Day 5: February 4, 2008
Sequential Optimization
(FSM Encoding)

Please work preclass example before we start lecture.

---

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

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

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Out1</th>
<th>Out2</th>
<th>Out3</th>
<th>Out4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 inp1 01</td>
<td>1101</td>
<td>1100</td>
<td>1111</td>
<td>0000</td>
</tr>
<tr>
<td>01 inp2 01</td>
<td>0101</td>
<td>1010</td>
<td>0011</td>
<td>0001</td>
</tr>
<tr>
<td>11 inp3 01</td>
<td>1110</td>
<td>0111</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>01 inp3 10</td>
<td>0110</td>
<td>1010</td>
<td>0011</td>
<td>0001</td>
</tr>
</tbody>
</table>

Input Encoding

- Output sharing: share input cubes to produce individual output bits
- Input sharing: encode inputs so cover set to reduce product terms

<table>
<thead>
<tr>
<th>Input</th>
<th>Out1</th>
<th>Out2</th>
<th>Out3</th>
<th>Out4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 inp1 01</td>
<td>1101</td>
<td>1100</td>
<td>1111</td>
<td>0000</td>
</tr>
<tr>
<td>01 inp2 01</td>
<td>0101</td>
<td>1010</td>
<td>0011</td>
<td>0001</td>
</tr>
<tr>
<td>11 inp3 01</td>
<td>1110</td>
<td>0111</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>01 inp3 10</td>
<td>0110</td>
<td>1010</td>
<td>0011</td>
<td>0001</td>
</tr>
</tbody>
</table>

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

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

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**

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>001000</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>010000</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>001000</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>010000</td>
</tr>
</tbody>
</table>

---

**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**

<table>
<thead>
<tr>
<th>State</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**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

---

**Encoding Example**

<table>
<thead>
<tr>
<th>State</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Encoding Example**

<table>
<thead>
<tr>
<th>State</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Example**

<table>
<thead>
<tr>
<th>State</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Encoding Example**

<table>
<thead>
<tr>
<th>State</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Example**

<table>
<thead>
<tr>
<th>State</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Encoding Example**

<table>
<thead>
<tr>
<th>State</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

---

**Example**

<table>
<thead>
<tr>
<th>State</th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>s5</th>
<th>s6</th>
<th>s7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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

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

(work on board)

<table>
<thead>
<tr>
<th>State</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td></td>
<td></td>
<td></td>
<td>o1</td>
</tr>
<tr>
<td>1100</td>
<td></td>
<td>S2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1010</td>
<td>S1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010</td>
<td>S3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>S3</td>
<td></td>
<td>S1</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>S3</td>
<td></td>
<td>S2</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>S3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1011</td>
<td>S3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0011</td>
<td>S3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1110</td>
<td></td>
<td>S3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td></td>
<td>S3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example

- 0100 (S1) (o1)
- 1100 (S2)
- 1010 (S2) ()
- 0010 (S3) ()
- 1001 (S3) (o1)
- 0101 (S3) (o1)
- 1110 (S1,S2)
- 1111 (S1,S2,S3) (o1)
- 0111 (S1,S3) ()
- 0011 (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…) () ⇒ does nothing

Example

- 0100 (S1) (o1)
- 1100 (S2)
- 1010 (S2) ()
- 0010 (S3) ()
- 1001 (S3) (o1)
- 0101 (S3) (o1)
- 1110 (S1,S2)
- 1111 (S1,S2,S3) (o1)
- 0111 (S1,S3) ()
- 0011 (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 $v$

\[
0 \ S1 \ S1 \ 1 \\
1 \ S1 \ S2 \ 0 \\
1 \ S2 \ S2 \ 0 \\
0 \ S2 \ S3 \ 0 \\
1 \ S3 \ S3 \ 1 \\
0 \ S3 \ S3 \ 1
\]

- For all minterms $m$
  - for all GPIs $\{r\text{-所有symbolic tags}\} e(tag\ state) = e(v)$

### Example

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

Consider 1101 (out1) covered by

- \[110- (\text{out1}, \text{out2})\]
- \[11-1 (\text{out1}, \text{out3})\]
- \[000- (\text{out4})\]
- \[1111 \text{out3}\]
- \[0001 \text{out4}\]

OR-plane gives me OR of these two

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

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

### To Satisfy

- Dominance and disjunctive relationships from encoding constraints
  - e.g.  
    - $e(\text{out1}) \cap e(\text{out2}) \cup e(\text{out1}) \cap e(\text{out3}) = e(\text{out1})$
    - 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. $e(\text{out1}) \cap e(\text{out3}) = e(\text{out1})$]
      - $e(\text{out2}) \geq e(\text{out3})$

### Encodeability Graph

For all minterms $m$

- $\forall\text{GPIs} \left\{ \left( \forall\text{symbolic tags}\right) e(\text{tag\ state}) = e(v) \right\}$

Sample Solution:

\[
\begin{align*}
1101 \text{ out1} & \quad 110- (\text{out1}, \text{out2}) \\
1100 \text{ out2} & \quad 11-1 (\text{out1}, \text{out3}) \\
1111 \text{ out3} & \quad 000- (\text{out4}) \\
x \text{ 0000 out4} & \\
x \text{ 0001 out4}
\end{align*}
\]

Think about PLA

\[
\begin{align*}
1101 \ e(\text{out1}) \cap e(\text{out2}) \cup e(\text{out1}) \cap e(\text{out3}) & = e(\text{out1}) \\
1100 \ e(\text{out1}) \cap e(\text{out2}) & = e(\text{out2}) \\
1111 \ e(\text{out1}) \cap e(\text{out3}) & = e(\text{out3}) \\
0000 \ e(\text{out4}) & = e(\text{out4}) \\
0001 \ e(\text{out4}) & = e(\text{out4})
\end{align*}
\]

One of:

\[
\begin{align*}
e(\text{out2}) & > e(\text{out1}) \\
e(\text{out3}) & > e(\text{out1}) \\
e(\text{out1}) & \leq e(\text{out2}) \cap e(\text{out3})
\end{align*}
\]
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

One of:

- cycle \( (\neg \text{out2}) \wedge \text{out1}) \wedge \neg (\text{out1}) \wedge \neg \neg \text{out3} \wedge \neg \text{out1}) = \text{out1} \)
- cycle \( \text{out1} \wedge \neg \text{out2} \wedge \neg \text{out1}) \wedge \neg \text{out1}) \wedge \neg \neg \text{out3} \wedge \neg \text{out1}) = \text{out1} \)
- cycle \( \text{out1} \wedge \neg \text{out2} \wedge \neg \text{out1}) \wedge \neg \text{out1}) \wedge \neg \text{out3} \wedge \neg \text{out1}) = \text{out1} \)
- cycle \( \text{out1} \wedge \neg \text{out2} \wedge \neg \text{out1}) \wedge \neg \text{out1}) \wedge \neg \text{out3} \wedge \neg \text{out1}) = \text{out1} \)

No cycles \( \rightarrow \) encodeable

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

• Syllabus on web
• Plan for Project
  – Simultaneous cover and place for delay
  – Warmup along with assignment 3
  – Then 3 2-week parts
• Poll C vs. Java
Today’s Big Ideas

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