# Verilog In One Day Part-II

Feb-9-2014





## Control Statements

Wait, what's this? **if, else, repeat, while, for, case** - it's Verilog that looks exactly like C (and probably whatever other language you're used to program in)! Even though the functionality appears to be the same as in C, Verilog is an HDL, so the descriptions should translate to hardware. This means you've got to be careful when using control statements (otherwise your designs might not be implementable in hardware).

## If-else

If-else statements check a condition to decide whether or not to execute a portion of code. If a condition is satisfied, the code is executed. Else, it runs this other portion of code.

```
1  // begin and end act like curly braces in C/C++.
2  if (enable == 1'b1) begin
3    data = 10; // Decimal assigned
4    address = 16'hDEAD; // Hexadecimal
5    wr_enable = 1'b1; // Binary
6  end else begin
7    data = 32'b0;
8    wr_enable = 1'b0;
9    address = address + 1;
10 end
```

You could download file one_day2.v here

One could use any operator in the condition checking, as in the case of C language. If needed we can have nested if else statements; statements without else are also ok, but they have their own problem, when modeling combinational logic, in case they result in a Latch (this is not always true).

## Case

Case statements are used where we have one variable which needs to be checked for multiple values. like an address decoder, where the input is an address and it needs to be checked for all the values that it can take. Instead of using multiple nested if-else statements, one for each value we're looking for, we use a single case statement: this is similar to switch statements in languages like C++.

Case statements begin with the reserved word **case** and end with the reserved word **endcase** (Verilog does not use brackets to delimit blocks of code). The cases, followed with a colon and the statements you wish executed, are listed within these two delimiters. It's also a good idea to have a **default** case. Just like with a finite state machine (FSM), if the Verilog machine enters into a non-covered statement, the machine hangs. Defaulting the statement with a return to idle keeps us safe.

```
1 case(address)
2   0 : $display ("It is 11:40PM");
3   1 : $display ("I am feeling sleepy");
4   2 : $display  ("Let me skip this tutorial");
5   default : $display  ("Need to complete");
6 endcase
```

You could download file one_day3.v here

Looks like the address value was 3 and so I am still writing this tutorial.

**Note:** One thing that is common to if-else and case statement is that, if you don't cover all the cases (don't have 'else' in If-else or 'default' in Case), and you are trying to write a combinational statement, the synthesis tool will infer Latch.

### While

A while statement executes the code within it repeatedly if the condition it is assigned to check returns true. While loops are not normally used for models in real life, but they are used in test benches. As with other statement blocks, they are delimited by begin and end.

```
1 while (free_time) begin
2  $display ("Continue with webpage development");
3 end
```

You could download file one_day4.v here

As long as free_time variable is set, code within the begin and end will be executed. i.e print "Continue with web development". Let's looks at a stranger example, which uses most of Verilog constructs. Well, you heard it right. Verilog has fewer reserved words than VHDL, and in this few, we use even lesser for actual coding. So good of Verilog... so right.

```
1 module counter (clk,rst,enable,count);
2 input clk, rst, enable;
3 output [3:0] count;
4 reg [3:0] count;
5
6 always @ (posedge clk or posedge rst)
```

```
 7 if (rst) begin
 8   count <= 0;
 9 end else begin : COUNT
10   while (enable) begin
11     count <= count + 1;
12     disable COUNT;
13   end
14 end
15
16 endmodule
```

You could download file one_day5.v here

The example above uses most of the constructs of Verilog. You'll notice a new block called always - this illustrates one of the key features of Verilog. Most software languages, as we mentioned before, execute sequentially - that is, statement by statement. Verilog programs, on the other hand, often have many statements executing in parallel. All blocks marked always will run - simultaneously - when one or more of the conditions listed within it is fulfilled.

In the example above, the always block will run when either rst or clk reaches a **positive edge** - that is, when their value has risen from 0 to 1. You can have two or more **always** blocks in a program going at the same time (not shown here, but commonly used).

We can disable a block of code, by using the reserve word disable. In the above example, after each counter increment, the COUNT block of code (not shown here) is disabled.

### For loop

For loops in Verilog are almost exactly like for loops in C or C++. The only difference is that the ++ and -- operators are not supported in Verilog. Instead of writing i++ as you would in C, you need to write out its full operational equivalent, i = i + 1.

```
1       for (i = 0; i < 16; i = i +1) begin
2               $display ("Current value of i is %d", i);
3       end
```

You could download file one_day6.v here

This code will print the numbers from 0 to 15 in order. Be careful when using for loops for register transfer logic (RTL) and make sure your code is actually sanely implementable in hardware... and that your loop is not infinite.

### Repeat

Repeat is similar to the for loop we just covered. Instead of explicitly specifying a variable and incrementing it when we declare the for loop, we tell the program how many times to run

through the code, and no variables are incremented (unless we want them to be, like in this example).

```
1 repeat (16) begin
2   $display ("Current value of i is %d", i);
3   i = i + 1;
4 end
```

You could download file one_day7.v here

The output is exactly the same as in the previous for-loop program example. It is relatively rare to use a repeat (or for-loop) in actual hardware implementation.

## Summary

- While, if-else, case(switch) statements are the same as in C language.
- If-else and case statements require all the cases to be covered for combinational logic.
- For-loop is the same as in C, but no ++ and -- operators.
- Repeat is the same as the for-loop but without the incrementing variable.

## Variable Assignment

In digital there are two types of elements, combinational and sequential. Of course we know this. But the question is "How do we model this in Verilog ?". Well Verilog provides two ways to model the combinational logic and only one way to model sequential logic.

- Combinational elements can be modeled using assign and always statements.
- Sequential elements can be modeled using only always statement.
- There is a third block, which is used in test benches only: it is called Initial statement.

## Initial Blocks

An initial block, as the name suggests, is executed only once when simulation starts. This is useful in writing test benches. If we have multiple initial blocks, then all of them are executed at the beginning of simulation.

### Example

```
1 initial begin
2             clk = 0;
3             reset = 0;
4             req_0 = 0;
5             req_1 = 0;
6 end
```

You could download file one_day8.v here

In the above example, at the beginning of simulation, (i.e. when time = 0), all the variables inside the begin and end block are driven zero.

Go on to the next page for the discussion of assign and always statements.






Do you have any Comment? mail me at:deepak@asic-world.com