Digital Logic Synthesis and Equivalence Checking Tools Tutorial

Hardware Verification Group

Department of Electrical and Computer Engineering,
Concordia University, Montreal, Canada
{n_ab, h_aridh, lahiouel}@encs.concordia.ca

CAD Tool Tutorial

May, 2016

Abstract

This document contains a brief introduction to Synopsys Design Vision, Synopsys Formality, and Cadence Conformal tools. You would need approximately three hours to finish this tutorial.

Revision History

Version	Authors	Modification	Section	Date
1.0	Naeem Abbasi	Document Creation	-	April 16, 2010
2.0	Naeem Abbasi	Section on Synopsys Formality added An additional appendix added Title modified	3 B -	April 18, 2010
2.1	Naeem Abbasi	Added Figures 1 and 2 and their descriptions	1, 2.1	April 19, 2010
3.0	Naeem Abbasi	One's counter example added (provided by Ted Obuchowicz)		May 3, 2010
3.1	Naeem Abbasi	Screen Captures Updated Remote Access Commands added to the appendix		May 3, 2010
3.2	Naeem Abbasi	Updated paths to various documentation, links and other useful information		June 8, 2011
3.3	Henda Aridhi	Updated paths to various documentation, updated section 2 GUI design vision instead of GUI design analyzer. updated all the paths to the documentation and the libraries.		April 30, 2012
3.4	Ons Lahiouel	Updated paths to various documentation, and libraries.		May 2, 2016

Table of Contents

List of Figures

1 Introduction

Logic synthesis usually refers to the process of translation of RTL design into an optimized gate level description. Logic equivalence checking refers to a technique that mathematically (i.e. with out simulation) verifies that the two design descriptions are functionally equivalent.

Section 2 of this tutorial describes how to setup and synthesize an RTL description into a structural netlist of gates using Synopsys Design Compiler. During this process, the design is read in, analyzed and elaborated. After that, the designer identifies critical nets in the design such as the clock signal. Then area and timing constraints are specified for guiding the design optimization. Once the design is optimized, it is exported to a gate level netlist. This completes the synthesis step.

Section 3 and 4 of this tutorial describe use of equivalence checking tools. An equivalence checking tool takes two descriptions of a design and verifies if they are functionally equivalent (see Fig. 1). In case if the two descriptions are not equivalent, a counter example is produced.

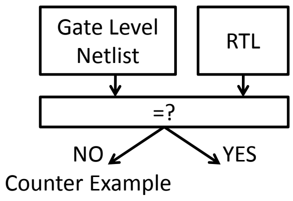


Fig. 1: Equivalence Checking

Section 3 of this tutorial describes how to check if the synthesized design is equivalent to the RTL design using Synopsys tool Formality. This new version of the tool supports SystemVerilog. The process begins by first reading in the two design descriptions. Then those ports and nets are selected for which we wish to verify functional equivalence. The equivalence checker is then run which either verifies the equivalence of the two designs or helps in debugging by identifying the failing points, ports, and nets. Failing points in the reference and implemented design can be viewed side by side in a schematic browser.

Cadence Conformal suite of tools contains a tool called Logic Equivalence Checker or LEC. Section 4 of this tutorial describes how to formally verify that the synthesized design is functionally equivalent to the RTL description using LEC. This section can be skipped if one chooses to use Formality for equivalence checking. During the equivalence checking process, the two design descriptions are first read in, then key mapping points in the two design descriptions are selected, and finally the equivalence checker is run. The generated reports show if the the two designs are equivalent. Similar to Formality, Conformal LEC also has Schematic and Netlist browser features, which are extremely helpful in debugging.

Appendix A, B and C contain the source code for the examples used in this tutorial along with some other useful information regarding these tools.

1.1 Synopsys Design Compiler

Synopsys Design Compiler (DC) is a logic synthesis and design optimization tool. The synthesis and optimization steps, described in this tutorial, can be easily converted to a script, which can later be modified and run from the command line interface.

More information about Synopsys design compiler (DC) can be found in
[/CMC/tools/synopsys/syn/doc/syn/tutorial] and
[/CMC/tools/synopsys/syn/doc/syn/examples]

1.2 Synopsys Formality

Formality is an equivalence checking tool. More information about Synopsys Formality can be found in
[/CMC/tools/synopsys/fm/doc/fm/quick_ref.pdf]and
[/CMC/tools/synopsys/fm/doc/fm/user.pdf]

1.3 Cadence Conformal

Cadence Conformal Logic Equivalence Checker (LEC) is a formal logic equivalence checking tool. More information about LEC can be found in
[/CMC/tools/cadence.2011a/CONFRML/doc]

2 Synopsys Design Compiler

2.1 Introduction

Logic synthesis is a process that translates an RTL description of a circuit into an optimized netlist consisting of flipflops, latches, and logic gates. Design engineers provide HDL descriptions and various constraints and bounds on the design to synthesis tools. These constraints reflect the needs that the design must meet. For example maximum area, minimum speed and maximum power dissipation.

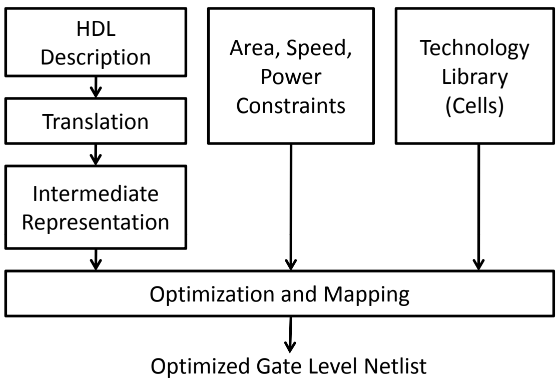

HDL Description
Translation
Intermediate Representation
Area, Speed, Power Constraints
Technology Library (Cells)
Optimization and Mapping
Optimized Gate Level Netlist

Fig. 2: Digital logic synthesis

Figure 2 shows a typical high level flow used in most logic synthesis tools. Logic synthesis is usually done in three steps. First the RTL description is translated to an un optimized boolean description. HDL Constructs such as IF, CASE, LOOPs and conditional assignments are converted to their equivalent boolean equivalents consisting of primitive gates such as NAND and NOR gates, flip flops and latches etc.

Such descriptions are functionally correct but are completely un optimized. This step is followed by the logical optimization step. The boolean optimization algorithms are used to produce an optimized equivalent description. These algorithms utilize logic flattening and logic factoring operations together with fan out and loading constraints to optimize the logic. During flattening operations remove structure while the factoring operations introduce new structure. These operations when applied in conjunction with each other help optimize the logic.

Finally, the optimized boolean equivalent description is mapped to actual logic gates by making use of a technology library of the target process. This step uses logical and timing information from the technology library to build a netlist. The generated netlist meets the area and speed needs of the user.

At the end of this process several techniques are used to ensure that the optimized netlist is functionally equivalent to the RTL design and also does not violate any of the rules of the technology.

2.2 Design example used in this tutorial

In this tutorial, we will utilize the following design example. This design implements a purely combinational circuit that counts the number of 1s in the 4 bit input vector(ones.vhd).

