# Encoding Problems

Some logic specifications are “symbolic” rather than binary-valued

– e.g., specification for an ALU

| Symbol | Operation |
|---|---|
| ADD | + |
| SUB | – |
| DIV | : |
| INC | Increment |

Can assign any binary code to the symbolic values, as long as they are different

# Input Encoding


| | | | |
|---|---|---|---|
| 00 → | ADD | 00 | 00 |
| | ADD | 01 | 10 |
| | ADD | 10 | 10 |
| | ADD | 11 | 01 |
| 01 → | SUB | 00 | 00 |
| | SUB | 01 | 11 |
| | SUB | 10 | 10 |
| | SUB | 11 | 00 |
| 11 → | MUL | 00 | 00 |
| | MUL | 01 | 00 |
| | MUL | 10 | 00 |
| | MUL | 11 | 10 |

# Optimum Encoding

Given a function with symbolic inputs or outputs, we wish to find a binary encoding of inputs and outputs so as to minimize the area of the <u>encoded and optimized</u> logic function.

– Once the symbolic function has been encoded, we have a Boolean function that can be minimized

# Difficulty in Optimum Encoding

Function → ENCODE → Two-Level Boolean Minimization → Sum-of-products implementation

Given N symbols, and P bits to encode the N symbols by ($2^P \geq N$), the number of possible encodings is $\dfrac{(2^P)!}{(2^P - N)!}$

In a trivial exhaustive search method, need to perform exact logic minimizations for an optimum encoding

$N = 8$
$P = 3$ 10080 minimizations

# Example of Input Encoding

**Symbolic function**

| | | |
|---|---|---|
| 10 | inp1 | 01 |
| 01 | inp1 | 10 |
| 1- | inp2 | 01 |
| 01 | inp2 | 10 |
| 11 | inp3 | 01 |
| 01 | inp3 | 10 |
| -- | inp4 | 11 |

**One possible encoding:**

| | |
|---|---|
| inp1 | 10 |
| inp2 | 01 |
| inp3 | 00 |
| inp4 | 11 |

minimize →

| | | |
|---|---|---|
| 10 | 1- | 01 |
| 01 | -- | 10 |
| 11 | 0- | 01 |
| 1- | -1 | 01 |
| -- | 11 | 11 |

**Another encoding:**

| | |
|---|---|
| inp1 | 00 |
| inp2 | 01 |
| inp3 | 11 |
| inp4 | 10 |

minimize →

| | | |
|---|---|---|
| 11 | -1 | 01 |
| 10 | 0- | 01 |
| 01 | -- | 10 |
| -- | 10 | 11 |

# Optimum Input Encoding

**Goal:** Given a function with binary-valued and (possibly multiple) symbolic inputs, find a binary encoding for each of the different symbolic values so as to minimize the area of an eventual sum-of-products implementation.

- area = f(# product terms, # encoding bits)

**Simpler problem:** Minimize the number of product terms in the eventual implementation, and secondarily the number of encoding bits.

# Don’t Care Codes

**Encoding may produce don’t cares corresponding to unused codes**

**e.g.,**

| | | |
|---|---|---|
| 10 | inp1 | 01 |
| 01 | inp1 | 10 |
| 1- | inp2 | 01 |
| 01 | inp2 | 10 |
| 11 | inp3 | 01 |
| 01 | inp3 | 10 |

**encoding**

| | |
|---|---|
| inp1 | 00 |
| inp2 | 01 |
| inp3 | 11 |

**we get**

| | | |
|---|---|---|
| 10 | 00 | 01 |
| 01 | 00 | 10 |
| 1- | 01 | 01 |
| 01 | 01 | 10 |
| 11 | 11 | 01 |
| 01 | 11 | 10 |
| -- | 10 | -- |

