XAPP414 (v1.3) January 21, 2002

Xilinx/Synopsys Formality Verification Flow

Author: 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, and checks the other design netlists (post-synthesis, post-implementation) against that. 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.

Sofware and Device Support

The formal verification flow between Xilinx and Synopsys Formality is supported with the following software versions:

- Xilinx Software: ISE 4.1I and later.
- Synopsys Software: FPGA Compiler II version 3.6 and newer, and Formality version 2001.06 and newer.
- Platform Support: Solaris.

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 and Formality is shown in Figure 1.

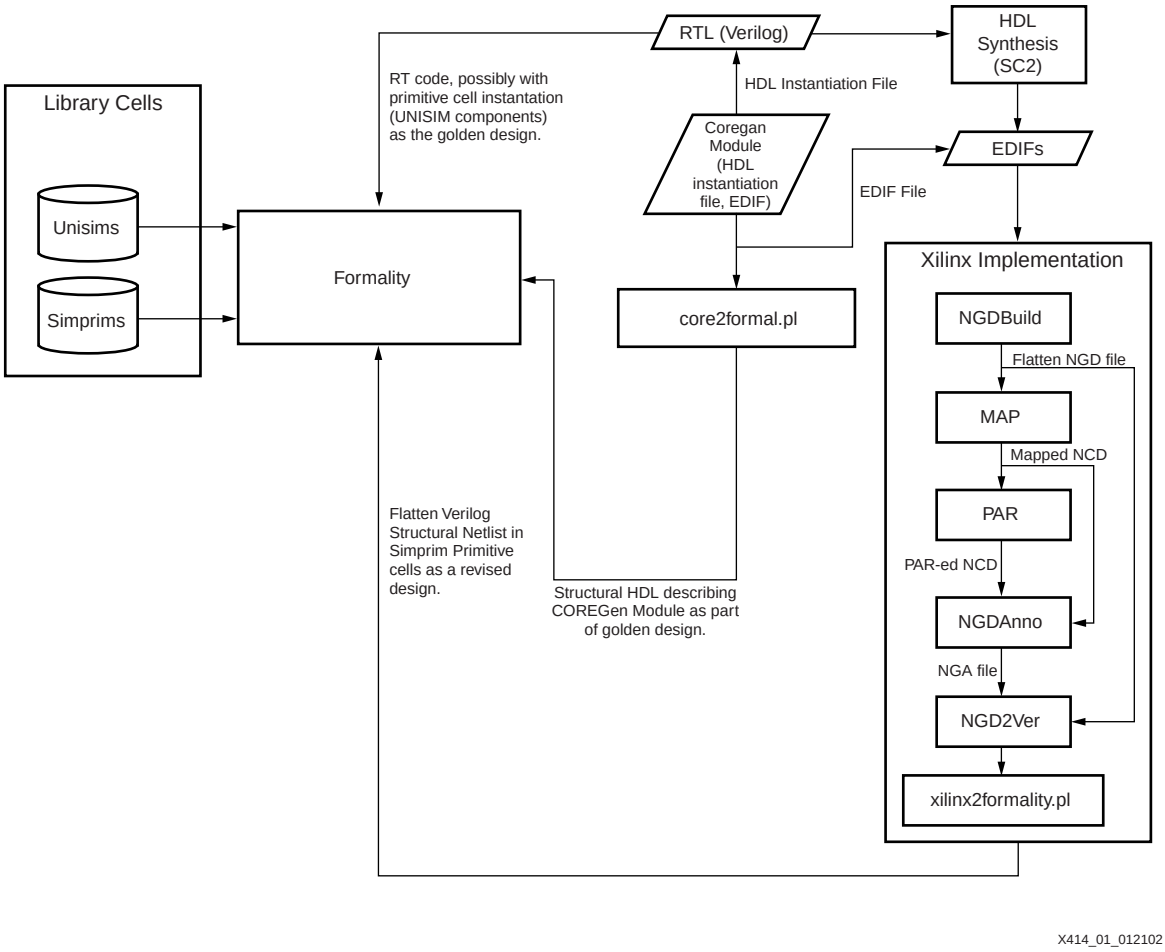


Figure 1: **Xilinx/Formality Formal Verification Flow**

Sample Flows

Below are examples of two sample flows that can be run using Xilinx and Formality. The first compares the logic equivalency between the RTL (Pre-synthesis) and the Post-NGDBUILD designs, and the second 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 comprises of the following steps:

1. Synthesize the Verilog design files with Synopsys FPGA Compiler II targeting a Xilinx FPGA (Virtex/Virtex-E/Virtex-II/Spartan-II). An EDIF netlist file will be produced at the end of this step.
2. Create the Post-NGDBuild Verilog netlist using 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** above.

