced state table!**

Example: 3 State Odd Parity Checker

Present State	Next State X=0	Next State X=1	Output
S_0	S_0	S_1	0
S_1	S_1	S_2	1
S_2	S_2	S_1	0

No way to combine states S0 and S2 based on Next State Criterion!

State Reduction

Implication Chart Method

New example FSM:

Single input X, Single output Z

Output a 1 whenever the serial sequence 010 or 110 has been observed at the inputs

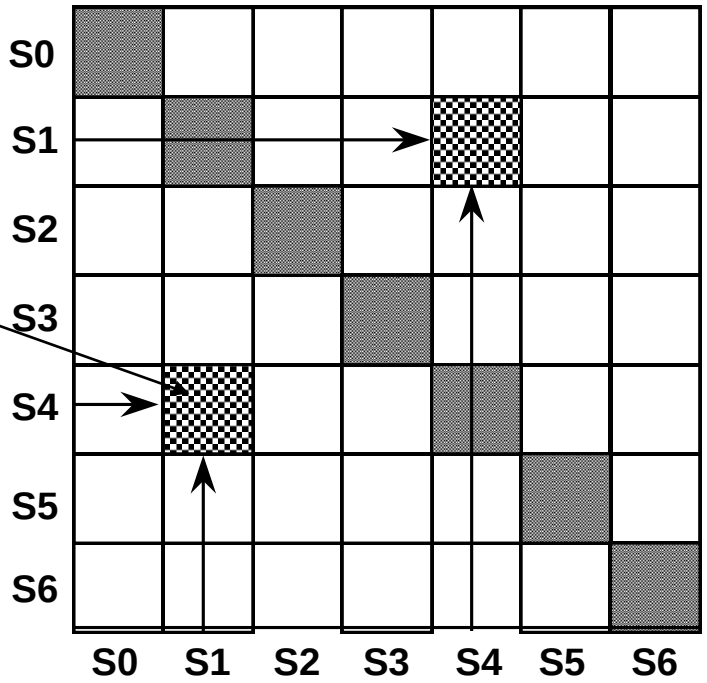
State transition table:

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_0	S_0	0	0
01	S_4	S_0	S_0	1	0
10	S_5	S_0	S_0	0	0
11	S_6	S_0	S_0	1	0

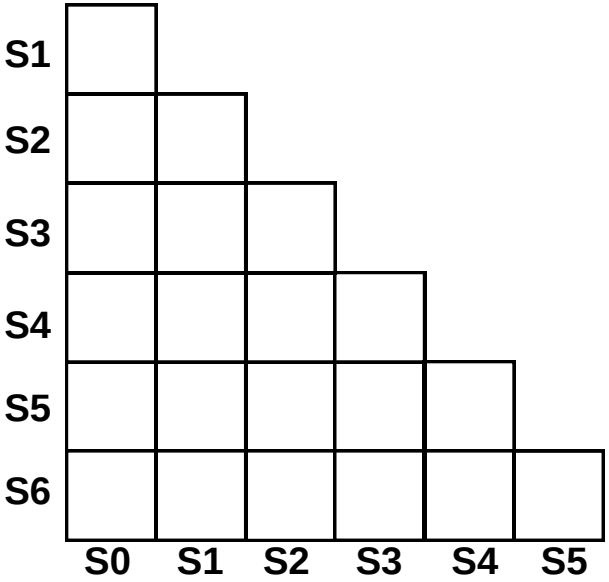
State Reduction

Implication Chart Method

Enumerate all possible combinations of states taken two at a time



Next States Under all Input Combinations



Implication Chart

Naive Data Structure:
Xij will be the same as Xji
Also, can eliminate the diagonal

State Reduction

Implication Chart

Filling in the Implication Chart

Entry Xij — Row is Si, Column is Sj

Si is equivalent to Sj if outputs are the same and
next states are equivalent

Xij contains the next states of Si, Sj which must be equivalent if
Si and Sj are equivalent

If Si, Sj have different output behavior, then Xij is crossed out

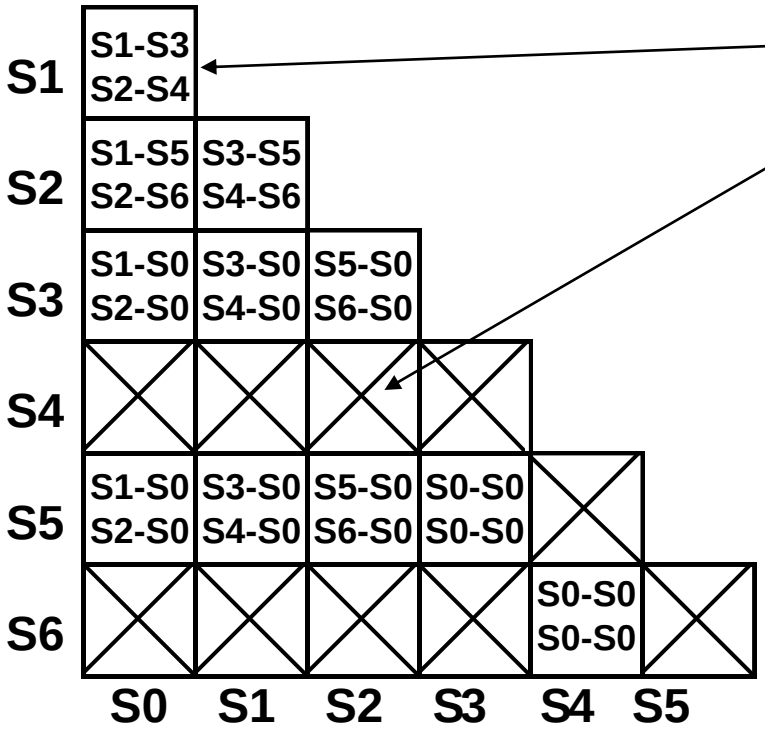
Example:
S0 transitions to S1 on 0, S2 on 1;
S1 transitions to S3 on 0, S4 on 1;

So square X<0,1> contains entries S1-S3 (transition on zero)
S2-S4 (transition on one)

S0	S1-S3 S2-S4
	S1

State Reduction

Implication Chart Method



Starting Implication Chart

S2 and S4 have different I/O behavior

This implies that S1 and S0 cannot be combined

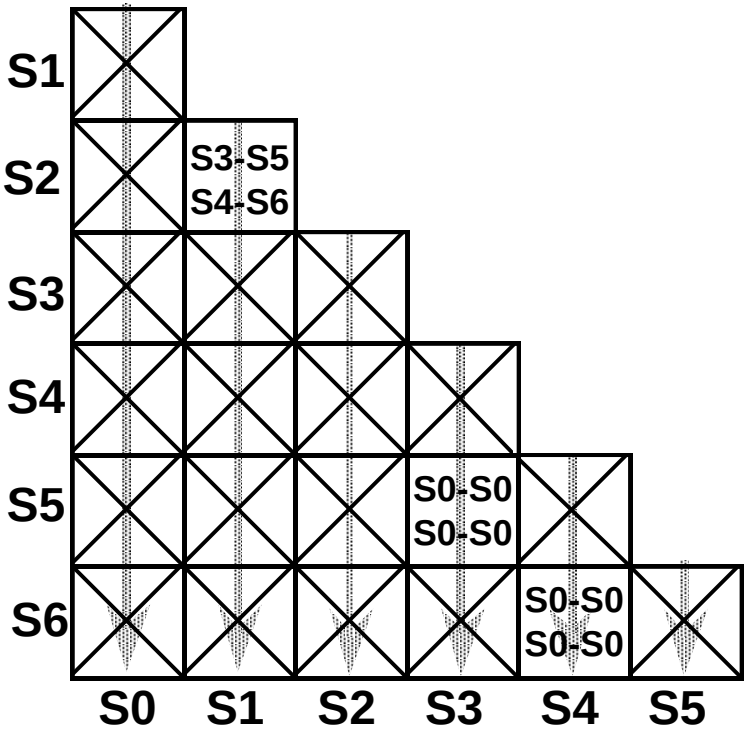
State Reduction

Implication Chart Method

Results of First Marking Pass

Second Pass Adds No New Information

- **S3 and S5 are equivalent**
- **S4 and S6 are equivalent**
- **This implies that S1 and S2 are too!**

