

FORMAL PROPERTY VERIFICATION OF COUNTER FSM AND I2C

SNEHA S¹, ROOPA G²

¹ PG Student, Dept. of Electronics and Communication Engineering, Nagarjuna College of Engineering, Bengaluru Karnataka

Email: sneha44enz@gmail.com

² Assistant professor, Dept. of Electronics and Communication Engineering, Nagarjuna College of Engineering, Bengaluru, Karnataka

Email: roopa.gurijepalli@gmail.com

Abstract - In the present days, due to the increased complexity of multimillion gates such as complex SOC, ASICs and DSP processors need to be verified effectively. About 70% of the VLSI design efforts to consumer verification. Some of the front end technologies are RTL functional verification, test bench verification coverage techniques, etc. are very difficult to design closure. These may not cover all the corner cases.

By considering the above concerns, formal verification has become more important to verify the complex circuits and protocols. Model checking verifies temporal logic in formal verification. One can check the perfections of circuits or designs using HDL's. Verification Interacting with Synthesis (VIS) is a tool. The tool is to implement model checking properties on up-counter and I2C protocols.

Key Words: Counter FSM, I2C protocol, VIS, Model checking, CTL property verification using VIS

1. INTRODUCTION

Verification process plays very important role in the chip designing industries, companies etc. Due to the increased complexity of multimillion gates such as complex SOC, ASICs and DSP processors, it needs to be verified effectively before the chip production in fabrication labs (FAB labs). The main goal of verification is to check and correct the defects in circuits, errors in programs and finally to get the defect free products. The term verification means "It is the process which is used to reveal that the purpose of design is stored in its application".

1.1 General Digital Circuit Design Flow

Fig-1 shows the sequence to design a digital circuit and we call this as design flow. The steps are listed below with a brief description and function of each. [2]

- Concept and market research: This gives the complete information about the design structure and it embodies the system requirements.
- Architectural specs: This mainly concentrates about the elements and components of the system.

- Floor planning: It identifies the placements of structures in a system and it allocates the space for each structure to meet some effective goals in a system.

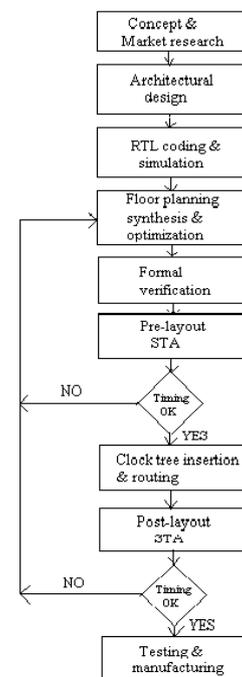


Fig-1: Digital circuit design flow

- Physical design: Here components are in the form of circuits (interconnections and systems) and those are changed to geometric representation of each circuit of design.
- Optimization: In this step removal of Wire Load Models (WLM) are taking care.
- Scan insertion: This step helps in verifying the circuit by designing the testing part or by simulation.
- Formal verification (RTL v/s Gates): The design of simple chip and also complex chip is verified by writing program in the form of properties.

- Pre layout STA: In this step it performs front end net-list for STA and also routing delay is calculated.
- Routing: In physical design process, there are two types of routing. Local routing assigns wiring to particular layer of the metal. Global routing which allocates wiring globally.
- Post layout STA: During this step, it provides back end net-list for STA. Comparative
- Testing and Manufacturing: This is the final step in block design flow of digital circuit.

There are two techniques in formal verification. Those are equivalence checking and model checking. Compare to the equivalence checking, model checking is more effective in real time projects verification. By accepting the temporal logic, writing the CTL properties on selected protocol using model checking concept also by using VIS tool version 2.4 and performing the complete verification of selected digital circuits like counter-FSM and I2C protocol is the main aim in this project, it will cover all the corner cases.

2. OVER VIEW OF VERIFICATION

These days, in the era of ASICs and System on Chip (SOC) models, verification plays very important role. Hence some of the core companies and industries are hiring more number of engineers to verification domains rather than development domains. [3]

2.1 Verification Types

There are 4 types of verification which are mentioned below.

- Static timing verification: It is a type of verification which uses static timing analysis (STA) and lint checking.
- Physical design based verification: It verifies certain criteria, performs after routing. Layout verification is done by using physical verification. It checks for Design Rule Check (DRC), Electrical Rule Check (ERC), Layout Versus Schematic (LVS), antenna check and XOR check.
- Functional verification: It verifies the design as its intended work is done.
- Formal verification: The formal properties are written to verify the chip design. In formal verification, there is no need of writing testbenches. Instead need to write some properties for the design to verify it.

