# Global Set/Reset Usage for Nexus Platform

# Application Note

FPGA-AN-02062-1.0

September 2023

## Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS, with all faults and associated risk the responsibility entirely of the Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice's product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

# Contents

Contents....3
Figures....4
Acronyms in This Document....5
1. Introduction....6
1.1. GSR Pros and Cons....6
1.1.1. Pros....6
1.1.2. Cons....6
1.2. Async GSR vs Sync GSR....7
2. GSR Usage Cases....8
2.1. Inferred GSR through Strategy Settings....8
2.1.1. Synthesis Inferred GSR....9
2.1.2. Map Inferred....11
2.2. User-Specified Inferred GSR....12
2.3. Instantiated GSR....15
2.4. LSR (No GSR)....16
3. Controlling GSR in Module/Component level....17
3.1. GSR Attribute Syntax and Values....17
4. GSR Usage Guidelines....20
4.1. General Considerations....20
4.2. GSR Usage Guidelines for Simulation....20
4.2.1. GSR Resource and RTL Functional Simulation....20
4.2.2. Mixed Language Simulation and the GSR Resource....27
4.2.3. Library Compilation on 3rd party Simulator....28
4.3. GSR Usage with Reveal....29
4.4. GSR hardware in FPGA....30
Appendix A. Workarounds for Reveal with GSR Issues....31
References....33
Technical Support Assistance....34
Revision History....35

# Figures

Figure 2.1. Synplify Pro Strategy Setting for Force GSR ..... 9
Figure 2.2. Lattice Synthesis Engine Strategy Setting for Force GSR ..... 9
Figure 2.3. Map Design Strategy Setting for Infer GSR ..... 9
Figure 2.4. Synplify Pro Report for GSR utilization ..... 10
Figure 2.5. Synplify Pro HDL Analyst showing GSR instance ..... 10
Figure 2.6. Radiant Netlist Analyzer showing GSR instance ..... 11
Figure 2.7. Map report showing GSR resource ..... 11
Figure 2.8. Map report showing GSR component ..... 12
Figure 2.9. PAR report showing GSR Utilization ..... 12
Figure 2.10. DCE's GSR net assignment ..... 12
Figure 2.11. Constraints for GSR net assignment ..... 13
Figure 2.12. Map and PAR report for GSR resource ..... 13
Figure 2.13. Physical Designer information for GSR net ..... 14
Figure 2.14. Instantiated GSR Netlist ..... 16
Figure 3.1. Example Verilog design that uses GSR attribute ..... 17
Figure 3.2. Example VHDL design that uses GSR attribute ..... 18
Figure 3.3. GSR enabled on PCLKDIV checked in *.vm ..... 18
Figure 3.4. GSR disabled on PCLKDIV checked in *.vm ..... 19
Figure 4.1. Example code for Verilog test bench ..... 22
Figure 4.2. Example code for VHDL test bench ..... 23
Figure 4.3. Simulation Model Example (Lattice component where GSR is enabled) ..... 24
Figure 4.4. Error encountered without GSR instantiation on design with Lattice components/primitives ..... 24
Figure 4.5. Example test bench with GSR primitive instantiation ..... 25
Figure 4.6. Showing the manual compilation of the design ..... 25
Figure 4.7. Showing how to start simulation and selecting test bench ..... 26
Figure 4.8. Showing the required libraries to compile prior simulation ..... 26
Figure 4.9. Adding signals for simulation ..... 26
Figure 4.10. Wrong GSR instance name in test bench ..... 27
Figure 4.11. modeltech.ini (modeltech/lib is where users can find the logical libraries) ..... 27
Figure 4.12. Radiant' s Verilog Source Libraries ..... 28
Figure 4.13. Radiant does not have a VHDL source library ..... 28
Figure 4.14. Synthesis attribute in Reveal Core for disabling GSR ..... 29
Figure 4.15. GSR_N functional block diagram ..... 30

# Acronyms in This Document

A list of acronyms used in this document.

| Acronym | Definition |
|---|---|
| DCE | Device Constraint Editor |
| DSP | Digital Signal Processor |
| EBR | Embedded Block RAM |
| FF | Flip-Flop |
| GSR | General Set/Reset |
| LSR | Local Set/Reset |
| LSE | Lattice Synthesis Engine |
| pdc | Post-Synthesis Constraint file |
| PIC | Programmable I/O Controller |
| PLC | Programmable Logic Controller |
| PMU | Power Management Unit |
| POR | Power-On-Reset |
| RTL | Register-Transfer Level |
| STA | Static Timing Analysis |

# 1. Introduction

The purpose of this document is to provide information on how the user Global Set/Reset (GSR) is utilized in Nexus Platform Devices. Guidelines and Instructions are provided on when to use or not to use GSR resource. This user guide also includes usage cases of GSR in hardware, simulation, and reveal.

Nexus Platform devices contain Global Set/Reset resources. GSR is a hardware resource provides dedicated routing on the registers found in the device, this allows users to initialize all the registered elements of the design that have the resource enabled. The resource is a convenient mechanism to allow the design components to be initialized without using any additional routing resources in the device.

There are two primary ways to take advantage of the GSR hardware resource in the design:

- Use the GSR to initialize all components on the FPGA.
- Use the GSR to eliminate any additional routing resources needed for one reset in a multiple reset design.

Typically, if a design has a single set or reset to Initialize all components on the FPGA, users can use GSR for this reset signal. For a multi reset design, the reset with the highest fanout is usually selected to drive the GSR resource.

The implementation of the GSR resource is tightly controlled by the Radiant tool as directed implicitly by tool defaults or explicitly by the user. Each Nexus Platform device contains 1 GSR. Radiant implements GSR resource to be Asynchronous and is utilized automatically by the tool. GSR resource can be controlled in the following ways:

- RTL-level: GSR primitive can be manually instantiated in user design.
- Synthesis Level: GSR utilization can be changed through the “Force GSR” setting found in Strategy options.
- Map Level: GSR can be used through ldc_set_attribute defined in *.pdc , or controlling through Radiant’ s DCE.

**Notes**:

- In the Inferred GSR and User-Specified Inferred GSR usage cases, the software only connects elements with an asynchronous set or reset to GSR. Elements requiring a synchronous set or reset uses only local routing.
- The primitive information of GSR can be found on Radiant help (Reference Guides > FPGA Libraries Reference Guide > Alphanumeric Primitives List > GSR).

## 1.1. GSR Pros and Cons

GSR may not be suitable for all use cases, and it has its own advantages and disadvantages. While it may not be suitable for larger FPGAs, it remains popularly used in small scale FPGA designs.

### 1.1.1. Pros

- The GSR can be used to set or reset all components on FPGA if there is only one set or reset signal for the entire design.
- Reduces routing congestions by using dedicated global resources.
- Easily inferable from user RTL.
- Lower skew than general purpose routing so it provides a better path for resetting registers distributed throughout the device.

### 1.1.2. Cons

- May cause initialization issues on high-speed designs due to greater latency GSR distribution delays and associated inconsistent GSR release timing with respect to clocks.
- Only one GSR available for the designs with multiple resets.
- GSR signals delays are not covered by Static Timing Analysis.

## 1.2. Async GSR vs Sync GSR

User GSR is designed for user-controlled set or reset that can be set as asynchronous or synchronous. The GSR primitive must be instantiated to make use of and specify a synchronous set or reset. The SYNCMODE parameter in the instantiated GSR primitive can take the values “SYNC” & “ASYNC” to specify a synchronous or an asynchronous GSR instantiation. Fundamentally, asynchronous user GSR activates as soon it is asserted while for synchronous user GSR only activates on a positive or negative clock edge.

The synchronous GSR has a requirement of de-assertion within 1 clock cycle to properly perform synchronous reset of PLC registers state. After the synchronous reset, the user logic can start on the next clock cycle.

# 2. GSR Usage Cases

The Global Set/Reset resource can be controlled in any stage of the implementation flow. GSR can be used through utilization in synthesis, map, RTL, and constraint file. Synthesis and Map infers the GSR resource according to the defined rules. GSR in RTL can be used by utilizing the GSR primitive in the design and manually defining the desired input net/port. GSR can also be defined using constraints in the *.pdc.

In Radiant, there are generally four usage cases with respect to initialization of set/resets; The four usage cases are as follows:

- Inferred GSR
- User-Specified GSR
- Instantiated GSR
- LSR (or No GSR)

Each usage is explained and implemented in the succeeding sections.

## 2.1. Inferred GSR through Strategy Settings

Inferred GSR is an option where software automatically determines which set or reset signal in the design utilizes the GSR resource/routing. This usage case a common choice for many applications. The software tool determines the set or reset signal with the most loads and selects it to drive the GSR resource. This generally frees the most fabric routing resources and leads to reduced congestion elsewhere in the design. The Inferred GSR usage case can also be used whether the design has a single or multiple resets.

When user design is synthesized and/or mapped, the tool attempts to infer the GSR resource to a set or reset net that is asynchronous and has the highest fanout.

- The tool enables Synthesis Inference of GSR for LSE. Thus, in the case of LSE, the synthesis tool automatically infers the GSR during the synthesis process by default.
- The tool enables Auto for Synplify Pro. Consequently, for Synplify Pro, the software tool determines whether the user's design requires GSR resource inference, providing flexibility in making such decisions.

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 implements a design using GSR for the set or reset signal with the highest fan-out.

**Notes**:

- Signal utilizing the GSR can change for Inferred Case if Design modifications are implemented.
- LSE checks encrypted modules for GSR instantiations. If LSE finds a GSR instance or attribute in a module, LSE does not infer GSR in Synthesis (Force GSR = False in Synthesis Design Strategy Settings) and instead passes the GSR information to Map to handle.

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

- If users need to use a specific reset signal for GSR.
- If users need to completely disable GSR.

There are two ways to infer GSR in Radiant:

- Synthesis Inferred GSR - In this option, the synthesis tool automatically selects the heavily loaded reset net, thus the user cannot choose which reset net should be used.
- MAP inferred GSR - For this option, the users can choose which reset net to be used. It can be assigned using the LDC constraint (i.e., ldc_set_attribute {GSR_NET=TRUE} [get_nets rstn_c]) or through the Device Constraint Editor > Global.

