XAPP108 (v2.0) May 22, 2000

HDL Simulation Using the Xilinx Alliance Series Software

Summary

This application note describes the basic flow and some of the issues to be aware of for HDL simulation with the Alliance Series software. The goal of this document is to familiarize the user with some of the concepts, but should not be considered a replacement for the Xilinx or HDL simulator's documentation. Please refer to the *Synthesis and Simulation Design Guide* and http://www.xilinx.com/apps/hdl.htm for more detailed information. Also, refer to the vendor documentation for specific information about a particular simulator or synthesis tool.

Introduction

Increasing design size and complexity, as well as recent improvements in design synthesis and simulation tools have made HDL the preferred design language of most designers for programming integrated circuit logic designs. The two leading synthesis and simulation languages today are Verilog and VHDL. Both of these languages have been adopted standards by IEEE, and the Xilinx Alliance Series software currently supports the Verilog IEEE 1364 Standard, VHDL IEEE Standard 1076.4 for Vital (Vital 95), and SDF version 2.1.

The Xilinx Alliance Series was designed to be used with several HDL synthesis and simulation tools to provide a solution for programmable logic designs from beginning to end. Libraries, netlist readers and netlist writers are provided along with the powerful place and route software that integrates with the HDL design environment.

Simulation Points

Xilinx supports functional and timing simulation of HDL designs at the following three points in the HDL design flow as shown in Figure 1.

1. Register Transfer Level (RTL) simulation which may include the following:
   - Instantiated UniSim library components
   - LogiBLOX modules
   - LogiCORE models
2. Post-synthesis functional simulation with one of the following:
   - Gate-level UniSim library components

   or
   - Gate-level pre-route SimPrim library components
3. Post-implementation back-annotated timing simulation with the following:
   - SimPrim library components
   - Standard Delay Format (SDF) file

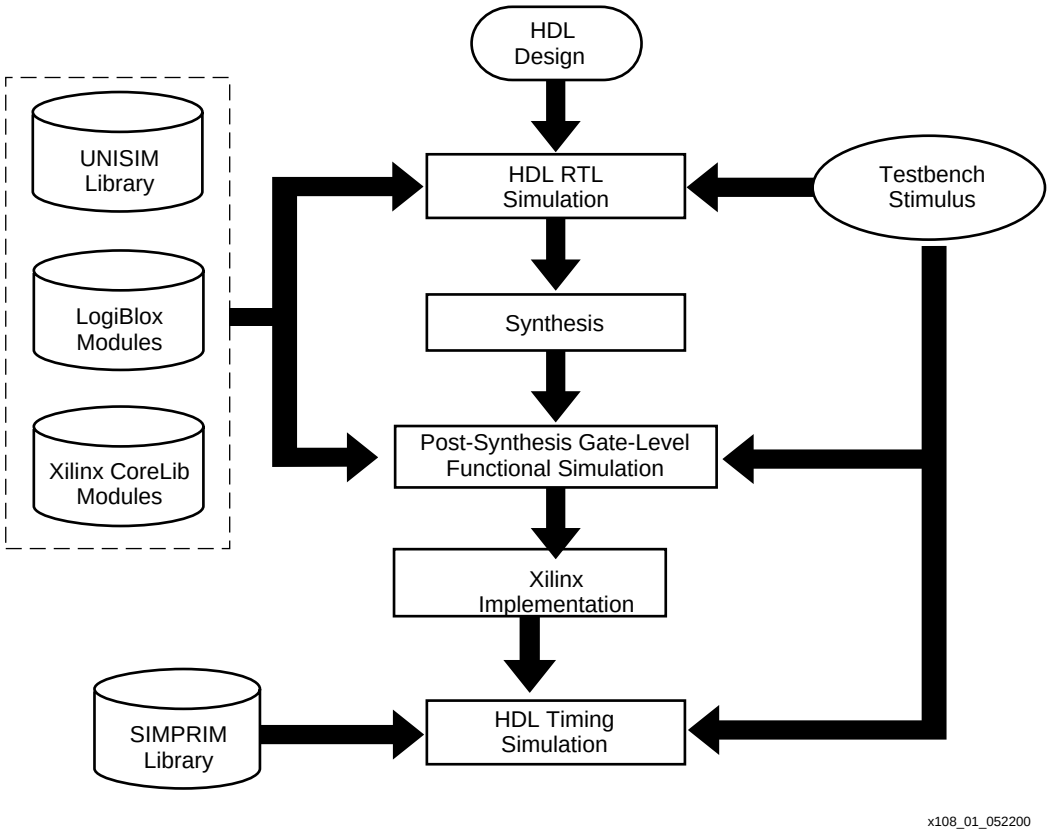

Figure 1: **Three Primary Simulation Points for HDL Designs**

RTL Simulation

The first simulation point is the RTL-level simulation that allows the user to verify or simulate a description at the system or chip level. This first pass simulation is typically performed to verify code syntax and to confirm that the functionality of the code is what is intended. At this step, no timing information is provided and simulation should be performed in a unit-delay mode to help the possibility of a race condition.

RTL simulation is not architecture-specific unless the design contains instantiated UniSim, COREGen, or LogiBLOX components. To support these instantiations, Xilinx provides the UniSim, LogiBLOX, and XilinxCoreLib library. The user can instantiate LogiBLOX or COREGen components if the user does not want to rely on the module generation capabilities of the synthesis tool, or if the design requires larger memory structures.

A general suggestion in the initial design creation is to keep the code behavioral. Avoid instantiating specific components unless necessary. This allows for more readable code, faster and simpler simulation, code portability (the ability to migrate to different device families) and code reuse (the ability to use the same code in future designs). However, at times it becomes necessary to instaniate components structurally in order to obtain the desired design structure rather than specifying behavioral code.

Post-Synthesis Simulation

Post synthesis simulation is used to verify the design functionality after the design has been synthesized and a structural representation of the design has been created. This may be done in any of three steps: post-synthesis, after design netlist translation (post-NGDBUILD) or post-mapping the design.

Post-Synthesis

Most synthesis tools have the capability of writing out a post-synthesis HDL netlist of the design. This may be used to simulate the design and evaluate the synthesis results, however, this is not supported by Xiinx. Since the netist is written in terms of the synthesis tool vendor's own simulation models, these models are not gauranteed by Xilinx.

Post-synthesis simulation is synthesis vendor-dependent, and the synthesis tool must write VHDL or Verilog netlists in terms of UniSim library components.

The models of instantiated LogiBLOX or COREGen is used for any post-synthesis simulation because the module is processed as a "black box" during synthesis. It is important that the initialization behavior is consistent for the behavioral model used for RTL and post-synthesis simulation and for the structural model used after implementation. In addition, the initialization behavior must work with the method used for synthesized logic and cores.