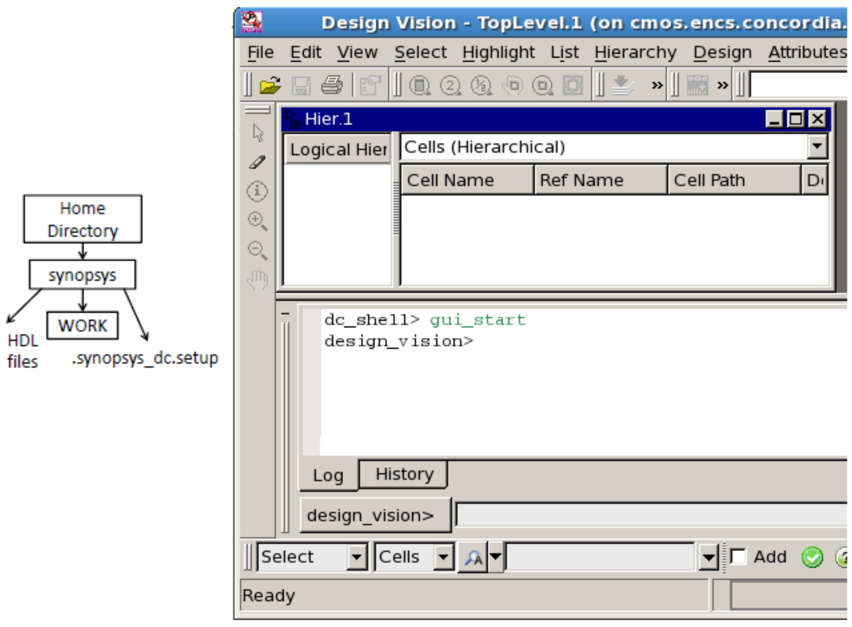
```
-- T. Obuchowicz
-- counts the number of 1s in the 4 bit input vector
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ones_counter is
    port(
        a : in  std_logic_vector(3 downto 0);
        f : out std_logic_vector(2 downto 0));
end;

architecture  rtl  of ones_counter  is
begin
    process(a)
    variable  ones : std_logic_vector(2 downto 0);
    begin
        ones := (others => '0');
        for index in a'range loop
            if a(index) = '1' then
```

```
                ones := ones + "1";
            end if;
        end loop;

        case ones is
            when "000" => f <= not("000");
            when "001" => f <= not("001");
            when "010" => f <= not("010");
            when "011" => f <= not("011");
            when others => f <= not("100");
        end case;
    end process;
end;
```

In the next section, we will synthesize this behavioral description of the design using Synopsys Design compiler.

2.3 Setting up the Environment and Important Libraries and File

In this tutorial, we will be using standard cell library ”class.db”. The following steps set-up the Design Compiler (DC) environment:



```
dc_shell> gui_start
design_vision>
```

Fig. 3: Directory structure and Design Vision graphical user interface

1. Create a separate directory. `mkdir synopsys`. Change directory to synopsys. `cd synopsys`

2. Source the environment script. `source /CMC/ENVIRONMENT/synopsys.env`
3. Copy the environment setup file and the HDL source files into the `synopsys` directory:

   1- `http://users.encs.concordia.ca/ lahiouel/coen6551/.synopsys_dc.setup`

   2- `http://users.encs.concordia.ca/ lahiouel/coen6551/ones.vhd`
4. Start DC GUI interface by typing `design_vision` at the unix prompt. You may run the design_vision in command mode by entering: `dc_shell -f ScriptFileName`
5. Observe DC generated messages carefully to make sure that correct libraries are loaded or via File ⇒ Setup. Or you may type in command mode: `get_design_lib_path work` to check that the work path is within your current directory.
6. Load top-level RTL file via File ⇒ Read menu (ones.vhd). Then press OK.
7. Select the top-level design module (ones_counter) via File ⇒ Analyze ⇒ Add. Then press OK. Or you may type in command mode: `analyze -library work -format vhdl {./ones.vhd}`
8. Elaborate the design via File ⇒ Elaborate. Then press OK. Or you may type in command mode: `elaborate ones_counter -architecture rtl -library default`
9. Check your design by selecting Design ⇒ Check Design. Or you may type in command mode: `check_design -multiple_designs`.

   If a value of 1 is returned it means that the design check command executed successfully and that there were no errors in the design. Most warnings can usually be ignored. Design Vision user interface allows one to navigate up and down through the design hierarchy. It is also possible to switch between symbol and schematic views (see Fig. 4).

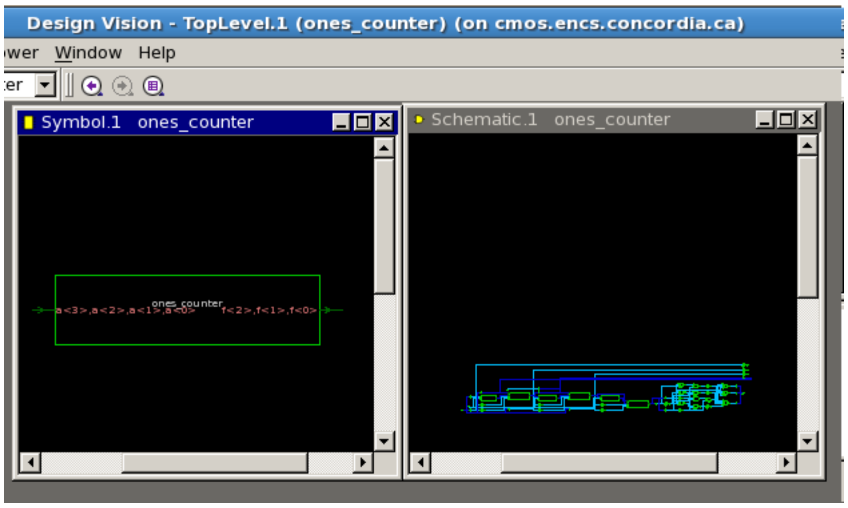


Fig. 4: Symbolic and schematic views

2.4 Setting the Design Attributes

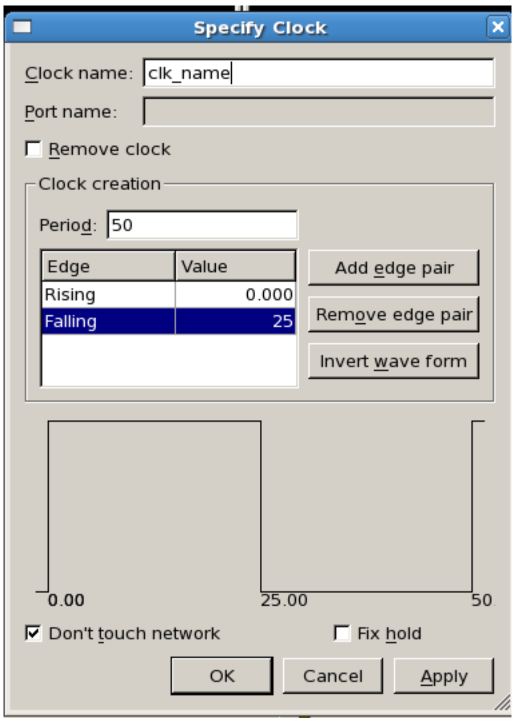
2.4.1 Clock Attributes

Following steps can be used to define the clock signal attributes such as the clock period and skew etc. (You can skip this section as the design used in this tutorial is purely combinational)

1. Select CLK pin in the Symbol View
2. Select Attributes ⇒ Clocks ⇒ Specify (see Fig. 5). Enter clock period in nanoseconds. Un check Don't Touch Network. This will force DC to insert clock tree buffers if needed to meet the design optimization requirements.
3. Click Apply and then close the dialog window.

Or you may use the followings in command mode:

```
create_clock -name "clk" -period 50 -waveform  0.000 25.000  clk_name
set_clock_uncertainty 0.1 clk
set_clock_latency 0.2 clk
set_clock_transition 0.1 clk
set_dont_touch_network clk
```



Edge	Value
Rising	0.000
Falling	25

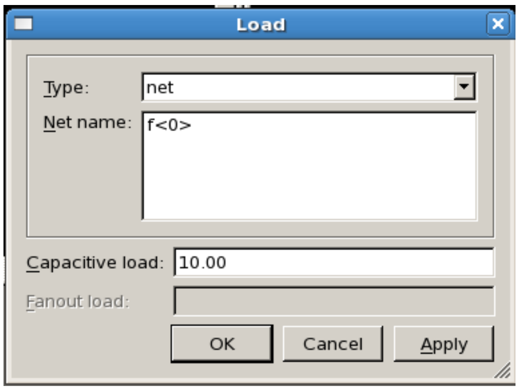
Fig. 5: Defining clock

2.4.2 Specifying the Output Load

Capacitive load on the design outputs can be specified as follows:

1. Select the desired input or output pin
2. Select Attributes ⇒ Operating Environment ⇒ Load (see Fig. 6)
3. Enter capacitive load in pF (e.g., 10pF)
4. Click Apply and close the dialog window.

