International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249 – 8958 (Online), Volume-9 Issue-6, August 2020

Viable System Verilog Assertions(SVA) Praxis in AMBA AHB-Lite Protocol Design

Seerapu Anil Nagendra, Pallavi, M. Murali Krishna

Abstract: ***Digital integrated circuit(ic design entanglement has been far reaching since the demonstration by kil by in 1958.in this day and age system on chip(soc) design verification accommodate billion , more specifically trillion transistors designs came into existence due to artificial intelligence(ai) designs. The expertise designers tried to ramp up design process by using effective eda tools , still the time wheel move in recursive process. In order to accelerate time wheel of design process specified design methodology needed for every design. The overview of various design methodologies followed in the market now a days. Emulation performance by using veloce platform in bfm mode on ahb lite transmissions. Simulation by using software eda tools is slow going on wheel of design when we go for higher abstraction. Accelerated simulation and emulation using hardware is costly in contrast with software simulation. Prototyping is expedient. Formal verification and intelligent software simulations are frail. The possibility of selection between various hardware engines become ravelled. It develop into perspicuous only amalgamation of engines will assist design verification teams to be triumphant. In this design combination of hardware accelerated simulator as a combination of emulator used to accelerate time wheel by using arm amba ahb lite protocol as a design.***

Keywords: ***AMBA, AHB LITE, Simulation, Emulation, Assertions***

I. INTRODUCTION

In this day and age many new vendors begin to develop their own digital IC intellectual property. one of those is on ARM and each vendor come up with the needed solution to integrate their electronic devices. these Intellectual property evolve with the help of electronic design automation (EDA)tools in order to pull off the design performance and other tradeoffs such as area , p9wer , and design time . the EDA tools routed by scripting algorithms developed by the design engineers and they self executing many parts of the design verification flow. so the design verification teams are more with hands on experience make design verification time wheel less and time to market is fast. the arm ahb lite protocol taken from arm site modified according to the design requirements and make sure that after modified design works properly and it had the same operation states that is integrated in ARM IP's. Detailed evaluation of design verification methodologies in market and comparison of methodologies, simulation tools of various vendors. Emulation platform working nature on SoC design SVA(System Verilog Assertions) usage in UVM Environment DUT

Revised Manuscript Received on July 05, 2020.
* Correspondence Author
S.Anil Nagendra*, VLSI Design(M Tech), GITAM University, Visakhapatnam, India.
Pallavi, VLSI And Embedded System (M Tech), RNSIT, BANGLORE, India. E-mail: Sugandhi2391@Gmail.Com
M. Murali Krishna, ECE Department, GITAM University, Visakhapatnam, India.

II. AHB LITE PROTOCOL

AHB Lite system can be single master multiple slave as shown in fig.1. Or multi layer AHB system in which only one ahb master is used.AHB Lite is extension of ahb protocol and deliberately useful in system where only need a single bus master.

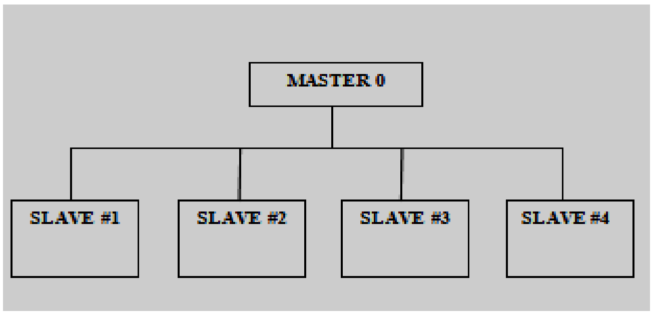


Fig.1.AHB Lite system

Fig.2. shows AHB Lite protocol architecture which consists of Single master module and three slave modules inn conjunction with the bus interconnection by using decoder and multiplexer.
operation:
when read or write transfer initiated by either master or slave by using address, control signals. those are communicate with decoder to obtain address and direction, size of the transfer .
Transfers can be incrementing , wrapping , undefined length , burst transfers
the transaction consists of two operation phases which are Address transaction phase and Data transaction phase
the write data bus operation initiated by master and written up in slave. read data bus initiated by slave and reads data from master.
HREADY signal used to develop wait states and slave has more time to sample data .