After Design Translation (Post NGDBUILD)

Simulation may also be performed after the netlist translation stage, NGDBUILD. Generally an HDL simulation may be performed here if the synthesis tool is not capable of writing out a structural simulation netlist in terms of UniSim components. The .ngd file produced from NGDBUILD is the input into one of the simulation netlisters, NGD2VER or NGD2VHDL. NGD2VER and NGD2VHDL create a structural simulation netlist based on the SIMPRIM models.

Post-Map

Simulation may also be performed after the maping of the design but before the place and route stage. This simulation will include the block delays for the design but not the routing delays. This is generally a good metric to test whether the design is meeting the timing requirements before spending the time to fully place and route the design.

As with the previous simulation, NGD2VER or NGD2VHDL will create the structural simulation netlist based on SIMPRIM models.

The delays for the design are stored in an SDF (Standard Delay Format) file which is created by the simulation netlister, NGD2VER or NGD2VHDL. This SDF file will contain all block or logic delays however will not contain any of the routing delays for the design since the design has not yet been placed and routed. All block delay values are worst case values. Actual device block delays are generally shorter under normal operating conditions.

Timing Simulation

A timing simulation netlist may be created after the design has completed placement and routing in the Alliance Series tools. It is not until this stage of design implementation that we get the full picture as to how the design will behave in circuit. The overall functionality of the design has been defined from the beginning stages but it is not until the design has been placed and routed that all of the timing information of the design can be accurately calculated.

As with the previous simulations, NGD2VER or NGD2VHDL creates a structural netlist based on SIMPRIM models, however, this netlist comes from the placed and routed .ncd file.

For timing simulation, an SDF file is created as with the after MAP simulation. However, this SDF file contains all block and routing delays for the design. As with the post-map version, all delays are worst case values.

During timing simulation, there are usually two occurrences to look for. First, ensure that the design is functioning as expected after placement and routing. Second, look for timing violations. Timing violations from the simulation are represented by a setup or hold time error message issued by the simulator.

Setup and Hold Times

The simulator will issue a setup or hold time violation any time data changes at a register input (data or clock enable) within the setup or hold time window for the particular register. These are a few typical causes of a setup or hold time violation:

- Data path delays are too long for clocking speeds.
- Clock skew is unaccounted for before simulation.
- Data paths cross out-of-phase or asynchronous clocks (to each other).
- Input data changing at the wrong time.

Some questions for debugging setup or hold time issues are:

- Did Trace or Timing Analyzer report the data path may run at speeds being clocked in simulation?
- Is clock skew being accounted for in this path delay? Was the clock path analyzed by Trace or Timing Analyzer? Does subtracting the clock path delay from the data path delay still allow clocking speeds?
- Will slowing down the clock speeds eliminate the setup/hold time violations?
- Does this data path cross clock boundaries (from one clock frequency to another)? Are the clocks synchronous to each other? Is there appreciable clock skew between these clocks?
- Is this path an input path to the device? Does changing the time at which the input stimulus is applied eliminate the setup/hold time violations?

Hopefully, the answers to these questions help determine the cause of the setup or hold time violation. Based on these responses, possible design changes may need to be made in order to accommodate the simulation conditions.

While Xilinx data sheets state that there are zero hold times on the internal registers and I/O registers with the default delay, it is still possible to receive a hold-time violation from the simulator. This hold-time violation is in reality a setup violation on the register but in order to get an accurate representation of the CLB delays, part of the setup time must be modeled as a hold time. For more information on this modeling, refer to http://support.xilinx.com/techdocs/782.htm.

Setup and hold violations may also occur on the write cycle of a RAM. Special care must be taken when writing to RAMs, especially asynchronous RAMs. It is suggested to use synchronous RAMs in the design whenever possible to simplify the write timing for a synchronous design.

VHDL/Verilog Libraries

The simulation points listed previously require the UniSim, SimPrim, LogiBLOX, and COREGen libraries.

UniSim Library

The UniSim library, used for functional simulation only, contains default unit delays. This library includes all the Xilinx Unified Library primitives that are inferred by most popular synthesis tools. Xilinx module generators (such as LogiBLOX or CORE Generator) should be used to generate higher order functions, such as adders, counters, and large RAMs.

UniSim Library Structure

The UniSim library directory structure is different for VHDL and Verilog. There is only one VHDL library for all Xilinx technologies because the implementation differences between architectures are not important for unit delay functional simulation except in a few cases where functional differences occur.

For example, the decode8 in XC4000 devices has a pull-up, and in XC5200 devices, it does not. In these few cases, configuration statements are used to choose between architectures for the components in VHDL. One library makes it easy to switch between technologies. It is left up to the user and the synthesis tool to use technology-appropriate cells.

For Verilog, separate libraries are provided for common technologies which share the same functionality. This combined library makes it easy to retarget to other technologies. It is not a requirement to change the library mapping statements when switching from Virtex™ to Virtex-E or Spartan™ to Virtex.

Some synthesis vendors have macros in their libraries, and can expand them to gates. The user can use the HDL output from synthesis to simulate these macros.

The VHDL UniSim Library source files are found in $XILINX/vhdl/src/unisims.

- unisim_VCOMP.vhd (component declaration file)
- unisim_VCOMP52K.vhd (substitutional component declaration file for XC5200 designs)
- unisim_VPKG.vhd (package file)
- unisim_VITAL.vhd (model file)
- unisim_VITAL52K.vhd (additional model file for XC5200 designs)
- unisim_VCFG4K.vhd (configuration file for XC4K edge decoders)
- unisim_VCFG52K.vhd (configuration file for XC5200 internal decoders)

The Verilog UniSim Library source files are found in the following locations:

- $XILINX/verilog/src/uni3000 (Series 3K)
- $XILINX/verilog/src/unisims (Series 4KE, 4KX, 4KL, 4KXV, Spartan, Virtex)
- $XILINX/verilog/src/uni5200 (Series 5200)
- $XILINX/verilog/src/uni9000 (Series 9500)

Note: Verilog reserves the names buf, pullup, and pulldown; the Xilinx versions are changed to buff, pullup1, pullup2, or pulldown2, and then mapped to the proper cell during implementation.

LogiBLOX Library

LogiBLOX is a module generator used for modules such as adders, counters, and large memory blocks. Refer to the *LogiBLOX Guide* for more information. The desired parameters can be entered into LogiBLOX and an HDL model selected as output. Most LogiBLOX modules contain registers and require the global set/reset (GSR) initialization. Since the modules do not contain output buffers going off-chip, the Global 3-state Enable (GTS) initialization does not apply.

LogiBLOX Library Structure

The LogiBLOX library is not a library of modules. It is a set of packages required by the LogiBLOX models that are created “on-the-fly” by the LogiBLOX tool.