In equivalence checking, there is a comparison between the two models have been takes place. In equivalence

checking, always it maintains the synthesis part as honest tool. It can be easily find out the actual problem in the design.

In model checking, it formally verifies the design properties. Some missing part of verification during simulation is complete by the model checking properties. Main concept is it will prove for the assertions (formula passed or failed). And also it checks for the design behavior.

3. VERIFICATION PLAN

Before starting any type of project, planning plays very important role. In verification, a design plan which is called as blue print. It gives an idea about verification process like what functionality is to be verified, how verification process is to carry out, how success the verification process is, which type of response can be expected by system etc. Main aim of verification plan is to achieve "First time success". [3]

Design specification plays an important role in verification plans. There are two types of specifications.

- Architectural specification
- Design specification

Architectural specification fulfills the device functional requirements and design specification gives implementation idea of certain architecture with down of block level. Specifications are given to both design team and verification team together, so that both works are parallel done by consuming very less time to reach the final stage.

3.1 Levels of Verification

After the verification specification under verification plan, need to know different verification levels. The levels are mentioned below.

- Unit-level verification: In this level it uses ad-hoc type of verification where the designer should only checks for the operations, syntax and functionality of selected unit.
- ASIC and FPGA-level verification: ASIC is SOC which performs verification at system level. FPGA is also an SOC. But only with 50% of effectiveness it allows to implement the functionally and checks the complex system.
- System-level verification: System is a collection of units and blocks which are individually verified. Mainly it verifies the components, and its interactions between components present on the system.
- Board-level verification: It is used to verify the correctness of the system present on the selected board. With the help of board capture tool, board-level models can be instinctively generated.

3.2 Verification Plan For Digital Circuits (Counter-FSM and I2C protocol).

With the help of specification, a plan can be done for the selected digital circuit and that is called as verification specification. Once the specification is done verification should be performed. Both counter and I2C should be verified for present-state and next-state transactions. This depends on clock, reset, given data, supply voltage and acknowledgement. Verification is done by using the CTL properties. [3]

4. FORMAL PROPERTY VERIFICATION TOOL

Formal verification is done by using the tool called as Verification Interacting with Synthesis (VIS). The existence of VIS from the University called Berkeley and Boulder. It is an interacting communication tool between the user and system which are designed using various FSM states. The generations before VIS are HSIS and SMV. Fig-2 shows the outer sketch of working of VIS. [1]

Mainly there are three divisions in VIS which are mentioned below.

- For reading design description consider common front end.
- Verification (VIS-v)
- Synthesis (VIS-s)

Verification in VIS is done on Verilog code and apply model checking on that code by writing CTL properties. It checks in the result that the properties are passed or failed.

Synthesis in VIS has interaction with SIS by simplifying system parts for purpose of verification. Therefore the main intension is to carry the flow of hierarchical synthesis and later Verilog interpretation is converted to gate level. [1]

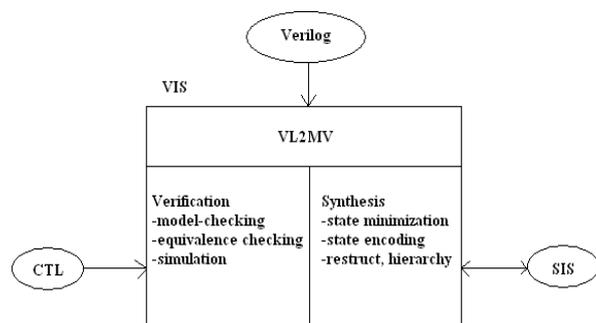


Fig-2: Outer sketch of working of VIS

VL2MV means the conversion of Verilog to BLIF-MV form. VIS does not directly work on Verilog code. It requires intermediary format which is called as BLIF-MV (Berkely Logic Interchange Format). Conversion from Verilog code BLIF-MV format requires a compiler which is called as

VL2MV. BLIF is to trace hierarchical circuit in logic level to textual type.

4.1 Temporal Logic in Formal Verification

It helps in forming the excellent temporal properties by making use of its operators. There are two types in temporal logic. [1]

- Linear Temporal Logic (LTL)
- Computation Tree Logic (CTL)

Consider LTL which gives linear properties operates with linear time. It is applicable to FSM. Example for FSM and LTL is shown in the fig-3(a) and 3(b).

