Report

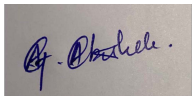
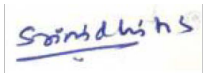
Confirmation of Originality

Faculty of Engineering and Computer Science

Course Name & Number/Term: COEN 6551 (e.g., ENGR410/2) **Section:** ____ (e.g. M) **Instructor:** Dr. Sofiène Tahar.

Having researched and prepared this report for submission to the Faculty of Engineering & Computer Science, the undersigned certify that the following statements are to the best of my/our knowledge true:

1. The undersigned have written this report myself/themselves.
2. This report consists entirely of ideas, observations, references, information and conclusions composed or paraphrased by the undersigned, as the case may be, except for statements contained within quotation marks and attributed to the best of my/our knowledge to their proper source in footnotes or otherwise referenced.
3. With the exception of material in appendices, the undersigned have endeavored to ensure that direct quotations make up a very small proportion of the attached report, not exceeding 5% of the word count.
4. Each paragraph of this report that contains material which the undersigned have paraphrased from a source (print sources, multimedia sources, web-based sources, course notes or personal interviews, etc), has been identified by numerical reference citation.
5. All of the sources that the undersigned consulted and/or included in the report have been listed in the Reference section of the report.
6. All drawings, diagrams, photos, maps or other visual items derived from sources have been identified by numerical reference citations in the caption.
7. Each of the undersigned has revised, edited and proofread this report individually.
8. In preparing this report the undersigned have read and followed the guidelines found in Form and Style, by Patrick MacDonagh and Jack Borden (Fourth Edition: May 2000), available at http://www.encs.concordia.ca/scs/Forms/Form&Style.pdf.

Name (please print clearly)	ID No:	Signature:	Date:
ABISHEK ARUMUGAM THIRUSELVI	40218896		13 JUNE 2022
AKASH RAJMOHAN	40218469		13 JUNE 2022
MANISH PEJATHAYA	40194909		13 JUNE 2022
SRINIDHI HONAKERE SRINIVAS	40221128		13 JUNE 2022

Do Not Write in this Space – Reserved for Instructor

Concordia University
Engineering and Computer Science

PROJECT REPORT

COEN 6551: FORMAL HARDWARE VERIFICATION

FORMAL VERIFICATION OF BAUDGEN CORE USING EQUIVALENCE CHECKING METHOD

Group: A

Authors:

Abishek Arumugam Thiruselvi - 40218896
Akash Rajmohan – 40218469
Manish Pejathaya – 40194909
Srinidhi Honakere Srinivas - 40221128

Instructor: Dr. Sofiene Tahar

Date: 13/06/2022

Abstract:

The idea of the project is Formal verification of BaudGen core using the method of equivalence checking tools Synopsys Formality and Cadence Conformal-LEC. Both the tools are industry-standard tools for formal verification. The project's strategy is to understand the BaudGen core VHDL design used in UART for serial communication capabilities. At the initial stage of the project, we tried to understand the given tools with buggy and non-buggy code. Then we synthesize the given non-buggy code using the Synthesis Design compiler. For performing equivalence checking, the synthesized code and buggy gate level code is fed into the two tools given to us to determine whether the design is Equivalent or not. If they are not equivalent, the tool will produce a test bench used for debugging. At the final stage of the project, the results are debugged, compared, and evaluated.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

1. Introduction:

1.1 Motivation:

Hardware verification is the most critical aspect of the ASIC development lifecycle, which takes nearly 70%-80% of the overall product development time. With the increased complexity of the circuits developed, the focus on hardware verification tools has also increased.

There are various methods in which hardware verification is done in the industry,

- Simulation and Emulation of the circuit/logic
- Functional verification
- Formal verification

Formal verification is somewhat new to the industry compared to other mentioned techniques. This verification method uses logical and mathematical models to exhaustively validate the design. Unlike simulation, formal methods of verification do not require any input stimuli. A mathematical model is constructed based on the requirement, a reference design is provided, and the implemented circuit is tested against it. Formal verification methods do not rely on test vectors; hence they are more suitable for verifying large circuits where it is impossible to achieve 100% coverage using simulation. Formal methods cannot be an alternative to simulation. Still, they can aid design verification by reducing the time required for verification and reducing the number of bugs before the IC is manufactured. *[1]*

In this report, we will focus on equivalence checking, a formal verification method in which the implemented design is tested against a golden design.

1.2 Objective:

Formal verification tools help us establish accuracy and bug-free design by verifying the implemented model against the formal specification written based on the specification provided. These formal specifications are mathematical and logical conditions. The verification tools use mathematical and logical techniques to compare the formal specification with the implementation. Formal verification tools are predominantly of two types model check and equivalence checking tools. Equivalence checking tools check for the equality of two circuit behaviours under all possible scenarios. They look for logical equivalence rather than just looking for the exact physical representation. *[2]*

In this project, we will be doing formal verification of the fixed frequency baud rate generator circuit. A Baud rate generator is a circuit that uses the system clock frequency (2Mhz) and divides it to get a lesser frequency baud rate (9600, 115200 etc.). Serial communication is possible only at these reduced baud rates *[3].* We will be using two equivalence checking tools, (1) Cadence conformal and (2) Synopsys formality, for verifying the system. We will also be using Synopsys design vision for synthesising the RTL level code into gate-level code. This generated gate level code will be our golden reference circuit. The gate-level buggy code will be tested against this golden design to perform equivalence checking.

1.3 Overview:

We follow specific processes before starting the formal verification of the given circuit. Firstly, we must convert the given RTL golden code to gate-level code. This process is called synthesis, where abstract system behavior is converted into the gate-level design by using specific tools. Here we use the design vision by Synopsys for performing the synthesis. The synthesis process requires understanding and analysis of the design and elaboration of RTL design *[4].* After getting the gate-level netlist for the RTL design, we must perform RTL to gate-level equivalence checking using either of the tools provided. Then we will be using Synopsys formality and Cadence conformal for Equivalence checking of the buggy code against the synthesised gate-level code. Equivalence checking is a formal verification method which verifies the two gate-level designs mathematically.

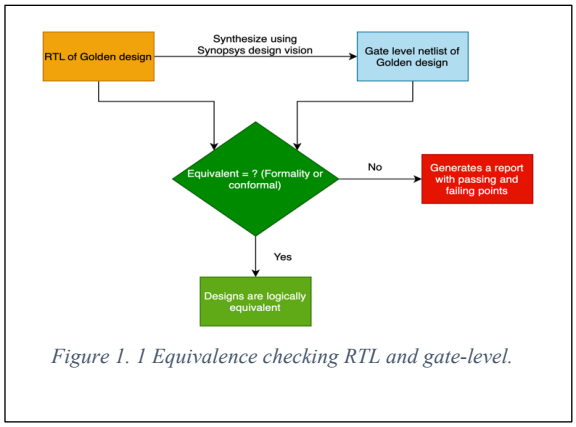


Figure 1. 1 Equivalence checking RTL and gate-level.

