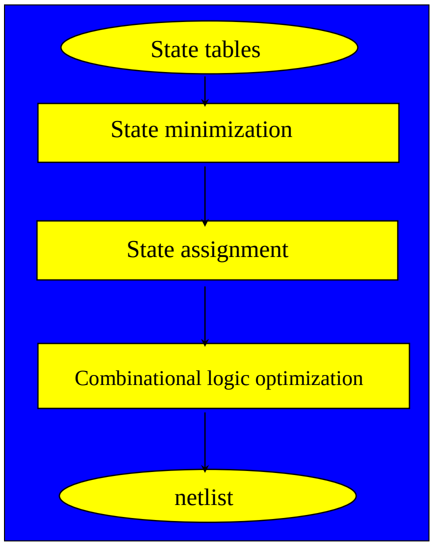
CSE140: Components and Design Techniques for Digital Systems

Tajana Simunic Rosing

Announcements

- HW#5 due, HW#6 assigned – due on MONDAY!!!!
- Midterm on Tuesday, 11/7 at class time
  - Everything up to and including chap 8, app. A,B,C
- Today:
  - FSM optimization
    - State minimization using Implicant method
    - State assignment
    - FSM partitioning

FSM Optimization Flow Chart



identify and remove equivalent states

assign unique binary code to each state

use unassigned state-codes as don’t care

Successive partitioning algorithm for state minimization

- Goal
  - identify and combine states that have equivalent behavior
- Algorithm sketch
  1. place all states in one set
  2. initially partition set based on the output behavior
  3. successively partition the resulting subsets based on next state transitions
  4. repeat (3) until no further partitioning is possible
     - states left in the same set are equivalent
- Polynomial time procedure

Method of successive partitions

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	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0

Minimized FSM

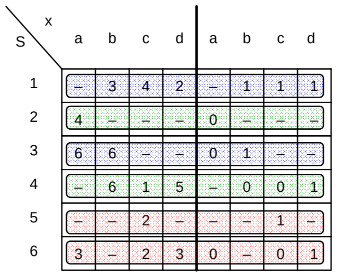
Input Sequence	Present State	Next State X=0	Next State X=1	Output X=0	Output X=1
Reset	S0	S1	S2	0	0
0	S1	S3	S4	0	0
1	S2	S5	S6	0	0
00	S3	S0	S0	0	0
01	S4	S0	S0	1	0
10	S5	S0	S0	0	0
11	S6	S0	S0	1	0

(S0) (S1 S2) (S3 S5) (S4 S6)

Implication chart method: Basic Concepts

Compatiblity:

Si, Sj are compatible if for each input they have consistent outputs, and their successors are the same or compatible.



S \ x	a	b	c	d	a	b	c	d
1	–	3	4	2	–	1	1	1
2	4	–	–	–	0	–	–	–
3	6	6	–	–	0	1	–	–
4	–	6	1	5	–	0	0	1
5	–	–	2	–	–	–	1	–
6	3	–	2	3	0	–	0	1

Conditionally compatible :

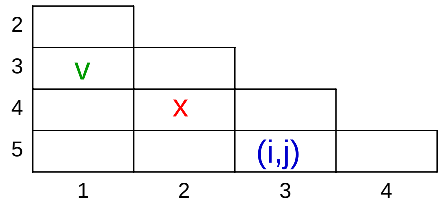
S_i, S_j are conditionally compatible if their outputs and next states are consistent for some pairs of successors

$(S_i, S_j) \neq (S_k, S_l)$

7

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Implication chart method: Triangular table definition



We fill the cells of triangular table as follows:

v – if pair of states is compatible,

x – if pair of states in incompatible,

(i,j) – pair (pair of successors), if the pair is conditionally compatible.

8

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Triangular table – example

	a	b	c	d	a	b	c	d
1	–	3	4	2	–	1	1	1
2	4	–	–	–	0	–	–	–
3	6	6	–	–	0	1	–	–
4	–	6	1	5	–	0	0	1
5	–	–	2	–	–	–	1	–
6	3	–	2	3	0	–	0	1

2					
3					
4					
5					
6					
	1	2	3	4	5

Triangular table - example

To get compatible states iteratively cross out all incompatibles

2	∨				
3	3,6	4,6			
4	×	∨	×		
5	2,4	∨	∨	×	
6	×	3,4	∨	1,2; 3,5	×
	1	2	3	4	5

Calculating Maximal classes of Compatibility

Compatible pairs: (1,2); (1,3); (1,5); (2,3); (2,4); (2,5); (3,5); (3,6); (4,6)

```
 1,2
 1,3
 1,5
----
 2,3
 2,4
 2,5
----
 3,5
 3,6
----
 4,6
```

11

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Minimization Algorithm

1) Find all **pairs of compatible states,**

2) Calculate **maximal sets of compatible states (MCC),**

3) Select sets that satisfy the so-called **Covering condition (a) and closure condition (b):**

   a) Each state must be in at least one class;

   b) For each input symbol all next states of each class must be included into one class.

