



San Francisco Bay University

EE461 Digital Design and HDL

Week#6 "if/case/casex/casez" Conditional Statements

1. Lab Outlines:

1. "if" Conditional statements
2. Continuous assignment with conditional operator
3. "case" conditional statement
4. "casex" conditional statements
5. "casez" conditional statements

2. Lab Procedures

Exercises

Show the example programs running results and explain why.

I. "if" Conditional statements

- if(a == yyyy) : For "==" or "!=" checking
if a has x or z, then (a == yyyy) = x (unknown) or (a != yyyy) = x (unknown).

Example:

```

module IfTest ();
    reg [3:0] X;
    reg [3:0] Y;
    initial begin
        X=4'b101x;
        Y=4'b101z;
        if(X==4'b101z)begin
            $display("Statement 1 has been selected!");
        end
        if(X==Y)begin
            $display("Statement 2 has been selected!");
        end
        if(X!=Y)begin
            $display("Statement 3 has been selected!");
        end
        if(X==X)begin
            $display("Statement 4 has been selected!");
        end
        $display("No one has been selected!");
    end
endmodule

```

Ans:

```

1 module IfTest();
2     reg [3:0] X;
3     reg [3:0] Y;
4     initial begin
5         X = 4'b101x;
6         Y = 4'b101z;
7
8         if (X == 4'b101z) begin
9             $display("Statement 1 has been selected!");
10        end
11        if (X == Y) begin
12            $display("Statement 2 has been selected!");
13        end
14        if (X != Y) begin
15            $display("Statement 3 has been selected!");
16        end
17        if (X == X) begin
18            $display("Statement 4 has been selected!");
19        end
20        $display("No one has been selected!");
21    end
22 endmodule
23

```

[2024-12-18 10:30:46 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
 No one has been selected!
 Done

CODE ANALYSIS

When we consider initial values

- X = 4'b101x: X contains an unknown bit (x) in the least significant position.
- Y = 4'b101z: Y contains a high-impedance bit (z) in the least significant position.

Breaking down conditions one-by-one

1. if (X == 4'b101z):
 - This compares X (101x) with the constant 4'b101z.
 - In Verilog, == operator treats x and z as unknowns.
 - The comparison evaluates to x (unknown)
 - Condition is considered false, and "Statement 1 has been selected!" is not displayed
2. if (X == Y):
 - This compares X (101x) with Y (101z).
 - Since the least significant bits (x and z) are different, the comparison evaluates to x (unknown)
 - Condition is false, and "Statement 2 has been selected!" is not displayed
3. if (X != Y):
 - This checks if X (101x) is not equal to Y (101z).
 - Since the equality check X == Y is unknown (x), the inequality also evaluates to x (unknown).
 - Condition is false, and "Statement 3 has been selected!" is not displayed
4. if (X == X):
 - This checks if X is equal to itself.
 - Normally, this would be true, but because X contains an x (unknown), the comparison evaluates to x (unknown).
 - Why Only "No one has been selected!"?
5. Since all the if conditions evaluate to false:
 - No statements inside the if blocks are executed.
 - The program defaults to displaying: \$display("No one has been selected!")

- if(a===yyyy): no matter what a's value is, even x or z, if LHS exactly matches RHS, (a===yyyy) = 1.

Example:

```
module IfTest1();
    reg [3:0] X;
    reg [3:0] Y;
    initial begin
        X=4'b101x;
        Y=4'b101z;
        if(X==4'b101z)begin
            $display("Statement 1 has been selected!");
        end
        if(X===Y)begin
            $display("Statement 2 has been selected!");
        end
        if(X!=Y)begin
            $display("Statement 3 has been selected!");
        end
        if(X===X)begin
            $display("Statement 4 has been selected!");
        end
        $display("No one has been selected!");
    end
endmodule
```

Ans:

The screenshot shows a Verilog code editor with the following code:

```
1 module IfTest1();
2     reg [3:0] X;
3     reg [3:0] Y;
4     initial begin
5         X = 4'b101x;
6         Y = 4'b101z;
7
8         if (X === 4'b101z) begin
9             $display("Statement 1 has been selected!");
10        end
11        if (X === Y) begin
12            $display("Statement 2 has been selected!");
13        end
14        if (X != Y) begin
15            $display("Statement 3 has been selected!");
16        end
17        if (X === X) begin
18            $display("Statement 4 has been selected!");
19        end
20        $display("No one has been selected!");
21    end
22 endmodule
23
```

Below the code editor, there is a simulation output window showing the following messages:

```
[2024-12-18 10:53:31 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && urbuffer vvp a.out
Statement 3 has been selected!
Statement 4 has been selected!
No one has been selected!
Done
```

CODE ANALYSIS

The === operator is used in Verilog to check bit-by-bit exact matching, including x (unknown) and z (high-impedance) values.

Considering initial values,

X = 4'b101x; X contains an unknown value (x) in the least significant bit.

Y = 4'b101z; Y contains a high-impedance value (z) in the least significant bit.

Breaking down conditions one-by-one

1. if (X === 4'b101z):
 - The === operator checks if X matches 4'b101z exactly, including x and z.
 - X = 4'b101x does not match 4'b101z because the least significant bit (x in X vs. z in 4'b101z) is different.
 - Condition is false and display of "Statement 1 has been selected!" is not executed.
2. if (X === Y):
 - The === operator checks if X and Y are exactly the same.
 - X = 4'b101x and Y = 4'b101z are different in their least significant bits (x vs. z).
 - Condition is false and display of "Statement 2 has been selected!" is not executed.
3. if (X !== Y):
 - The !== operator checks if X and Y are not exactly the same.
 - Since X = 4'b101x and Y = 4'b101z are different in their least significant bits (x vs. z), they are not exactly the same.
 - Condition Result is true and display of "Statement 3 has been selected!" is executed.
4. if (X === X):
 - The === operator checks if X is exactly the same as itself.
 - Since X is compared to itself, all bits match, even the x in the least significant bit.
 - Condition Result is true and display of "Statement 4 has been selected!" is executed.
5. Since conditions 3 and 4 are true, the final output:
 - Statement 3 has been selected!
 - Statement 4 has been selected!

II. Continuous assignment with conditional operator

- If a has x or z in conditional operation, then $a = x$ instead of other values in assignment.

Example:

```

module CondOp();
    reg [3:0] X;
    reg [3:0] Y;
    wire Z;
    assign Z=(X==Y)? 1'b1:1'b0;
    //assign Z=(X===Y)? 1'b1:1'b0;
    //assign Z=(X!==Y)? 1'b1:1'b0;
    initial begin
        X=4'b101x;
        Y=4'b101z;
        $monitor("Z's values: %d\n", Z);
    end
endmodule

```

Ans:

```

1 module CondOp();
2     reg [3:0] X;
3     reg [3:0] Y;
4     wire Z;
5     assign Z=(X==Y)? 1'b1:1'b0;
6     //assign Z=(X===Y)? 1'b1:1'b0;
7     //assign Z=(X!=Y)? 1'b1:1'b0;
8     initial begin
9         X=4'b101x;
10        Y=4'b101z;
11        $monitor("Z's values: %d\n", Z);
12    end
13 endmodule
14

```

Log Share

[2024-12-18 12:01:27 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out

Z's values: x

Done

CODE ANALYSIS

Considering variable initialization:

X = 4'b101x;

Y = 4'b101z;

- X has an **unknown bit** (x) in the least significant bit (LSB).
- Y has a **high-impedance bit** (z) in the LSB.

For assign Z = (X == Y) ? 1'b1 : 1'b0;

- The ternary operator evaluates (X == Y).
- If (X == Y) is **true**, Z is assigned 1'b1.
- If (X == Y) is **false**, Z is assigned 1'b0.