The VHDL source files are in $XILINX/vhdl/src/logiblox.

- mvlutil.vhd
- mvlarith.vhd
- logiblox.vhd

Note: for Verilog, the LogiBLOX model is a structural netlist of SimPrim models. Do *not* synthesize this netlist; it is for functional simulation only.

CORE Generator Library

CORE Generator is also a module generator used for higher-order logic functions such as FIFOs, FIR filters, etc. The user can enter the desired parameters, and select the appropriate HDL model as output.

The CORE Generator models do not use library components for global signals.

CORE Generator Library Structure

The VHDL CORE Generator source files are found in $XILINX/vhdl/src/XilinxCoreLib.

The Verilog CORE Generator source files are found in $XILINX/verilog/src/XilinxCoreLib.

SimPrim Library

The SimPrim library is used for post-Ngdbuild (gate level functional), post-Map (partial timing), and post-place-and-route (full timing) simulations. This library is architecture independent.

SimPrim Library Structure

The VHDL SimPrim Library source files are found in $XILINX/vhdl/src/simprims.

The Verilog SimPrim Library source files are found in $XILINX/verilog/src/simprims.

Compiling HDL Libraries

For some simulators, such as NC-Verilog, VCS, VSS, and ModelSim, the user may need to compile the HDL libraries before using them for design simulations. The advantages of the compiled approach are speed of execution and economy of memory.

Xilinx provides an utility to specifically compile the HDL with today's most popular simulators. The utility is available at $XILINX/bin/<platform>/compile_hdl.pl

Running NGD2VER or NGD2VHDL

Creating a Simulation Netlist

In order to create the Verilog or VHDL simulation netlist, perform the following steps on the synthesis produced EDIF or XNF netlist, referenced here by `<design>`.

can create a simulation netlist from the Design Manager or from the command line as described in this section.

From the Design Manager

1. Select **`Setup`** → **`Options`** in the Flow Engine.
2. The Options dialog box appears.
3. Select the Produce Timing Simulation Data button in the Optional Targets field.
4. Select the Edit Template button next to the Implementation drop-down list in the Program Options Templates field.
5. The Implementation Template dialog box appears.
6. Select the Interface tab.
7. In the Simulation Data Options field, select applicable options as follows.
   - Format

   Specify the netlist format to use for simulation. The format is usually VHDL or Verilog for synthesis designs.
   - Correlate Simulation Data to Input Design

   This option enables signal back annotation to the original compiled netlist. By default, this option is on. Since many of the internal signal and instance names of the design were probably created by the synthesis tool and contain names that have no meaning to the

designer, this step is not usually necessary and disabling this feature will decrease the run time of the Alliance Series software.

- Bring Out Global Set/Reset Net as a Port

This option creates an external port in the simulation netlist to allow control of the power-on reset from a port. This option is not necessary for most simulators, and is off by default.

a. Click **OK** in the Implementation Template dialog box.

b. Click **OK** in the Options dialog box.

c. When implementing the design, the Flow Engine produces timing simulation data files.

From the Command Line

1. Run NGDAnno on the placed and routed .ncd file.

   For back-annotated output (signals correlated to original netlist), enter the following.

   ngdanno –p *design*.**pcf** *design*.**ncd** *design*.**ngm**

   For output that is not back-annotated (faster run time), enter the following.

   ngdanno *design*.**ncd**

2. Run the *NGD2XXX* program for the particular netlist to create.

   For VHDL, enter the following.

   ngd2vhdl [options] *design*.**nga**

   For Verilog, enter the following.

   ngd2ver [options] *design*.**nga**

Enabling "X" propagation

Introduced in the 3.1i release, the "X" propagation can be enabled or disabled in timing simulation on memory elements such as flops and memories.

For Verilog, "X" propagation is enabled by default. To disable "X" propagation use the **+no_notifier** option which is native to the simulator. This option disables the toggling of the notifier register argument of the timing check system tasks. By default, the notifier is toggled when there is a timing check violation, and the notifier usually causes a UDP to propagate an "X". Therefore, the **+no_notifier** option suppresses "X" propagation on timing violations.

For VHDL, this is done with the **-xon** option within NGD2VHDL. The **-xon** option specifies the output behavior when timing violations occur on synchronous elements. If this option is set to equal true, any synchronous elements that violate a setup time trigger "X" on the outputs. If the option is set to equal false, the signal's previous value is retained. If this option is not set by the user, **-xon** is set to true by default.

Min/Typ/Max Simulation

While simulating, the user has the option of specifying one delay mode from either mindelays, typdelays, or maxdelay . However, the tripilicate values (mindelays, typdelays, and maxdelays) within a SDF file all hold the maxdelays since the Xilinx Alliance tools state the worst case numbers. Currently, it is not possible to generate one SDF file that contains separate values for the Min:Typ:Max fields.

The user can specify to get the minimum, but all three fields of the triplicate will contain the mindelays. To generate a simulation netlist with mindelays, type the following:

ngdanno -s min *design*.**ncd**

For VHDL, enter the following.

ngd2vhdl [options] *design*.**nga**

For Verilog, enter the following.

ngd2ver [options] *design*.**nga**

Mindelays may only be available for select FPGA families. For further details, please see http://support.xilinx.com/techdocs/4506.htm.

Prorating Simulation

Prorating is a linear scaling operation on existing speed file delays and is applied globally to all delays. The prorating constraints, VOLTAGE and TEMPERATURE, provide a method for determining timing delay characteristics based on known environmental parameters.

The VOLTAGE constraint provides a means of prorating delay characteristics based on the specified voltage. The UCF syntax is as follows:

VOLTAGE=*value*[V]

Where *value* is an integer or real number specifying the voltage and units is an optional parameter specifying the unit of measure.

The TEMPERATURE provides a means of prorating device delay characteristics based on the specified junction temperature. The UCF syntax is as follows:

TEMPERATURE=*value*[C |F| K]

Where *value* is an integer or a real number specifying the temperature. C, K, and F are the temperature units: F is degrees Fahrenheit, K is degrees Kelvin, and C is degrees Celsius, the default.

Note: Each architecture has its own specific range of valid operating temperatures and voltages. If the entered temperature or voltage does not fall within the supported range, the constraint is ignored and an architecture-specific default value is used instead.

For simulation, the VOLTAGE and TEMPERATURE constraints will be processed from the UCF file into the PCF file.

ngdanno -p *design*.**pcf** *design*.**ncd**

For VHDL, enter the following:

ngd2vhdl [options] *design*.**nga**

For Verilog, enter the following:

ngd2ver [options] *design*.**nga**

Note: Do not combine both minimum timing and prorating (-s and -p) as minimum values would override prorating.

