XAPP414 (v1.1) October 2, 2001

Xilinx/Synopsys Formality Verification Flow

Authors: Mujtaba Hamid and Yenni Totong

Summary

This application note covers the logic equivalency flow using Xilinx ISE software with Synopsys Formality. The target audience is designers familiar with the independent Xilinx HDL software design flow.

Introduction

With rapid increases in FPGA design sizes, new simulation and logic verification methodologies must be explored to expedite the verification of design logic and functionality. For logic equivalency checking, formal verification is quickly gaining acceptance by designers creating multi-million gate designs, because of its accuracy and speed. Using Synopsys Formality with Xilinx FPGA designs, designers can check logic equivalency between the RTL (pre-synthesis) and post-implementation (after PAR) designs.

Formal verification requires the presence of a reference (verified) design, against which it checks the other design netlists (post-synthesis, post-implementation). A netlist at any point in the design flow, for example pre-synthesis or post-implementation, can be used as the reference. However, the RTL (pre-synthesis) netlist is most commonly used as the reference. The Xilinx/Formality formal verification flow currently supports only the Verilog language.

Software and Device Support

The formal verification flow between Xilinx designs and Synopsys Formality is supported by the following software:

- Xilinx Software: ISE Alliance 4.1I (UNIX version only) and later
- Synopsys Software: FPGA Compiler II version 3.6 and later, and Formality version 2001.06 and later
- Platform Support: Solaris 2.7 and later

Formal Verification is available for the following devices:

- Spartan™-II
- Virtex™, Virtex-E, and Virtex-II

Flow Summary

The following verification points are available for the Xilinx - Formality formal verification flow:

1. **RTL** — This is the pre-synthesis design code, usually used as the reference design.
2. **Post-NGDBuild** — This is equivalent to the post-synthesis netlist, consisting of gate-level SIMPRIM primitives.
3. **Post-MAP** — At this stage, the design has been mapped into the target device by the Xilinx implementation tools, but has not been routed as yet.
4. **Post-PAR** — At this stage, the design is completely placed and routed, and the resulting structural netlist closely resembles the design layout as it will appear in silicon.

Verifications can be done between any two points listed above, for example RTL vs. Post-NGDBuild, RTL vs. Post-PAR, or Post-NGDBuild vs. Post-PAR. The formal verification flow with Xilinx designs and Formality is shown in Figure 1.

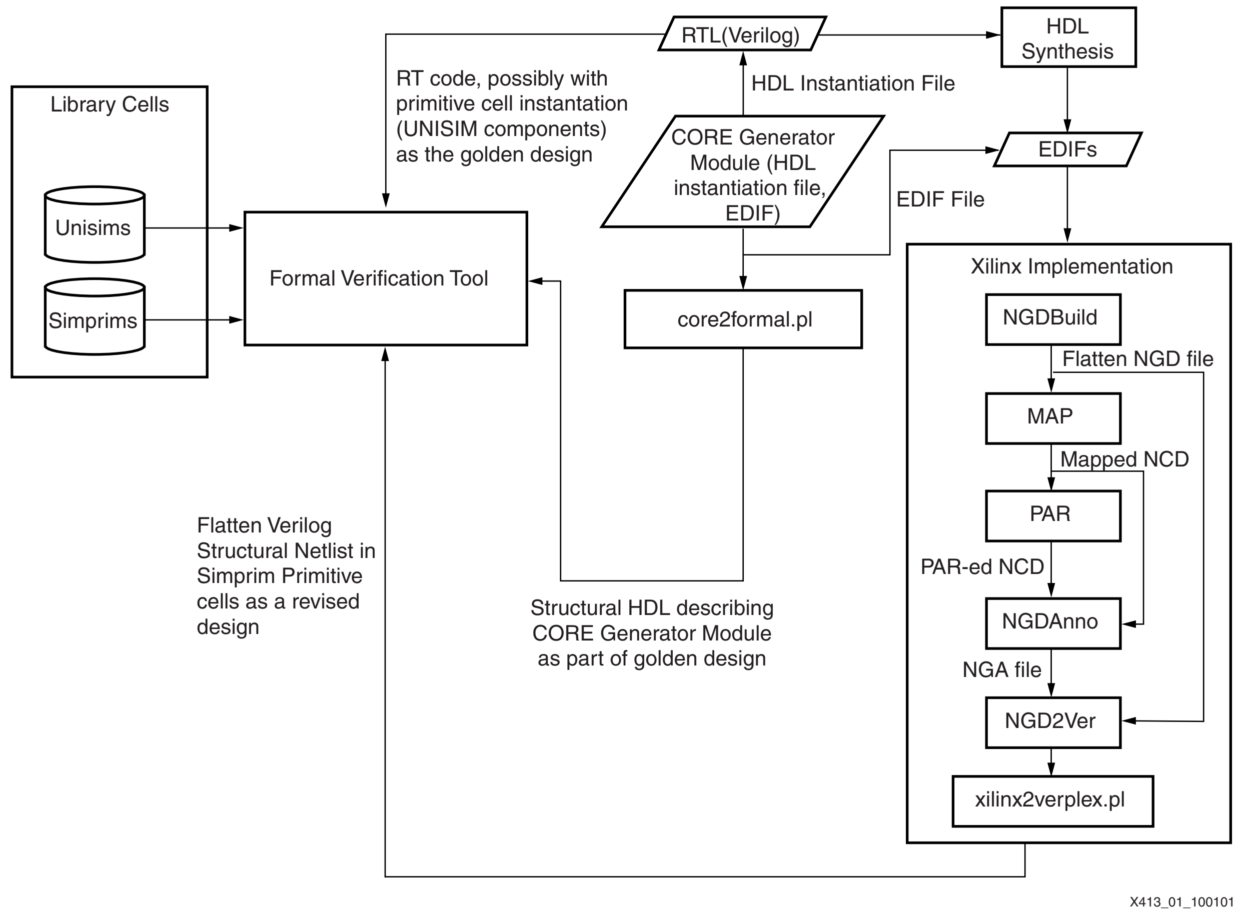

RTL(Verilog)
HDL Synthesis
Library Cells
Unisims
Simprims
RT code, possibly with primitive cell instantation (UNISIM components) as the golden design
HDL Instantiation File
CORE Generator Module (HDL instantiation file, EDIF)
EDIFs
EDIF File
Formal Verification Tool
core2formal.pl
Xilinx Implementation
NGDBuild
Flatten NGD file
MAP
Mapped NCD
PAR
PAR-ed NCD
NGDAnno
NGA file
NGD2Ver
xilinx2verplex.pl
Flatten Verilog Structural Netlist in Simprim Primitive cells as a revised design
Structural HDL describing CORE Generator Module as part of golden design
X413_01_100101

Figure 1: **Xilinx/Formality Formal Verification Flow**

Sample Flows

Below are two sample flows that can be run using Xilinx designs and Formality. The first example compares the logic equivalency between the RTL (pre-synthesis) and the Post-NGDBUILD designs. The second example checks the equivalency between the RTL and Post-PAR (back-annotated) designs. Neither of these flows check for timing issues, since Formality is a logic equivalency checker.

