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

0/1 1/0 0/0

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

### Example: Encoding Difference

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>01</th>
<th>01</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1 S1</td>
<td>1</td>
<td>01</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>S1 S2</td>
<td>0</td>
<td>11</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>S2 S3</td>
<td>0</td>
<td>11</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>S3 S3</td>
<td>1</td>
<td>11</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

Similar outputs, code so S1+S2 is simple cube

S1=01
S2=11
S3=10

S1+S2 = -1

### 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

- \( A_{pla} = (2 \times \text{ins+outs}) \times \text{prods} + \text{flops} \times \text{wflop} \)
- 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>Output 1</th>
<th>Output 2</th>
<th>Output 3</th>
<th>Output 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 inp1</td>
<td>1101</td>
<td>101</td>
<td>110</td>
<td>0000</td>
</tr>
<tr>
<td>01 inp1</td>
<td>1100</td>
<td>011</td>
<td>000</td>
<td>0001</td>
</tr>
<tr>
<td>01 inp2</td>
<td>110</td>
<td>000</td>
<td>111</td>
<td>1000</td>
</tr>
<tr>
<td>11 inp3</td>
<td>101</td>
<td>110</td>
<td>000</td>
<td>0100</td>
</tr>
<tr>
<td>01 inp3</td>
<td>000</td>
<td>011</td>
<td>101</td>
<td>0101</td>
</tr>
</tbody>
</table>

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

Output sharing:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output 1</th>
<th>Output 2</th>
<th>Output 3</th>
<th>Output 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 inp1</td>
<td>1101</td>
<td>101</td>
<td>110</td>
<td>0000</td>
</tr>
<tr>
<td>01 inp1</td>
<td>1100</td>
<td>011</td>
<td>000</td>
<td>0001</td>
</tr>
<tr>
<td>01 inp2</td>
<td>110</td>
<td>000</td>
<td>111</td>
<td>1000</td>
</tr>
<tr>
<td>11 inp3</td>
<td>101</td>
<td>110</td>
<td>000</td>
<td>0100</td>
</tr>
<tr>
<td>01 inp3</td>
<td>000</td>
<td>011</td>
<td>101</td>
<td>0101</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Output 1</th>
<th>Output 2</th>
<th>Output 3</th>
<th>Output 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 inp1</td>
<td>1101</td>
<td>101</td>
<td>110</td>
<td>0000</td>
</tr>
<tr>
<td>01 inp1</td>
<td>1100</td>
<td>011</td>
<td>000</td>
<td>0001</td>
</tr>
<tr>
<td>01 inp2</td>
<td>110</td>
<td>000</td>
<td>111</td>
<td>1000</td>
</tr>
<tr>
<td>11 inp3</td>
<td>101</td>
<td>110</td>
<td>000</td>
<td>0100</td>
</tr>
<tr>
<td>01 inp3</td>
<td>000</td>
<td>011</td>
<td>101</td>
<td>0101</td>
</tr>
</tbody>
</table>

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

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

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</th>
<th>One-hot:</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>inp1</td>
<td>10</td>
<td>1000 1</td>
</tr>
<tr>
<td>inp2</td>
<td>01</td>
<td>0100 0</td>
</tr>
<tr>
<td>inp3</td>
<td>01</td>
<td>0010 1</td>
</tr>
</tbody>
</table>

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

### 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
- (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>State</th>
<th>One-hot:</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>010</td>
<td>s0 10</td>
</tr>
<tr>
<td>s2</td>
<td>110</td>
<td>s2 110</td>
</tr>
<tr>
<td>s3</td>
<td>111</td>
<td>s3 11</td>
</tr>
<tr>
<td>s4</td>
<td>000</td>
<td>s4 00</td>
</tr>
<tr>
<td>s5</td>
<td>010</td>
<td>s5 01</td>
</tr>
<tr>
<td>s6</td>
<td>001</td>
<td>s6 01</td>
</tr>
<tr>
<td>s7</td>
<td>100</td>
<td>s7 100</td>
</tr>
</tbody>
</table>

s2+s3+s7=1--

No 111 code

s4+s7=0-0
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

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

[Devadas+Newton/TR CAD v10n1p13]
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

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

0 101  (S1) (x)
1 101  (S2) (x)
0 010  (S3) ()
1 011  (S3) (x)
0 011  (S3) (x)
0 110  (S1) (x)
1 110  (S2) (x)
0 111  (S1) (x)
1 111  (S2) (x)
1 101  (S2) (x)
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

Example

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

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

\[ 110- \rightarrow e(out1) \cap e(out2) \]
\[ 11-1 \rightarrow e(out1) \cap e(out3) \]

OR-plane gives me OR of these two
Want output to be e(out1)

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

To Satisfy

- Dominance and disjunctive relationships from encoding constraints
  - e.g.
    - \( e(out1) \cap e(out2) \cup e(out1) \cap e(out3) = e(out1) \)
    - one of:
      - \( e(out2) > e(out1) \) [i.e. \( e(out1) \cap e(out2) = e(out1) \)]
      - \( e(out3) > e(out1) \) [i.e. \( e(out1) \cap e(out3) = e(out1) \)]
      - \( e(out2) \cap e(out3) > e(out1) \)

Encodeability Graph

One of:
- \( e(out2) > e(out1) \)
- \( e(out3) > e(out1) \)
- \( e(out1) \cap e(out2) \cap e(out3) \)

\[ 1100 \leftrightarrow e(out1) \cap e(out2) \cup e(out1) \cap e(out3) = e(out1) \]
\[ 1100 e(out1) \cap e(out2) = e(out2) \]
\[ 1111 e(out1) \cap e(out3) = e(out3) \]
\[ 0000 e(out4) = e(out4) \]
\[ 0001 e(out4) = e(out4) \]
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

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