Tutorials Home

VERILOG BASIC TUTORIAL

Verilog Introduction

Installing Verilog and Hello World

Simple comparator Example

Code Verification

Simulating with verilog

Verilog Language and Syntax

Verilog Syntax Contd..

Verilog Syntax - Vector Data

Verilog $monitor

Verilog Gate Level Modeling

Verilog UDP

Verilog Bitwise Operator

Viewing Waveforms

Full Adder Example

Multiplexer Example

Always Block for Combinational ckt

if statement for Combinational ckt

Case statement for Combinational ckt

Hex to 7 Segment Display

casez and casex

full case and parallel case

Verilog for loop

Verilog localparam and parameter

SEQUENTIAL CKT TUTORIAL

Sequential Circuits

Serial Shift Register

Binary Counters

Ring Counter

MISC VERILOG TOPICS

Setting Path Variable

Verilog Tutorial Videos

Verilog Interview questions #1

Verilog Interview questions #2

Verilog Interview questions #3

Verilog Books

Synchronous and Asynchronous Reset

Verilog Always block

Always block

An always block is used to execute a block of statements depending upon if the values of any of the inputs to the block (called a sensitivity list) changes.

As usual we will first give an example and then give explanation. Let us rewrite the 1 bit comparator we had studied earlier, using always block.

```
module comparator(
    input x,
    input y,
    output reg z
    );

reg p,q;
always @(x,y)
begin

p = (~x & ~y);
q =  x & y;

z = p |q;
end

endmodule
```

We have used always block as follows

always @(x,y);

This essentially means that the code follwed by always statement will run ONLY when the values of the variable x or y changes. We can have more than two variables in the sensitivity list.

One more thing to notice is that the output z has been assigned a reg and not a wire. However, this register will not be really synthesezes to actual register since its value is simple assignment statement depending upon two inputs.

A register variable in verilog simply means a variable that can "hold" value.

The statements in the assigment block are executed sequentially and order does matter. So we will have to be careful about the order.

Multiple statements within the always block are surrounded by begin and end.

The assign statement could also be used as `always @*`

The stimulus block stays the same and it is being reproduced below.

```

`timescale 1ns / 1ps
module stimulus;
        // Inputs
        reg x;
        reg y;
        // Outputs
        wire z;
        // Instantiate the Unit Under Test (UUT)
        comparator uut (
                .x(x),
                .y(y),
                .z(z)
        );

        initial begin
```

Left and Right shift << and >>

Negative Numbers

Blocking Vs Non Blocking

wand and wor

delay in verilog

$dumpfile and $dumpvars

Useful Resources

Verilog Examples

VERILOG QUIZS

Verilog Quiz # 1

Verilog Quiz # 2

Verilog Quiz # 3

Verilog Quiz # 4

Verilog Quiz # 5

Verilog Quiz # 6

Verilog Quiz # 7

Verilog Quiz # 8

Verilog Quiz # 9

OTHER TUTORIALS

Verilog Simulation with Xilinx ISE

VHDL Tutorial

```
        $dumpfile("test.vcd");
    $dumpvars(0,stimulus);
                // Initialize Inputs
                x = 0;
                y = 0;

        #20 x = 1;
        #20 y = 1;
        #20 y = 0;
        #20 x = 1;
        #40 ;

        end

                initial begin
                 $monitor("t=%3d x=%d,y=%d,z=%d \n",$time,x,y,z  );
                 end

endmodule

```

As usual we get the following result when we compile and run it.

```
t=  0 x=0,y=0,z=1

t= 20 x=1,y=0,z=0

t= 40 x=1,y=1,z=1

t= 60 x=1,y=0,z=0
```

The verilog always statement could also be written as

```
always @( a or b or c)
```

which is equivalent to

```
always @( a , b , c)
```

Thumb Rule for always block in combinatorial block In order to create Verilog code that can generate synthesizable circuit, all inputs to the hardware must appear in the sensitivity list. If it does not, then a latch will result in place of a combinatorial logic.

Exercise
1. Extend the full adder example presented earlier using always block. Check it using the same stimulus.

< Previous Next >

Copyright © 2009 Reference Designer

Tutorials | Products | Services | Contact Us