Prorating may only be available for select FPGA families, and it is not intended for military and industrial ranges. It is applicable only within the commercial ranges.

Understanding the Global Signals for Simulation

Xilinx FPGAs have register (flip-flops and latches) set/reset circuitry that pulses at the end of the configuration mode and after power-up. This pulse is automatic and does not need to be programmed. All the flip-flops and latches receive this pulse through a dedicated global set/reset (GSR), or reset (GR) net. The registers either set or reset, depending on how the registers are defined.

It is important to address the built-in reset circuitry behavior in the designs starting with the first simulation to ensure that the simulations agree at the three primary points.

If the user does not simulate GSR behavior prior to synthesis and place and route, the RTL and possibly post-synthesis simulations will not initialize to the same state as the post-route timing simulation. As a result, various design descriptions are not functionally equivalent and simulation results will not match. In addition to the behavioral representation for GSR, a Xilinx implementation directive needs to be added. This directive is used to specify to the place and

route tools to use the special purpose GSR net that is pre-routed on the chip, and not to use the local asynchronous set/reset pins. Some synthesis tools can identify, from the behavioral description, the GSR net, and will place the STARTUP module on the net to direct the implementation tools to use the global network. However, other synthesis tools interpret behavioral descriptions literally, and will introduce additional logic into the design to implement a function. Without specific instructions to use device global networks, the Xilinx implementation tools will use general purpose logic and interconnect resources to redundantly build functions already provided by the silicon.

Even if GSR behavior is not described, the actual chip initializes during configuration, and the post-route netlist will have this net that must be driven during simulation.

In addition to the set/reset pulse, all output buffers are tristated during configuration mode and after power-up with the dedicated global output tristate enable (GTS) net.

See Table 1 for a listing of other Xilinx device's global signal names and polarities.

Table 1: **Global Reset and Tristate Names for Xilinx Devices**

Device Family	Global Reset Name	Global Tristate Name	Default Reset Polarity
XC3000	GR	Not Available	Low
XC4000	GSR	GTS	High
XC5200	GR	GTS	High
SPARTAN	GSR	GTS	High
Virtex	GSR	GTS	High

These GSR and GR nets require special handling during synthesis, simulation, and implementation to prevent them from being assigned to normally routed nets, which uses valuable routing resources and degrades design performance. The GSR or GR net receives a reset-on-configuration pulse from the initialization controller, as shown in Figure 2.

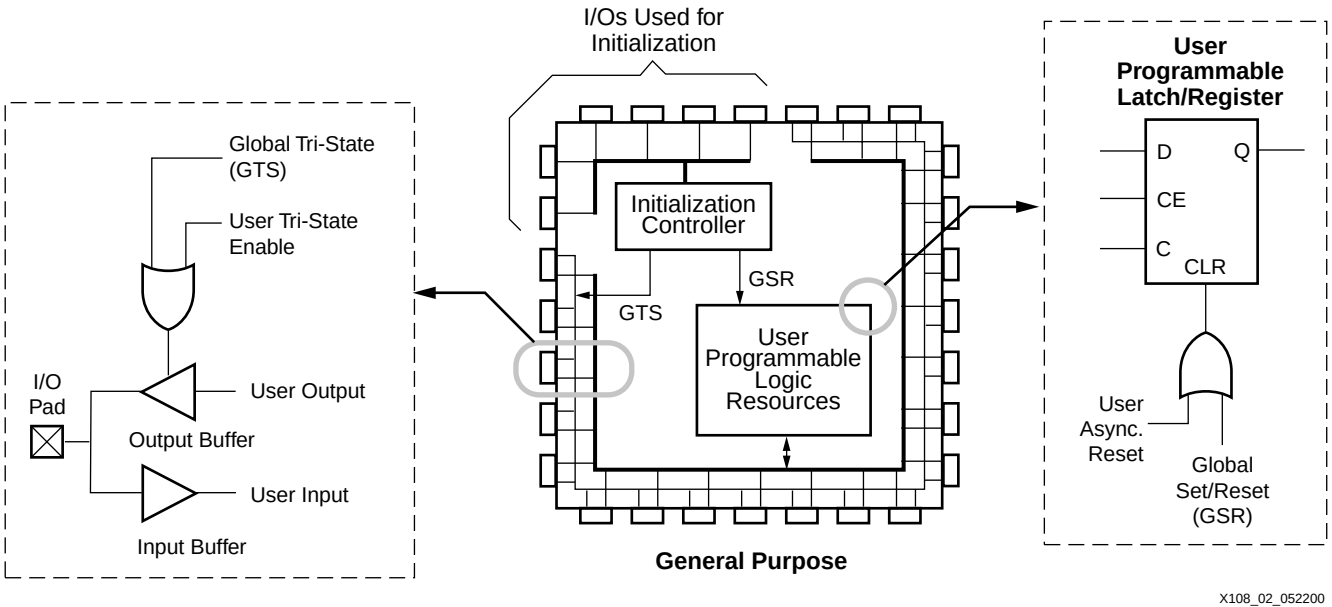

Figure 2: **Built-in FPGA Initialization Circuitry**

This pulse occurs during the configuration or power-up mode of the PLD. However, for ease of simulation, it is usually inserted at time zero of the test bench, before logical simulation is initiated. The pulse width is device-dependent and can vary widely, depending on process voltage and temperature changes. The pulse is guaranteed to be long enough to overcome all net delays on the reset special-purpose net. The parameter for the pulse width is T_{MRW} (Minimum Reset Width) as described in each product specific data sheet.

The high impedance on-configuration circuit shown in Figure 2 also occurs during the configuration or power-up mode of the PLD. Just as for the reset-on-configuration simulation, it is usually inserted at time zero of the test bench before logical simulation is initiated. The pulse drives all outputs to the tristate condition they are in during the configuration of the PLD. All general-purpose outputs are affected whether they are regular, tristate, or bidirectional outputs during normal operation. This ensures that the outputs do not erroneously drive other devices as the PLD is being configured. The pulse width is device-dependent and can vary widely with process and temperature changes. The pulse is guaranteed to be long enough to overcome all net delays on the GTS net. The generating circuitry is separate from the reset-on-configuration circuit. The pulse width parameter is T_{GTS} as described in each product specific data sheet. Simulation models use this pulse width parameter for determining HDL simulation for global reset and tristate circuitry.

If a global set/reset is desired for behavioral simulation, it must be included in the behavioral code. Any described register in the code must have a common signal that will asynchronously set or reset the register depending on the desired result. Similarly, if a global high-impedance state is desired for simulation, it should be described in the code as well.

Defining Global Signals in VHDL

The UNISIM library also includes a few models to assist in Vital VHDL simulation of the global set/reset and High-Z signals. Depending on the desired circuit, one or a combination of the following UNISIM models should be connected to the described global set/reset and/or High-Z signals in the design.