1. The == operator in Verilog performs bitwise comparison but treats x and z as unknowns as follows:

Bit Position	X Bit	Y Bit	Comparison
3 (MSB)	1	1	Equal
2	0	0	Equal
1	1	1	Equal
0 (LSB)	X	Z	Unknown (x)

The LSB comparison (x vs. z) results in an unknown (x). Since the comparison must return a definitive result (either true or false), the overall result of (X == Y) is unknown (x)

2. The ternary operator checks the value of (X == Y) for assign Z = (X == Y) ? 1'b1 : 1'b0;
 - If (X == Y) is 1 (true), Z is assigned 1'b1.
 - If (X == Y) is 0 (false), Z is assigned 1'b0.
 - If (X == Y) is x (unknown), the result of the ternary operator is x.
 - So, Z is assigned x because (X == Y) evaluates to x
3. The \$monitor statement continuously tracks changes in Z. Since Z is assigned the value x, the output is "Z's values: x" because:
 - The == operator can't handle x or z values in a way that resolves to true or false
 - Any comparison involving x or z results in unknown (x), making the entire condition of the ternary operator evaluate to unknown.
 - In Verilog, assigning x to a wire results in x being propagated through the simulation.

III. "case" conditional statement

- "case" is to match exactly for 1, 0, x and z

Example:

```
module caseTest();
    reg [3:0] X;
    reg [3:0] Y;
    initial begin
        X=4'b101x;
        Y=4'b101z;
        case(X)
            4'b1010: $display("Statement 1 has been selected!");
            4'b101x: $display("Statement 2 has been selected!");
            4'b101z: $display("Statement 3 has been selected!");
            4'bxxxx: $display("Statement 4 has been selected!");
            4'bzzzz: $display("Statement 5 has been selected!");
            default: $display("Default has been selected!");
        endcase
    end
endmodule
```

Ans:

The screenshot shows a Verilog code editor with the following code:

```
1 module caseTest();
2     reg [3:0] X;
3     reg [3:0] Y;
4     initial begin
5         X=4'b101x;
6         Y=4'b101z;
7         case(X)
8             4'b1010: $display("Statement 1 has been selected!");
9             4'b101x: $display("Statement 2 has been selected!");
10            4'b101z: $display("Statement 3 has been selected!");
11            4'bxxxx: $display("Statement 4 has been selected!");
12            4'bzzzz: $display("Statement 5 has been selected!");
13            default: $display("Default has been selected!");
14        endcase
15    end
16 endmodule
17
```

Below the code editor, the simulation output is shown:

```
[2024-12-18 12:21:39 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
Statement 2 has been selected!
```

The output indicates that the second statement was selected, which corresponds to the case 4'b101x in the code.

CODE ANALYSIS

The case statement in Verilog compares X against the listed cases. Verilog treats x and z values differently depending on the following context:

- Exact Matching in case:
 - A case branch matches if the value of X is identical to the case literal, including x or z.
 - First branch does not match. 4'b101x from second branch matches only 4'b101x (not 4'b101z)
 - So "Statement 2 has been selected!" is executed and displayed as output
- Branch Priority:
 - The case statement evaluates branches in order sequentially, from top to bottom
 - The first matching branch is selected and executed, and subsequent branches are ignored and not executed
- Default Case:
 - The default branch is only executed if no branches match

