Managing Verilog Parameters for Synthesis and Simulation

1. Problem statement

When Verilog code is written using parameters, it is often convenient to use different values for a parameter depending upon whether the design is synthesized or simulated. A common example is a clock divider circuit. If the divisor value is large (providing a low clock frequency), simulation of the design may require many millions of simulation cycles due to the large divisor value. Commonly the designer will modify the parameter in the source files to accommodate a more efficient simulation. The ideal situation is using a methodology that allows rapid modification of parameter values based on the need to synthesize or simulation the design.

The example files are included in VerilogParameters.zip.

2. EDA Tools tested

The approach discussed in this paper has been verified using both Modelsim and iVerilog simulators. The code will compile without problems in Intel Quartus II. The methodology as defined here is general, so should work for other EDA tools as well.

3. Notes

a. Parameter names are in upper case.
b. Verilog-1995 will not work, use Verilog-2001 or later. The syntax used when defining parameters in a Verilog module requires at least Verilog-2001.
c. DO NOT add the parameter file as a Quartus source file.
d. When using iVerilog, DO NOT compile the parameter file.
e. Note that the Modelsim testbench.tcl file does not compile the parameter file, but does use the +incdir+ option to point to the directory containing the parameter file.
f. The directory structure for the examples looks like this:

```
project_top
     source
     modelsim
```

The source directory contains all Verilog source, parameter, and testbench files. Note that the testbench files differ depending on whether Modelsim (tb_modelsim.v) or iVerilog (tb_iverilog.v) is used.

The Modelsim directory contains the testbench.tcl and the wave.do files.

4. Use of preprocessor directives

The following preprocessor directives will be used:

`` `ifdef ``

The **`` `ifdef ``** directive tests for the existence or definition of a macro. The macro name is unimportant, but will indicate the selection of a simulation flow, and for this reason the variable used in in this example project is **`simulation`**.

`` `else `` **i**s identical to any if / then selection.

`` `endif `` is used to close a **`` `ifdef ``** block.

`` `define `` is used to define the existence of a macro.

`` `include `` performs a direct text substitution from the included file.

5. Example Design

The example design consists of a top level module (top.v) and a lower level clock divider module (clkdiv.v). The clock divider module is instantiated twice with different divider values. As the code is written, the output clock will be divided by DIV*2 – for example, if DIV1 is set to 3, the output clock frequency will be 1/6 of the input clock frequency.

For the synthesis case (**`simulation`** not defined in top.v), one clock divider output is clkin/20, and the second clock divider output is clkin/10. For simulation, the first clock divider output is changed to clkin/6.

6. Step 1 – Create a parameter file

To begin, put all design parameters into a single file. Use a .vh file extension instead of a .v extension, in order to allow differentiation of the parameter file, and as a reminder that this file does not need to be added as a source file. For this example, the file will be called params.vh.

The macro **`simulation`** will be defined when the testbench is compiled, but will not be defined otherwise. Here is the entire params.vh file:

```
`ifdef simulation
    parameter DIV1 = 3;  // Divide clock by 6 in simulation
`else
    parameter DIV1= 10;  // Divide clock by 20 for synthesis
`endif
parameter DIV2 = 5;      // Divide second clock by 10 in both cases
```

7. Step 3 – Modify the top level Verilog module

Immediately following the **`module`** statement, add the **`` `define simulation``** statement. For a synthesis run, comment out this line. For simulation, the line should not be commented out. The **`` `include``** statement should be next. In this example the params.vh file is in the same directory as the other source files. You can use a relative or absolute path if required, for example **`` `include "..\source\params.vh"``**. Note that Modelsim requires the slashes used in the directory path to be forward slashes.

```
module top (input clkin, input reset_n, output clkout1, output clkout2);
`define simulation
`include "params.vh"

clkdiv #(.DIVIDER(DIV1)) U0 (.clk(clkin), .reset_n(reset_n), .clkout(clkout1));
clkdiv #(.DIVIDER(DIV2)) U1 (.clk(clkin), .reset_n(reset_n), .clkout(clkout2));

endmodule
```