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

Note: Refer to the ISE documentation available in the ISE Quick Start Guide (found in the ISE 4.1I software box) or the Xilinx support site at http://support.xilinx.com for more information on running ISE.

From the MS-DOS window (PC) or the 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 will output the same file name as input file.

3. At the MS-DOS or 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.Xilinx2formality.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. Reference **Verification of Designs containing Xilinx CoreGEN components** for more information.

5. Run the Formality flow to compare the two versions of the Verilog netlists. See 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 FPGA (Virtex/Virtex-E/Virtex-II/Spartan-II). An EDIF netlist file will be 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 will run NGDBuild, MAP, PAR, and NGDANNO and NGD2Ver to create Post-PAR Verilog netlist.

From the MS-DOS window (PC) or 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 Post-PAR Verilog file

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

3. At the MS-DOS or 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.Xilinx2formality.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. Reference **Verification of Designs containing Xilinx CoreGEN components** for more information.

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

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 install. The variables that are required 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 setup file, ".synopsys_fm.setup" file in the project directory by copying the template from $XILINX/verilog/formality/template.synopsys_fm.setup and renaming to ".synopsys_fm.setup". For more information on the use and customization possibilities of the ".synopsys_fm.setup" file, contact Synopsys.

Setting up the Xilinx Verification Libraries

There are two Xilinx Verification libraries that need to be used along with Formality for Formal Verification. These 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 the DCM or BlockRAM.
- **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 subsequent section.

Verifying Design

All formality commands can be run either from the GUI or the Unix terminal prompt. We will step through the flow using the GUI, however, these commands can be added into a script that can be launched from the 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. These comprise of the following three lines:

   ```
   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 to the correct location of the Xilinx install directory.

2. Start up the GUI by typing "formality" at the Unix terminal prompt. The Formality GUI that comes up is shown in Figure 2.

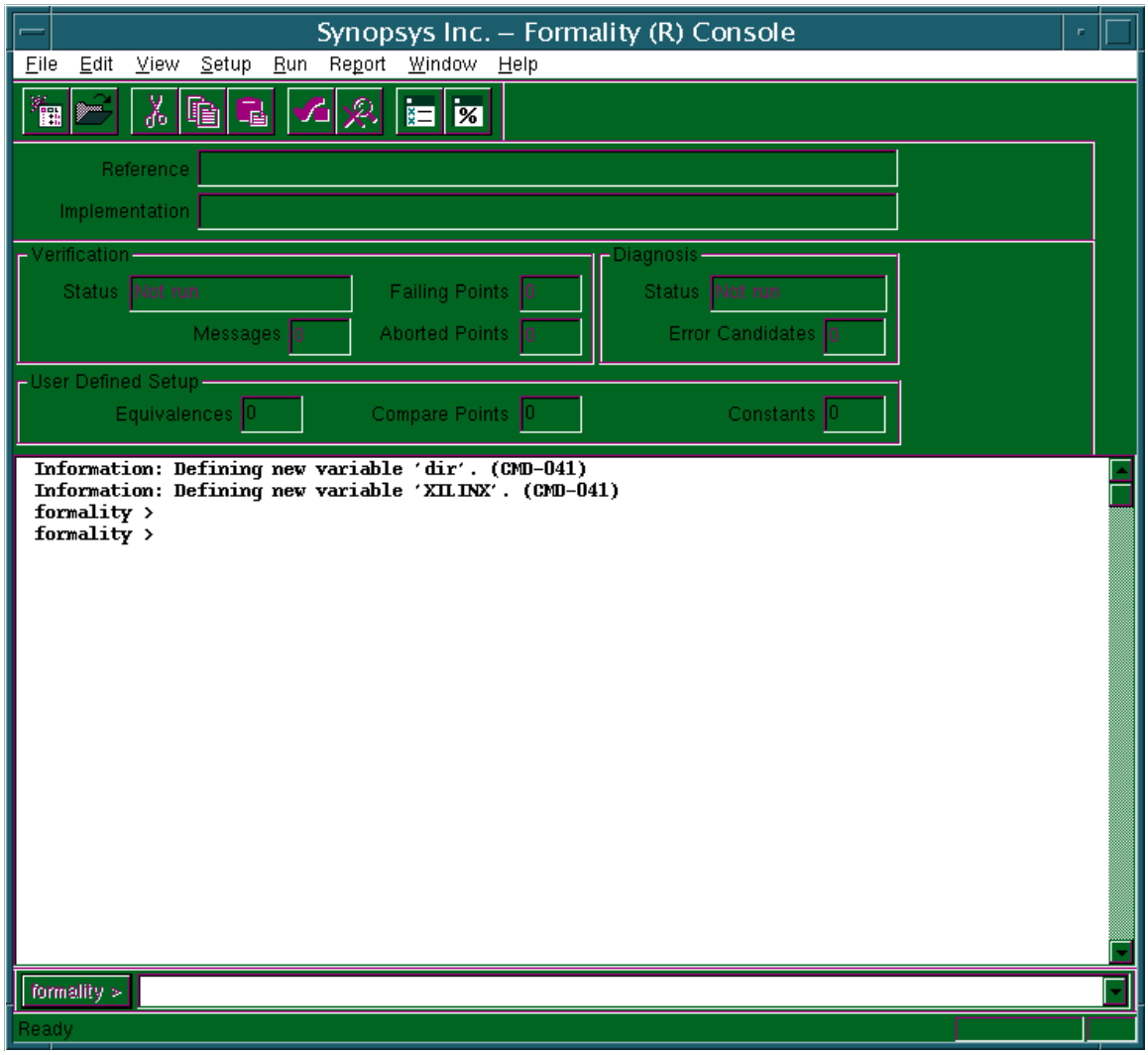


Figure 2: **Formality Main GUI**

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, need to be read into Formality before comparing the designs. Enter the following commands at the formality command prompt to read the libraries:

   set XILINX /path/to/xilinx/install/

   source $XILINX/verilog/formality/unisims/unisims.fms

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

   Formality will read the contents of the Xilinx verification libraries. This may take upto 10 minutes, depending on the network connection speed and processor speed.

4. Select File-> New container.

5. Enter rtl on container name. We will use this container to store the RTL (reference) design. The RTL container GUI will open up, as shown in Figure 3.

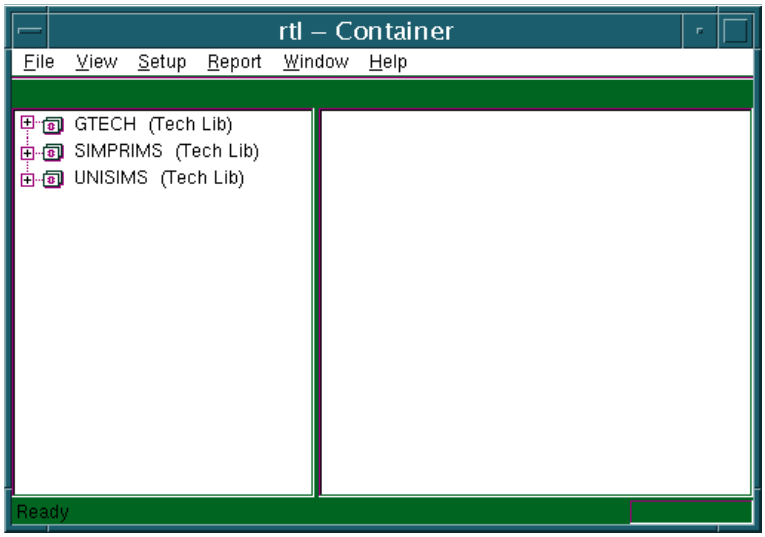


Figure 3: **RTL Container GUI**

6. In the rtl - Container window, select File > Read Design from the pull-down menu..