Or you may use the followings in command mode:
`set_load 10 [get_ports "output_name"]`



Type: net
Net name: f<0>
Capacitive load: 10.00
Fanout load:

Fig. 6: Specifying capacitive load

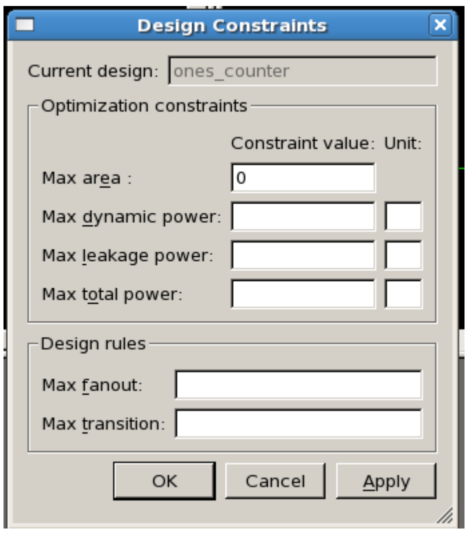
2.5 Specifying the Design Constraints

2.5.1 The Area Constraints

The Area constraints guide the area optimization process during design synthesis.(you can skip this section as we are not interested in area optimization at this time)

1. Select top level module in Symbol View. Select Attributes ⇒ Optimization constraints ⇒ Design constraints (see Fig. 7)
2. Set Max Area to 0 (μm^2). This will force the design compiler to optimize for smallest possible area.
3. Apply the area constraints and then close the dialog window.

Or you may use the followings in command mode:
`set_max_area 0`


Current design: ones_counter

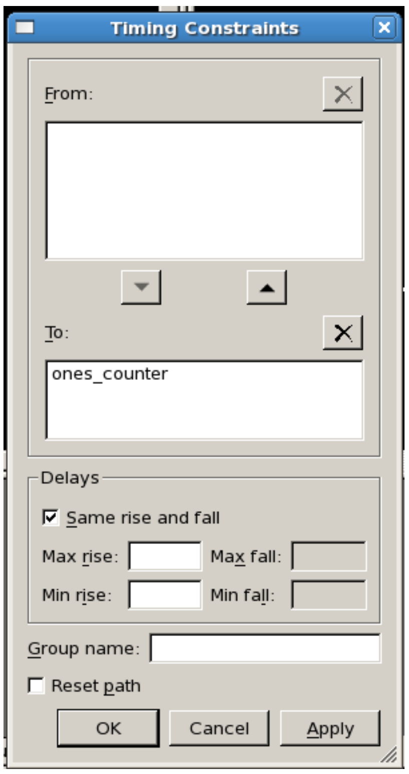
Optimization constraints

	Constraint value:	Unit:
Max area :	0	
Max dynamic power:		
Max leakage power:		
Max total power:		

Design rules

Max fanout:	
Max transition:	

Fig. 7: Area constraints


To: ones_counter

Delays

☑ Same rise and fall

Max rise:		Max fall:	
Min rise:		Min fall:	

Group name:

☐ Reset path

Fig. 8: Timing constraints

2.5.2 The Timing Constraints

Following steps can be used to specify the rise/fall and delay times constraints for the design.

1. Select the top-level module in Symbol View. Select Attributes ⇒ Optimization Constraints ⇒ Timing Constraints (see Fig. 8)
2. Specify the timing constraints by selecting appropriate Input/Output ports in the schematic view
3. Set the Rise and Fall time option as desired. Equal rise and fall times option is most commonly used.
4. Click Apply and update the timing constraints and then close the dialog window.

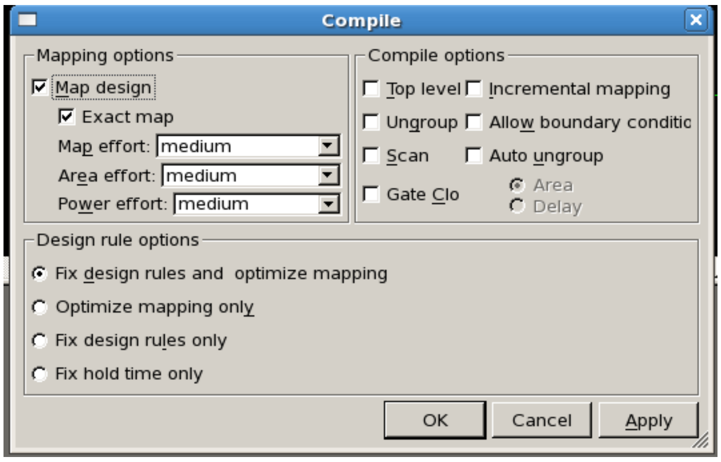
By default, Design Compiler optimizes for timing first and then for the area when presented with conflicting goals.

2.6 Synthesis and Optimization

The steps for synthesizing and optimizing the design are as follows:

1. Click Design ⇒ Compile Design (see Fig. 9)
2. Select Map Effort level (Low, Medium, High) and then click OK.
3. Observe the log file for errors and warnings. If the synthesis and optimization process completes successfully a value of 1 is returned. Most warnings can usually be ignored.
4. If there were no errors then close the dialog window.

The synthesis and optimization steps can take a long time to finish if high map and verify effort options are selected.



Mapping options
- [✓] Map design
- [✓] Exact map
- Map effort: medium
- Area effort: medium
- Power effort: medium

Compile options
- [ ] Top level
- [ ] Incremental mapping
- [ ] Ungroup
- [ ] Allow boundary conditio
- [ ] Scan
- [ ] Auto ungroup
- [ ] Gate Clo
- [✓ Area] Delay

Design rule options
- [✓ Fix design rules and optimize mapping]
- Optimize mapping only
- Fix design rules only
- Fix hold time only

Fig. 9: Design optimization

2.7 Analysis Report

To produce analysis report for the design:

1. Select the appropriate report from the list on Design ⇒ Report.
2. Analysis report can be saved to a file by selecting the "File" option in "Send Output To" field.
3. Finally, click on Apply to generate the synthesis and optimization report.
4. The critical path can be viewed by selecting Analysis ⇒ Highlight ⇒ Critical Path, in the Schematic View.

2.8 Exporting the Design Files

The results of DC synthesis and analysis can be saved as a VHDL or a verilog netlist:

1. Select File ⇒ Save As. Enter file name. Select VHDL or Verilog under File Format and then press OK. [Lets save the synthesized design in VHDL format. Name the synthesized design as "ones_syn.vhd"]
2. To export timing constraints, click on Files ⇒ Save Info ⇒ Constraints. Select a filename, and then press OK. Further explore the Setup Menu. (1) Open a Command Window (Setup ⇒ Command Window). Go through the synthesis example once again and observe the commands being executed. (2) Explore Setup ⇒ Defaults, Setup ⇒ Variables, and Setup ⇒ Execute Scripts.
3. Close the dialog window and exit Design Vision.

Main steps involved in the synthesis and optimization of RTL designs were described in this section. For more information, please see
[/CMC/tools/synopsys/syn/doc/syn/tutorial] and
[/CMC/tools/synopsys/syn/doc/syn/examples]

Next two sections of this tutorial will describe two equivalence checking tools, namely, Cadence Conformal Logic Equivalence Checker, and Synopsys Formality.

3 Synopsys Formality

3.1 Introduction

Formality is a functional equivalence checking tool from Synopsys. The equivalence checking process flow is shown in Fig. 10. After initial environment setup, separate containers for the reference and implemented design are created. Libraries and the design files are then loaded. Verification is run after top level design modules and compare points have been identified. In case the two designs are not equivalent, Formality identifies the problem areas and displays them in both schematic and HDL source views. Formality can be run in both command line and gui modes.

More information can be found in [/CMC/tools/synopsys/fm/doc/fm]

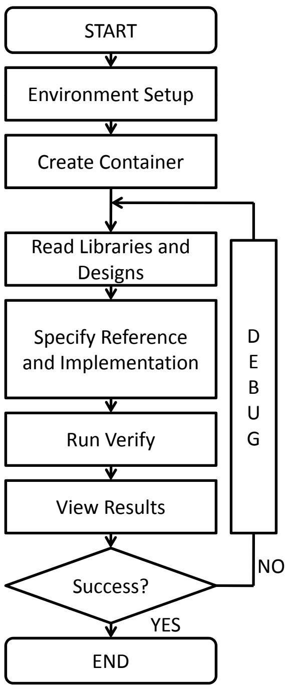


Fig. 10: Formality equivalence checking flow

3.2 Setting up the Environment

Before you begin, setup the environment as follows:

1. Create a separate work directory. `mkdir formality`. Change directory to formality. `cd formality`
2. Source the formality environment script. `source /CMC/ENVIRONMENT/formality.env`
3. Copy the HDL files into the formality directory. The files used in this tutorial can be found here: [http://users.ece.concordia.ca/∼tahar/coen6551/coen6551.proj.html].