To infer the GSR resource, The GSR implementation in the Synthesis and Map Strategy settings are used. In Synthesis (LSE or Synplify Pro), “Force GSR” Parameter can be set into three different options:

- AUTO, SW tools infers the GSR resource whenever it is feasible.
- TRUE (Synplify Pro) / YES (LSE), which always infers the GSR in synthesis.
- FALSE (Synplify Pro) / NO (LSE), which completely disables inferring of GSR. In Map, setting of inferred GSR is implement either TRUE or FALSE during mapping process.

### 2.1.1. Synthesis Inferred GSR

This GSR resource is inferred during design synthesis. The following are the required Settings and Conditions:

1. Under Synthesis Design Strategy settings: 'Force GSR' = "True" (Synplify Pro) / "Yes" (LSE). Inferring GSR during Synthesis Design is enabled.

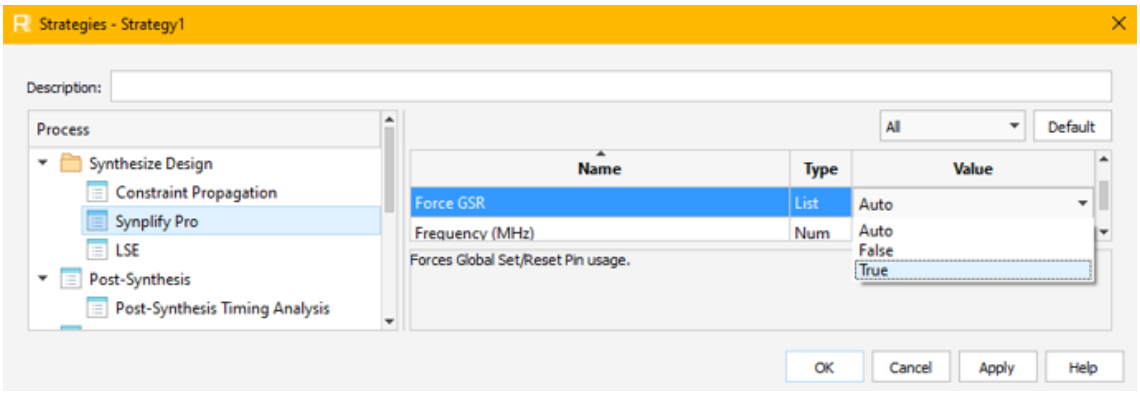


**Figure 2.1. Synplify Pro Strategy Setting for Force GSR**

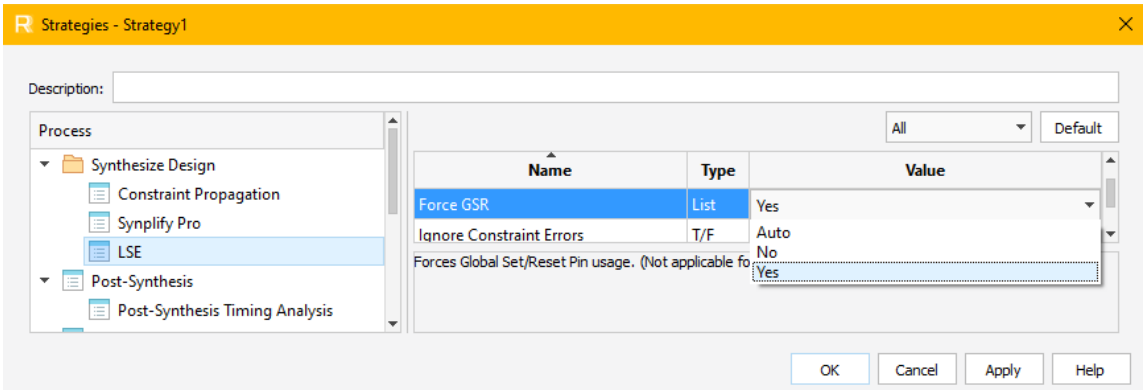


**Figure 2.2. Lattice Synthesis Engine Strategy Setting for Force GSR**

2. Under Map Design Strategy settings: Infer GSR = "F" (UNTICKED). Inferring GSR during Map Design is disabled.

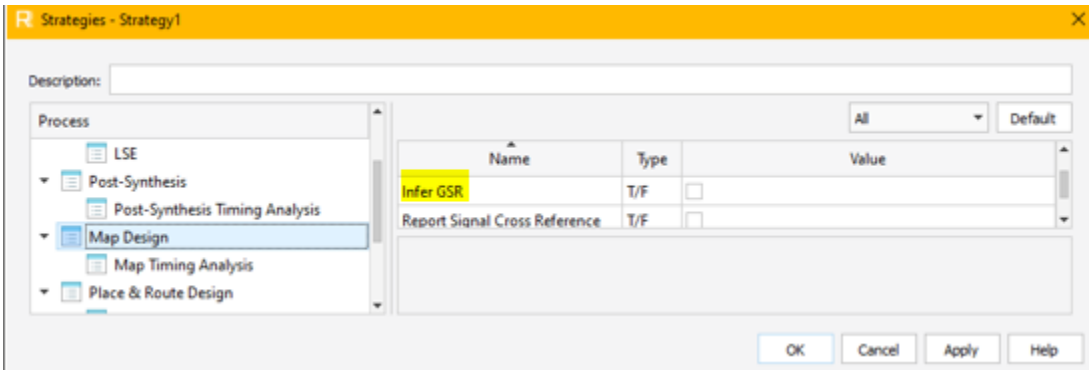


**Figure 2.3. Map Design Strategy Setting for Infer GSR**

3. Instantiation of GSR is not present in user RTL.
4. Constraint file does not contain any user selected signal for GSR resource.