RTL vs. Post-NGDBuild

Reference design - Behavioral (RTL) Verilog design (netlist)

Implementation design - Post-NGDBuild Verilog netlist

The flow is comprised of the following steps:

1. Synthesize the Verilog design files with Synopsys FPGA Compiler II, targeting a Xilinx (Virtex/Virtex-E/Virtex-II/Spartan-II) FPGA. An EDIF netlist file is produced at the end of this step.
2. Create the Post-NGDBuild Verilog netlist, using the Xilinx implementation tool from the GUI or the command line.

   From the GUI:

   a. Create a Xilinx ISE Project using the EDIF netlist from Step 1.

   b. Create a Post-NGDBUILD Verilog netlist using the Xilinx ISE tools.

   Note: For more information on running ISE, refer to the ISE documentation available in the ISE Quick Start Guide (with the ISE 4.1I software) or the http://support.xilinx.com Xilinx support site.

From UNIX Terminal window:

a. Set up the Xilinx environment variables.

b. Process the EDIF file:

```
>ngdbuild <filename>.edf
```

c. Create Post-NGDBuild Verilog:

```
>ngd2ver  <infile>.ngd <outfile>.v
```

Note: If <outfile>.v is not supplied, ngd2ver outputs the same filename as the input file.

3. From the UNIX terminal window, run the ‘xilinx2formality.pl’ script:

```
>xilperl   $XILINX/verilog/bin/<platform>/xilinx2formality.pl  <filename>.v  >  <outfile>.v
```

Notes:

1. xilperl is a Perl application available with the Xilinx ISE software.
2. <platform> can be "sol" for solaris UNIX workstation, "hp" for HP UNIX workstation, or "nt" for PC platform.
3. Xilinx2verplex.pl removes extra cells in the Verilog netlist that are not needed for formal verification.

4. If a CORE Generator module is instantiated in your design, run ‘core2formal’ to create a ‘reference’ description for the module. Refer to the **Verification of Designs Containing Xilinx CoreGEN Components** section for more information.

5. Run the Formality flow to compare the two versions of the Verilog netlists. Refer to the **Formality Flow** section.

RTL vs post PAR

Reference design: Behavioral (RTL) Verilog netlist.

Implemented design: Post-PAR Verilog netlist.

The flow is as follows:

1. Synthesize the Verilog design files with Synopsys FPGA Compiler II, targeting a Xilinx (Virtex/Virtex-E/Virtex-II/Spartan-II) FPGA. An EDIF netlist file is produced at the end of this step.

2. Create Post-PAR Verilog netlist from the GUI or the command line.

   From the GUI:

   a. Launch the Xilinx software, and create a Xilinx ISE Project, using the EDIF netlists from Step 1.

   b. Create a Post-PAR Verilog netlist using the Xilinx ISE tools.

   The Xilinx ISE tools run NGDBuild, MAP, PAR, and NGDANNO and NGD2Ver to create a Post-PAR Verilog netlist.

From the UNIX Terminal window:

a. Process EDIF:

```
>ngdbuild <filename>.edf
```

b. Run MAP:

```
>map -o <mapped>.ncd <filename>.ngd
```

c. Run PAR:

```
>par <mapped>.ncd <par>.ncd <pcffile>.pcf
```

d. Process Post-PAR NCD for annotation:

```
>ngdanno  <par>.ncd
```

e. Create a Post-PAR Verilog file:

```
>ngd2ver <par>.nga <outfile>.v
```

3. From the UNIX terminal window, run the 'xilinx2formality.pl' script:

```
>xilperl  $XILINX/verilog/bin/<platform>/xilinx2formality.pl <filename>.v > <outfile>.v
```

Notes:

1. xilperl is a Perl application available with the Xilinx ISE software.
2. <platform> can be "sol" for solaris UNIX workstation, "hp" for HP UNIX workstation, or "nt" for PC platform.
3. Xilinx2verplex.pl removes extra cells in the Verilog netlist that are not needed for formal verification.

4. If a CORE Generator module is instantiated in your design, run 'core2formal' to create a 'reference' description for the module. Refer to the **Verification of Designs Containing Xilinx CoreGEN Components** section for more information.
5. Run the Formality flow to compare the two versions of the Verilog netlists. Refer to the **Formality Flow** section.

Formality Flow

This section briefly describes the Formality verification flow. For more detailed information or assistance on this flow, contact Synopsys Technical Support.

Before running Formality, make sure that the required design files have been created, as outlined in the previous section.

Setting Up the Environment

Setup environment variables to point to a Formality and Xilinx installation. Required variables are shown in Table 1.

Table 1: **Environment Variables Needed to Setup Synopsys Formality and Xilinx**

Name of Variable	Location Pointed by the Variable
Synopsys Variables	
SYNOPSYS	<synopsys_install_directory>/<platform>
LM_LICENSE_FILE	<port>@<license_server> or <license_install_directory>/<license_file>
PATH	$SYNOPSYS/fm/bin $path
Xilinx Variables	
XILINX	<xilinx_install_directory>
PATH	$XILINX/bin/sol

Synopsys uses a setup file for each project. Create this ".synopsys_fm.setup" file in the project directory by copying the template from $XILINX/verilog/formality/template.synopsys_fm.setup and renaming it to ".synopsys_fm.setup".

For more information on the use and customization possibilities of the ".synopsys_fm.setup" file, contact Synopsys.