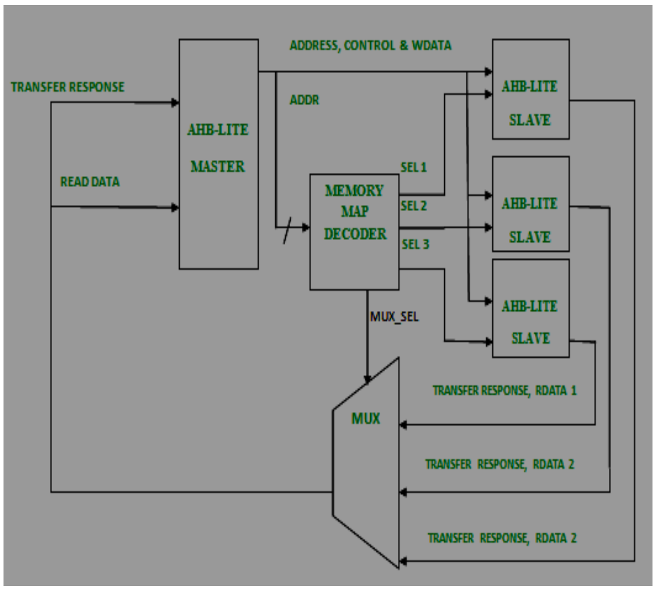


Fig.2.AHB LITE PROTOCOL ARCHITECTURE

Slave

Fig.3. shows AHB Lite slave Top Module
AHB Lite slave uses HSELx signal generated by decoded to response the bus transactions controlled master in the protocol. Slave proffer Response either Success ,failure, or waiting transfers

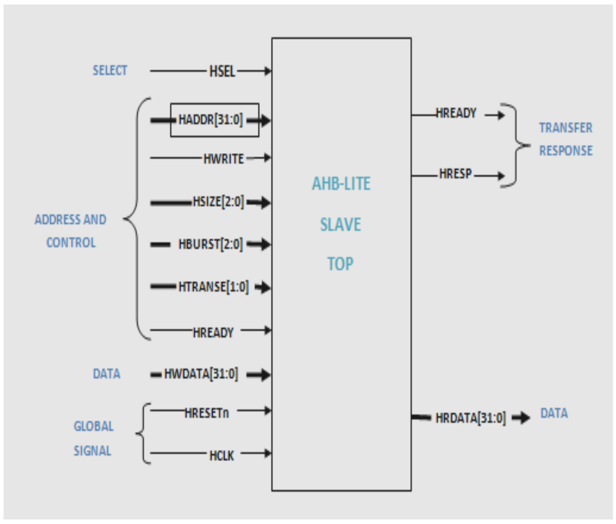


Fig.3.AHB LITE SLAVE TOP

III. ENVIRONMENT

Master: is written/programmed using a Bus Functional Model Approach (not Synthesizable)
-It contains a host of tasks –read, write-that encompass the way the DUT is used.
-By calling those tasks a read or a write would occur on the bus/interface of the DUT
Slave: number of memory modules used in design is parameterized. slave is a RAM, simple memory module.
Monitor: The Current Transactions developed by assertions monitored by using this module and also checks the bus traffic signals.
Interface: Slave MODPORT provide a proper lookup for the slave.

Fig.4.shows SV design verification environment used in simulation and emulation and all the AHB Lite bus signals described in it