12

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Covering Condition - example

	a	b	c	d	a	b	c	d
1	–	3	4	2	–	1	1	1
2	4	–	–	–	0	–	–	–
3	6	6	–	–	0	1	–	–
4	–	6	1	5	–	0	0	1
5	–	–	2	–	–	–	1	–
6	3	–	2	3	0	–	0	1

MCC = {{1,2,3,5}, {3,6}, { 2,4}, {4,6}}

Closure condition - example

	a	b	c	d	a	b	c	d
1	–	3	4	2	–	1	1	1
2	4	–	–	–	0	–	–	–
3	6	6	–	–	0	1	–	–
4	–	6	1	5	–	0	0	1
5	–	–	2	–	–	–	1	–
6	3	–	2	3	0	–	0	1

For selected classes {1,2,3,5},{4,6}} we calculate their successors

Condition of covering and closure – second try

MCC = {{1,2,3,5}, {3,6}, { 2,4}, {4,6}}

	a	b	c	d	a	b	c	d
1	–	3	4	2	–	1	1	1
2	4	–	–	–	0	–	–	–
3	6	6	–	–	0	1	–	–
4	–	6	1	5	–	0	0	1
5	–	–	2	–	–	–	1	–
6	3	–	2	3	0	–	0	1

Another example

	0	1	0	1
1	2	6	0	0
2	3	1	1	1
3	–	4	–	0
4	–	5	–	0
5	3	–	1	–
6	7	–	1	–
7	–	8	–	0
8	–	–	–	1

2							
3							
4							
5							
6							
7							
8							
	1	2	3	4	5	6	7

Maximal compatibility class

2	✗						
3	46	✗					
4	~~56~~	✗	45				
5	✗	v	v	v			
6	✗	~~37~~	v	v	~~37~~		
7	68	✗	~~48~~	58	v	v	
8	✗	v	✗	✗	v	v	✗
	1	2	3	4	5	6	7

Compatibles: 1,3
1,7
2,5
2,8
3,4
3,5
3,6
4,5
4,6
4,7
5,7
5,8
6,7
6,8

MCC:

Testing successor states

	0	1	0	1
1	2	6	0	0
2	3	1	1	1
3	–	4	–	0
4	–	5	–	0
5	3	–	1	–
6	7	–	1	–
7	–	8	–	0
8	–	–	–	1

MCC: 2,5,8
3,4,5
3,4,6
4,5,7
4,6,7
1,3
1,7
6,8

Table of successors

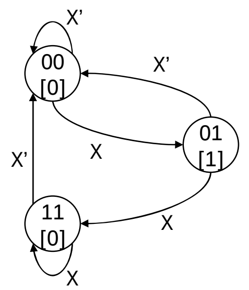
Covering and closure – final state selection

	0	1	0	1
1	2	6	0	0
2	3	1	1	1
3	–	4	–	0
4	–	5	–	0
5	3	–	1	–
6	7	–	1	–
7	–	8	–	0
8	–	–	–	1

	2,5,8	3,4,5	3,4,6	4,5,7	4,6,7	1,3	1,7	6,8
$\delta(0,S_i)$	3	3	7	3	7	2	2	7
$\delta(1,S_i)$	1	45	45	58	58	46	68	–

Minimizing states may not yield best circuit

- Example: edge detector - outputs 1 when last two input changes from 0 to 1



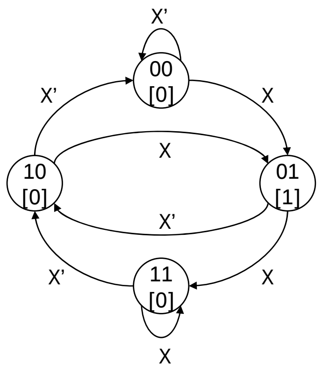
X	Q_1	Q_0	Q_1^+	Q_0^+
0	0	0	0	0
0	0	1	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	1	1	1
–	1	0	0	0