Setting Up Xilinx Verification Libraries

The two Xilinx Verification libraries that must be used along with Formality for Formal Verification are:

- **UNISIMS** — The UNISIMS library contains the Xilinx primitives in RTL format. This library is required if the design contains any Xilinx primitives; for example, DCMs or block RAM.
- **SIMPRIMS** — The SIMPRIMS library contains the Xilinx primitives for back-annotated Verification (Post-NGDBUILD, Post-PAR). Since the back-annotated library is comprised completely of these gate-level primitives, this library must be compiled before verifying a Post-NGDBUILD or Post-PAR design.

The libraries must be read into Formality upon the start of the Verification flow. More details on this are provided in the **Verifying Design** section.

Verifying Design

All formality commands can be run either from the GUI or the UNIX terminal prompt. This section steps through the flow using the GUI; however, these commands can be added into a script that can be launched from a terminal prompt.

1. The ".synopsys_fm.setup" file should already be present in the project directory. The setup file must contain the Xilinx verification library path information, project path information, and other necessary directives for the Xilinx-Formality flow. The following lines comprise a sample file:

   ```
   set signature_analysis_matching true
   set dir [exec pwd]
   set XILINX /path/to/xilinx/install
   set search_path
       ". $XILINX/verilog/formality/unisims $XILINX/verilog/formality/simprims "
   ```

   Add these commands to the ".synopsys_fm.setup" file, if needed. Save and close the file.

   Additionally, ensure that the $XILINX variable is pointing at the correct location for the Xilinx install directory.

2. Start up the GUI by typing "formality" at the UNIX terminal prompt. Figure 2 shows the Formality GUI.

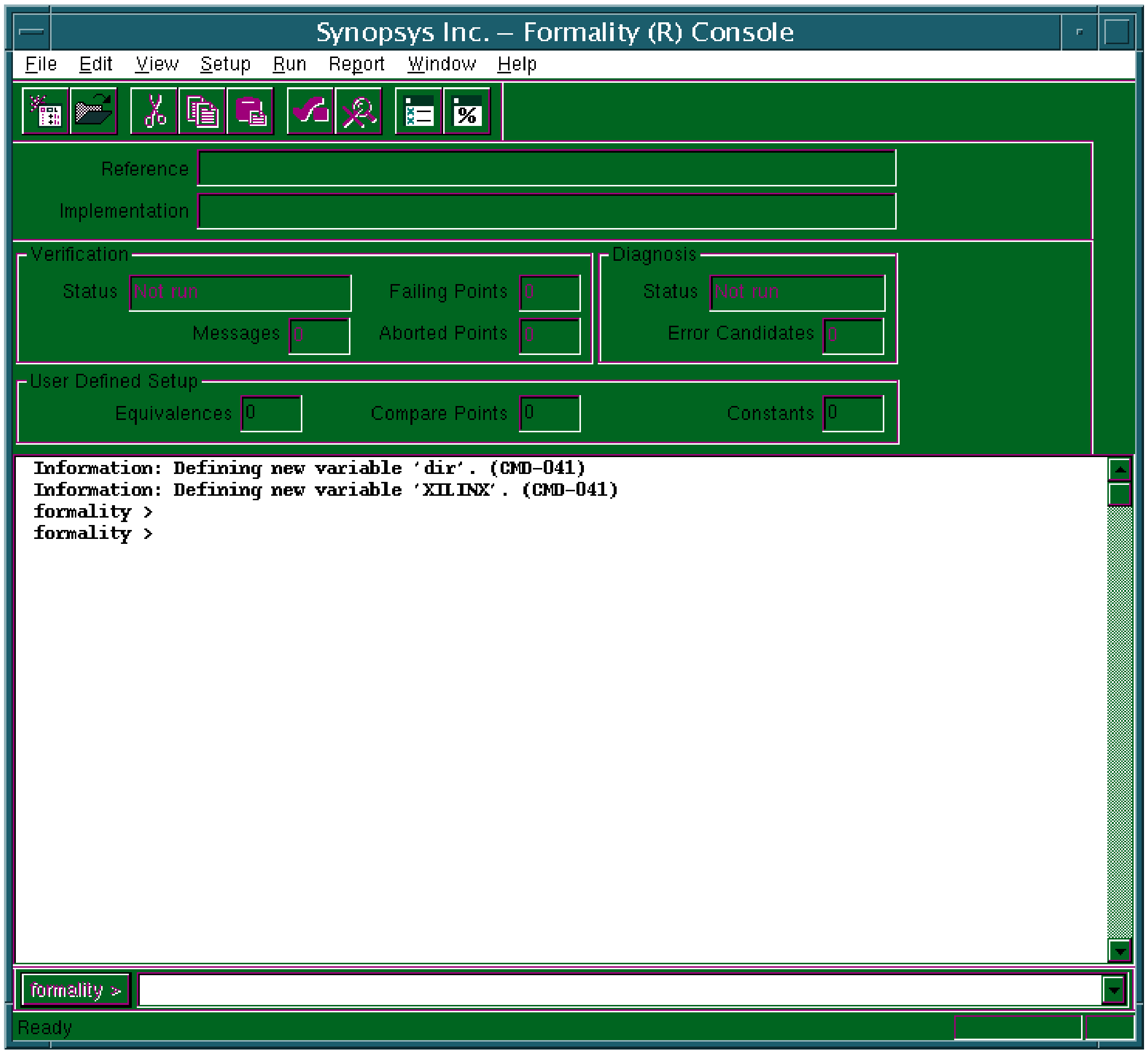
   The main section of the GUI shows the transcript of the commands and the log file generated. The commands can be entered in the bottom section of the GUI. Alternatively, a script can be called from the GUI by selecting File > Run Script from the pull-down menu.

3. The Xilinx Verification libraries, UNISIMS and SIMPRIMS, must be read into Formality before comparing the designs. To read the libraries, enter the following commands at the Formality command prompt:

   ```
   set XILINX /path/to/xilinx/install/
   source $XILINX/verilog/formality/unisims/unisims.fms
   source $XILINX/verilog/formality/simprims/simprims.fms
   ```

   Formality reads the contents of the Xilinx verification libraries. This might take up to ten minutes, depending on your network connection speed and processor speed.