5. Check Synplify Pro report or HDL Analyst or Radiant Netlist Analyzer to verify GSR utilization.

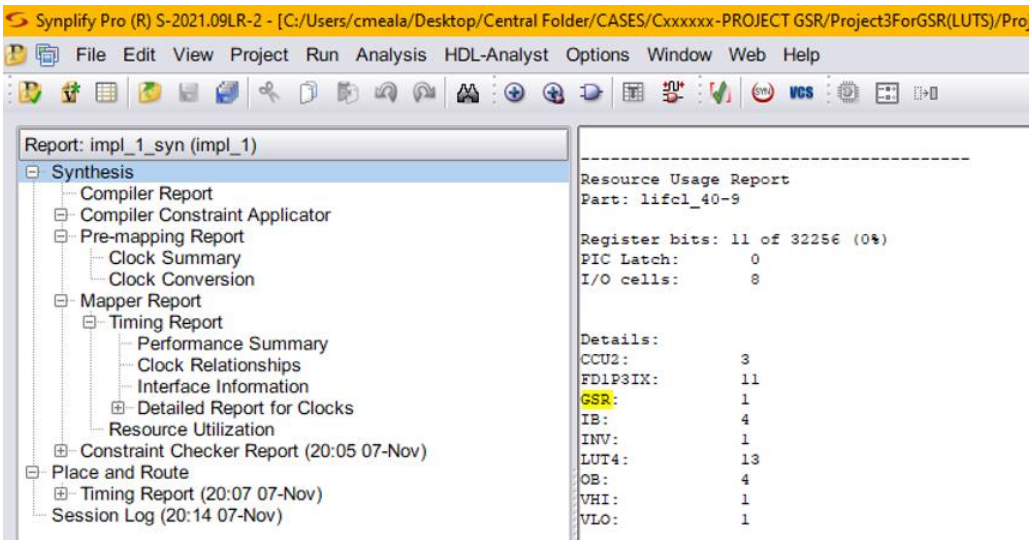


**Figure 2.4. Synplify Pro Report for GSR utilization**

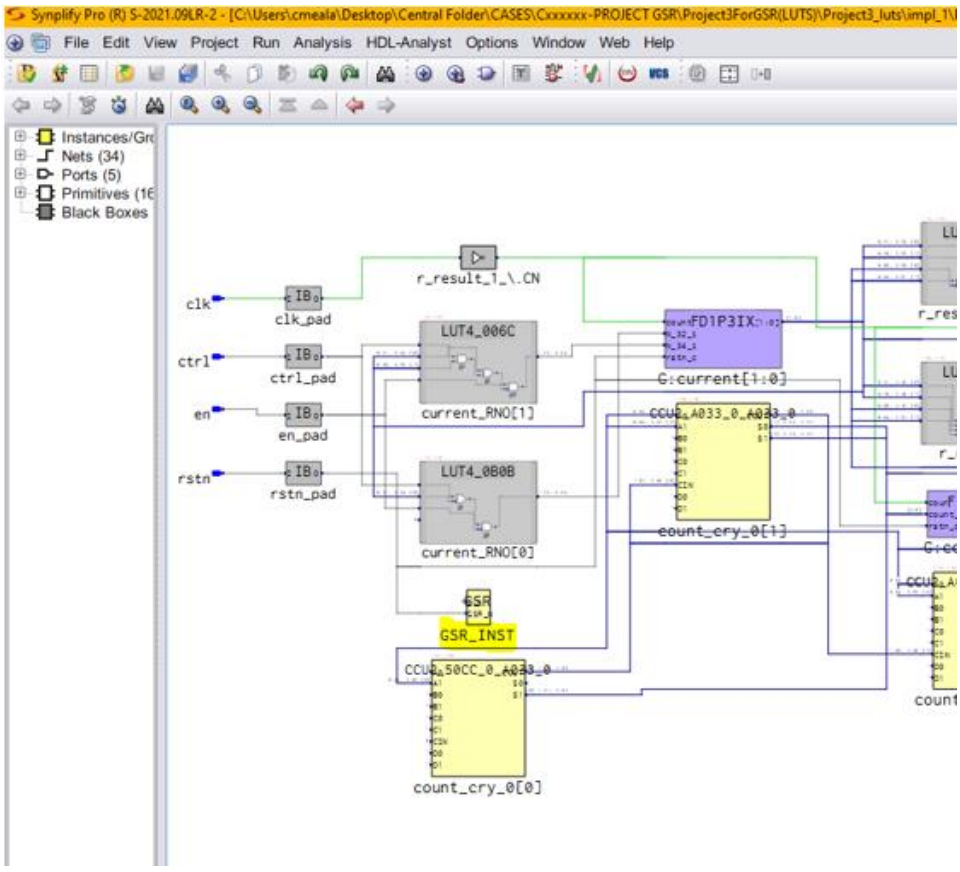


**Figure 2.5. Synplify Pro HDL Analyst showing GSR instance**

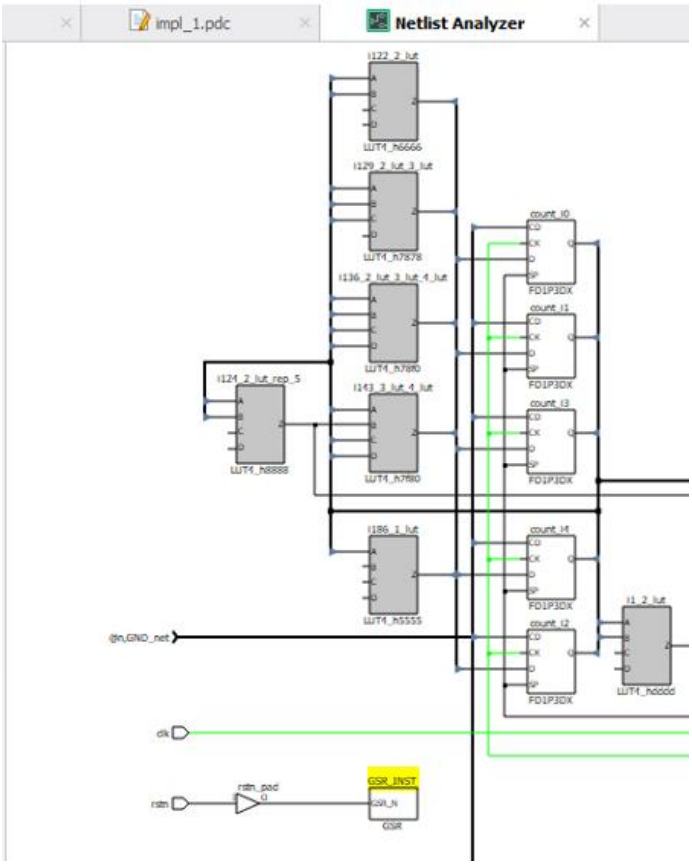


**Figure 2.6. Radiant Netlist Analyzer showing GSR instance**

### 2.1.2. Map Inferred.

This GSR resource is inferred during logic Mapping. Required Settings and Conditions:

1. Under Synthesis Design Strategy settings: Force GSR = “FALSE” (Synplify Pro) / ”NO” (LSE).
2. Inferring GSR during Synthesis Design is disabled.
3. Under Map Design Strategy settings: Infer GSR = “T” (TICKED).
4. Inferring GSR during Map Design is enabled.
5. Instantiation of GSR is not present in user RTL.
6. Constraint file does not contain any User selected Signal for GSR resource.
7. With the above conditions, to verify that GSR is successfully inferred, check Map and Place and Route Reports.

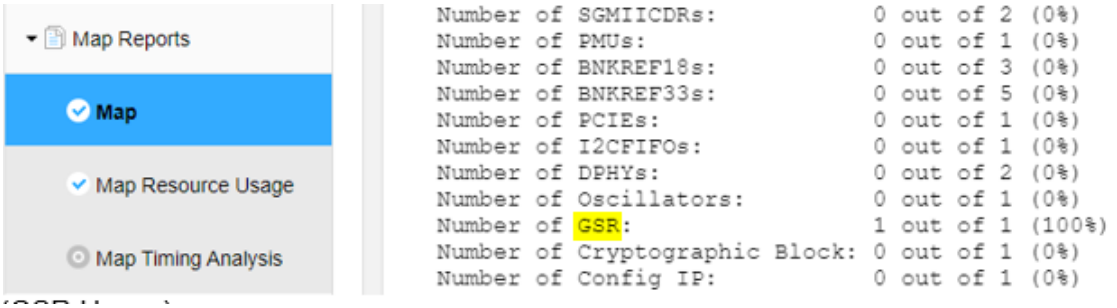


**Figure 2.7. Map report showing GSR resource**

```
GSR Usage
---------

GSR Component:
   The Global Set Reset (GSR) resource has been used to implement a global reset
        of the design. The reset signal used for GSR control is 'rstn_c'.
```

**Figure 2.8. Map report showing GSR component**

```
Device utilization summary:

   GSR                        1/1           100% used
```

**Figure 2.9. PAR report showing GSR Utilization**

## 2.2. User-Specified Inferred GSR

This is the same as the Map Inferred GSR usage except that the reset signal that is specified in the constraint (*.pdc) file determines which signal uses the GSR resource regardless of the fan-out of the signal. When there is no GSR component instantiated in the design and a constraint exist for GSR_NET, then the software implements 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 fanout of the net. User Specified inferred GSR is the preferred way of controlling GSR as it prevents changes of GSR implementation whenever Design is changed as the resource is restricted to the declared signal.

The required setting to implement the method is stated below:

- Under Synthesis Design Strategy settings: Force GSR = “FALSE” (Synplify Pro) / ”NO” (LSE)
- Under Map Design Strategy settings: Infer GSR = “T” (TICKED)
- Instantiation of GSR is not present in user RTL.
- User-Selected signal for GSR in Constraint File

**There are two ways to specify the GSR signal:**

- Method 1 - Using Device Constraint Editor GSR Net settings. Tool > Device Constraint Editor > Global > Global Set Net.

Note: Implementation through Device Constraint Editor Automatically add in the constraint file (*.pdc) once users save the changes.

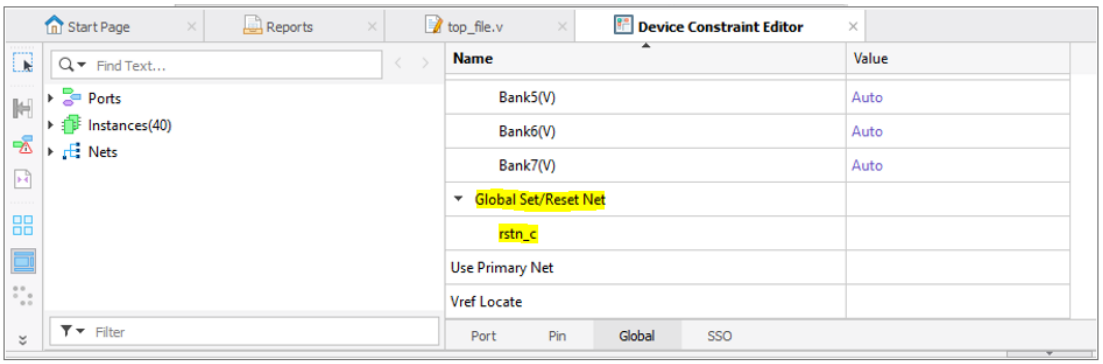


**Figure 2.10. DCE’s GSR net assignment**

- Method 2 – Manually defining in Constraint File (*.pdc).