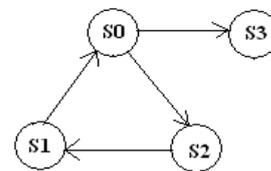


Fig-3(a): FSM sample



Fig-3(b): LTL sample shown for FSM given in 3(a)

Consider CTL which gives the properties that can be easily develop and then verified. Consider an example of FSM shown in the fig-3(a) and the CTL sample is formed in subject tree pattern as shown in the fig-3(c).

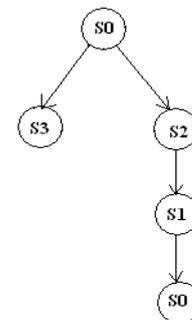


Fig-3(c): CTL sample shown for FSM given in fig-3(a)

4.1.1 Some Properties Example in CTL

User should be aware of CTL formulas before using VIS tool. Below mentioned are some examples of CTL formulas. [6] & [4]

- AG (send → A (send U receive) – this means if always send takes place, then receive is true and till send should also appears true.
- AG AF restart – starting from reachable state the all paths beginning from one state to other state, restart is asserted.

- AG (inp → AX out) – always if input is high, then output also high for one cycle.
- AG EF enabled – From reachable state there should be the path starts at one state then reaches state where enable is asserted.
- AG (REQ → AF ACK) – this means for reachable state AG, REQ is asserted in future state AF and finally ACK is asserted.

4.2 Fairness Constraints

It is important for model checking formal verification. Combination of CTL and fairness constraints gives better assertion and it is called as fair CTL. It is same as CTL, but its semantics is different from CTL which is having fair paths where all path quantifiers are mentioned in it. [1]

5. Formal verification on Counter FSM

Counter is a simple component helps in incrementing or decrementing the counts or numbers. It has vast applications in mathematical field. Example: used in ALU unit, excel applications etc. A 3-bit up-counter state diagram is shown in the fig-4.

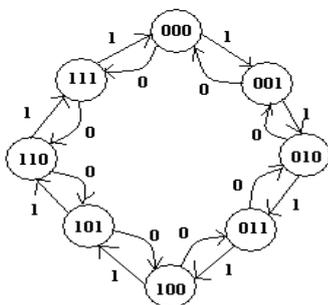


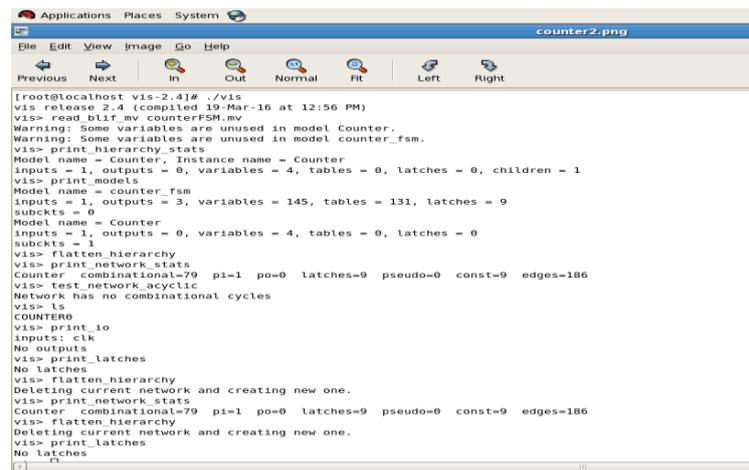
Fig-4: State diagram of 3-bit counter FSM

Again there are two types in FSM which is Mealy machine and Moore machine. Counter belongs to Mealy machine type where it requires present state, next state and given input. [2]

5.1 Design, commands and results regarding counter FSM

The design part of counter requires 8 states along with reset state. It consists of present state (prs) and next state (nxs) to express the state changes. Different state parameters should be initialized before. It is synchronous counter and FSM design should be done using case statements.