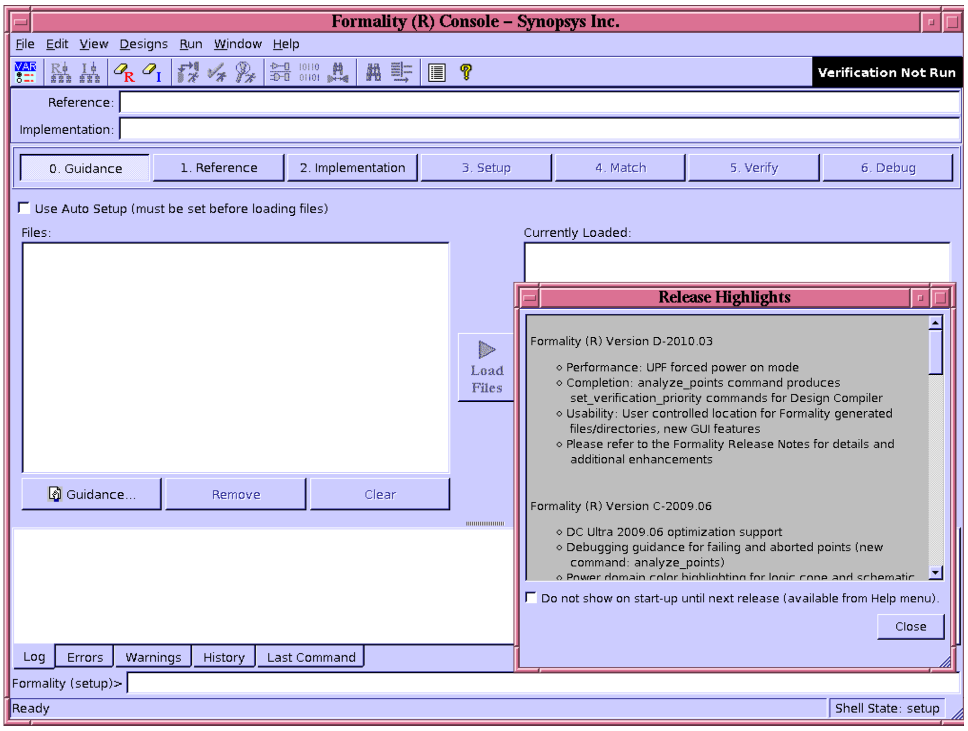
3.2.1 Required Libraries and Files

We will be using following files in this tutorial:

1. The reference design (RTL design): `ones.vhd`
2. The implemented design (gate level design): `ones_syn.vhd`
3. The primitive cells library: class.lib

3.3 Starting Formality

To start formality, type `formality &` at the unix prompt. The main formality window will appear after a few second (see Fig. 11). Formality can be run in command mode by entering: `fm_shell`



Formality (R) Version D-2010.03

- Performance: UPF forced power on mode
- Completion: analyze_points command produces set_verification_priority commands for Design Compiler
- Usability: User controlled location for Formality generated files/directories, new GUI features
- Please refer to the Formality Release Notes for details and additional enhancements

Formality (R) Version C-2009.06

- DC Ultra 2009.06 optimization support
- Debugging guidance for failing and aborted points (new command: analyze_points)
- Power domain color highlighting for logic cone and schematic

Fig. 11: Starting formality

Please see the Formality Users Guide (user.pdf) for a detailed description of the graphical user interface [/CMC/tools/synopsys/fm/doc/fm].

3.4 Design Input

3.4.1 Reading the Reference Design

Click on the "1. Reference" button.

1. Read Design Files: In VHDL tab, click on VHDL button. Select `ones.vhd`, then click open. Select the `ones.vhd` file and then click on Load Files button to load the reference design file. Repeat the same steps until all reference design files are loaded.
2. Read DB Files: Observe that GTECH library is selected by default. Read in the "class.db" file.
3. Set Top Design: Choose `ones_counter` and click on Set Top Design. You will now see a green check mark on "1. Reference" button. In the formality window, observe the log of executed commands. Click on Set Reference button to add the design files to the reference container.

3.4.2 Reading the Implemented Design

Click on the "2. Implementation" button.

1. Read Design Files: In VHDL tab, click on VHDL button. Select `ones_syn.vhd`, then click open. Select the `ones_syn.vhd` file and then click on Load Files button to load the reference design files. Repeat the same steps until all reference design files are loaded.
2. Read DB Files: Observe that GTECH library is selected by default. Read in the "class.db" file.
3. Set Top Design: Choose `ones_counter` and click on Set Top Design. You will now see a green check mark on "2. Implementation" button. In the formality window, observe the log of executed commands. Click on Set Reference button to add the design file to implementation container.

Note: You may have to read in certain libraries and packages, in some cases, before you read the reference and/or implemented design netlists.

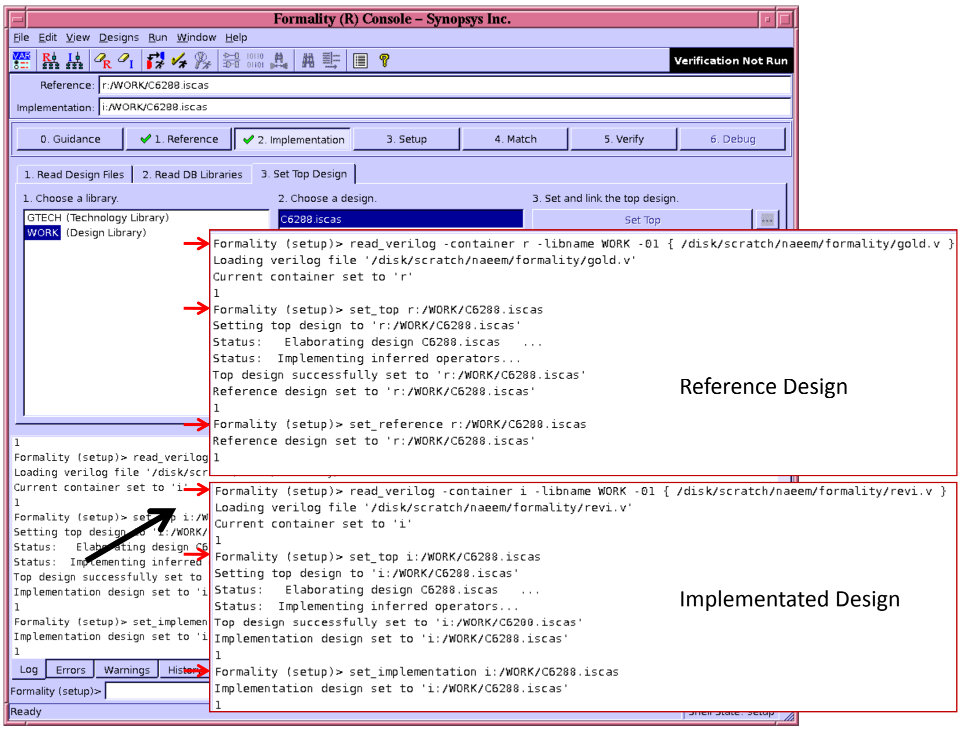
Figure 12 shows Formality window when reference and implemented designs have been successfully loaded into two separate containers and are ready for equivalence checking.

3.5 Setup

Click on "3. Setup" button. Constants, design parameters, and equivalences between ports and nets can be set here. In this tutorial, we will skip this step. By default Formality will check for port equivalence. For more information about this step please consult Formality users guide (user.pdf) [/CMC/tools/synopsys/fm/doc/fm].

