

ESE535: Electronic Design Automation

Day 12: March 2, 2009
Sequential Optimization
(FSM Encoding)



Penn ESE535 Spring 2009 -- DeHon

Today

- Encoding
 - Input
 - Output
- State Encoding
 - “exact” two-level

Penn ESE535 Spring 2009 -- DeHon

2

Input Encoding

- Pick codes for input cases to simplify logic
- *E.g.* Instruction Decoding
 - ADD, SUB, MUL, OR
- Have freedom in code assigned
- Pick code to minimize logic
 - *E.g.* number of product terms

Penn ESE535 Spring 2009 -- DeHon

3

Output Encoding

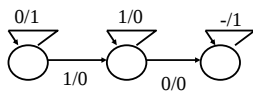
- Opposite problem
- Pick codes for output symbols
- *E.g.* allocation selection
 - Prefer N, Prefer S, Prefer E, Prefer W, No Preference
- Again, freedom in coding
- Use to maximize sharing
 - Common product terms, CSE

Penn ESE535 Spring 2009 -- DeHon

4

Finite-State Machine

- Logical behavior depends on state
- In response to inputs, may change state



Penn ESE535 Spring 2009 -- DeHon

5

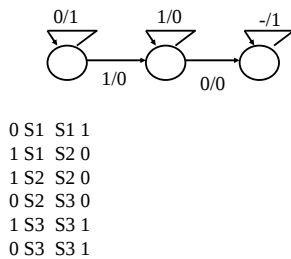
State Encoding

- State encoding is a logical entity
- No *a priori* reason any particular state has any particular encoding
- Use freedom to simply logic

Penn ESE535 Spring 2009 -- DeHon

6

Finite State Machine



Penn ESE535 Spring 2009 -- DeHon

7

Example: Encoding Difference

0 S1 S1 1	S1=01	0 01 01 1	0 01 01 1
1 S1 S2 0	S2=11	1 01 11 0	1 -1 11 0
1 S2 S2 0	S3=10	1 11 11 0	- 11 10 0
0 S2 S3 0		0 11 10 0	- 10 10 1
1 S3 S3 1	S1+S2 = -1	1 10 10 1	
0 S3 S3 1		0 10 10 1	

Similar outputs, code so S1+S2 is simple cube	S1=00	0 00 00 1	1 11 01 0
	S2=11	1 00 11 0	1 00 11 0
	S3=10	1 11 11 0	0 -0 00 1
		0 11 10 0	- 10 00 1
		1 10 10 1	- 1- 10 0
		0 10 10 1	

Penn ESE535 Spring 2009 -- DeHon

8

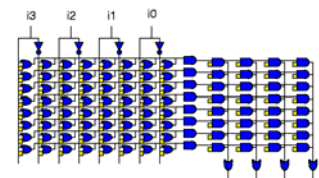
Problem:

- **Real:** pick state encodings (si's) so as to minimize the implementation area
 - two-level
 - multi-level
- Simplified variants
 - minimize product terms
 - achieving minimum product terms, minimize state size
 - minimize literals

Penn ESE535 Spring 2009 -- DeHon

9

Two-Level



- $A_{pla} = (2 \cdot \text{ins} + \text{outs}) \cdot \text{prods} + \text{flops} \cdot \text{wflop}$
- inputs = PIs + state_bits
- outputs = state_bits + POs
- products terms (prods)
 - depend on state-bit encoding
 - **this is where we have leverage**

Penn ESE535 Spring 2009 -- DeHon

10

Multilevel

- More sharing → less implementation area
- Pick encoding to increase sharing
 - maximize common sub expressions
 - maximize common cubes
- Effects of multi-level minimization hard to characterize (not predictable)

Penn ESE535 Spring 2009 -- DeHon

11

Two-Level Optimization

1. **Idea:** do **symbolic** minimization of two-level form
 - This represents effects of sharing
2. Generate encoding constraints from this
 - Properties code must have to maximize sharing
3. Cover
 - Like two-level (mostly...)
4. Select Codes