If it is desired to have either the Global Set/Reset (GSR) or Global 3-state (GTS) signal as a port in the implemented design, it is suggested to use the STARTBUF cell for simulation and implementation for non Virtex and non Spartan-II devices. For Virtex and Spartan-II devices, it is suggested to use the local routing resources, unless routing is congested. If it is desired to have access to the GSR signal during simulation but not implementation, it is suggested to use the ROCBUF cell. Similarly, if it is desired to have access to the GTS signal during simulation but not implementation, use the TOCBUF cell. If it is not important to have access to these global signals but if the user still wishes to simulate the startup characteristics of the device, use the ROC and TOC cells.

ROC (Reset On Configuration)

The ROC cell is a VHDL simulation-only component used to generate a single high pulse for simulating the global clearing or setting of registers that occurs in an implemented design during device configuration. This cell may be used in the simulations throughout the design process. Figure 3 depict models for ROC cell simulation and implementation. The ROC cell is reinserted when the VHDL netlist for timing is written out. The default pulse width for the ROC cell is 100 ns for functional and timing simulation. To change ROC pulse width value in timing simulation, use the NGD2VHDL **-rp** switch to specify the pulse width.

The ROC cell must be instantiated into the code to connect to the design. The following is an example of such an instantiation:

```
<instance_name>: ROC port map (O => GSR_NET);
```

The width of the ROC initialization pulse is generally passed in the design HDL testbench. The proper value for the reset pulse width may be found in the data book for a particular device by finding the T_{MRW} specification.

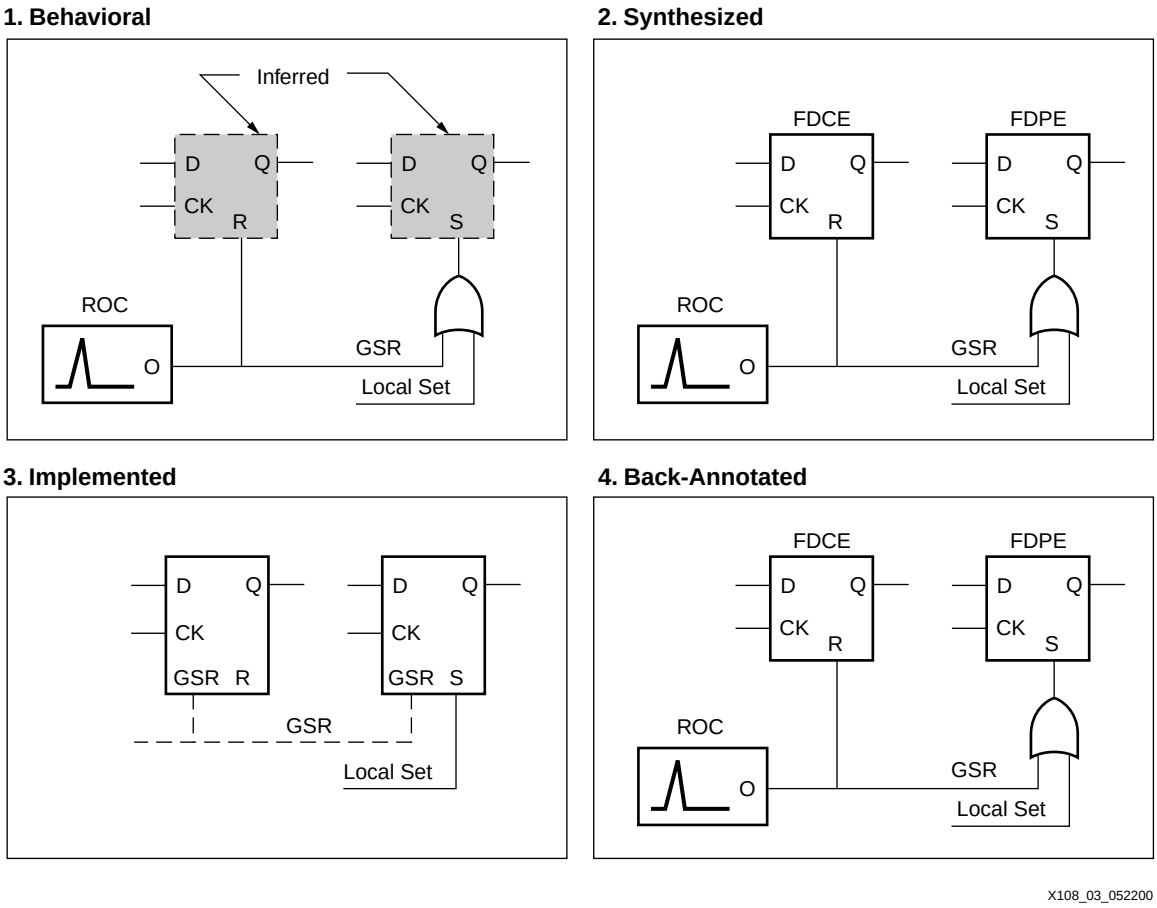

Figure 3: **ROC Simulation and Implementation**

ROCBUF (Reset On Configuration Buffer)

The ROCBUF cell is used in similar VHDL cases to the ROC cell but instead of providing a component to drive the net, it allows the user's testbench to drive the net without actually implementing it on chip. Therefore the user can issue the reset initialization pulse and subsequent reset pulses during the simulation via the simulation reset port if multiple configuration cycles are being simulated, however, the simulation reset port will not appear as an input pin in the implemented design.

As with the ROC cell, the ROCBUF cell must be instantiated into the code, however, this added port must now be driven from the testbench file. If this port is desired for the timing simulation as well, use the **–gp** switch for NGD2VHDL to recreate the simulation reset port.

TOC (3-state On Configuration)

The TOC cell is used in VHDL to generate a single high pulse for simulating the global 3-stating of the I/Os that occurs in an implemented design during device configuration. Figure 4 depict the simulation and implementation models for the TOC cell. The default pulse width for the TOC cell is 100 ns for functional and 0 ns for timing simulation. To change TOC pulse width value in timing simulation, use the NGD2VHDL **-tpw** switch to specify the pulse width.

The TOC cell is reinserted when the VHDL timing netlist is written. As with the ROC cell, the TOC cell must be instantiated into the code. The following is an example of this instantiation:

```
<instance_name>: TOC port map (O => GTS_NET);
```

Typically, the duration of the 3-state is similar to that of power on reset. So the same value, T_{POR}, can be used for TOC cell. These times may change depending on the device, configuration method and options chosen.

