How to Use the Global Set/Reset (GSR) Signal

This topic provides guidelines and specific instructions for using the Global Set/Reset Interface (GSR) signal of a Lattice FPGA device simulation model for use with all Lattice FPGAs.

Details regarding the architecture of global signals are described in the data sheet for each device family on the Lattice Web site. For more information on the GSR library element, see the FPGA Libraries Help system.

The items presented in this topic are as follows:

- GSR Hardware Resource
- GSR Usage Cases
  - Inferred GSR
  - User Specified Inferred GSR
  - Global GSR
  - LSR (No GSR)
- Using the GSR Resource with Lattice IPs
- RTL Functional Simulation and the GSR Resource
- Mixed Language Simulation and the GSR Resource
- Block Modular Design (BMD) GSR Considerations
- Comparison of ispLever 7.2 GSR Flow Versus Previous Software Versions
- Using Attributes to Control GSR Resource Usage

GSR Hardware Resource Lattice FPGAs contain both GSR (Global Set Reset) and PUR (Power Up Reset) resources. The GSR hardware resource in Lattice FPGAs provides a convenient mechanism to allow design components to be reset without using any routing resources.

Note: The PUR resource is used to reset the device after configuration is complete when the device is powered on. A PUR component is provided to allow simulation testbenches to simulate this pulse, but this component is never used as part of the design. For more information on PUR, see the How to Use the Power Up Set/Reset (PUR) Global Signal topic.

There are two primary ways to take advantage of the GSR hardware resource in your design: *use the GSR to reset all components on your FPGA* or to *use the GSR to eliminate any routing resources needed for one reset in a multiple reset design*. If there is only one reset signal for the entire design, you would want to use the GSR in the first way, to reset all components on the FPGA. When using the GSR to eliminate any routing resources needed for one reset in a multiple reset design, typically the GSR would be used for the reset with the highest fan-out.

Note: The GSR should only be used for asynchronous active low resets since the hardware will always assert GSR asynchronously. The software will take

this into account automatically and will not connect a synchronous reset to GSR when the Inferred GSR or User Specified Inferred GSR modes are used, However there are ways to override the software default behavior. You are responsible for ensuring correct functionality when overriding the normal software defaults or when using the Global GSR usage mode. The following sections of this document detail how to take advantage of this resource with the Lattice ispLEVER design software.

GSR Usage Cases In ispLEVER, there are four usage cases with respect to initialization set/resets: *Inferred GSR*, *User-Specified Inferred GSR*, *Global GSR*, and *LSR (No GSR)*.

The four GSR usage cases are defined in the bullet list below:

- *Inferred GSR* – In this case the software automatically determines which reset signal has the highest fan-out (for either single or multiple reset designs) and uses the GSR resource as the routing for that reset signal. This usage case is the default condition in ispLEVER if there is no user-instantiated GSR component in the design.

  This usage case is the best choice for most applications. The software determines the reset with the most loads and uses the GSR resource for that signal which provides the largest reduction in needed routing resources. The Inferred GSR usage case can also be used whether the design has a single or multiple resets.
- *User-Specified Inferred GSR* – This is the same as the Inferred GSR usage except that the reset signal that is specified in the preference (.lpf) file determines which signal uses the GSR resource regardless of the fan-out of the signal.
- *Global GSR* – This case treats the GSR resource as a reset for all elements in the design.
- *LSR (No GSR)* – LSR (local set/reset) specifies that no GSR is to be used, that is, that all resets will use local routing resources instead of using the GSR resource.

Note: In the Inferred GSR and User-Specified Inferred GSR usage cases, the software will only connect elements with an asynchronous reset to GSR. Elements requiring a synchronous reset will use only local routing.

Inferred GSR The Inferred GSR usage case is the simplest to use. If everything is left to default software settings and no GSR component is instantiated in the design, then the software will implement a design using GSR for the reset signal with the highest fan-out.

Inferred GSR is the recommended usage case unless any of the following conditions exist:

- If you need to use a specific reset signal for GSR.
- If you need to globally use GSR even when there are multiple resets.
- If you need to completely disable GSR.

To use the Inferred GSR usage case, there are no design changes necessary. You simply implement the design using ispLEVER with default settings. The necessary software settings are listed below.

Inferred GSR usage case software settings:

- Synplify Synthesis Properties: **Force GSR: False (default)**
- Precision Synthesis Properties: **Force GSR: False (default)**
- Map Properties: **Infer GSR: True (default)**
- Preferences: **Do not specify GSR_NET in the .lpf file**

User Specified Inferred GSR The User Specified Inferred GSR usage case is identical to the Inferred GSR usage case except that the reset signal on which GSR is inferred is specified in the preference (.lpf) file instead of being automatically determined by fan-out.

If default software settings are enabled and no GSR component is instantiated in the design, and a preference exists for GSR_NET, then the software will implement a design using GSR for the reset signal specified by GSR_NET. This usage case is best for a design with multiple resets where a specific reset is required to use GSR regardless of the fan-out of the net.

