International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Verification of I2C Master Core using System Verilog-UVM

T Tarun Kumar[1], CY Gopinath[2]

[1]Bangalore Institute of Technology, Bangalore, Karnataka - 560004, India

[2]Associate Professor, Bangalore Institute of Technology, Department of Electronics and Communication
Bangalore, Karnataka - 560004, India

Abstract: ***This paper contracts the reusability of the I2C Bus protocol under various design environments, and by following Universal Verification Methodology (UVM) we can test the design and its functionality in these environments. The RTL design of I2C is open source and is obtained from Opencore.org, and its functional verification is carried by self, using System Verilog and UVM. The main advantage of this type of methodology is it does not interfere with the DUT and it is reusable with little or no modification. The design and verification in UVM is carried out on Mentor Graphics Questasim 10c.The coverage so obtained is 100% for assertion based coverage and 90.15% functional coverage using SV (SystemVerilog). The total coverage so obtained is 95.07%.***

Keywords: I2C, SystemVerilog, UVM, Functional Verification, Coverage, Assertion.

1. Introduction

In consumer electronics, telecommunications and industrial electronics, there are often many similarities between seemingly unrelated designs. For example, nearly every system includes:

- Some intelligent control, usually a single-chip microcontroller
- General-purpose circuits like LCD and LED drivers, remote I/O ports, RAM, EEPROM, real-time clocks or A/D and D/A converters
- Application-oriented circuits such as digital tuning and signal processing circuits for radio and video systems, temperature sensors, and smart cards

To exploit these similarities to the benefit of both systems designers and equipment manufacturers, as well as to maximize hardware efficiency and circuit simplicity, in early 1980's Philips Semiconductor (now NXP Semiconductors) developed a simple bidirectional 2-wire bus for efficient inter-IC communication and control. This bus is called the Inter-IC or I2C-bus and is de facto world standard that is now implemented in over 1000 different ICs manufactured by more than 50 companies [6]. All I2C-bus compatible devices incorporate an on-chip interface which allows them to communicate directly with each other via the I2C-bus. This design concept solves the many interfacing problems encountered when designing digital control circuits. Additionally, the versatile I2C-bus is used in various control architectures such as System Management Bus (SMBus), Power Management Bus (PMBus), Intelligent Platform Management Interface (IPMI), Display Data Channel (DDC) and Advanced Telecom Computing Architecture (ATCA).

In Verilog or VHDL, a testbench consists of a hierarchy of modules containing testbench code that are connected to the design under test (DUT). The modules contain stimulus and response checking code which is loaded into simulator memory along with the DUT at the beginning of the simulation and is present for the duration of the simulation. Therefore, the classic Verilog test bench wrapped around a DUT consists of what are known as static objects. SystemVerilog builds on top of Verilog by adding abstract language constructs targeted at helping the verification process. One of the key additions to the language was the class. SystemVerilog classes allow Object Orientated Programming (OOP) techniques to be applied to testbenches. Unlike all of the design modules and interfaces that are called during compilation and elaboration, none of the UVM testbench environment is setup until after simulation starts.

The Importance of verification is [1]-[4]-[12]:

- *70% of design effort goes to verification*
- *Verification is on the critical path*
- *Verification time can be reduced through abstraction*
- *Using abstraction reduces control over low level details*
- *Verification time can be reduced through automation*
- *Randomization can be used as an automation tool*

The UVM itself is a library of base classes which facilitate the creation of structured testbenches using code which is open source and can be run on any SystemVerilog IEEE 1800 simulator [1]-[5]. UVM 1.1d Reference Implementation was released in March, 2013 [5].This paper details important basics on requirements for the verification flow to check the functionality of any digital design. It lists the important steps that are needed to be followed in creating UVM environment and also the required components in order to achieve them.

2. Specification of the Design under Test

The Design under Test is an I2C master controller core. It produces SDA and SCL signals as per the configuration of its internal registers. SCL controls clock and SDA is used to transfer data. Each device connected to the bus is software addressable by aunique address and simple master/ slave relationships exist at alltimes; masters can operate as master-transmitters or asmaster-receivers. The number of ICs that

Licensed Under Creative Commons Attribution CC BY

can be connected to the same bus islimited only by a maximum bus capacitance of 400 pF. Figure 1 shows the basic block diagram of the design to be verified.

2.1 I2C BUS

A bus connects the components of a system, e.g. connection between CPU and main memory, memory and I/O ports or between peripheral devices. Different types of bus are available for data transfer. They are I2C, PCI, WISHBONE, AMBA, XBUS, SPI, and USB.

Each bus has different protocol and bus speed. A faster bus speed allows faster data transfer. In a typical computer or SOC bus is used for address transfer and data transfer. The name I2C translates into "Inter IC". Sometimes the bus is called IIC or I²C bus[6]-[9]. I2C is used on single boards and to connect components which are linked via cable. Key characteristics that make this bus attractive to many applications are simplicity and flexibility.