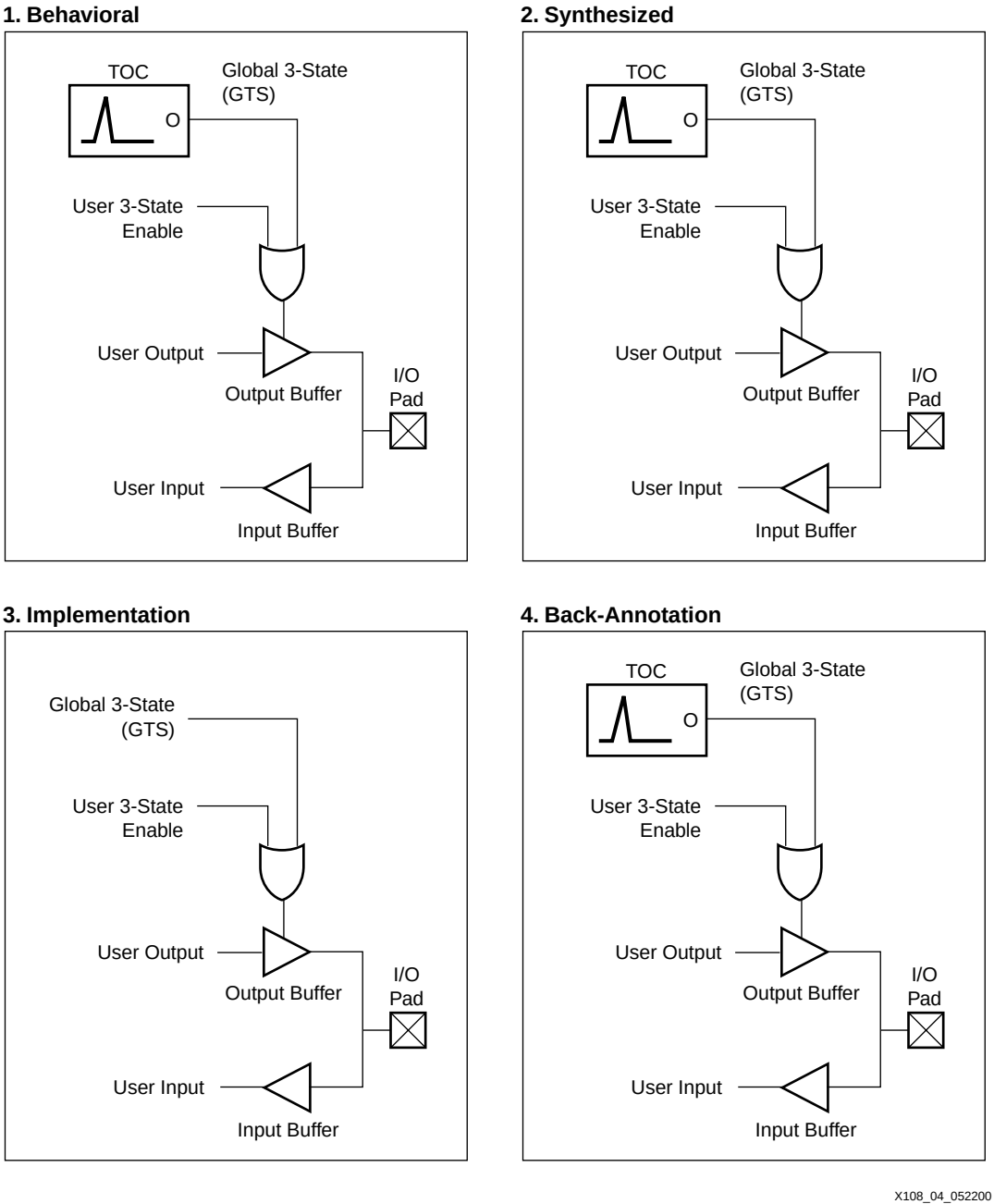

Figure 4: **TOC Simulation and Implementation**

TOCBUF (3-state On Configuration Buffer)

The TOCBUF cell allows VHDL test benches to access to the net connected to the 3-state pins of a design's I/Os via a user-defined input port referred to as the simulation 3-state port. Therefore, the user can not only issue the reset initialization pulse, emulating the 3-stating of I/Os in the design during configuration, but can also issue subsequent initialization pulses via the simulation 3-state port if multiple configuration cycles are being simulated. However, that the simulation 3-state port will not appear as an input pin in the implemented design.

As with the TOC cell, the TOCBUF cell must be instantiated into the code however this 3-state simulation port must now be driven by the testbench file. If this port is desired for the timing simulation as well, use the **–tp** switch for NGD2VHDL to create the simulation 3-state port.

STARTBUF (Startup Buffer)

The STARTBUF cell allows VHDL designs access to all of the input and output ports of the STARTUP cell, and adds two extra ports (GSROUT and GTSOUT) for simulation purposes. The port names for the STARTBUF cell differ slightly from that of the STARTUP cell. See Table 2 for details.

Note: for Virtex and Spartan-II devices, please see the section STARTBUF_VIRTEX and STARTBUF_SPARTAN2 (Startup Buffer)

Table 2: **STARTBUF/STARTUP Pin Descriptions**

STARTBUF Pin Name	Connection Point	XC4000 STARTUP Pin Name	XC5200 STARTUP Pin Name	Spartan
GSRIN	Global Set/Reset Port of Design	GSR	GR	GSR
GTSIN	Global 3-state Port of Design	GTS	GTS	GTS
GSROUT	All Registers Asynchronous Set/Reset	Not Available For Simulation Only	Not Available For Simulation Only	Not Available For Simulation Only
GTSOUT	All Output Buffers 3-state Control	Not Available For Simulation Only	Not Available For Simulation Only	N/A
CLKIN	Port or INternal Logic	CLK	CLK	CLK
Q2OUT	Port Or Internal Logic	Q2	Q2	Q2
Q3OUT	Port Or Internal Logic	Q3	Q3	Q3
OUT	Port Or Internal Logic	Q1Q4	Q1Q4	Q1Q4
DONEINOUT	Port Or Internal Logic	DONEIN	DONEIN	DONEIN

The input ports GSRIN and GTSIN can be connected either directly or indirectly via combinational logic to input ports of the design. The design input ports will appear as input pins in the implemented design. Note that use of the STARTBUF implies that the design has user-defined signals driving the global set/reset and/or 3-state resource(s) available in the

implemented design. This is in addition to the automatic pulse that occurs during configuration. The STARTBUF is not reinserted, and the GSR/GTS are directly wired to the ports.

Figure 5 and Figure 7 show how the STARTBUF is connected for Functional and Timing simulation while Figure 6 depicts the implemented circuit.