**unused code 10 is a don’t care condition**

# A One-Hot Encoding

Encode symbolic input(s) with a one-hot code

$$\begin{array}{ll}
 & \overbrace{100 \quad \cdots \quad 0}^{N} \\
\text{inp 1} & 100 \quad \cdots \quad 0 \\
\text{inp 2} & 010 \quad \cdots \quad 0 \\
: & \\
\text{inpN} & 000 \quad \cdots \quad 1
\end{array}$$

As many bits as there are symbolic values

There is a large set of don't cares associated with a one-hot coding

$$\begin{array}{cc}
 & \overbrace{1\,1\,-\,-\,-\,-}^{N} \\
\text{don't cares} & 1\,-\,1\,-\,-\,- \\
\text{or} & : \\
\text{unused codes} & 0\,0\,0\,0\,0\,0
\end{array}$$

# Optimality of One-Hot Encoding

**Theorem: A one-hot input encoding results in a minimum number of product terms after an exact two-level Boolean minimization with the Don't Care Set.**

**Proof: (Intuitive argument) Say, two symbolic values assert the same output combination**

| | |
|---|---|
| inp1 | 1010 |
| inp2 | 1010 |

| | | | | | |
|---|---|---|---|---|---|
| 1000 | 1010 | | | | |
| 0100 | 1010 | | minimize | – – 00 | 1010 |
| 11 – – | – – – – | d.c's | | | |
| 0000 | – – – – | | | | |

# Optimality - 2

The don't care set allows merging of any subset of symbolic values that assert the same outputs (or partially assert the same outputs)

| | | |
|---|---|---|
| 100000 | inp1 | 0101 |
| 000010 | inp5 | 0101 |
| 000001 | inp6 | 0101 |

In a one-hot coding with DC set:

- 000- - 0101

In a different encoding:

inp1 011 inp5 110 inp6 101

( 011, 110, 101 ) cannot merge!

# One-Hot Encoding Example

| | | |
|---|---|---|
| 10 | inp1 | 01 |
| 01 | inp1 | 10 |
| 1- | inp2 | 01 |
| 01 | inp2 | 10 |
| 11 | inp3 | 01 |
| 01 | inp3 | 10 |

**One hot coding:**

| | | |
|---|---|---|
| 10 | 100 | 01 |
| 01 | 100 | 10 |
| 1- | 010 | 01 |
| 01 | 010 | 10 |
| 11 | 001 | 01 |
| 01 | 001 | 10 |
| -- | 11- | -- |
| -- | 1-1 | -- |
| -- | -11 | -- |
| -- | 000 | -- |

$\xrightarrow{\text{minimize}}$

| | | |
|---|---|---|
| 10 | --0 | 01 |
| 11 | 0-- | 01 |
| 01 | --- | 10 |

**Same can be done for a one-low code**

**i.e.,** inp1 011 inp2 101 inp3 110

# Problems with One-Hot Encoding

- **Too many encoding bits >> minimum-bit encoding**
  - 32 symbolic values:
    one-hot coding requires 32 bits
    minimum-bit coding requires 5 bits
- **Area of the eventual implementation will be large, even though # product terms minimum.**
- **Don't care sets required to minimize the function are huge.**

# Multiple-Valued Variables

N-valued variable:

$$\begin{array}{ll} a1: & 100 \ldots 0 \\ a2: & 010 \cdots 0 \\ & \vdots \\ aN: & \underbrace{000 \cdots 1}_{N} \end{array}$$

Dealing with functions that have multiple-valued inputs and binary-valued outputs

Any subset of the N values can merge in multiple-valued minimization

e.g., (1000, 0100, 0001) $\rightarrow$ 1101

(1000, 0001, 0100, 0010) $\rightarrow$ 1111

Just like in the one-hot coding case

# Multiple-Valued Minimization

- A one-hot binary coding minimized with an appropriate don't care set
- Replace the symbolic input with a multiple-valued variable as input

| | | |
|---|---|---|
| 10 | inp1 | 01 |
| 01 | inp1 | 10 |
| 1- | inp2 | 01 |
| 01 | inp2 | 10 |
| 11 | inp3 | 01 |
| 01 | inp3 | 10 |

