ESE535: Electronic Design Automation

Day 20: April 8, 2015
Sequential Optimization (FSM Encoding)

Today

- Encoding
  - Input
  - Output
- State Encoding
  - “exact” two-level
    - at least input constraints
    - Flavor of output
  - Energy-oriented

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

- What’s a FSM?
  - Or DFA = Deterministic Finite Automata?
FSM

- Logic depends on past inputs
- Behaves differently based on state
- Logic selects outputs and next state
  - Based on inputs and current state

FSM Examples

- What are examples where need an FSM rather than just combination logic?
  - Parsing
  - Protocols
  - Datapath control
  - Data dependent branching

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

Motivating Example

Preclass 1—3

- How many PTERMs after optimization?
  01010
  01111
  10010
  10101
  11011
  11101
- What caused savings?

Preclass 4—5

- How many PTERMs after optimization?
  000--
  001--
  01010
  01111
  10101
  10111
  11011
  11101
- What caused savings?

Preclass 6

- Anyone get smaller? Encoding?

Two Issues

- Input Coding
  - Use a single input cube to select an output
  - Capture the union of things that behave similarly on a single cube
- Output Coding
  - Only need to cover the 1’s
  - Share logic producing 1’s between states
What input cases produce same output?

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST1</td>
<td>0</td>
<td>ST2</td>
</tr>
<tr>
<td>ST1</td>
<td>1</td>
<td>ST3</td>
</tr>
<tr>
<td>ST2</td>
<td>0</td>
<td>ST2</td>
</tr>
<tr>
<td>ST2</td>
<td>1</td>
<td>ST1</td>
</tr>
<tr>
<td>ST3</td>
<td>0</td>
<td>ST3</td>
</tr>
<tr>
<td>ST3</td>
<td>1</td>
<td>ST1</td>
</tr>
</tbody>
</table>

Produce same output?

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST1</td>
<td>0</td>
<td>ST2</td>
</tr>
<tr>
<td>ST1</td>
<td>1</td>
<td>ST3</td>
</tr>
<tr>
<td>ST2</td>
<td>0</td>
<td>ST2</td>
</tr>
<tr>
<td>ST2</td>
<td>1</td>
<td>ST1</td>
</tr>
<tr>
<td>ST3</td>
<td>0</td>
<td>ST3</td>
</tr>
<tr>
<td>ST3</td>
<td>1</td>
<td>ST1</td>
</tr>
</tbody>
</table>

If we can code ST1+ST2 and ST2+ST3 as cubes, we can save due to input encodings.
- How could we make these cubes?
  - 3b state code?
  - 2b state code?

Problem:
- **Real**: pick state encodings (s_i’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 $\Rightarrow$ 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
- 11 inp2 01
- 01 inp2 01
- 11 inp3 01
- 01 inp3 10
- 10 inp1+inp2=01
- 11 inp2+inp3=01

Output sharing:
- share input cubes to produce individual output bits
- 10 11
- 01 01

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 \rightarrow 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

One-hot Minimum

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

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)
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

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>010</td>
<td>S</td>
</tr>
<tr>
<td>110</td>
<td>s2</td>
</tr>
<tr>
<td>011</td>
<td>s3</td>
</tr>
<tr>
<td>000</td>
<td>s4</td>
</tr>
<tr>
<td>100</td>
<td>s5</td>
</tr>
<tr>
<td>001</td>
<td>s6</td>
</tr>
<tr>
<td>101</td>
<td>s7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>s2+s3+s7=1--</td>
</tr>
<tr>
<td>010</td>
<td>(no 111 code)</td>
</tr>
</tbody>
</table>

Input and Output

Concept

Idea

- Input constraints → state sets can select with single cube
  - (previous section)
- Output constraints
  - Track set of states encode together
  - OR of asserted minterms is desired state
- Encode constraints
- Solve with SAT solver

Encoding Constraints

- Minterm to symbolic state v
  - should assert v
- For all minterms m
  - Uall GPIs [(\forall all symbolic tags) e(tag state)] = e(v)
Example: Output Constraints

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

Consider 1101 (out1) covered by 0110- (out1,out2) 1110- (out1,out2) 1101 (out1,out3) 1111 (out1,out3)