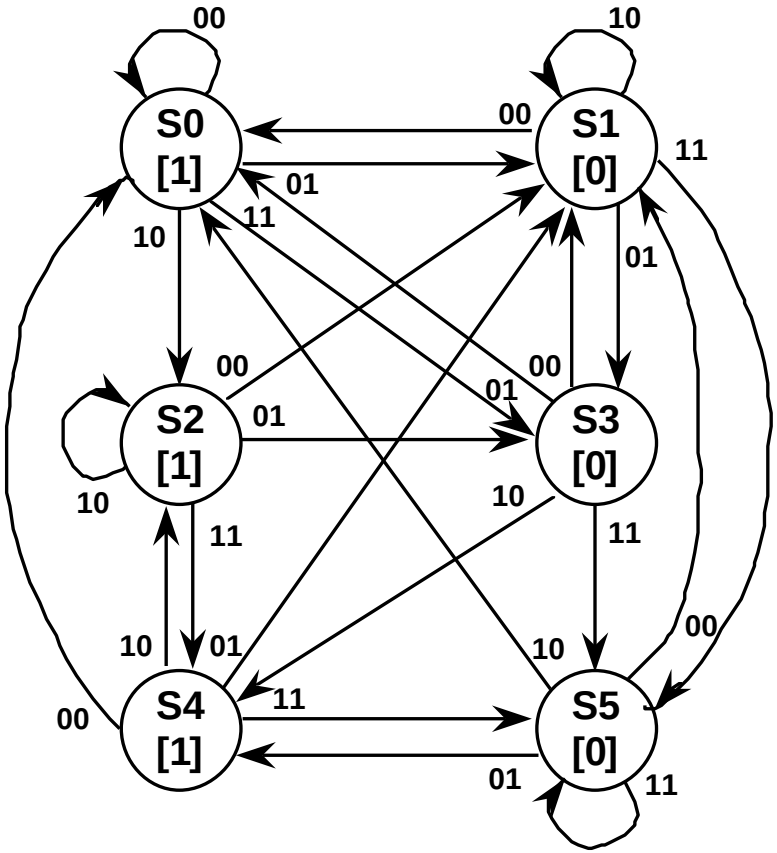


Reduced State Transition Table

Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S_0	S'_1	S'_1	0	0
0 or 1	S'_1	S'_3	S'_4	0	0
00 or 10	S'_3	S_0	S_0	0	0
01 or 11	S'_4	S_0	S_0	1	0

State Reduction

Multiple Input State Diagram Example



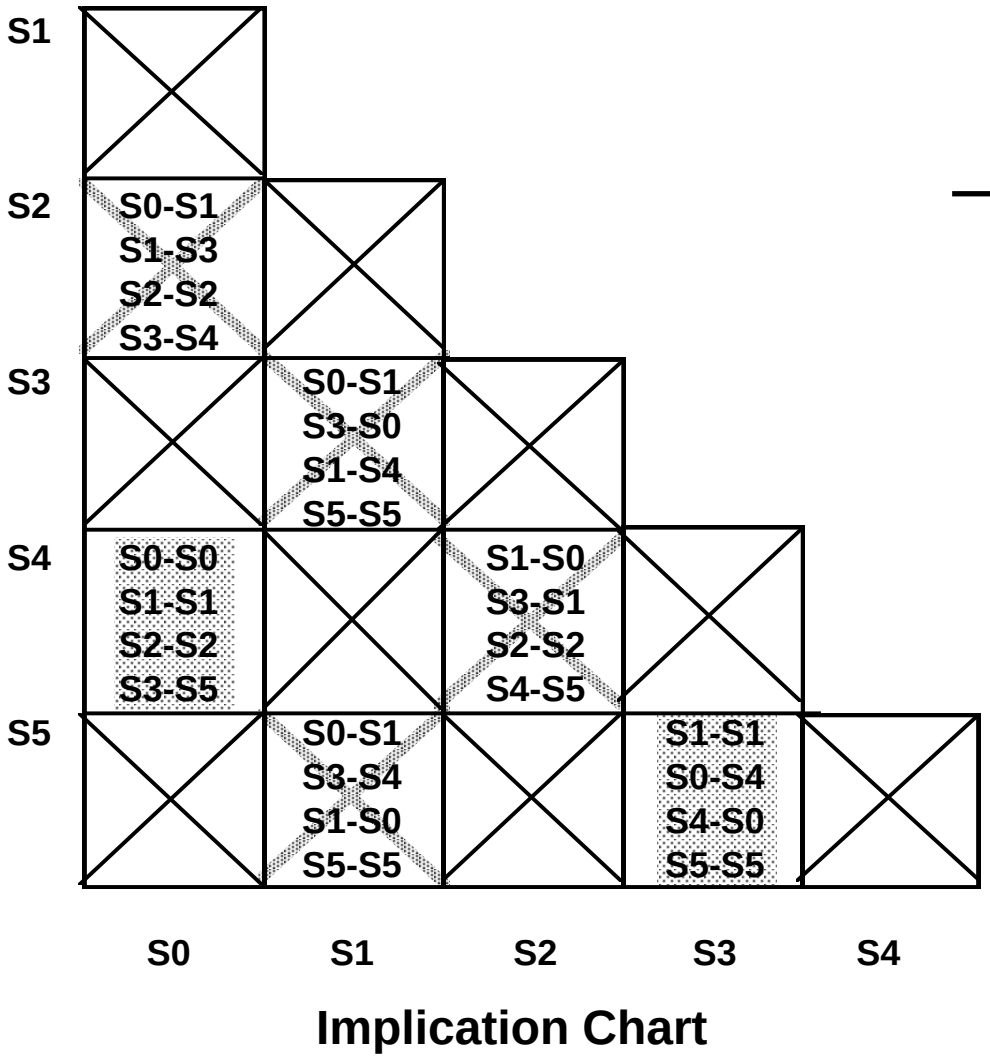
State Diagram

Present State	Next State 00	01	10	11	Output
S_0	S_0	S_1	S_2	S_3	1
S_1	S_0	S_3	S_1	S_5	0
S_2	S_1	S_3	S_2	S_4	1
S_3	S_1	S_0	S_4	S_5	0
S_4	S_0	S_1	S_2	S_5	1
S_5	S_1	S_4	S_0	S_5	0

Symbolic State Diagram

State Reduction

Multiple Input Example



Implication Chart

Present State	Next State 00	01	10	11	Output
S'_0	S_0'	S_1	S_2	S_3'	1
S_1	S_0'	S_3'	S_1	S_3'	0
S_2	S_1	S_3'	S_2	S_0'	1
S'_3	S_1	S'_0	S'_0	S'_3	0

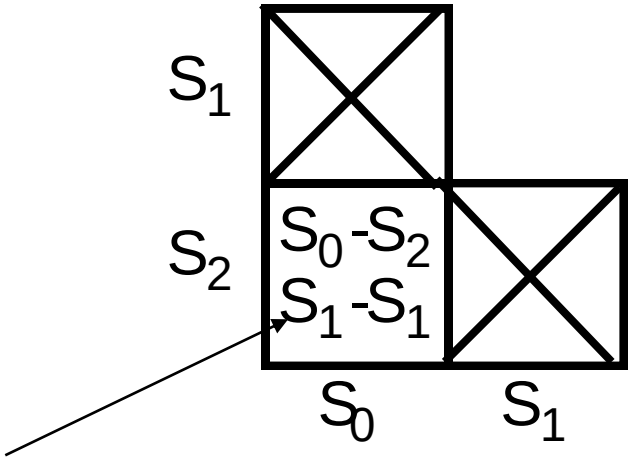
Minimized State Table

State Reduction

Implication Chart Method

Does the method solve the problem with the odd parity checker?

Implication Chart



S0 is equivalent to S2
since nothing contradicts this assertion!

State Reduction

Implication Chart Method

The detailed algorithm:

1. **Construct implication chart, one square for each combination of states taken two at a time**

2. **Square labeled Si, Sj, if outputs differ than square gets "X". Otherwise write down implied state pairs for all input combinations**

3. **Advance through chart top-to-bottom and left-to-right. If square Si, Sj contains next state pair Sm, Sn and that pair labels a square already labeled "X", then Si, Sj is labeled "X".**

4. **Continue executing Step 3 until no new squares are marked with "X".**

5. **For each remaining unmarked square Si, Sj, then Si and Sj are equivalent.**

State Assignment

When FSM implemented with gate logic, number of gates will depend on mapping between symbolic state names and binary encodings

4 states = 4 choices for first state, 3 for second, 2 for third, 1 for last
= 24 different encodings (4!)