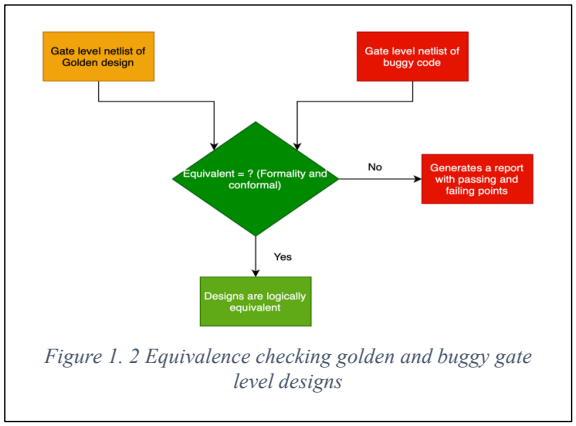


Figure 1. 2 Equivalence checking golden and buggy gate level designs

2.0 Description of system behavior and function:

2.1 Definition of UART and Baud rate

In asynchronous communication, a Baud is a unit of measurement for transmission speed. This word is frequently misused when characterising the data speeds in modern devices due to advancements in modem communication technology. Traditionally, a Baud Rate refers to the number of bits sent through the media rather than the amount of data sent from one device to the next. The overhead bits Start, Stop, and Parity, generated by the sending UART and deleted by the receiving UART, are included in the Baud count. This indicates that seven-bit words of data require ten bits to send ultimately. Figure 3 shows the Block diagram of UART serial data transmission below.

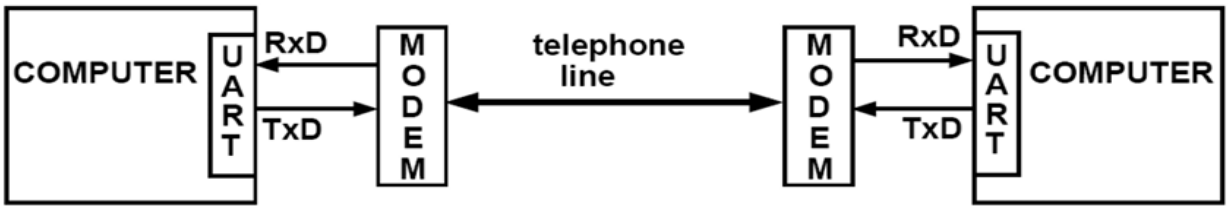


Figure 2. 1 Equivalence checking RTL and gate-level.

2.2 Baud Gen

The Baud Gen is usually used in conjunction with a Universal Asynchronous Receiver/Transmitter core to provide serial communication timing. Other applications include generating timed 'enables' for SPI, I2C, and One Wire peripherals. The BaudGen is a stand-alone system purely based on VHDL. Generates an enable pulse and n times enable pulse, both of which are 'programmed' by generics at 'compile' time. Generics allow easily specifying the baud rate (bit rate) and input clock frequency. *[3]*

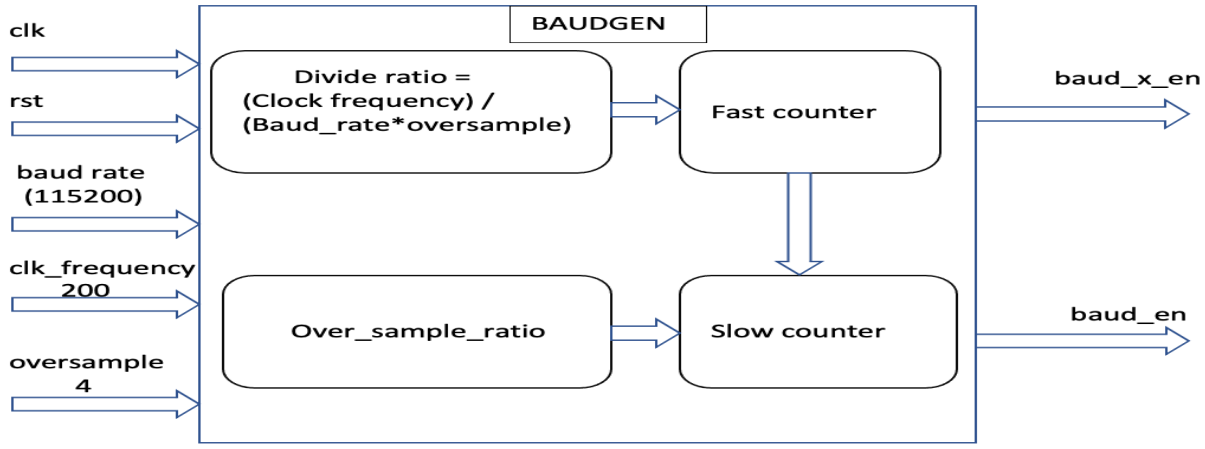


Figure 2. 2 Block diagram of Baudgen [3]

This BaudGen core is straightforward, with only two configurable counters. It's a fixed frequency Baud-rate generator because programming is done with generics. The ‘smart' part, and the motivation for building the core, is that the user doesn't have to 'think' about divider rations or setting the baud rate using a bus interface because it's all built-in. It's ideal for debugging UART ports because you know what the UART speed will be. The BaudGen can also be used to give any other timing pulses needed in a design, as a baud is another way of indicating Bit per second. Aside from the baud rate in a UART, a one us period pulse has been employed in a UART. Below Table 1 shows the port description of Baudgen core. *[3]*

Port	Width	Direction	Description
clk	1	Input	
rst	1	Input	Active High synchronous reset
baud_x_en	1	Output	Active High one clk wide enable at x times baud rate
baud_en	1	Output	Active High one clk wide enable at baud rate

Table 2. 1 IO port configuration [3]

Generic	Type	Default	Description
baudrate	int	115200	Baud rate required (in Hz)
clock_freq_mhz	real	200.0	Frequency of the clock in (in MHz)
over_sample	int	4	Number of over sample pulses per 'Bit'

Table 2. 2 Generics configuration [3]

The desired baud rate and the frequency of the clock to divide down from are defined using generics. Set over the sample to 2 and ignore the baud_x_en output if no x times output is required.

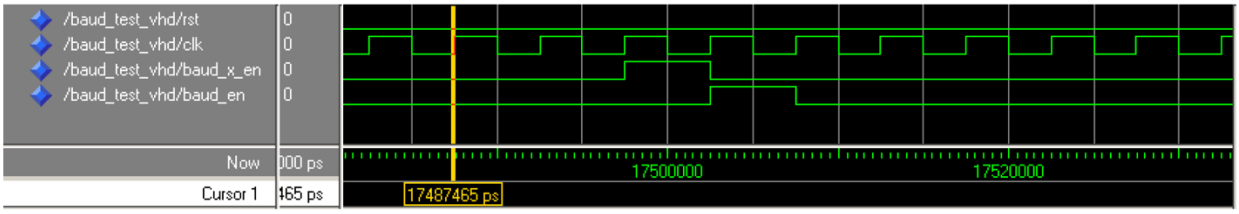


Figure 2. 3 Output graph [3]

The relationship between the two outputs is depicted in Figure 5. In this scenario, the oversample is set to 4, and for each baud_en, four evenly spaced pulses are created on baud_x_en.