To use the User-Specified Inferred GSR usage case, there are no design changes necessary. You simply implement the design using ispLEVER with default settings and a GSR_NET preference specified. The necessary software settings are listed below.

User-Specified Inferred GSR usage case software settings:

- Synplify Synthesis Properties: **Force GSR: False (default)**
- Precision Synthesis Properties: **Force GSR: False (default)**
- Map Properties: **Infer GSR: True (default)**
- Preferences: **Specify GSR_NET in the .lpf file.** The syntax of the preference is GSR_NET net “reset_signal_name”.

Global GSR The Global GSR usage case is intended for all elements in a design to be reset using the GSR resource. This usage is a good fit for a design with a single reset. It can also be used with multiple resets in the design, but it can produce unexpected functionality in this case. For example, if the GSR resource is used for the reset with the largest fan-out, elements on a second reset signal will still be reset by the first reset signal. Additionally, any registers with a synchronous reset will be attached to their local reset as well as the GSR reset (which asserts asynchronously).

Note also that the GSR resource is active low. In the case of a register using an active high reset, that register will remain connected to the signal but will have GSR enabled. This will cause the register to remain in reset. If a mix of active low and active high resets are used in a design or a mix of synchronous and asynchronous resets are used, then the Inferred GSR usage case should be used. In the Inferred GSR usage case, the software will automatically take into account whether a reset is synchronous, asynchronous, active high, or active low. In the Global GSR usage case, the software assumes the user

knows what the design intent is and makes minimal design changes in order to allow the user intent to be implemented. Global GSR is meant for a true global reset application.

To use the Global GSR usage case, a GSR component must be instantiated in the design and connected to the signal that is targeted as the reset signal, usually a primary input. If a GSR component is not instantiated in the design, the software will not treat the design as a Global GSR usage case. The GSR component must be instantiated into the design itself, not into the testbench for the design. Below are examples of how to instantiate the GSR library element in both Verilog and VHDL.

GSR VHDL Example:

```
GSR_INST:   GSR port map (GSR=><global reset sig>);
```

GSR Verilog HDL Example:

```
GSR     GSR_INST (.GSR (<global reset sig>));
```

Note that in Verilog the GSR instantiation must be in the top-level module of the design and it must be named GSR_INST. The necessary software settings are listed below.

Global GSR usage case software settings:

- Synplify Synthesis Properties: **Force GSR: False (default)**
- Precision Synthesis Properties: **Force GSR: False (default)**
- Map Properties: **Infer GSR: True (default)**. If a GSR component is already instantiated, this property will be ignored.
- Preferences: **Do not specify GSR_NET in the .lpf file**. If a GSR_NET preference is specified and a GSR component is already instantiated, the preference will be ignored.

LSR (No GSR) The LSR (local set/reset) usage case always uses local routing for the reset signals and does not use the GSR resource. This is the recommended usage case if there is a requirement to do timing analysis on the reset signals or if synchronous reset is being used throughout the design.

To use the LSR usage case there must be no GSR instantiated in the design, no GSR_NET preference specified, and the software settings used do not infer any GSR resource. The necessary software settings are listed below.

LSR (No GSR) usage case software settings:

- Synplify Synthesis Properties: **Force GSR: False (default)**
- Precision Synthesis Properties: **Force GSR: False (default)**
- Map Properties: **Infer GSR: False**
- Preferences: **Do not specify GSR_NET in the .lpf file**.

Using the GSR Resource with Lattice IPs Lattice IPs use the GSR resource when running in evaluation mode and may require it on an individual basis for performance reasons. How the GSR options should be used with Lattice IPs varies depending on whether the IP was released prior to the ispLever 7.2 software release or after. See the sections below for details.

Lattice IP Released Prior to ispLever 7.2 Software Release Lattice IP released prior to the ispLever 7.2 software release included an option in the IP generator to enable or disable GSR. If the GSR option is enabled on these earlier IPs, then a GSR component is instantiated and placed on the IP reset signal. Using these earlier IPs with this option enabled and ispLEVER 7.2 or later will cause the design to be implemented as a Global GSR usage case. This will occur even if there are multiple resets in the design. If there is already a GSR component instantiated in the design and it is connected to a different reset signal, then an error will occur when the design is mapped.

Recommended: For IPs released prior to the ispLever 7.2 software release, you should generate them with the GSR option set to disabled.

Note that when the IP GSR option is disabled and if all of the following conditions are met, then the IP cannot be used in evaluation mode:

- If an Inferred GSR, User-Specified Inferred GSR, or LSR (No GSR) usage case is used
- If the reset signal connected to the IP does not have GSR inferred on it

However, the IP can be used if an IP license is present even if the IP reset signal does not use GSR. To use an earlier IP with ispLEVER 7.2 or later in evaluation mode, the reset signal to the IP must be connected to GSR either through the IP generator option for GSR, a GSR component placed in the design by the user on the IP reset signal, or by GSR being inferred on the reset signal connected to the IP.