3.6 Match

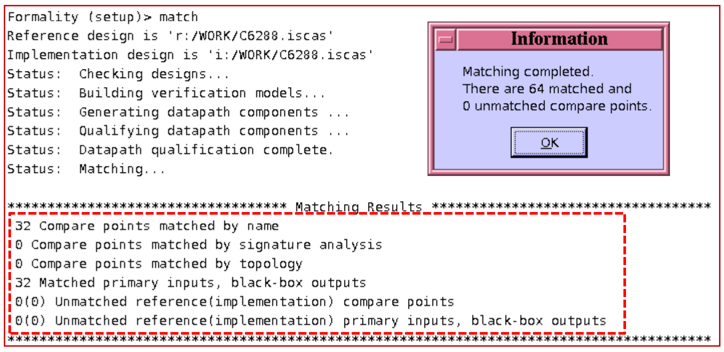
Click on "4. Match" button. Click on Run Matching button. You will see match command execution and its results in the Formality console (see Fig. 13).



```
Formality (setup)> read_verilog -container r -libname WORK -01 { /disk/scratch/naeem/formality/gold.v }
Loading verilog file '/disk/scratch/naeem/formality/gold.v'
Current container set to 'r'
1
Formality (setup)> set_top r:/WORK/C6288.iscas
Setting top design to 'r:/WORK/C6288.iscas'
Status:   Elaborating design C6288.iscas   ...
Status:  Implementing inferred operators...
Top design successfully set to 'r:/WORK/C6288.iscas'
Reference design set to 'r:/WORK/C6288.iscas'
1
Formality (setup)> set_reference r:/WORK/C6288.iscas
Reference design set to 'r:/WORK/C6288.iscas'
1
```

```
Formality (setup)> read_verilog -container i -libname WORK -01 { /disk/scratch/naeem/formality/revi.v }
Loading verilog file '/disk/scratch/naeem/formality/revi.v'
Current container set to 'i'
1
Formality (setup)> set_top i:/WORK/C6288.iscas
Setting top design to 'i:/WORK/C6288.iscas'
Status:   Elaborating design C6288.iscas   ...
Status:  Implementing inferred operators...
Top design successfully set to 'i:/WORK/C6288.iscas'
Implementation design set to 'i:/WORK/C6288.iscas'
1
Formality (setup)> set_implementation i:/WORK/C6288.iscas
Implementation design set to 'i:/WORK/C6288.iscas'
1
```

Fig. 12: Reference and implemented designs loaded and ready for equivalence checking



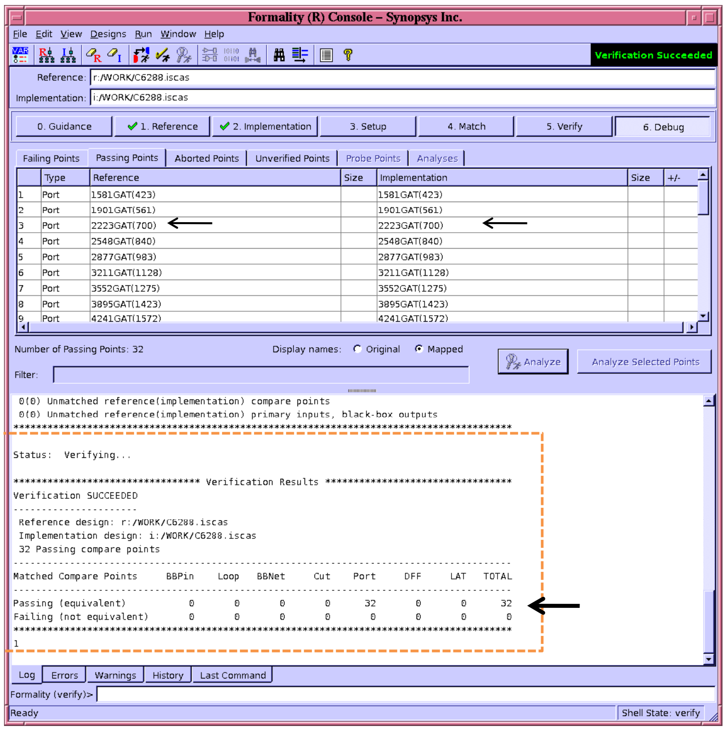
```
Formality (setup)> match
Reference design is 'r:/WORK/C6288.iscas'
Implementation design is 'i:/WORK/C6288.iscas'
Status:  Checking designs...
Status:  Building verification models...
Status:  Generating datapath components ...
Status:  Qualifying datapath components ...
Status:  Datapath qualification complete.
Status:  Matching...

********************************** Matching Results **********************************
32 Compare points matched by name
0 Compare points matched by signature analysis
0 Compare points matched by topology
32 Matched primary inputs, black-box outputs
0(0) Unmatched reference(implementation) compare points
0(0) Unmatched reference(implementation) primary inputs, black-box outputs
**************************************************************************************
```

```
Matching completed.
There are 64 matched and
0 unmatched compare points.
```

Fig. 13: Match command execution results

3.7 Verify

Click on "5. Verify" button. As the verification process runs you will see its progress and when it finishes the verification results are printed in the Formality console window (see Fig. 14).



Reference: r:/WORK/C6288.iscas

Implementation: i:/WORK/C6288.iscas

	Type	Reference	Size	Implementation	Size	+/-
1	Port	1581GAT(423)		1581GAT(423)		
2	Port	1901GAT(561)		1901GAT(561)		
3	Port	2223GAT(700)		2223GAT(700)		
4	Port	2548GAT(840)		2548GAT(840)		
5	Port	2877GAT(983)		2877GAT(983)		
6	Port	3211GAT(1128)		3211GAT(1128)		
7	Port	3552GAT(1275)		3552GAT(1275)		
8	Port	3895GAT(1423)		3895GAT(1423)		
9	Port	4241GAT(1572)		4241GAT(1572)		

Number of Passing Points: 32   Display names: ( ) Original [✓ Mapped]

Filter:

```
 0(0) Unmatched reference(implementation) compare points
 0(0) Unmatched reference(implementation) primary inputs, black-box outputs
*********************************************************************************

Status:  Verifying...

********************************* Verification Results ********************************
Verification SUCCEEDED
----------------------
 Reference design: r:/WORK/C6288.iscas
 Implementation design: i:/WORK/C6288.iscas
 32 Passing compare points
-------------------------------------------------------------------------------------
Matched Compare Points     BBPin    Loop   BBNet     Cut    Port     DFF     LAT   TOTAL
-------------------------------------------------------------------------------------
Passing (equivalent)           0       0       0       0      32       0       0      32
Failing (not equivalent)       0       0       0       0       0       0       0       0
*********************************************************************************
1
```

Formality (verify)>

Ready   Shell State: verify

Fig. 14: Equivalence successfully verified

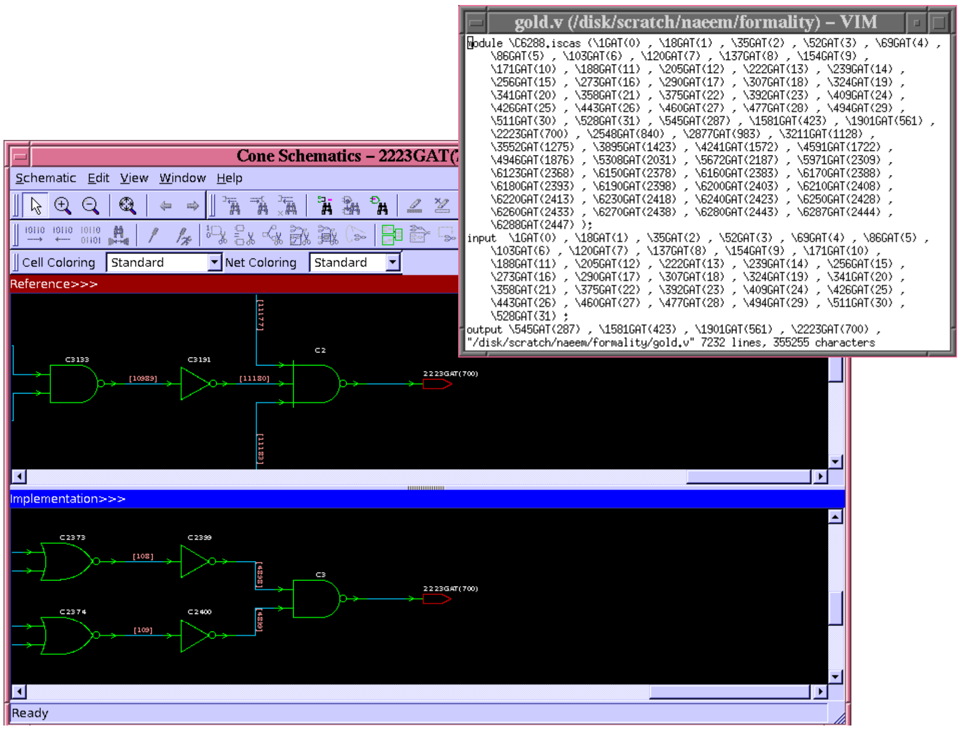
3.8 Debug

In case if equivalence checking fails, one can run the diagnose command to locate points which did not match. To trace the possible problem, select a port or a net. Right click and then select a desired option from many available options. These options contain, viewing logic cones, schematic and source views. Figure 15 shows the schematic and source views. For more details consult chapter 8 of the Formality users guide (user.pdf) [/CMC/tools/synopsys/fm/doc/fm].

