EECS151/251A Discussion 4

Christopher Yarp

Feb. 15, 2019

Plan for Today

- Show another Verilog trick
- Talk briefly about Boolean Algebra and Optimization
- Answer your questions!
- Experiment!
  - Break into groups to do a practice problem
  - Reconvene to discuss solution
  - Work on a problem as a class

The casez statement

Making HW2 Problem 2c Easier!

casez

- You want to use a case block but you want to include “don’t cares” / wildcards in your case statements instead of listing every possible case
- casez allows you to insert ‘?’ to signify don’t cares / wildcards

```
module circuit2_casez(a, y);
  input [3:0] a;
  output reg [1:0] y;
  always @(*)
    casez(a)
      4'b???1: y = 2'b11;
      4'b??10: y = 2'b10;
      4'b?100: y = 2'b01;
      4'b1000: y = 2'b00;
      default: y = a[1:0];
    endcase
endmodule
```

Boolean Algebra

Boolean Algebra: A mathematical way of looking at logic

- Basic operators: AND (·, ∧), OR (+, ∨), NOT (¬, ‘, !, ~, or “bar” – ex: $\bar{a}$)
- Like standard algebra, AND (·) takes precedence over OR (+). NOT takes precedence over AND
  - $a'b + bc = ((a') \cdot b) + (b \cdot c)$
- Like standard algebra, there are a set of laws that can be applied to Boolean expressions
  - We can use these laws to simplify expressions

Important Properties

- Many properties are listed in Lecture 6 Slides
  - Make a note of these properties, they will be useful!
  - Here is a short summary of some (but not all) of them:

<table>
<tr><td>$ab = bc$</td><td>$a + b = b + a$</td><td colspan="2">$a'' = a$</td></tr>
<tr><td>$(ab)c = a(bc)$</td><td>$(a + b) + c = a + (b + c)$</td><td>$a \cdot 0 = 0$</td><td>$a + 1 = 1$</td></tr>
<tr><td>$a(b + c) = ab + ac$</td><td>$a + bc = (a + b)(a + c)$</td><td>$a \cdot 1 = a$</td><td>$a + 0 = a$</td></tr>
<tr><td>$(a + b + \cdots + c)' = a'b' \cdots c'$</td><td>$(ab \cdots c)' = a' + b' + \cdots + c'$</td><td>$a \cdot a = a$</td><td>$a + a = a$</td></tr>
<tr><td colspan="2">$ab' + ab = a(b' + b) = a(1) = a$</td><td>$a \cdot \bar{a} = 0$</td><td>$a + \bar{a} = 1$</td></tr>
</table>

Useful Tricks

- You can introduce redundant terms
  - $AB + BC = AB + BC + BC$
  - Why would you want to do this? To introduce terms from which you can factor

Canonical Forms

- A Boolean expression can be converted to a set of canonical forms
  - Sum of Product (SOP) is one canonical form consisting of the sum (OR) of a series of products (ANDs)
    - Ex: $abc + cd + efgh$
  - Product of Sum (POS) is another canonical form consisting of the product (AND) of a series of sums (ORs)
    - Ex: $(x + y + z)(z + h)(h + i + j + k)$
- There are methods to derive the optimal 2 level logic expression (in SOP or POS form)
  - K-maps are a tool which we can use by hand to find these simplified expressions

K-Maps

- Method by which we can more easily observe adjacencies in the truth table
- By constructing large rectangles that are even powers of 2, we can derive the minimal SOP or POS expression

CD\AB	00	01	11	10
00				
01				
11				
10				

Example K-Map for 4 Input Expression

K-Maps

	$A'B'$	$A'B$	AB	AB'
CD\AB	00	01	11	10
00				
01				
11				
10				

	CD\AB	00	01	11	10
$C'D'$	00				
$C'D$	01				
CD	11				
CD'	10				

K-Maps

	A'		A	
CD\AB	00	01	11	10
00				
01				
11				
10				

	CD\AB	00	01	11	10
C'	00				
	01				
C	11				
	10				

	B'	B		B'
CD\AB	00	01	11	10
00				
01				
11				
10				

	CD\AB	00	01	11	10
D'	00				
D	01				
	11				
D′	10				

Finite State Machines (FSMs)

Finite State Machines (FSMs)

- Allows us to design/model complex systems by viewing a system as having a set of possible states it can be in
  - The machine can only be in one state at a time
  - There are rules dictating how the machine moves between states
  - The output is either based solely on the current state (Moore style), the current state and current inputs (Mealy style), or a combination of these
- Very common in digital logic
  - Often used to design “control logic”
    - ASIC and FPGA labs will both be using FSMs like this
  - So common that many EDA tools (including Vivado) have special optimization passes specifically for FSMs
- Can also be used in software, particularly in Real-Time Systems & Mechatronics

Group Work

Problem 1

- Simplify the following expressions using a k-map:

```
module circuit1(a, b, c, y z)
  input a, b, c;
  output y,z;

  assign y = a & b & c | a & b & ~c | a & ~b & c;
  assign z = a & b | ~a & ~b;
endmodule
```

- Use Boolean Algebra to transform the original expression into the simplified one from your k-map

Problem 1

y

		A'		A	
		$A'B'$	$A'B$	AB	AB'
	C\AB	00	01	11	10
C'	0			1	
C	1			1	1

Simplified Expr: $AB + AC$

$$ABC + ABC' + AB'C$$
$$= ABC + ABC' + ABC + AB'C$$
$$= AB(C + C') + AC(B + B')$$
$$= AB(1) + AC(1)$$
$$= AB + AC$$

z

		A'		A	
		$A'B'$	$A'B$	AB	AB'
	C\AB	00	01	11	10
C'	0	1		1	
C	1	1		1	

Simplified Expr: $AB + A'B'$

Unchanged