Verilog HDL

INTERVIEW QUESTIONS

Jairaj Mirashi
Design Verification Engineer

SWIPE

Difference between blocking and non-blocking?

Blocking

1. The blocking assignment statement (= operator) acts much like in traditional programming languages.
2. The whole statement is done before control passes on to the next statement.
3. Blocking assignments evaluate their RHS expression and update their LHS value without interruption.

Non-Blocking

1. The non-blocking (<= operator) evaluates all the right-hand sides for the current time unit and assigns the left-hand sides at the end of the time unit.

SWIPE

Write a verilog code to swap contents of two registers with and without a temporary register?

With temp reg ;

```
always @ (posedge clock)
  begin
temp=b;
  b=a;
a=temp;
  end
```

Without temp reg;

```
always @ (posedge clock)
  begin
a <= b;
b <= a;
  end
```

SWIPE

Difference between task and function?

Function

1. A function is unable to enable a task however functions can enable other functions.
2. A function will carry out its required duty in zero simulation time. (The program time will not be incremented during the function routine)
3. Within a function, no event, delay or timing control statements are permitted
4. In the invocation of a function their must be at least one argument to be passed.
5. Functions will only return a single value and can not use either output or inout statements.

Task

1. Tasks are capable of enabling a function as well as enabling other versions of a Task
2. Tasks also run with a zero simulation however they can if required be executed in a non zero simulation time.
3. Tasks are allowed to contain any of these statements.
4. A task is allowed to use zero or more arguments which are of type output, input or inout.
5. A Task is unable to return a value but has the facility to pass multiple values via the output and inout statements .

Difference between inter statement and intra statement delay?

intra assignment delays

```
reg a, b, c;
//intra assignment delays
initial
begin
a = 0; c = 0;
b = #5 a + c; //Take value of a and c
at the time=0, evaluate
//a + c and then wait 5 time units to
assign value
//to b.
end
```

inter assignment delays

```
initial
begin
a = 0; c = 0;
 #5 b = a + c;  //Take value of a + c at
the current time and
//store it in a temporary variable. Even
though a and c
//might change between 0 and 5,
//the value assigned to b at time 5 is
unaffected.
end
```

SWIPE

What is the difference between $display, $write, $monitor and $strobe

1. $display : Print the values immediately when executed.
2. $strobe : Print the values at the end of the current timestep.
3. $monitor : Print the values at the end of the current timestep if any values change. If $monitor is called more than once, the last call will override previous one.
4. $write : This is same as $display but doesn’t terminate with a newline (\n).

SWIPE

What is the difference between wire and reg?

wire

- Wire cannote hold the value
- Net types: (wire,tri)Physical connection between structural elements. Value assigned by a continuous assignment or a gate output.

reg

- reg can hold the value
- Register type: (reg, integer, time, real, real time) represents abstract data storage element. Assigned values only within an always statement or an initial statement.

Default values:wire is Z,reg is x.

SWIPE

What is the difference between Continuous and Procedural Assignment?

Continuous

1. Assign values to nets
2. This assignment occurs whenever the value of the RHS changes
3. Executes parallel
4. Continuously Active
5. Order independent

Procedural

1. Procedural assignments occur within procedures such as always , initial , task and functions and are used to place values onto variables.
2. The variable will hold the value until the next assignment to the same variable.

SWIPE

What is the difference between Initial and Always?

Initial

- Used for testbench(not synthesisable)
- Exicution starts at 0 Simulation time
- Once Exicution only

Always

- Repeat continuously throughout the duration of the simulation(loop continuous).

```
always@();
```

SWIPE

How many flip-flops will be needed when following two codes are synthesized?

```
always @(posedge clk) begin
    B = A;
    C = B;
  end
```

- One Flip-flop

- Blocking assignments are used and hence the value of A will be assigned to B and the new value will be reflected onto C in same cycle and hence the variable B and C results in a wire. So, only one flip flow will be needed.

```
always @(posedge clk) begin
    B <= A;
    C <= B;
  end
```

- Two Flip-flops

- Old value of B is sampled before the new value is reflected in each cycle.

Hence value of A reflects to C only in 2 cycles, resulting in two flip-flops.

What is the difference between “==” and “===” operators?

Both of these are equality or comparison operators.

1. The “==” tests for logical equality for two states (0 and 1)
2. If “==” is used to compare two 4-state variables (logics) and if at least one of them has an X or Z, the result will be X.