Penn ESE535 Spring 2009 -- DeHon

12

Kinds of Sharing

Input sharing:
encode inputs so
cover set to reduce
product terms

Output sharing:
share input cubes
to produce individual
output bits

	10 inp1+inp2=01			
10 inp1 01	11 inp2+inp3=01			Out1=11
01 inp1 10		1101 out1		Out2=01
1- inp2 01	10 1- 01	1100 out2		Out3=10
01 inp2 01	Inp1=10	1111 out3		Out4=00
11 inp3 01	Inp2=11	0000 out4		
01 inp3 10	Inp3=01	01 11 01	110- 01	
		01 01 10	11-1 10	

Penn ESE535 Spring 2009 -- DeHon

13

Input Encoding

Penn ESE535 Spring 2009 -- DeHon

14

Two-Level Input Oriented

- Minimize product rows
 - by exploiting common-cube
 - next-state expressions
- Does not account for possible sharing of terms to cover outputs

[DeMicheli+Brayton+SV/TR CAD v4n3p269]

Penn ESE535 Spring 2009 -- DeHon

15

Outline Two-Level Input

- Represent states as one-hot codes
- Minimize using two-level optimization
 - Include: combine compatible next states
 - 1 S1 S2 0
 - 1 S2 S2 0 → 1 {S1,S2} S2 0
- Get disjunct on states deriving next state
- Assuming no sharing due to outputs
 - gives minimum number of product terms
- Cover to achieve
 - Try to do so with minimum number of state bits

Penn ESE535 Spring 2009 -- DeHon

16

Multiple Valued Input Set

- Treat input states as a multi-valued (not just 0,1) input variable
- Effectively encode in **one-hot** form
 - One-hot: each state gets a bit, only one on
- Use to merge together input state sets

0 S1 S1 1	0 100 S1 1
1 S1 S2 0	1 100 S2 0
1 S2 S2 0	1 010 S2 0
0 S2 S3 0	0 010 S3 0
1 S3 S3 1	1 001 S3 1
0 S3 S3 1	0 001 S3 1

Penn ESE535 Spring 2009 -- DeHon

17

One-hot Minimum

- One-hot gives minimum number of product terms
- i.e. Can **always** maximally combine input sets into single product term

Penn ESE535 Spring 2009 -- DeHon

18

One-hot example

10 inp1 01
 01 inp1 10
 1- inp2 01
 01 inp2 10
 11 inp3 01
 01 inp3 10

One-hot:
 inp1=100
 inp2=010
 inp3=001

10 100 01
 01 100 10
 1- 010 01
 01 010 10
 11 001 01
 01 001 10
 -- 11- --
 -- 1-1 --
 -- -11 --
 -- 000 --

10 --0 01
 11 0-- 01
 01 --- 10

Key: can define a cube to cover any subset of states

10 --0 01 says $10*(inp1+inp2) \rightarrow 01$

Penn ESE535 Spring 2009 -- DeHon

19

State Combining

- Follows from standard 2-level optimization with don't-care minimization
- Effectively groups together common predecessor states as shown
- (can define to combine directly)

Penn ESE535 Spring 2009 -- DeHon

20

Example

0 S s6 00	0 1000000 0000010 00	
0 s2 s5 00	0 0100000 0000100 00	
0 s3 s5 00	0 0010000 0000100 00	
0 s4 s6 00	0 0001000 0000010 00	0 0110001 0000100 00
0 s5 S 10	0 0000100 1000000 10	0 1001000 0000010 00
0 s6 S 01	0 0000010 1000000 01	1 0001001 0000010 10
0 s7 s5 00	0 0000001 0000100 00	0 0000010 1000000 01
1 S s4 01	1 0000010 0100000 01	1 0000100 0100000 10
1 s2 s3 10	1 0000100 0100000 10	0 0000100 1000000 10
1 s3 s7 10	1 0001000 0000010 10	1 1000000 0001000 00
1 s4 s6 10	1 0000001 0000010 10	1 0000010 0100000 01
1 s5 s2 00	1 1000000 0001000 00	1 0100000 0010000 00
1 s6 s2 00	1 0100000 0010000 00	1 0010000 0000001 00
1 s7 s6 00	1 0010000 0000001 00	