7. Select all 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. We will add the CoreGEN modules at a later time. This is because the coregen modules are already instantiated as a black box in your design. In order to read the description of coregen modules, you'll need to override the black box component. Otherwise, the following error will occur:

   > Error: You are declaring a module 'tenths' which is already declared.
   > (File:<coregen_module>.v)

8. Select Open on the "Read Design" browser window (Figure 4) .

9. Click OK on the next window to accept all the default settings.

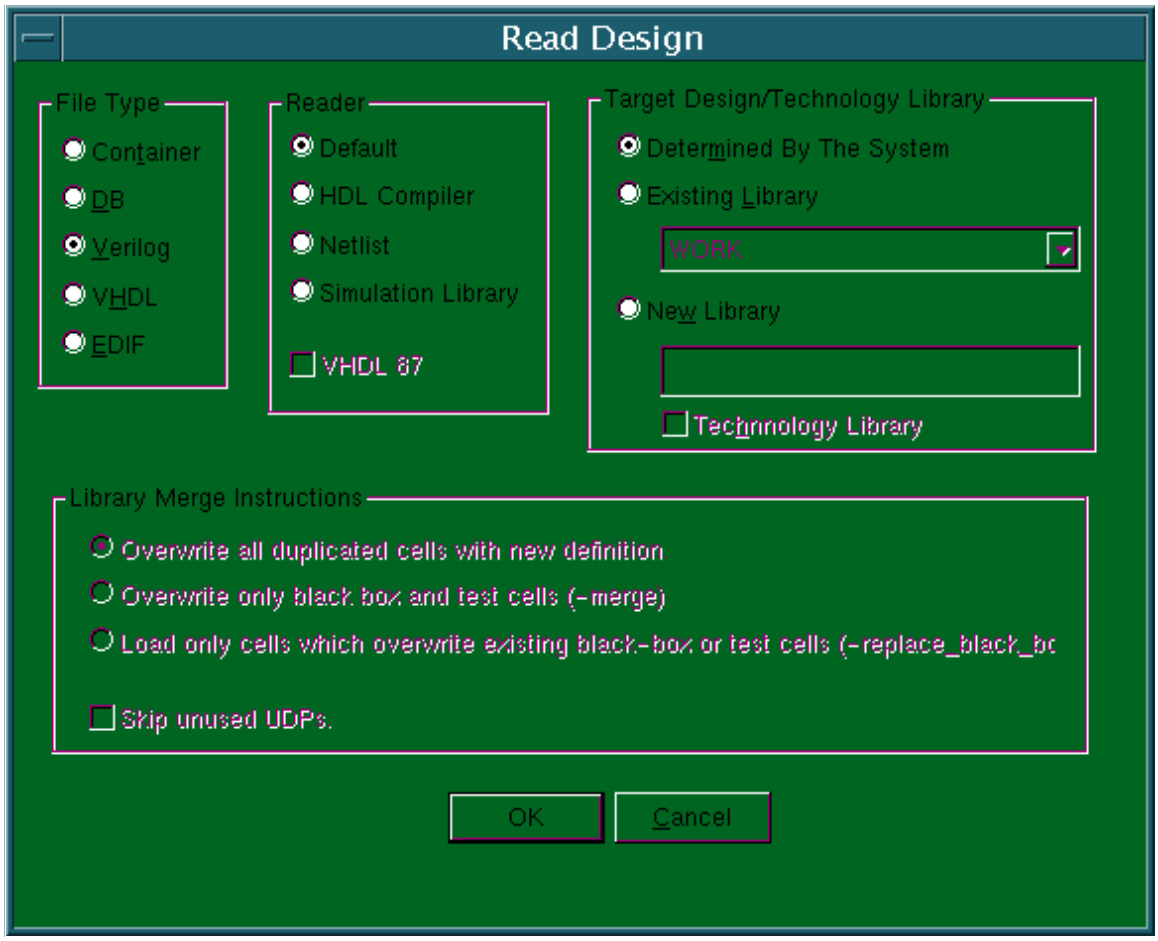


Figure 4: **Read Design Options GUI**

10. Click on the + sign next to WORK and select the top-level design module.
11. Select the top-level module, Right-Click on that module and select "Set as Reference."
12. Select File > Link Design > Use Default Options from the rtl-container's pull-down menu.
13. Repeat steps 3 to 5 to create a container called "imp." This container will be used to store the implemented design.
14. In the imp-Container window, select File > Read Design from the pull-down menu.
15. Select the implemented Verilog file. This can be the after NGDBUILD or after PAR. Ensure that this file has been processed through the xilinx2formality.pl script. See section **Sample Flows** for more details.
16. Repeat steps 8 to 10 for the "imp" container window.
17. Right Click on the toplevel.v and select "Set as Implementation."