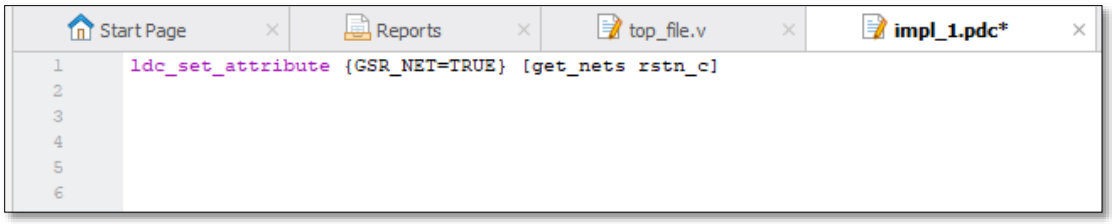


**Figure 2.11. Constraints for GSR net assignment**

**To verify that the GSR net is successfully assigned:**

1. Check the Map and Place & Route Report.

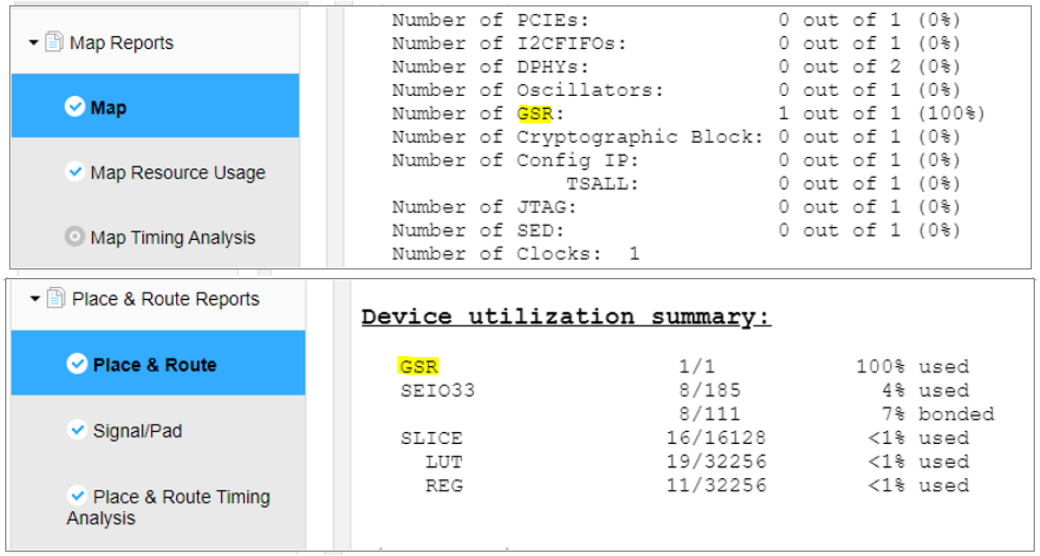


**Figure 2.12. Map and PAR report for GSR resource**

2. Check the Physical Designer that GSR inferred on the defined net.

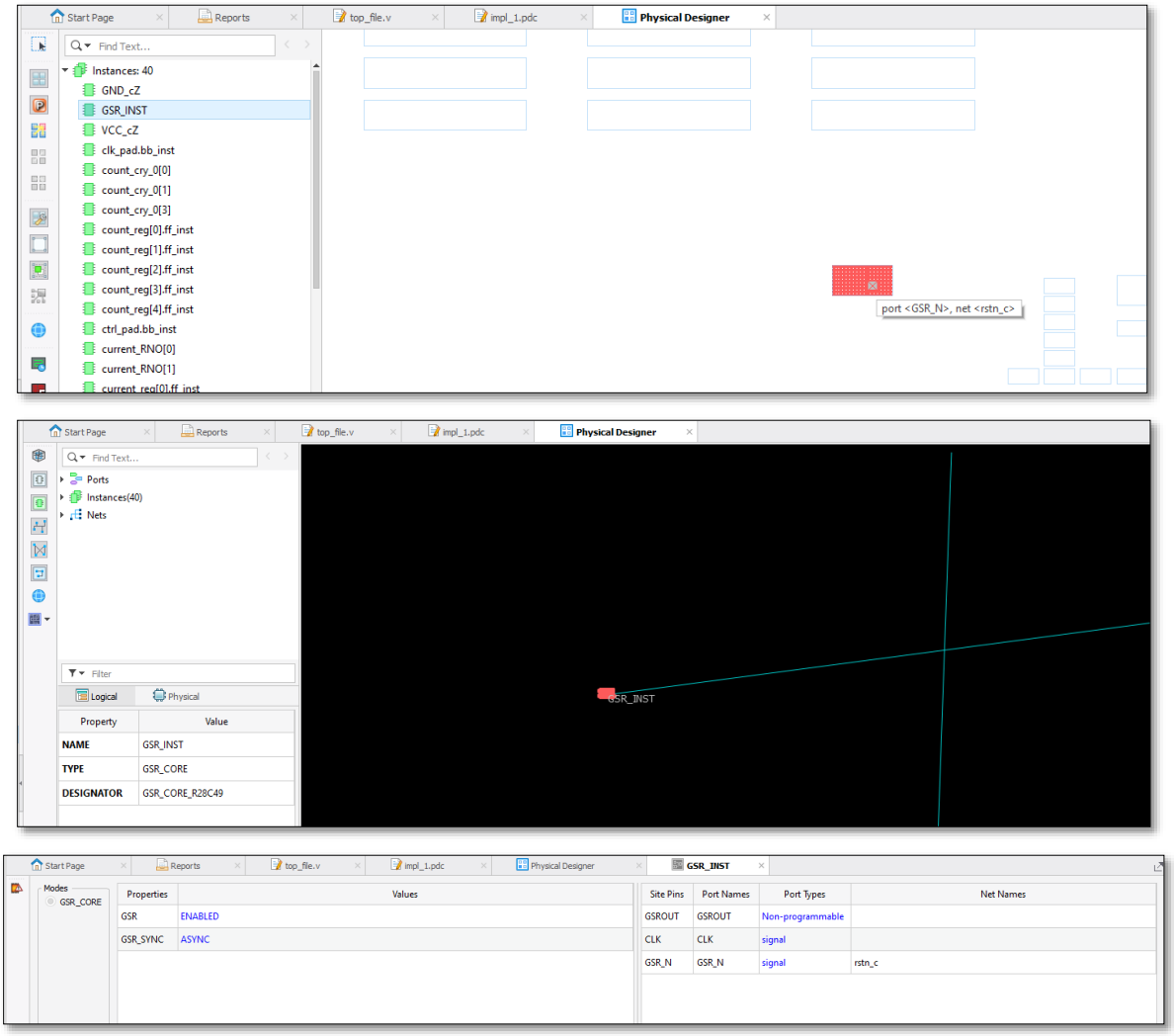


**Figure 2.13. Physical Designer information for GSR net**

## 2.3. Instantiated GSR

The instantiated GSR is GSR implementation through instantiation of the GSR component in user design and connecting it to the target reset signal. The GSR component must be instantiated into the FPGA design itself, not into the testbench for the design.

**GSR VHDL Syntax:**

```
component GSR is
  port (
      GSRN : in std_logic;
      CLK : in std_logic);
end component;
GSR_INST: GSR port map (GSR_N=> <rst_signal>, CLK => <clk_signal>);
```

**GSR Verilog HDL Syntax:**

```
GSR GSR_INST (.GSR_N (<rst_signal>), .CLK(<clk_signal>));
```

When the GSR component is instantiated in user design, it is considered Global and connected to all elements. This usage is a good fit for a design with a single reset.

**Note**: It can also be used with multiple resets in the design, but it can produce unexpected functionality in this case hence it is not recommended.

For example, if the GSR resource is used for the reset with the largest fan-out, elements on a second reset signal are still reset by the first reset signal. One possible solution is to set the GSR (Global Set/Reset) disable attribute on the logic that should not be affected by the GSR signal (Refer to Section 3. Controlling GSR in Module/Component level to know how use the attribute). By doing this, the specific logic will not be sensitive to the GSR signal and will not be affected by it. This approach allows for more control and flexibility in handling the GSR signal within the system.

Additionally, any registers with a synchronous reset are attached to their local reset as well as the GSR reset (which asserts asynchronously), in result of Sync (LSR) vs Async (GSR), active-high (LSR) vs active-low (GSR) conflicts.

Note also that the GSR resource is active low. In the case of a register using an active high reset, that register remains connected to the signal but has the GSR enabled. This causes 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 is able to accommodate a mix of synchronous, asynchronous, active high, or active low. In the Instantiated GSR usage case, the user must exercise care to be consistent with the GSR assertion levels and types. The software assumes the user knows what the design intent is and makes minimal design changes to allow the user intent to be implemented.

**To explicitly instantiate the GSR primitive:**

1. Under Synthesis Design Strategy settings: Force GSR = “FALSE”(Synplify Pro) / ”NO”(LSE)
2. Under Map Design Strategy settings: Infer GSR = “T” (TICKED)
3. Instantiation of GSR is present in user RTL.
4. Constraint file does not contain any User selected Signal for GSR resource (If a GSR_NET constraint is specified and a GSR component is already instantiated, the constraint is ignored).

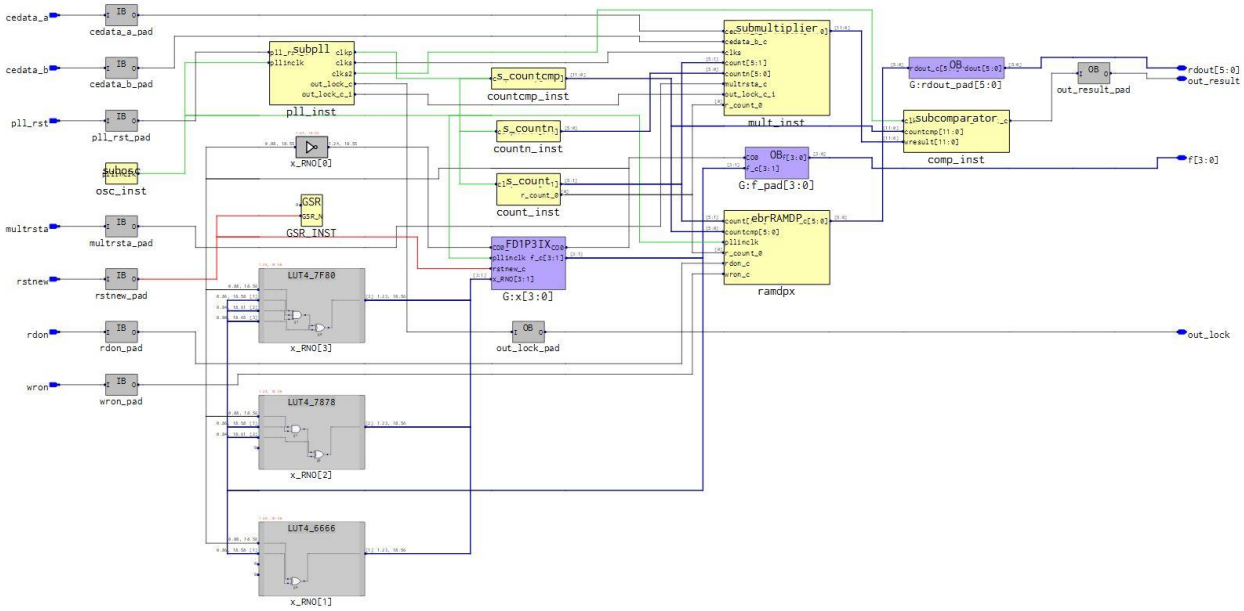


**Figure 2.14. Instantiated GSR Netlist**

## 2.4. LSR (No GSR)

LSR (local set/reset) specifies that no GSR is to be used, that is, that all resets use local routing resources instead of using the GSR resource. If the user needs to check the timing for their reset net, GSR is not the best option as it is not timed by the timing analysis tool. LSR is included in the timing report and can be used by the user instead of GSR. To use the LSR usage case there must be no GSR instantiated in the design, no GSR_NET constraint specified, and the software settings used do not infer any GSR resource.

Required setting are stated as below:

1. Under Synthesis Design Strategy settings for GSR: Force GSR = “FALSE” (Synplify Pro) / ”NO”(LSE)
2. Under Map Design Strategy settings for GSR: Infer GSR = “F” (UNTICKED)
3. Instantiation of GSR is not present in user RTL.
4. Constraint file does not contain any user selected signal for GSR resource.

# 3. Controlling GSR in Module/Component level

This section contains details on how to control the behavior of GSR usage on a module or component level. In some cases, we wanted to allow a section of the design to continue functioning whenever a reset is asserted. Controlling the GSR with the use of attributes to prevent influence of the GSR resource to a particular component or module. Similarly, it is also possible for a component or a module to response to the GSR even if the component/module reset is different with the inferred GSR case.

## 3.1. GSR Attribute Syntax and Values

By default, all Lattice primitives responds to the GSR signal if it is present, hence users must specify which component/module in the design must be altered in terms of responding with the GSR resource. The available values are shown below:

- ENABLED – This is the default value on most library elements. This allows module or component to respond to GSR.
- DISABLED – This prevents the hierarchy or element from responding to the GSR value.