$\longrightarrow$

| | | |
|---|---|---|
| 10 | 100 | 01 |
| 01 | 100 | 10 |
| 1- | 010 | 01 |
| 01 | 010 | 10 |
| 11 | 001 | 01 |
| 01 | 001 | 10 |

- Minimize function:

| | | |
|---|---|---|
| 10 | 110 | 01 |
| 11 | 011 | 01 |
| 01 | 111 | 10 |

# Constructing a Binary Encoding

| | | |
|---|---|---|
| 10 | 110 | 01 |
| 11 | 011 | 01 |
| 01 | 111 | 10 |

Each multiple-valued implicant (e.g., 110) represents a constraint that binary encoding has to satisfy

If we satisfy the constraint we can represent the term corresponding to the MV-implicant as a SINGLE implicant in the encoded binary cover

Satisfying ALL constraints will produce a binary cover with the same cardinality as minimized MV cover $\Rightarrow$ always possible!

# What do these constraints mean?

| | | |
|---|---|---|
| 10 | 110 | 01 |
| 11 | 011 | 01 |
| 01 | 111 | 10 |

The smallest cube that contains the codes of all symbolic values whose bit position has a 1 in the constraint = C

C should not intersect the codes of the symbolic values whose bit position has a 0 in the constraint

| | | |
|---|---|---|
| inp1 | 100 | inp1 and inp2 are in |
| inp2 | 010 | the constraint 110, |
| inp3 | 001 | inp3 is not |

# Supercubes

The smallest cube that contains a set of codes is called the supercube of that set of codes

Supercube is simply the bitwise OR of the set of codes

| MV | Symbol | Code |
|---|---|---|
| 100 | inp1 | 10010 |
| 010 | inp2 | 01010 |
| 001 | inp3 | 00110 |

Constraint 110 supercube (inp1, inp2)

10010
01010
- - 010

Constraint 011 supercube (inp2, inp3)

01010
00110
0- - 10

# Example

**M-V Cover:**

| | | |
|---|---|---|
| 10 | 1100 | 10 |
| 00 | 0110 | 01 |
| 11 | 0111 | 01 |
| 01 | 0001 | 01 |

**Encoding:**

| | | |
|---|---|---|
| inp1 | 00 | (1000) |
| inp2 | 01 | (0100) |
| inp3 | 11 | (0010) |
| inp4 | 10 | (0001) |

| Constraint | Codes of symbolic values in constraint | Supercube | Codes of symbolic values not in constraint | Satisfied |
|---|---|---|---|---|
| 1100 | 00, 01 | | 11, 10 | |
| 0110 | 01, 11 | | 00, 10 | |
| 0111 | 01, 11, 10 | | 00 | |
| 0001 | 10 | | 00, 01, 11 | |

# Constructing a Cover

If an encoding satisfied ALL constraints a minimum cardinality encoded binary cover can be constructed directly

Just replace each multiple-valued implicant by its supercube

M-V Cover :

| | | |
|---|---|---|
| 10 | 110 | 01 |
| 11 | 011 | 01 |
| 01 | 111 | 10 |

Encoding :

| | |
|---|---|
| inp1 | 00 |
| inp2 | 01 |
| inp3 | 11 |

Minimized binary cover :

| | | |
|---|---|---|
| 10 | 0- | 01 |
| 11 | -1 | 01 |
| 01 | -- | 10 |

supercube ( )
supercube ( )
supercube ( )

Note: Used 10 as a don't-care!

# Optimal Encoding Strategy

Replace symbolic variable with multiple-valued variable

Minimize multiple-valued cover

Find encoding with minimum or minimal length that satisfies constraints given by multiple-valued cover $\leftarrow$

Replace each multiple-valued implicant by supercube corresponding to the encoding found

Resulting binary cover has minimum cardinality and exploits unused codes as don't cares