18. Select File > Link Design > use Default Options from the imp-container pull-down menu.
19. To include any CoreGEN modules to the reference design, refer to the section **Verification of Designs containing Xilinx CoreGEN components**.

20. From the Formality Console window, click Run -> Verify -> All Compare Points. If the designs compare successfully, the result will be shown in the GUI under the "Verification" section, as shown in Figure 5.

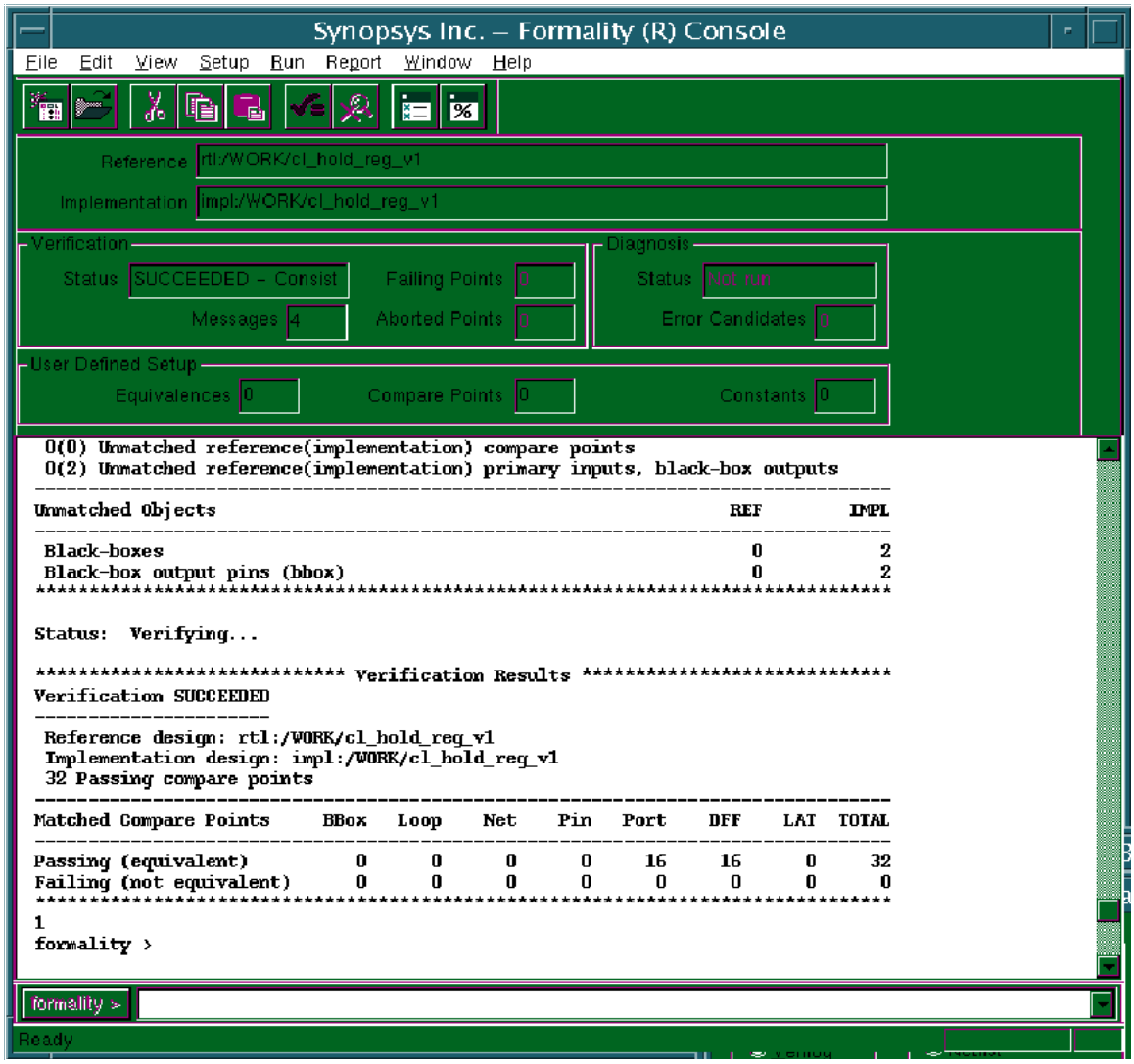
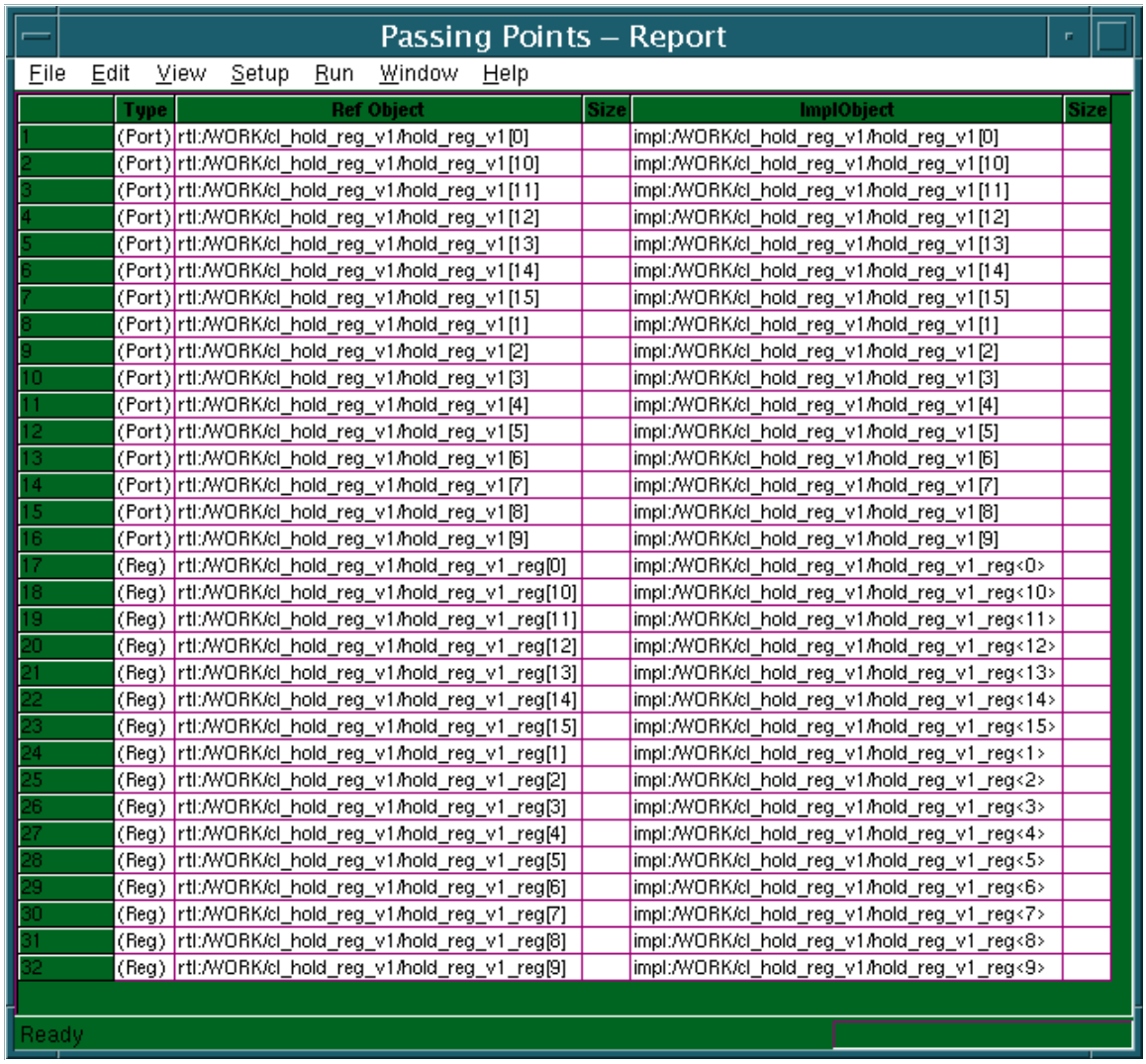


Figure 5: **Formality Transcript Showing Results of Verification**

Once verification has completed, several reports can be generated to see passing and failing points. All 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

Xilinx provides IP of varying complexity to designers to assist in the completion of Xilinx FPGA designs. This IP is provided with the CoreGEN tool, part of the Xilinx ISE software package. However, since the CoreGEN IP is not provided in synthesizable Verilog but as a EDIF netlist, a few extra steps are required to add the Xilinx CoreGEN macros into the Golden RTL design for checking in Formality. The netlist needs to be run through the Xilinx NGDBUILD and NGD2VER tools, and then processed through the xilinx2formality.pl utility to convert it into a format acceptable to Formality. Xilinx provides a PERL script "core2formal.pl" to run all the commands necessary.