3. RTL Synthesis using Synopsys design compiler:

We use the Synopsys design vision to synthesise our RTL code to gate-level code. The synthesised gate-level code will be used as the reference code for equivalence checking.

- Select the given RTL VHDL code in the design compiler by selecting the read design option.
- Elaborate RTL design by giving the required baud rate, clock frequency and over sample rate. (115200hz, 200Mhz, 4)

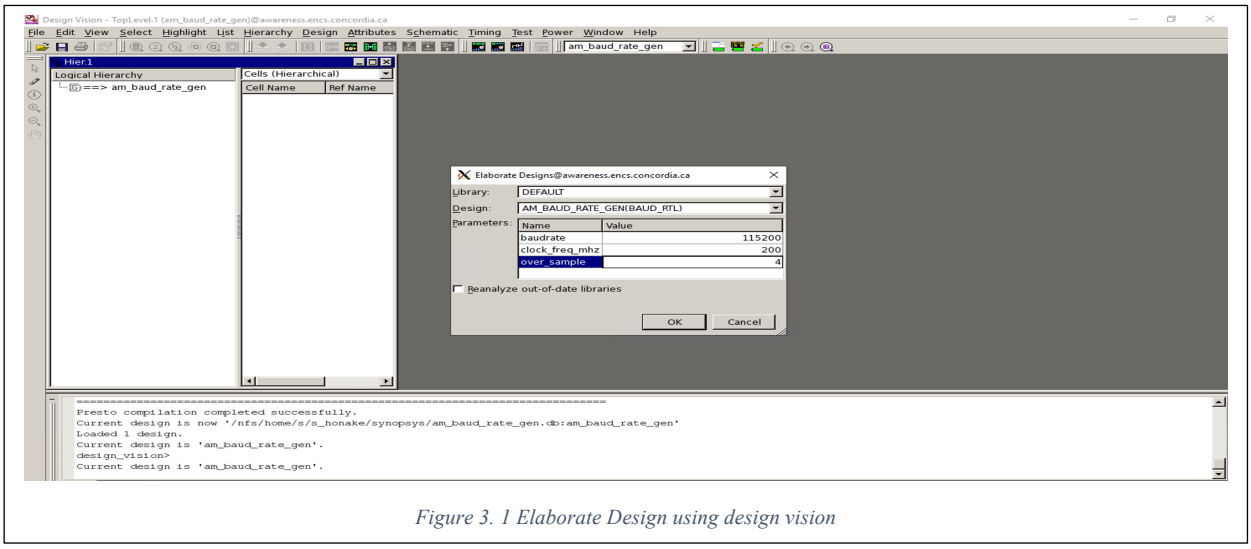


Figure 3. 1 Elaborate Design using design vision

Upon successful elaboration of the circuit, you will get the below results.

Figure 3. 2 Elaborate Design output

- Compile the successfully elaborated design using the compile design option as shown below, and the code gets successfully compiled.

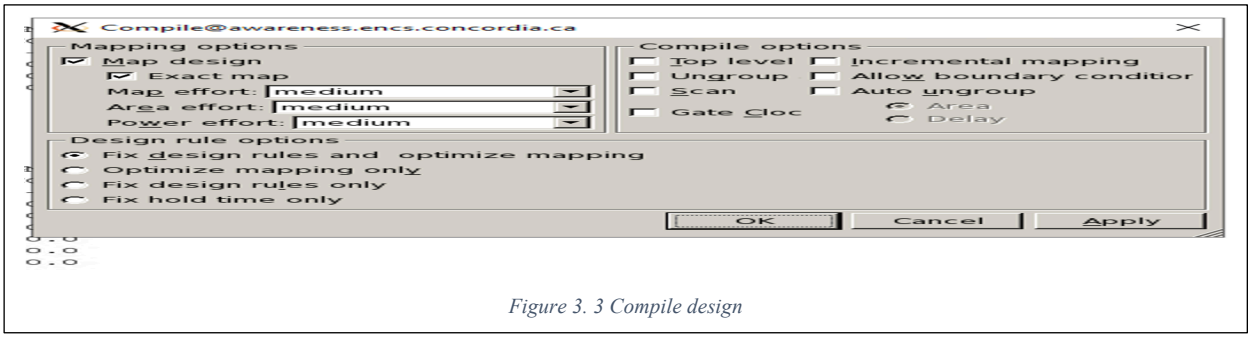


Figure 3. 3 Compile design

- Design gets compiled successfully, and optimisation is complete.

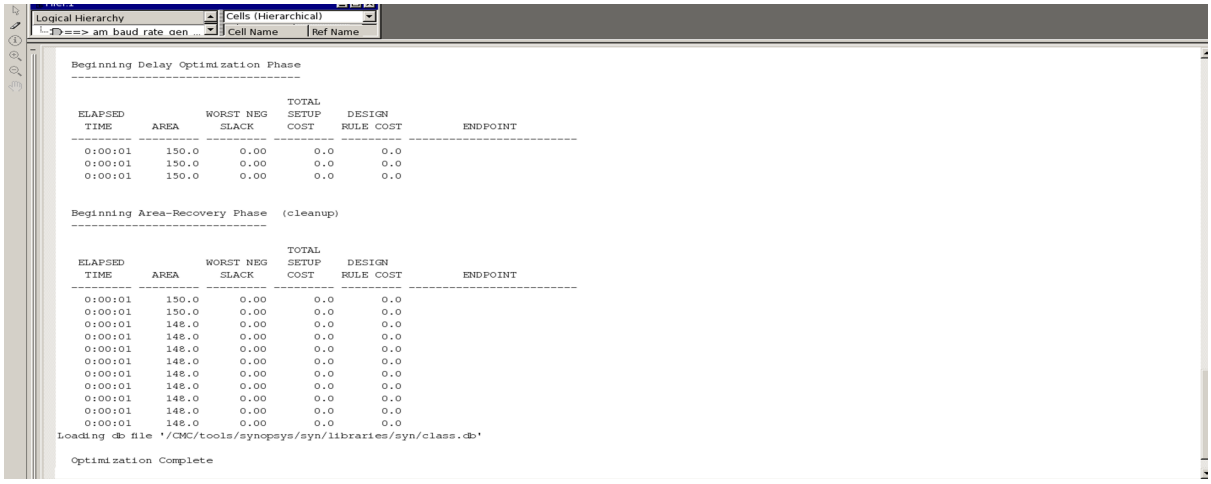


Figure 3. 4 Compile Design output

- Once the code is compiled, use the check design option.