4. Select File-> Open container.
5. Browse to xilinxlib.fsc and click "Open".


Synopsys Inc. – Formality (R) Console
File Edit View Setup Run Report Window Help
Reference
Implementation
Verification
Diagnosis
User Defined Setup
formality >
Ready

Status	Not run	Failing Points	0	Status	Not run
Messages	0	Aborted Points	0	Error Candidates	0
Equivalences	0	Compare Points	0	Constants	0

```
Information: Defining new variable 'dir'. (CMD-041)
Information: Defining new variable 'XILINX'. (CMD-041)
formality >
formality >
```

Figure 2: **Formality Main GUI**

6. Enter 'rtl' for the container name. This container is used to store the RTL (reference) design. The RTL container GUI displays, as shown in Figure 3.

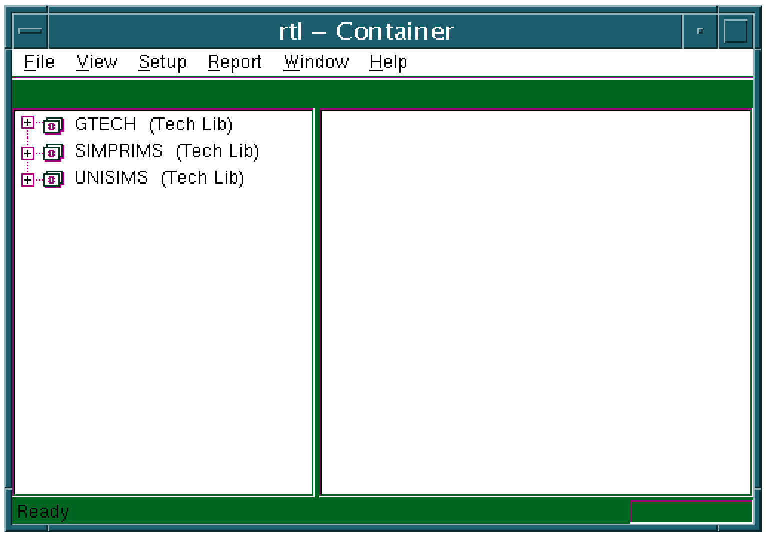
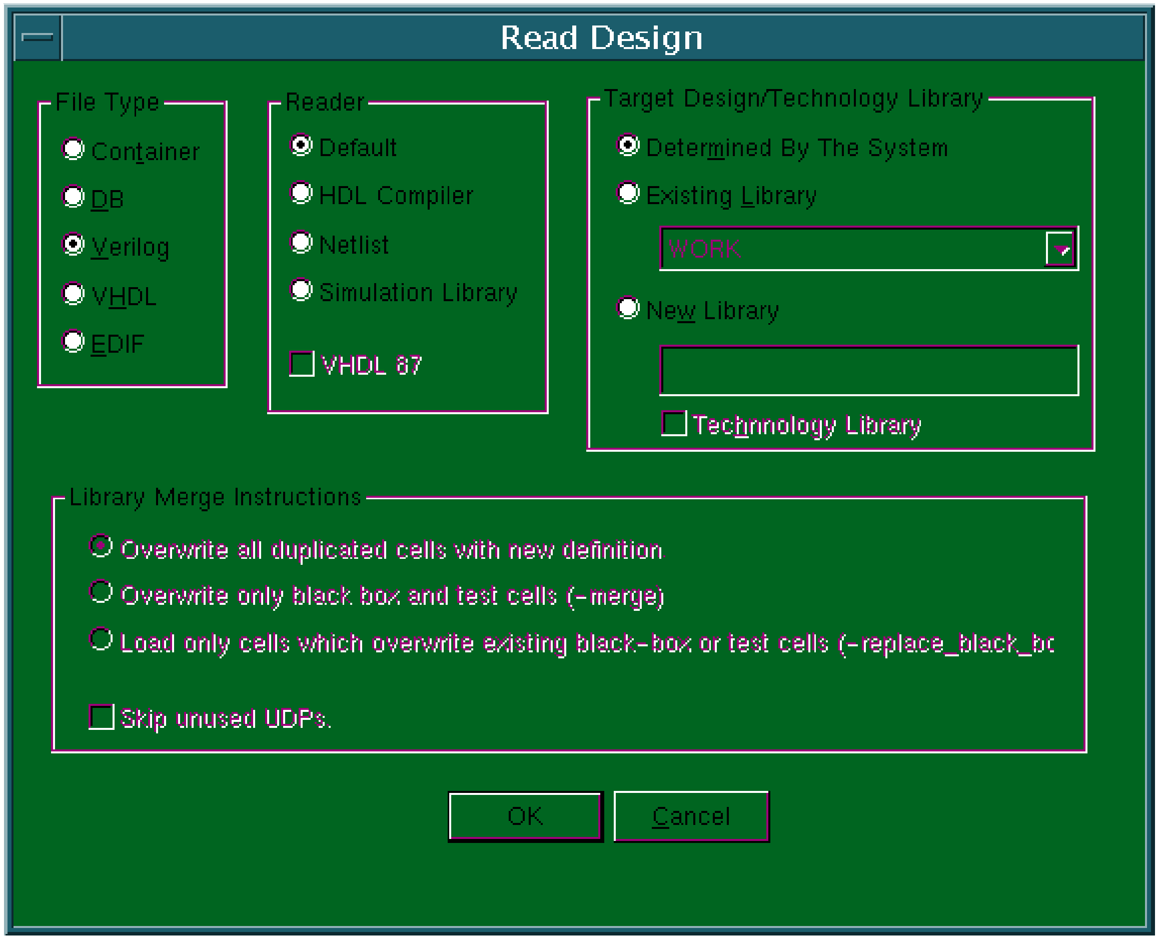


Figure 3: **RTL Container GUI**

7. In the rtl - Container window, select File > Read Design from the pull-down menu..
8. Select all of the RTL files in the design.

   Note: If you have CoreGEN modules in your design, do not select the <coregen_module>.v files at this time. The CoreGEN modules are added later. This is because the coregen modules are already instantiated as a black box in your design. To read the description of coregen modules, you must override the black box component. Otherwise, the following error occurs:

   ```
   Error: You are declaring a module 'tenths' which is already declared.
   (File:<coregen_module>.v)
   ```
9. Select Open on the "Read Design" browser window (see Figure 4) .
10. Click OK on the next window, to accept all the default settings.
11. Click on the + sign next to WORK and select the top-level design module.
12. Select the top-level module, Right-Click on that module and select "Set as Reference."
13. Select File > Link Design > Use Default Options from the rtl-container's pull-down menu.
14. Repeat steps 3 to 5 to create a container called "imp." This container is used to store the implemented design.
15. In the imp-Container window, select File > Read Design from the pull-down menu.


