XAPP413 (v1.1) October 2, 2001

Xilinx/Verplex Conformal Verification Flow

Authors: Mujtaba Hamid and Yenni Totong

Summary

This application note covers the logic equivalency flow using Xilinx ISE software with Verplex Conformal LEC. 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 checking logic equivalency, formal verification is quickly gaining acceptance by designers creating multi-million gate designs, because of its accuracy and speed. Using Conformal (previously known as Tuxedo) LEC 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 golden (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 golden design. However, the RTL (pre-synthesis) netlist is most commonly used as the reference. The Xilinx/Conformal formal verification flow currently supports only the Verilog language.

Software and Device Support

The formal verification flow between Xilinx designs and Verplex Conformal LEC is supported by the following software:

- Xilinx Software: ISE Alliance 4.1i (UNIX version only) and later
- Verplex Software: Conformal LEC version 2.1.1.a 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/Verplex 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 Conformal LEC is shown in Figure 1.

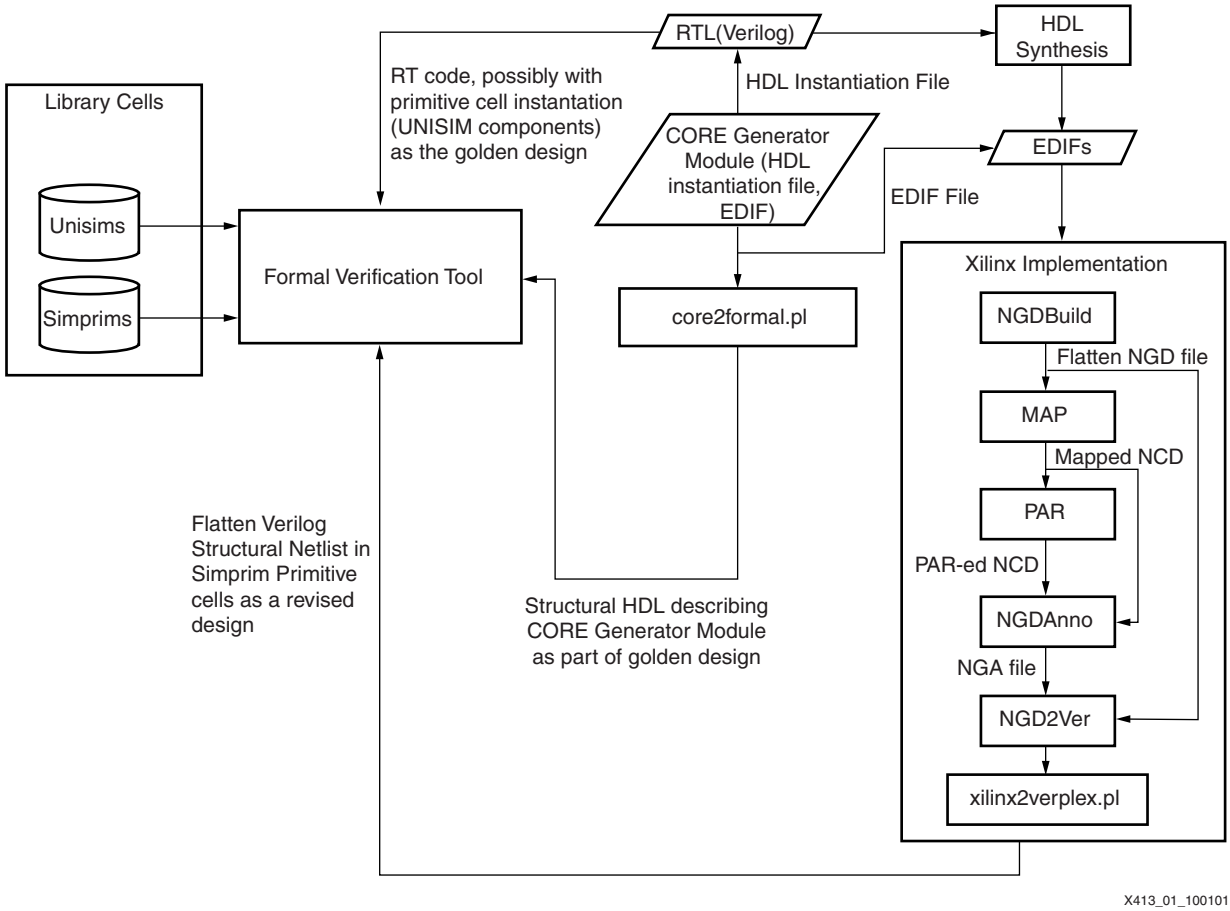

Figure 1: **Xilinx/Conformal Formal Verification Flow**

Sample Flows

Below are two sample flows that can be run using Xilinx designs and Conformal LEC. 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 Conformal is a logic equivalency checker.

RTL vs. Post-NGDBuild

Golden design: Behavioral RTL Verilog (pre-synthesis)

Revised design: Post-NGDBuild Verilog

The flow is comprised of the following steps:

1. Synthesize the Verilog design files with your synthesis tool, 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 code, using the Xilinx implementation tool from the GUI or the command line.

   From the GUI:

   a. Create a Xilinx ISE Project using the EDIF netlists 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 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 outputs the same filename as the input file.