**The syntax of the attributes is shown below:**

**Verilog Syntax – Synplify**

module [module_name] (<ports>) /* synthesis GSR= "[value]"*/;

**VHDL Syntax**

ATTRIBUTE GSR : string;

ATTRIBUTE GSR of [component_instance]: label IS "[value]";

or

ATTRIBUTE GSR : string;

ATTRIBUTE GSR OF [module_name]: entity IS "[value]";

**Verilog Example Code:**

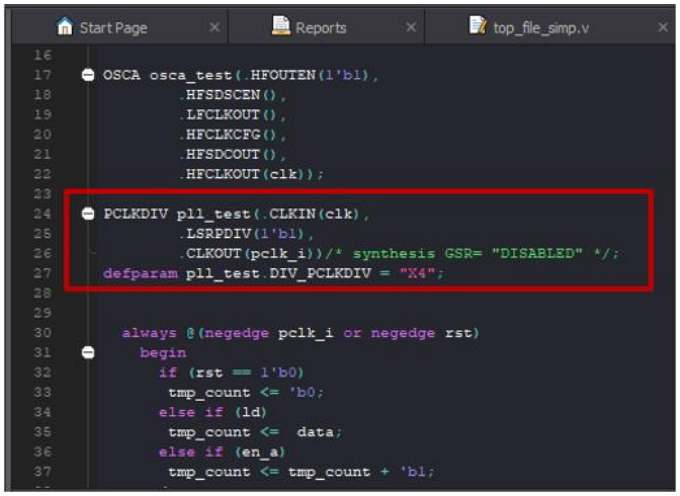


**Figure 3.1. Example Verilog design that uses GSR attribute**

### VHDL Example Code

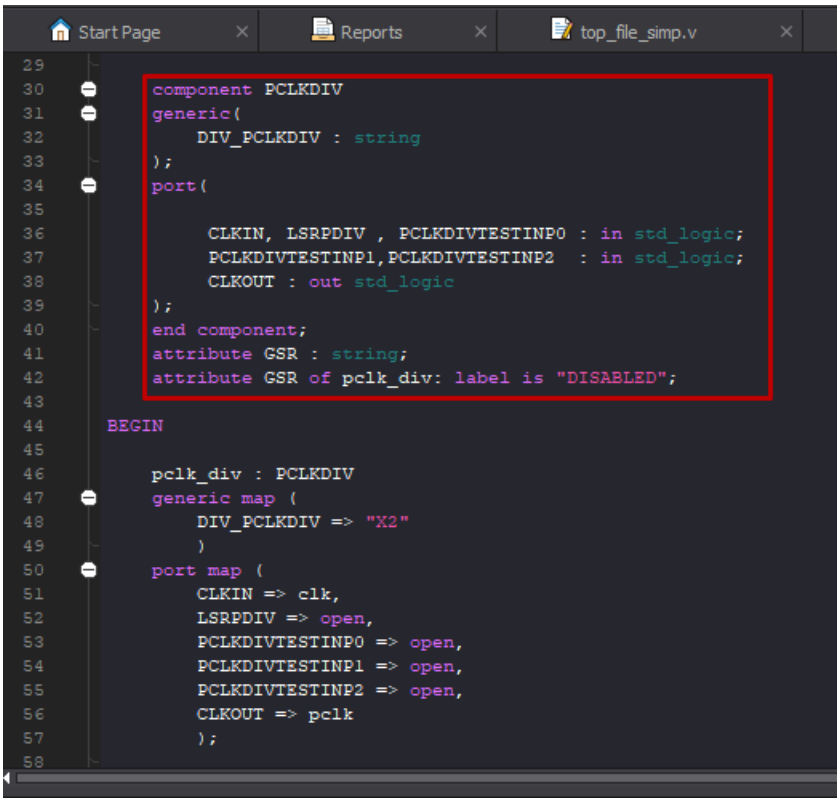


**Figure 3.2. Example VHDL design that uses GSR attribute**

To verify that the Attributes are properly propagated, check the *.vm file found in the design directory. For Example, In the below *.vm file we can see that the PCLKDIV's GSR response is Enabled, and Disabled respectively based on the attribute value set on above example codes:

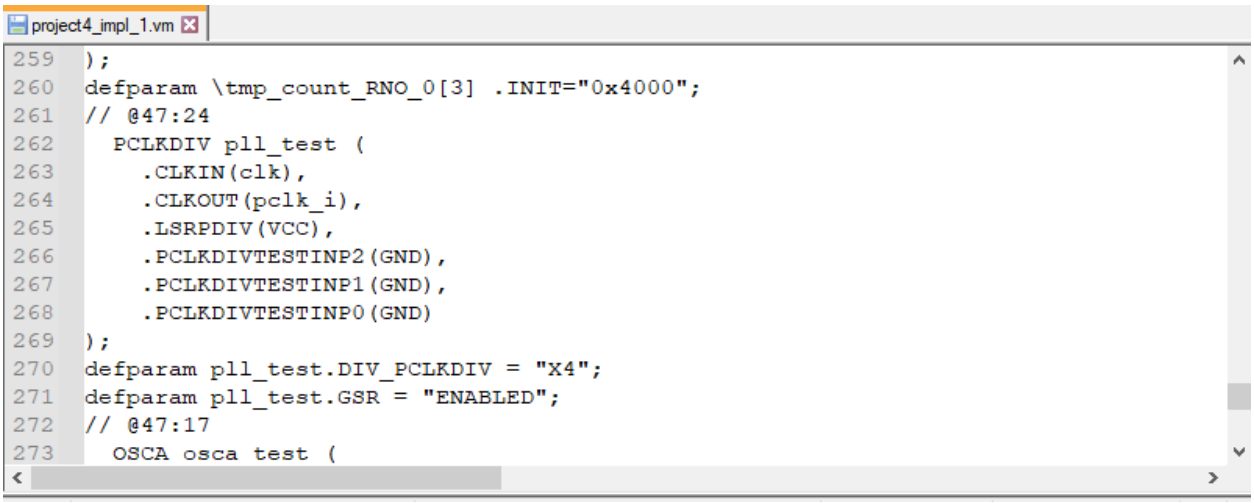


**Figure 3.3. GSR enabled on PCLKDIV checked in *.vm**

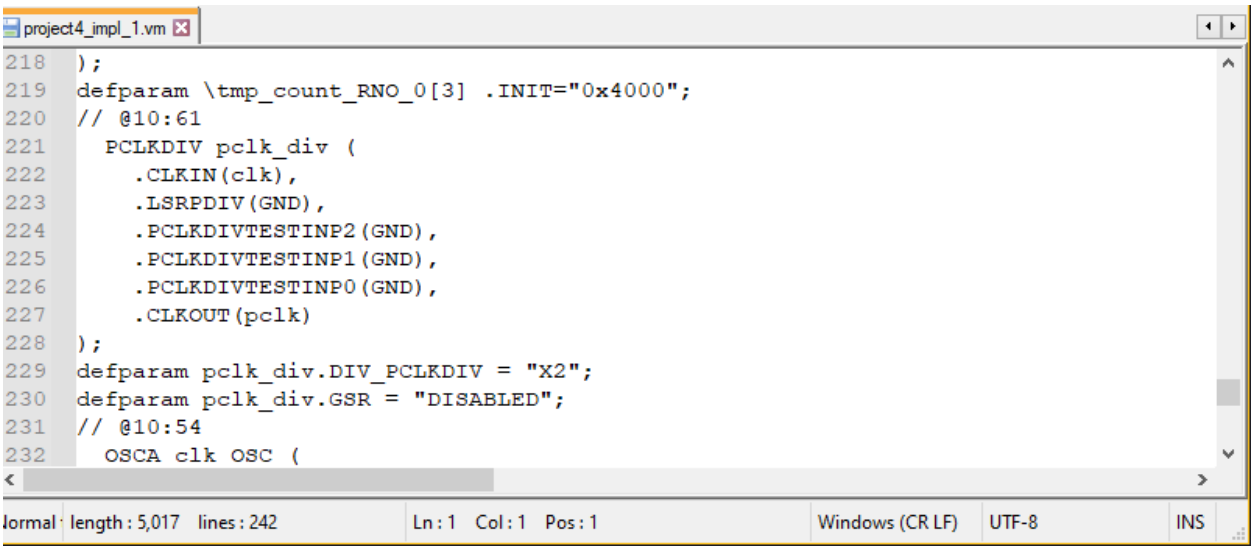


**Figure 3.4. GSR disabled on PCLKDIV checked in *.vm**

# 4. GSR Usage Guidelines

This section discusses the general guideline of when to use and not to use GSR. GSR must be used with caution and must be used only when absolute necessary.

## 4.1. General Considerations

1. Use GSR when applicable with the following considerations:
- Generally safe for small size and/or low speed design.
- Consider LSR when GSR does not provide stable initialization for reset release timing sensitive logic.
- Avoid the use of GSR for reset timing sensitive designs. Instead, use LSR to get the STA covered.
2. It is advisable to use inferred GSR modes rather than instantiated GSR. Also, when a reset net needs to take GSR resource, use user specified inferred GSR mode.
3. Avoid instantiated GSR mode in multiple reset designs since there is only one component.
4. Make sure GSR options are properly set to select a desired GSR mode in Synthesis and Map Design Strategy settings.

## 4.2. GSR Usage Guidelines for Simulation

This section tackles the common issues found in performing simulation on a design with GSR implementation. It also covers the proper usage of GSR when performing simulation.

### 4.2.1. GSR Resource and RTL Functional Simulation

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 is easily implemented correctly for both the RTL (pre-synthesis) design and the netlist (post-synthesis and post-route) versions of the design.

However, reset functionality may incorrectly be implemented during RTL functional simulation if GSR resource usage case is an Instantiated GSR and design implements multiple resets. The Instantiated 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 post-map netlist, the entire design responds correctly to the reset used for GSR. This is because the design now consists of Lattice components that have been modeled to take the GSR information into account as part of their functionality. The post-map design resets 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 still resets 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 or VHDL simulations produce errors if there are library elements in the design that use the GSR information and instantiations of GSR is not present in the top-level of the design. Users 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).