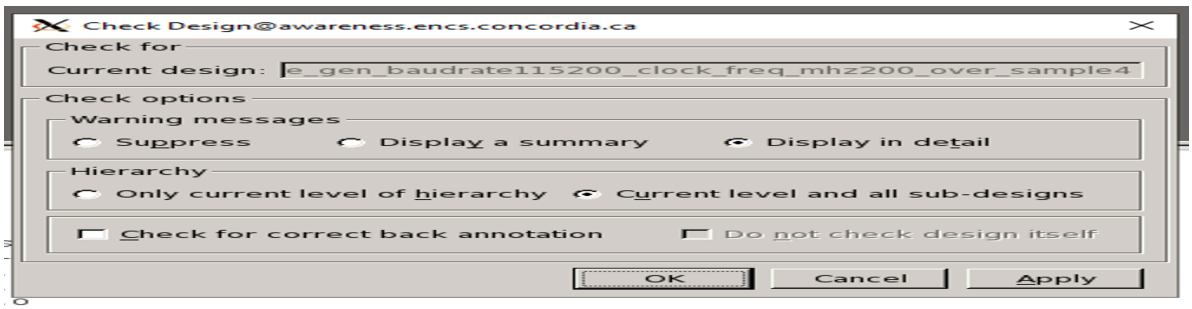


Figure 3. 5 Check Design output

- Save the gate level netlist and use this as the golden/reference design for the equivalence checking in formality and conformal.

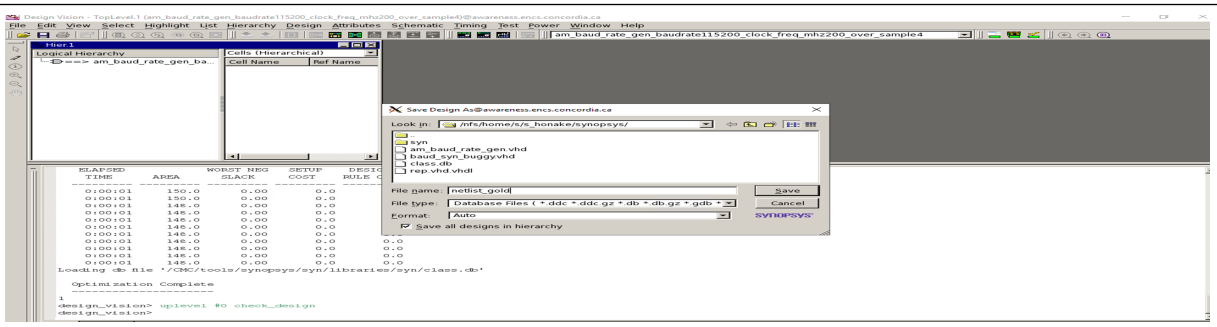


Figure 3. 6 Save reference gate-level code

4. RTL-GATE LEVEL EQUIVALENCE CHECKING

In the previous section, we synthesised RTL to gate level using the Synopsys design vision tool, which is now compared with the RTL code using the Synopsys formality tool.

For RTL to Gate level equivalence checking using the Synopsys formality tool, we followed the same steps as mentioned in section 7.