Lattice IP Released After ispLever 7.2 Software Release For Lattice IP released after the ispLEVER 7.2 software release, no individual IP GSR option is available. These IPs will function correctly in all usage cases for GSR. For any IP that requires the use of GSR on it's reset for performance issues, the User-Specified Inferred GSR usage case or Global GSR usage case should be used. For the User-Specified Inferred GSR usage case, the reset signal the IP is connected to should be entered in the preferences as the GSR_NET preference. Any IPs released after ispLEVER 7.2 software release are not compatible with earlier ispLever software releases. Using these IPs in earlier software releases will result in errors in the implementation flow.

RTL Functional Simulation and the GSR Resource If the GSR resource is used as a routing replacement for one of the reset signals in a design using the Inferred GSR or User-Specified GSR usage cases, then the functional simulation will be correct for both the RTL (pre-synthesis) design and the netlist (post-synthesis and post-map) versions of the design.

If the GSR resource is used as a global reset, however, the RTL functional simulation will not correctly simulate the reset functionality if the design uses multiple resets. Global GSR causes all Lattice components in the design to respond to the reset used for the GSR whether they are connected to that

reset or not. In the RTL representation the use of the GSR resource is not modeled in synthesizable RTL code, but in the post-map netlist, the entire design will respond correctly to the reset used for GSR. This is because the design will now consist of Lattice components that have been modeled to take the GSR information into account as part of their functionality. The post-map design will reset all elements sensitive to GSR when the reset signal assigned to GSR is used. Even if an element is connected to a different reset, it will still reset when the GSR signal is used if it is sensitive to GSR.

If the design contains both RTL code and Lattice component instantiations, the design may need some additions to prevent simulation problems. Verilog simulations will produce errors if there are library elements in the design that use the GSR information and instantiations of both GSR and PUR components are not present in the top-level of the design. For Verilog designs, you must instantiate the GSR component in the top level of the design hierarchy with the instance name of GSR_INST if the design contains any instantiation of library elements that are affected by the GSR settings (sequential and memory components in general). Additionally, you must also instantiate the PUR component in the top level of the design hierarchy with the instance name of PUR_INST for these same components. Below are examples of the Verilog instantiations of GSR and PUR.

GSR Verilog HDL Example:

```
GSR     GSR_INST (.GSR (<global reset sig>));
```

PUR Verilog HDL Example:

```
PUR     PUR_INST (.PUR (<powerup reset sig>));
```

If any usage case other than the Global GSR usage case is desired, the GSR and PUR components must be removed from the top-level hierarchy before synthesizing the design (**Build Database** process step) if the top level is part of the design. The one exception to this requirement is if the GSR and PUR components are connected to VCC instead of a design signal. The GSR and PUR components do not require removal if they are connected to VCC. If the top level of the design hierarchy is the testbench, then the GSR and PUR components do not need to be removed for any of the usage cases.

Mixed Language Simulation and the GSR Resource The ability to simulate the GSR resource is handled differently in Verilog and VHDL due to differences in the languages. This difference can make simulating the GSR functionality in a mixed language difficult.

Verilog simulation models access the GSR state by referring to a signal inside the GSR component that must be instantiated at the top level of the design. This can be seen in the following piece of Verilog code inside a simulation library module:

```
parameter GSR = "ENABLED";

tri1 GSR_sig = GSR_INST.GSRNET;
tri1 PUR_sig = PUR_INST.PURNET;
```

```
always @ (GSR_sig or PUR_sig ) begin
if (GSR == "ENABLED") begin
SRN = GSR_sig & PUR_sig ;
end
else if (GSR == "DISABLED")
SRN = PUR_sig;
end
```

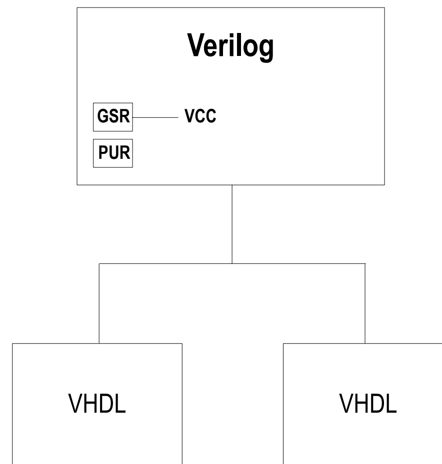
VHDL simulation models access the GSR state by referring to a signal in a package that is used by the model. Below are sections of the package and its use in a simulation model.

```
-------------Package Declaration----------------------
PACKAGE global IS
SIGNAL gsrnet: std_logic := 'H';
SIGNAL purnet: std_logic := 'H';
SIGNAL tsallnet: std_logic := 'H';
END global;

------------Entity Usage Statement------------------
USE work.global.gsrnet;
USE work.global.purnet;

-------------Entity Declaration-------------------------
GENERIC (
gsr             : String  := "ENABLED");

-------------Architecture Body------------------------
IF (gsr = "DISABLED") THEN
set_reset := purnet;
ELSE
set_reset := purnet AND gsrnet;
END IF;
```

If the desired simulation is post-synthesis, or post-map, the easiest solution is to generate a netlist in the same language as the top-level testbench. Then the simulation can be done in a single language with respect to GSR and the correct functionality can be simulated. This occurs automatically if the simulation is run from Project Navigator by selecting the testbench and then selecting **Post-Route Functional Simulation** or **Post-Route Timing Simulation**.

