Concordia University
Engineering and Computer Science

# Formal verification of I2C design using Equivalence checking tools

# CEON 6551: Formal Hardware Verification

## Introduction

There are multiple methods for ensuring the correctness of a system design, with Simulation and Formal Verification being the most used approaches.
Simulation-based verification involves creating a series of test cases by the designer or verifier, based on the design's specifications.
On the other hand, Formal Verification rigorously validates the design's functionality against all possible inputs using mathematical proofs. Formal verification has the advantage of detecting bugs that may not be identified by traditional verification techniques.
Moreover, it can identify these bugs at a faster pace, even before the design is ready for simulation and emulation-based verification.

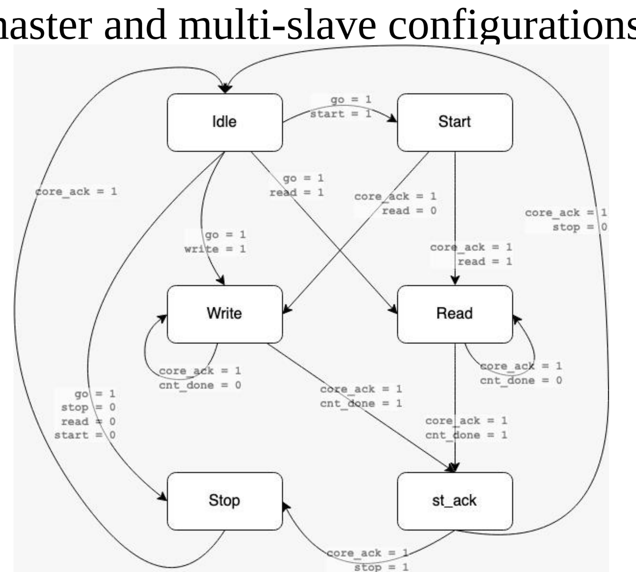
## System Behavior

I2C is a serial communication protocol used to connect components in a system. It requires only two wires: a data line (SDA) and a clock line (SCL). Devices on the I2C bus can communicate with each other using unique addresses. It supports multi-master and multi-slave configurations, enabling flexible system architectures.

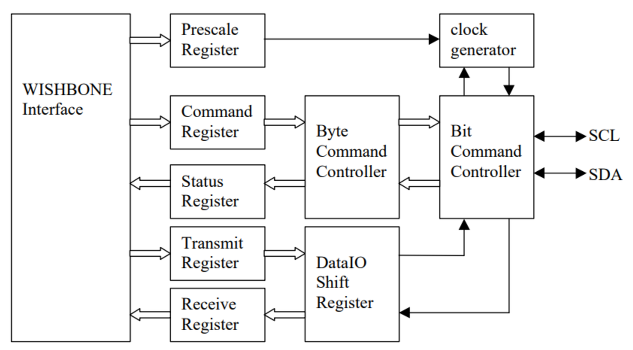
The code consists of an entity declaration (i2c_master_top) and its corresponding architecture (structural). The architecture includes a component declaration (i2c_master_byte_ctrl), instantiated as byte_ctrl, which represents the I2C protocol's byte controller.

The i2c_master_byte_ctrl component manages byte-level operations in an I2C communication system and acts as an interface to other system components. It utilizes a state machine to handle various operations such as starting and stopping processes, sending and receiving data bytes, generating acknowledgements, and shifting data within a register. This component relies on the i2c_master_bit_ctrl bit-level controller for lower-level I2C instructions.

The i2c_master_bit_ctrl component contains procedures for generating signals, decoding registers, and controlling the state of the I2C master.