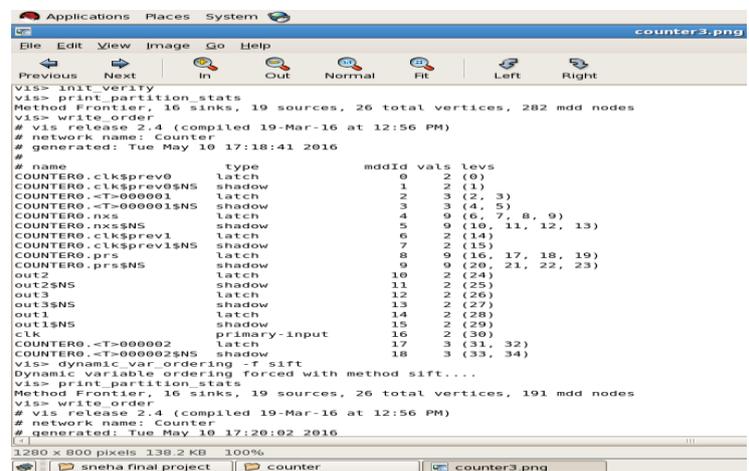
Some commands which are used to check out the design after conversion from Verilog to VL2MV are given in the fig-5(a) and fig-5(b).



```

    [root@localhost vis-2.41# ./vis
    vis release 2.4 (compiled 19-Mar-16 at 12:56 PM)
    vis> read_blif_mv counterFSM_mv
    Warning: Some variables are unused in model Counter.
    vis> print_hierarchy_stats
    Model name = Counter, Instance name = Counter
    inputs = 1, outputs = 0, variables = 4, tables = 0, latches = 0, children = 1
    vis> print_models
    Model name = counter_fsm
    inputs = 1, outputs = 3, variables = 145, tables = 131, latches = 9
    subkts = 0
    Model name = Counter
    inputs = 1, outputs = 0, variables = 4, tables = 0, latches = 0
    subkts = 1
    vis> flatten_hierarchy
    vis> print_network_stats
    Counter combinational=79 pi=1 po=0 latches=9 pseudo=0 const=9 edges=186
    vis> test_network_acyclic
    Network has no combinational cycles
    vis> ls
    COUNTER0
    vis> print_io
    inputs: clk
    No outputs
    vis> print_latches
    No latches
    vis> flatten_hierarchy
    Deleting current network and creating new one.
    vis> print_network_stats
    Counter combinational=79 pi=1 po=0 latches=9 pseudo=0 const=9 edges=186
    vis> flatten_hierarchy
    Deleting current network and creating new one.
    vis> print_latches
    No latches
  
```

Fig-5(a): Commands to build flattened network

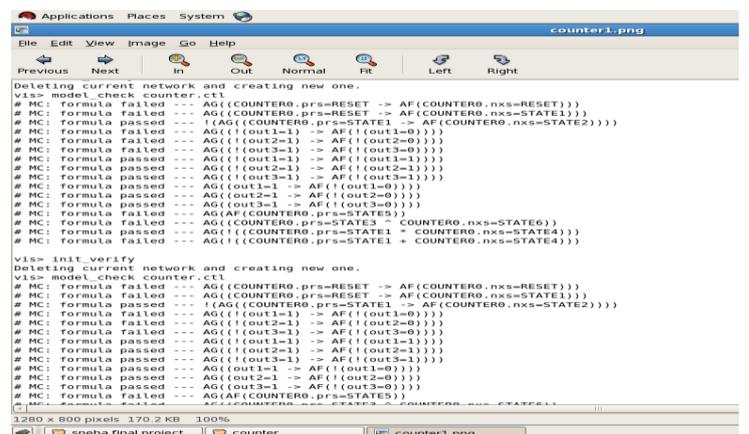


```

    VIS> INIT_Verify
    Method Frontier, 16 sinks, 19 sources, 26 total vertices, 282 mdd nodes
    vis> write_order
    # vis release 2.4 (compiled 19-Mar-16 at 12:56 PM)
    # network name: Counter
    # generated: Tue May 10 17:18:41 2016
    #
    # name type mddId vals levs
    COUNTER0.clkspreV0 latch 0 2 (0)
    COUNTER0.clkspreV0SNS shadow 1 2 (1)
    COUNTER0.<T=000001 latch 2 3 (2, 3)
    COUNTER0.<T=00001SNS shadow 3 3 (4, 5)
    COUNTER0.nxs latch 4 9 (6, 7, 8, 9)
    COUNTER0.nxsSNS shadow 5 9 (10, 11, 12, 13)
    COUNTER0.clkspreV1 latch 6 2 (14)
    COUNTER0.clkspreV1SNS shadow 7 2 (15)
    COUNTER0.prs latch 8 9 (16, 17, 18, 19)
    COUNTER0.prsSNS shadow 9 9 (20, 21, 22, 23)
    out2 latch 10 2 (24)
    out2SNS shadow 11 2 (25)
    out3 latch 12 2 (26)
    out3SNS shadow 13 2 (27)
    out1 latch 14 2 (28)
    out1SNS shadow 15 2 (29)
    COUNTER0.<T=000002 latch 16 2 (30)
    COUNTER0.<T=00002SNS shadow 17 3 (31, 32)
    vis> dynamic_var_ordering -f sift
    Dynamic variable ordering forced with method sift...
    vis> print_partition_stats
    Method Frontier, 16 sinks, 19 sources, 26 total vertices, 191 mdd nodes
    # vis release 2.4 (compiled 19-Mar-16 at 12:56 PM)
    # network name: Counter
    # generated: Tue May 10 17:20:02 2016
  