Example for State Assignment: Traffic Light Controller

HG	HY	FG	FY
00	01	10	11
00	01	11	10
00	10	01	11
00	10	11	01
00	11	01	10
00	11	10	01
01	00	10	11
01	00	11	10
01	10	00	11
01	10	11	00
01	11	00	10
01	11	10	00

HG	HY	FG	FY
10	00	01	11
10	00	11	01
10	01	00	11
10	01	11	00
10	11	00	01
10	11	01	00
11	00	01	10
11	00	10	01
11	01	00	10
11	01	10	00
11	10	00	01
11	10	01	00

Inputs			Present State	Next State	Outputs		
C	TL	TS	Q_1 Q_0	P_1 P_0	ST	H_1 H_0	F_1 F_0
0	X	X	HG	HG	0	00	10
X	0	X	HG	HG	0	00	10
1	1	X	HG	HY	1	00	10
X	X	0	HY	HY	0	01	10
X	X	1	HY	FG	1	01	10
1	0	X	FG	FG	0	10	00
0	X	X	FG	FY	1	10	00
X	1	X	FG	FY	1	10	00
X	X	0	FY	FY	0	10	01
X	X	1	FY	HG	1	10	01

24 state assignments for the traffic light controller

Symbolic State Names: HG, HY, FG, FY

State Assignment

Some Sample Assignments for the Traffic Light Controller

Espresso input format:

```
.i 5
.o 7
.ilb c tl ts q1 q0
.ob p1 p0 st h1 h0 f1 f0
.p 10
0--   HG  HG 00010
-0-   HG  HG 00010
11-  HG  HY 10010
--0   HY  HY 00110
--1   HY  FG 10110
10-  FG  FG  01000
0--   FG  FY 11000
-1-   FG  FY 11000
--0   HY  FY  01001
--1   HY  HG 11001
.e
```

Two assignments: HG = 00, HY = 01, FG = 11, FY = 10
HG = 00, HY = 10, FG = 01, FY = 11

State Assignment

Random Assignments Evaluated with Espresso

```
.i 5
.o 7
.ilb c tl ts q1 q0
.ob p1 p0 st h1 h0 f1 f0
.p 9
11-00 0110000
10-11 1101000
--101  1010000
--010  1001001
---01   0100100
--110  0011001
---0-   0000010
0--11  1011000
-1-11  1011000
.e
```

26 literals
9 unique terms
several 5 and 4 input gates
13 gates total

```
.i 5
.o 7
.ilb c tl ts q1 q0
.ob p1 p0 st h1 h0 f1 f0
.p 8
11-0- 1010000
--010 1000100
0--01 1010000
--110 0110100
--111 0011001
----0  0000010
---01  0101000
--011 1101001
.e
```

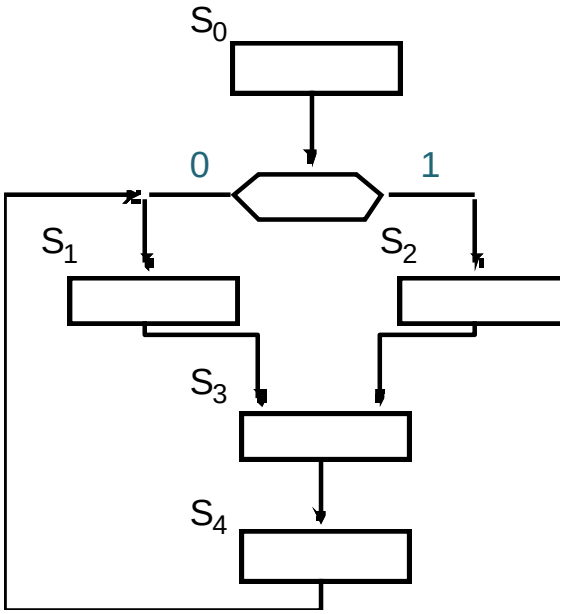
21 literals
8 unique terms
no 5 input gates, 2 4 input gates
14 gates total

State Assignment

Pencil & Paper Heuristic Methods

State Maps: similar in concept to K-maps

If state X transitions to state Y, then assign "close" assignments to X and Y

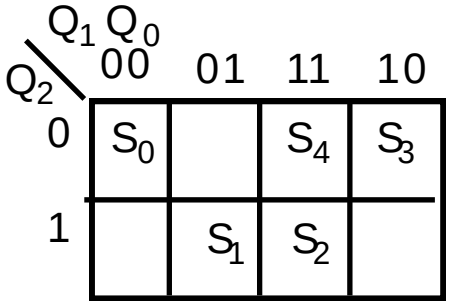


State Name	Assignment Q_2	Q_1	Q_0
S_0	0	0	0
S_1	1	0	1
S_2	1	1	1
S_3	0	1	0
S_4	0	1	1

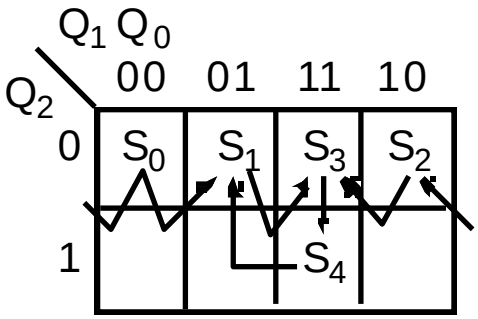
Assignment

State Name	Assignment Q_2	Q_1	Q_0
S_0	0	0	0
S_1	0	0	1
S_2	0	1	0
S_3	0	1	1
S_4	1	1	1

Assignment



State Map



State Map

State Assignment

Paper and Pencil Methods

Minimum Bit Distance Criterion

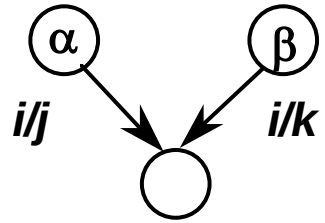
Transition	*First Assignment Bit Changes*	*Second Assignment Bit Changes*
S0 to S1:	2	1
S0 to S2:	3	1
S1 to S3:	3	1
S2 to S3:	2	1
S3 to S4:	1	1
S4 to S1:	2	2
	13	7

Traffic light controller: HG = 00, HY = 01, FG = 11, FY = 10
yields minimum distance encoding but not best assignment!

State Assignment

Paper & Pencil Methods

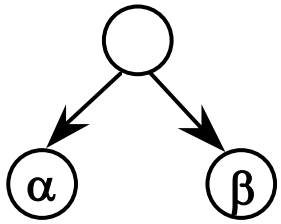
Alternative heuristics based on input and output behavior as well as transitions:



Highest Priority

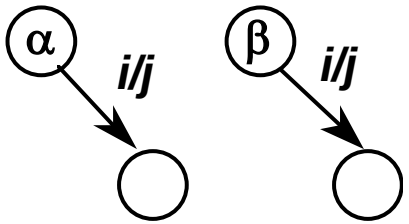
Adjacent assignments to:

states that share a common next state (group 1's in next state map)



Medium Priority

states that share a common ancestor state (group 1's in next state map)



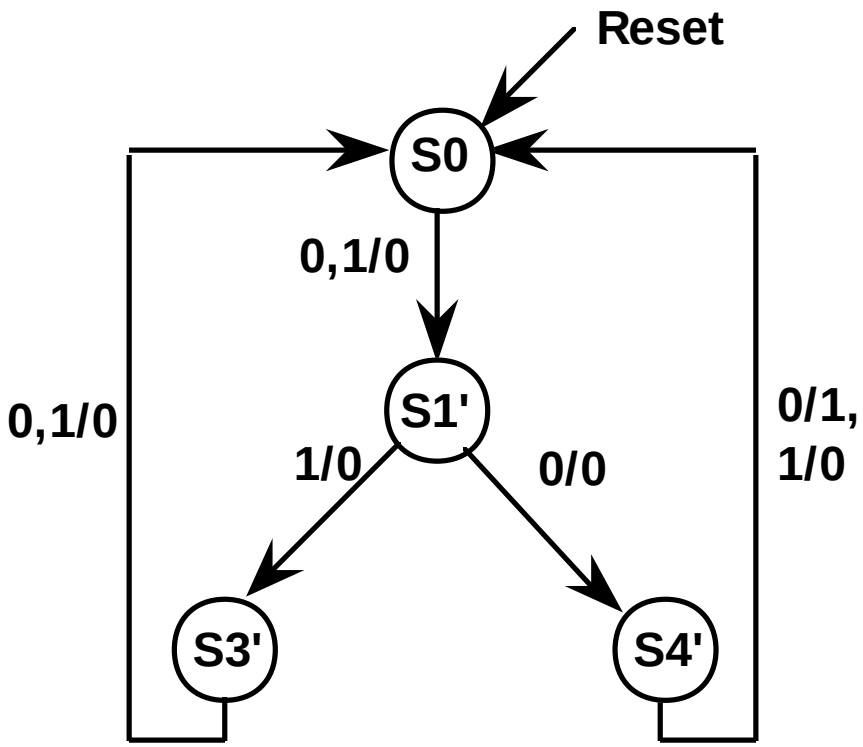
Lowest Priority

states that have common output behavior (group 1's in output map)

State Assignment

Pencil and Paper Methods

Example: 3-bit Sequence Detector



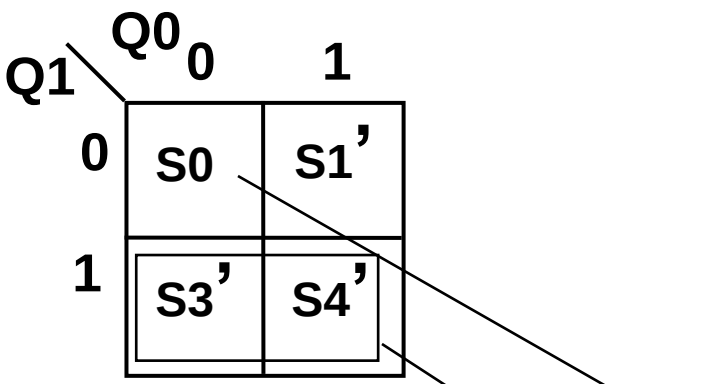
Highest Priority: (S3', S4')