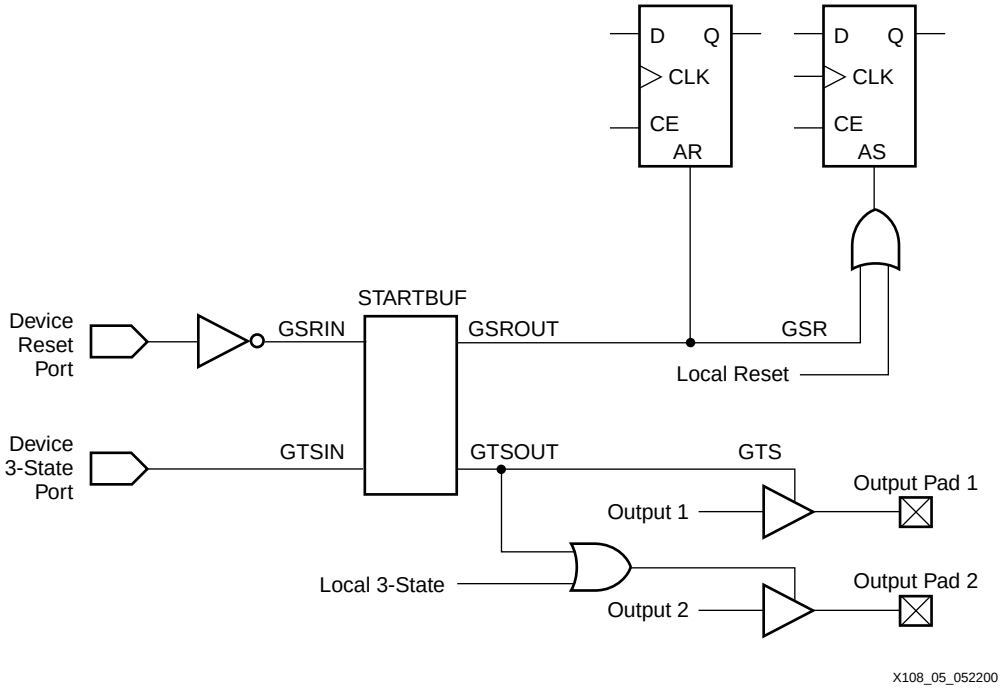

Figure 5: **Model of STARTBUF Cell for Behavioral and Post-synthesis Simulation**

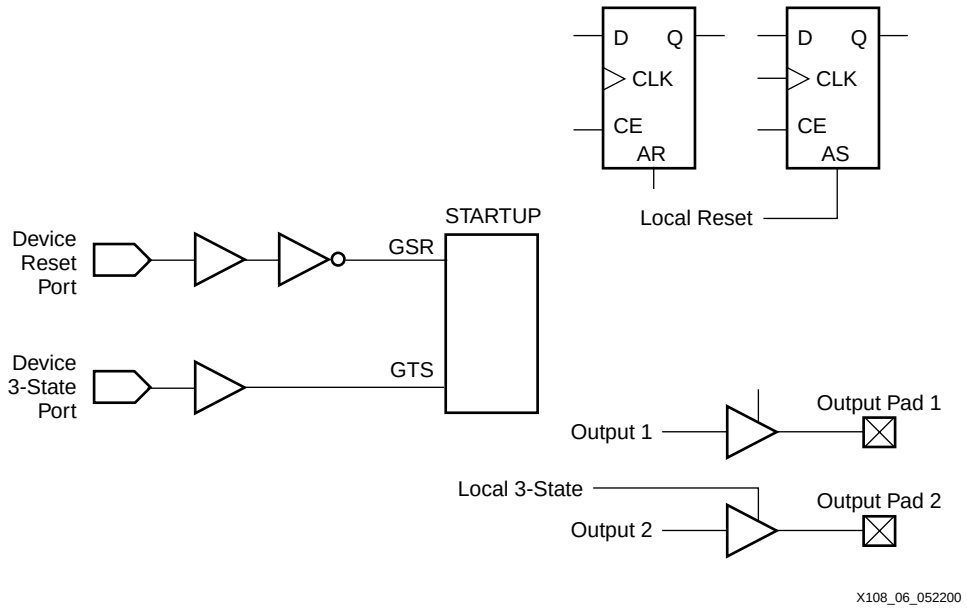

Figure 6: **Diagram of Implementation Circuit After Logic Trimming**

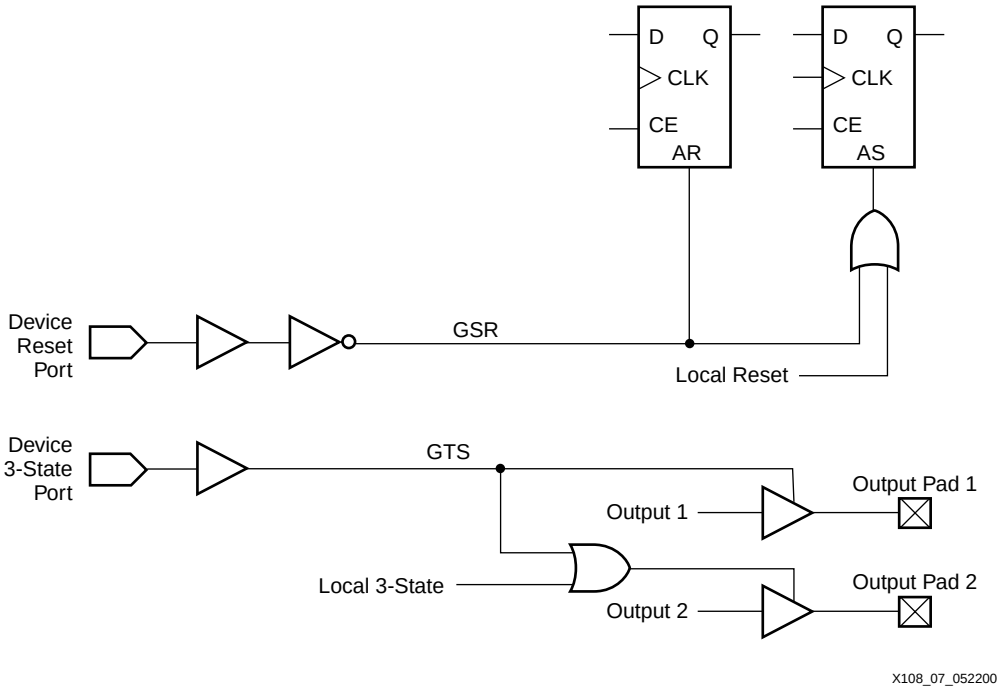

Figure 7: **Model of Design Using STARTBUF for Timing Simulation**

STARTBUF_VIRTEX and STARTBUF_SPARTAN2 (Startup Buffer)

If the global reset or tristate is connected to a chip pin, the STARTUP_VIRTEX and STARTUP_SPARTAN2 blocks can be instantiated to identify the GSR signals for implementation. However, these cells can not be simulated as there is no simulation model for them. The VHDL STARTBUF_VIRTEX and STARTBUF_SPARTAN2 blocks can not be used to simulate the GSR signal during any pre-NGDBuild Unisim VHDL simulations. The design cannot be reset after simulation time "0". However, the design will start up in the correct state, and a pre-NGDBuild UniSim simulation of the GTS signal can be implemented.