At this point you can close Formality GUI. It is possible to save the state of the session at any point in the verification process and reload it at a later stage (File ⇒ Save Session, File ⇒ Restore Session).

3.9 How to create a command script

1. A list of all the commands executed in any session is kept in the `fm_shell_command.log` file. This script file can later be modified and re run from `fm_shell`.
2. A transcript of the session can be saved as a `Tcl` script (File ⇒ Save Transcript).



```
module \C6288.iscas (\1GAT(0) , \18GAT(1) , \35GAT(2) , \52GAT(3) , \69GAT(4) ,
    \86GAT(5) , \103GAT(6) , \120GAT(7) , \137GAT(8) , \154GAT(9) ,
    \171GAT(10) , \188GAT(11) , \205GAT(12) , \222GAT(13) , \239GAT(14) ,
    \256GAT(15) , \273GAT(16) , \290GAT(17) , \307GAT(18) , \324GAT(19) ,
    \341GAT(20) , \358GAT(21) , \375GAT(22) , \392GAT(23) , \409GAT(24) ,
    \426GAT(25) , \443GAT(26) , \460GAT(27) , \477GAT(28) , \494GAT(29) ,
    \511GAT(30) , \528GAT(31) , \545GAT(287) , \1581GAT(423) , \1901GAT(561) ,
    \2223GAT(700) , \2548GAT(840) , \2877GAT(983) , \3211GAT(1128) ,
    \3552GAT(1275) , \3895GAT(1423) , \4241GAT(1572) , \4591GAT(1722) ,
    \4946GAT(1876) , \5308GAT(2031) , \5672GAT(2187) , \5971GAT(2309) ,
    \6123GAT(2368) , \6150GAT(2378) , \6160GAT(2383) , \6170GAT(2388) ,
    \6180GAT(2393) , \6190GAT(2398) , \6200GAT(2403) , \6210GAT(2408) ,
    \6220GAT(2413) , \6230GAT(2418) , \6240GAT(2423) , \6250GAT(2428) ,
    \6260GAT(2433) , \6270GAT(2438) , \6280GAT(2443) , \6287GAT(2444) ,
    \6288GAT(2447) );
input  \1GAT(0) , \18GAT(1) , \35GAT(2) , \52GAT(3) , \69GAT(4) , \86GAT(5) ,
    \103GAT(6) , \120GAT(7) , \137GAT(8) , \154GAT(9) , \171GAT(10) ,
    \188GAT(11) , \205GAT(12) , \222GAT(13) , \239GAT(14) , \256GAT(15) ,
    \273GAT(16) , \290GAT(17) , \307GAT(18) , \324GAT(19) , \341GAT(20) ,
    \358GAT(21) , \375GAT(22) , \392GAT(23) , \409GAT(24) , \426GAT(25) ,
    \443GAT(26) , \460GAT(27) , \477GAT(28) , \494GAT(29) , \511GAT(30) ,
    \528GAT(31) ;
output \545GAT(287) , \1581GAT(423) , \1901GAT(561) , \2223GAT(700) ,
"/disk/scratch/naeem/formality/gold.v" 7232 lines, 355255 characters
```

Fig. 15: Debug schematic and source views

4 Cadence Conformal

4.1 Introduction

Conformal LEC is a powerful equivalence checking tool. It can provide a formal proof that the output from Synthesis matches the original RTL code. It can do all of that without having to run a single simulation. In this section of the tutorial, we will learn how to read in a RTL and a synthesized design and how to prove that they are functionally equivalent. The original RTL netlist is usually referred to as the "golden" design. It serves as the reference for the comparison. The synthesized gate-level netlist is also called the "Revised" design.

4.2 Setting up the Environment

Before you begin, setup the environment as follows:

1. Create a separate work directory. `mkdir conformal`. Change directory to conformal. `cd conformal`
2. Source the conformal environment script. `source /CMC/ENVIRONMENT/conformal.env`
3. Copy the HDL files (both golden and revised) into the conformal directory. The files used in this tutorial can be found here:

   [http://users.ece.concordia.ca/∼tahar/coen6551/coen6551.proj.html].

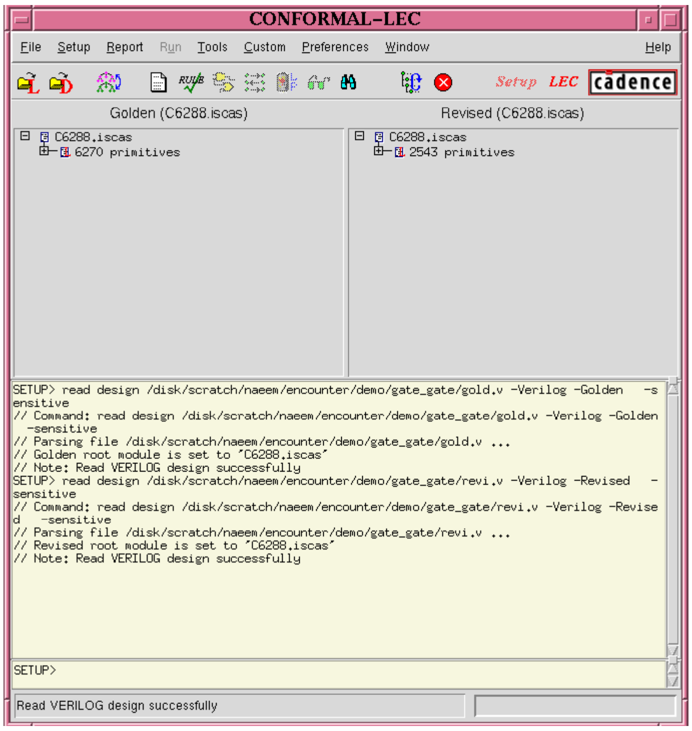
4.2.1 Required Libraries and Files

Following files are used in this tutorial:

1. The RTL design: `ones.vhd`
2. The gate level design: `ones_syn.vhd`
3. The primitive cells library: class.lib
4. The conformal script: `lec.do`

4.3 Starting Conformal

To start the tool, type `lec -XL &` at the unix prompt. The main Conformal LEC window will appear (see Fig. 16). Conformal can be run in command mode by entering: `lec -nogui`



```
SETUP> read design /disk/scratch/naeem/encounter/demo/gate_gate/gold.v -Verilog -Golden   -s
ensitive
// Command: read design /disk/scratch/naeem/encounter/demo/gate_gate/gold.v -Verilog -Golden
  -sensitive
// Parsing file /disk/scratch/naeem/encounter/demo/gate_gate/gold.v ...
// Golden root module is set to 'C6288.iscas'
// Note: Read VERILOG design successfully
SETUP> read design /disk/scratch/naeem/encounter/demo/gate_gate/revi.v -Verilog -Revised   -
sensitive
// Command: read design /disk/scratch/naeem/encounter/demo/gate_gate/revi.v -Verilog -Revise
d   -sensitive
// Parsing file /disk/scratch/naeem/encounter/demo/gate_gate/revi.v ...
// Revised root module is set to 'C6288.iscas'
// Note: Read VERILOG design successfully
```

```
SETUP>
```

```
Read VERILOG design successfully
```