Medium Priority: (S3', S4')

Lowest Priority:
- **0/0: (S0, S1', S3')**
- **1/0: (S0, S1', S3', S4')**

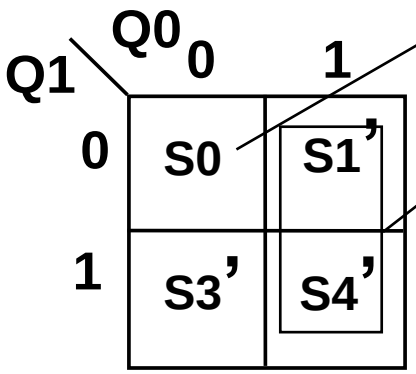
State Assignment

Paper and Pencil Methods



Reset State = 00

Highest Priority Adjacency

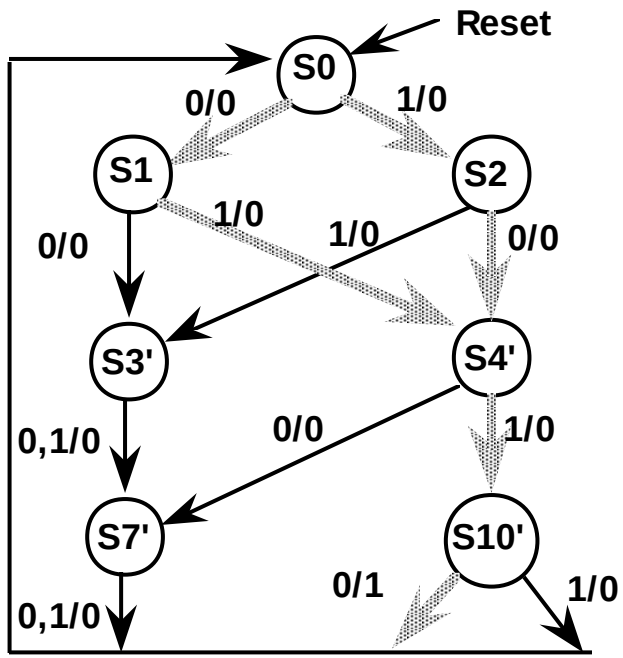


Not much difference in these two assignments

State Assignment

Paper & Pencil Methods

Another Example: 4 bit String Recognizer



Highest Priority: (S3', S4'), (S7', S10')

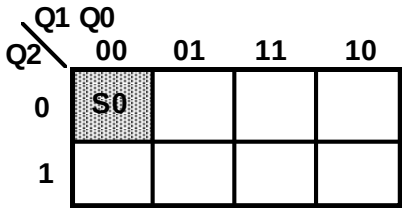
Medium Priority:
(S1, S2), 2x(S3', S4'), (S7', S10')

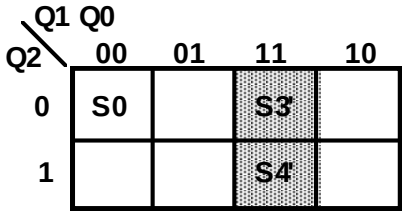
Lowest Priority:
0/0: (S0, S1, S2, S3', S4', S7')
1/0: (S0, S1, S2, S3', S4', S7')

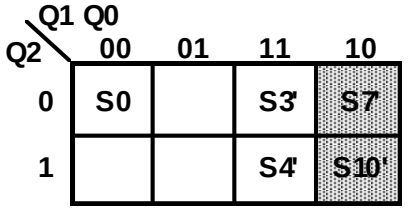
State Assignment

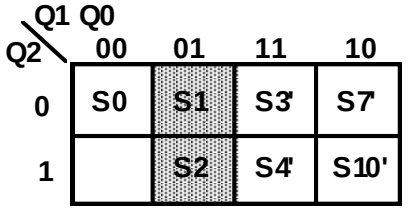
Paper & Pencil Methods

State Map

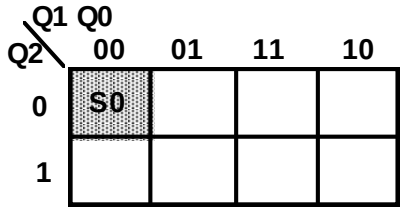


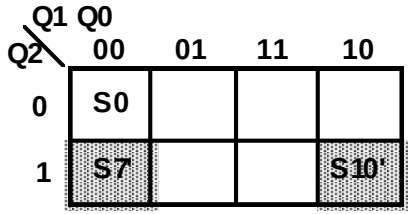


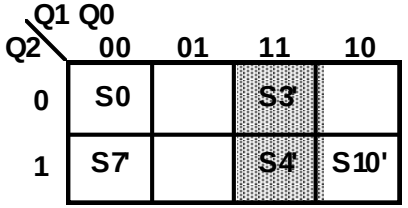


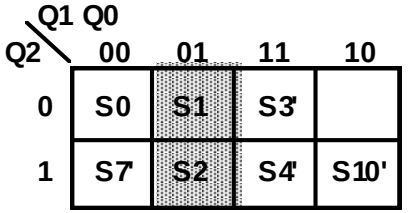


(a)









(b)

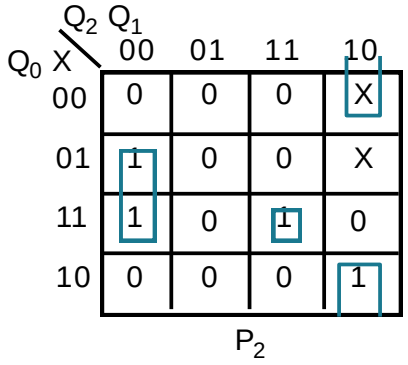
00 = Reset = S0

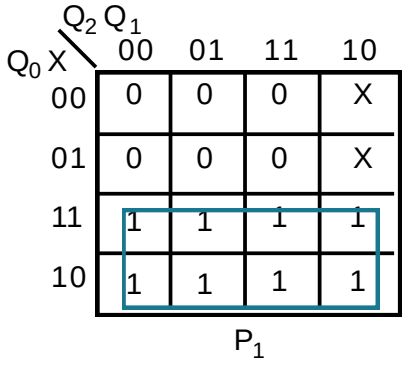
(S1, S2), (S3', S4'), (S7', S10') placed adjacently

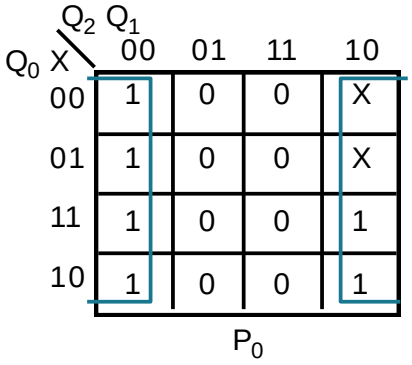
State Assignment

Effect of Adjacencies on Next State Map

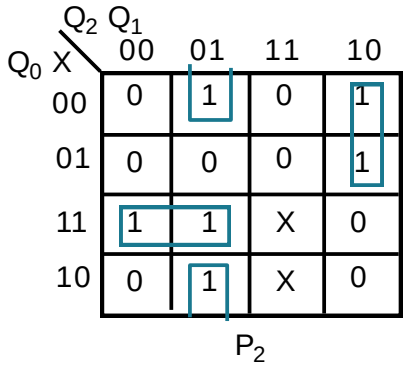
Current State	Next State X = 0	X = 1
(S_0) 000	001	101
(S_1) 001	011	111
(S_2) 101	111	011
(S'_3) 011	010	010
(S'_4) 111	010	110
(S'_7) 010	000	000
(S'_{10}) 110	000	000

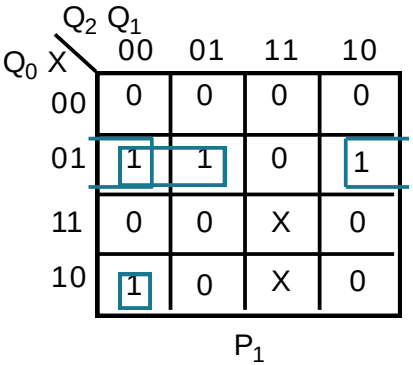


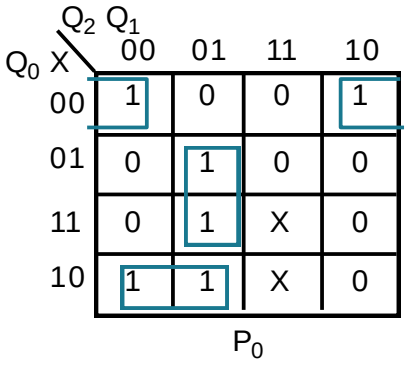




Current State	Next State X = 0	X = 1
(S_0) 000	001	010
(S_1) 001	011	100
(S_2) 010	100	011
(S'_3) 011	101	101
(S'_4) 100	101	110
(S'_7) 101	000	000
(S'_{10}) 110	000	000