Read Design

File Type: Container, DB, [✓ Verilog], VHDL, EDIF

Reader: [✓ Default], HDL Compiler, Netlist, Simulation Library; ☐ VHDL 87

Target Design/Technology Library: [✓ Determined By The System], Existing Library (WORK), New Library; ☐ Technnology Library

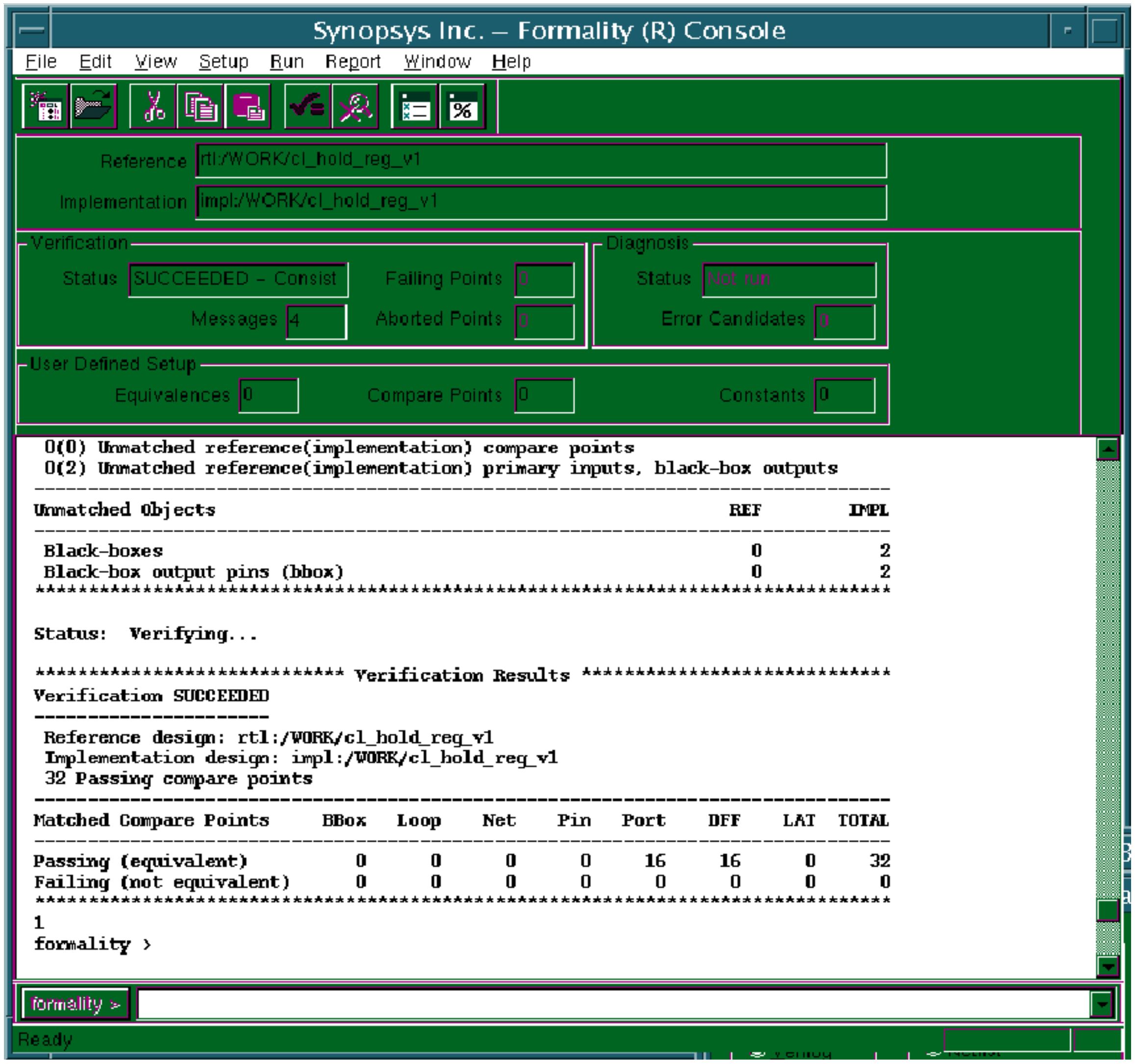
Library Merge Instructions:
[✓ Overwrite all duplicated cells with new definition]
Overwrite only black box and test cells (-merge)
Load only cells which overwrite existing black-box or test cells (-replace_black_bc
☐ Skip unused UDPs.

OK Cancel

Figure 4: **Read Design Options GUI**

16. Select the implemented Verilog file. This can be after NGDBUILD or after PAR. Ensure that this file has been processed through the xilinx2formality.pl script. See the **Sample Flows** section for more details.

17. Repeat steps 8 to 10 for the "imp" container window.
18. Right Click on the toplevel.v and select "Set as Implementation."
19. Select File > Link Design > use Default Options from the imp-container pull-down menu.
20. To include any CoreGEN modules to the reference design, refer to the **Verification of Designs Containing Xilinx CoreGEN Components** section.
21. From the Formality Console window, click Run -> Verify -> All Compare Points. If the designs compare successfully, the result is shown in the GUI under the "Verification" section, as shown in Figure 5.