1. The “===” operator tests for logical equality for four states (0, 1, X and Z)
2. If the “===” is used to compare two 4-state variables, then comparison is done on all 4 states including X and Z, and the result is 0 or 1.

EXAMPLE

If A and B are two 3-bit vectors initialized as follows:

A = 3‘b1x0

B = 3‘b1x0

What would be value of following?

1) A==B

2) A===B

1) A==B will only compare non-X/Z values and hence will result in an output “X” if any of the operands has an unknown bit

2) A===B will compare bits including X and Z and hence the comparison would return a 1 as both bit 1 are X for A and B.

Given the following Verilog code, what value of "a" is displayed?

```
always @(clk) begin
a = 0;
a <= 1;
$display(a);
end
```

Verilog scheduling semantics basically imply a four-level deep queue for the current simulation time:

1. Active Events (blocking statements)
2. Inactive Events (#0 delays, etc)
3. Non-Blocking Assign Updates (non-blocking statements)
4. Monitor Events ($display, $monitor, etc).

Since the "a = 0" is an active event, it is scheduled into the 1st "queue".
The "a <= 1" is a non-blocking event, so it's placed into the 3rd queue.
Finally, the display statement is placed into the 4th queue. Only events in the active queue are completed this sim cycle,
so the "a = 0" happens, and then the display shows a = 0.
If we were to look at the value of a in the next sim cycle, it would show 1.

Write an RTL code for D flip-flop in both synchronous and asynchronous mode.

Synchronous

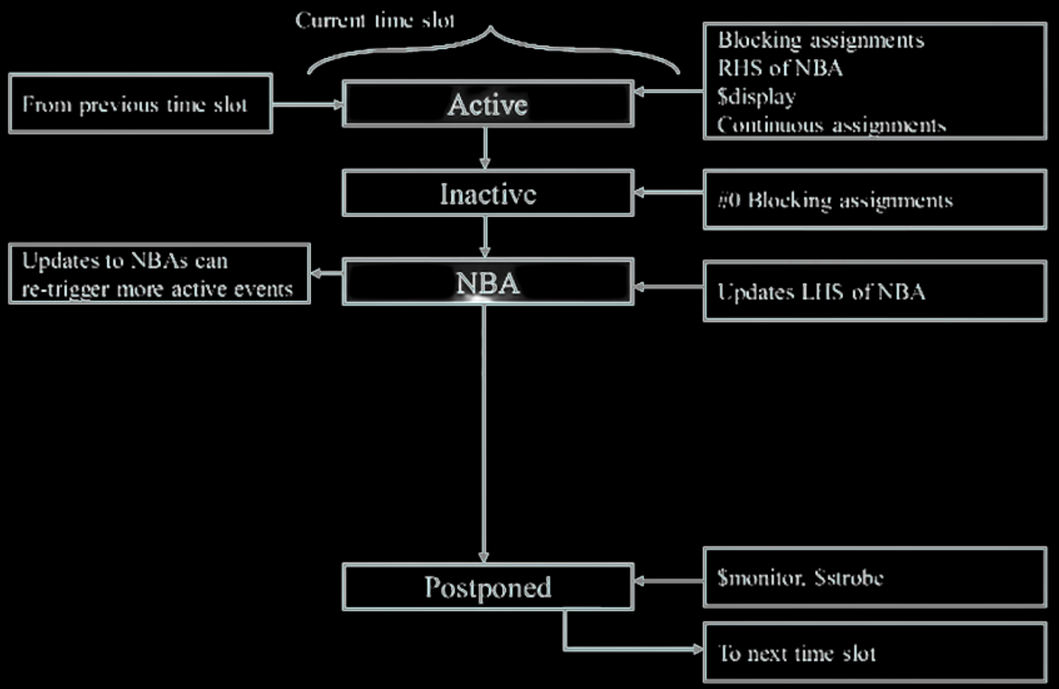
```
always @(posedge clk)
    begin
    if(sync_reset)
      Q <= 1'b0;
    else
      Q <= D;
    end
```