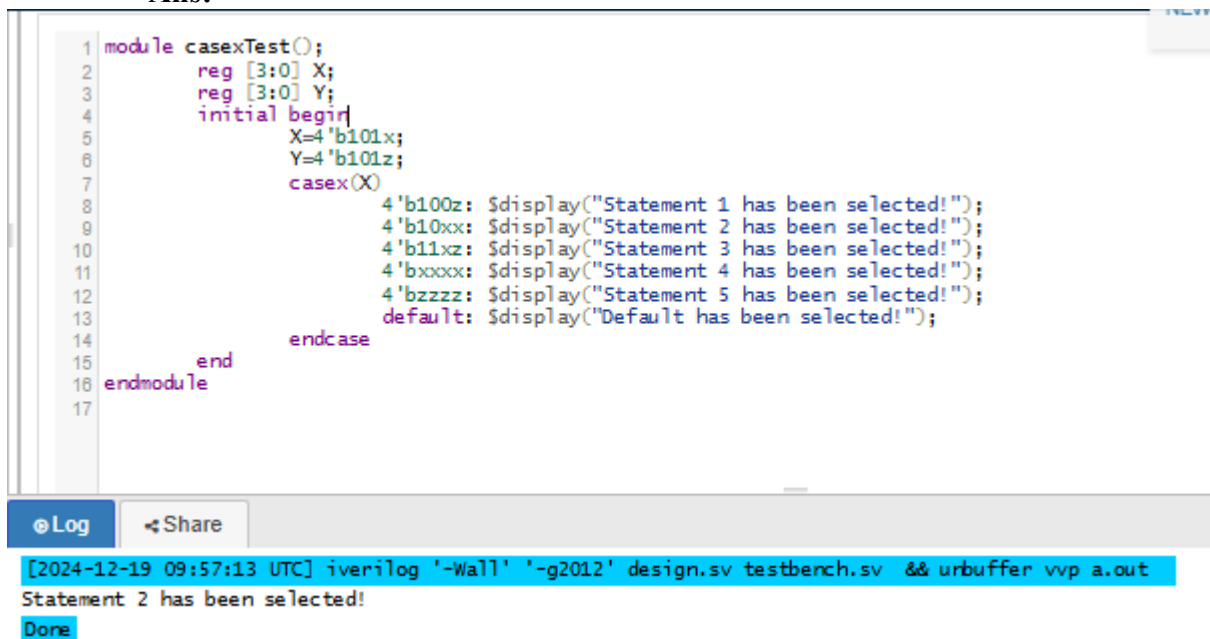
IV. "casex" conditional statement

- "casex" is to make x/z match 1, 0, x and z

Example:

```
module casexTest();
    reg [3:0] X;
    reg [3:0] Y;
    initial begin
        X=4'b101x;
        Y=4'b101z;
        casex(X)
            4'b100z: $display("Statement 1 has been selected!");
            4'b10xx: $display("Statement 2 has been selected!");
            4'b11xz: $display("Statement 3 has been selected!");
            4'bxxxx: $display("Statement 4 has been selected!");
            4'bzzzz: $display("Statement 5 has been selected!");
            default: $display("Default has been selected!");
        endcase
    end
endmodule
```

Ans:



The screenshot shows a Verilog code editor with the following code:

```
1 module casexTest();
2     reg [3:0] X;
3     reg [3:0] Y;
4     initial begin
5         X=4'b101x;
6         Y=4'b101z;
7         casex(X)
8             4'b100z: $display("Statement 1 has been selected!");
9             4'b10xx: $display("Statement 2 has been selected!");
10            4'b11xz: $display("Statement 3 has been selected!");
11            4'bxxxx: $display("Statement 4 has been selected!");
12            4'bzzzz: $display("Statement 5 has been selected!");
13            default: $display("Default has been selected!");
14        endcase
15    end
16 endmodule
17
```

Below the code editor, the simulation output is shown:

```
[2024-12-19 09:57:13 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
Statement 2 has been selected!
Done
```

CODE ANALYSIS

casex treats x and z as wildcards, allowing them to match any value (0, 1, x , or z), and matches the first branch that satisfies the condition.

1. Branch Evaluation: $X = 101x$ matches $4'b10xx$ because:
 - The first two bits match exactly (10).
 - The last two bits match due to wildcards (xx).
2. Priority:
 - The order of branches matters. Once a match is found, subsequent branches are ignored.