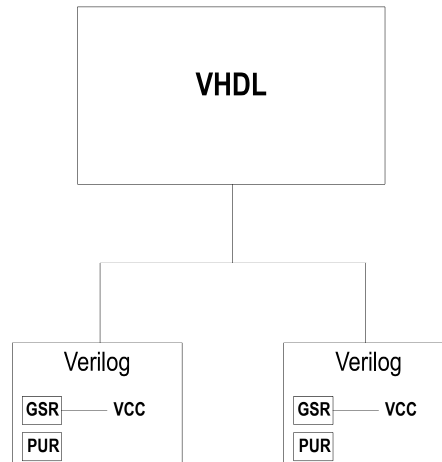
If the simulation is an RTL functional simulation, the same issues described above in the RTL functional simulation section exist along with the complication of the differences in how the languages deal with GSR. There are two different issues (initialization and functionality) and different usage cases to be considered in mixed language simulation. The different relevant combinations are discussed below.

- *Simulation Initialization, Verilog top, VHDL bottom*. When a simulator starts or initializes a design, it elaborates the design and assembles all the required information. In Lattice VHDL simulation libraries, components access GSR information via a VHDL package.



The package is provided with the simulation library so no extra information is required to initialize the simulator. In Lattice Verilog simulation libraries, the GSR information is accessed from an instantiation at the top of the Verilog part of the design. This requires a GSR and PUR component instantiation as shown in the section, RTL Functional Simulation and the GSR Resource. These instantiations are required in the Verilog hierarchy if it contains any Lattice Verilog library components that access GSR information even if GSR is not functionally used in the design. For this design structure, the Verilog GSR and PUR instantiation must be at the top of the hierarchy, either the top-level netlist or the testbench.

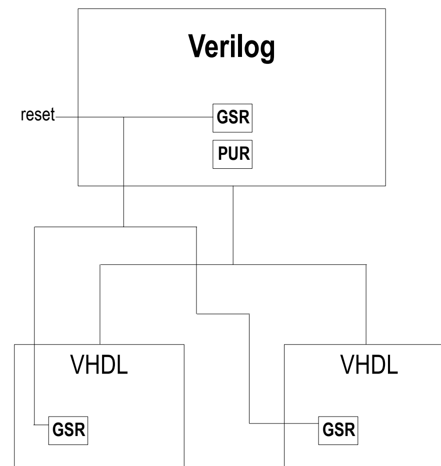
- *Simulation Initialization, VHDL top, Verilog bottom*. When a simulator starts or initializes a design, it elaborates the design and assembles all the required information. In Lattice VHDL simulation libraries, components access GSR information via a VHDL package.



The package is provided with the simulation library so no extra information is required to initialize the simulator. In Lattice Verilog simulation libraries, the GSR information is accessed from an instantiation at the top of the Verilog part of the design. This requires a GSR and PUR component instantiation as shown in the section, RTL Functional Simulation and the GSR Resource. These instantiations are required in the Verilog hierarchy by any Lattice Verilog library component that accesses GSR information even if GSR is not functionally used in the design. For this design

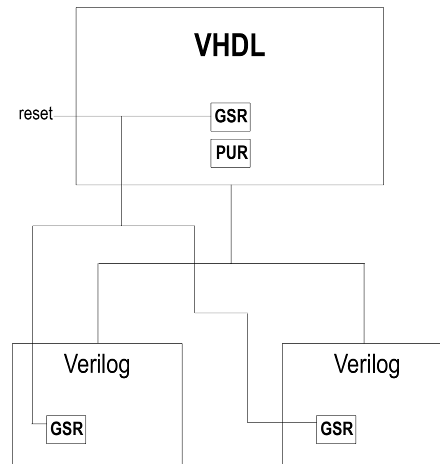
structure, the Verilog GSR and PUR instantiations must be at the top of the Verilog hierarchy, not the top of the whole hierarchy since the top is VHDL. If the design has a VHDL top and there are multiple Verilog instantiations or modules within the VHDL design, then each separate Verilog hierarchy must have its own GSR and PUR component instantiation. The GSR and PUR component instantiations in one Verilog hierarchy are not visible to other separate hierarchies.