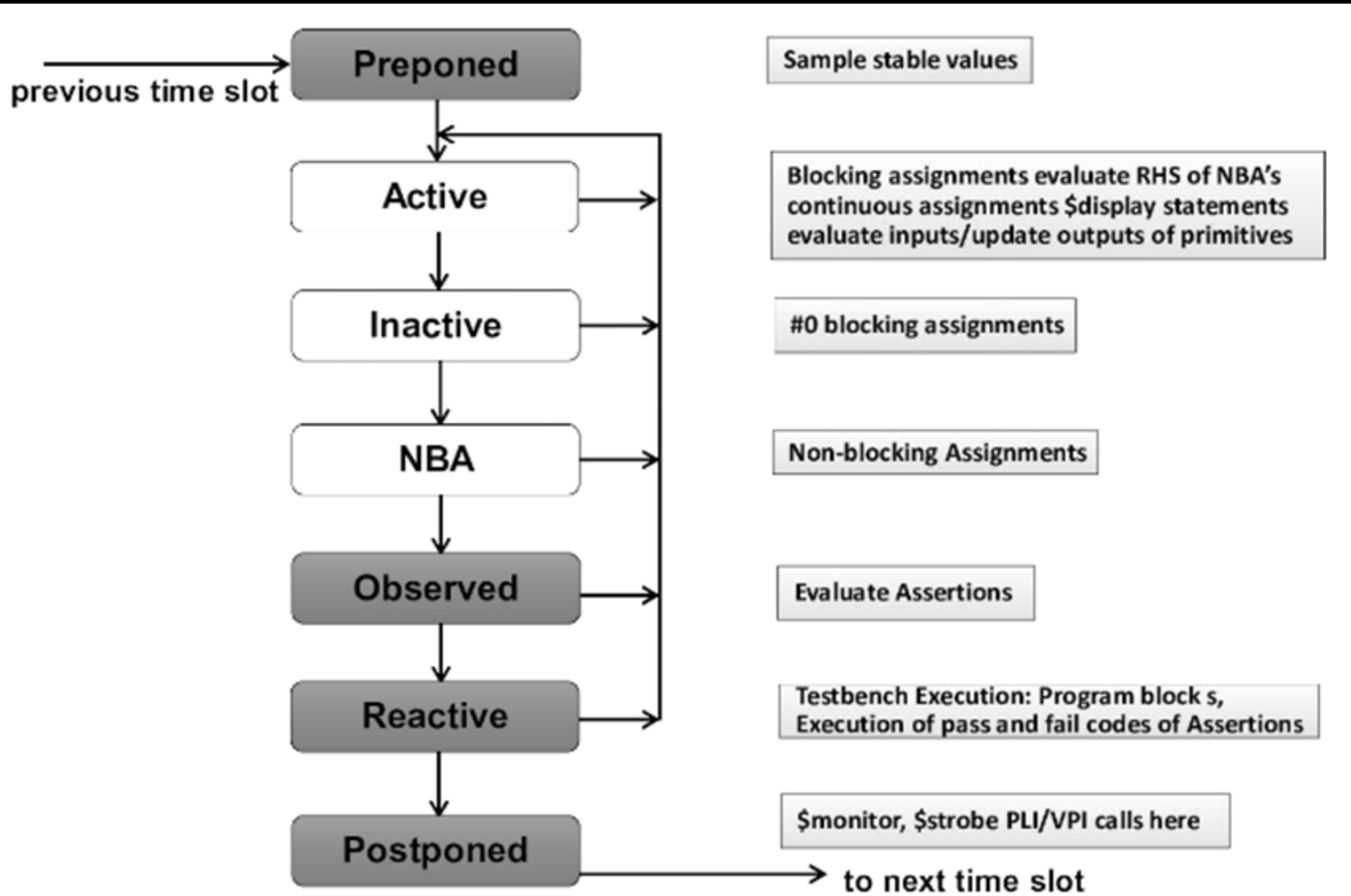
Asynchronous

```
always @(posedge clk or posedge async_reset)
    begin
    if(async_reset)
      Q <= 1'b0;
    else
      Q <= D;
    end
```

SWIPE

Explain the Verilog event scheduler.





What is the difference between begin - end and fork - join?

begin - end

1. The statement executes in sequence, one after the other.
2. Delay values for each statement are treated relative to the simulation time of the execution of the previous statement.
3. Control is passed out of the block after the last statement executes.

fork - join

1. The statement executes concurrently/parallel.
2. Delay values for each statement are treated relative to the simulation time of entering the block.
3. Control is passed out of the block when the last time-ordered statement executes.