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

In order to run these commands, the Xilinx environment must be setup.

The command is as follows:

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

Notes:

1.For Synopsys Formality, the <vendor> option must be "formality".

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 will run the following commands:

ngdbuild –p <family> <coregen_module>.edn

ngd2ver –r –w <coregen_module>.ngd <coregen_module>_ngd.v

xilperl xilinx2formality.pl <coregen_module>_ngd.v > <coregen_module>_for.v

Known Issues

There are some known issues with the Formal Verification flow using Xilinx and Formality. They are listed below. Any known issues reported in the future are going to be documented in Xilinx Answer Records, which can be found from http://support.xilinx.com.

1. Verification of RAM's 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 re-timing turned on in synthesis is not supported by Formality. Synthesis tools change and move around logic during re-timing, and this causes difficulty for formal verification tools to find appropriate compare points between designs. If re-timing is turned on, there will be some points during formal verification that will not compare successfully.
3. Designs using distributed SelectRAM will have difficulty matching to back-end designs. This is 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 SelectRAMs. The recommendation is either to not use SelectRAMs, or to use the small SelectRAM primitive, RAM16x1, in the RTL design, as shown in Figure 7.

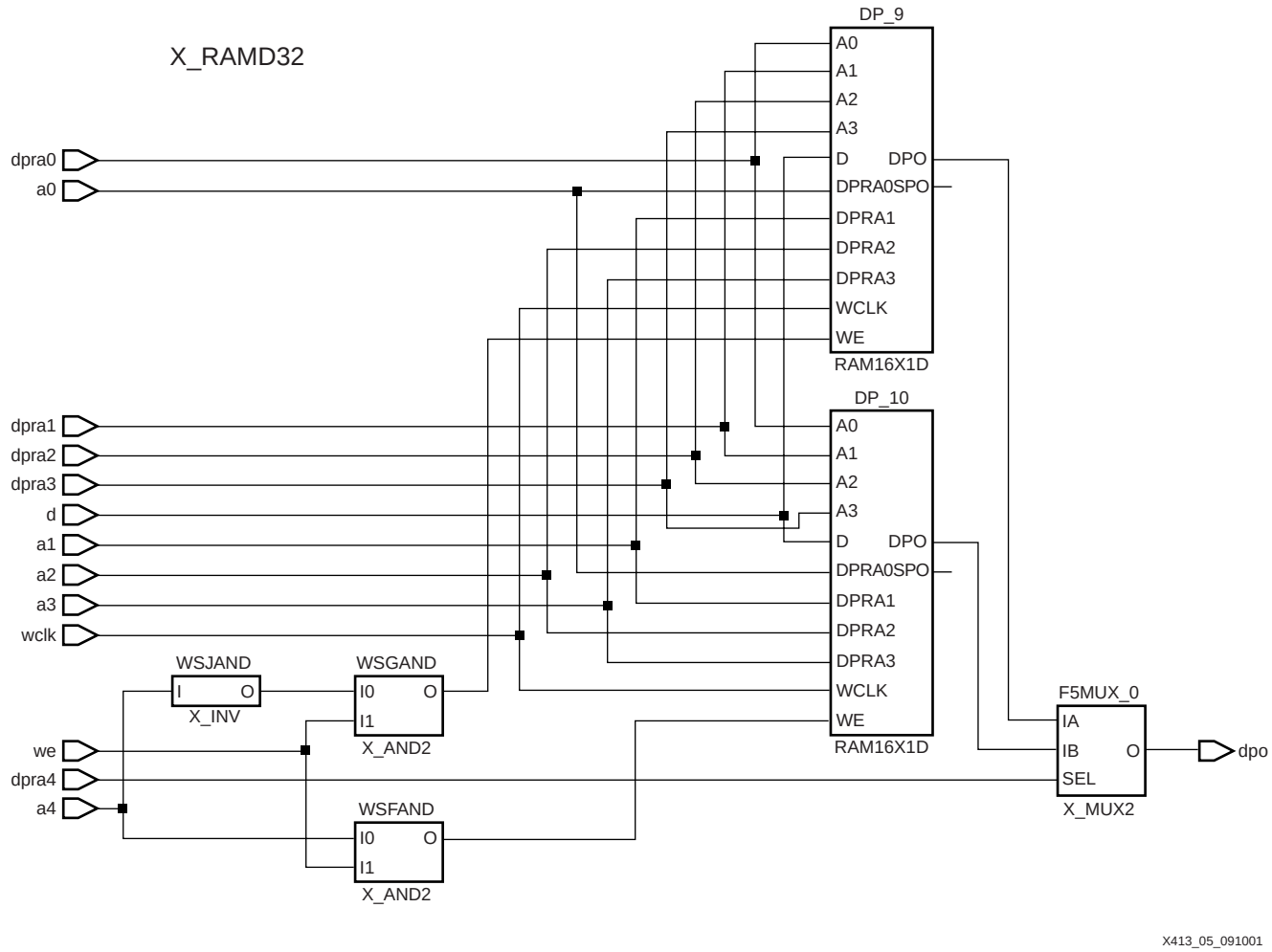


Figure 7: **32-bit RAM Composed of Smaller Primitives**

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 will be 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.

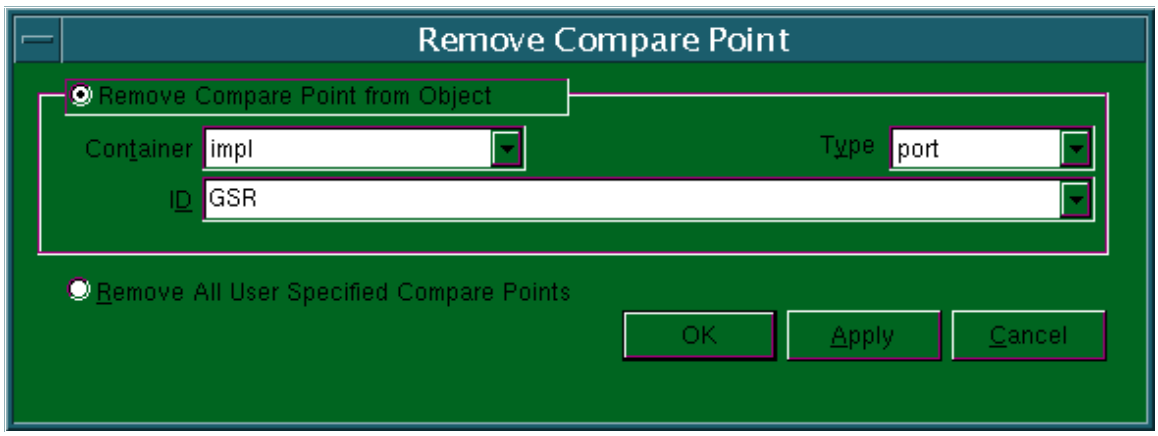


Figure 8: Removing Compare Point

5. Designs that are re-targeted from one device to another without re-synthesizing the design can cause errors in Formality. Since the size and width of the BlockRAM's is different in different devices, it is recommended to re-synthesize to the new device when re-targeting a design. If the re-targeting is done only at the back-end, some components will be mis-matched in formal verification, since they will 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, will need to be renamed to match the component used in the front-end. Without doing this, these points will result in mis-compares in Formality.
7. If the "-bp" option is used in Xilinx Map during Implementation, Formal verification will not work correctly. The "-bp" switch pushes logic into unused BlockRAM areas, but these changes makes logic equivalency checking impossible for the parts of the design pushed inside the BlockRAM's as Formality cannot see inside the BlockRAM.
8. If the synthesis tool merges registers during the optimization process, this causes errors during formal verification as 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 only works with FPGA Compiler II. This script is provided in text format in the following section.
9. Map inserts a pull-up for internal tri-state buffers. This will cause some uncompared points in formal verification. These can be worked around by either:
   - Manually instructing formality to not compare these points.
   - Remove the pull-up instantiations manually from the back-end netlist.
   - Add pull-ups to internal tri-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 scripts that can be used at the command line.

Formality Flow Script

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

```
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 Known Issue number 12 in this document.

```
#!/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: Suppor_center@synopsys.com

Phone: (650) 546-4200

(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.
09/28/01	1.1	Changed Support Information.
01/17/02	1.2	Fixed typographical errors.
01/21/02	1.3	Updated Figure 1.