First encoding exhibits a better clustering of 1's in the next state map

State Assignment

One Hot Encodings

n states encoded in n flipflops

HG = 0001
HY = 0010
FG = 0100
FY = 1000

Complex logic for discrete gate implementation

```
.i 7
.o 9
.ilb c tl ts q3 q2 q1 q0
.ob p3 p2 p1 p0 st h1 h0 f1 f0
.p 10
0--0001 0001 00010
-0-0001 0001 00010
11-0001 0010 10010
--0 0010 0010 00110
--1 0010 0100 10110
10-0100 0100 01000
0--0100 1000 11000
-1-0100 1000 11000
--0 0010 1000 01001
--1 0010 0001 11001
.e
```

Espresso Inputs

```
.i 7
.o 9
.ilb c tl ts q3 q2 q1 q0
.ob p3 p2 p1 p0 st h1 h0 f1 f0
.p 8
10-0100 010001000
11-0001 001010010
-0-0001 000100010
0--0001 000100010
0--0100 100011000
-1-0100 100011000
--00010 101001111
--10010 010111111
.e
```

Espresso Outputs

State Assignment

Computer-Based Methods

NOVA: State Assignment for 2-Level Circuit Networks

Input Constraints: states with same i/o mapped to same next state

Output Constraints: states which are successors of same predecessor

NOVA Input File for the Traffic Light Controller

inputs	*current_state*	*next_state*	*outputs*
0--	HG	HG	00010
-0-	HG	HG	00010
11-	HG	HY	10010
--0	HY	HY	00110
--1	HY	FG	10110
10-	FG	FG	01000
0--	FG	FY	11000
-1-	FG	FY	11000
--0	FY	FY	01001
--1	FY	HG	11001

```
nova -e <encoding strategy> -t <nova input file>
```

State Assignment

Computer-Based Methods

Greedy: satisfy as many input constraints as possible

Hybrid: satisfy input constraints, more sophisticated improvement strategy

I/O Hybrid: satisfy both input and output constraints

Exact: satisfy ALL input conditions

Input Annealing: like hybrid, but uses an even improvement strategy

1-Hot: uses a 1-hot encoding

Random: uses a randomly generated encoding

State Assignment

Computer-Based Methods

	HG	HY	FG	FY	# of Product Terms
Greedy (-e ig):	00	11	01	10	9
Hybrid (-e ih):	00	11	10	01	9
Exact (-e ie):	11	10	01	00	10
IO Hybrid (-e ioh):	00	01	11	10	9
I Annealing (-e ia):	01	10	11	10	9
Random (-e r):	11	00	01	10	9

-z HG on the command line forces NOVA to assign 00 to state HG

State Assignment

Computer Based Methods

More NOVA Examples:

3-bit Recognizer

```
-S0  S1 0
0  S1  S3 0
1  S1  S4 0
-S3  S0 0
0  S4  S0 1
1  S4  S0 0
```

	S0	S1'	S3'	S4'	terms
Greedy:	00	01	11	10	4
Hybrid:	00	01	10	11	4
Exact:	00	01	10	11	4
IO Hyb:	00	10	01	11	4
I Ann:	00	01	11	10	4
Random:	00	11	10	01	4

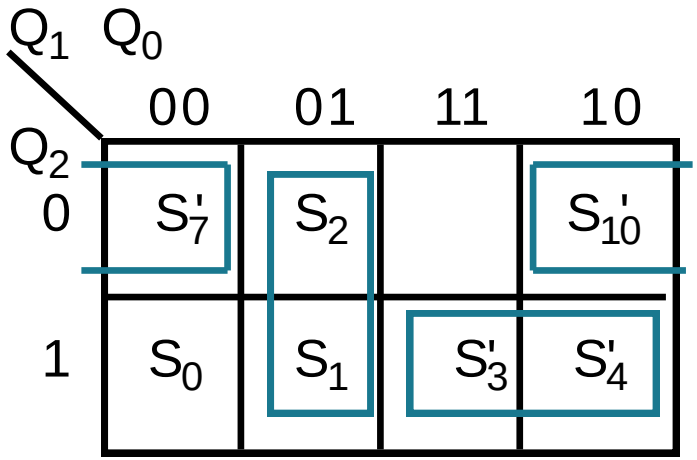
State Assignment

Computer Based Methods

More NOVA Examples:

4-bit Recognizer

```
0   S0   S1 0
1   S0   S2 0
0   S1   S3 0
1   S1   S4 0
0   S2   S4 0
1   S2   S3 0
- S3   S7 0
0   S4   S7 0
1   S4   S7 0
- S7   S0 0
0   S10  S0 1
1   S10  S0 0
```



Same adjacencies as the heuristic method

	S0	S1	S2	S3'	S4'	S7'	S10'	terms
Greedy:	100	110	010	011	111	000	001	7
Hybrid:	101	110	111	001	011	000	010	7
Exact:	101	110	111	001	011	000	010	7
IO Hyb:	110	011	001	100	101	000	010	7
I Ann:	100	101	001	111	110	000	010	6
Rand:	011	100	101	110	111	000	001	7

State Assignment

Mustang

Oriented towards multi-level logic implementation

Goal is to reduce literal count, rather than product terms

Input Format:

```
.i 3                                  # inputs
.o 5                                  # outputs
.s 2                                  # of state bits
0--   HG HG   00010
-0-   HG HG   00010
11-   HG HY   10010
--0   HY HY   00110
--1   HY FG   10110
10-   FG FG   01000
0--   FG FY   11000
-1-   FG FY   11000
--0   FY FY   01001
--1   FY HG   11001
```

State Assignment

Mustang

Goal: maximum common subexpressions in next state function

Encoding Strategies:

Random

Sequential

One-Hot

Fan-in: **states with common predecessors given adjacent assignment**

Fan-out: **state with common next state and outputs given adjacent assignments**

State Assignment

Mustang

Traffic Light:

	HG	HY	FG	FY	# product terms
Random:	01	10	11	00	9
Sequential:	01	10	11	00	9
Fan-in:	00	01	10	11	8
Fan-out:	10	11	00	01	8

3 Bit String Recognizer:

	S0	S1	S3'	S4'	# product terms
Random:	01	10	11	00	5
Sequential:	01	10	11	00	5
Fan-in:	10	11	00	01	4
Fan-out:	10	11	00	01	4

4 Bit String Recognizer:

	S0	S1	S2	S3'	S4'	S7'	S10'	# product terms
Random:	101	010	011	110	111	001	000	8
Sequential:	001	010	011	100	101	110	111	8
Fan-in:	100	010	011	000	001	101	110	8
Fan-out:	110	010	011	100	101	000	001	6

To obtain multilevel implementations, must run misII

State Assignment

JEDI

Input Format:

# of states; encoding bit length

Kinds of states

Kinds of outputs

```
.i 4
.o 4
.enum States 4 2 HG HY FG FY
.enum Colors 3 2 GREEN RED YELLOW
0--HG HG 0 GREEN RED
-0-HG HG 0 GREEN RED
11-HG HY 1 GREEN RED
--0 HY HY 0 YELLOW RED
--1 HY FG 1 YELLOW RED
10-FG FG 0 RED GREEN
0--FG FY 1 RED GREEN
-1-FG FY 1 RED GREEN
--0 FY FY 0 RED YELLOW
--1 FY HG 1 RED YELLOW
```

General encoding problems: best encodings for states and outputs

Encoding Strategies:

- **Random**
- **Straightforward**
- **One Hot**
- **Input Dominant**
- **Output Dominant**
- **Modified Output Dominate**
- **I/O Combination**

State Assignment

JEDI

Traffic Light:

	HG	HY	FG	FY	Grn	Yel	Red	# product terms
Input:	00	10	11	01	11	01	00	9
Output:	00	01	11	10	10	11	01	9
Comb:	00	10	11	01	10	00	01	9
Output':	01	00	10	11	10	00	01	10

3 Bit String Recognizer:

	S0	S1	S3'	S4'	# product terms
Input:	01	10	11	00	4
Output:	01	10	11	00	4
Comb.:	10	11	00	01	4
Output':	10	11	00	01	4

4 Bit String Recognizer:

	S0	S1	S2	S3'	S4'	S7'	S10'	# product terms
Input:	111	101	100	010	110	011	001	7
Output:	101	110	100	010	000	111	011	7
Comb.:	100	011	111	110	010	000	101	7
Output':	001	100	101	010	011	000	111	8

