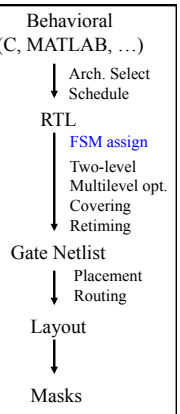
ESE535: Electronic Design Automation

Day 18: March 28, 2011
Sequential Optimization
(FSM Encoding)

Today

- Encoding
  - Input
  - Output
- State Encoding
  - "exact" two-level



Input Encoding

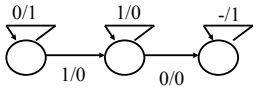
- Pick codes for input cases to simplify logic
- *E.g.* Instruction Decoding
  - ADD, SUB, MUL, OR
- Have freedom in code assigned
- Pick code to minimize logic
  - *E.g.* number of product terms

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

Finite-State Machine

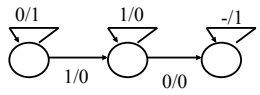
- Logical behavior depends on state
- In response to inputs, may change state



State Encoding

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

Finite State Machine



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

Example: Encoding Difference

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

S1=01
S2=11
S3=10

S1+S2 = -1

Similar outputs, code so S1+S2 is simple cube

S1=00
S2=11
S3=10

0 01 01 1
1 01 11 0
1 11 11 0
0 11 10 0
1 10 10 1
0 10 10 1

0 01 01 1
1 -1 11 0
- 11 10 0
- 10 10 1

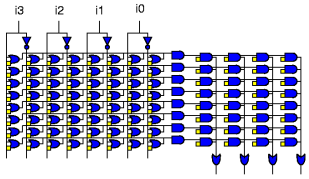
0 00 00 1
1 00 11 0
1 11 11 0
0 11 10 0
1 10 10 1
0 10 10 1

1 11 01 0
1 00 11 0
0 -0 00 1
- 10 00 1
- 1- 10 0

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

Two-Level



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

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)

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

Kinds of Sharing

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

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

10 inp1+inp2=01
11 inp2+inp3=01

Inp1=10
Inp2=11
Inp3=01

10 1- 01
01 10 10
11 –1 01
01 11 01
01 01 10

Output sharing:
share input cubes to produce individual output bits

1101 out1
1100 out2
1111 out3
0000 out4
0001 out4

Out1=11
Out2=01
Out3=10
Out4=00

110- 01
11-1 10

Input Encoding

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]

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

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

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

One-hot Minimum

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

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) → 01

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)

Example

0 S s6 00
0 s2 s5 00
0 s3 s5 00
0 s4 s6 00
0 s5 S 10
0 s6 S 01
0 s7 s5 00
1 s6 s2 01
1 s5 s2 10
1 s4 s6 10
1 s7 s6 10
1 S s4 00
1 s2 s3 00
1 s3 s7 00

0 1000000 0000010 00
0 0100000 0000100 00
0 0010000 0000100 00
0 0001000 0000010 00
0 0000100 1000000 10
0 0000010 1000000 01
0 0000001 0000100 00
1 0000010 0100000 01
1 0000100 0100000 10
1 0001000 0000010 10
1 0000001 0000010 10
1 1000000 0001000 00
1 0100000 0010000 00
1 0010000 0000001 00

0 0110001 0000100 00
0 1001000 0000010 00
1 0001001 0000010 10
0 0000010 1000000 01
1 0000100 0100000 10
0 0000100 1000000 10
1 1000000 0001000 00
1 0000010 0100000 01
1 0100000 0010000 00
1 0010000 0000001 00

Two-Level Input

- One-hot identifies multivalue minimum number of product terms
- May be fewer 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

Encoding Example

0 S s6 00
0 s2 s5 00
0 s3 s5 00
0 s4 s6 00
0 s5 S 10
0 s6 S 01
0 s7 s5 00
1 S s4 01
1 s2 s3 10
1 s3 s7 10
1 s4 s6 10
1 s5 s2 00
1 s6 s2 00
1 s7 s6 00

0 0110001 0000100 00
0 1001000 0000010 00
1 0001001 0000010 10
0 0000010 1000000 01
1 0000100 0100000 10
0 0000100 1000000 10
1 1000000 0001000 00
1 0000010 0100000 01
1 0100000 0010000 00
1 0010000 0000001 00

S 010
s2 110
s3 101
s4 000
s5 001
s6 011
s7 100

s2+s3+s7=1--
No 111 code

Encoding Example

0 0110001 0000100 00
0 1001000 0000010 00
1 0001001 0000010 10
0 0000010 1000000 01
1 0000100 0100000 10
0 0000100 1000000 10
1 1000000 0001000 00
1 0000010 0100000 01
1 0100000 0010000 00
1 0010000 0000001 00

S 010
s2 110
s3 101
s4 000
s5 001
s6 011
s7 100

s4+s7=-00

S+s4=0-0

0 1-- 001 00
0 0-0 011 00
1 -00 011 10
0 011 010 01
1 001 110 10
0 001 010 10
1 010 000 00
1 011 110 01
1 110 101 00
1 101 100 00

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

Input and Output

Skip?