**Note**: Errors from the simulation tool are generated if the instantiation of the below primitives is not present.

**GSR Verilog Example:**

```
GSR GSR_INST (.GSR_N (<global reset sig>), .CLK(<clock signal>));
```

**GSR VHDL Example:**

```
PORT (GSR_N: IN std_logic; CLK: IN std_logic  );
END COMPONENT;

GSR_VHDL: GSR
      PORT MAP (
            GSR_N => <global reset sig>,
            CLK => < <clock signal>);
```


```
PORT (GSR_N: IN std_logic; CLK: IN std_logic  );
END COMPONENT;

GSR_VHDL: GSR
      PORT MAP (
            GSR_N => <global reset sig>,
            CLK => < <clock signal>);
```

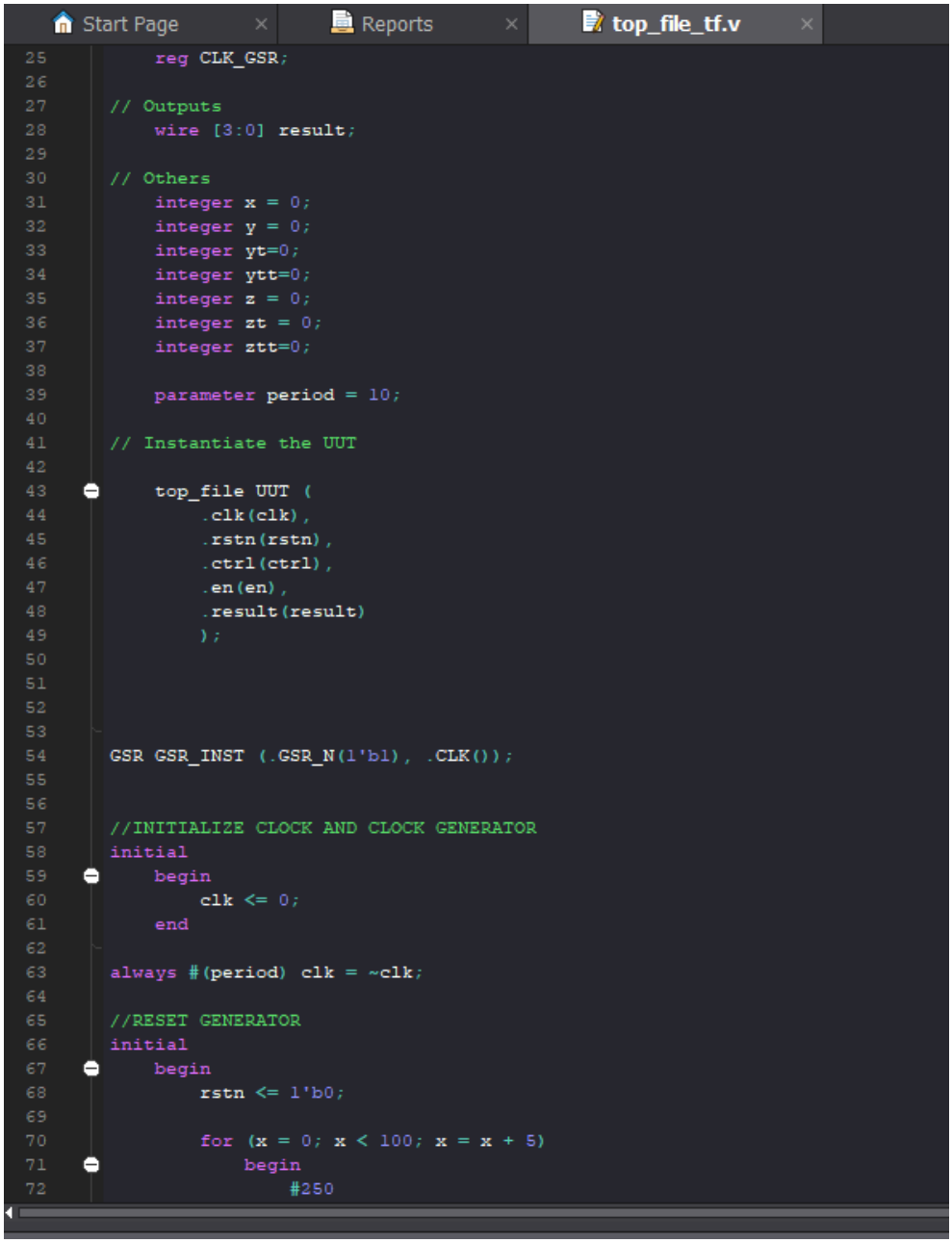


**Figure 4.1. Example code for Verilog test bench**


**Figure 4.2. Example code for VHDL test bench**

In general, The GSR instantiation in testbench is required when RTL design is composed of Lattice Components/Primitives that requires a GSR net connection in its simulation model. Some of examples of these components are IO/DDR, Registers, DSP, and Hard Ips (like PCIe, SGMII, DPHY). It is also required if a Netlist simulation (post-synthesis and post-route) is performed as it contains synthesized and translated primitives from the User RTL. The best place to add the GSR is the testbench, do not instantiate the GSR primitive in the RTL unless it is needed.

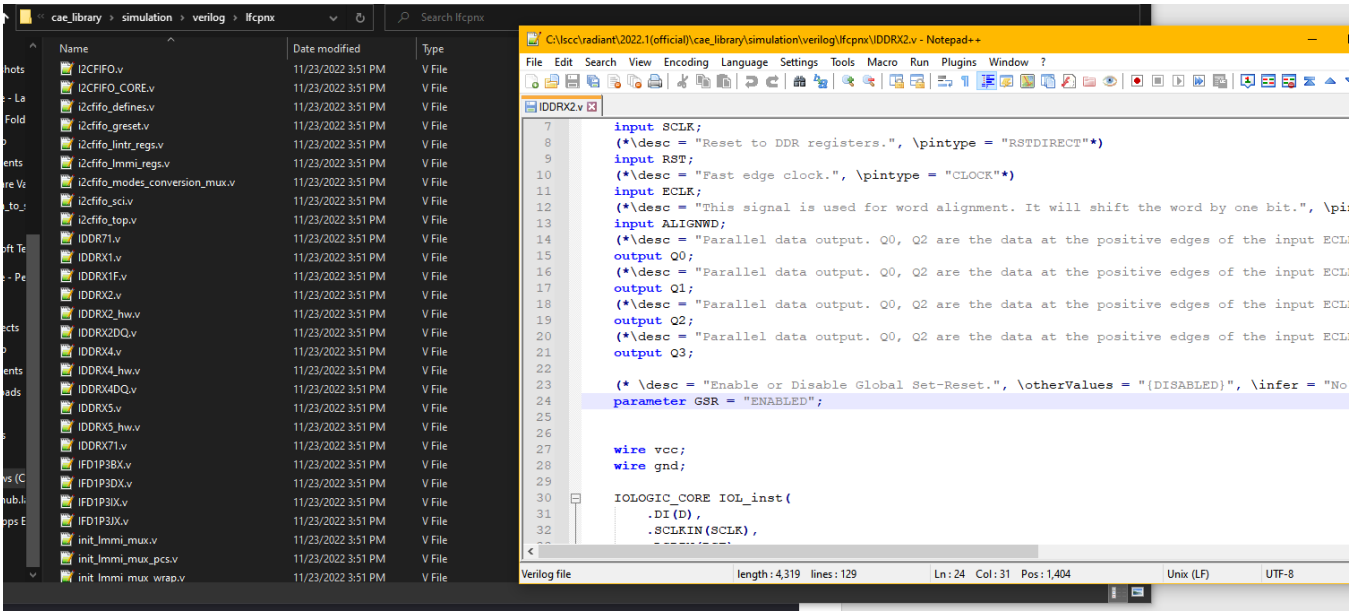


**Figure 4.3. Simulation Model Example (Lattice component where GSR is enabled)**

Typically, when simulating a design that has an IP or user IP with GSR implementation, error may arise in running the simulation as shown on the Figure 4.4.

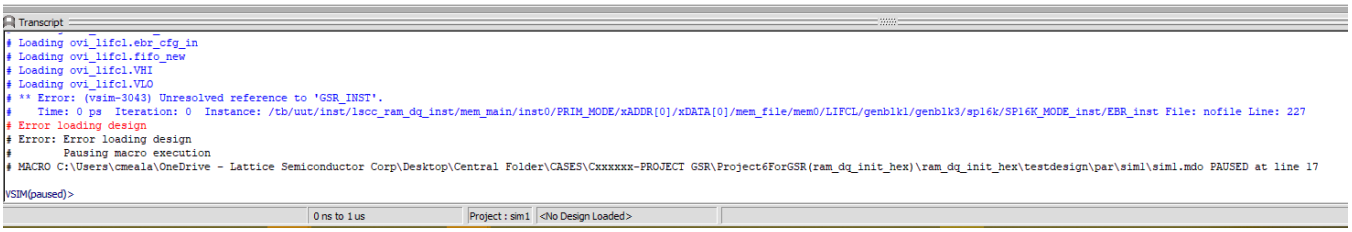


**Figure 4.4. Error encountered without GSR instantiation on design with Lattice components/primitives**

The recommended resolution to this problem illustrated below:

1. Add the GSR primitive in the test bench.

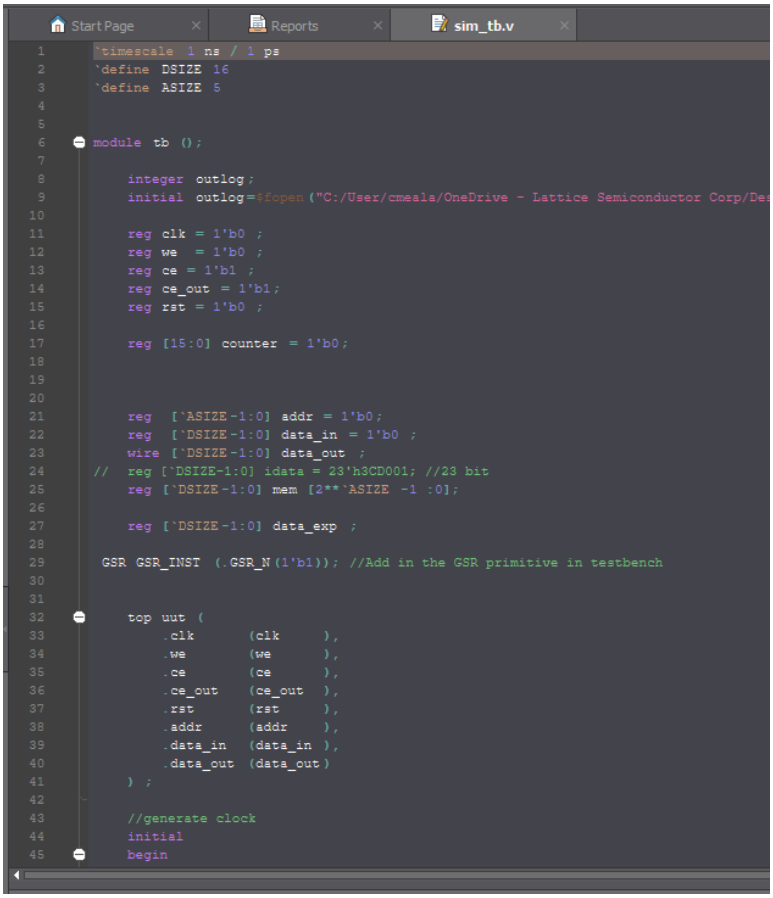


**Figure 4.5. Example test bench with GSR primitive instantiation**

2. Recompile the design.

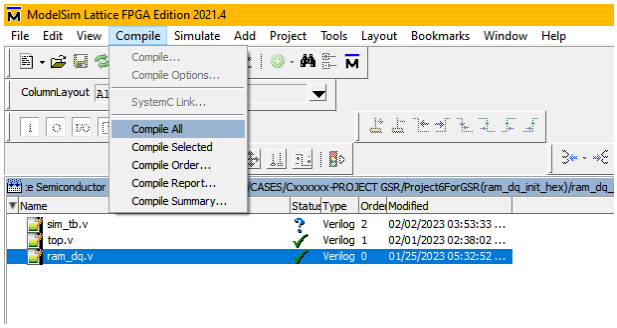


**Figure 4.6. Showing the manual compilation of the design**

3. Start the simulation and press ‘OK’.

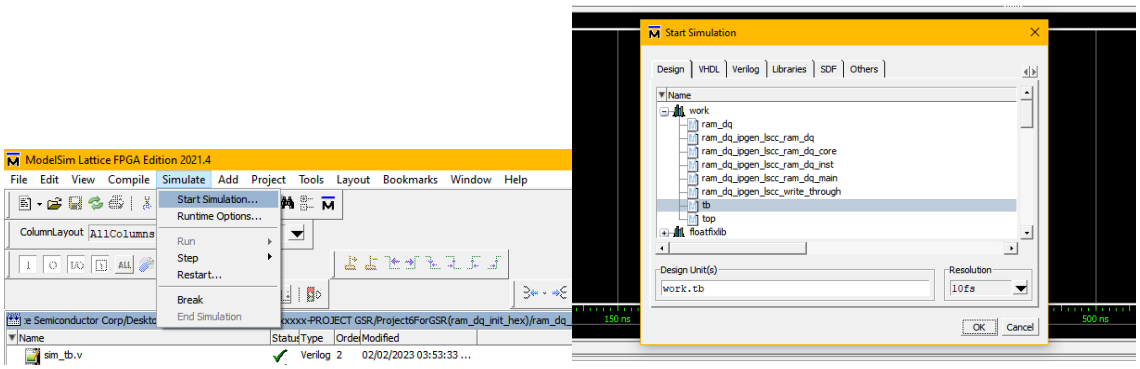


**Figure 4.7. Showing how to start simulation and selecting test bench**

**Note**: In case ‘design unit was not found error’ is encountered, add in the necessary libraries for the simulation in libraries window as shown below:

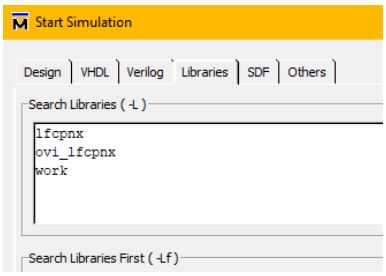


**Figure 4.8. Showing the required libraries to compile prior simulation**

4. Add the signals desire to simulate and monitor.

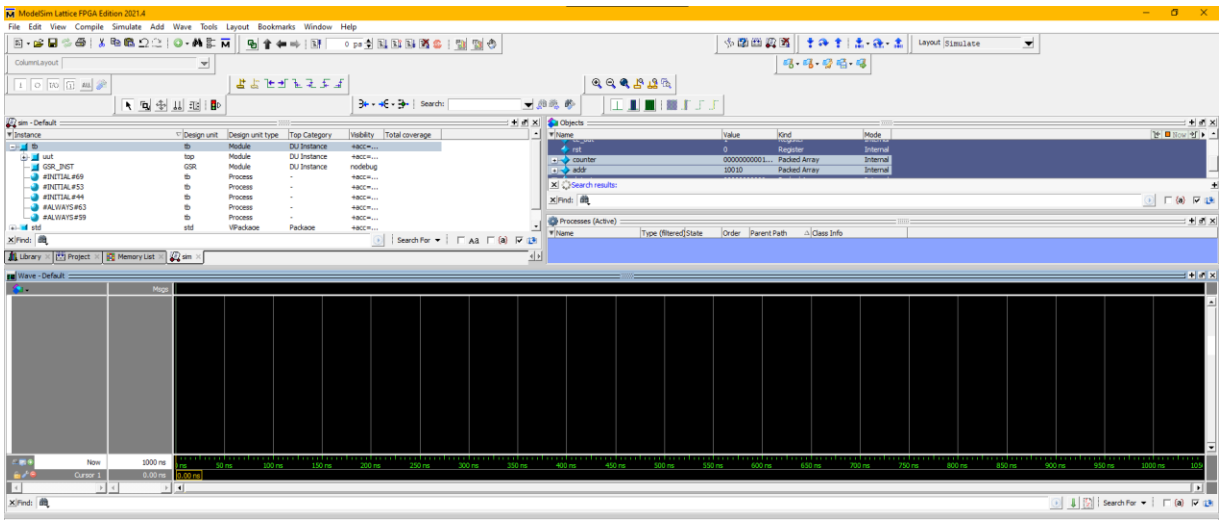

**Figure 4.9. Adding signals for simulation**

**Note**: Other possible mistake is having a different GSR instance name aside from GSR_INST since this is the default naming from Lattice’s IPs.

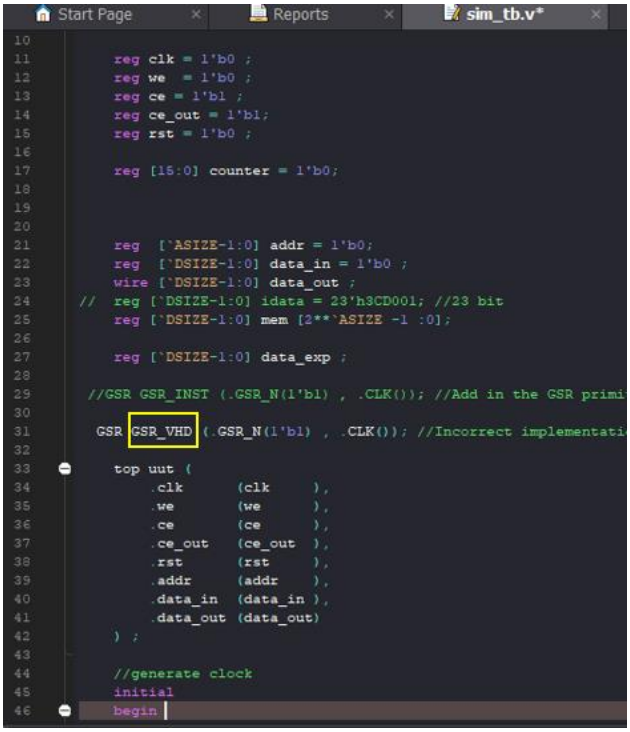


**Figure 4.10. Wrong GSR instance name in test bench**

### 4.2.2. Mixed Language Simulation and the GSR Resource

Compared to Diamond, Radiant does not utilize any VHDL libraries, instead, it uses the same library source files as the Verilog libraries. If users look into the modelsim.ini file found in <radiant_directory>/modeltech (as shown below) , for example, ovi_lifcl where “ovi_” prefix denotes that it is a Verilog logical library , and lifcl wherein missing prefix denotes that it is a VHDL logical library; Although the naming is different, Both the ovi_lifcl , and the lifcl logical library are compiled with the same Verilog source files.

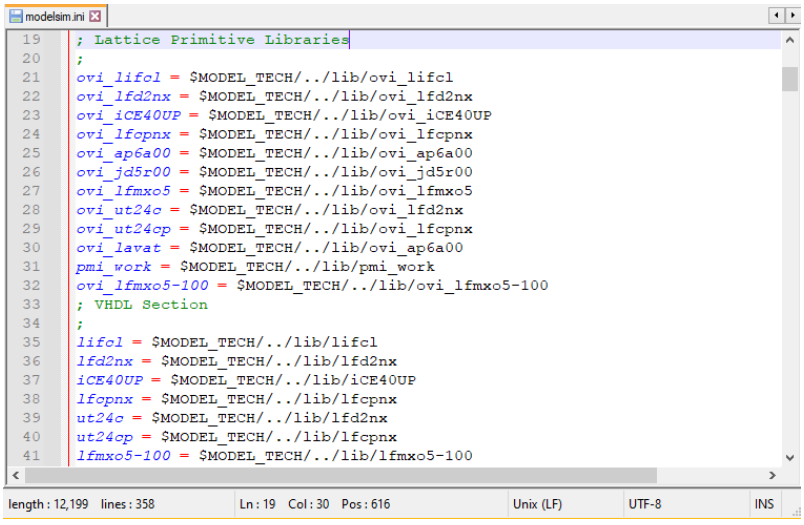


**Figure 4.11. modeltech.ini (modeltech/lib is where users can find the logical libraries)**

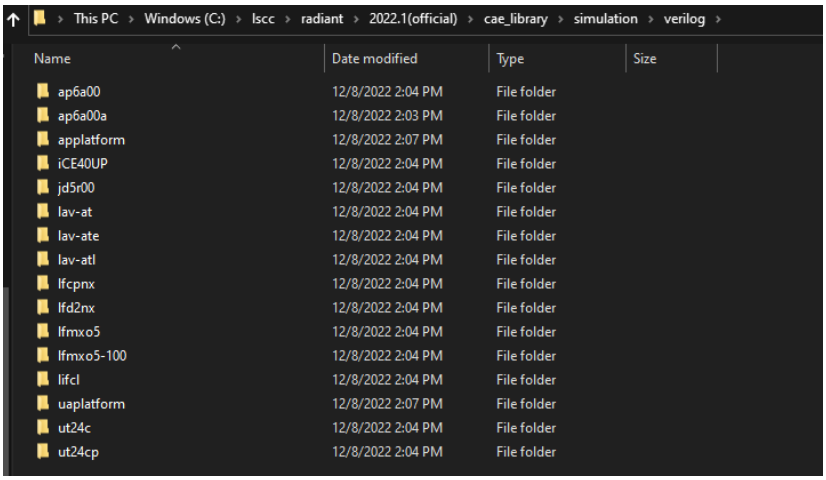


**Figure 4.12. Radiant' s Verilog Source Libraries**

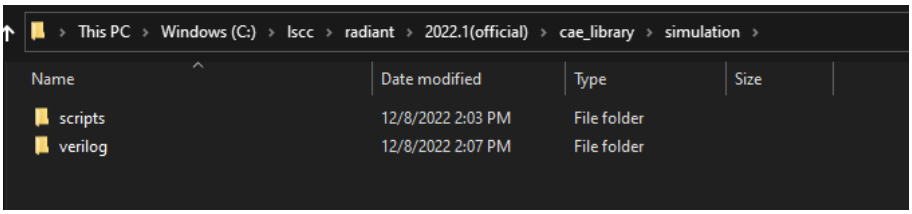


**Figure 4.13. Radiant does not have a VHDL source library**

With the above information, the ability to simulate GSR resource is simplified when using either Verilog and/or VHDL. In Diamond, this is not the case as the GSR resource is handled differently in Verilog and VHDL, 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 and VHDL simulation models access the GSR state by referring to a signal in a package that is used by the model.