3. From the UNIX terminal window, run the "xilinx2verplex" command:

```
>xilperl  $XILINX/verilog/bin/<platform>/xilinx2verplex.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 "golden" description for the module. Refer to the **Verification of Designs Containing Xilinx CORE Generator Components** section for more information.

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

RTL vs. Post-PAR

Golden design : Behavioral RTL Verilog (pre-synthesis)

Revised design: Post-PAR Verilog

The flow is as follows:

1. Synthesize the Verilog design files with your synthesis tool, 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 "xilinx2verplex" command:

>xilperl $XILINX/verilog/bin/<platform>/xilinx2verplex.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 "golden" description for the module. Refer to the **Verification of Designs Containing Xilinx CORE Generator Components** section for more information.
5. Run the Conformal flow to compare the two versions of the Verilog netlists. Refer to the **Conformal LEC Flow** section.

Conformal LEC Flow

This section briefly steps through the Conformal-LEC flow. For more details on this flow, contact Verplex customer support.

All the commands in this section can either be entered at the Conformal LEC command prompt in the GUI, or compiled into a command (.DO) file that can be executed from Conformal.

1. Launch Conformal LEC at the command prompt by typing "lec" to start the GUI, shown in Figure 2. Commands are entered in this GUI at the bottom, in the window labeled "SETUP >".
