Verilog - Revision Notes

Prepared by : Kitu K Patel.

Imp: Apologies for my handwriting. Engineering taught me logic, not calligraphy. ☺

Important Topics in Verilog:

i) Abstraction level
ii) Modules & Module Instantiation
iii) Data types
iv) Port assignment
v) Numbers
vi) Vectors
vii) Parameters
viii) String
ix) Operators
x) Gate delay
xi) Assignment
xii) Delay
xiii) function
xiv) Task
xv) System Task.

1) Abstraction Level.

There are four level of abstraction in Verilog
→ Switch level
→ Gate level
→ Dataflow level
→ Behavioral level.

a) Switch level

→ It's the lowest level of abstraction.
→ It's very complex to model your RTL using switch level because here you've to model your RTL using pmos/nmos.

```
eg. module Kittu (out, in);
      output out;
      input in;

      supply1 Vdd;
      supply0 Gnd;

      pmos (out, Vdd, in); //conduct when in=0
      nmos (out, Gnd, in); //conduct when in=1

    endmodule.
```

b) Gate level Modelling

→ It describes the circuit using logic Gates.
→ There are in-built verilog primitives are there like and, or, xor, xnor, not, nand, nor
→ Close to hardware structure

eg. module kittu (a, b, out);
input a, b;
output out;

and a1 (out, a, b);
endmodule.

Logical Questions.

Let a = 1'b1 and b = 1'b0
and gate => out = 0
or gate => out = 1
xor gate => out = 1.
xnor gate => out = 0
nand gate => out = 1
nor gate => out = 0
not gate => out = z {If we pass two i/p}
↳ You will get z but it also depends on compiler. In synopsys you'll get "z" whereas in Questa or Cadence they are just performing an and operation a' between a & b & then gives an invert output.

buf is also there in verilog primitive it act as an buffer.

c) Data flow level modelling
→ Describes how data flow from input to output
→ Uses continuous assignment (assign)

→ Mainly used to model your combinational logic.

eg.
```
module Kittu (out, a, b);
    output out;
    input a, b;

assign out = a & b;
    endmodule.
```

out should be either of reg or wire.

d) Behavioural Level Modelling
→ It is used to describe functionality / behaviour of circuit
→ Higher level of abstraction

eg.
```
module Kittu (out, a, b);
    input a, b;
    output reg y;

    always @(*)
        y = a & b;

    endmodule.
```

Imp: y should always be of reg datatype if u use wire then compiler throw an error.

2) Module & Module Instantiation.

A module is a basic building block of any hardware design.

It can be instantiated as well.
Every Verilog design must have atleast one module.

Structure:

```
module <module_name> (port_list);
  //port_declaration;
  // Internal_logic;
endmodule.
```

eg. Module Instantiation.

```
module 2_to_1 (i0, i1, sel, out);
   input i0, i1, sel;
   output reg out;

  always @(*)
       out = sel ? i1 : i0;
  endmodule.

module 4_to_1 (i0, i1, i2, i3, s1, s0, out);
  input i0, i1, i2, i3, s1, s0;
     output reg out;
     wire w1, w2;
  mux 2_to_1 m1 (i0, i1, s1, w1);
  mux 2_to_1 m2 (i2, i3, s1, w2);
  mux 2-to-1 m3 (w1, w2, s0, out);
endmodule.
```

3) Data types

a) Net Datatypes

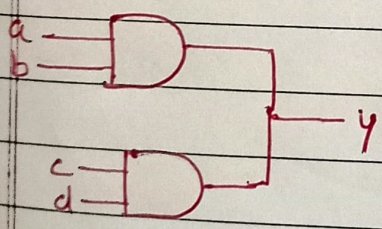
→ It represent physical connection.
→ It does not store any value.
→ Mostly used in continuous assignment.

→ Default Value => z

Net datatypes include => wire, wand, wor

In order to remove multi driver situation we ues wor.

Problem

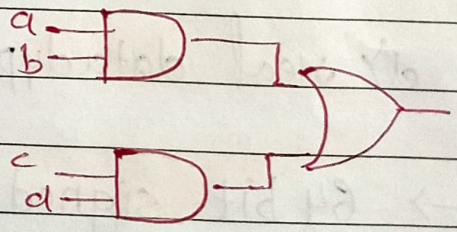


If u use wire

eg:
wire y;
assign y = a;
assign y = b;

Create Problem

Solution.



If you use wor.

eg.
wor y;
assign y = a;
assign y = b;

Solved

uwire (unresolved wire)
→ Allow only one driver
→ Multiple driver → Compile Error

b) reg datatype

→ Store value until next assignment
→ Use inside always & initial
→ Synthesizable.