```

Fig-5(b): Ordering, computing FSM information, advanced ordering and FSM traversal commands.

After applying CTL properties and fair CTL, during model checking, the results are obtained in the form of pass or fail which is shown in the fig-6.



```

    Deleting current network and creating new one.
    # MC: formula failed --- AG((COUNTER0.prs=RESET => AF(COUNTER0.nxs=RESET)))
    # MC: formula failed --- AG((COUNTER0.prs=RESET => AF(COUNTER0.nxs=STATE1)))
    # MC: formula failed --- !AG((COUNTER0.prs=STATE1 => AF(COUNTER0.nxs=STATE2)))
    # MC: formula failed --- AG((out1=1) => AF((out1=0)))
    # MC: formula failed --- AG((out2=1) => AF((out2=0)))
    # MC: formula failed --- AG((out3=1) => AF((out3=0)))
    # MC: formula passed --- AG((out1=1) => AF((out1=1)))
    # MC: formula passed --- AG((out2=1) => AF((out2=1)))
    # MC: formula passed --- AG((out3=1) => AF((out3=1)))
    # MC: formula passed --- AG(out1=1 => AF((out1=0)))
    # MC: formula passed --- AG(out2=1 => AF((out2=0)))
    # MC: formula passed --- AG(out3=1 => AF((out3=0)))
    # MC: formula failed --- AG(AF(COUNTER0.prs=STATE5))
    # MC: formula passed --- AG((COUNTER0.prs=STATE3 * COUNTER0.nxs=STATE6))
    # MC: formula passed --- AG((COUNTER0.prs=STATE1 * COUNTER0.nxs=STATE4))
    # MC: formula failed --- AG(!((COUNTER0.prs=STATE1 * COUNTER0.nxs=STATE4)))

    vis> INIT_Verify
    Deleting current network and creating new one.
    vis> model_check counter.ctf
    # MC: formula failed --- AG((COUNTER0.prs=RESET => AF(COUNTER0.nxs=RESET)))
    # MC: formula failed --- !AG((COUNTER0.prs=STATE1 => AF(COUNTER0.nxs=STATE1)))
    # MC: formula failed --- AG((out1=1) => AF((out1=0)))
    # MC: formula failed --- AG((out2=1) => AF((out2=0)))
    # MC: formula passed --- AG((out1=1) => AF((out1=1)))
    # MC: formula passed --- AG((out2=1) => AF((out2=1)))
    # MC: formula passed --- AG((out3=1) => AF((out3=1)))
    # MC: formula passed --- AG((out1=1) => AF((out1=0)))
    # MC: formula passed --- AG((out2=1) => AF((out2=0)))
    # MC: formula passed --- AG((out3=1) => AF((out3=0)))
    # MC: formula failed --- AG(AF(COUNTER0.prs=STATE5))
  
```

Fig-6: Results regarding counter FSM after model checking.

Model checking in the formal verification checks for the corner cases and finally gives the actual result.

6. Formal verification on I2C protocol

It creates the Communication between IC's, processors and microcontrollers. There are two types of nodes present. Those are master node which sends or receives the data information and slave node which receives or sends the data information. Block diagram of I2C protocol is shown in the fig-7.