2. Read the golden design and the design that needs to be verified, as follows:

   read design <top>.v <lower_level>.v -f verilog.vc -verilog -golden -replace

   read design <the_revised_version>.v -f verilog.vc -verilog -revised –replace

   Note: If a CORE Generator module is instantiated in your design, refer to the **Verification of Designs Containing Xilinx CORE Generator Components** section to create a "golden" description of the module.
3. Tie the GSR and GTS pins in the Xilinx design netlists to known values:

   add tied signal 0 glbl.GSR -rev

   add tied signal 0 glbl.GTS -rev
4. Finalize the setup of the design before proceeding with formal verification:

   set flatten model -seq_constant

   set system mode lec
5. Instruct Conformal LEC to check for all compared points between the two designs:

   add compared points -all
6. Proceed with comparing the two designs for logic equivalency:

   compare

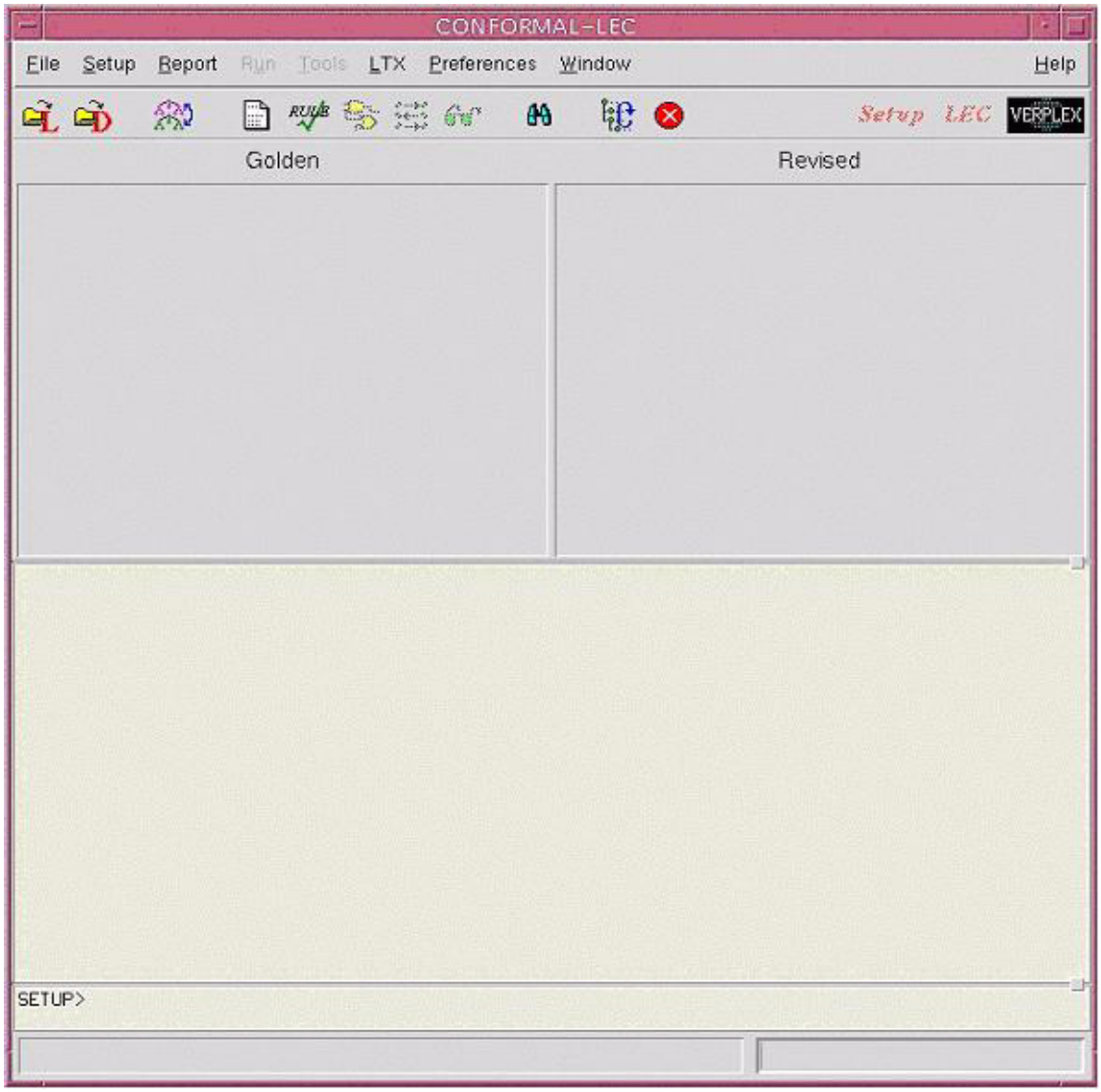

Figure 2: **Conformal LEC GUI After Launch.**

Conformal LEC runs the comparison on the two designs, and displays the report on the bottom half of the screen, as shown in Figure 3.

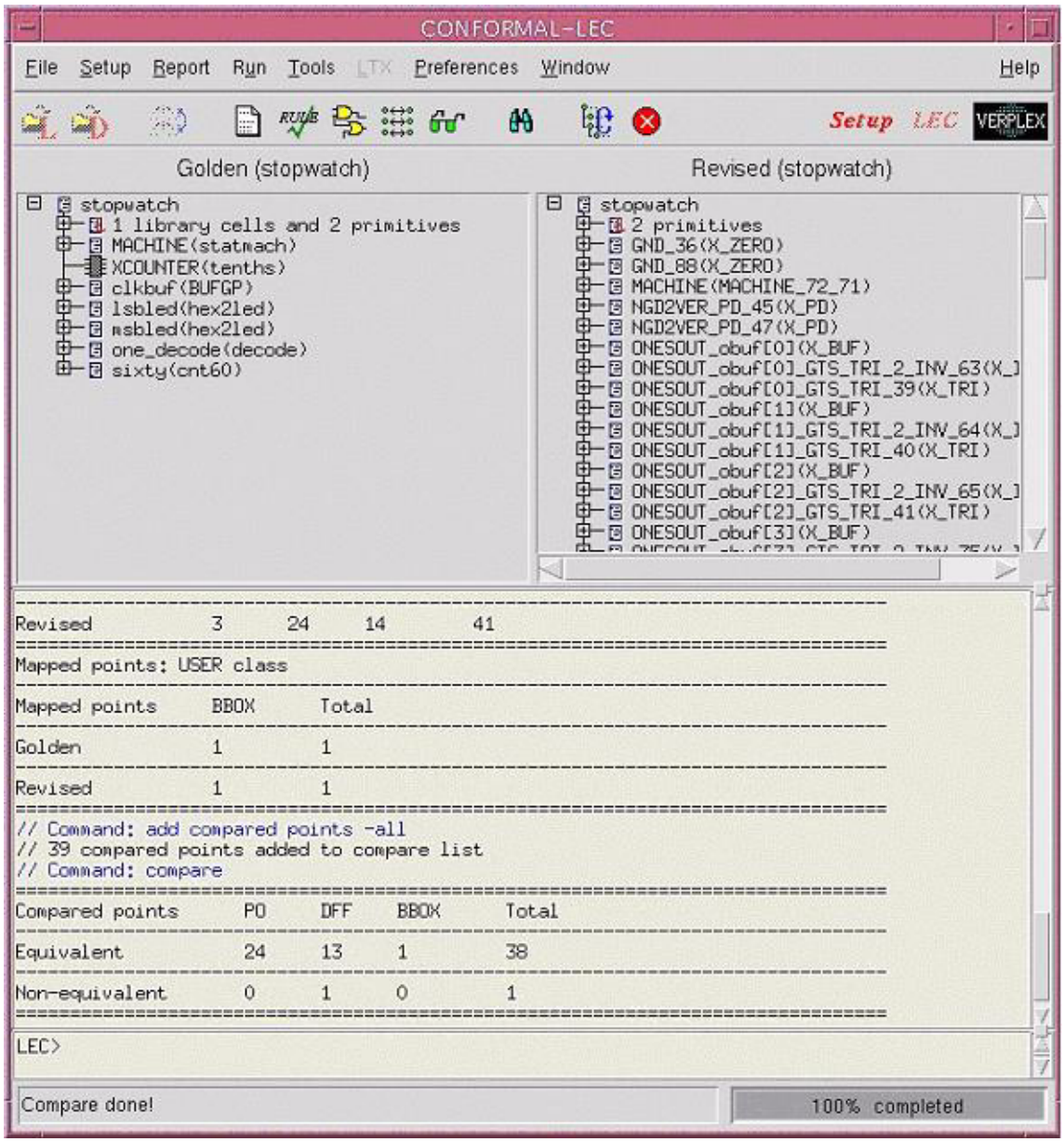

Figure 3: **Conformal LEC Main GUI Showing the Results of the Design Comparison**