In general, RTL, Post-Synthesis, and Post-Route simulation encounters the same issue as described in a single language design, a GSR instantiation should be utilized in the testbench specially if lattice components that utilize the GSR resource are used in the design to avoid error seen in Figure 4.4.

### 4.2.3. Library Compilation on 3rd party Simulator

As discussed on the previous section, simulation fails due to the presence of GSR instance that should be declared from the library or test bench. This is a common issue experiencing when simulating IPs that has GSR implementation. In this section it covers some guideline on how to compile the user design to be able to run simulation using 3rd party tool.

The Radiant tool has tcl script called "cmpl_libs" to help the user compile the libraries for the 3rd party simulators.

This is located at /<Lattice_Radiant_Install_path>/cae_library/simulation/scripts/cmpl_libs.tcl. Inside the script it has full description of its usage as shown below.

```
cmpl_libs.tcl -sim_path <sim_path> [-sim_vendor {mentor<default>|cadence|aldec}] [-
device {ice40up|lifcl|lfcpnx|lfd2nx|lfmxo5|ap6a00|ut24c|ut24cp|all<default>}] [-
target_path <target_path>]
```

More information can be found on the Radiant help: User Guides > Simulating the Design > Third-Party Simulators

**Example problem in using Xcelium 20.09.004:**

```
Error received after compiling libraries:
xmelab: *E,CUVMUR (./pciphy/rtl/pciphy.v,5560|24): instance
'tb_top.u_pciphy@pciphy<module>.lscc_mpcs_top_inst@pciphy_ipgen_lscc_mpcs_top<module
>.u_pcsrefmux' of design unit 'PCSREFMUX' is unresolved in
'worklib.pciphy_ipgen_lscc_mpcs_top:v'.
```

```
GSR GSR_INST (.GSR_N(reset_n), .CLK(mpcs_clk));
            |
xmelab: *E,CUVMUR (./pciphy/testbench/tb_top.v,1747|11): instance 'tb_top.GSR_INST'
of design unit 'GSR' is unresolved in 'worklib.tb_top:v'.
xrun: *E,ELBERR: Error during elaboration (status 1), exiting.
```

- Solution: performing library compilation using xrun -v_version -f <script> solves the issue.

```
Sample script:
-reflib /home/<user>/<folder>/lfcpnx -reflib /home/<user>/<folder>/uaplatform
././testbench/tb_top.v
-y ././rtl
-y ././misc
-y /./testbench
-incdir ././rtl
-incdir ././misc
-incdir ././testbench
-libext .v
-sv

Commands to run the script: xrun -v_XCELIUM20.09.004 -f run_xcelium
```

**Note**: This is a similar case across third-party simulators wherein compiling the necessary libraries solves the GSR instance issue.

## 4.3. GSR Usage with Reveal

When debugging a design with GSR implementation using Reveal, users need to take certain considerations into account. Reveal is a powerful tool that allows users to trace signals routed to the fabric. However, when Reveal is inserted into a design with GSR, there are certain things to consider.

- Attribute to Disable GSR: Synthesis tools automatically add an attribute to disable GSR on the Reveal core when it is inserted into the design (See Figure 4.14). This attribute ensures that the Reveal debugging process does not interfere with the GSR implementation.


**Figure 4.14. Synthesis attribute in Reveal Core for disabling GSR**

- Trace Signal Routing: With Reveal, users can trace signals within the design's fabric, gaining insights into the signal flow and identifying any potential issues. Ensure to exclude hard routes or routes with physical restrictions so that Reveal provides a clear view of the signal path.
- Reveal error: In Radiant versions 2023.1 and earlier, users may experience unexpected issues with the Reveal tracing. Please refer to Appendix A. Workarounds for Reveal with GSR Issues.

## 4.4. GSR hardware in FPGA

This section discusses the functional block diagram of GSR and its signal functions. The GSR hard block does not have configuration logic, but it uses logic gates to generate similar logic to configure the GSR_N. The Nexus Platform GSR uses synchronizers to enable synchronous reset application. By default, in hardware, GSR functions as an asynchronous reset.

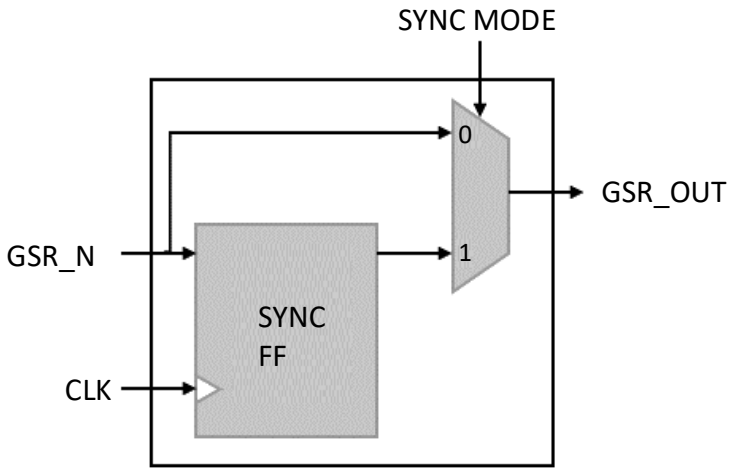


**Figure 4.15. GSR_N functional block diagram**

Figure 4.15 shows how the logic gates configure the GSR_N (input) into either Synchronous GSR or Asynchronous GSR. If the SYNC MODE is set to '0' or set to asynchronous, the GSR_OUT directly uses the value of the GSR_N input. On the other hand, if the SYNC MODE is set, the GSR_N is the input for the internal synchronizer circuit with respect to the applied user clock. In this mode, the reset is only activated at the clock edge. Note that the CLK is not needed if set as ASYNC.

# Appendix A. Workarounds for Reveal with GSR Issues

In Lattice Radiant versions 2023.1 and earlier, Reveal contained a structural issue which resulted in undesired behavior and inability to capture trace data. This is indicated by error message "*core0 incorrect pattern readout"* as shown on Figure A.1.

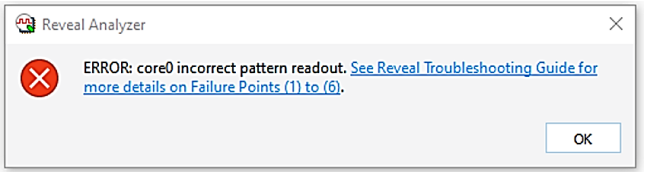


**Figure A.1. Reveal Analyzer error when using GSR reset as trigger source**

To confirm, an initial check can be done to verify whether the Reveal core has successfully added the attribute to disable GSR. Follow these steps:

1. Navigate to the implementation folder (i.e., impl1) of the Radiant project.
2. Within the implementation folder, locate the *reveal_workspace* folder.
3. In the *reveal_workspace* folder, find and open the file named **_la0_inst.v*.
4. Inside this RTL-generated file of Reveal, please verify if this line "**object /* synthesis GSR=DISABLED */**" is present just below the module instantiation.

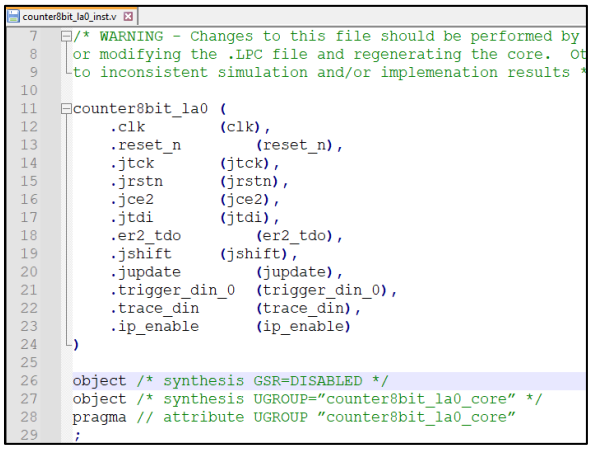


**Figure A.2. Sample Reveal RTL instance**

If the synthesis attribute that disables GSR is successfully added but error persist, follow these steps:

- From Synthesis Strategy Options, set Force GSR to FALSE (For Synplify Pro), No (For LSE).


**Figure A.3. GSR Synthesis Strategy Setting for Synplify Pro**

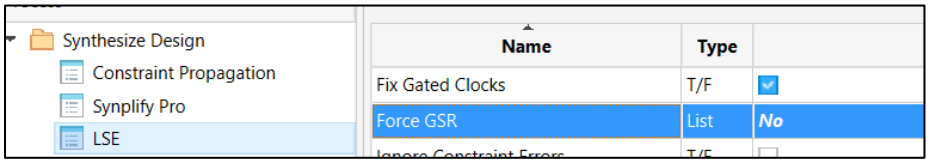


**Figure A.4. GSR Synthesis Strategy Setting for LSE**

- Uncheck/Untick MAP Infer GSR.

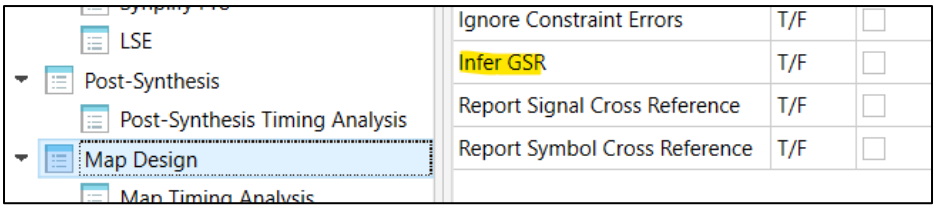


**Figure A.5. GSR Map Design Strategy Setting**

- Remove any GSR instantiation from user RTL. (i.e., //GSR GSR_inst (.CLK(clk_125), .GSR_N (perst_n_i));).

# References

For more information, refer to:

- Lattice Radiant Software User Guide.
- Lattice Nexus Platform website
- Lattice Radiant FPGA design software
- Lattice Insights for Lattice Semiconductor training courses and learning plans

# Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

For frequently asked questions, please refer to the Lattice Answer Database at https://www.latticesemi.com/Support/AnswerDatabase.

# Revision History

**Revision 1.0, September 2023**

| Section | Change Summary |
|---|---|
| All | Initial release. |


www.latticesemi.com