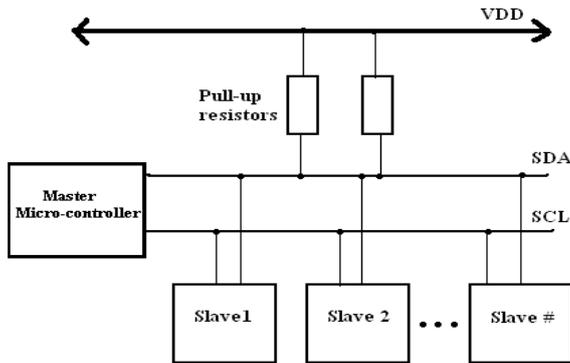


Fig-7: Block diagram of I2C protocol

There are two bus lines. Those are serial clock line and serial data line. Synchronous clock line is for timing purpose and data line is for sending and receiving data between master and slave. Supply voltage is given to buses to maintain the normal condition. Otherwise it is in ideal state. [5]

6.1 I2C Design and operation

Using 24 bit I2C which consist of start bits, slave address bits, acknowledgement bits, slave memory address bits, slave data bits and stop bits. Totally there are 32 states in an I2C design. [2]

Some steps involved during operation are mentioned below.

- Master pulls SCL to "HIGH" and SDA to "LOW" to start the operation.
- Master sends address of slave.
- Master waits for acknowledgement bit from the slave.
- Master sends read(1) or write(0) operation to slave.
- Again waits for acknowledgement bit.
- Master pulls SDA to "HIGH" and SCL to "HIGH" to stop the operation.

6.2 Commands and results regarding I2C protocol

Fig-8(a) and fig-8(b) shows the executed commands on I2C protocol. Fig-8(c) and fig-8(d) are regarding results after finishing model checking.