Reference: rtl:/WORK/cl_hold_reg_v1
Implementation: impl:/WORK/cl_hold_reg_v1

Verification
Status: SUCCEEDED – Consist
Failing Points: 0
Messages: 4
Aborted Points: 0

Diagnosis
Status: Not run
Error Candidates: 0

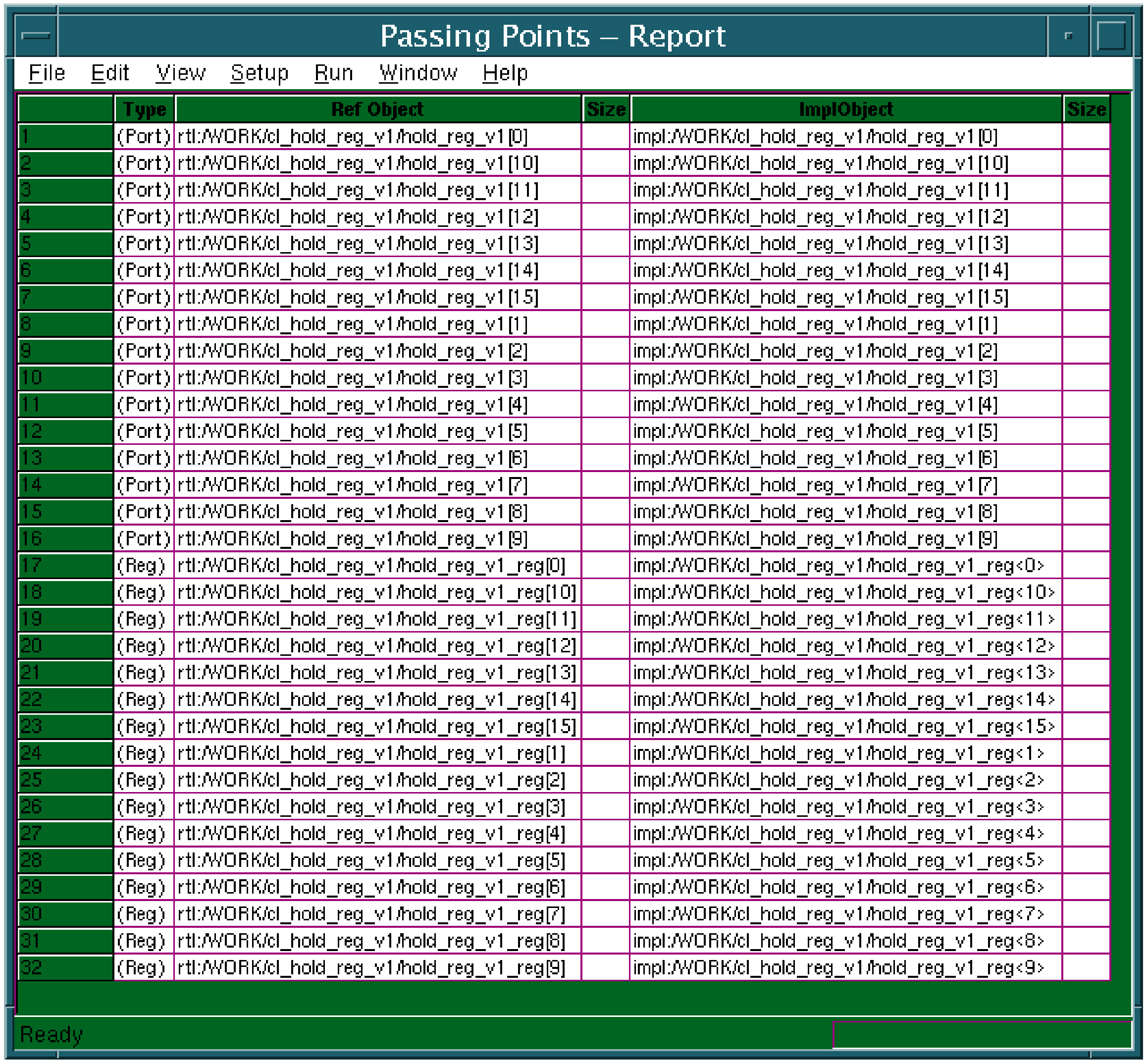
User Defined Setup
Equivalences: 0
Compare Points: 0
Constants: 0

```
 0(0) Unmatched reference(implementation) compare points
 0(2) Unmatched reference(implementation) primary inputs, black-box outputs
----------------------------------------------------------------------------
Unmatched Objects                                              REF      IMPL
----------------------------------------------------------------------------
 Black-boxes                                                     0         2
 Black-box output pins (bbox)                                    0         2
****************************************************************************

Status:  Verifying...

**************************** Verification Results ***************************
Verification SUCCEEDED
----------------------
 Reference design: rtl:/WORK/cl_hold_reg_v1
 Implementation design: impl:/WORK/cl_hold_reg_v1
 32 Passing compare points
----------------------------------------------------------------------------
Matched Compare Points     BBox   Loop    Net    Pin   Port    DFF    LAT  TOTAL
----------------------------------------------------------------------------
Passing (equivalent)          0      0      0      0     16     16      0     32
Failing (not equivalent)      0      0      0      0      0      0      0      0
****************************************************************************
1
formality >
```

Figure 5: **Formality Transcript Showing Results of Verification**

Once verification has completed, several reports can be generated to see passing and failing points. All of these report options are available under the "Report" pull-down menu in Formality.

An example of a "Passing Points" report is shown in Figure 6. This report is divided into two columns, one each for the Reference and Implementation designs. On each row, the report shows the signals in the two designs that were deemed logically equivalent by Formality.



	Type	Ref Object	Size	ImplObject	Size
1	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[0]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[0]	
2	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[10]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[10]	
3	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[11]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[11]	
4	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[12]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[12]	
5	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[13]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[13]	
6	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[14]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[14]	
7	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[15]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[15]	
8	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[1]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[1]	
9	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[2]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[2]	
10	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[3]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[3]	
11	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[4]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[4]	
12	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[5]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[5]	
13	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[6]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[6]	
14	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[7]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[7]	
15	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[8]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[8]	
16	(Port)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1[9]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1[9]	
17	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[0]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<0>	
18	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[10]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<10>	
19	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[11]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<11>	
20	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[12]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<12>	
21	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[13]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<13>	
22	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[14]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<14>	
23	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[15]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<15>	
24	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[1]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<1>	
25	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[2]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<2>	
26	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[3]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<3>	
27	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[4]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<4>	
28	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[5]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<5>	
29	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[6]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<6>	
30	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[7]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<7>	
31	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[8]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<8>	
32	(Reg)	rtl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg[9]		impl:/WORK/cl_hold_reg_v1/hold_reg_v1_reg<9>	

Figure 6: **Passing Points Report in Formality**

Verification of Designs Containing Xilinx CoreGEN Components

To assist completion of FPGA designs, Xilinx provides designers with IP of varying complexity. This IP is provided with the CoreGEN tool, part of the Xilinx ISE software package. However, since CoreGEN IP is provided as an EDIF netlist rather than as synthesizable Verilog code, a few steps are required to add the Xilinx CoreGEN macros into the Golden RTL design for checking in Conformal LEC. To convert the IP into a format acceptable to Conformal, the netlist must be run through the Xilinx NGDBUILD and NGD2VER tools and then processed by the xilinx2verplex.pl utility. Xilinx provides a "core2formal.pl" PERL script to run the commands necessary.