Default Value => x

c) integer datatype

→ 32 bit signed variable
→ Default value => x
→ Mainly used in Counters

d) real datatype.

→ 64 bit signed variable
→ Used to model delay
→ Non-synthesizable
→ Default Value => 0

e) time datatype.

→ 64 bit unsigned variable
→ Store simulation time
→ Non-synthesizable.

f) realtime datatype

Same as real.
Used specifically for time value with fraction.

Default Value: 0.0000000.

4] Port:

→ A port is a signal that connects a module to its environment.

→ They define data direction.

There are three types of port:

→ input
→ output
→ inout.

input port : Receive data from outside the module

Default type is Wire

Can be used in RHS.

Input represent stimulus or condition applied to a block.

output port

→ Send data from the module to outside.
→ Default ⇒ wire

inout port

→ Can act as input or output
→ An inout port must be driven with z when not actively driving.

eg:
```
module kittu(
    input en,
    input d_out,
    output d_in,
    inout data
);
assign data = en ? d_out : 1'bz;
assign d_in = data;

endmodule
```

When en ⇒ 1 o/p mode
en ⇒ 0 High impedence.

Remarks: Ports define the data direction

5) Number

b or B → Binary → Default Size => 1 bit
h or H → Hexadecimal → Default Size => 4 bit
o or O → Octal → Default Size => 3 bit.

Some tricky Question:

```
reg [2:0] a;
  initial begin
    a='b011;
    $display(a);
  end
```

O/P => a=>11.
↳ MSB will be truncate.

wire datatype is not used in initial and always block.

6) Operator {Imp}.

|Notes|

Date .....

☆ Construct

~ Branching Construct

- if-else statement
- case statement.

⇒ If-else statement.

1st

```
if (<condition>)
    statement-1;
```

2nd

```
if (<condition>)
    statement-1;
else
    statement-2;
```

3rd Multiway Decision

```
if (<cond-1>)
    statement=1;
else if (<cond-2>)
    statement-2;
else
    statement-3;
```

|Notes|

Date .....

* Verilog also support nested-if statement.

```
if (cond-1) begin
    statement = 1;
    if (cond-2)
        statement = 2;
end.
```

Case statement:

Anything can be written
↳ const, var, exp.

```
case ( <case_expression> )
    <case_item> : statement 1;
    <case_item> : statement 2;
    .
    .
    .
endcase
```

No need to use begin-end in case-endcase.

Bcs the intention of code is to execute parallely

It can also be written as

```
2'b01, 2'b00 : y = i[0]
    2'b10    : y = i[1]
```

|Notes|

Date .....

What if there was not any match in the case?

So it will not execute anything.
So good practice is to add default case in our case statement.

You are allowed to use if-else statement case vice-versa is also true.

These all are synthesizable in nature.

Q 8:3 priority encoder using if-else
Q 3:8 decoder using case
Q 8:3 priority encoder using case also
Q Bidirectional
Q ALU (8,16)

Ternary operator: Compare bit by bit.

★ Looping Construct:

i) forever loop.

→ Run until simulation get terminate.

Deadlock Condition:

```
forever a=5;
```

Here verilog can't able to understand when to stop.

→ Use timing statement whenever you want to execute forever.

Rem: Timing statement are not synthesizable.

forever is not synthesizable.

ii) Repeat loop.

→ repeat (<arguement>)
↳ anything

```
initial begin
int i=0;
$display ("%-0d", i);
#10 i++;
end
```

iii) While condition / loop.

```
While (condition)
  statement;
```

It will continued the exeution until condition get false.

post/pre inrement is not supported in verilog

Date .....

Assignments:

→ Continuous Assignment
→ Procedural Assignment

In continuous Assignment, we use construct like assign.

eg: assign y = a & b;

Whenever the variable on RHS side change its value the assign statement execute.

→ It is continuously active.

→ As all assign statement execute concurrently, this assignment is also known as concurrent assignment.

* Procedural Assignment.

|Notes|

Date .....

LHS always should be reg type

initial statement

→ Execute at zero simulation time

→ Once finished, it will terminated.

Multiple initial statement will exeute at zero simulation time.

Initial is not synthesizable.

★ always: will run at infinite time

always @

↓

event => Change in value of a Variable.

always block with sensitivity list is synthesizable.

|Notes|

Date .....

// level sens - wait statement
wait (a==5);
these are not synthesizable

Blocking Assignment:

'='

→ It blocks the execution of the following code until current statement is not executed.

→ Next statement won't execute until current statement is not finished.

It is an order dependent assignment.

```
always @(posedge clk) begin
b=d;
c=b;
d=c;
end
```

Block assignment is also called immediate assignment.

It gives the o/p at same clock cycle

| Notes |

Date .....