EXAMPLE

```
initial
beign
#10 a=x;
#30 b=y;
#20 c=z;
end
```

Delays will be added.
so,at 60 time unit
This block will finish
execution after the **last statement** is executed

```
initial
fork
#10 a=x;
#30 b=y;
#20 c=z;
join
```

Delays will not be added
So,at 30 time unit
This block will finish
execution after **b=y statement** is executed(maximum delay)

Difference between case, casex and casez.

The case statement is a multiway decision statement that tests whether an expression matches one of a number of possible values and branches accordingly. Following is a simple example to implement a 3:1 MUX using a case statement

```
case (select[1:0])
2'b00: out_sig = in0;
2'b01: out_sig = in1;
2'b10: out_sig = in2;
default: out_sig = 'x
endcase
```

In the above example of using a "case" statement, the expression match happens exactly with what is specified. For example, in above case statement, if at least one of the select lines is X or Z, then it will not match any conditions and will execute the default statement.

" casez " is a special version of case expression which allows don't cares in comparison of the expressions.
These are typically useful in decoder logic which only treats fewer bits.
Here is an example where a 3 bit interrupt request queue (irq) bus is decoded into 3 separate interrupt pins based on which bit in the bus is high, while other bits are don't care.

```
casez (irq)
3'b1?? : int2 = 1'b1;
3'b?1? : int1 = 1'b1;
3'b??1 : int0 = 1'b1;
endcase
```

"casex" is another special version where in addition to don't cares, it also ignores X and Z values in comparison.

Coding for synchronous and asynchronous FIFO

The main diffrence between synchronous and asynchronous FIFO is Read clock is operating at different frequency and Write Clock is operating at different frequency

READ AND WRITE CLOCK GENERATION

```
initial
 always
 #50000 wr_clk=~wr_clk;
 rd_clk=1'b0;
 initial
 begin
 always
 #10000 rd_clk=~rd_clk;
 end
```

The Write clock is operating at 50 MHz

Read Clock is operating at 10 MHz

```
module fifo(clk,rst,we,re,data_in, data_out,full, empty);

input clk,rst,we,re;
input [7:0] data_in;
output reg [7:0] data_out;
output full,empty;
reg [3:0] rd_pt, wr_pt;
reg [7:0] mem[15:0];
integer i;

always@(posedge clk)
if(rst)
begin
data_out <= 0;
wr_pt <=0;
rd_pt <=0;
for(i=0; i< 16; i=i+1)
mem[i] <=0;
end

else if(we==1 && full==0)
begin
mem[wr_pt] <= data_in;
wr_pt <= wr_pt+1;
end

else if(re==1 && empty==0)
begin
data_out <= mem[rd_pt];
rd_pt <= rd_pt+1;
end

assign empty = (rd_pt ==4'b1111) ? 1'b1 : 1'b0;
assign full = ( wr_pt==4'b1111 ) ? 1'b1 : 1'b0;

endmodule
```

Difference between $stop and $finish

$stop

Causes simulation to be suspended

```
#100 $stop;
```

simulation suspended at time 100

$finish

This task makes the simulator exit and pass control back to the host operating system

```
#100 $finish;
```

simulation terminates at time 100

Generate a random number in verilog

```
module Tb();
   integer add_1;

   initial
    begin
     repeat(5)
      begin
      #1;
      add_1 = {$random} % 10;
      add_1 = $unsigned($random) % 10;
      end
    end

   initial
    $monitor("add_1 = %0d",add_1);
  endmodule
```

RESULT:

```
add_1 = 8;
add_1 = 7;
add_1 = 5;
add_1 = 9;
add_1 = 9;
```

will_generate random numbers between MIN and MAX

```
MIN + {$random} % (MAX - MIN )
```

EXAMPLE:

```
   module Tb();
    integer add;

    initial
     begin
      repeat(5)
       begin
        #1;
        add = 40 + {$random} % (50 - 40) ;
        $display("add = %0d",add);
       end
     end
   endmodule
```

RESULT:

```
add = 48
add = 47
add = 49
add = 46
add = 47
```

Coding for up/down counter

```
module updown(clk, rst, load, data_in, control, count);

input clk, control, load, rst;
input [3 : 0] data_in;
output reg [3:0] count;

always@(posedge clk)
if(rst)
count <= 4'b0000;
else if(load)
count <= data_in;

else if(control)
count <= count+1;
else
count <= count-1;
endmodule
```


Thank you
C20
C18
C24
C26
C37
U2
CS42418-CQZ
119717
WACXCK0419