This PERL script is available in $XILINX/verilog/bin/<platform>/core2formal.pl.

To run these commands, you must set up the Xilinx environment.

The command is as follows:

```
>xilperl $XILINX/verilog/bin/<platform>/core2formal.pl -<vendor> -<family>
<coregen_module>.edn
```

Notes:

1. For Conformal LEC, the <vendor> option must be "verplex".
2. The <family> option can be virtex, virtexe, virtex2, and spartan2.
   - <platform> can be "sol" for solaris UNIX workstation, "hp" for HP UNIX workstation, or "nt" for PC platform.

The PERL script runs the following commands:

```
ngdbuild –p <family> <coregen_module>.edn
ngd2ver –r –w <coregen_module>.ngd <coregen_module>_ngd.v
xilperl xilinx2verplex.pl <coregen_module>_ngd.v > <coregen_module>_for.v
```

Known Issues

Known issues with the Formal Verification flow using Xilinx designs and Formality are listed below. Known issues reported after the date of this publication are documented in Xilinx Answer Records, available at http://support.xilinx.com on the web.

1. Verification of RAM resources inferred by the synthesis tools is not supported by Formality. This is because inferred components make it difficult for formal verification tools to find appropriate compare points in the designs.
2. Verification with retiming turned on in synthesis is not supported by Formality. Synthesis tools change and move around logic during retiming, and this causes difficulty for formal verification tools attempting to find appropriate compare points between designs. If retiming is turned on, some points do not compare successfully during formal verification.
3. Designs using distributed SelectRAM resources have difficulty matching to back-end designs, because the implementation tools break up the SelectRAM instances into smaller components, which makes it difficult for formality to compare the RTL and Implemented versions of SelectRAM resources. The recommendation is either to not use SelectRAM resources, or to use the small RAM16x1 SelectRAM primitive in the RTL design, as shown in Figure 7.

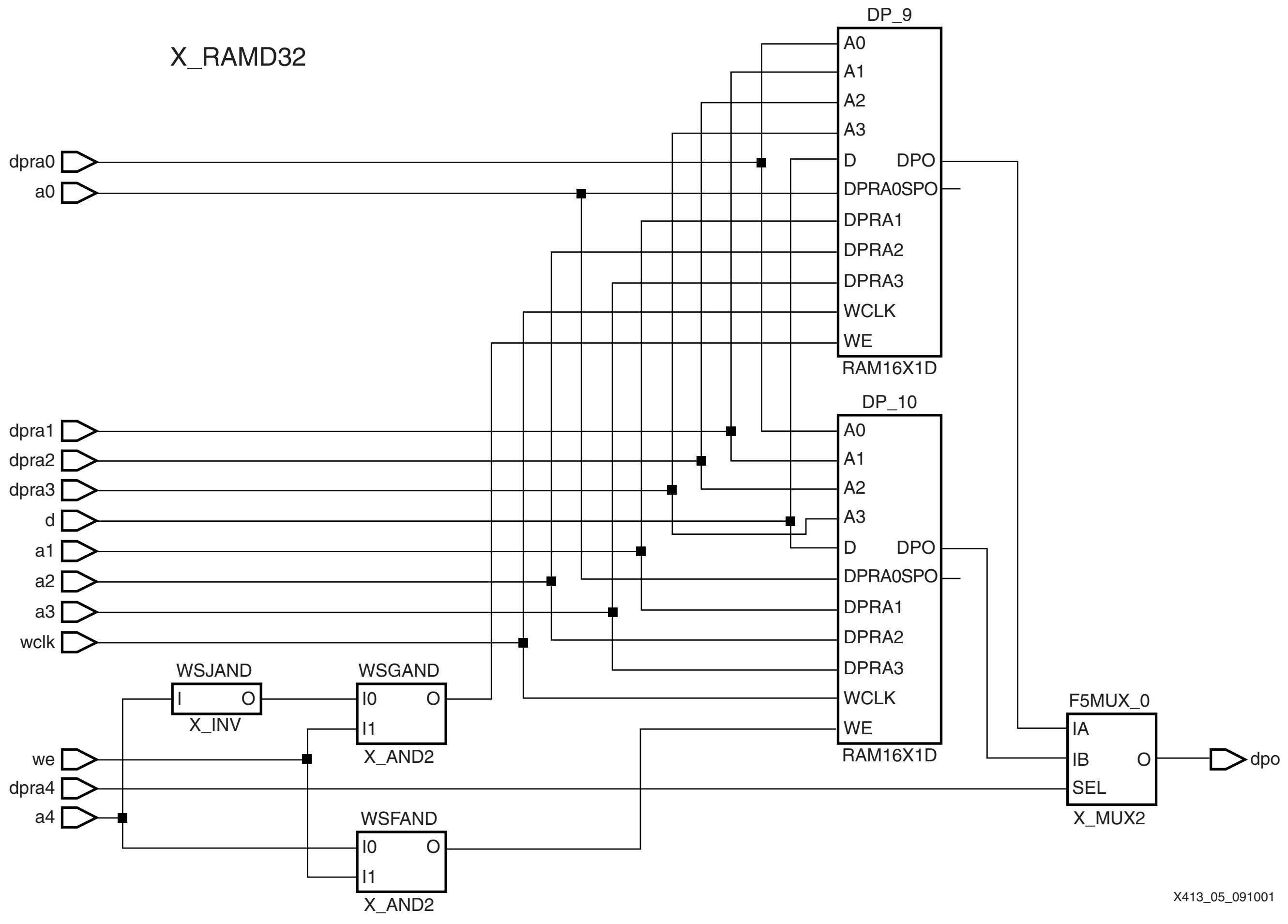
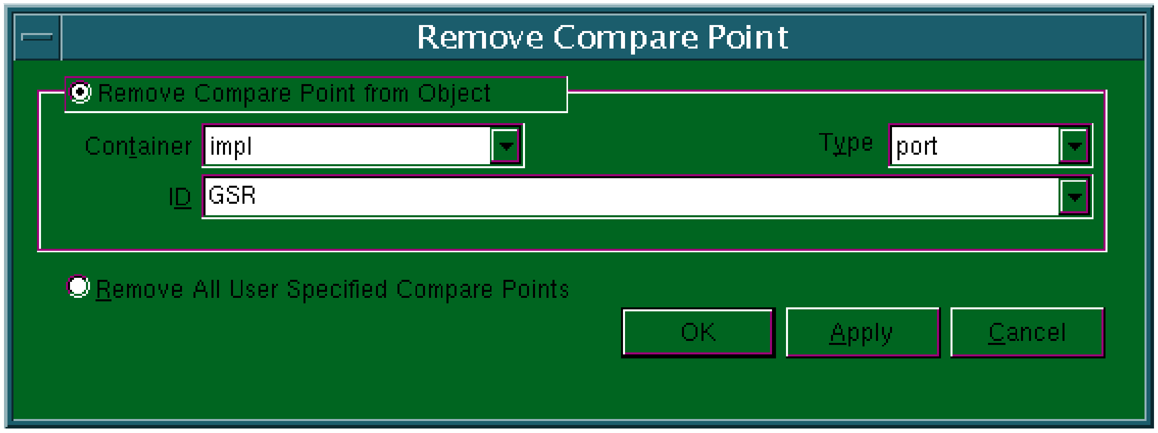