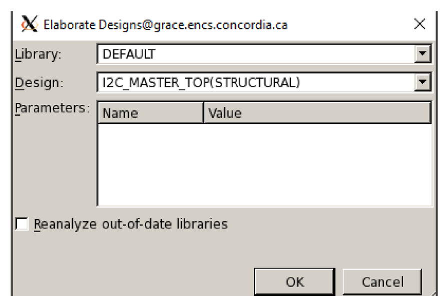
State Diagram



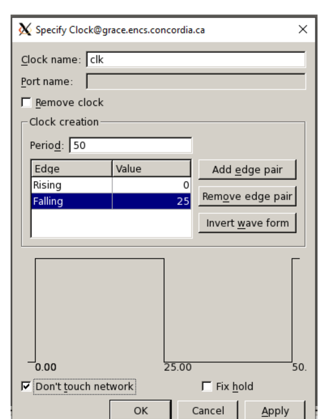
I2C Architecture

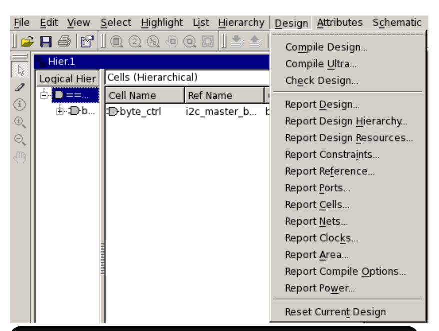
## RTL Synthesis using Synopsys design compiler

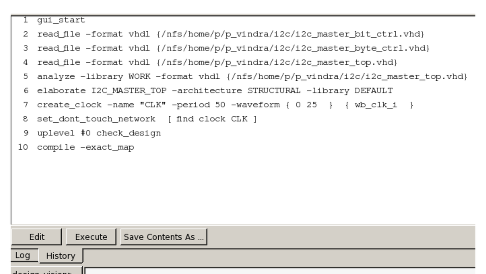
Design Compiler, developed by Synopsys, is a synthesis tool that converts an RTL design written in HDL into an optimized gate-level representation. It provides a graphical interface for exploring and analyzing the design's behavior at the gate level.
Design Compiler employs optimization techniques such as Timing, Power, Area, and Datapath optimization to enhance the design's efficiency.
Step are involved to use Synopsys tools are as follows:
Read all design files and analyze the top file.
Elaborate the design and then check the design.
Then specify the clock.
Check and compile the design.


Elaborate design selecting the top module


I2C core


Specify Clock


Check and Compile the design


Design Vision History

## Equivalence checking using Synopsys Formality

Synopsys Formality is a formal verification tool used to verify equivalence between at RTL to Gate level and Gate to Gate level designs.

**Verification steps: -**

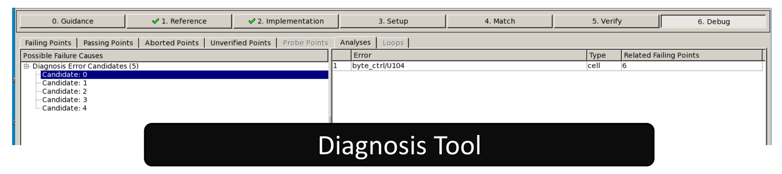
1. **RTL to Gate Level verification:** Synthesized design from Synopsys design compiler is loaded in Implementation design, and RTL file loaded with the library in the Reference file. Then Matching is done to check and map designs. In last, Verifying and Debugging are done to evaluate errors/nonequivalent points
2. **Gate-Gate Level verification:** The synthesis file is loaded in the implementation container for equivalence checking and in reference another Synthesis file (Verified) is loaded. Apart from that, all steps are the same as above.

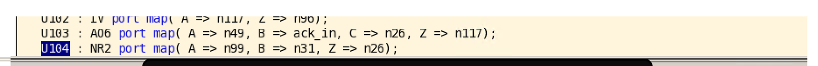
**Result of RTL to Gate level and Gate to Gate level: -**

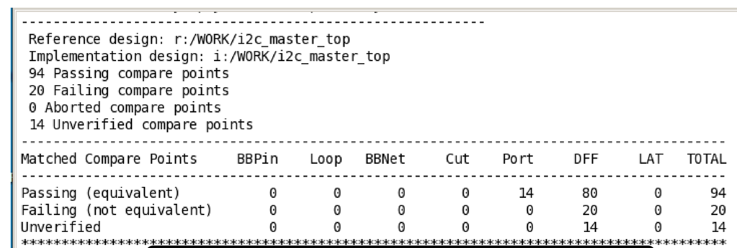
**1. Verification RTL and Gate level**: RTL vs Gate level comparison is done when Synthesized file and RTL files are compared. As a result, it was succeeded, and they are equivalent and the synthesized file is correct.