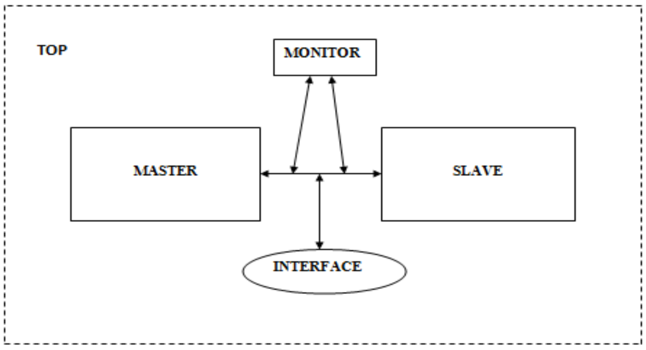


Fig.4.SYSTEM VERILOG ENVIRONMENT

IV. VERIFICATION PLAN AND TEST CASES

```
property error_check;
@(posedge Bus.HCLK) disable iff (Bus.HTRANS == BUSY
|| Bus.HTRANS == IDLE)
(Bus.HADDR > ((2**10)* `NoOfSlaves))|-> (Bus.HRESP
== 1'b1);
endproperty
property  read_only_error_check;
@(posedge Bus.HCLK) disable iff ( (Bus.HADDR[9:0] >
9'd3))
 (Bus.HWRITE == 1'b1) |=> (Bus.HRESP == 1);
endproperty
property basic_write;
@(posedge Bus.HCLK) disable iff ((Bus.HTRANS==IDLE ||
Bus.HTRANS==BUSY) && Bus.HBURST>'0)
$rose(Bus.HWRITE) |=> Bus.HREADY;
   endproperty.
```

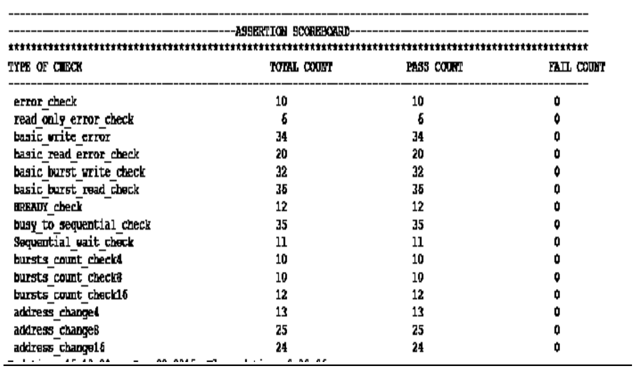
TEST CASES

```
InterfaceInstance.burst_write (32'd5,4,0, data_burst);
      #20;
      InterfaceInstance.burst_read (32'd5,4,0,
data_burst_read);
      #20;
      `ifdef DEBUG
      $display ("DATA - 0 BUSY - BURST4  = %d, %d, %d,
%d\n", data_burst_read[0],      data_burst_read[1],
data_burst_read[2], data_burst_read[3] );
      `endif
InterfaceInstance.burst_write (32'd10,8,0, data_burst);
      #20;
      InterfaceInstance.burst_read (32'd10,8,0,
data_burst_read);
      #20;
      `ifdef DEBUG
      $display ("DATA - 0 BUSY - BURST8  = %d, %d, %d,
%d, %d, %d, %d, %d", data_burst_read[0],
data_burst_read[1], data_burst_read[2], data_burst_read[3],
data_burst_read[4], data_burst_read[5], data_burst_read[6],
data_burst_read[7] );
      `endif
        InterfaceInstance.burst_write (32'd25,16,0,
data_burst);
      #20;
```

```
        InterfaceInstance.burst_read (32'd25,16,0,
data_burst_read);
        #20;
        `ifdef DEBUG
        $display ("DATA - 0 BUSY - BURST16 = %d, %d, %d,
%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d",
data_burst_read[0], data_burst_read[1], data_burst_read[2],
data_burst_read[3], data_burst_read[4], data_burst_read[5],
data_burst_read[6], data_burst_read[7],
data_burst_read[8], data_burst_read[9],
data_burst_read[10], data_burst_read[11],
data_burst_read[12], data_burst_read[13],
data_burst_read[14], data_burst_read[15]);
        `endif
```