Once Conformal has completed the equivalency checking on the designs, more details on the results can be obtained. As shown in Figure 4, the "Mapping Manager" window, which you can launch by selecting Tools > Mapping Manager from the Conformal LEC main GUI, is broken up into three sections: unmapped points, mapped points, and compared points. The unmapped points have a red circle next to them, whereas the mapped points do not.

Additionally, the GUI is broken up into two columns: one for the golden design and one for the compared design. The mapping manager thus visually shows the points between two designs that do or do not match. Additionally, it shows all points that were compared between the two designs. You can deselect points that do not need to be compared.

You can also create a number of other reports, based on results obtained from the design comparison. These are available under the "Report" pull-down menu in the main Conformal LEC GUI.

Commands can also be compiled into a command file (.DO file). You can execute this script from the Conformal LEC GUI by selecting the "File > Do Dofile" option from the pull-down menu. A sample command file is shown in the **Conformal Command Files** section.

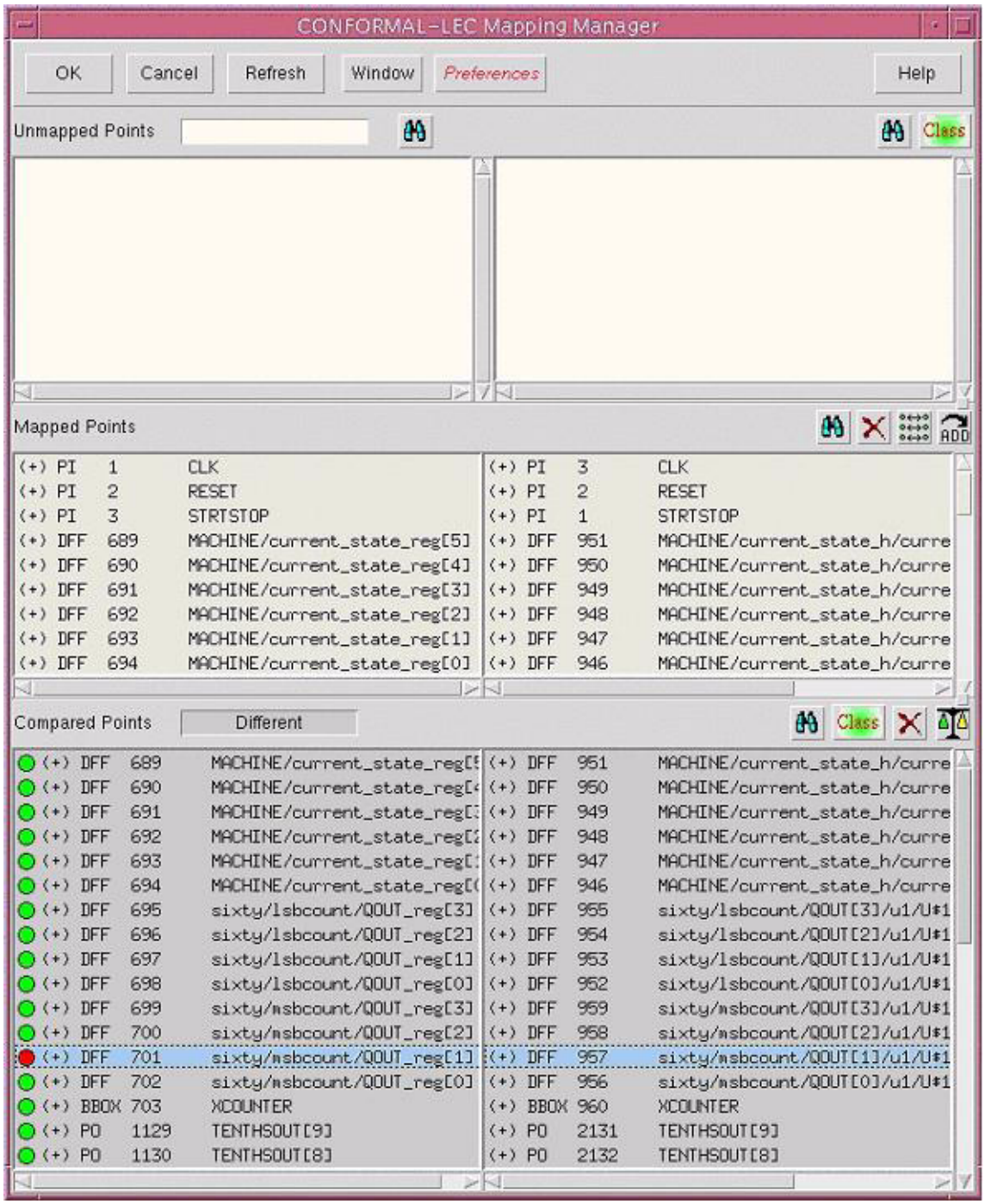

Figure 4: **Conformal LEC Mapping Manager**

Verification of Designs Containing Xilinx CORE Generator Components

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

The location of the PERL script is: $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 Conformal LEC are listed below:

1. Verification of RAM resources inferred by the synthesis tools is not supported by Conformal LEC. 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 Conformal LEC. 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. Verification returns errors if the synthesis tools use register merging to optimize logic, because this results in unmapped points between the golden and the implemented design. This can be worked around by using the following command in Conformal:

   ```
   Set flatten model -all_seq_merge
   ```

4. Designs with distributed SelectRAM+ (for example, RAM16X1D) contain an unmapped register for each RAM bit during verification. This is because the RAM16X1D is decomposed into an X_RAMS16 and X_RAMD16. The functionality of the golden vs. revised design is the same, but registers are introduced that become unmapped points. The recommendation is to code SelectRAM using the RAMD16x1 primitives, as shown in Figure 5. To work around this, mark the register from X_RAMS16 and X_RAMD16 as equivalent. Add the following line after 'read design':

   ```
   add inst equiv inst1 inst2 -revised
   ```

   where inst1 and isnt2 are the equivalent registers from the two RAM components in the revised design.

To get the name of inst1 and inst2, you must run the compare once. The additional registers show up as unmapped points in the Mapping Manager window. Note the names and match

them with the mapped registers (DFF) in the Mapped Points section bit per bit. Next, add the above workaround in your .do file or type them on the Conformal Tcl prompt. For example:

add inst equiv page1_i30_G/mem_reg[0]

page1_i30_F/mem_reg[0] -revised

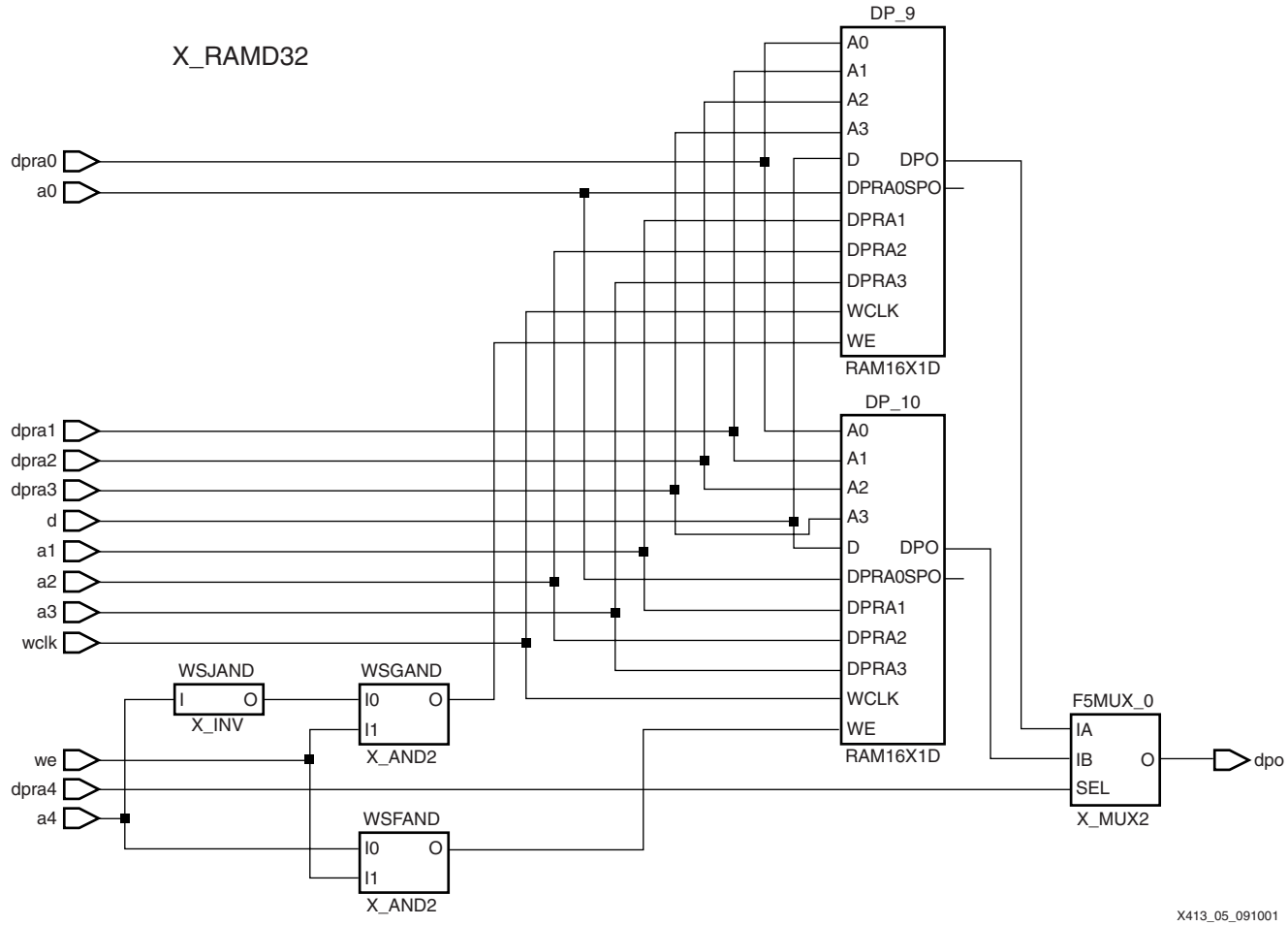

Figure 5: **32-bit RAM Composed of Smaller Primitives**

5. Designs with FDCP/FDCP_1/FDCPE/FDCPE_1/FDDRCPE. The asynchronous CLR has priority over asynchrous PRE. LEC is warning that the golden has gated PRE and the revised version does not.
6. Virtex-II designs:The RAM128X1S is converted to 8 X_RAMD16.

   set mapping effort high
7. With Virtex-II devices, RAMB4 becomes X_RAMB16 after NGDBuild.

   add tied signal 0 wr_mode[0] -net -module

   X_RAMB16_S1_INIT -revised

   add tied signal 0 wr_mode[1] -net -module

   X_RAMB16_S1_INIT -revised
8. Designs that are retargeted from one device without resynthesizing the design can cause problems. 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 might not match the components used in the RTL design.

9. If the design instantiates a FDDRRSE or FDDRCPE component for dual-data rate, the SIMPRIMS component, X_MUXDDR, must be renamed to match the component used in the front end. Without this, these points result in mis-compares. This can be worked around by using the following command in Conformal to rename the component:

   Add mapped points /<design_hierarchy>/<component> -type BBOX BBOX -module <module_name> <module_name>

   Where the <component> can be the dual-data-rate flipflop needed to be renamed, and <module_name> is the name of the design.