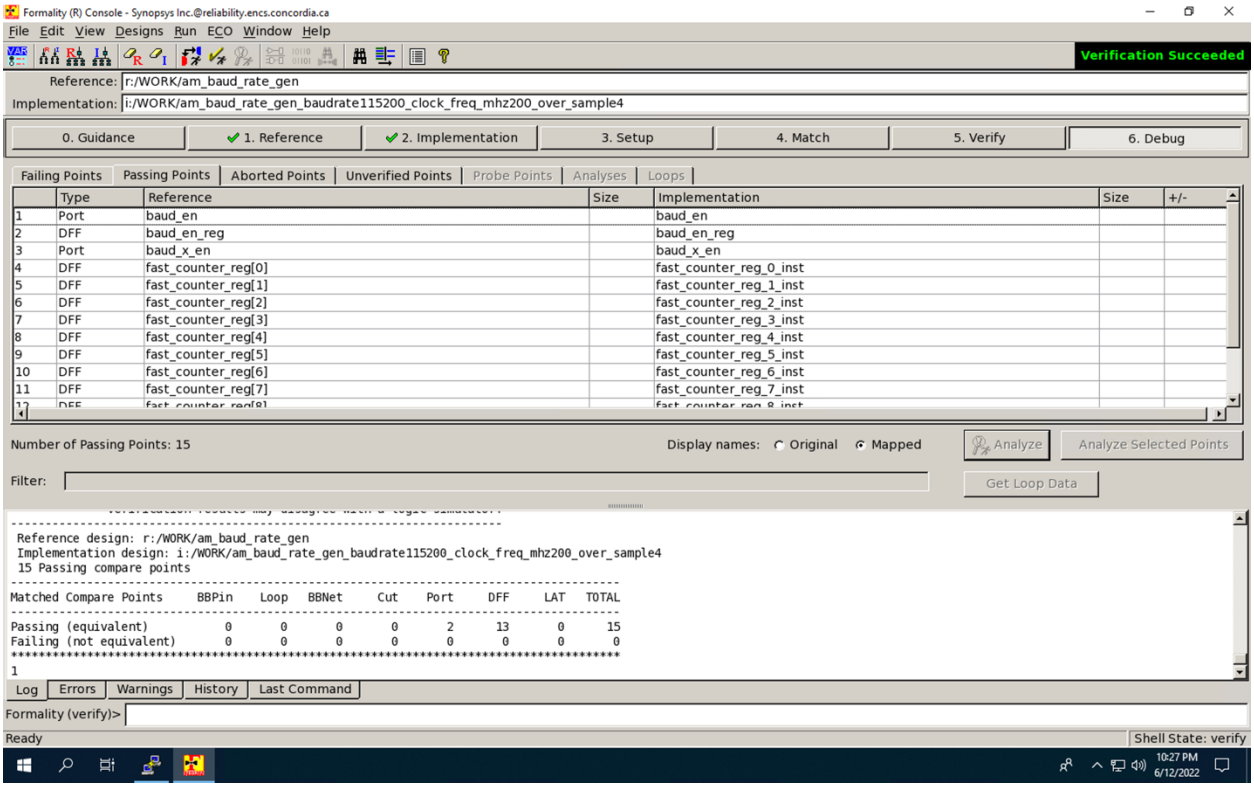


Figure 4. 1 RTL-Gate level equivalence checking

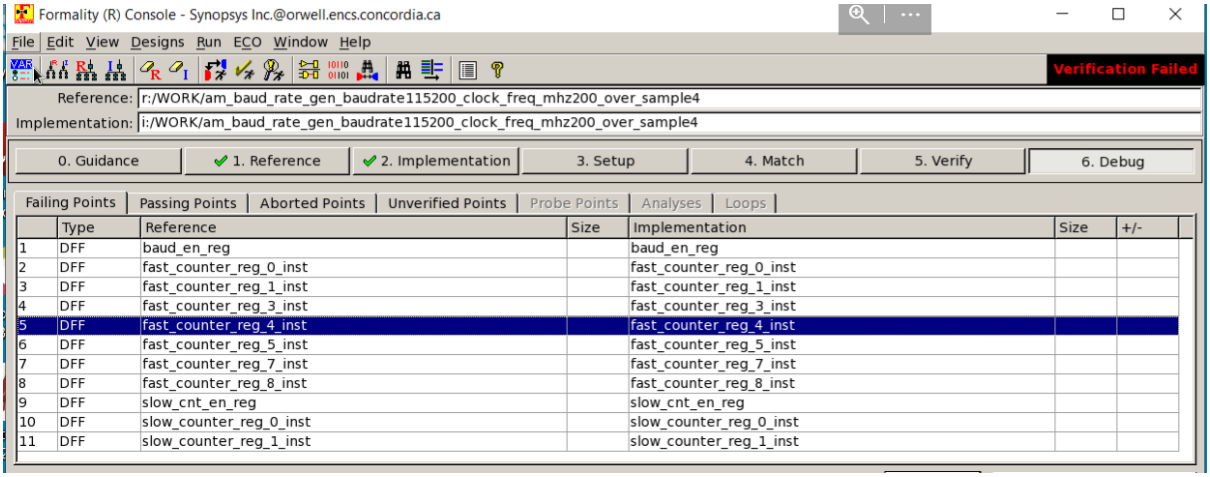
5.GATE – GATE LEVEL EQUIVALENCE CHECKING

5.1 Equivalence checking in Synopsys formality:

A formality tool is used for equivalence checking two designs given as inputs. The objective of this project is to check the equivalence of the fixed frequency baud rate generator. The gate-level codes (synthesised and buggy) are given as input in the Synopsys formality tool, as mentioned in Figure 6.2.

Below we explain how we solve the bugs found using the formality tool,

Let us take one bug and see how it is being solved by modifying the buggy code.



	Type	Reference	Size	Implementation	Size	+/-
1	DFF	baud_en_reg		baud_en_reg		
2	DFF	fast_counter_reg_0_inst		fast_counter_reg_0_inst		
3	DFF	fast_counter_reg_1_inst		fast_counter_reg_1_inst		
4	DFF	fast_counter_reg_3_inst		fast_counter_reg_3_inst		
5	DFF	fast_counter_reg_4_inst		fast_counter_reg_4_inst		
6	DFF	fast_counter_reg_5_inst		fast_counter_reg_5_inst		
7	DFF	fast_counter_reg_7_inst		fast_counter_reg_7_inst		
8	DFF	fast_counter_reg_8_inst		fast_counter_reg_8_inst		
9	DFF	slow_cnt_en_reg		slow_cnt_en_reg		
10	DFF	slow_counter_reg_0_inst		slow_counter_reg_0_inst		
11	DFF	slow_counter_reg_1_inst		slow_counter_reg_1_inst		

Figure 5. 1 Bug found after comparing using formality

We will show how bug five is solved step by step by step.

Step 1: Follow the instruction in section 6 to set up the environment and follow the below steps.

Step 2: Right-click on failing point 5 and show logic cones to see the gate-level circuit comparison between synthesised and buggy code. You will see the below diagram, Figure 5.2.

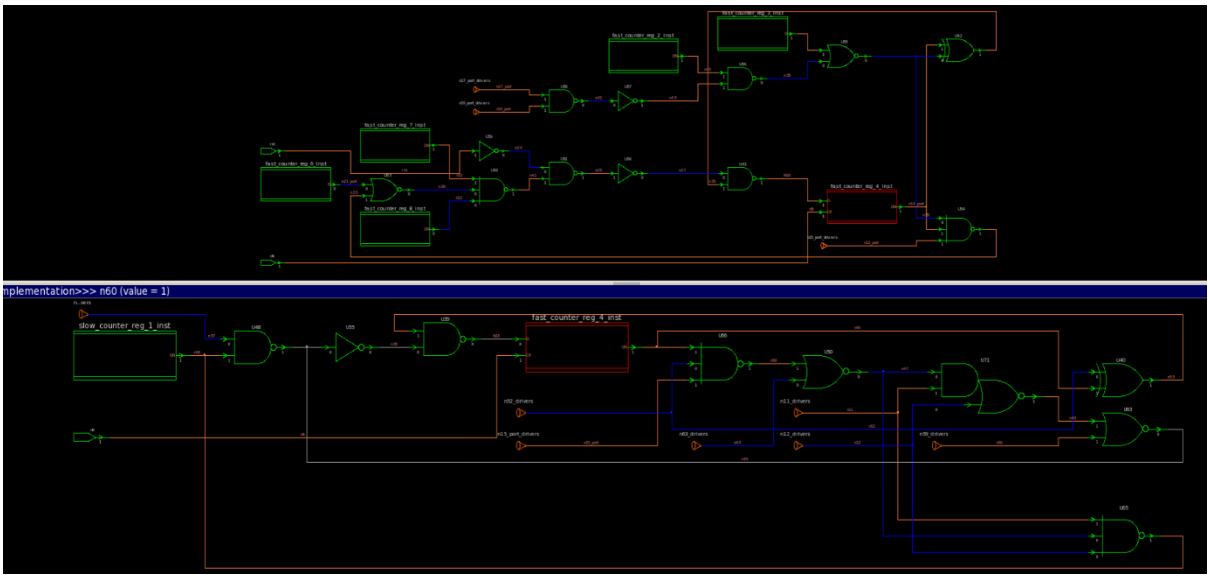


Figure 5. 2 Failing point 5

Step 3: Change the following in buggy code to fix the above bug.

Change the port connections of gate U63 (2 input NOR gate); the output port is changed from n45 to n68.

Gate U71 is changed from AO6 to ND3.

Change port connections of gate U50 (2 input NOR gate); input port B changed from n64 to n_1000.

Change U52 input port B n55 to n57.

Step 4: Save the modified code and *rerun the comparison* to fix the failing point 5.

