CS137: Electronic Design Automation

Day 7: October 12, 2005
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

Example: Encoding Difference

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

- \( A_{\text{pla}} = (2*\text{ins+outs})*\text{prods} + \text{flops} * \text{wflop} \)
- 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

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

```
10 inp1 01
01 inp1 10
1 inp2 01
01 inp2 01
11 inp3 01
01 inp3 10
```

```
1101 out1
1100 out2
0111 out3
0110 out4
```

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

Input Encoding

```
Output sharing:
- share input cubes
- to produce individual
- output bits
```

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
```

```
1101 out1
1100 out2
0111 out3
0110 out4
```

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

```
Inp1=10
Inp2=11
Inp3=01
```

```
10 1- 01
01 10 10
1- inp2 01
01 inp2 01
11 inp3 01
01 inp3 10
```

```
101-01
100-02
111-03
000-04
```

```
110-01
111-01
```

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

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

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

- **inp1 = 1**: 0100
- **inp2 = 1**: 0010
- **inp3 = 1**: 0001

#### One-hot:

- **inp1 = 1**: 1000
- **inp2 = 1**: 0100
- **inp3 = 1**: 0010

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

---

### 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)

---

### Encoding Example

- **S**: 000
- **s**: 001
- **s2**: 010
- **s3**: 011
- **s4**: 100
- **s5**: 101
- **s6**: 110
- **s7**: 111

#### Encodings:

- **s = 010**: 00100000 00000100 00
- **s2 = 110**: 00100000 00000100 00
- **s3 = 101**: 00100000 00000100 00
- **s4 = 000**: 00100000 00000100 00
- **s5 = 001**: 00100000 00000100 00
- **s6 = 110**: 00100000 00000100 00
- **s7 = 111**: 00100000 00000100 00

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

---

### Two-Level Input

- One-hot identifies multivalue 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 less bits than one-hot
  - this part typically heuristic
  - Remember one-hot already minimized prod terms
Input and Output

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>State</th>
<th>Input</th>
<th>Symbolic Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0</td>
<td>S1 1</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>S2 1</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>S3 1</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>S1 1</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>S2 1</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>S3 1</td>
</tr>
<tr>
<td>S1</td>
<td>0</td>
<td>S1 0</td>
</tr>
<tr>
<td>S2</td>
<td>0</td>
<td>S2 0</td>
</tr>
<tr>
<td>S3</td>
<td>0</td>
<td>S3 0</td>
</tr>
<tr>
<td>S1</td>
<td>1</td>
<td>S1 0</td>
</tr>
<tr>
<td>S2</td>
<td>1</td>
<td>S2 0</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>S3 0</td>
</tr>
</tbody>
</table>

Generate GPIs

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

Merging

- Cubes merge if
  – distance one in input
    - 00 000
    - 001 001
  – inputs same, differ in multi-valued input (state)
    - 000 000
    - 000 010
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) (o1,03) ➔ 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 (o1) ➔ does nothing

Example (work on board)

- 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 (o1) ➔ does nothing
Encoding Constraints

• Minterm to symbolic state v
  – should assert v

• For all minterms m
  – ∪ all GPs ((all symbolic tags) e(tag state)) = e(v)

Example

Consider 1101 (out1) covered by
110- (out1,out2)
1100 (out2)
1111 (out3)
000- (out4)

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)

To Satisfy

• Dominance and disjunctive relationships from encoding constraints
• e.g.
  – means strictly more bits on
  – (one cannot dominate other)

Encodability Graph

• 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

Out1 Out2
Out3
Out4

1100
1111

One of:

- cycle
- cycle
- cycle

1101 <out1> \wedge <out1> \wedge <out1> <out1>
1100 <out1> \wedge <out2> = <out2>
1111 <out1> \wedge <out3> = <out3>
0000 <out4> = <out4>
0001 <out4> = <out4>

No cycles \rightarrow encodeable

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
  
  - If fail, branch-and-bound again on additional
    GPIs to add to satisfy encodeability

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

- Friday:
  - Guest Lecturer: Dr. Gary Burke (JPL)
Today’s Big Ideas

• Exploit freedom
• Bounding solutions
• Dominators
• Formulation and Reduction
• Technique:
  – branch and bound
  – Understanding structure of problem