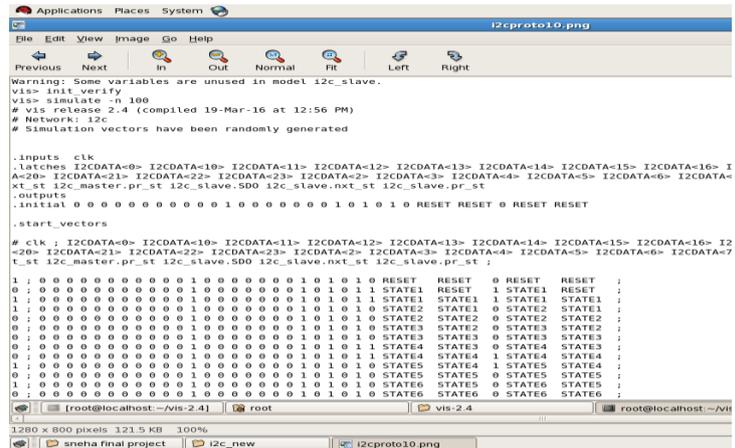


Fig-8(a): Simulation results

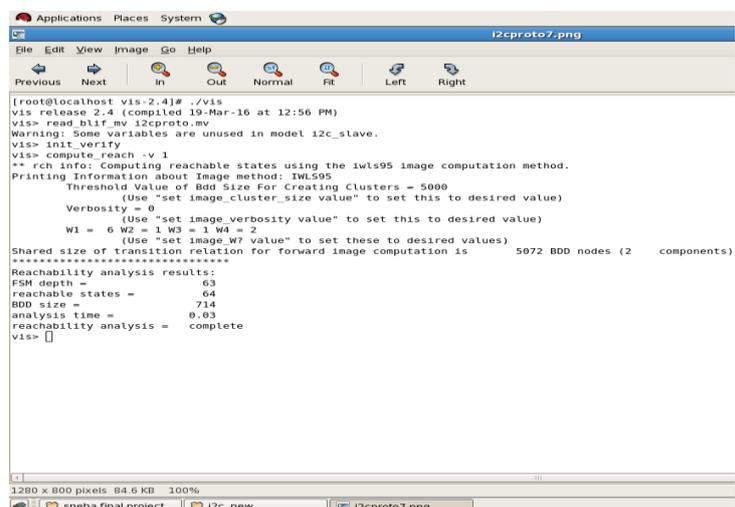


Fig-8(b): Commands on computation information

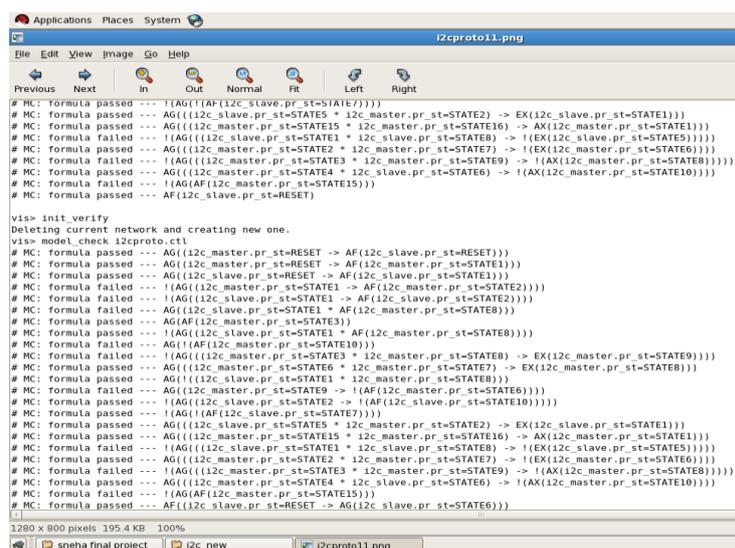
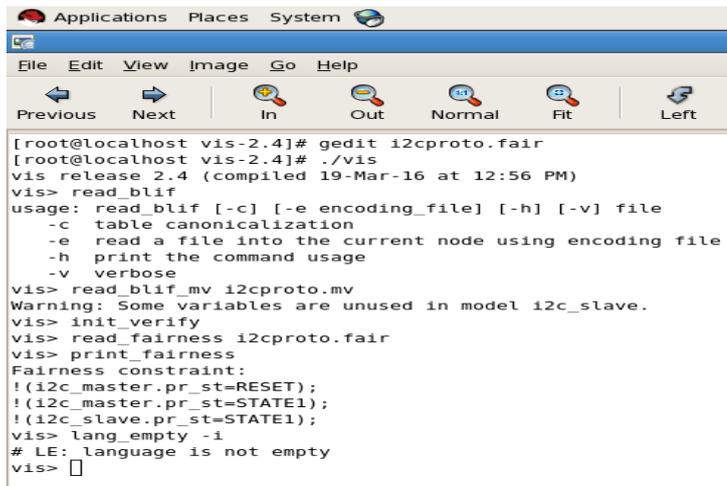


Fig-8(c): Obtained CTL results on model checking



```

[root@localhost vis-2.4]# gedit i2cproto.fair
[root@localhost vis-2.4]# ./vis
vis release 2.4 (compiled 19-Mar-16 at 12:56 PM)
vis> read_blif
usage: read_blif [-c] [-e encoding_file] [-h] [-v] file
  -c table canonicalization
  -e read a file into the current node using encoding file
  -h print the command usage
  -v verbose
vis> read_blif_mv i2cproto.mv
Warning: Some variables are unused in model i2c_slave.
vis> init_verify
vis> read_fairness i2cproto.fair
vis> print_fairness
Fairness constraint:
!(i2c_master.pr_st=RESET);
!(i2c_master.pr_st=STATE1);
!(i2c_slave.pr_st=STATE1);
vis> lang_empty -i
# LE: language is not empty
vis>

```

[6] Lopamudra Sen and Subir.K.Roy, DFT logic verification through property based formal methods, 2010 FMCAD Inc.

Fig-8(d): Obtained Fairness constraints results on model checking

By the concept of model checking in FPV which is power full method involved in verification.

7. CONCLUSIONS

In this paper we study about FPV designs using VIS using model checking properties, can be applied for controlling designs, successfully verified two control oriented designs counter and I2C, Corner cases are verified using FPV.

ACKNOWLEDGEMENT

Author would like to thank Mrs. Roopa. G for her support to this project and also author wants to express an esteemed gratitude to Mr. Yaseen Basha for his guidance to this project. Also thank full to husband, parents, and friends.

REFERENCES

- [1] vis@ic.eecs.berkeley.edu VIS group, University of California, Berkeley, University of Colorado, Boulder.
- [2] www.opencores.org
- [3] Janick Bergeron, Writing Testbenches – Functional verification of HDL models, Springer, 2nd edition 2003
- [4] A. Aziz T, Cheng and R. Hojati. “HSIS: BDD-based environment for formal verification. In university of California at Berkeley, 1997
- [5] Nirmal Saeed, Ayesha Inam, Aisha Khan, Osman Hasan, “V-HOLT verifier-An Automatic Formal Verification Tool for Combinational Circuits, Islamabad @2012 IEEE