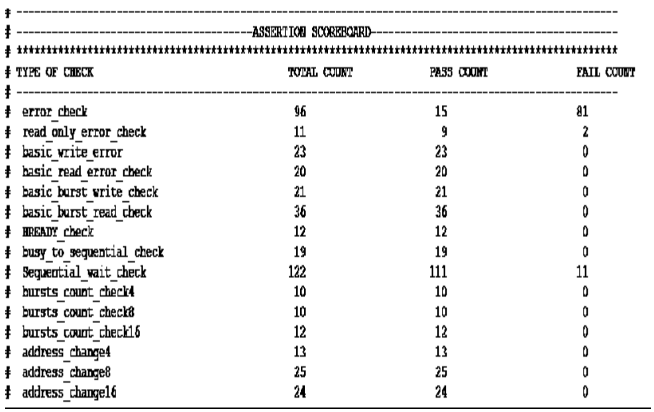
V. RESULTS

ASEERTION SCOREBOARD WITHOUT ERRORS



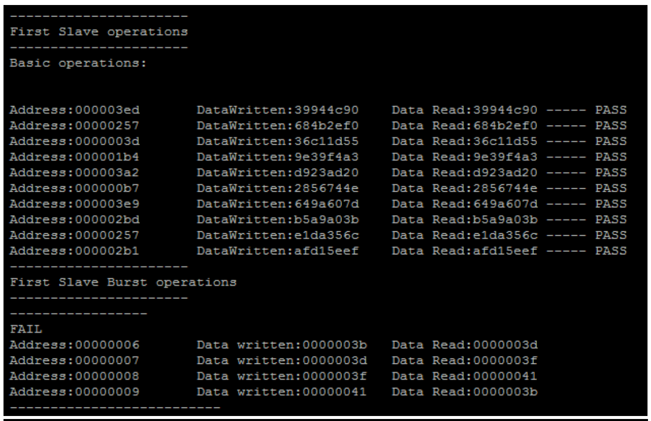
```
------------------------------------------------------------------------------------------------
--------------------------------------ASSERTION SCOREBOARD--------------------------------------
************************************************************************************************
TYPE OF CHECK                                TOTAL COUNT           PASS COUNT           FAIL COUNT
------------------------------------------------------------------------------------------------
 error_check                                      10                   10                   0
 read_only_error_check                             6                    6                   0
 basic_write_error                                34                   34                   0
 basic_read_error_check                           20                   20                   0
 basic_burst_write_check                          32                   32                   0
 basic_burst_read_check                           35                   35                   0
 HREADY_check                                     12                   12                   0
 busy_to_sequential_check                         35                   35                   0
 Sequential_wait_check                            11                   11                   0
 bursts_count_check4                              10                   10                   0
 bursts_count_check8                              10                   10                   0
 bursts_count_check16                             12                   12                   0
 address_change4                                  13                   13                   0
 address_change8                                  25                   25                   0
 address_change16                                 24                   24                   0
```

ASEERTION SCOREBOARD WITH ERRORS



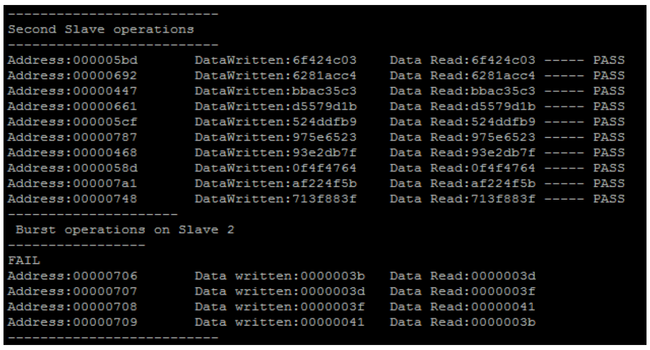
```
# ------------------------------------------------------------------------------------------------
# --------------------------------------ASSERTION SCOREBOARD--------------------------------------
# ************************************************************************************************
# TYPE OF CHECK                                TOTAL COUNT           PASS COUNT           FAIL COUNT
# ------------------------------------------------------------------------------------------------
#  error_check                                      96                   15                  81
#  read_only_error_check                            11                    9                   2
#  basic_write_error                                23                   23                   0
#  basic_read_error_check                           20                   20                   0
#  basic_burst_write_check                          21                   21                   0
#  basic_burst_read_check                           36                   36                   0
#  HREADY_check                                     12                   12                   0
#  busy_to_sequential_check                         19                   19                   0
#  Sequential_wait_check                           122                  111                  11
#  bursts_count_check4                              10                   10                   0
#  bursts_count_check8                              10                   10                   0
#  bursts_count_check16                             12                   12                   0
#  address_change4                                  13                   13                   0
#  address_change8                                  25                   25                   0
#  address_change16                                 24                   24                   0
```

