HWL: Feb 21 (Monday Week) (*7)
(1 week later for VBEE)
A NTU
Design Example ... Verilog

Continuous Assignment:
assign is used to implement combinational logic directly

Questions
1. When is the procedural block following the always@(posedge clock) executed?
   \( \text{clock} \uparrow \)
2. How is a comment done?
   \( \text{II a // ++ //} \)
3. What does 1'b1 mean?
   \( \text{1 bit library} 1 \)
4. What does reg [3:0] value; declare?
   \( 4\text{-bits. value assigned procedurally.} \)
5. What does #1 mean and why do I use it?
   \( \text{delay 1us} \)
6. Convert the continuous assignment statement to a procedural block.
reg zero;
always @ (value)
  Zero = (value == 4'd0);
// execute this block whenever 'value' changes

// modeling a piece of combinational logic
assign output = inputs and operators

end

if-then-else or case statements

begin

always (input or input2 or ..)

Case (Case Logic)

else register-output = #1 register-output;

begin

always (positive clock)

For Registers and their associated input logic:

Here is a suggested basic style for Verilog usage:

It is also very easy to describe hardware that has no direct correlation to HW

It is very easy to describe functions that do NOT map well (or synthesize into) hardware

Verilog is a powerful and flexible language

A Basic Verilog Style
Examples: 100000 = 0...01011 (32 bits) 3,011

Other bases: 

h = hexadecimal, d = decimal (which is the default)

NOTE: Zero padded to left

size, base value: size = # bits, base = binary

Integers:

<table>
<thead>
<tr>
<th>Unknown, uninitialized, or Don't Care</th>
<th>X or x</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Impedance, Tri-Stated or Floating</td>
<td>Z or z</td>
</tr>
<tr>
<td>One, High or True</td>
<td>1</td>
</tr>
<tr>
<td>Zero, Low or False</td>
<td>0</td>
</tr>
</tbody>
</table>

Logic Values:

Lexical Conventions in Verilog
Internal Operation of VL

(clock) \downarrow

triggers all pos always (posedge clock)

blocks

in each block

nextc = a + b

nenxtd = nextc + a

All blocks reach end

c = nextc

d = nextd

ctc

nextc = a + b

nextd = c + a
Always use \( \Rightarrow \) when building registers to prevent possible races.

\( \Rightarrow \) is referred to as non-blocking assignment. Essentially, \( c \rightarrow \) is performed in parallel, \( a \rightarrow \) are performed in series, and \( d \rightarrow c \) blocked until this statement completes. Essentially, \( c = a + b \) and \( d = c + a \).

What is the difference between the following code segments?

\begin{align*}
\text{Before code:} & \\
& a \rightarrow c \rightarrow a, \\
& b \rightarrow a + b, \\
& \text{begin} \\
& \text{always@posedge clock} \\
& \text{end}
\end{align*}

\begin{align*}
\text{After code:} & \\
& d = c + a, \\
& c = a + b, \\
& \text{begin} \\
& \text{always@posedge clock} \\
& \text{end}
\end{align*}

NC STATE UNIVERSITY

ECE 250 Class Notes
1. Note "paper exec. paper simulation" → understanding

2. This latch was unintentional " " caused by us not specifying what happened to z if x = 0 → ie built memory

To prevent unintentional latches make sure that each output is assigned for all possible combinations of inputs.
\[ f(x) = x^2 + 2x - 1 \]
What do the following code fragments synthesize to?

Procedural Block Examples

select ten
end

HVX

end

else
endif
always @(w or x or y)

end

z = y

end

always @(x or y)

z = y

end

z = y

end

select ten
end

end