General Problem

- Track both input and output encoding constraints

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]

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)

Generate GPIs

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

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

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)

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)

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

Example

(copy to board…work;

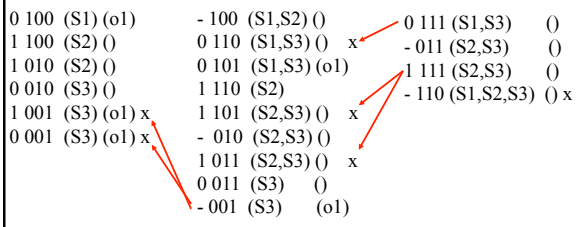
Note inclass exercise, back of preclass)

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

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

Example



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

Encoding Constraints

- Minterm to symbolic state v
  - should assert 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

- For all minterms m
  - ∪all GPIs [(∩all symbolic tags) e(tag state)] = e(v)

Example

∪all GPIs [(∩all symbolic tags) e(tag state)] = e(v)

1101 out1
1100 out2
1111 out3
x 0000 out4
x 0001 out4

110- (out1,out2)
11-1 (out1,out3)
000- (out4)

Consider 1101 (out1)
covered by
110- (out1,out2)
11-1 (out1,out3)

110- → e(out1) ∩ e(out2)
11-1 → e(out1) ∩ e(out3)

OR-plane gives me OR of these two

Want output to be e(out1)

1101 e(out1) ⋂e(out2) ⋃ e(out1) ⋂e(out3)=e(out1)

Example

∪all GPIs [(∩all symbolic tags) e(tag state)] = e(v)

1101 out1
1100 out2
1111 out3
x 0000 out4
x 0001 out4

110- (out1,out2)
11-1 (out1,out3)
000- (out4)

Sample Solution:
out1=11
out2=01 110- 01
out3=10 11-1 10
out4=00
Think about PLA

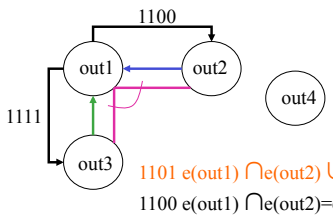
1101 e(out1) ⋂e(out2) ⋃ e(out1) ⋂e(out3)=e(out1)
1100 e(out1) ⋂e(out2)=e(out2)
1111 e(out1) ⋂e(out3)=e(out3)
0000 e(out4)=e(out4)
0001 e(out4)=e(out4)

To Satisfy

- Dominance and disjunctive relationships from encoding constraints
- *e.g.*
  - e(out1) ⋂ e(out2) ⋃ e(out1) ⋂ e(out3)=e(out1)
  - one of:
    - e(out2)>e(out1) [i.e. e(out1) ⋂e(out2)=e(out1)]
    - e(out3)>e(out1) [i.e. e(out1) ⋂e(out3)=e(out1)]
    - e(out2)|e(out3)≥ e(out1)

> Means strictly more bits on

Encodeability Graph



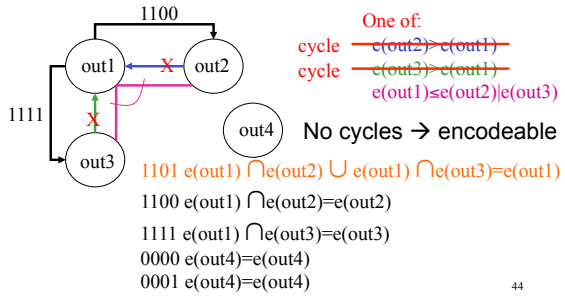
One of:
e(out2)>e(out1)
e(out3)>e(out1)
e(out1)≤e(out2)|e(out3)

1101 e(out1) ⋂e(out2) ⋃ e(out1) ⋂e(out3)=e(out1)
1100 e(out1) ⋂e(out2)=e(out2)
1111 e(out1) ⋂e(out3)=e(out3)
0000 e(out4)=e(out4)
0001 e(out4)=e(out4)

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

- Assignment 6, 7 out
  - For Assignment 6 you essentially write the assignment for 7
- Wednesday Reading on Web
- Normal office hours this week (T4:30pm)

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