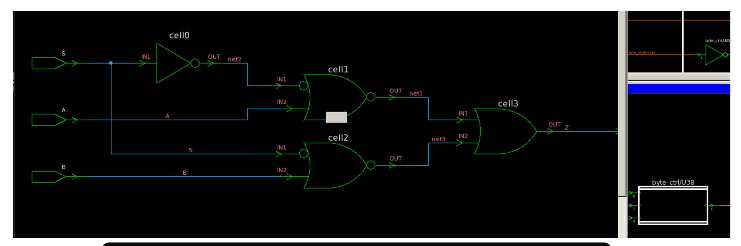
**2. Verification of synthesized and buggy design:** verification fails when we try to run equivalence and produces 34 non-equivalent points in total.
Bug tracking and bug resolving: In this process where both designs were compared for all bugs and find the source of the bug using schematic logical comparison or using Diagnosis tool of Formality.


Overview of Formality


Diagnosis Tool


Tracking Error in file


Console


Expanding the component

## Equivalence checking using Synopsys Conformal

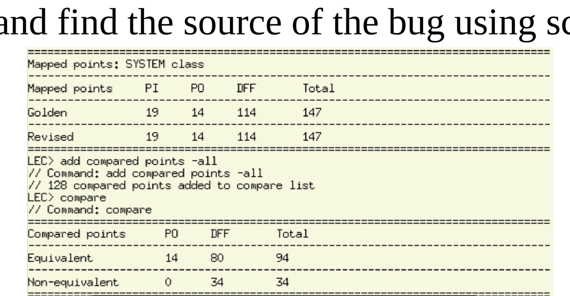
Cadence Conformal is a formal verification tool used to verify equivalence between at RTL to Gate level and Gate to Gate level designs.

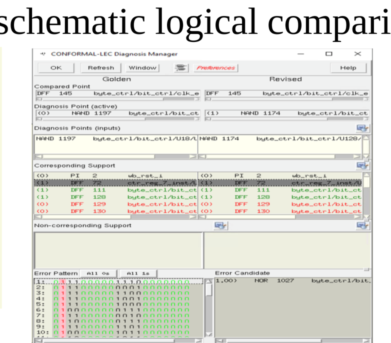
**Verification steps: -**

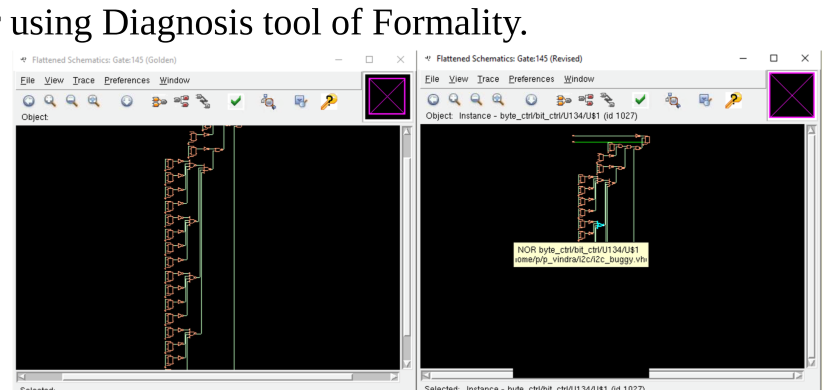
1. **RTL to Gate Level verification**: Synthesized design from Synopsys design compiler is loaded in the Revised file, library file loaded as a Both type and RTL file loaded in the Golden type. Then, first change mode setup -> LEC and after Run and Compare the design. Lastly, Check the Mapping manager to find non-equivalent points.
2. **Gate-Gate Level verification:** The synthesized (Verified) file is set as a Golden file, another synthesized file loaded as a Revised, and a Class file is loaded as a both. Apart from that, all steps are the same as above.