5.2 Equivalence Checking in Cadence Conformal:

Conformal is an equivalence testing tool developed by Cadence. Cadence performs equivalence testing on golden and implemented gate-level designs.
For a detailed procedure on how to run the tool, please refer to the section; figure 7.1 refers to the unmapped and non-equivalent errors,

Below we explain how we solve the bugs found using the Conformal tool,

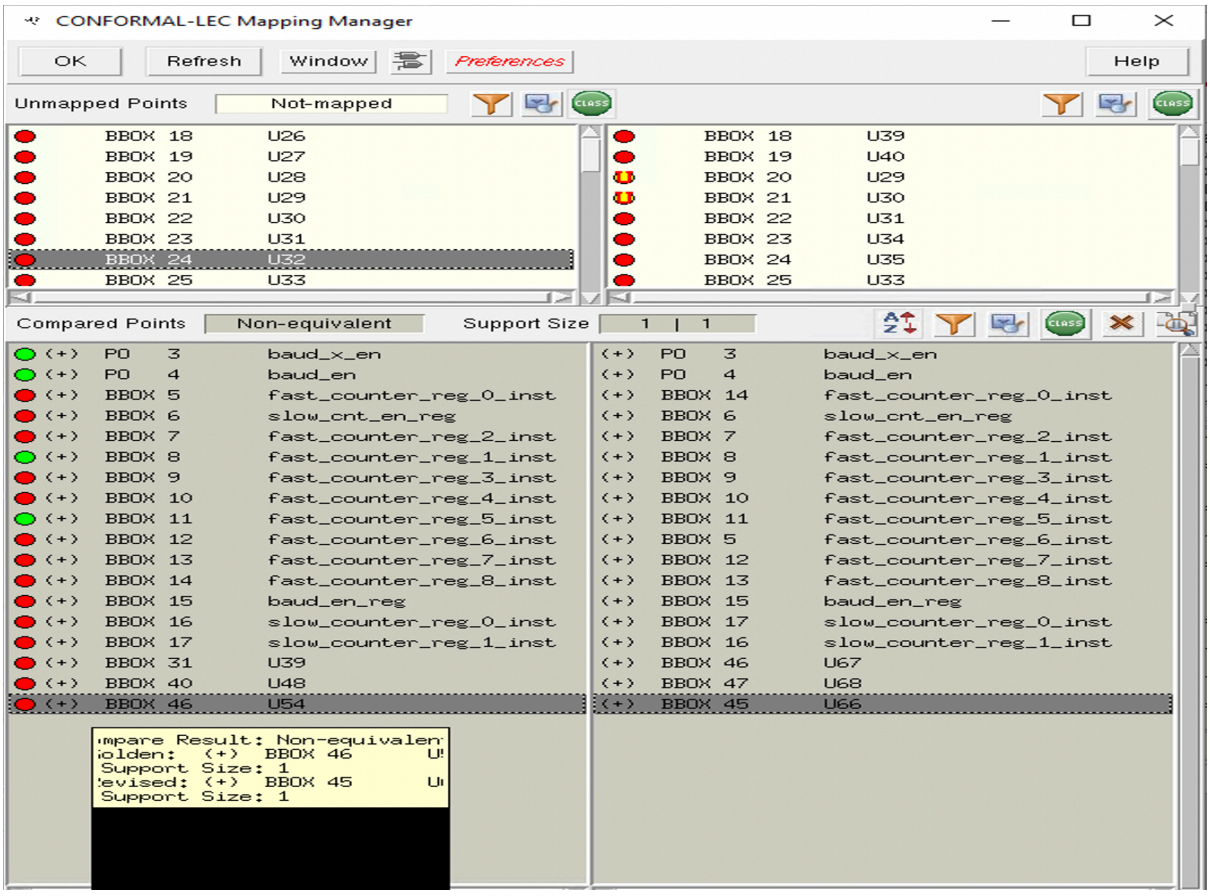


Figure 5. 3 Bugs in U54 and U66 gates

After finding the root cause of the bugs by right-clicking and using the diagnose option, we modify the source code to make the revised code logically equivalent to the golden design. In this case, U66 and U54. Figures 5.3 and 5.4 show before and after logs of bug resolution.

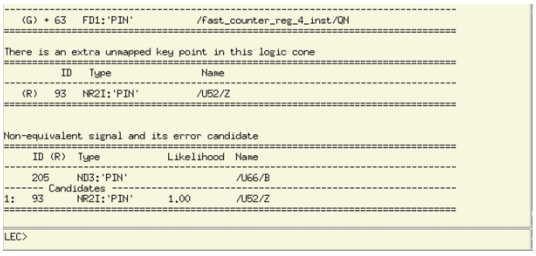


Figure 5. 4 BEFORE DIAGNOSE

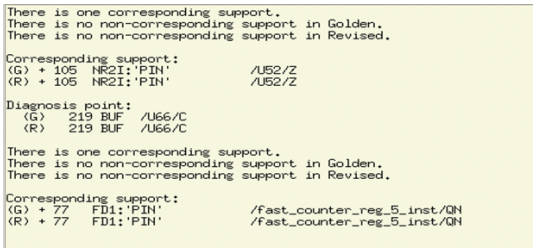


Figure 5. 5 AFTER DIAGNOSE

5.3 Compare points and bug fix:

Compare Points	Error Gate	Correct Gate	Error Port	Correct port
U31	OR2	NR2I	-	-
U41	-	-	A=> n54, Z=> N17	A=> n37, Z=>n54
U45	-	-	B=>n17_port	B=> n35
U47	ND2I	AN2I	B=>n16_port	B=>n17_port
U48	-	-	B=>n60	B=>n46
U50	-	-	B=>n63	B=>n_1000
U52	-	-	B=> n55	B=> n57
U53	-	-	B=>n16_port	B=>n_1007
U54	-	-	A=>n56	A=>59
U26	OR2	ND2I	-	-
U62	NR2I	OR2	B=>n46	B=>n81
U63	-	-	Z=> n45	Z=>n68
U64	-	-	A=>n55, B=>n65, C=>n52, Z=>n54	A=>n23_port, B=>n71, C=>n72, Z=> N10
U65	-	-	Z=>n60	Z=>n69
U68	-	-	A=>n17_port, B=>n16_port	A=>n_1007, B=> n17_port
U70	-	-	A=>n63	A=>n_1000
U71	AO6	ND3	-	-
U72	AO3	AO6	A=>n43, B=>n67, C=>n44, D=>n37, Z=>n19_port	A=>n47, B=>n11, C=>n12, Z=>n81

Table 5. 1 Compare points and bug fix

6. DESCRIPTION OF VERIFICATION PROCESS IN SYNOPSYS FORMALITY

6.1 EQUIVALENCE CHECKING USING SYNOPSYS FORMALITY

Formality detects any unexpected variations that may have been introduced into a design during development. It employs a formal verification comparison engine to prove or refute the equivalence of the two provided designs and present any differences for further analysis and diagnosis. The comparison engine uses mathematical algorithms to compare reference design and implementation design and gives the visual implementation to diagnose the failure scenarios. The figure shows a basic view of the formality tool and its functionality, as discussed in this section. *[5]*

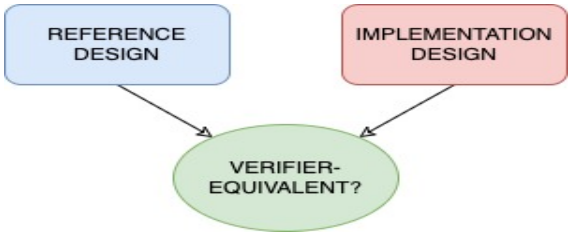


Figure 6. 1 Generic view of Synopsys formality *[5]*

The formality tool can be used in both GUI and command mode. The flow chart represents the flow of LEC in the Synopsys formality tool. A brief view of each step will be discussed later in this section. *[5]*