\(x\) 0000 out4 \(x\) 0001 out4

OR-plane gives me OR of these two

Want output to be e(out1)

1101 e(out1) \(\bigcap\) e(out2) \(\bigcup\) e(out1) \(\bigcap\) e(out3) = e(out1)

Sample Solution:

<table>
<thead>
<tr>
<th></th>
<th>out1</th>
<th>out2</th>
<th>out3</th>
<th>out4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1101</td>
<td>11</td>
<td>01</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>1100</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>1111</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>00</td>
</tr>
</tbody>
</table>

Think about PLA

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

<table>
<thead>
<tr>
<th></th>
<th>out1</th>
<th>out2</th>
<th>out3</th>
<th>out4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1101</td>
<td>11</td>
<td>01</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>1100</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td>1111</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>00</td>
</tr>
</tbody>
</table>

Idea

- Input constraints \(\rightarrow\) state sets can select with single cube
  - (previous section)
- Output constraints
  - Track set of states encode together
  - OR of asserted minterms is desired state
- Encode constraints
- Solve with SAT solver

Encoding for Energy Minimization

Energy-Minimization

- How would select encodings to minimize energy?

Energy-Minimization

- Spend energy when state bits switch
  - Driving outputs, driving inputs
- Minimize distance between states
  - Esp. common state transitions
Energy Cost

\[ \sum_{s_i, s_j} P(s_i, s_j) \times \text{Weight}(e_i \otimes e_j) \]

Energy Encoding

- How approach?
  - Greedy
  - Simulated Annealing
  - ILP

Summary

- Encoding can have a big effect on area, energy
- 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

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

Admin

- Monday Reading on Web

Input and Output Details

Don’t expect to cover
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

Output Symbolic Sets

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

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>010</td>
<td>001</td>
</tr>
<tr>
<td>010</td>
<td>001</td>
<td>010</td>
</tr>
<tr>
<td>001</td>
<td>010</td>
<td>001</td>
</tr>
<tr>
<td>010</td>
<td>001</td>
<td>010</td>
</tr>
<tr>
<td>001</td>
<td>010</td>
<td>001</td>
</tr>
<tr>
<td>010</td>
<td>001</td>
<td>010</td>
</tr>
</tbody>
</table>

Generate GPIs

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

Merging

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

Merging

- When merge
  - binary valued output contain outputs asserted in both (and)
    - 000 100 (foo) (o1.o2)
    - 001 100 (bar) (o1.o3) \(\Rightarrow\) 00- 100 ? (o1)
  - next state tag is union of states in merged cubes
    - 000 100 (foo) (o1.o2)
    - 001 100 (bar) (o1.o3) \(\Rightarrow\) 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) \(\rightarrow\) 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…) () \(\rightarrow\) does nothing

Example

(copy to board…work; Note inclass exercise, back of preclass)

0 100  (S1) (o1)
1 100  (S2) ()
0 100 (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)
    - 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…) () \(\rightarrow\) does nothing

Example

- 100 (S1) (o1)
  - 100 (S2) ()
  - 100 (S3) ()
  - 001 (S3) (o1)
  - 001 (S3) (o1)

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

• 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)

Sample Solution:
- out1=1
- out2=0
- out3=0
- out4=0

Think about PLA
- 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)

To Satisfy

• Dominance and disjunctive relationships from encoding constraints
  > Means strictly more bits on
  - e(out1) ∩ e(out2) ∪ e(out1) ∩ e(out3)=e(out1)
  - one of:
    • e(out2)>e(out1)
    • e(out3)>e(out1)
    • e(out2)∪e(out3)= e(out1)

Encodeability Graph

One of:
- e(out2)>e(out1)
e(out3)>e(out1)
e(out1)=e(out2)∪e(out3)

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

No cycles \(\rightarrow\) encodeable

- \(1101 e(out1) \cap e(out2) \cup e(out1) \cap e(out3) = e(out1)\)
- \(1100 e(out1) \cap e(out2) = e(out2)\)
- \(1111 e(out1) \cap e(out3) = e(out3)\)
- \(0000 e(out4) = e(out4)\)
- \(0001 e(out4) = e(out4)\)

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