Emulation -veloce

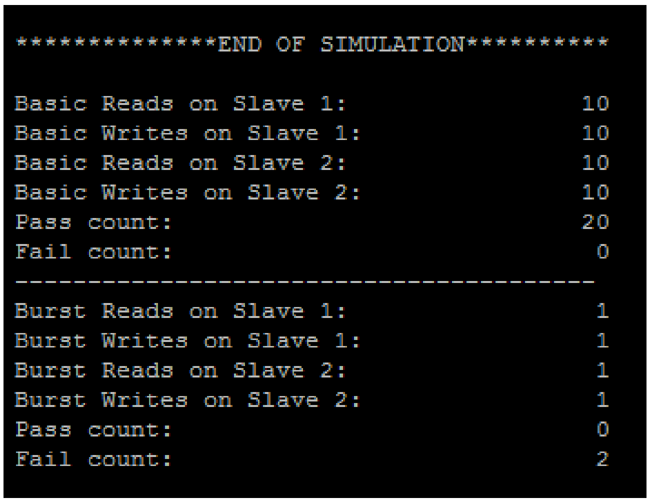


```
----------------------
First Slave operations
----------------------
Basic operations:


Address:000003ed       DataWritten:39944c90    Data Read:39944c90 ----- PASS
Address:00000257       DataWritten:684b2ef0    Data Read:684b2ef0 ----- PASS
Address:0000003d       DataWritten:36c11d55    Data Read:36c11d55 ----- PASS
Address:000001b4       DataWritten:9e39f4a3    Data Read:9e39f4a3 ----- PASS
Address:000003a2       DataWritten:d923ad20    Data Read:d923ad20 ----- PASS
Address:000000b7       DataWritten:2856744e    Data Read:2856744e ----- PASS
Address:000003e9       DataWritten:649a607d    Data Read:649a607d ----- PASS
Address:000002bd       DataWritten:b5a9a03b    Data Read:b5a9a03b ----- PASS
Address:00000257       DataWritten:e1da356c    Data Read:e1da356c ----- PASS
Address:000002b1       DataWritten:afd15eef    Data Read:afd15eef ----- PASS
----------------------
First Slave Burst operations
----------------------
-----------------
FAIL
Address:00000006       Data written:0000003b   Data Read:0000003d
Address:00000007       Data written:0000003d   Data Read:0000003f
Address:00000008       Data written:0000003f   Data Read:00000041
Address:00000009       Data written:00000041   Data Read:0000003b
--------------------------
```



```
--------------------------
Second Slave operations
--------------------------
Address:000005bd       DataWritten:6f424c03    Data Read:6f424c03 ----- PASS
Address:00000692       DataWritten:6281acc4    Data Read:6281acc4 ----- PASS
Address:00000447       DataWritten:bbac35c3    Data Read:bbac35c3 ----- PASS
Address:00000661       DataWritten:d5579d1b    Data Read:d5579d1b ----- PASS
Address:000005cf       DataWritten:524ddfb9    Data Read:524ddfb9 ----- PASS
Address:00000787       DataWritten:975e6523    Data Read:975e6523 ----- PASS
Address:00000468       DataWritten:93e2db7f    Data Read:93e2db7f ----- PASS
Address:0000058d       DataWritten:0f4f4764    Data Read:0f4f4764 ----- PASS
Address:000007a1       DataWritten:af224f5b    Data Read:af224f5b ----- PASS
Address:00000748       DataWritten:713f883f    Data Read:713f883f ----- PASS
---------------------
 Burst operations on Slave 2
-----------------
FAIL
Address:00000706       Data written:0000003b   Data Read:0000003d
Address:00000707       Data written:0000003d   Data Read:0000003f
Address:00000708       Data written:0000003f   Data Read:00000041
Address:00000709       Data written:00000041   Data Read:0000003b
--------------------------
```



```
**************END OF SIMULATION**********

Basic Reads on Slave 1:               10
Basic Writes on Slave 1:              10
Basic Reads on Slave 2:               10
Basic Writes on Slave 2:              10
Pass count:                           20
Fail count:                            0
---------------------------------------
Burst Reads on Slave 1:                1
Burst Writes on Slave 1:               1
Burst Reads on Slave 2:                1
Burst Writes on Slave 2:               1
Pass count:                            0
Fail count:                            2
```

VI. CONCLUSIONS

the initial functionality defining RTL code written from AHB lite specifications taken from ARM documentation. which is exhaustively created test bench environment simulated by using Questasim tool.

design verification issues rapidly increases day by day due to radically increasing in gate count and functionalities . more than 75% time , resources and endeavor takes in detecting bugs in interconnection on SoC chipsets is boundless problem.

simulation and emulation engines combination is unique and SV methodologies proffer accelerated performance ,flexible design, scalability in design area. these methods support real test environment for the design under test (DUT).By wrapping the emulation DUT platform via intelligent interfaces with components enables high speed in design verification.

the systems had high level abstractions can be debugged easily by co emulation constructed test bench. the design verification teams moves to strategic development to accelerate the design performance and time to market

An acceleration ready frame work describes benefits in yielding higher gain over pure simulation and significant decrease in development time for emulation. Users will be able to write block level SV environment that can be uses directly in emulation. this approach will be a solution in getting SoC design verification in shorter times for block, subsystem, system level.