- *Global GSR functional simulation, Verilog top, VHDL bottom*. Since the GSR information for Lattice simulation libraries is simulated in Verilog and VHDL using different mechanisms, it is required to have a GSR component instantiation in each language section.



A GSR component should be instantiated in the Verilog top hierarchy and connected to the desired reset signal. A GSR component should be instantiated in the VHDL hierarchy and connected to the same reset signal. If there are separate VHDL hierarchies within the Verilog top hierarchy, each VHDL hierarchy must have its own GSR component instantiated and connected to the reset signal. GSR components instantiated in one VHDL hierarchy do not affect other VHDL hierarchies. A PUR component is still required in the Verilog for simulation initialization purposes.

- *Global GSR functional simulation, VHDL top, Verilog bottom*. Since the GSR information for Lattice simulation libraries is simulated in Verilog and VHDL using different mechanisms, it is required to have a GSR component instantiation in each language section.



A GSR component should be instantiated in the VHDL top hierarchy and connected to the desired reset signal. A GSR component should be instantiated in the Verilog hierarchy and connected to the same reset signal. If there are separate Verilog hierarchies within the VHDL top hierarchy, each Verilog hierarchy must have its own GSR component instantiated and connected to the reset signal. GSR components instantiated in one Verilog hierarchy do not affect other Verilog hierarchies.

- *Inferred GSR, User Specified Inferred GSR, and LSR (No GSR) functional simulation*. Unlike the Global GSR usage case, the functionality of the design can be simulated correctly for the other usage cases without requiring special simulation techniques since the functionality will be simulated using the reset signals in the design. However, the issue with initializing the Verilog section of the design is the same as described in the simulation initialization sections above. A GSR and PUR component instantiation must be present in the Verilog hierarchy or in each separate Verilog hierarchy if there are multiple ones under a VHDL top hierarchy. The signal input to the GSR component can be assigned to the desired reset signal or to a logical value of ‘1’ if the GSR functionality is not used.

Block Modular Design (BMD) GSR Considerations In the Block Modular Design (BMD) implementation flow, different parts of a large design are implemented in different projects and then assembled together in a separate step. This flow is not compatible with the Inferred GSR or User Specified Inferred GSR usage cases.

Inferred GSR and User Specified Inferred GSR usage cases are used to allow the software to determine the reset automatically or from a preference. However, since Map is run in separate projects individuality there is no step in the BMD flow where the software has full visibility over the entire design to accomplish this. The BMD flow, however, is compatible with the Global GSR case as long as care is used to select the correct reset signal in each BMD project.

For the Global GSR usage case, a GSR component must be instantiated in the design for each project and connected to the correct reset signal. For the LSR (No GSR) usage case, the map property **Don’t Infer GSR** needs to be set to True so as not infer GSR for each project.

Comparison of ispLever 7.2 GSR Flow Versus Previous Software Versions Prior to ispLever 7.2 software release, the control of the GSR resource was accomplished with different mechanisms. Below is a summary of the differences between the two flows.

ispLEVER software prior to 7.2 release:

- GSR usage inferred by synthesis automatically
- GSR instantiation used as Global reset and prevents inferencing of GSR
- Option in synthesis to prevent GSR inferencing
- Several IPexpress modules included GSR enable option
- IPexpress IPs contained option to instantiate GSR component

ispLEVER 7.2 software release and later:

- No GSR inferencing by synthesis
- GSR inferencing of whole design done automatically by map
- Ability for user to specify which signal to use to infer GSR on
- GSR instantiation used as Global reset by map and prevents inferencing of GSR
- Option included in Map to prevent GSR inferencing
- No GSR options in IPexpress modules
- No GSR options in IPexpress IPs

The GSR flow in ispLEVER 7.2 uses Map as the central point in determining how to use the GSR resource. A significant advantage of this flow is that Map has visibility into the entire design to make the correct choice on how to use the GSR resource. Sometimes a design is synthesized as multiple sections. Setting the GSR resource during synthesis allows the potential of errors since each synthesized section could be using different options. Additionally, since Map has visibility of the whole design, there is no longer any requirement to have individual GSR options in modules and IPs. Finally the Map-based flow provides a new option to allow the desired signal for GSR inferencing to be specified and allows a single point to disable any usage of the GSR resource.

Using Attributes to Control GSR Resource Usage This section contains details on how to control the behavior of GSR usage on a module or component level. This information is intended for expert usage only. This information is not required for any of the usage flows discussed earlier in this topic to function correctly.

One example of the need to use this information is in the case of using Global GSR. If one section of the design is required to continue functioning during the use of reset (e.g., such as a communication port), then it is possible to use attributes to prevent the GSR resource from being implemented in a particular hierarchy or component. Likewise, it is possible to force a hierarchy or component to respond to GSR even if it was on a different reset in the Inferred GSR case. To correctly use these attributes, it is necessary to understand both the hierarchy inheritance order and how each attribute is treated in the different usage cases.