- Note: in this case, x looks like wildcard (?=1/0/x/z). So, we can write as follows:

Example:

```
module casex1();
    reg [3:0] X;
    reg [3:0] Y;
    initial begin
        X=4'b101x;
        Y=4'b101z;
        casex(Y)
            4'b???1: $display("Statement 1 has been selected!");
            4'b???1?: $display("Statement 2 has been selected!");
            4'b?1??: $display("Statement 3 has been selected!");
            4'b1???: $display("Statement 4 has been selected!");
            default: $display("Default has been selected!");
        endcase
    end
endmodule
```

Ans:

The screenshot shows a Verilog code editor with the following code:

```
1 module casex1();
2     reg [3:0] X;
3     reg [3:0] Y;
4     initial begin
5         X=4'b101x;
6         Y=4'b101z;
7         casex(Y)
8             4'b???1: $display("Statement 1 has been selected!");
9             4'b???1?: $display("Statement 2 has been selected!");
10            4'b?1??: $display("Statement 3 has been selected!");
11            4'b1???: $display("Statement 4 has been selected!");
12            default: $display("Default has been selected!");
13        endcase
14    end
15 endmodule
16
```

Below the code editor, there is a simulation output window showing the following text:

```
[2024-12-19 10:03:04 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && unbuffer vvp a.out
Statement 1 has been selected!
Done
```

CODE ANALYSIS

The casex statement treats both x (unknown) and z (high-impedance) as wildcards, meaning they match any value (0, 1, x, or z). This makes casex less strict compared to a case statement.

In this example, we compare Y (4'b101z) against patterns that include wildcards (?), which further expands the match criteria.

- Wildcard Matching in casex:
 - Both x and z are treated as wildcards, matching any value (0, 1, x, or z).
- Branch Priority: The casex statement evaluates branches in order
 - In Branch 1 the pattern ???1 matches any value with an LSB of 1.
 - Since z in the LSB of Y is treated as a wildcard, it matches
 - Once Branch 1 is matched, execution stops after the first match, and no further comparisons are performed
- Output Display : X = 101x matches 4'b10xx because:
 - The first matching branch (4'b???1) is selected
 - So the output is "Statement 1 has been selected!"

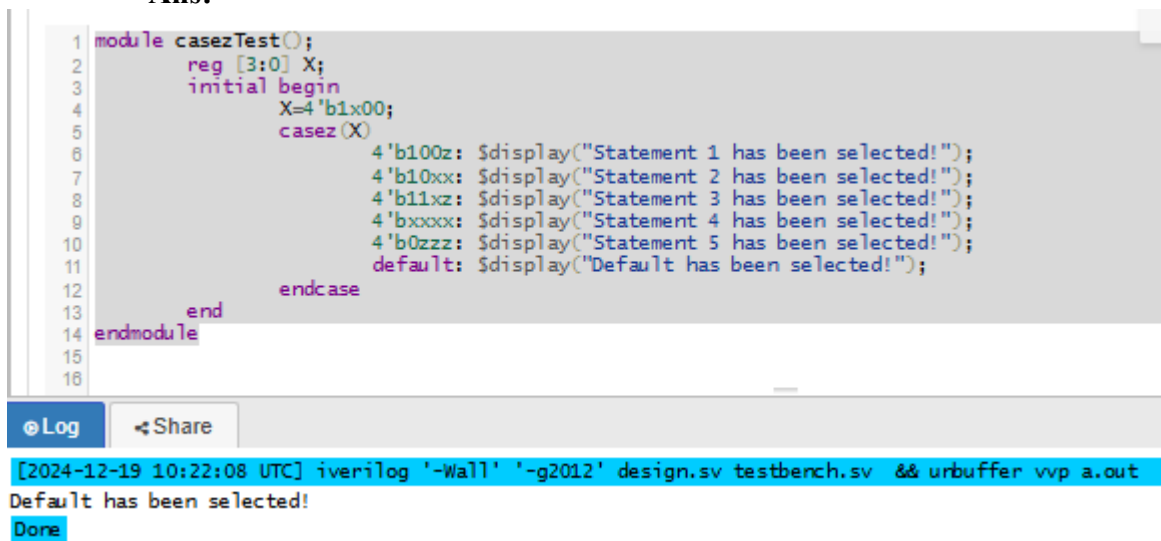
V. "casez" conditional statement

- "casez" is to make z match 1, 0 and z, but not x.