VII. FUTURE SCOPE

in the day and age SoC designs are high level abstraction based with innumerable functionalities. to verify SoC deign the teams required more and correlation between each team is mandatory.

If we take a simulator as the baseline, emulation, acceleration and prototyping cam all run faster than a simulator, but the speed-up is dependent on many factors AI SoC Development engineers grasp the design verification environment which will reduce the design services time wheel to come into market .while generating these new generation IP's and soc chipsets .which are highly built in advantages over currently used IP's and soc chipsets

design verification grasp by AI technology makes accelerate the design time and availability in market , which will provide extra time to design teams to make clear cut design features.

REFERENCES

1. AHB, AMBA Multi-Layer. "AHB-Lite,”." Bus Specification http://www. arm. com.
2. Taraate, Vaibbhav. "Buses and Protocols in SOC Designs." In Advanced HDL Synthesis and SOC Prototyping, pp. 97-117. Springer, Singapore, 2019.
3. Mulani, Purvi D. "SoC level verification using System Verilog." In 2009 Second International Conference on Emerging Trends in Engineering & Technology, pp. 378-380. IEEE, 2009.
4. Foster, Harry, Lawrence Loh, Bahman Rabii, and Vigyan Singhal. "Guidelines for creating a formal verification testplan." Proc. DVCon (2006).
5. Cohen, Ben, Srinivasan Venkataramanan, and Ajeetha Kumari. SystemVerilog Assertions Handbook:--for Formal and Dynamic Verification. vhdlcohen publishing, 2005.
6. Shah, Saurin, and Nirav Patel. "Systemverilog Based AMBA AHB Verification Environment." International Journal of Engineering Research 3, no. 5 (2014).
7. Gandhi, Ashima. "Functional Verification of AMBA AHB-Lite using Layered Testbench Technology of System Verilog." Journal of VLSI Design Tools & Technology 6, no. 2 (2016): 104-112.
8. Kante, Sravya, Hari Kishore Kakarla, and Avinash Yadlapati. "Design and Verification of AMBA AHB-Lite protocol using Verilog HDL." International Journal of Engineering and Technology, EISSN 0975-4024: 734-741.
9. Sinha, Richa, Akhilesh Kumar, and Archana Kumari Sinha. "Verification analysis of AHB-LITE protocol with coverage." International journal of advances in Engineering & technology 2, no. 1 (2012): 121.
10. M. B. R. Srinivas and S. Musala, "Verification of AHB_LITE protocol for waited transfer responses using re-usable verification methodology," 2016 International Conference on Inventive Computation Technologies (ICICT), Coimbatore, 2016, pp. 1-3, doi: 10.1109/INVENTIVE.2016.7830137.
11. A. K. Singh, A. Shrivastava and G. S. Tomar, "Design and Implementation of High Performance AHB Reconfigurable Arbiter for Onchip Bus Architecture," 2011 International Conference on Communication Systems and Network Technologies, Katra, Jammu, 2011, pp. 455-459, doi: 10.1109/CSNT.2011.99.
12. Goswami, Disha, and Dharmesh J. Shah. "Implementation of SystemVerilog Environment for Functional Verification of AHB-DMA Bridge." International Journal on Recent and Innovation Trends in Computing and Communication 3, no. 5: 2799-2802.

AUTHORS PROFILE

Seerapu Anil Nagendra Pursuing M Tech in VLSI Design from GITAM Deemed to be University, Visakhapatnam. Currently Doing Internship as SoC DV , SION SEMICONDUCTORS PVT LTD, Bangalore.

Pallavi Completed B.Tech In ECE From Godutai Engineering College For Women, Kalaburigi. Currently Pursuing M.tech in VLSI and Embedded systems from RNSIT, Bangalore.

Muralikrishna. M working as Asst. professor in Dept of ECE,GIT,GITAM deemed to be university. Had hands on experience in various VLSI Design methodologies and Worked on Analog IC design as well as Digital IC design methodology flows.