To obtain multilevel implementations, must run misII

State Assignment

Mustang vs. Jedi

Traffic Light Controller

Mustang

$$Q1 = HL0 \cdot TS + FL0 \cdot TS' + FL0' \cdot P1$$
$$Q0 = Q1 \cdot C' \cdot P1 + C \cdot TL \cdot P0' + TS' \cdot P0$$
$$ST = Q0 \cdot P0' + Q0' \cdot P0$$
$$HL1 = FL0 + P1 \cdot P0'$$
$$HL0 = P1' \cdot P0$$
$$FL1 = P1'$$
$$FL0 = HL0' \cdot P0$$

Fewer wires

Jedi

$$Q1 = HL1 \cdot C \cdot TL + HL0 + FL1 \cdot C \cdot TL'$$
$$Q0 = HL0 \cdot TS + FL1 + FL0 \cdot TS'$$
$$ST = Q1 \cdot HL1 + Q1' \cdot FL0 + HL1' \cdot FL1' \cdot TS$$
$$HL1 = P1' \cdot P0'$$
$$HL0 = P1 \cdot P0'$$
$$FL1 = HL0' \cdot P1$$
$$FL0 = HL1' \cdot P1'$$

Fewer literals

Choice of Flipflops

J-K FFs: reduce gate count, increase # of connections

D FFs: simplify implementation process, decrease # of connections

Procedure:

1. **Given state assignments, derive the next state maps from the state transition table**
2. **Remap the next state maps given excitation tables for a given FF**
3. **Minimize the remapped next state function**

Choice of Flipflops

Examples

4 bit Sequence Detector using NOVA derived state assignment

Encoded State Transition Table

Present State	Next State I=0	Next State I=1	Output I=0	Output I=1
000 (S_0)	011 (S_1)	010 (S_2)	0	0
011 (S_1)	101 (S'_3)	111 (S'_4)	0	0
010 (S_2)	111 (S'_4)	101 (S'_3)	0	0
101 (S'_3)	100	100 (S'_7)	0	0
111 (S'_4)	(S'_7)	110	0	0
100 (S'_7)	100	(S'_{10})	0	0
110 (S'_{10})	(S'_7)	000	1	0

Encoded Next State Map

Present State	Next State I=0	Next State I=1
000	011	010
001	XXX	XXX
010	111	101
011	101	111
100	000	000
101	100	100
110	000	000
111	100	110

Choice of Flipflops

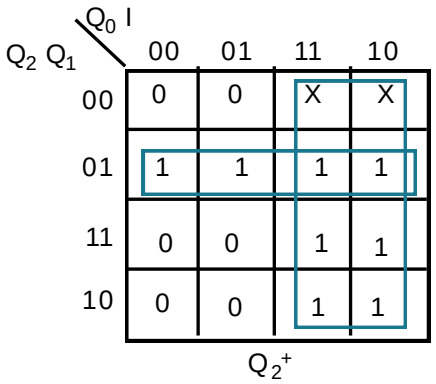
D FF Implementation

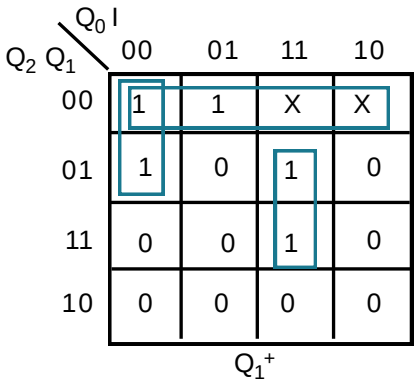
$$D_{Q2+} = \overline{Q2} \bullet Q1 + Q0$$

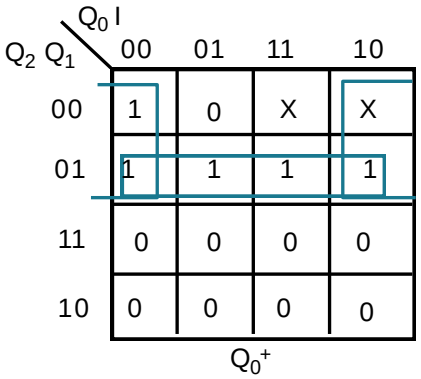
$$D_{Q1+} = Q1 \bullet Q0 \bullet I + \overline{Q2} \bullet \overline{Q0 \bullet I} + Q2 \bullet Q1$$

$$D_{Q0+} = \overline{Q2} \bullet Q1 + \overline{Q2 \bullet I}\ \overline{\ }$$

6 product terms
15 literals







Choice of Flipflops

J-K Implementation

	Next State		Remapped Next State			
			J	K	J	K
Present State	I=0	I=1	I=0	I=0	I=1	I=1
000	011	010	011	XXX	010	XXX
001	XXX	XXX	XXX	XXX	XXX	XXX
010	111	101	1X1	X0X	1X1	X1X
011	101	111	1XX	X10	1XX	X00
100	000	000	X00	1XX	X00	1XX
101	100	100	X0X	0X1	X0X	0X1
110	000	000	XX0	11X	XX0	11X
111	100	110	XXX	011	XXX	001

Remapped Next State Table

Choice of Flipflops

J-K Implementation (continued)

$$J_{Q2+} = Q1$$

$$K_{Q2+} = \overline{Q0}$$

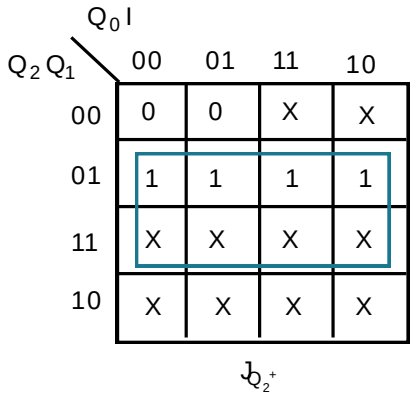
$$J_{Q1+} = \overline{Q2}$$

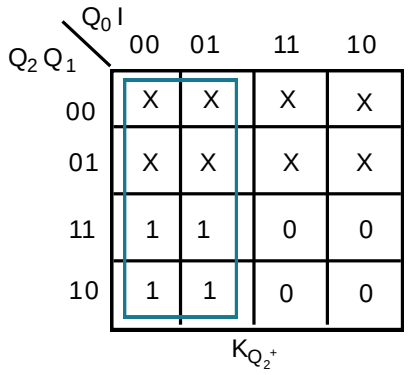
$$K_{Q1+} = \overline{Q0 \bullet I} + \overline{Q0 \bullet I} + \overline{Q2 \bullet Q0}$$

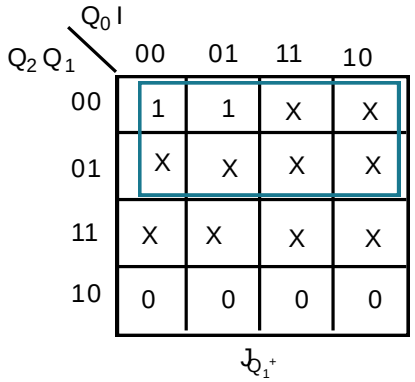
$$J_{Q0+} = \overline{Q2 \bullet Q1} + \overline{Q2 \bullet I}$$