10. Formal verification does not work correctly if the retiming option is selected during Synthesis. In retiming, the synthesis tools readjust the logic to obtain better timing results, but these changes make logic equivalency checking impossible.
11. If the "-bp" option is used in Map during Implementation, formal verification does not work correctly. The "-bp" switch pushes logic into unused block RAM areas, but this change makes logic equivalency checking impossible, because the formal verification tool cannot see inside the block RAM.

Conformal Command Files

The following is an example of a .do file for running Conformal:

```
read design <top>.v <lower_level>.v  -f verilog.vc -verilog -golden -replace
read design <the_revised_version>.v -f verilog.vc -verilog -revised -replace
set flatten model -seq_constant

//Connect GSR and GTS to 0(GND)
add tied signal 0 glbl.GSR -rev
add tied signal 0 glbl.GTS -rev

set system mode lec
add compared points -all
compare
set system mode setup
```

Note:

The verilog.vc file contains the path to the SIMPRIMS and UNISIMS library cells. An example of this file exists in the Xilinx install directory at $XILINX/verilog/verplex/verilog.vc and is also shown below:

```
-y $XILINX/verilog/verplex/unisims
-y $XILINX/verilog/verplex/simprims
```

Support Information

For additional support on the Xilinx/Conformal LEC flow, contact Verplex customer support:

Email: Support@verplex.com

Phone: (408) 586-0300

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	Minor corrections. Replaced figures 2, 3, and 4.