8. Clock Divider module

Note that the parameter DIVIDER is part of the **`module`** statement, an enhancement added in Verilog-2001.

```
/* The DIVIDER parameter can be set to any default value.
   In this example, the expectation is that the calling module must
   provide the value, otherwise compilation will fail.
*/
module clkdiv #(parameter DIVIDER = 0) (clk, reset_n, clkout);
input clk, reset_n;
output reg clkout;

reg [31:0] counter;

always @ (posedge clk or negedge reset_n)
    if (reset_n == 1'b0)
        begin
            counter <= 0;
            clkout <= 1'b0;
        end
    else if (counter == DIVIDER - 1)
        begin
            counter <= 0;
            clkout <= ~clkout;
        end
    else
        counter <= counter + 1'b1;

endmodule
```

9. Step 2 – Modify the testbench file

Add the **`` `include “params.vh” ``** line to your testbench.

```
`timescale 1 ns / 100 ps
module testbench();
`include "params.vh"
```

10. Example using iVerilog and GTKWave

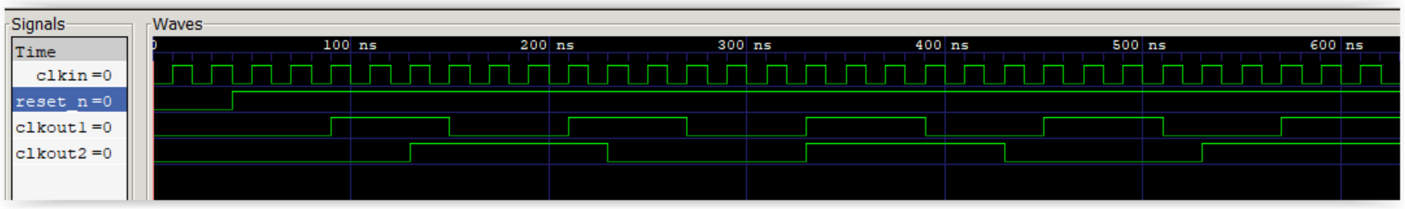
The testbench file for iVerilog and GTKWave is shown here:

```
`timescale 1 ns / 100 ps
module tb_iverilog();
`include "params.vh"

reg clkin, reset_n;
wire clkout1, clkout2;

top DUT (.clkin(clkin), .reset_n(reset_n),
         .clkout1(clkout1), .clkout2(clkout2));

initial
  begin
       $dumpfile("tb_iverilog.vcd");
       $dumpvars;
    clkin = 0; reset_n = 0;
  end

always
  #10 clkin = ~clkin;

initial
  begin
    # 40 reset_n = 1;
    # 10000 $finish;
  end

initial
  $monitor($time, "  clkin = %b, reset_n = %b, clkout1 = %b , clkout2 = %b",
                     clkin, reset_n, clkout1, clkout2);

endmodule
```

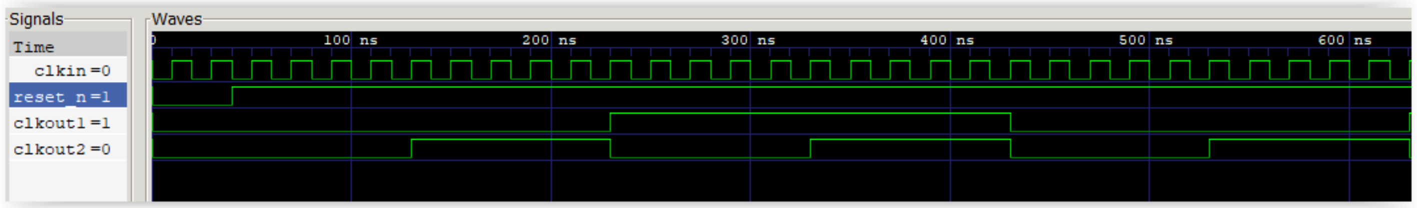
These iVerilog commands were run:

```
iverilog tb_iverilog.v top.v clkdiv.v
vvp a.out
gtkwave tb_iverilog.vcd
```

With the **simulation** macro defined in top.v (DIV1 = 3, DIV2 = 5), the simulation output is shown below:



The same iVerilog commands are rerun after the `define simulation line in top.v is commented out:



Note that the clkout1 output frequency is now based on DIV1 = 10.