Fig. 16: Confomral LEC

4.4 Design Input

4.4.1 Reading the RTL netlist

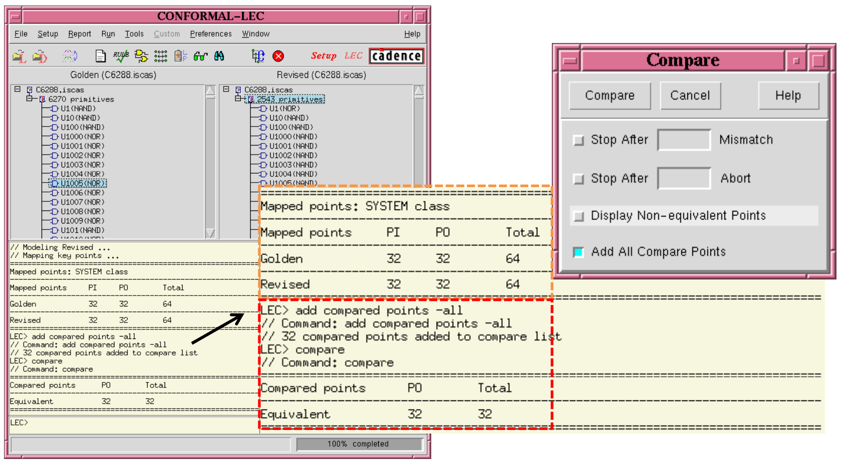
Read the RTL design: Click on "File ⇒ Read Design". In the Design option: Select VHDL format. Click on "ones.vhd". Set the "Type" to "Golden". Click on Add Selected. Then load the file by clicking "OK" (see Fig. 16)

4.4.2 Reading the Gate-level Netlist

If your gate level design uses a standard cell library, you may have to load the library files before you load the design files. File *Rightarrow* Library. Select "class.lib". Read the synthesized design: Click on File ⇒ Read Design. In the Design option: Select VHDL format. Select "ones_syn.vhd". Set the "Type" to "Revised". Click on Add Selected. Then load the file by clicking OK (see Fig. 16).
Note: You may have to read in certain libraries and packages in some cases, before you read the golden and/or revised netlists.

4.5 Design Preparation and Key point Mapping

Since both the designs have been successfully loaded, we can now start the verification process. Conformal has 2 operating modes, the "Setup" and the "LEC" mode. Switch to the LEC mode by Clicking on the "LEC" icon in the upper right hand corner of the window (see Fig. 17).



```
// Modeling Revised ...
// Mapping key points ...
==============================================
Mapped points: SYSTEM class
----------------------------------------------
Mapped points      PI      PO        Total
----------------------------------------------
Golden             32      32        64
----------------------------------------------
Revised            32      32        64
==============================================
LEC> add compared points -all
// Command: add compared points -all
// 32 compared points added to compare list
LEC> compare
// Command: compare
==============================================
Compared points      PO        Total
----------------------------------------------
Equivalent           32        32
==============================================
LEC>
```

[ ] Stop After [ ] Mismatch
[ ] Stop After [ ] Abort
[ ] Display Non-equivalent Points
[✓] Add All Compare Points

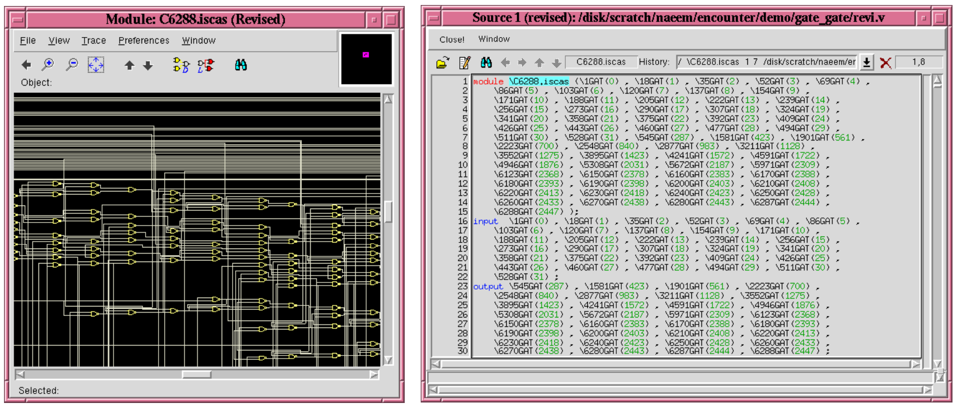
Fig. 17: Conformal map points and equivalence results

A table is now printed in the conformal LEC window. It lists the primary inputs (PI) and primary outputs (PO) in both the revised and golden designs. They are equal if the golden and revised designs have the same number of inputs and outputs. To run the equivalence checker, select "Run ⇒ Compare" and click "OK" (see Fig. 17). The equivalence checker now reduces the two designs into canonical representations and then checks to see if they are equal. In the example shown in Fig. 17, all 3 outputs are equivalent. Or in other words, the RTL and the synthesized designs are functionally equivalent.

4.6 Reporting Results

4.6.1 Schematic and Source Views

This feature is very helpful in debugging the design. In the "Golden" or "Revised" column of the main conformal LEC window, right click on a file name or a cell name and then select either "Schematic" or "Source" as desired. The schematic or source view will be displayed in a separate window (see Fig. 18).



Module: C6288.iscas (Revised)
File View Trace Preferences Window
Object:
Selected:
Source 1 (revised): /disk/scratch/naeem/encounter/demo/gate_gate/revi.v
Close! Window
C6288.iscas
History: / \C6288.iscas 1 7 /disk/scratch/naeem/er
1,8

```
module \C6288.iscas (\1GAT(0) , \18GAT(1) , \35GAT(2) , \52GAT(3) , \69GAT(4) ,
    \86GAT(5) , \103GAT(6) , \120GAT(7) , \137GAT(8) , \154GAT(9) ,
    \171GAT(10) , \188GAT(11) , \205GAT(12) , \222GAT(13) , \239GAT(14) ,
    \256GAT(15) , \273GAT(16) , \290GAT(17) , \307GAT(18) , \324GAT(19) ,
    \341GAT(20) , \358GAT(21) , \375GAT(22) , \392GAT(23) , \409GAT(24) ,
    \426GAT(25) , \443GAT(26) , \460GAT(27) , \477GAT(28) , \494GAT(29) ,
    \511GAT(30) , \528GAT(31) , \545GAT(287) , \1581GAT(423) , \1901GAT(561) ,
    \2223GAT(700) , \2548GAT(840) , \2877GAT(983) , \3211GAT(1128) ,
    \3552GAT(1275) , \3895GAT(1423) , \4241GAT(1572) , \4591GAT(1722) ,
    \4946GAT(1876) , \5308GAT(2031) , \5672GAT(2187) , \5971GAT(2309) ,
    \6123GAT(2368) , \6150GAT(2378) , \6160GAT(2383) , \6170GAT(2388) ,
    \6180GAT(2393) , \6190GAT(2398) , \6200GAT(2403) , \6210GAT(2408) ,
    \6220GAT(2413) , \6230GAT(2418) , \6240GAT(2423) , \6250GAT(2428) ,
    \6260GAT(2433) , \6270GAT(2438) , \6280GAT(2443) , \6287GAT(2444) ,
    \6288GAT(2447) );
input  \1GAT(0) , \18GAT(1) , \35GAT(2) , \52GAT(3) , \69GAT(4) , \86GAT(5) ,
    \103GAT(6) , \120GAT(7) , \137GAT(8) , \154GAT(9) , \171GAT(10) ,
    \188GAT(11) , \205GAT(12) , \222GAT(13) , \239GAT(14) , \256GAT(15) ,
    \273GAT(16) , \290GAT(17) , \307GAT(18) , \324GAT(19) , \341GAT(20) ,
    \358GAT(21) , \375GAT(22) , \392GAT(23) , \409GAT(24) , \426GAT(25) ,
    \443GAT(26) , \460GAT(27) , \477GAT(28) , \494GAT(29) , \511GAT(30) ,
    \528GAT(31) ;
output \545GAT(287) , \1581GAT(423) , \1901GAT(561) , \2223GAT(700) ,
    \2548GAT(840) , \2877GAT(983) , \3211GAT(1128) , \3552GAT(1275) ,
    \3895GAT(1423) , \4241GAT(1572) , \4591GAT(1722) , \4946GAT(1876) ,
    \5308GAT(2031) , \5672GAT(2187) , \5971GAT(2309) , \6123GAT(2368) ,
    \6150GAT(2378) , \6160GAT(2383) , \6170GAT(2388) , \6180GAT(2393) ,
    \6190GAT(2398) , \6200GAT(2403) , \6210GAT(2408) , \6220GAT(2413) ,
    \6230GAT(2418) , \6240GAT(2423) , \6250GAT(2428) , \6260GAT(2433) ,
    \6270GAT(2438) , \6280GAT(2443) , \6287GAT(2444) , \6288GAT(2447) ;
```