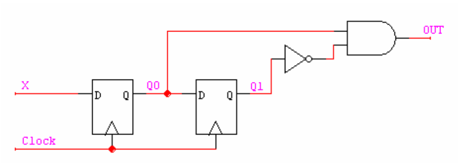
$Q_1^+ = X\ (Q_1 \text{ xor } Q_0)$

$Q_0^+ = X\ Q_1'\ Q_0'$

Edge detector – ad hoc solution

- "Ad hoc" solution - not minimal but cheap and fast

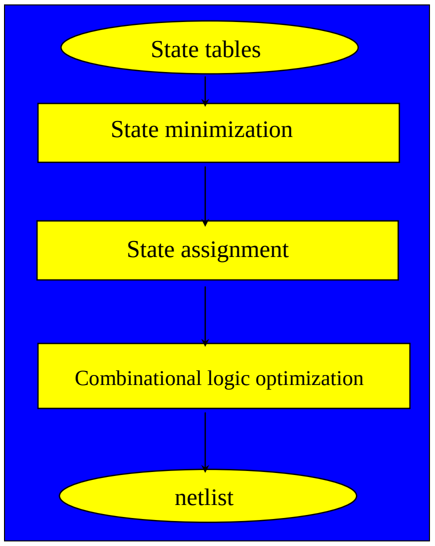




Minimizing incompletely specified FSMs

- Equivalence of states
  - transitive when machine is fully specified
  - not transitive when don't cares are present

FSM Optimization Flow Chart



identify and remove equivalent states

assign unique binary code to each state

use unassigned state-codes as don’t care

Sources: TSR, Katz, Boriello, Vahid, Perkowski

State assignment strategies

- Choose bit vectors to assign to each “symbolic” state
  - huge number even for small values of state bits and states
    - intractable for state machines of any size
    - heuristics are necessary for practical solutions – **no guarantee of optimality**
  - optimize some metric for the combinational logic
    - size (amount of logic and number of FFs)
    - speed (depth of logic and fanout)
    - dependencies (decomposition)
- Possible strategies
  - sequential – just number states as they appear in the state table
  - random – pick random codes
  - one-hot – use as many state bits as there are states
  - output – use outputs to help encode states
  - heuristic – rules of thumb that seem to work in most cases

24

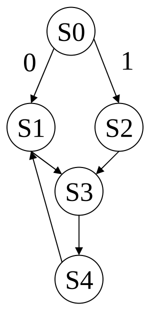
Sources: TSR, Katz, Boriello, Vahid, Perkowski

One-hot state assignment

- Simple
  - easy to encode
  - easy to debug
- Small logic functions
  - each state function requires only predecessor state bits as input
- Good for programmable devices
  - lots of flip-flops readily available
  - simple functions with small support (signals its dependent upon)
- Impractical for large machines
  - too many states require too many flip-flops
  - decompose FSMs into smaller pieces that can be one-hot encoded

Heuristics for state assignment

- Encode adjacent states to minimize # of state bit changes
  - Use state maps



	00	01	11	10
0	S0		S4	S3
1		S1	S2	

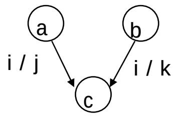
	# of bit changes	
Transition	1st case	2nd case
S0 - S1	-	-
S0 - S2	-	-
S1 - S3	-	-
S2 - S3	-	-
S3 - S4	-	-
S4 - S1	-	-
Total	-	-

Heuristics for state assignment

- Goal: maximize groupings of 1s in the next state & output functions
  - Helps minimize next state logic
- Guidelines:
  1. Adjacent codes to states that share a common next state

I	Q	Q+	O
i	a	c	j
i	b	c	k

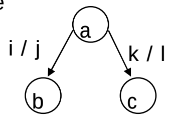
c = i * a + i * b



  2. Adjacent codes to states that share a common ancestor state

I	Q	Q+	O
i	a	b	j
k	a	c	l

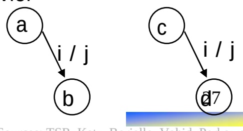
b = i * a
c = k * a



  3. Adjacent codes to states that have a common output behavior

I	Q	Q+	O
i	a	b	j
i	c	d	j

j = i * a + i * c
b = i * a
d = i * c



Output-based encoding

- Reuse outputs as state bits
  - why create new functions for state bits when output can serve as well
  - fits in nicely with synchronous Mealy implementations

Inputs			Present State	Next State	Outputs		
C	TL	TS			ST	H	F
0	–	–	HG	HG	0	00	10
–	0	–	HG	HG	0	00	10
1	1	–	HG	HY	1	00	10
–	–	0	HY	HY	0	01	10
–	–	1	HY	FG	1	01	10
1	0	–	FG	FG	0	10	00
0	–	–	FG	FY	1	10	00
–	1	–	FG	FY	1	10	00
–	–	0	FY	FY	0	10	01
–	–	1	FY	HG	1	10	01