Post-NGDBuild SimPrim VHDL simulation of GSR is supported. To correctly back-annotate a GSR signal, instantiate a STARTUP_VIRTEX, STARTBUF_VIRTEX, STARTUP_SPARTAN2, or STARTBUF_SPARTAN2 symbol and correctly connect the GSR input signal of the component. When back annotated, the GSR signal is correctly connected to the associated registers and RAM blocks.

For the port names for the STARTBUF cells please see the following Table 3 for details.

Table 3: **Virtex and Spartan-II STARTBUF/STARTUP Pins**

STARBUF Pin Names	Connection Points	Virtex STARTUP Pin Names	Spartan-II STARTUP Pin Names
GSRIN	Global Set/Reset Port of Design	GSR	GSR
GTSIN	Global Tristate Port of Design	GTS	GTS
CLKIN	Port or Internal Logic	CLK	CLK
GTSOUT	All Output Buffers 3-state Control	N/A	N/A

Xilinx recommends that the user use the local routing for Virtex and Spartan-II devices as opposed to using the dedicated GSR. If the design resources are available using this method will provide better performance. If the GSR pin out will not be connected to a device pin, but want to have access to it for simulation, use the ROC or ROCBUF with any device including Virtex and Spartan-II.

Defining Global Signals in Verilog

The general procedure for specifying global set/reset or global reset involves the definition of the global reset signals with the $XILINX/verilog/src/glbl.v module. The VHDL UniSims library contains the ROC, ROCBUF, TOC, TOCBUF, and STARTBUF cells to assist in VITAL VHDL simulation of the global set/reset and tristate signals. However, Verilog allows a global signal to be modeled as a wire in a global module, and, thus, does not contain these cells.

Using the glbl.v Module

The glbl.v module connects the global signals to the design, which is why it is necessary to compile this module, and load it along with the design.v file and the testfixture.v file for simulation.

```
‘timescale 1 ns / 1 ps

module glbl();

wire GR;
wire GSR;
wire GTS;
wire PRLD;

endmodule
```

Defining GSR/GTS in a Test Bench

There are two cases to consider: designs without a STARTUP block and designs with a STARTUP block.

Designs without a STARTUP Block

For RTL simulation using the UniSim libraries, the value of the appropriate Verilog global signals (glbl.GSR, glbl.GR, or glbl.GTS) must be set to the name of the GSR, GR, or GTS net, qualified by the appropriate scope identifiers.

The global set/reset net is present in the implemented design even if the STARTUP block in the design is not instantiate. The function of STARTUP is to give the user the option to control the global reset net from an external pin. The following should be added to the design code and test fixture:

- To set the GSR and GTS pin of the XC4000XLA, Spartan/XL, or Virtex devices

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

reg GTS;
assign glbl.GTS = GTS;

initial begin
GSR = 1; GTS = 1;
#100 GSR = 0; GTS = 0;
end
```

- To set the GR pin of the XC3000 devices

```
reg GR;
assign glbl.GR = GR;

initial begin
GR = 0;
#100 GR = 1;
end
```

- To set the GR and GTS pin of the XC5200 devices

```
reg GR;
assign glbl.GR = GR;

reg GTS;
assign glbl.GTS = GTS;

initial begin
GR = 1; GTS = 1;
#100 GR = 0; GTS = 0;
end
```

Designs with a STARTUP Block

For RTL simulation using the UniSim libraries, asserting global set/reset and global 3-state when the STARTUP block is specified in the design is similar to asserting global set/reset and global 3-state without a STARTUP block in the design. The GSR/GR pin is connected to an external input port, and glbl.GSR/glbl.GR is defined within the STARTUP block to make the connection between the user logic and the global GSR/GR net embedded in the UniSim models.

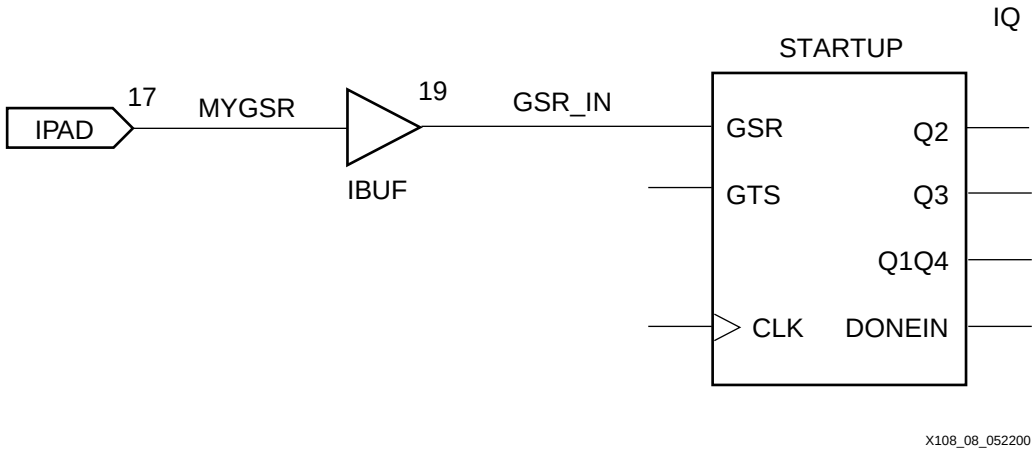

Figure 8: **User-Controlled GSR**

Consider Figure 8 as an example. To set the GSR pin to set an user defined port, the testfixture would be written as the following:

```
reg MYGSR;


initial begin
MYGSR = 1;
#100 MYGSR = 0;
end
```

The assign statement for the global signal must be omitted. This is because the net connections exist in the UniSim STARTUP model for RTL simulation and the post-NGDBuild Verilog netlist that is created by NGD2VER after implementation. Retaining the assign definition causes a possible conflict with these connections.

Conclusion

As Xilinx devices continue to grow in density and complexity, it becomes increasingly important to create simple yet powerful design simulation and implementation tools. Xilinx recognizes this need and will continue to focus on the HDL design entry and verification as an important part of the design process and will continue its integration into this design methodology.

Visit the Xilinx World Wide Web site for the latest information about Xilinx devices and software at **http://www.xilinx.com**.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/21/98	1.0	Initial Release
05/22/00	2.0	Major updated to reflect current products and updated format.