Penn ESE535 Spring 2009 -- DeHon

21

Two-Level Input

- One-hot identifies multivalued minimum number of product terms
- May be less product terms if get sharing (don't cares) in generating the next state expressions
 - (was not part of optimization)
- Encoding places each disjunct on a unique cube face
 - Can distinguish with a single cube
- Can use fewer bits than one-hot
 - this part typically heuristic
 - Remember one-hot already minimized prod terms

Penn ESE535 Spring 2009 -- DeHon

22

Encoding Example

0 S s6 00		s 010
0 s2 s5 00		s2 110
0 s3 s5 00		s3 101
0 s4 s6 00	0 0110001 0000100 00	s4 000
0 s5 S 10	0 1001000 0000010 00	s5 001
0 s6 S 01	1 0001001 0000010 10	s6 011
0 s7 s5 00	0 0000010 1000000 01	s7 100
1 S s4 01	1 0000100 0100000 10	
1 s2 s3 10	0 0000100 1000000 10	
1 s3 s7 10	1 1000000 0001000 00	
1 s4 s6 10	1 0000010 0100000 01	s2+s3+s7=1--
1 s5 s2 00	1 0100000 0010000 00	No 111 code
1 s6 s2 00	1 0010000 0000001 00	
1 s7 s6 00		

Penn ESE535 Spring 2009 -- DeHon

23

Encoding Example

0 0110001 0000100 00	s 010	0 1-- 001 00
0 1001000 0000010 00	s2 110	0 0-0 011 00
1 0001001 0000010 10	s3 101	1 -00 011 10
0 0000010 1000000 01	s4 000	
1 0000100 0100000 10	s5 001	1 001 110 10
0 0000100 1000000 10	s6 011	0 001 010 10
1 1000000 0001000 00	s7 100	1 010 000 00
1 0000010 0100000 01		0 011 010 01
1 0100000 0010000 00	s4+s7=-00	1 110 101 00
1 0010000 0000001 00	s1+s4=0-0	1 101 100 00

s2+s3+s7=1--
(no 111 code)

Penn ESE535 Spring 2009 -- DeHon

24

Input and Output

Skip?

Penn ESE535 Spring 2009 -- DeHon

25

General Problem

- Track both input and output encoding constraints

Penn ESE535 Spring 2009 -- DeHon

26

General Two-Level Strategy

1. Generate "Generalized" Prime Implicants
2. Extract/identify encoding constraints
3. Cover with minimum number of GPIs that makes encodeable
4. Encode symbolic values

[Devadas+Newton/TR CAD v10n1p13]

Penn ESE535 Spring 2009 -- DeHon

27

Output Symbolic Sets

- Maintain output state, PIs as a set
- Represent inputs one-hot as before

0 S1 S1 1	0 100 S1 1	0 100 (S1) (o1)
1 S1 S2 0	1 100 S2 0	1 100 (S2) ()
1 S2 S2 0	1 010 S2 0	1 010 (S2) ()
0 S2 S3 0	0 010 S3 0	0 010 (S3) ()
1 S3 S3 1	1 001 S3 1	1 001 (S3) (o1)
0 S3 S3 1	0 001 S3 1	0 001 (S3) (o1)

Penn ESE535 Spring 2009 -- DeHon

28

Generate GPIs

- Same basic idea as PI generation
 - Quine-McKlusky
- ...but different

Penn ESE535 Spring 2009 -- DeHon

29

Merging

- Cubes merge if
 - distance one in input
 - 000 100
 - 001 100 → 00- 100
 - inputs same, differ in multi-valued input (state)
 - 000 100
 - 000 010 → 000 110

Penn ESE535 Spring 2009 -- DeHon

30

Merging

- When merge
 - binary valued output contain outputs asserted in both (and)
 - 000 100 (foo) (o1,o2)
 - 001 100 (bar) (o1,o3) → 00- 100 ? (o1)
 - next state tag is union of states in merged cubes
 - 000 100 (foo) (o1,o2)
 - 001 100 (bar) (o1,o3) → 00- 100 (foo,bar) (o1)

