

Lab 4

.Part 1: Logic Minimizer

- Implement a logic minimizer in Java using the Tabulation Method, Quine McCluskey, and Petrick's Method algorithms

.Part 2: Static Hazards

- Fix static hazards given a set of transition-don't-cares

Basic Definitions

.Implicant:

- Product of one or more terms (e.g. A , AB)

.Two-level-sum-of-product expression:

- Sum of implicants (e.g. $AB + AC$)

.Minterm:

- Smallest decomposition of a minterm for a given number of variables (e.g. for 5 variables, $abcde$ or $a'bcde'$)

.Don't Care:

- For a given input combination does not matter if it outputs a 0 or a 1

Logic Minimization

.Can easily be accomplished with a Karnaugh Map for functions of up to 4 variables

wx\yz	00	01	11	10
00	1	1	0	X
01	1	1	0	0
11	0	1	0	0
10	1	0	0	1

.Simplifies to: $w'y' + x'z' + xy'z$

.Does not scale well for functions with more than 4 inputs

Prime Implicant Generation

- A minimal sum-of-products expression can be written as a sum of prime implicants
- Basic algorithm for finding prime implicants:
 - Start out with minterms
 - Iteratively combine minterms and don't cares that are Hamming distance 1 into implicants
 - Iteratively combine resulting implicants that are Hamming distance 1
 - Continue iterating until nothing can be combined

Tabulation Method

- Reduce number of comparisons by grouping minterms, don't cares, and implicants based on number of 1's
 - Minterms and implicants with the same number of 1's will never be Hamming distance 1
 - Minterms and implicants that differ in the number of 1's by more than 1 will never be Hamming distance 1
- Each group with n 1's will only need to be compared with groups with $n-1$ 1's and $n+1$ 1's

Tabulation Method

• Example: Gx is group with x 1's

- Partition the minterms and don't cares (0-subcubes) into groups

wx\yz	00	01	11	10
00	1	1	0	X
01	1	1	0	0
11	0	1	0	0
10	1	0	0	1



Groups	0-subcube
G0	0000
G1	0001 0010 0100 1000
G2	0101 1010
G3	1101

Tabulation Method

•Example:

- Group partitions into 1-subcubes, crossing out covered minterms

Groups	0-subcube	1-subcube
G0	0000	000- 00-0 0-00 -000
G1	0001 0010 0100 1000	010- -010 10-0 0-01
G2	0101 1010	-101
G3	1101	None

Tabulation Method

.Example:

- Group 1-subcubes into 2-subcubes, crossing out covered minterms

Groups	0-subcube	1-subcube	2-subcubes
G0	0000	000- 00-0 0-00 -000	0-0- -0-0
G1	0001 0010 0100 1000	010- -010 10-0 0-01	None
G2	0101 1010	-101	None
G3	1101	None	None

Quine McCluskey

- The set of prime implicants can be redundant
- We can remove redundant prime implicants by iteratively applying the following rules:
 - If prime implicant p is essential, any additional minterms covered by p do not need to be considered
 - If a prime implicant p covers a superset of minterms covered by prime implicant p_1 , p_1 can be eliminated from consideration
 - If the prime implicants that cover minterm m are a superset of the prime implicants that cover m_2 , then we do not need to consider m

Quine McCluskey

.Example

	0	1	2	3	4	5	6	8	10	11	12	15	19	22	23	24	26	27	30	31
b'cde'							x							x						
a'b'd'	x	x			x	x														
a'b'e'	x		x		x		x													
a'd'e'	x				x			x			x									
c'de				x						x			x					x		
bde										x		x						x		x
ade													x		x			x		x
acd														x	x				x	x
abd																	x	x	x	x
a'c'	x	x	x	x				x	x	x										
bc'								x	x	x						x	x	x		