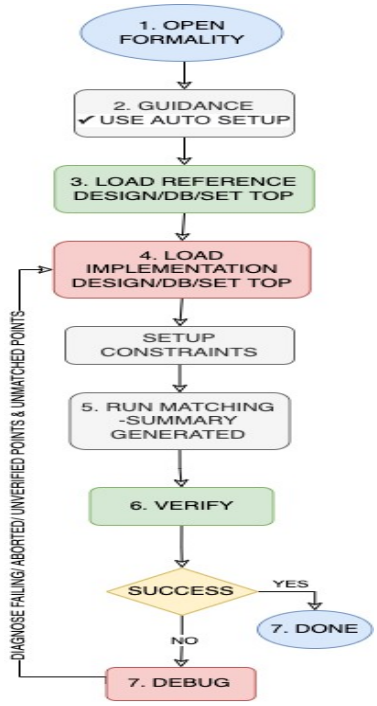


Figure 6. 2 How to use Synopsys Formality [7]

6.2 LEC using Synopsys formality

6.2.1 Environment Setup and open formality

1. Start Xming and initialise putty to work on CAD tools.
2. Create a formality folder locally using the Linux command “mkdir formality” and point formality as the current work folder “cd formality”.
3. Move all the project files to the formality folder.
4. Create an environment using the command “Source /CMC/ENVIRONMENT/ formality.env”.
5. Instantiate the tool by using the command “formality &”.

6.2.2 Guidance

The tool recommends the setup files “. svf” in this step, though it is optional. It eliminates the need to manually enter the setup information, which is time-consuming and error-prone. This is required for the successful verification of the design. To do that, we need to check the auto-setup check box and proceed to the next step.

6.2.3 Loading reference and implementation design

The tool requires setting design files for reference and implementation in .vhd format and library files in .db format. We load reference and implementation files in a separate section. These two sections are compared using the LEC engine for equivalence.

6.2.4 Match

In this section, the LEC engine gives the compared outputs of the design files and categorises the comparisons as matched, unmatched summaries. Here in matching, both designs are matched to verify the implementation design.

6.2.5 Verifying

On verification, formality sets the statuses in one of the following passing, failing, aborted, unverified, and not verified. The verification analysis uses logic cones and patterns compared against passing and failing points.

6.2.6 Debugging

After the verification report, we need to trace back to the source by diagnosing that.
Repeat the debugging process by resolving the logical inequalities until the verification is successful. *[5]*

6.3 Why Synopsys formality...?

Synopsys Formality is an important equivalence testing tool for formal verification. It also provides greater predictability. The system has fewer chances of missing defects, higher quality

because the system can highlight destructive RTL design code, and more confidence and test coverage because the system can highlight poor RTL design code. Also, Synopsys is simple and easy to use with auto-setup mode. *[7]*

7. DESCRIPTION OF VERIFICATION PROCESS IN CADENCE CONFORMAL

7.1 EQUIVALENCE CHECKING USING CADENCE CONFORMAL

Cadence Conformal enables the verification and debugging of multimillion–gate architectures without needing test vectors. Conformal compares two circuits, and capacity restrictions mean they won't make false negatives or abort without finding proof now and again.*[6]*

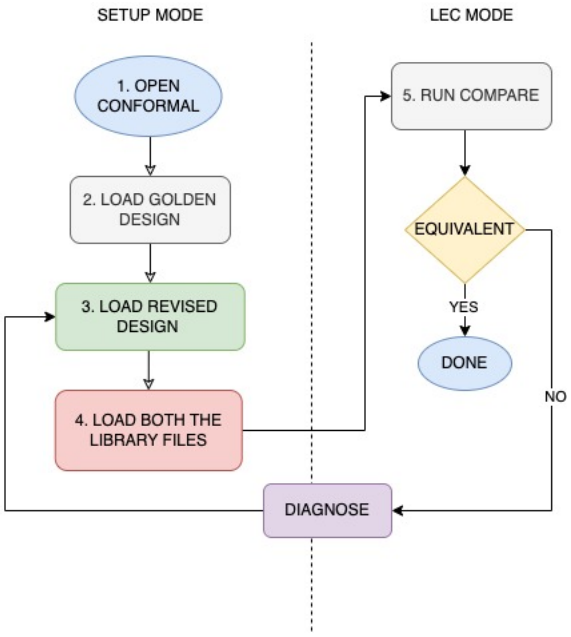


Figure 7. 1: How to use Cadence Conformal [8]

7.2 LEC using Cadence Conformal

7.2.1 Environment setup

1. Start Xming and initialise putty to work on CAD tools.
2. Create a formality folder locally using the Linux command “mkdir conformal” and point formality as the current work folder “cd Conformal”.
3. Move all the project files to the Conformal folder.
4. Create an environment using the command “Source /CMC/ENVIRONMENT/ conformal.env”.
5. Instantiate the tool by using the command “LEC -XL”. *[6]*

7.2.2 Loading library

In this process, we load the library by following the below-mentioned steps,

1. Go to File -> Read Library. A Read library window will appear.
2. In the Library option, select the format as liberty and type as both. Select the class.lib file -> Add selected -> OK. *[6]*

7.2.3 Loading Golden and revised design

In this step, the reference, and the implementation code are loaded to Conformal. The following step should be completed in the setup mode.

1. Go to File -> Read Design. A Read design window will appear.
2. To load the Golden code, select the format as VHDL and type it as golden. Then select the synthesized code -> Add selected -> OK.
3. To load the Revised code, select the format as VHDL and type as revised. Then select the buggy code -> Add selected -> OK. *[6]*

7.2.4 Compare

Once the setup is completed, change the mode to LEC. A small window will appear at the bottom of the screen showing all the unmapped points. Then go to Run -> compare. In the compare window select add all compare points. The total number of non-equivalent points will be shown in the LEC window. *[6]*

7.2.5 Report

This step is followed to debug the non-equivalent points. Go to report -> mapped points. This will open the Conformal LEC-mapping manager. Go to Preferences-> compared points on.

7.2.6 Diagnose/Debug

Right-click on the non-equivalent points shown in the compared points section of the LEC mapping manager and select the diagnose option. This will open the LEC diagnosis manager and give us the diagnosis point for both the golden and the revised code. This point can be used to resolve the bug. *[6]*

7.3 Why Cadence Conformal …?

Cadence Conformal is one of the most used tools for equivalence checking around the globe. Conformal allows us to detect essential bugs a lot faster in comparison to other tools that are available. *[8]*

8. Analysis of Verification Results and Conclusion

8.1 Summary

Initially, verification failed in formality. We had 11 failing points and two passing points on the repeated diagnosis. By following the steps in section 6, we were able to debug and make those failing points into passing points. The Figures 8.1, 8.2, 8.5, 8.6 show the logs of the formality tool before and after resolving bugs.