GSR Attribute Values and Syntax You must specify which portions of the design that you wish to alter the way in which they respond to the GSR reset signal. Unless specified otherwise, by default, all design elements will respond to the global reset signal if it is present. The available values are as follows.

- ENABLED – This is the default value on most library elements. This value allows the software to determine the final value and will be overridden by the parent hierarchy if the parent has a value of anything other than ENABLED.
- DISABLED – This prevents the hierarchy or element from responding to the GSR value. It cannot be changed by the parent's value.
- FORCEENABLE – This forces the hierarchy or element to respond to the GSR value. It cannot be changed by the parent's value.
- IPENABLE – This forces the hierarchy or element to respond to the GSR value when IP is being used in evaluation mode. It cannot be changed by the parent's value. This is only meant for internal Lattice IP to use. This value should never be used within a design itself.

These values are then placed as synthesis attributes in the design which synthesis will pass to the Map program. The attributes can be placed on any instance whether it is a level of hierarchy or a leaf component. Map will then interpret the attributes along with the usage case and make the necessary changes to the design. After Map is finished, all elements in the design will have GSR values of either ENABLED or DISABLED only. The syntax of the attributes is shown below for Verilog and VHDL with "VALUE" shown as the placeholder for legal values.

Synplify GSR Attributes

Verilog GSR attribute syntax

```
modname modinst (signal list) /* synthesis GSR=VALUE */;
```

VHDL GSR attribute syntax

```
attribute GSR : string;
attribute GSR of comp_instance: label is "VALUE";
component_instance: component_name port map (signal list);
```

Note

Replace the placeholder VALUE in the above syntax examples with legal values from the value list.

Precision GSR Attributes

Verilog GSR attribute syntax

```
modname modinst (signal list) //exemplar attribute modinst GSR
VALUE;
```

VHDL GSR attribute syntax

```
attribute GSR : string;
```

```
attribute GSR of comp_instance: label is "VALUE";
component_instance: component_name port map (signal list);
```

Note

Replace the placeholder VALUE in the above syntax examples with legal values from the value list.

Determining Actual Pre-Map GSR Attribute Value Following are the precedence rules to use to determine the “actual” GSR value an instance has in the pre-map netlist.

The first column in the table below represents the possible explicit values an instance can have. An explicit value is one which has been put directly on the instance. The first row represents the possible implicit values a library element can have via GSR attribute set at a parent module that includes the instance. The “parent” term is used loosely here as more precisely, the GSR value could be sitting on one or more levels of hierarchy above the parent.

Table 1: Determining “Actual” GSR value

Explicit child GSR value in pre-map netlist	Parent module GSR value (explicit)				
	<none>	DISABLED	ENABLED	FORCEENABLE	IPENABLE
<not set> (component default value ENABLED)	ENABLED	DISABLED	ENABLED	FORCEENABLE	IPENABLE
<not set> (component default value DISABLED)	DISABLED	DISABLED	DISABLED	DISABLED	DISABLED
<not set> (hierarchy level not leaf component)	ENABLED	DISABLED	ENABLED	FORCEENABLE	IPENABLE
DISABLED	DISABLED	DISABLED	DISABLED	DISABLED	DISABLED
ENABLED	ENABLED	DISABLED	ENABLED	FORCEENABLE	IPENABLE
FORCEENABLE	FORCEENABLE	FORCEENABLE	FORCEENABLE	FORCEENABLE	FORCEENABLE
IPENABLE	IPENABLE	IPENABLE	IPENABLE	IPENABLE	IPENABLE

Determining Final Post-Map GSR Attribute Value Following are the rules that the map software uses to determine the GSR attribute value of the physical component that a pre-map library element gets mapped into in the post-map netlist.

The first column represents the possible “actual” values that a pre-map library element can have. The top row represents the usage case map is operating under.

The final post-map GSR value a component has will depend on the following conditions.

- The usage case (Global GSR or either Inferred or User Specifed Inferred)
- If the component is on the inferred reset domain or not
- If any Lattice IP is being used in evaluation mode

If the "actual" value of a pre-map library element is,

- DISABLED - It will be mapped to a component which has GSR=DISABLED regardless of the usage case.
- ENABLED – This value allows the software to automatically determine the necessary value, so final value depends on the usage case
  - Global GSR - Final value is ENABLED since this should respond to the global reset
  - Inferred GSR or User Specified Inferred GSR & element is on the inferred reset - Final value is ENABLED since this will cause the element to respond to the reset being put on to the GSR resource
  - Inferred GSR or User Specified Inferred GSR & element is NOT on the inferred - Final value is DISABLED since this component is not on the inferred reset and therefore should not respond to GSR resource
- FORCEENABLE – This will always be ENABLED regardless of the usage case. This is how a component can be reset by GSR resource in the Inferred GSR usage case even if that component is not on the inferred reset.
- IPENABLE – This will always be ENABLED regardless of the usage case except when an IP license is present and the IP is not on the inferred reset. In this case only, GSR will be set to DISABLED.