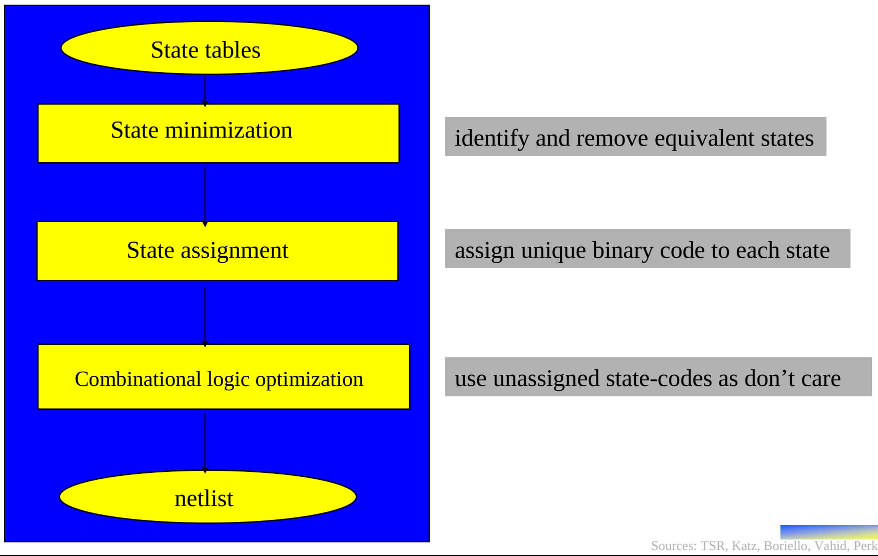
Current state assignment approaches

- For tight encodings using close to the minimum number of state bits
  - used in custom chip design
- One-hot encoding
  - easy for small state machines
  - generates small equations with easy to estimate complexity
  - common in FPGAs and other programmable logic
- Output-based encoding
  - ad hoc - no tools
  - most common approach taken by human designers
  - yields small circuits for most FSMs

29

Sources: TSR, Katz, Boriello, Vahid, Perkowski

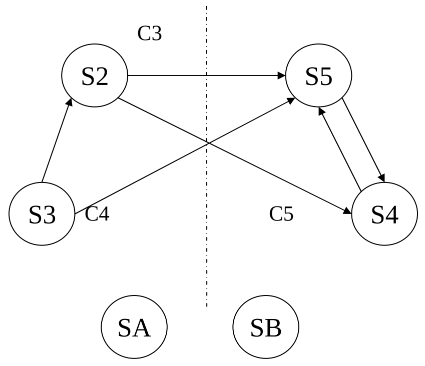
FSM Optimization Flow Chart


State tables
State minimization
State assignment
Combinational logic optimization
netlist
identify and remove equivalent states
assign unique binary code to each state
use unassigned state-codes as don't care

Sources: TSR, Katz, Boriello, Vahid, Perkowski

State Partitioning

- Helps for large state machines
  - E.g. when next state logic is too large to implement in a programmable logic component
- Introduce *idle* states to synchronize partitioned FSMs

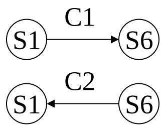


31

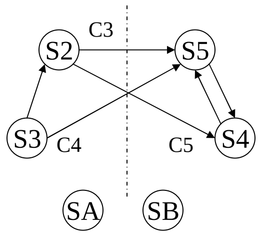
Sources: TSR, Katz, Boriello, Vahid, Perkowski

Partition rules

- Source/destination transformation



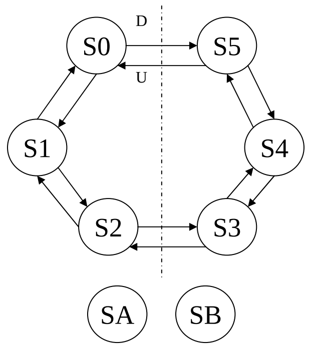
- Hold condition for the idle state
- Multiple transitions to same source/destination



32

Sources: TSR, Katz, Boriello, Vahid, Perkowski

State Partitioning Example



33

Sources: TSR, Katz, Boriello, Vahid, Perkowski

Summary of FSM Optimization

- State minimization
  - straightforward in fully-specified machines
  - computationally intractable, in general (with don’t cares)
- State assignment
  - many heuristics
  - best-of-10-random just as good or better for most machines
  - output encoding can be attractive (especially for PAL implementations)
- State partitioning
  - Used for larger state machines for ease of implementation
  - Introduce “idle” states at the interface
  - Change transition conditions according to the rules of partitioning

34

Sources: TSR, Katz, Boriello, Vahid, Perkowski