Penn ESE535 Spring 2009 -- DeHon

31

Merged Outputs

- Merged outputs
 - Set of things asserted by this input
 - States would like to turn on together
 - 000 100 (foo) (o1,o2)
 - 001 100 (bar) (o1,o3) → 00- 100 (foo,bar) (o1)

Penn ESE535 Spring 2009 -- DeHon

32

Cancellation

- K+1 cube cancels k-cube **only if**
 - multivalued input is identical
 - AND next state and output identical
 - 000 100 (foo) (o1)
 - 001 100 (foo) (o1)
 - Also cancel if multivalued input contains all inputs
 - 000 111 (foo) (o1)
- Discard cube with next state containing all symbolic states and null output
 - 111 100 (foo,bar,baz...) () → does nothing

Penn ESE535 Spring 2009 -- DeHon

33

Example (work on board)

```
0 100 (S1) (o1)
1 100 (S2) ()
1 010 (S2) ()
0 010 (S3) ()
1 001 (S3) (o1)
0 001 (S3) (o1)
```

Penn ESE535 Spring 2009 -- DeHon

34

Cancellation

- K+1 cube cancels k-cube **only if**
 - multivalued input is identical
 - AND next state and output identical
 - 000 100 (foo) (o1) 00- 100 (foo) (o1)
 - 001 100 (foo) (o1)
 - Also cancel if multivalued input contains all inputs
 - 000 111 (foo) (o1)
- Discard cube with next state containing all symbolic states and null output
 - 111 100 (foo,bar,baz...) () → does nothing

Penn ESE535 Spring 2009 -- DeHon

35

Example

```
0 100 (S1) (o1)      - 100 (S1,S2) ()      0 111 (S1,S3) ()
1 100 (S2) ()        0 110 (S1,S3) () x      - 011 (S2,S3) ()
1 010 (S2) ()        0 101 (S1,S3) (o1)      - 111 (S2,S3) ()
0 010 (S3) ()        1 110 (S2)              - 110 (S1,S2,S3) () x
1 001 (S3) (o1) x    1 101 (S2,S3) () x
0 001 (S3) (o1) x    - 010 (S2,S3) ()
                    1 011 (S2,S3) () x
                    0 011 (S3) ()
                    - 001 (S3) (o1)
```

Penn ESE535 Spring 2009 -- DeHon

36

Covering

- Cover with branch-and-bound similar to two-level
 - row dominance only if
 - tags of two GPIs are identical
 - OR tag of first is subset of second
- Once cover, check encodeability
 - [talk about next]
- If fail, branch-and-bound again on additional GPIs to add to satisfy encodeability

Penn ESE535 Spring 2009 -- DeHon

37

Encoding Constraints

- Minterm to symbolic state v
 - should assert v
- For all minterms m
 - $\bigcup \text{all GPIs } [(\bigcap \text{all symbolic tags}) e(\text{tag state})] = e(v)$

```

0 S1 S1 1
1 S1 S2 0
1 S2 S2 0
0 S2 S3 0
1 S3 S3 1
0 S3 S3 1
    
```

Penn ESE535 Spring 2009 -- DeHon

38

Example

$\bigcup \text{all GPIs } [(\bigcap \text{all symbolic tags}) e(\text{tag state})] = e(v)$

Consider 1101 (out1) covered by

1101 out1	110- (out1,out2)	110- (out1,out2)
1100 out2	11-1 (out1,out3)	11-1 (out1,out3)
1111 out3	000- (out4)	
x 0000 out4		
x 0001 out4		

110- $\rightarrow e(\text{out1}) \cap e(\text{out2})$
 11-1 $\rightarrow e(\text{out1}) \cap e(\text{out3})$

OR-plane gives me OR of these two

Want output to be $e(\text{out1})$

$1101 e(\text{out1}) \cap e(\text{out2}) \cup e(\text{out1}) \cap e(\text{out3}) = e(\text{out1})$

Penn ESE535 Spring 2009 -- DeHon

39

Example

$\bigcup \text{all GPIs } [(\bigcap \text{all symbolic tags}) e(\text{tag state})] = e(v)$