Quine McCluskey

.Example

- $a'b'd'$, $a'd'e'$, bde , and bc' are essential

	0	1	2	3	4	5	6	8	10	11	12	15	19	22	23	24	26	27	30	31
$b'cde'$							x							x						
$a'b'd'$	x	x			x	x														
$a'b'e'$	x		x		x		x													
$a'd'e'$	x				x			x			x									
$c'de$				x						x			x					x		
bde										x		x						x		x
ade													x		x			x		x
acd														x	x				x	x
abd																	x	x	x	x
$a'c'$	x	x	x	x				x	x	x										
bc'								x	x	x						x	x	x		

- Cross out essential rows and columns
- Stored essentials: $a'b'd'$, $a'd'e'$, bde , bc'

Quine McCluskey

.Example

- Rinse and repeat
 - acd is essential
 - Stored essentials: a'b'd', a'd'e', bde, bc', acd

	2	3	6	19	22	23	30
b'cde'			x		x		
a'b'e'	x		x				
c'de		x		x			
ade				x		x	
acd					x	x	x
~~abd~~							~~x~~
a'c'	x	x					

Quine McCluskey

.Example

- Rinse and repeat
 - $a'b'e'$ and $c'de$ are essential
 - Stored essentials: $a'b'd'$, $a'd'e'$, bde , bc' , acd , $a'b'e'$, $c'de$

	2	3	6	19
$b'cde'$			x	
$a'b'e'$	x		x	
$c'de$		x		x
ade				x
$a'c'$	x	x		

Petrick's Method

•Applying the three Quine McCluskey rules may yield a cyclic core

•Example:

	0	1	2	5	6	7
P1 = $a'b'$	x	x				
P2 = $b'c$		x		x		
P3 = ac				x		x
P4 = ab					x	x
P5 = bc'			x		x	
P6 = $a'c'$	x		x			

•No row or column dominations

• What do we do?

Petrick's Method

• Formulate as product-of-sums:

	0	1	2	5	6	7
P1 = a'b'	x	x				
P2 = b'c		x		x		
P3 = ac				x		x
P4 = ab					x	x
P5 = bc'			x		x	
P6 = a'c'	x		x			

• $(P1+P6)(P1+P2)(P5+P6)(P2+P3)(P4+P5)(P3+P4)$

Petrick's Method

.Convert Product-of-sums to Sum-of-products

- $(P1+P6)(P1+P2)(P5+P6)(P2+P3)(P4+P5)(P3+P4) =$
- $(P1+P1P6+P1P2+P2P6)(P5+P6)(P2+P3)(P4+P5)(P3+P4) =$
- $(P1+P2P6)(P5+P6)(P2+P3)(P4+P5)(P3+P4) =$
- $P1P2P4P5 + \mathbf{P1P3P5} + P1P3P4P6 + P2P3P4P5 + \mathbf{P2P4P6} + P2P3P5P6$
- Bolded product terms indicate two options for a minimal cover

Representing Implicants

•Implicant encoding:

- Represent each variable as:
 - 00 = invalid
 - 01 = uncomplemented variable
 - 10 = complemented variable
 - 11 = both

•Examples:

- $ABC' = 01\ 01\ 10$
- $AC' = 01\ 11\ 10$

•An implicant with '00' for any variable can be replaced with an implicant with all 0's

Using The Encoding

.Subset definition:

- $S2$ is a subset of $S1$ if $S1 \cup S2 = S1$
- $S2$ is a subset of $S1$ if $S1 \cap S2 = S2$

.Use *bitwise-and* for intersection operator and *bitwise-or* for union operator

- $AB \cup ABC = 01\ 01\ 11 \mid 01\ 01\ 01 = 01\ 01\ 11 = AB$
- $AB \cap ABC = 01\ 01\ 11 \& 01\ 01\ 01 = 01\ 01\ 01 = ABC$

.So, we know that ABC is a subset of AB and can be eliminated

Using The Encoding

•Intersect BC and AB to get one potential part of decomposition:

- $BC \cap AB = 11\ 01\ 01 \ \& \ 01\ 01\ 11 = 01\ 01\ 01$

•Intersect BC and A'C to get another potential part of decomposition:

- $BC \cap A'C = 11\ 01\ 01 \ \& \ 10\ 11\ 01 = 10\ 01\ 01$

- $01\ 01\ 01 \wedge 10\ 01\ 01 = 11\ 00\ 00$

Using The Encoding

• Verify that these two compositions can be combined

- Implicants can be combined if they differ by 1 variable
 - e.g. $abc + a'bc = (a + a')bc = bc$
- 01 01 01 and 10 01 01 differ by one variable since bitwise-xor gives a pair of 1's in a variable location, while the rest of the variable locations are 0s:
 - $01\ 01\ 01 \wedge 10\ 01\ 01 = 11\ 00\ 00$

• Upon verification, combine the decompositions and see if you obtain bc:

- $01\ 01\ 01 \mid 10\ 01\ 01 = 11\ 01\ 01 = bc$

Implementation Guidelines

.Implicant.java

- Modifies the encoding by splitting the two bits into separate longs
 - Represents implicants with myMSB and myLSB
- Organization enables easy checks for
 - Differing by single variable
 - Seeing if an implicant is invalid
- You should maintain a list of minterms that each implicant covers

Implementation Guidelines

.BooleanExpression.java

- Constructs list of implicants based off of minterms
- evaluate method

.BitVector.java

- Efficient data structure for sets

Implementation Guidelines

.Prime Implicant Generation

- Start out with implicant objects that correspond to your minterms
- Perform Tabulation Method as you normally would
 - Check if Hamming distance 1 by using bitwise-xor and bitcount
 - Be sure to update the minterm sets that each implicant covers after combining them

Implementation Guidelines

.Quine McCluskey

- For each prime implicant, map to a set of minterms
 - You should already have a list of minterms that the prime implicant covers
 - Map list of minterms to a bitvector
 - 5-variable implicant with minterms $\{0,1,2,3,8,9,10,11\}$ will map to the following right-to-left indexed integer:
 - 000000000000000000000000111100001111

Implementation Guidelines

.Quine McCluskey

- For each minterm, map to a set of prime implicants
 - Should use bitvectors
 - Check subset relationship using union and/or intersection operators
 - Union is bitwise-or, intersection is bitwise-and

Implementation Guidelines

.Petrick's Method

- Encode implicant term as a bitvector such that:
 - A 1 is present at the *i*th index of the bitvector for prime implicant P_i
 - Example: $P_1 = 000001$, $P_2 = 000010$,
 $P_4 = 001000$

Implementation Guidelines

.Petrick's Method

- Sum term is a set of implicants:
 - $(P1 + P4) = \{000001, 001000\}$
- Product term is encoded as:
 - $P1P4 = 001001$
- Convert from product-of-sums to sum-of-products using a cartesian product with a bitwise-or
 - $(P1 + P4)(P2 + P3) = \{000001, 001000\} \times \{000010, 000100\}$
 - $P1P2 + P1P3 + P2P4 + P3P4 = \{000011, 000101, 001010, 001100\}$