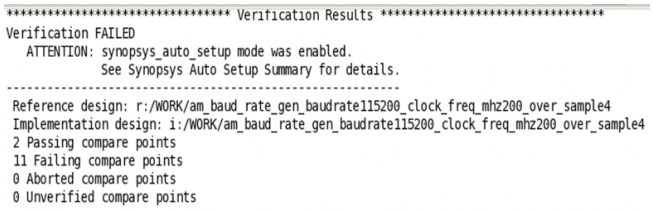


Figure 8. 1 Formality log before verification

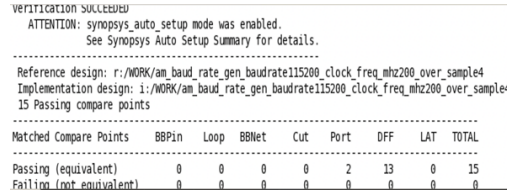


Figure 8. 2 Formality log after verification

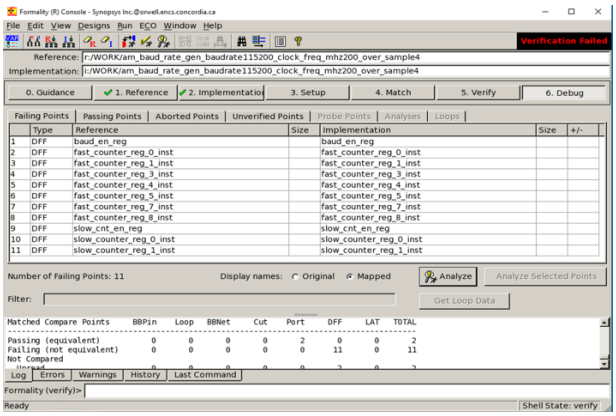


Figure 8. 3 Verification failed in initial analysis

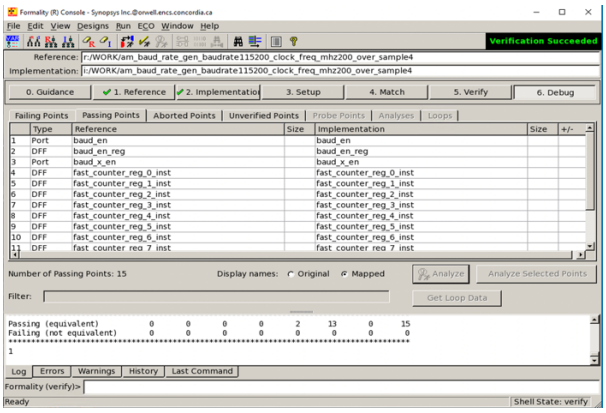


Figure 8. 4 Verification successful after analysis

In the conformal tool number of unmapped scenarios is five, and non-equivalent scenarios are thirteen in the revised file compared with the golden file, following the steps in section 7. Multiple reports could be generated conformally for individual scenarios like an unmapped black box, etc. The most common bug found is logically inequivalent.

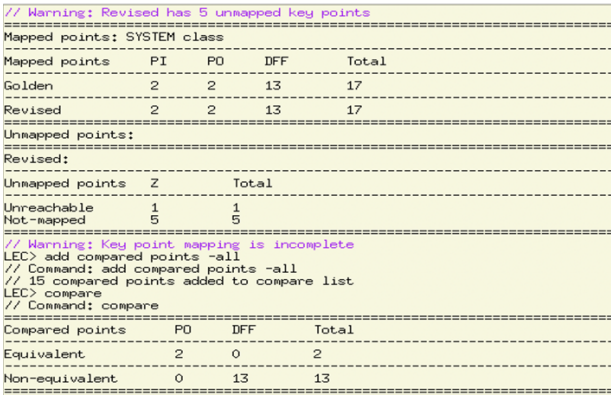


Figure 8. 5 Conformal log before comparison

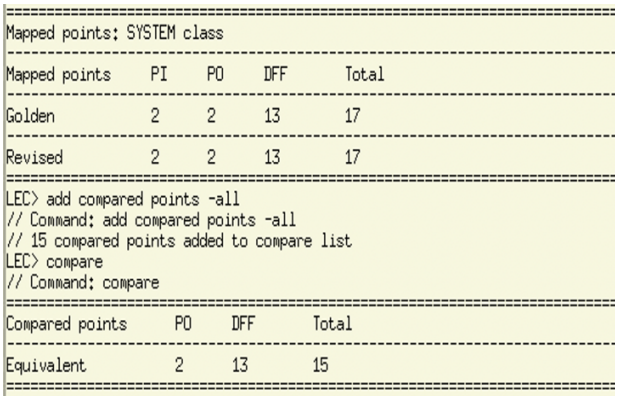


Figure 8. 6 Conformal log after comparison

8.2 Discussion

Based on the results we obtained from the above tools. The formality tool helps us show the patterns and logic cones which were easy to debug. In the conformal tool, we wanted to compare the source code of two design files. In both cases, once the reference design is matched with implemented design, the debug process is more straightforward.

8.3 Challenges

List of challenges faced while working on the project,

1. One of the most complex parts was the time management issues.
2. Initially, we were working from a different point of view. On debugging, all our bugs went into aborted points instead of passing, which we figured out in later stages. So, our understanding and functionality of the tools were very limited in the earlier stage.
3. The personal laptop was hard to use file debugging. Later, we worked on the whole project in-lab computers.
4. Trying to match pattern flow was hard, but we learnt how to trace back to the issues.
5. In conformal, comparing source codes and using the tool was challenging.

9. References:

1. *Design verification by Charles E. Stroud, Yao-Wen Chang, in Electronic Design Automation, 2009, Weblink: https://www.sciencedirect.com/topics/computer-science/design-verification, Accessed on: June 9th 2022.*
2. *How much verification is necessary? – Blog by Semiengineering, Weblink: https://semiengineering.com/how-much-verification-is-necessary/, Accessed on: June 9th 2022.*
3. *"BaudGen.pdf", Andrew Mullock, 26-Nov-2007, Viewed on: Accessed on: June 9th 2022.*
4. *Tutorial: Formality and Conformal equivalence checking, "dc-slides16.pdf", Weblink: https://moodle.concordia.ca/moodle/mod/folder/view.php?id=3127036, Accessed on: June 9th 2022.*
5. *Tutorial: Formality and Conformal equivalence checking, "formality-slides16.pdf", Weblink: https://moodle.concordia.ca/moodle/mod/folder/view.php?id=3127036, Accessed on: June 9th 2022.*
6. *Tutorial: Formality and Conformal equivalence checking, "lec-slides16.pdf", Weblink: https://moodle.concordia.ca/moodle/mod/folder/view.php?id=3127036, Accessed on: June 9th 2022.*
7. *Synopsys Formality datasheet, Weblink: https://www.synopsys.com/content/dam/synopsys/verification/datasheets/formality-and-formality-ultra-ds.pdf, Accessed on: June 9th 2022.*
8. *Conformal equivalence checker data sheet, Weblink: https://www.cadence.com/content/dam/cadence-www/global/en_US/documents/tools/digital-design-signoff/conformal-equivalence-checker-ds.pdf, Accessed on: June 9th 2022.*