Figure 7: **32-bit RAM Composed of Smaller Primitives**

4. If the option is selected to bring GSR or GTS as a port in the back-annotated design, then the compare points for these ports must be removed; otherwise, they are reported as unmatched points by Formality. This can be accomplished via the Formality GUI by selecting Setup > Compare Points > Remove in the pull-down menu, as shown in Figure 8.


Remove Compare Point

[✓ Remove Compare Point from Object]

Container: impl

Type: port

ID: GSR

Remove All User Specified Compare Points

Figure 8: Removing Compare Point

5. Designs that are retargeted from one device to another without resynthesizing the design can cause errors in Formality. Since the size and width of block RAM resources are different in different devices, it is recommended to resynthesize to the new device when retargeting a design. If the retargeting is done only at the back-end, some components are mismatched in formal verification, since they do not match the components used in the RTL design.

6. If the design instantiates the FDDRRSE or FDDRCPE components for dual-data-rate ports, the SIMPRIMS component, X_MUXDDR, must be renamed to match the component used in the front-end. Without doing this, these points result in mis-compares in Formality.

7. If the "-bp" option is used in Xilinx Map during Implementation, Formal verification does not work correctly. The "-bp" switch pushes logic into unused block RAM areas, but these changes makes logic equivalency checking impossible for the parts of the design pushed inside block RAM areas, because Formality cannot see inside block RAM resources.

8. If the synthesis tool merges registers during the optimization process, this causes errors during formal verification, because some compare points are missing for Formality. This problem can be resolved by using a script called "makeconstraints," provided by Synopsys, that generates a constraints file to inform Formality of the registers that were merged during Synthesis by FPGA Compiler II. Note that this script works only with FPGA Compiler II. This script is provided in text format in the **Formality Run Scripts** section.

9. Map inserts a pull-up for internal 3-state buffers. This causes some uncompared points in formal verification. These can be worked around in these ways:
   - Manually instruct formality to not compare these points.
   - Remove the pull-up instantiations manually from the back-end netlist.
   - Add pull-ups to internal 3-state buffers in the RTL design.

Formality Run Scripts

Formality can be run using either the GUI or command line scripts. This section provides two command line scripts.

Formality Flow Script

This script lays out the basic formality verification flow and runs through most of the steps outlined in the **Verifying Design** section.

```
create_cont rtl
create_cont post_syn
set search_path ". ./RTL $XILINX/verilog/formality/simprims
$XILINX/verilog/formality/unisims"

set hdlin_ignore_full_case false
set hdlin_ignore_parallel_case false
set hdlin_error_on_mismatch_message false

read_ver -c rtl [glob ./RTL/*.v]
source $XILINX/verilog/formality/unisims/unisims.fms
set_ref rtl:/WORK/<design_dir>
link $ref

current_cont post_syn
source $XILINX/verilog/formality/simprims/simprims.fms

read_verilog -container post_syn ./<design>.v.MOD
set_imp post_syn:/WORK/<design_dir>
link $impl

#set name_match_allow_subset_match true
set signature_analysis_matching true
set_compare_rule $impl -from {/f1} -to {}

#current_design $ref
#current_design $impl

source ./remove_cp_impl.fms
source ./set_constraints.fms

verify -no

report_failing_points > failing_points
```

Workaround for Register Merging

This script provides the workaround for register merging by FPGA Compiler II. More details on this issue are provided in the **Known Issues** section.

```
#!/bin/sh
fc2_report=$1
tmpfile=${fc2_report}$$
# grep and get all of the Duplicate cells merges messages
# get just the two paths
# remove the single quotes

grep "Warning: Duplicate cells .*merged." ${fc2_report} | awk '{printf("%s
%s\n",$4,$6) }' | \
sed -e "s/\'/ /g"  > ${tmpfile}

# get the design name
designName=`cat  ${tmpfile} | awk -F/ '{print $2}' | uniq | head -1`


# remove the /designname
# change <> to []
# print the set_constraint commands
cat  ${tmpfile} | sed -e "s/\/${designName}//g" | sed -e "s/</[/g" | sed -e "s/>/]/g" |    \
awk '{printf("set_constraint coupled \" [ file tail %s ] [ file tail  %s] \" \
[ file dirname [ tr $ref%s ]] \n",$1,$2,$1)}' > ${fc2_report}.constraints


if [ -f ${tmpfile} ] ; then
       rm ${tmpfile}
fi
```

Support Information

For additional support on the Xilinx/Formality flow, contact Synopsys customer support. The contact information is:

Email: Support_center@synopsys.com

Phone: (650) 584-4200
(800) 245-8005

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
09/18/01	1.0	Initial Xilinx release.
10/02/01	1.1	Changed Support Information.