Note: A consequence of the IPENABLE functionality is that the IP reset will function differently when it is not on the inferred GSR reset between licensed and evaluation mode. This is an unavoidable consequence of how the evaluation mode functions. It is important to note the licensed mode is functionality correct and will not cause any functional issues in the final design. This issue is not present in the Global GSR usage case.

Table 2: Determining Post-Map GSR Value

Actual Pre-Map Value	Global GSR Usage Case	Inferred GSR and User-Specified Inferred GSR Usage Cases		
		On Inferred Reset	No Inferred Reset	No Inferred Reset plus IP Evaluation Mode
DISABLED	DISABLED	DISABLED	DISABLED	DISABLED
ENABLED	ENABLED	ENABLED	DISABLED	DISABLED
FORCEENABLE	ENABLED	ENABLED	ENABLED	ENABLED
IPENABLE	ENABLED	ENABLED	DISABLED	ENABLED

Issues Using Global GSR and GSR=DISABLED Attribute Due to Synthesis Optimizations In the GSR flow with ispLever 7.2 and later, synthesis does not perform any GSR inferencing done in earlier releases. Additionally, synthesis does not understand the attributes used for GSR control, but will pass them in the design to the map software which will perform the correct changes.

However, synthesis normally will perform optimizations it is aware of. Synthesis will see a GSR component that has been instantiated in a design for the Global GSR usage case and may automatically optimize out the signal connections for the reset signal to components if an active low reset signal is defined. However, since synthesis is unaware of the meaning of the GSR attributes, any components in a Global GSR usage case that have GSR=DISABLED on them may be left without any ability to be reset (reset signal disconnected and GSR=DISABLED). Note that this issue will not occur on the Inferred GSR or User Specified Inferred GSR usage cases.

In order to avoid this issue the User Specified Inferred GSR usage case can be used. The desired reset signal should be specified in the preferences as defined for this usage case. Additionally at the top of the design, the attribute GSR=FORCEENABLED should be specified. This will force all components, even in a multiple reset design, to be reset by GSR similar to the Global GSR usage case. Design elements that need to be prevented from resetting can use the GSR=DISABLED attribute to block reset for those components or hierarchies since they will not be affected by the GSR=FORCEENABLED as shown in Table 1.

Issues with Synthesis Hierarchy Flattening or Optimization Synthesis tools can potentially flatten a hierarchy or optimize hierarchies to improve results. When this occurs, attributes assigned to a hierarchy level will be lost. This is true for the GSR attributes since synthesis is not aware of any special meaning for these attributes. If this occurs, you can prevent the flattening from occurring by adding an additional attribute to the hierarchy.

Below are code examples that illustrate how to preserve hierarchical attributes for the Synplify and Precision synthesis tools:

Synplify Hierarchy Attributes

Verilog hierarchy attribute syntax

```
modname modinst (signal list) /* synthesis syn_hier=hard */;
```

VHDL hierarchy attribute syntax

```
attribute syn_hier : string;
attribute syn_hier of comp_instance: label is "hard";
component_instance: component_name port map (signal list);
```

Precision Hierarchy Attributes

Verilog hierarchy attribute syntax

```
modname modinst (signal list) //pragma attribute modinst
hierarchy preserve;
```

VHDL hierarchy attribute syntax

```
attribute syn_hier : string;
attribute syn_hier of comp_instance: label is "preserve";
component_instance: component_name port map (signal list);
```

How to Use the Power Up Set/Reset (PUR) Global Signal

This topic provides guidelines and specific instructions for interfacing to the Power Up Set/Reset (PUR) signals of a Lattice FPGA device simulation model for use with all Lattice FPGAs. Details regarding the architecture of global signals are described in the data sheet for each device family. For more information on the PUR library element, see the FPGA Libraries Help system. For more information on using Lattice library elements in HDL, see Lattice Synthesis Header Libraries topic in the ispLEVER help system.

About the GSR and PUR Signals Both Global Set/Reset (GSR) and PUR are global RESET signals that will initialize the chip to some known reset state. In a given device, GSR has a dedicated input pad; however, PUR is not driven by any external pad. PUR is only activated upon device startup, just after configuration is complete, which is basically power-up of the device. Unlike PUR, the GSR signal can be called at any time during operation to reset the devices initial values.

Major items discussed in this topic are as follows:

- Using the PUR Library Element
- PUR Verilog HDL Example
- PUR VHDL Example

Using the PUR Library Element The Power Up Set/Reset (PUR) signal is activated during device configuration and is driven by internal circuitry. PUR is often used in the context of a system-level simulation where the power up control of the PC board is verified. You can model the behavior of this signal by adding the PUR library element to your test fixture. When the PUR input port is driven all register elements of the design will respond as if a global set/reset was asserted. PUR is in an active LOW state by default.

You commonly instantiate the PUR element in the test fixture with the instance name PUR_INST.