Fig. 18: Conformal schematic and source view

4.7 How to create a command script

1. The DO file script can be generated from the LEC GUI by:
   File ⇒ Save dofile
2. The DO file script can be run from the LEC GUI by:
   File ⇒ Do dofile
3. The DO scripts can also be run directly from the unix prompt by:
   `lec -nogui -do lec.do`

References

[1] M. Moris Mano, C. R. Kime, *Logic and Computer Design Funcdamentals*, Second Edition, ISBN 0-12-012468-0

[2] D. R. Perry, *VHDL*, McGraw-Hill series in Computer Engineering, Second Edition, ISBN 0-07-049434-7, TK 7885.7.P47 1993

A Synopsys Design Vision

A.1 Design Vision Setup Files

```
#             This is the .synopsys_dc.setup file required
#             for use with the class.db synthesis technology library
#             This library is used by Synopsys for synthesis training
#             courses.
#
#             Henda ARIDHI
#             April 22, 2012
set search_path [concat $search_path ./  ./CMC/tools/synopsys/syn/libraries/syn]
#if your src files are in a diffrent folder update the search path for example:
#set search_path [concat $search_path ./vhdl_source/  ./CMC/tools/synopsys/syn/libraries/syn]
set target_library  class.db
set link_library  class.db
set symbol_library  class.sdb
set compile_fix_multiple_port_nets true
define_design_lib WORK -path ./work
#             the bus naming syle determines how one refers to elements of
#             a bus in a dc_shell script such as count<2> if "%s<%d>" is the style
#             or count_2 if "%s_%d" is used as the style
#             the <%d> causes SDF problems since the SDF file refers to buses as count_reg_0
#             and does not use <>

set bus_naming_style  "%s<%d>"

#             if you want to back annotate Synopsys SDF files onto Synopsys gate-level netlists
#             use the %s_%d bus naming style... but make sure that in your script
#             you refer to elements of buses as count_2, count_1 , etc !!!
#             For Xilinx generated netlists and SDF files there is no problem
#             can use the standard %s<%d> style */
#             bus_naming_style = "%s_%d"

set bus_inference_style  "%s<%d>"
set bus_dimension_separator_style "><"

#             to write out EDI netlists, this must be set to "true"
set edifout_netlist_only  "true"
```

The class.db library file used in this tutorial is located in:
[/CMC/tools/synopsys/syn/libraries/syn/class.db

A.2 Design Example Files

See [/CMC/tools/synopsys.2005a/syn/doc/syn/tutorial]
and [/CMC/tools/synopsys.2005a/syn/doc/syn/examples]

B Synopsys Formality

B.1 The reference

See [/CMC/tools/cadence.2011a/CONFRML/share/cfm/lec/demo/gate_gate]
Also see [/CMC/tools/cadence.2011a/CONFRML/share/cfm/lec/demo]

B.2 The implementation

See [/CMC/tools/cadence.2007a/CONFRML/share/cfm/lec/demo/gate_gate]
Also see [/CMC/tools/cadence.2011a/CONFRML/share/cfm/lec/demo]

Note: we will be using the same design files for verification with Cadence LEC in this tutorial.

B.3 The command script

B.4 Other useful information

See user and reference documents in [/CMC/tools/synopsys/fm/doc/fm]

If you are interested in reading more about the tool, a good starting point is the user.pdf document.

One can also find many good tool tutorials on the web. For example,
see [http://www.chiptalk.org]

C Cadence Conformal

C.1 The golden design

See [/CMC/tools/cadence.2007a/CONFRML/share/cfm/lec/demo/gate_gate]
Also see [/CMC/tools/cadence.2007a/CONFRML/share/cfm/lec/demo]

C.2 The revised design

See [/CMC/tools/cadence.2007a/CONFRML/share/cfm/lec/demo/gate_gate]
Also see [/CMC/tools/cadence.2007a/CONFRML/share/cfm/lec/demo]

C.3 The command script

```
//**********************************************
// design input
//**********************************************
Read Design -golden  gold.v
Read Design -revised revi.v

report design data -both

//**********************************************
// design preparation and key points mapping
//**********************************************
set system mode lec

//**********************************************
// compare mapped key points
//**********************************************
add compare point -all
compare

//**********************************************
// reporting result
//**********************************************
report compare data -summary
```

C.4 ¬ (Rarely Asked Questions)

C.5 Remote access

Following information was kindly provided by our CAD tool GURU (Ted Obuchowicz).
If you are connecting remotely using 'ssh' take note of the following:

1. On a solaris host use : /usr/bin/ssh to connect to a solaris 10 remote host
2. On a solaris host use : /common/bin/ssh -X to connect to a solaris 10 remote host
3. On a Linux host use : /usr/bin/ssh -X to connect to a solaris 10 remote host

C.6 Commonly used UNIX commands

C.7 Have you ever wondered?

1. Checking memory usage?
2. How to list active processes and how to kill a process?
3. How to check workstation information?
4. what is emacs? and what is nEdit?

C.8 Screen Capture

1. Type xv & in an xterm on solaris workstations.
2. Right click on the main window to get a menu.
3. Use GRAB button to capture the screen and SAVE button to save the file.

C.9 Tutorial Files

`http://users.encs.concordia.ca/~n_ab/coen7501files/tut_files/`

C.10 Getting started - May 12, 2010 in class tool demonstration

`http://users.encs.concordia.ca/~n_ab/coen7501files/information_from_may_12_demo.txt`

C.11 Unix Tutorial

```
[http://www.ee.surrey.ac.uk/Teaching/Unix/]

Unix Commands

ls
dir
cp
mv
man
apropos
chmod
rm

pwd
cd




ps
kill -9 pid
top
```

```
prtdiag

ssh
sftp

file redirection using ">"

A simple example:
Create a few files.
create a directory structure
copy files back and forth
change permissions of the files
look up more information about the files
rm some of the files
move files

some useful commands for printing and converting file formats.

ps2pdf
pdf2ps

also see http://users.encs.concordia.ca/~n_ab/coen7501files/fmradio/fmradio.txt for more information.


lpr -Pprintername filename
qpr -d -Pprintername filename


Printing multiple pages on a single page:
psnup -pa4 -d -nup 2 test.ps test1.ps

open both test.ps and test1.psin gv and see the difference. use man psnup for more information.


Writing a simple script using basic unix commands

clean script: removes files
---
rm -rf FM*
rm -rf *.lck
rm -rf *.log
---
```

C.12 Unix Commands

1. tee
2. PS to PDF: ps2pdf fulltext.pdf fulltext.ps
3. PDF 2 PS: psnup -pa4 -d -nup 2 fulltext.ps fulltext1.ps

4. ASCII to PS: a2ps test.txt -o test.ps ps2pdf test.ps test.pdf pdf2ps test.pdf test.ps
5. Printing multiple pages on a single page: psnup -pa4 -d -nup 2 test.ps test1.ps
6. Printing on both sides of a page: qpr -d -Plint test1.ps
7. Check the man pages for more details
8. You can also write custom scripts to meet your needs

C.13 How to force 2004 version of formality to start instead of 2011 version of formality

Source synopsys.env first and then formality.env to force use of 2004 version of formality.

C.14 Other useful information

See user and reference documents in [/CMC/tools/cadence.2007a/CONFRML/doc/]

Complete documentation is found in [/CMC/tools/cadence/CONFRML/doc]

If you are interested in reading more about the tool, a good starting point is the Conformal_User.pdf document.

Currently we have Version 6.1 of conformal installed on our workstations.

One can also find many good tool tutorials on the web. For example,
see [http://www.chiptalk.org]