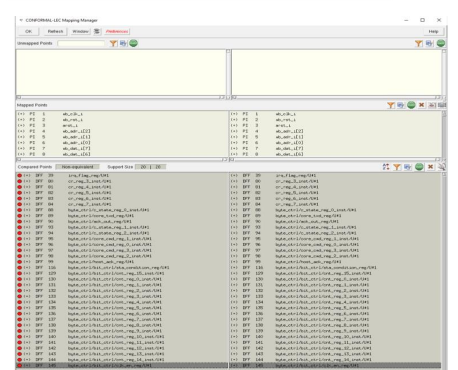
**Result of RTL to Gate level and Gate to Gate level: -**

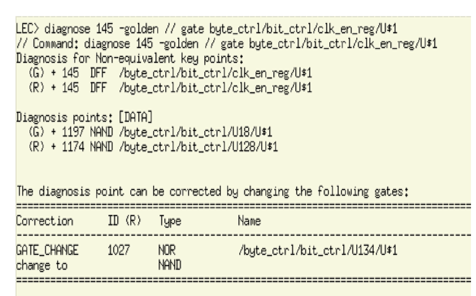
1. **Verification RTL and Gate level:** RTL vs Gate level comparison is done when Synthesized file and RTL files are compared. As a result, it was succeeded, and they are equivalent and the synthesized file is correct.
2. **Verification of synthesized and buggy design**: verification fails when we try to run equivalence and produces 34 non-equivalent points in total. Bug tracking and bug resolving: In this process where both designs were compared for all bugs and find the source of the bug using schematic logical comparison or using Diagnosis tool of Formality.


Console


Error Candidate


Schematic of circuit


Mapping Manager


Description of Error

## Analysis of result

We have used two software for verification purposes, Synopsys Formality, and Cadence Conformal. In Formality, after setting the gate level synthesized design as reference and gate level buggy design as implementation, schematics of both were compared for rectifying the bugs. In formality the total number of thirty-four non-equivalent points. As we progressed with bug resolving, we were able to resolve 33 bugs with few tweaks but unfortunately at the end, we got stuck with one bug.

In the Conformal tool, once the design files are set in the golden and revised respectively, diagnosis were performed. The total number of non-equivalent points were the same thirty-four as well. They were diagnosed by comparing the schematics and following the Conformal recommendation in certain errors. We were able to solve 33 out of 34 non-equivalent points were only able to be resolved and we again got stuck with the same bug.

## Comparison of tools

| Tool/Feature | User Friendly | User Interface | Evaluation | Schematic Wise | Operation Time |
|---|---|---|---|---|---|
| Synopsys Formality | Most Preferred | Most Preferred | Least Preferred | Most Preferred | Most Preferred |
| Cadence Conformal | Least Preferred | Least Preferred | Most Preferred | Least Preferred | Least Preferred |

## Conclusion and Challenges

**Conclusion**: The main goal of this project was to use Equivalence Checking, a formal verification method, to verify the correctness of two designs at different levels of abstraction. Equivalence Checking is typically employed during Implementation Verification. The process begins by synthesizing the RTL design using Synopsys Design Vision software, which generates a Gate level netlist. To validate the synthesis process, the gate level netlist is then compared with the RTL netlist in Synopsys Formality and Cadence Conformal using Equivalence Checking. Once this verification is successful, the next step involves applying Equivalence Checking between the synthesized gate level netlist and the Buggy gate level netlist. This step aims to identify any bugs or discrepancies, as both netlists are at the same level of abstraction.

**Challenges:** In our case, we were not able to attain 100% verification success using formality as well as conformal. We failed to resolve one bug in the end, but we tried to analyze the bug in numerous ways and during that process, we were able to understand the tools better and got familiarized about the concept of equivalence checking in depth. In future, we are that with the experience gained during this whole project can come handy at any point and be helpful in future learning as well.

## References


1. Killer software: 4 lessons from the deadly 737 MAX crashes https://www.fierceelectronics.com/electronics/killer-software-4-lessons-from-deadly-737-max-crashes
2. N° 33–1996: Ariane 501 - Presentation of Inquiry Board report https://www.esa.int/Newsroom/Press_Releases/Ariane_501_-_Presentation_of_Inquiry_Board_report
3. Prof Dr. S. Tahar, COEN 6551 Formal Hardware Verification, Lecture notes, Summer term 2023.
4. I2C Master Core Description https://opencores.org/projects/i2c



Contributor: Rushik Shingala (40221905) Prithvik Adithya Ravindran (40195464) Behnam Shakibafar
Submitted to: Dr. Sofiene Tahar