Note: Verilog simulations will produce errors if there are library elements in your design that require the instantiation of GSR, PUR, and TSALL elements and they are not present. For Verilog designs, you must instantiate the PUR and GSR elements in their top-level netlists with instance names PUR_INST and GSR_INST respectively if the design contains any instantiation of library elements which are affected by GSR.

The examples below shows proper syntax for instantiating a PUR element in Verilog HDL or VHDL.

PUR Verilog HDL Examples

```
PUR     PUR_INST (.PUR (<powerup reset sig>));
```

The PUR element instantiation below also shows a parameter called RST_PULSE with a value of 10 ns that is used to set the required reset pulse length. You can pass a required numerical value such as 10 ns or 100 ns. If you do not specify this parameter, by default it will be set at 1 ns.

```
PUR     PUR_INST (.PUR (<powerup reset sig>));
defparam PUR_INST.RST_PULSE = 10;
```

PUR VHDL Examples

```
PUR_INST:   PUR port map (PUR=><powerup reset sig>);
```

As shown in the Verilog PUR example, this VHDL PUR element instantiation below also shows a parameter called RST_PULSE with a value of 10 ns that is used to set the required reset pulse length.

```
PUR_INST : PUR
generic map (RST_PULSE => 10)
port map (PUR => <signal>);
```

Note: PUR library elements should be used as part of a Verilog test fixture or VHDL test bench to model power up set/reset only. PUR library elements can not interface to a signal of the FPGA design itself.

References For more information refer to the following documentation.

- PUR library element description in the FPGA Libraries Help system.

How to Use the Tristate Interface (TSALL) Global Signal

This topic provides guidelines and specific instructions for interfacing to the Global Tristate Interface (TSALL) signal of a Lattice FPGA device for use with the Lattice MachXO and LatticeSC/M FPGA devices. Details regarding the architecture of global signals are described in the data sheet for each device family. For more information on the TSALL library element, see the FPGA Libraries Help system. For more information on using Lattice library elements in HDL, see Lattice Synthesis Header Libraries topic in the ispLEVER help system.

Major items discussed in this topic are as follows:

- Using the TSALL Library Element
- TSALL Verilog HDL Example
- TSALL VHDL Example

Using the TSALL Library Element By default the tristate enable of PIO blocks will be assigned to local output enable signals of the design. To explicitly connect a design signal to the global tristate network of the device then it is necessary to instantiate the TSALL library element in your design. TSALL is considered active LOW to enable output.

Important: You must instantiate the TSALL with the instance name TSALL_INST. The port name for the MachXO TSALL library element is TSALL. The port name for the LatticeSC/M TSALL library element is TSALLN.

Note: Verilog simulations will produce errors if there are library elements in your design that require the instantiation of GSR, PUR, or TSALL elements and they are not present.

The examples below show proper syntax for instantiating the MachXO and LatticeSC/M TSALL element in Verilog HDL or VHDL, respectively.

TSALL Verilog HDL Example (MachXO)

```
TSALL   TSALL_INST (.TSALL ());
```

TSALL VHDL Example (MachXO)

```
    component TSALL
        port( TSALL: in STD_ULOGIC );
    end component;
    -- Attributes for Synplify
    attribute syn_black_box: boolean ;
    attribute syn_black_box of TSALL: component is true;
    attribute syn_noprune: boolean ;
    attribute syn_noprune of TSALL: component is true;
    -- Attributes for Precision RTL
    attribute BLACK_BOX : boolean;
    attribute BLACK_BOX of TSALL: component is true;
    attribute DONT_TOUCH : boolean;
    attribute DONT_TOUCH of TSALL_INST: label is true;
begin

    TSALL_INST: TSALL port map (TSALL=><global tristate sig>);
```

TSALL Verilog HDL Example (LatticeSC/M)

```
TSALL   TSALL_INST (.TSALLN ());
```

TSALL VHDL Example (LatticeSC/M)

```
    component TSALL
        port( TSALLN: in STD_ULOGIC );
    end component;
    -- Attributes for Synplify
    attribute syn_black_box: boolean ;
    attribute syn_black_box of TSALL: component is true;
    attribute syn_noprune: boolean ;
    attribute syn_noprune of TSALL: component is true;
    -- Attributes for Precision RTL
    attribute BLACK_BOX : boolean;
```

```
    attribute BLACK_BOX of TSALL: component is true;
    attribute DONT_TOUCH : boolean;
    attribute DONT_TOUCH of TSALL_INST: label is true;
begin
    TSALL_INST: TSALL port map (TSALLN=><global tristate sig>);
```

Note

Lattice Applications recommends using "don't touch" synthesis directives, like syn_noprune and DONT_TOUCH, with library elements that you instantiate in VHDL designs. For more information on Lattice library elements, see the FPGA Libraries Help system.

References For more information refer to the following documentation.

- TSALL library element description in the FPGA Libraries Help system.