Example 1:

```
module casezTest();
    reg [3:0] X;
    initial begin
        X=4'b1x00;
        casez(X)
            4'b100z: $display("Statement 1 has been selected!");
            4'b10xx: $display("Statement 2 has been selected!");
            4'b11xz: $display("Statement 3 has been selected!");
            4'bxxxx: $display("Statement 4 has been selected!");
            4'b0zzz: $display("Statement 5 has been selected!");
            default: $display("Default has been selected!");
        endcase
    end
endmodule
```

Ans:



The screenshot shows a Verilog code editor with the following code:

```
1 module casezTest();
2     reg [3:0] X;
3     initial begin
4         X=4'b1x00;
5         casez(X)
6             4'b100z: $display("Statement 1 has been selected!");
7             4'b10xx: $display("Statement 2 has been selected!");
8             4'b11xz: $display("Statement 3 has been selected!");
9             4'bxxxx: $display("Statement 4 has been selected!");
10            4'b0zzz: $display("Statement 5 has been selected!");
11            default: $display("Default has been selected!");
12        endcase
13    end
14 endmodule
```

Below the code editor, there is a simulation output window showing the command: `[2024-12-19 10:22:08 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && urbuffer vvp a.out`. The output shows "Default has been selected!" and a "Done" button.

- In "casez" block, wildcard ? = 1/0/x/z.

CODE ANALYSIS

The casez statement treats z (high-impedance) as a wildcard that matches any value (0, 1, or z). However, it treats x (unknown) as a literal, meaning x must match exactly with x in the case pattern. This behavior is how casez is different from casex - where both x and z are treated as wildcards.

- The casez statement evaluates branches in order, and stops at the first match.
- No branches, from branch 1 to branch 5, match $X = 4'b1x00$ due to the x in the second bit, which does not align with the provided patterns.
- No match is found in any branch, so the default case is executed
- Output displays "Default has been selected!"

Example 2:

```

module casez1();
    reg [3:0] X;
    initial begin
        X=4'b1x00;
        casez(X)
            4'b???1: $display("Statement 1 has been selected!");
            4'b??1?: $display("Statement 2 has been selected!");
            4'b?1??: $display("Statement 3 has been selected!");
            4'b1??? : $display("Statement 4 has been selected!");
            default: $display("Default has been selected!");
        endcase
    end
endmodule

```

Ans: ?

Notice that "casez" and "casex" should be avoided considering synthesis and other issues, if you can use simple "case" or "if" chains. If you can't, use "casez" instead of "casex".

The screenshot shows a Verilog code editor with the following code:

```

1 module casez1();
2     reg [3:0] X;
3     initial begin
4         X=4'b1x00;
5         casez(X)
6             4'b???1: $display("Statement 1 has been selected!");
7             4'b??1?: $display("Statement 2 has been selected!");
8             4'b?1??: $display("Statement 3 has been selected!");
9             4'b1??? : $display("Statement 4 has been selected!");
10            default: $display("Default has been selected!");
11        endcase
12    end
13 endmodule
14

```

Below the code editor, there is a terminal window showing the output of the simulation:

```

[2024-12-19 10:41:58 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv && urbuffer vvp a.out
Statement 4 has been selected!
Done

```

CODE ANALYSIS

1. casez Behavior:
 - Again, casez treats z and ? as wildcards.
 - While casez treats x as a literal, meaning it must explicitly match an x in the pattern.
2. Branch Priority:
 - The casez statement evaluates branches in order, stopping at the first match.
3. Matches: No branches, from Branch 1 to Branch 3, and then in Branch 4:
 - The pattern 4'b1??? only requires MSB to match 1, while the other three bits to match any value, because of the wildcards ???
 - So in Branch 4, the pattern 1??? matches any value with an MSB of 1, regardless of the other bits
 - This is the first match that happens so it is selected
 - The default branch is not executed, since it is only executed if none of the preceding branches match.
4. Output Display:
 - Case for Branch 4 is executed
 - Output displays "Statement 4 has been selected!"

VI. Exercises

- Show the example program running results and explain why.

All program results are shown with explanations as to why.