Verilog GSR/GTS Simulation Methodology–Changes in the Alliance Series 2.1i Software

by Roberta Fulton, Technical Marketing Engineer, Xilinx, roberta.fulton@xilinx.com

With the release of Alliance Series 2.1i software, Xilinx has added support for compiled Verilog simulators such as Synopsys VCS, Model Technology VLOG, and Cadence NC-Verilog. These simulators are full Verilog simulators but behave differently in a few significant ways from interpretive Verilog simulators like Verilog-XL. Please see the individual tool manuals for details.

The previous GSR/GTS approach allowed you to define the GSR/GTS signal as a text macro with the **‘define** construct, at any time prior to the actual simulation of the GSR signal by an interpretive simulator like Verilog-XL. For example, **‘define GSR my_gsr_signal** could be entered on the command line at any time prior to simulation. But in compiled Verilog it is a requirement that all constructs be fully elaborated at the time they are compiled. This prohibits a text macro from being used by a **‘ifdef** construct contained in a library module if the text macro is defined in a higher level module or on the command line after the library has been compiled.

The new methodology works for both compiled and interpretive simulators. It defines a module called **glbl** that contains the GSR signal. The Unified Library component modules drive their internal GSR behavior by the glbl.GSR wire. The glbl.GSR wire can in turn be driven by any user-defined wire attached to it in the test fixture.

The New Simulation Methodology

While GSR alone is described here for simplicity, this methodology equally applies for the CPLD PRLD signal and the GR signal in older FPGA technologies, and works similarly for the global GTS signal on the output buffers. For a more complete explanation of the GSR/GTS methodology, please refer the 2.1i version of the Xilinx Synthesis and Simulation Guide on the Alliance Series 2.1i Documentation CD, or www.support.xilinx.com.

Describing the GSR Behavior (Verilog)

For Verilog simulation, all behaviorally described (inferred) and instantiated registers should have a common signal (GSR) that asynchronously sets or resets the register. Toggling GSR emulates the automatic Reset-on-Configuration mode of the FPGA. This is similar to the Power-on-Reset of an ASIC. If you do not do this, the flip-flops and latches in your simulation may initialize to an unknown state in the simulation.

The GSR net is present and may be pulsed in your implemented design (as it is in silicon) even if you do not instantiate the STARTUP block in your design. The function of STARTUP is to give you the option to control the global reset net from an external pin or from user-defined internal circuitry.

Often, mismatches between the Register Transfer Level (RTL) and gate-level simulations are caused by not fully defining the GSR behavior in the RTL. The new methodology allows the GSR behavior to be fully defined in the RTL, while properly synthesizing and implementing to a post-route netlist that can use the same test fixture. The Verilog UniSim library is used in RTL or gate-level simulations prior to implementation. Simulation at other points in the flow uses the Verilog SimPrims Libraries.

(Continued)

Describing the GSR Behavior (UniSim)

For UniSim functional simulation, you must set the value of the **glbl.GSR** net to the same name value as the GSR net, qualified by the appropriate scope identifiers. The scope identifiers are a combination of the test module scope and the design instance scope. The scope qualifiers are required because the scope information is needed when the **glbl.GSR** wires are interpreted by the Verilog UniSim simulation models to emulate a GSR signal.

For post-route timing simulation the test fixture template (.tv file), produced by running NGD2VER with the **-tf** option, contains most of the code required for defining and toggling GSR.

- **Without a STARTUP module -** Add the following to the test fixture file:

```
reg GSR;
assign glbl.GSR = GSR;
assign testfixture_name.instance_name.GSR=GSR;
// Only for RTL modeling of GSR
```

  For post-route timing simulation, you must omit the "assign" statement for GSR. This is because the net connections exist in the post-route design; retaining the assign definition causes a possible conflict with these connections.

- **With a STARTUP module -** If you do have a STARTUP block in your design, the signal you toggle is the external input port that controls the global reset pin of the STARTUP block. You should add the following to the test fixture module for the RTL modeling of the global reset pin:

```
reg port_connected_to_GSR_pin;
assign glbl.GSR = port_connected_to_GSR_pin;
```

For post-route timing simulation, you must omit the assign statement for the global reset signal. This is because the GSR net connections explicitly exist in the post-route design, and retaining the assign definition causes a possible conflict with these connections.

A Verilog global signal called **glbl.GSR** is defined within the STARTUP and STARTUP_VIRTEX modules to make the connection between the user logic and the global GSR net embedded in the Unified Library models. For post-route timing simulation, **glbl.GSR** is defined in the Verilog netlist that is created by NGD2VER.