: Blocking Assignment is preferable for combinational ckt.

: It's better to avoid making seq ckt using blocking assignment

Non-Blocking

'<='

→ It does not block the execution of following code when current statement is executing unlike blocking assignment.

→ Next statement will execute along with current statement

→ Order of execution is parallel

→ It's an order independent assignment

→ Mainly used for sequential ckt design.

function:

* function only support input port
  Min. one arguement is req.
  Max no limit.

* Verilog alway return one value

```
                <type>
function <function_identifier> (<arg>);
  begin
  .
  end
endfunction
```

Calling a function:

```
<var> = <fun_identifier> (<arg>);
```

```
        [1:0]
function add (input a, input b);
    return a+b;
endfunction
reg [1:0] y, z;
```

function can call inside function, task,

y = add (1, 0);
z = add (p, q, r);

Task:

Task support all ports.

* If we haven't any arguement then it will be supported by task but not by function.

* Task support timing control

Task can be call inside procedural block, but not call by function

eg: task add (input a, b, output [..:0] sum = a+b;
end task.

add (1, 0, y);

|Notes|

Date .....

★ System Task:

To call any system task we've to use prefix $.

$display : $display ("Message");
$write :
$strobe :
$monitor :

None of the system task is synthesizable

let a=12;
$display(a);
↳ By default it will print value in decimal format.

After execution display will go to next line.
Whereas write will be there in same line.

$write is same as $display if we use \n in $write.

$strobe =>
=> Exeute at the end of current simulation time.

$monitor => exeute at the end of current simulation time. It will continue monitor the variable.

$monitoroff & $monitoron we can use to off or on the monitor.

$random
return 32 bit value
-2^31 to (2^31-1)

When we need only +ve value.
{$random} -> 0 to 2^32-1

$random %n -> -n to -(n-1)
{$random} %n -> 0 to (n-1)

a= {$random};
a= {$random}%50;

$time → Capture Simulation time.
64 bit unsigned value.

$realtime will give the accurate time.

time & realtime is a datatype

$finish ⇒ terminate the simulation & return the control back to host.

$finish (arguements) → 0, 1, 2.
↳ It can also take arguement

$finish (0) ⇒ print nothing
$finish (1) ⇒ print sim^{n} time & loc^{n}.
$finish (2) ⇒ " & statastics about memory.

$stop → suspend the simulation time
stop is just like pause

| Notes |

Date .....

→ integer
- => 32 bit
- => signed

→ time
- => 64 bit
- => unsigned.

=> Default = x.

→ real
to store floating value.
Non-synthesizable

=> Array

Width ↗ Depth ↗

```
reg [3:0] a [0:2]; // 1-D array
```

Invalid assignment:

```
reg a[0:3];
a = 4'b1011; { Invalid }
```

```
a[0] = 1'b1; { These is valid }
a[1] = 1'b0;
```

```
mem [2] [2:1] = 2'b01; { Valid }
```

Even verilog support multi-dimensional array.

|Notes|

Date .....

Q. Make Particular 2 bit write only & read only

Q %3d

✱ fork-join

In verilog we've two block → bigin-end & fork-join

Both is used to cover multiple state.
initial-begin-end will execute sequentially

fork-join will execute parallely.
fork-join is not synthesizable

fork-join can contain begin-end.

```
fork
  begin
    line-1;
    line-2;
  end
  begin
    line-3
    line-4
end
```

| Notes |

Date .....

Compiler Directive

→ Nothing is allowed outside the module in verilog except compiler Directive

All are synthesizable.

`define <Macroname> <TEXT>

`define <Macroname>

`define ABC 16

`ABC => 16

`define ABC "HELLO"

`ABC => "Hello"

Whatever we write as a text will get substituted.

`define is global to all module which is not possible with parameter

TOMH

| Notes |

Date .....

`Parameter can be override but `define can't.

Override means we can't change parameter from file but we can change it from somewhere

ii) `include "FILENAME"
We can copy eNtire code which was there in file.

iii) `timescale <time_unit>/<time_prec>
i) s, ms, us, ns, ps, fs these all unit are allowed.
ii) 1, 10, 100 timescale will work
iii) time unit must be greater then or equal to time precision.

`timescale 10ns/1ps
#10 ; // delay * time_unit => 100 ns

Return time type of $time is time_prec.

(1/2) & (1/2.0) diff

Doit.

|Notes|

Date .....

Event Scheduler

→ Active Region
* $display
* $write
* Blocking Assign
* RHS of NBA

→ Inactive Region
#0 delay blocking assignment

→ Non-blocking Region
LHS of NBA

→ Postponed Region
$strobe
$monitor

If event are a part of same region then these will follow top to down approach