

# EDA (CS286.5b)

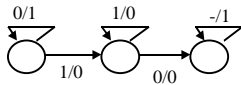
Day 17  
Sequential Logic Synthesis  
(FSM Optimization)

# Today

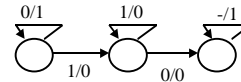
- State Encoding
  - “exact” two-level
  - heuristic multi-level

# Finite-State Machine

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



# 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

# 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

- $A = (2 * \text{inputs} + \text{outputs}) * \text{products} + \text{flops} * \text{wflop}$
- $\text{inputs} = \text{PIs} + \text{state\_bits}$
- $\text{outputs} = \text{state\_bits} + \text{POs}$
- products depend on state-bit encoding

## 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

- **Idea:** do symbolic minimization of two-level form
- This represents effects of sharing
- Generate encoding constraints from this
- Cover
- Select Codes

## Two-Level Input Oriented

- Minimize product rows by exploiting common-cube, next-state expressions

[DeMicheli+Brayton+SV/TR CAD v4n3p269]

## Multiple Valued Input Set

- Treat input states as a multi-valued (not just 0,1) input variable
- Effectively encode in one-hot form
- Use to merge together input state sets

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

## Two-Level Input

- Represent states as one-hot codes
- Minimize using two-level optimization
  - Include: combine compatible next states
- 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

## Example

0 S s6 00	0 1000000	0000010 00	
0 s2 s5 00	0 0100000	0000100 00	
0 s3 s5 00	0 0010000	0000100 00	0 0110001 0000100 00
0 s4 s6 00	0 0001000	0000010 00	0 1001000 0000010 00
0 s5 S 10	0 0000100	1000000 10	1 0001001 0000010 10
0 s6 S 01	0 0000010	1000000 01	0 0000010 1000000 01
0 s7 s5 00	0 0000001	0000100 00	1 0000100 0100000 10
1 S s4 01	1 0000010	0100000 01	0 0000100 1000000 10
1 s2 s3 10	1 0000100	0100000 10	1 1000000 0001000 00
1 s3 s7 10	1 0001000	0000010 10	1 0000010 0100000 01
1 s4 s6 10	1 0000001	0000010 10	1 0100000 0010000 00
1 s5 s2 00	1 1000000	0001000 00	1 0010000 0000001 00
1 s6 s2 00	1 0100000	0010000 00	
1 s7 s6 00	1 0010000	0000001 00	

## Two-Level Input

- One hot identifies multivalued 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 use less bits than one-hot
  - this part heuristic

## General Two-Level Strategy

- Generate “Generalized” Prime Implicants
- Extract/identify encoding constraints
- Cover with minimum number of GPIs that makes encodeable
- Encode symbolic values

[Devadas+Newton/TR CAD v10n1p13]

## Output Symbolic Sets

- Maintain output state, PIs as a set

0 100 S1 1	0 100 (S1) (o1)
1 100 S2 0	1 100 (S2) ()
1 010 S2 0	1 010 (S2) ()
0 010 S3 0	0 010 (S3) ()
1 001 S3 1	1 001 (S3) (o1)
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
  - inputs same, differ in multi-valued input (state)
- When merge
  - binary valued output contain outputs asserted in both (and)
  - next state tag is union of states in merged cubes

## Cancellation

- K+1 cube cancels k-cube only if
  - multivalued input is identical
  - AND next state and output identical
  - Also cancel if multivalued input contains all inputs
- Discard cube with next state containing all symbolic states and null output

### Example

```

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

### Example

```

0 100 (S1) (o1)      - 100 (S1,S2) ()      0 111 (S1,S3) ()
1 100 (S2) ()        0 110 (S1,S3) () x    - 011 (S2,S3) ()
1 010 (S2) ()        0 101 (S1,S3) (O1)    1 111 (S2,S3) ()
0 010 (S3) ()        1 110 (S2)                - 110 (S1,S2,S3) () x
1 001 (S3) (o1) x    1 101 (S2,S3) () x
0 001 (S3) (o1) x    - 010 (S2,S3) ()
                    1 011 (S2,S3) () x
                    0 011 (S3) ()
                    - 011 (S3) (O1)
    
```

### Encoding Constraints

- Minterm to symbolic state v should assert v
- For all minterms m
- Call GPIs (all symbolic tags) e(tag state) = e(v)

### Example

```

1101 out1      110- (out1,out2)
1100 out2      11-1 (out1,out3)
1111 out3      000- (out4)
0000 out4
0001 out4

1101 e(out1)Ee(out2)Çe(out1)Ee(out3)=e(out1)
1100 e(out1)Ee(out2)=e(out2)
1111 e(out1)Ee(out3)=e(out1)
0000 e(out4)=e(out4)
0001 e(out4)=e(out4)
    
```

### To Satisfy

- Dominance and disjunctive relationships from encoding constraints
- e.g.
  - e(out1)Ee(out2)Çe(out1)Ee(out3)=e(out1)
  - one of:
    - e(out2)>e(out1)
    - e(out3)>e(out1)
    - e(out1)Ee(out2)|e(out3)

### Encoding Constraints

- No directed cycles (proper dominance)
- siblings in disjunctive have not directed paths between
- no two disjunctives equality can have exactly the same siblings for different parents
- parent of disjunctive should not dominate all sibling arcs

## Encoding Constraints (more)

- For any tuple  $s_1, s_2, s_3$ 
  - such that  $s_1 > s_2, s_2 > s_3$
  - no input relation should force  $s_1, s_3$  to 1
    - while  $s_2 = 0$
- no input relation
  - all siblings (ancestors of sibling) of disjunctive equality arc have a 1 and the parent 0
  - all immediate ancestors of each conjunctive sibling have a 1 and the parent 0

## 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

## Multilevel

- Two level only tries to minimize product terms
  - (optimally  $io * product\_terms$ )
- Doesn't capture structure (CSE) available to multilevel logic
- Multilevel logic terms not easy to predict

[Devadas+Ma+Newton+SV/TR CAD v7n12p1290]

## Mustang

- **Idea:** maximize common cubes
- Code similar states "close" together (differ in fewest inputs) to maximize cube sharing

## Mustang Approach

- Construct full graph with node for every state
- Select weights for edges
- Use weights to construct groups
- Code groups close together (all undistant)

### Mustang Fanout

- For each state compute output set and weight
- For each state compute predecessor states
- Node edge weighted by common predecessors (common output treatment)

### Mustang Fanin

- Similar, but weight by common successors

### Clustering

- Minimize sum of
  - product of edge weights x distance
- Pick clusters of cardinality at most one greater than number of bits willing to expend
  - (can all be unidistant...assuming don't run into anything else)

### 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

### Today's Big Ideas

- Exploit freedom
- Bounding solutions
- Dominators
- Technique:
  - branch and bound