You can compile the Verilog source files in any order because Verilog is compiled independently of the source module hierarchy except in cases of constructs like **'define** or **'defparam** as previously noted. However, Xilinx recommends that you specify the test fixture file before the Verilog netlist of your design, as in the following examples.

Invoking the Simulators

- **For RTL simulation,** enter the following:

```
verilog -y $XILINX/verilog/src/unisims
design.stim design.v $XILINX/verilog/src/glbl.v
```

  The path specified with the **-y** switch points the simulator to the UniSim models and is only necessary if Xilinx primitives are instantiated in your code. When targeting a device family other than the XC4000E/L/X, Spartan/XL, or Virtex families, change the UniSims reference in the path to the targeted device family.

- **For post-route simulation,** enter the following:

```
verilog design.stim time_sim.v
$XILINX/verilog/src/glbl.v
```

  In this example, the test fixture file is declared first followed by the simulation netlist created by the Xilinx tools. The name of the Xilinx simulation netlist may change depending on how the file was created. For Verilog-XL, it is also assumed that the -ul switch was specified during NGD2VER to specify the location of the SimPrims libraries using the **uselib** directive.

(Continued)

MTI ModelSim

- **For RTL simulation,** enter the following:

```
vlog design.stim design.v
$XILINX/veilog/src/glbl.v
vsim -L unisims testfixture_name glbl
```

This example targets the XC4000E/L/X, Spartan/XL, or Virtex families and assumes the UniSim libraries are properly compiled and named "unisims". For more information on the compilation of the ModelSim libraries, refer to http://www.xilinx.com/tech docs/1923.htm.

- **For post-route simulation,** enter the following:

```
vlog design.stim time_sim.v
$XILINX/verilog/src/glbl.v
vsim -L simprims testfixture_name glbl
```

This example is based on targeting the SimPrim libraries, which have been properly compiled, named, and mapped to simprims. Also, the name of the simulation netlist may change depending on how the file is created.

Note: Xilinx recommends giving the name "test" to the main module in the test fixture file. This name is consistent with the name of the test fixture module that is written later in the design flow by NGD2VER during post-route simulation. If this naming consistency is maintained, you can use the same test fixture file for simulation at all stages of the design flow with minimal modification.

Examples

```
module my_counter (CLK, D, Q, COUT);
input CLK, D;
output Q;
output [3:0] COUT;

wire GSR;
reg [3:0] COUT;

always @(posedge GSR or posedge CLK)
 begin
 if (GSR == 1'b1)
 COUT = 4'h0;
 else
 COUT = COUT + 1'b1;
 end

// FDCE instantiation
// GSR is modeled as a wire within a global module. So,
// CLR does not need to be connected to GSR and the flop
// will still be reset with GSR.

FDCE U0 (.Q (Q), .D (D), .C (CLK), .CE (1'b1), .CLR (1'b0));

endmodule
```

Because GSR is declared as a floating wire and is not in the port list, the synthesis tool optimizes the GSR signal out of the design. GSR is replaced later by the implementation software for all post-implementation simulation netlists. In the test fixture file, set GSR to **"test.uut.GSR"** (the name of the global set/reset signal, qualified by the name of the design instantiation instance name and the test fixture instance name). Because there is no STARTUP block, a connection to GSR is made in the test fixture via an assign statement.

In this example, the active high GSR signal in the XC4000 family device is activated by driving it High. 100 ns later, it is deactivated by driving it Low. (100 ns is an arbitrarily chosen value.)

You can use the same test fixture for simulating at other stages in the design flow if this method is used.

```
`timescale 1 ns / 1 ps
module test;
reg CLK, D;
wire Q;
wire [3:0] COUT;

reg GSR;
assign glbl.GSR = GSR;
assign test.uut.GSR = GSR;

my_counter uut (.CLK (CLK), .D (D), .Q (Q), .COUT (COUT));

initial begin
$timeformat(-9,1,"ns",12);
$display("\t T C G D Q C");
$display("\t i L S O");
$display("\t m K R U");
$display("\t e T");
$monitor("%t %b %b %b %b %h", $time, CLK, GSR, D, Q, COUT);
end

initial begin
CLK = 0;
forever #25 CLK = ~CLK;
end

initial begin
#0 {GSR, D} = 2'b11;
#100 {GSR, D} = 2'b10;
#100 {GSR, D} = 2'b00;
#100 {GSR, D} = 2'b01;
#100 $finish;
end

endmodule
```