Sample Solution:

1101 out1	110- (out1,out2)	out1=11
1100 out2	11-1 (out1,out3)	out2=01 110- 01
1111 out3	000- (out4)	out3=10 11-1 10
x 0000 out4		out4=00
x 0001 out4		Think about PLA

$1101 e(\text{out1}) \cap e(\text{out2}) \cup e(\text{out1}) \cap e(\text{out3}) = e(\text{out1})$
 1100 $e(\text{out1}) \cap e(\text{out2}) = e(\text{out2})$
 1111 $e(\text{out1}) \cap e(\text{out3}) = e(\text{out3})$
 0000 $e(\text{out4}) = e(\text{out4})$
 0001 $e(\text{out4}) = e(\text{out4})$

Penn ESE535 Spring 2009 -- DeHon

40

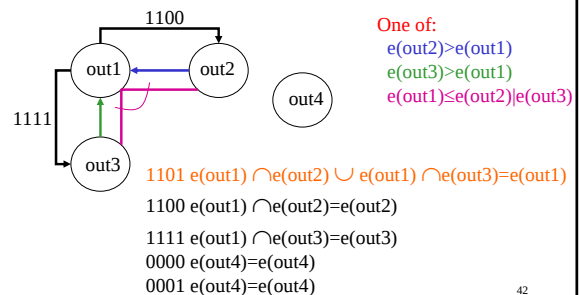
To Satisfy

- Dominance and disjunctive relationships from encoding constraints
- e.g.
 - $e(\text{out1}) \cap e(\text{out2}) \cup e(\text{out1}) \cap e(\text{out3}) = e(\text{out1})$ > Means strictly more bits on
 - one of:
 - $e(\text{out2}) > e(\text{out1})$ [i.e. $e(\text{out1}) \cap e(\text{out2}) = e(\text{out1})$]
 - $e(\text{out3}) > e(\text{out1})$ [i.e. $e(\text{out1}) \cap e(\text{out3}) = e(\text{out1})$]
 - $e(\text{out2}) | e(\text{out3}) \geq e(\text{out1})$

Penn ESE535 Spring 2009 -- DeHon

41

Encodeability Graph



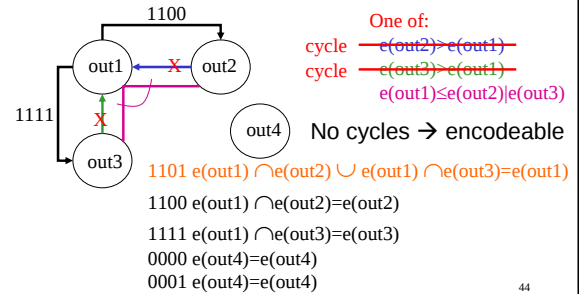
Penn ESE535 Spring 2009 -- DeHon

42

Encoding Constraints

- No directed cycles (proper dominance)
- Siblings in disjunctive have no directed paths between
 - (one cannot dominate other)
- No two disjunctive equality can have exactly the same siblings for different parents
- Parent of disjunctive should not dominate all sibling arcs

Encodeability Graph



Determining Encoding

- Can turn into boolean satisfiability problem for a target code length
- All selected encoding constraints become boolean expressions
- Also uniqueness constraints

What we've done

- Define another problem
 - Constrained coding
- This identifies the necessary coding constraints
 - Solve optimally with SAT solver
 - Or attack heuristically

Summary

- Encoding can have a big effect on area
- Freedom in encoding allows us to maximize opportunities for sharing
- Can do minimization around unencoded to understand structure in problem outside of encoding
- Can adapt two-level covering to include and generate constraints
- Multilevel limited by our understanding of structure we can find in expressions
 - heuristics try to maximize expected structure

Admin

- Assign 3 due Wednesday
- Wednesday Reading on Web

Today's Big Ideas

- Exploit freedom
- Bounding solutions
- Dominators
- Formulation and Reduction
- Technique:
 - branch and bound
 - SAT
 - Understanding structure of problem
 - Creating structure in the problem