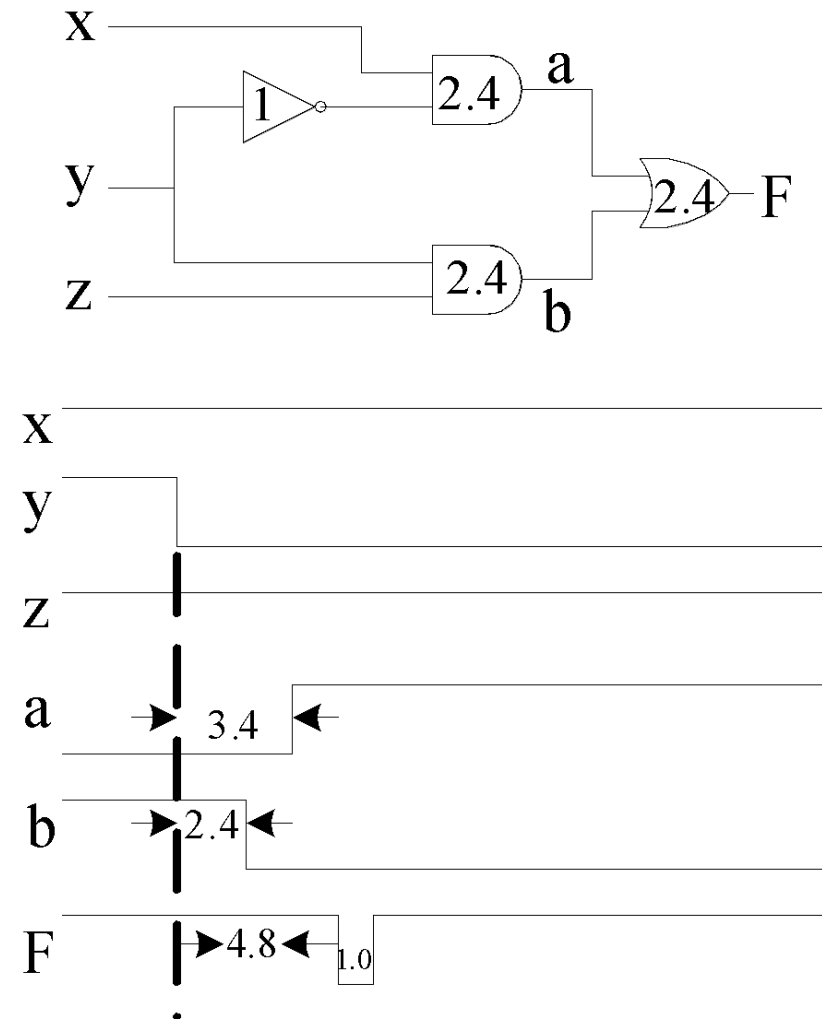
Part 1 Summary

- Implement a logic minimizer in Java
 - Modify Tabulation Method to handle *Don't Cares*
 - Implement Quine-McCluskey algorithm
 - Implement Petrick's Method algorithm
- Java Files
 - Main.java
 - BooleanExpression.java
 - Implicant.java
 - BitVector.java

Static Hazards

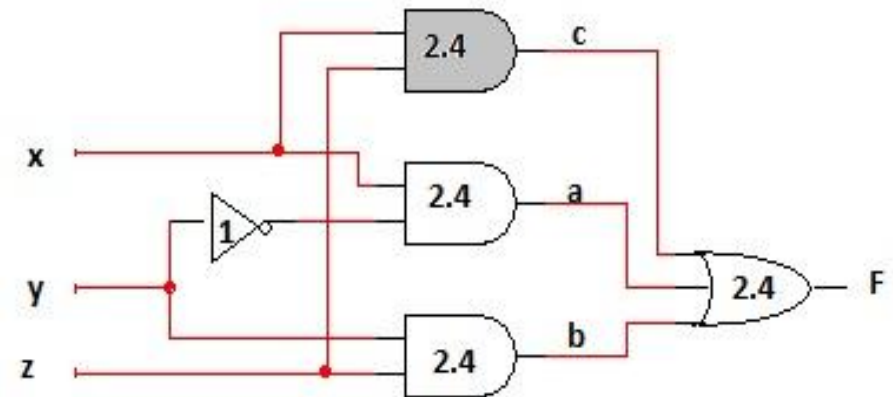
- Effect:** short unwanted glitches at the output
- Reason:** different propagation delays on converging paths

How do we fix it?



Static Hazards

- Extra gate as shown takes care of the hazard!



Minimal Hazard-free Circuit

- ***Transition Don't Cares:*** input transitions that will never occur during the regular operation of the circuit
- ***Reduce redundant hardware:*** remove the gates that were added for handling a static hazard if the transition that could have generated the static hazard is in the *Transition Don't Care* set.

Design 3 Circuits

- Circuit 1 and 2
 - Both the **Minimal** Implicant and the **Minimal** Implicate representations generate a hazards
 - Given a set of *transition don't cares*, implement a static hazard free design
- Circuit 3
 - Multiple naïve implementations
 - Only one makes the most use of the transition don't cares
 - Using the *transition don't cares* provided, implement a static hazard free design that minimizes the number of gates used
- Remember that static hazards only occur in one direction!