11. Modelsim Details

a. Launch Modelsim standalone (not from within Quartus).
b. Change directory to the example project modelsim directory.
c. At the Modelsim prompt, type **`do testbench.tcl`**

testbench.tcl

```
# stop any simulation that is currently running
quit -sim
# create the default "work" library
vlib work;
# compile the Verilog source code in the source folder
# the +incdir+ is necessary so that the params.vh file is found
vlog ../source/top.v ../source/clkdiv.v +incdir+../source
# compile the Verilog testbench
vlog ../source/tb_modelsim.v +incdir+../source
# start the Simulator, including some libraries that may be needed
vsim work.testbench -Lf 220model -Lf altera_mf_ver -Lf verilog
# show waveforms specified in wave.do
do wave.do
# advance the simulation the desired amount of time
run 2 us
```

wave.do

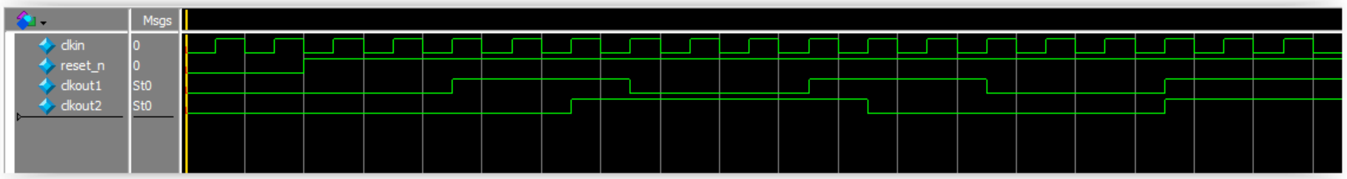
```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -label clkin /testbench/clkin
add wave -noupdate -label reset_n /testbench/reset_n
add wave -noupdate -label clkout1 /testbench/clkout1
add wave -noupdate -label clkout2 /testbench/clkout2
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {10000 ps} 0}
quietly wave cursor active 1
configure wave -namecolwidth 80
configure wave -valuecolwidth 40
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ns
update
WaveRestoreZoom {15 ns} {1000 ns}
```

12. Example using Modelsim

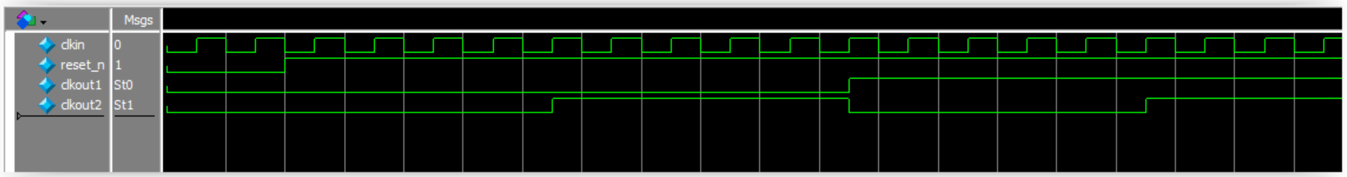
The testbench file for Modelsim is shown below. For Modelsim, there is no need to define a dumpfile for the .vcd output, and the duration of the simulation can be controlled in the testbench.tcl file. Notice the forward slashes in the directory path for params.vh.

```
`timescale 1 ns / 100 ps
module tb_modelsim();
`include "../source/params.vh"

reg clkin, reset_n;
wire clkout1, clkout2;

top DUT (.clkin(clkin), .reset_n(reset_n),
         .clkout1(clkout1), .clkout2(clkout2));

always
    #10 clkin = ~clkin;

initial
    begin
        clkin = 0; reset_n = 0;
    #40 reset_n = 1;
    end
initial
  $monitor($time, "  clkin = %b, reset_n = %b ,clkout1 = %b , clkout2 = %b",
                     clkin, reset_n, clkout1, clkout2);

endmodule
```

With the **`simulation`** macro defined in top.v (DIV1 = 3, DIV2 = 5), the simulation output is shown below:



Commenting out the `define lines gives:



13. Summary

This paper describes a straightforward approach to quickly changing parameter values as needed when simulating a design. With this approach, the only change required to modify parameters for synthesis or simulation is to comment or uncomment one line in the top level file.

This approach has been tested with both Modelsim and iVerilog simulators, and is completely compatible with Intel Quartus II software.