For Loop - VHDL and Verilog Example

Write synthesizable and testbench For Loops

For loops are one of the most misunderstood parts of any HDL code. For loops can be used in both synthesizable and non-synthesizable code. However **for loops perform differently in a software language like C than they do in VHDL**. You must clearly understand how for loops work before using them!

Converting A Software-Style For Loop to VHDL/Verilog

For loops are an area that new hardware developers struggle with. You have likely seen for loops dozens of times in C, so you think that they are the same in Verilog and VHDL. Let me be clear here: For loops do *not* behave the same way in hardware as in software.

For loops in synthesizable code are used to *expand replicated logic*. They are simply a way of shrinking the amount of code that is written by the hardware designer. Again, until you understand how exactly this expansion of replicated logic works, do not use for loops. Instead think about how you want your code to behave and figure out a way to write it in C without using a for loop, then write your code in VHDL or Verilog. Below is an example of this:

```
// Example Software Code:
For (int i=0; i<10; i++)
   data[i] = data[i] + 1;
```

This code will take every value in the array "data" and increment it by 1. Here is equivalent code in VHDL:

```
P_INCREMENT : process (clock)
begin
  if rising_edge(clock) then
    if index < 10 then
      data(index) <= data(index) + 1;
      index       <= index + 1;
    end if;
  end if;
end process P_INCREMENT;
```

And here is equivalent code in Verilog:

```
always @(posedge clock)
  begin
    if (index < 10)
      begin
        data[index] <= data[index] + 1;
        index       <= index + 1;
      end
  end
```

Usually all you need is to add a counter signal (like index in the example above) to do the same thing that the for loop will do.

Using For Loops in Synthesizable Code

For loops *can* be synthesized. For loops in synthesizable code are used for expanding replicated logic. They are simply a way of shrinking the amount of code that is written by the hardware designer. They do *not* loop like a C program loops. They only expand replicated logic. Let's look at an example of this. Note that the code below is written in both VHDL and Verilog, but the simulation results are the same for both languages.

VHDL Synthesizable for loop example code:

The two processes perform exactly the same functionality except the for loop is more compact. For loops can also be used to expand combinational logic outside of a process or always block. For that, you need to use a Generate Statement.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Example_For_Loop is
  port (
    i_Clock : std_logic
    );
end Example_For_Loop;

architecture behave of Example_For_Loop is

  signal r_Shift_With_For : std_logic_vector(3 downto 0) := X"1";
  signal r_Shift_Regular  : std_logic_vector(3 downto 0) := X"1";

begin

  -- Creates a Left Shift using a For Loop
  p_Shift_With_For : process (i_Clock)
  begin
    if rising_edge(i_Clock) then
```

```
      for ii in 0 to 2 loop
        r_Shift_With_For(ii+1) <= r_Shift_With_For(ii);
      end loop;  -- ii
    end if;
  end process;

  -- Performs a shift left using regular assignments
  p_Shift_Without_For : process (i_Clock)
  begin
    if rising_edge(i_Clock) then
      r_Shift_Regular(1) <= r_Shift_Regular(0);
      r_Shift_Regular(2) <= r_Shift_Regular(1);
      r_Shift_Regular(3) <= r_Shift_Regular(2);
    end if;
  end process;


end behave;
```

Verilog Synthesizable For Loop Example Code

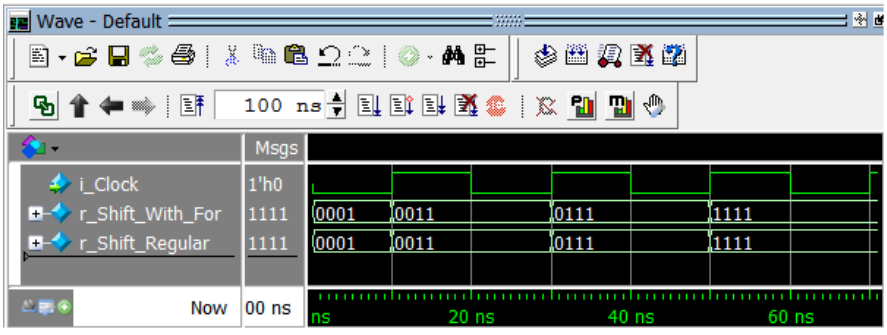
The two always blocks below perform the same purpose, except one uses a for loop and the other does not. Again, all the loop does is to expand replicated logic.

```
module for_loop_synthesis (i_Clock);
  input i_Clock;
  integer ii=0;
  reg [3:0] r_Shift_With_For = 4'h1;
  reg [3:0] r_Shift_Regular  = 4'h1;

  // Performs a shift left using a for loop
  always @(posedge i_Clock)
    begin
      for(ii=0; ii<3; ii=ii+1)
        r_Shift_With_For[ii+1] <= r_Shift_With_For[ii];
    end

  // Performs a shift left using regular statements
  always @(posedge i_Clock)
    begin
      r_Shift_Regular[1] <= r_Shift_Regular[0];
      r_Shift_Regular[2] <= r_Shift_Regular[1];
      r_Shift_Regular[3] <= r_Shift_Regular[2];
    end
endmodule


module for_loop_synthesis_tb ();    // Testbench
  reg r_Clock = 1'b0;
  // Instantiate the Unit Under Test (UUT)
  for_loop_synthesis UUT (.i_Clock(r_Clock));
  always
    #10 r_Clock = !r_Clock;
endmodule
```



For Loop Simulation Results

As can be seen in the example above, **all the for loop does for synthesis is to expand replicated logic**. It will essentially unwrap the entire loop and replace the loop with the expanded code. The signals r_Shift_With_For and r_Shift_Regular behave exactly the same way! Now let's look how for loops can be used in simulation.

Using For Loops in Simulation

For loops when used in a simulation environment can behave more like the traditional for loop that you have seen in other software programming languages. They can have delays inside them and can actually delay the simulation while executing them.

The example below will initialize r_Data in an incrementing pattern, assigning one value every 10 ns of simulation time. It also displays the values as it assigns them. This is not synthesizable code!

VHDL Implementation

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity for_loop_simulation is
end entity for_loop_simulation;

architecture behave of for_loop_simulation is
  type t_Data is array (0 to 5) of integer;
begin

  process is
    variable r_Data : t_Data;  --  Create 6 words deep array of integers
  begin

    for ii in 0 to 5 loop
      r_Data(ii) := ii*ii;
      report("r_Data at Index " & integer'image(ii) & " is " &
             integer'image(r_Data(ii)));
      wait for 10 ns;
    end loop;

    assert false report "Test Complete" severity failure;
  end process;
end architecture behave;
```

Verilog Implementation

```verilog
module for_loop_simulation ();
  integer ii=0;
  reg [7:0] r_Data[5:0]; // Create reg 8 bit wide by 6 words deep.

  initial
    begin
      for (ii=0; ii<6; ii=ii+1)
        begin
          r_Data[ii] = ii*ii;
          $display("Time %2d: r_Data at Index %1d is %2d", $time, ii, r_Data[ii]);
          #10;
        end
    end
endmodule
```

Console Result from Modelsim Simulation:
(Same for both VHDL and Verilog Examples above)

```
# Time  0: r_Data at Index 0 is  0
# Time 10: r_Data at Index 1 is  1
# Time 20: r_Data at Index 2 is  4
# Time 30: r_Data at Index 3 is  9
# Time 40: r_Data at Index 4 is 16
# Time 50: r_Data at Index 5 is 25
```