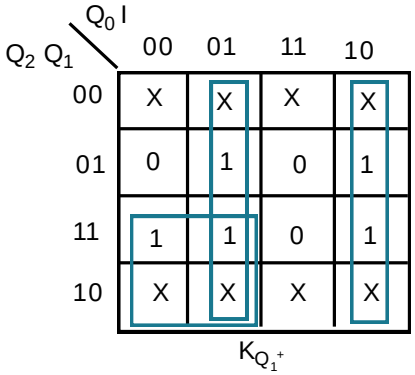
$$K_{Q0+} = Q2$$

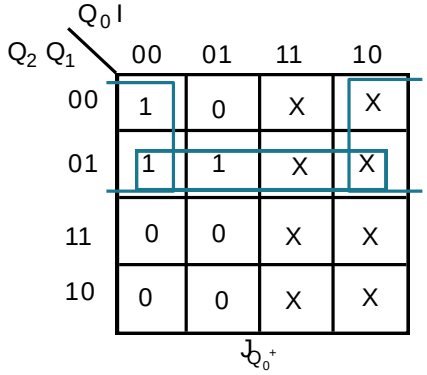
9 unique terms
14 literals

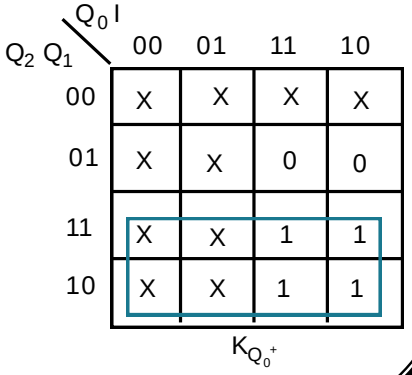










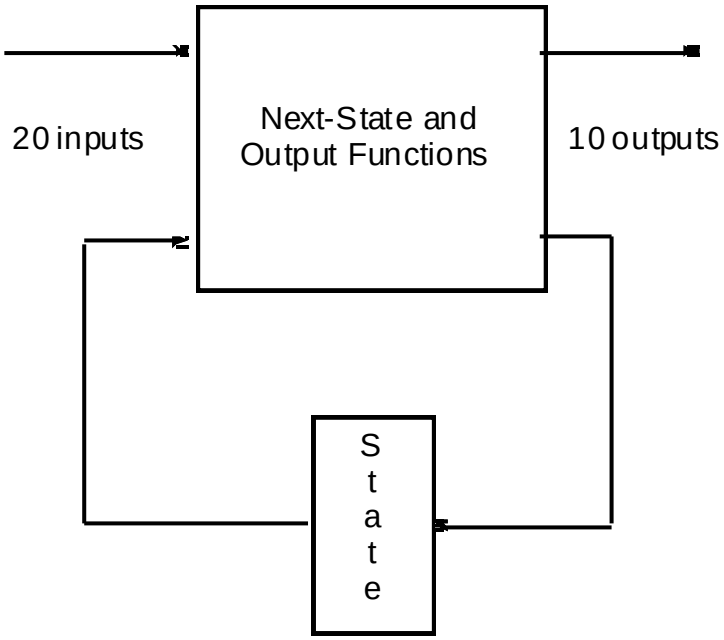


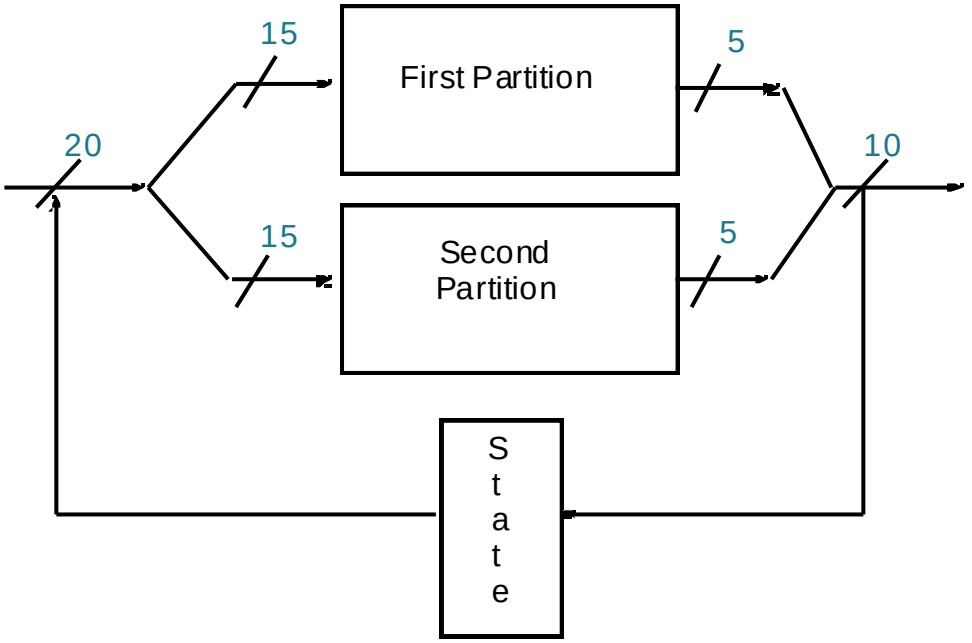
Finite State Machine Partitioning

Why Partition?

mapping FSMs onto programmable logic components:

- **limited number of input/output pins**
- **limited number of product terms or other programmable resources**





Example of Input/Output Partitioning:

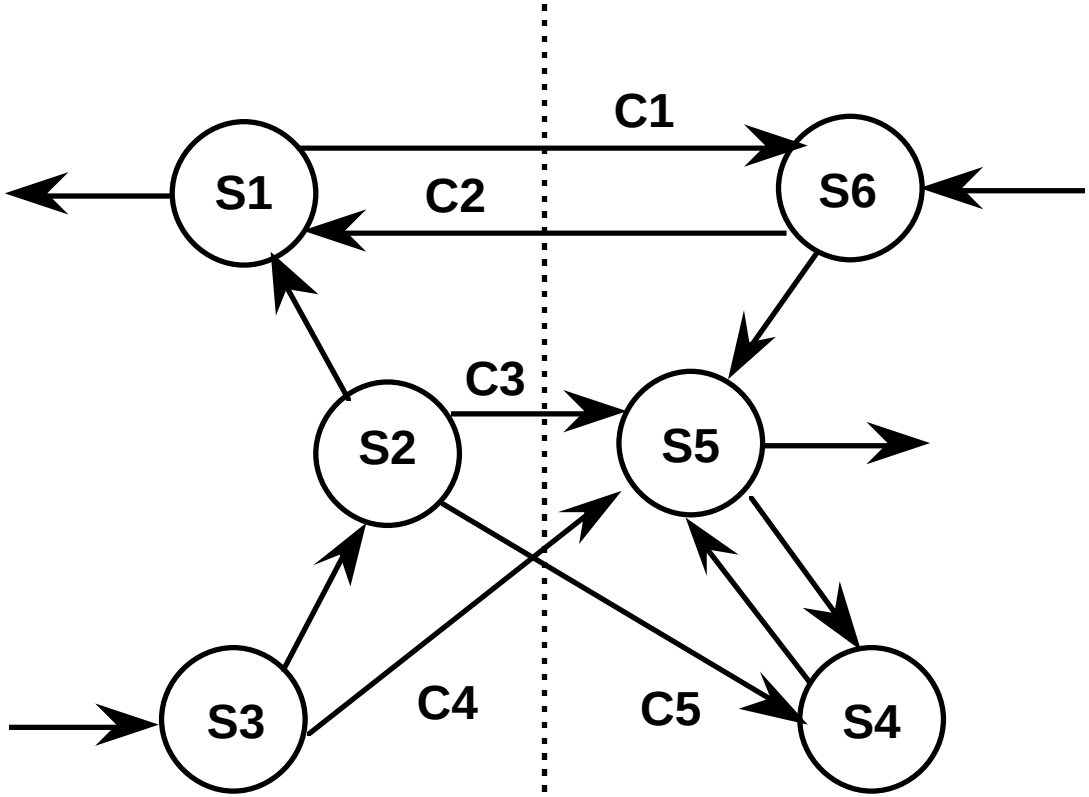
5 outputs depend on 15 inputs

5 outputs depend on different overlapping set of 15 inputs

Finite State Machine Partitioning

Introduction of Idle States

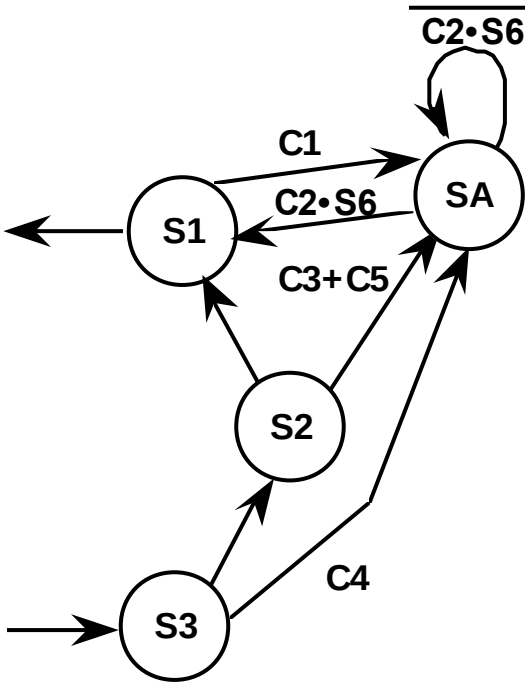
Before Partitioning

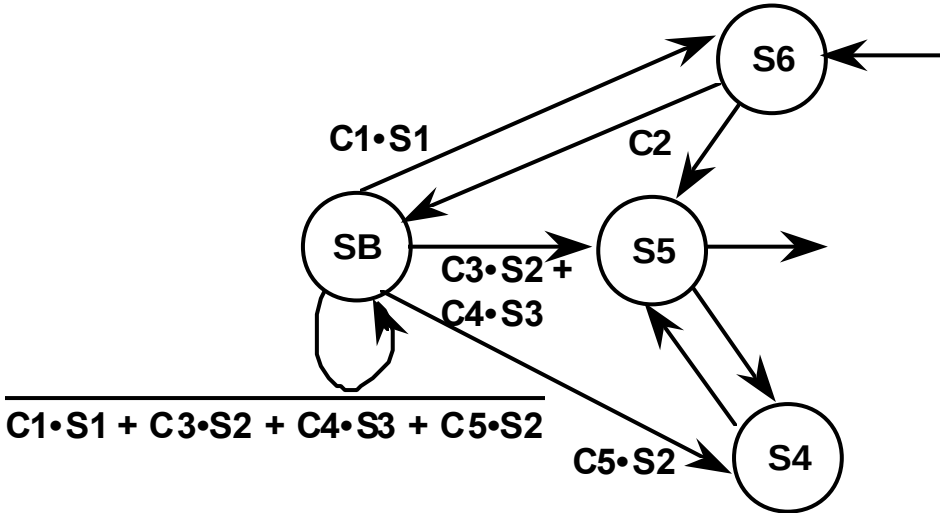


Finite State Machine Partitioning

Introduction of Idle States

After Partitioning

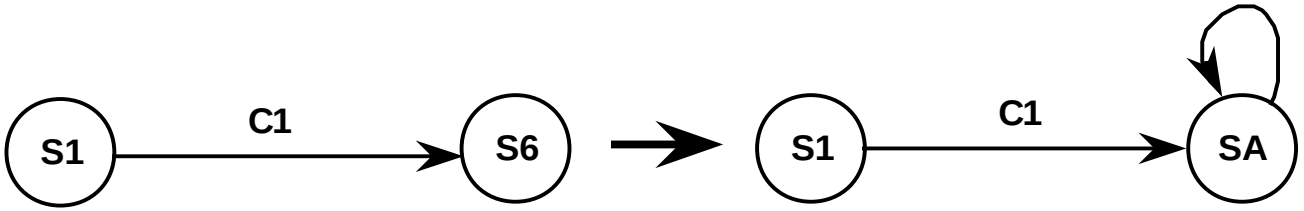




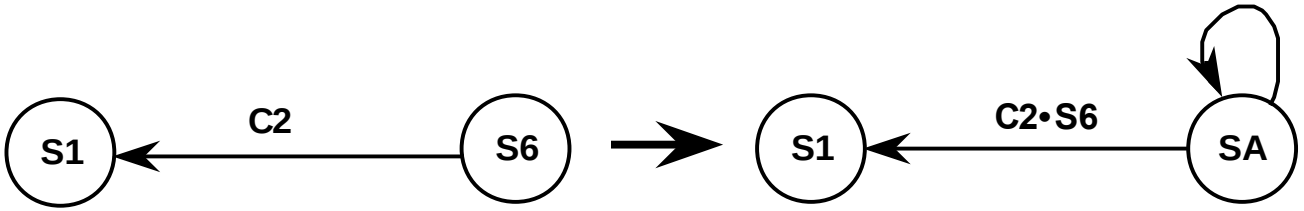
Finite State Machine Partitioning

Rules for Partitioning

Rule #1: Source State Transformation; SA is the Idle State



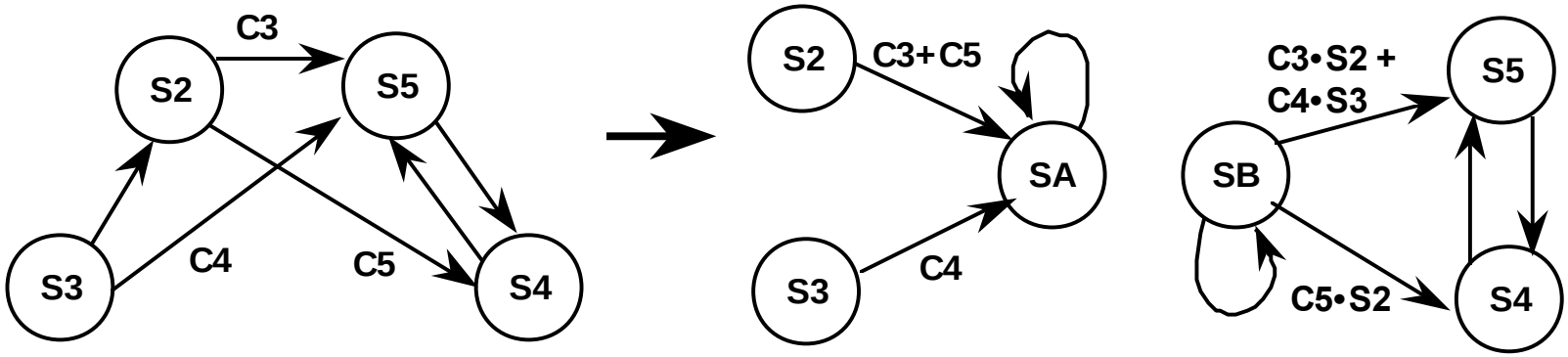
Rule #2: Destination State Transformation



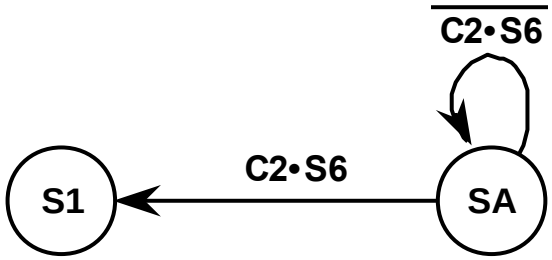
Finite State Machine Partitioning

Rules for Partitioning

Rule #3: Multiple Transitions with Same Source or Destination

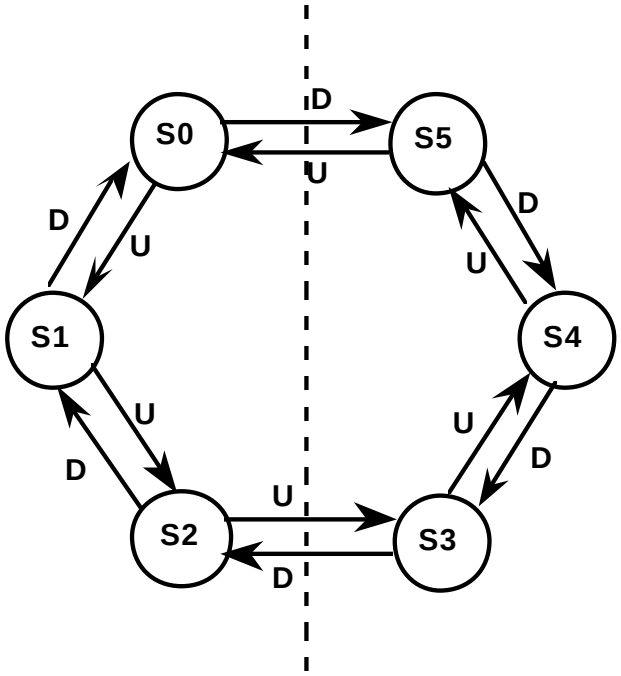


Rule #4: Hold Condition for Idle State



Finite State Machine Partitioning

Another Example

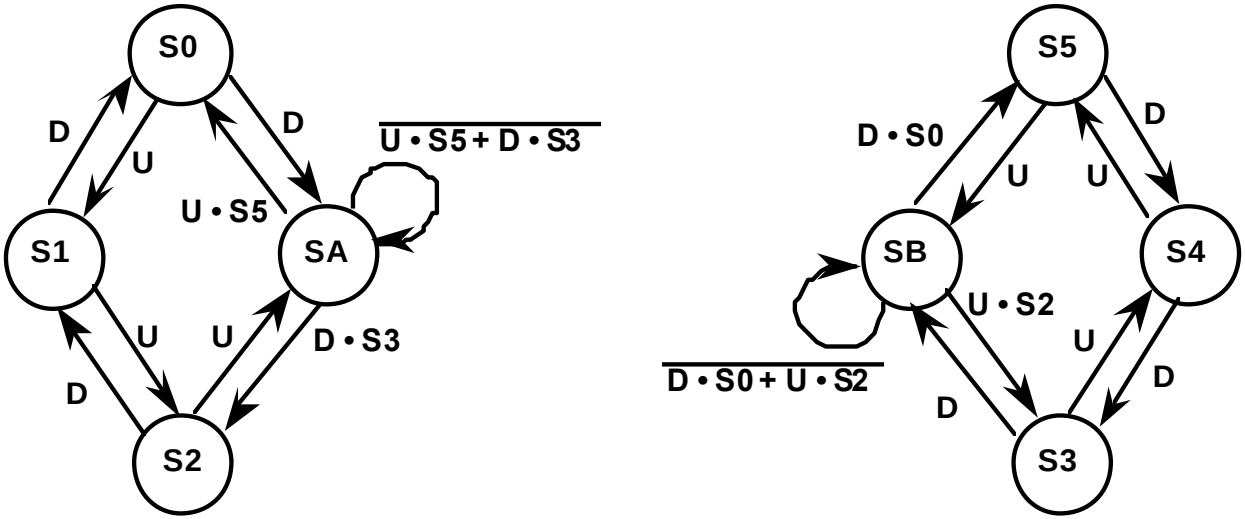


6 state up/down counter

building block has 2 FFs + combinational logic

Finite State Machine Partitioning

6 State Up/Down Counter



Introduction of the two idle state SA, SB

Count sequence S0, S1, S2, S3, S4, S5:
S2 goes to SA and holds, leaves after S5
S5 goes to SB and holds, leaves after S2

Down sequence is similar

Chapter Summary

- ***Optimization of FSM***

  State Reduction Methods: Row Matching, Implication Chart

  State Assignment Methods: Heuristics and Computer Tools

- ***Implementation Issues***

  Choice of Flipflops

  Finite State Machine Partitioning