- I2C bus has three speeds:
  - Slow (under 100 Kbps)
  - Fast (400 Kbps)
  - High- speed (3.4 Mbps)

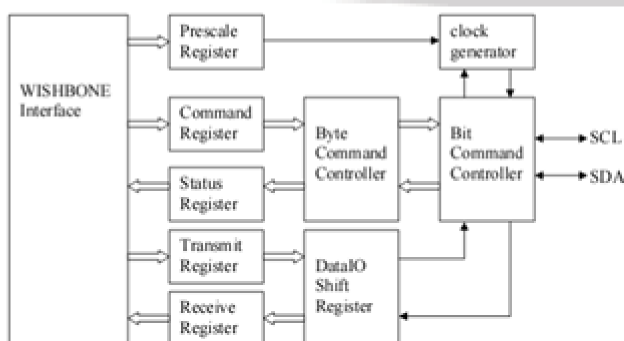

Figure 1: Block diagram of the master controller core

3. I2C Communication Procedure

The IC that initiates communication is called Master and the one that is addressed is called slave. Once an IC that wants to communicate with another IC [6].

a) Check whether there is any bus activity is occurring or not. If both SDA and SCL line are high then bus is free. If the bus is available master generates START condition.
b) SCL provides clock signal to all the ICs connected through the bus as reference clock signal. The data on the data wire (SDA) must be valid at the time the clock wire (SCL) switches from 'low' to 'high' voltage.
c) Address of each device is put on serial form on the SDA line.
d) One bit signal is put on the SDA line to know whether data is to be transmitted or received from the slave.
e) One bit represents acknowledgement bit to inform the master that slave is ready to receive or transmit data.
f) After the acknowledgement bit is received by the master it puts data serially on the SDA line.
g) The first IC sends or receives as many 8- bit words of data as it wants. After every 8- bit data word the sending IC expects the receiving IC to acknowledge that the data is received.
h) When all data is received STOP condition is generated and the bus is free again.

The various control signals in I2C bus protocol are defined as follows [6]:

- START – high-to-low transition of the SDA line while SCL line is high.
- STOP – low-to-high transition of the SDA line while SCL line is high.
- ACK – receiver pulls SDA low while transmitter allows it to float high.
- DATA – transition takes place while SCL is slow, valid while SCL is high.

Figure 2 shows a complete data transfer covering all the above mentioned states.

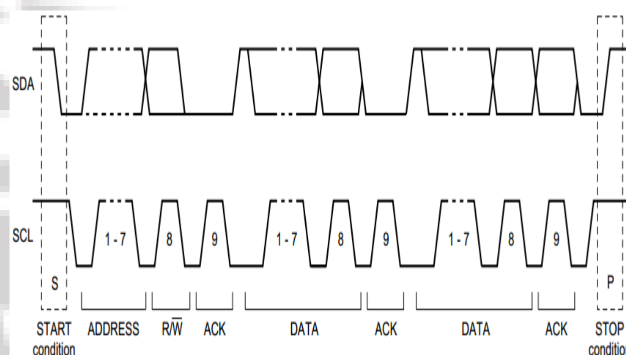

Figure 2: A Complete Data Transfer

4. Verification Methodology

4.1 Compiling Designs & Running UVM

To help understand how UVM simulations work within the SystemVerilog testbench environment, it is useful to have a big-picture view of the entire simulation flow as shown in figure 3.

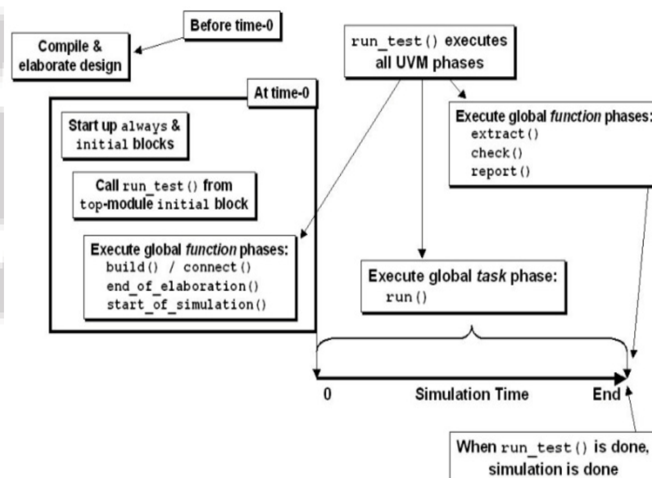

Figure 3: Compiling designs & running UVM – overview

Licensed Under Creative Commons Attribution CC BY

A design and testbench are first compiled, and then the design and testbench are elaborated. Design and elaboration happen before the start of simulation at time-0 [11].

At time-0, the procedural blocks (**initial** and **always** blocks) in the top-level module and in the rest of the design start running. In the top-level module is an **initial** block that calls the **run_test( )** task from **uvm_top**, which is the testcase we want to run. It is passed to simulator by passing the test name or by using "+UVM_TESTNAME=" switch. When **run_test ( )** is called at time-0, the UVM pre-**run( )** global function phases (**build( )**, **connect( )**, **end_of_elaboration( )**, **start_of_simulation( )**) all execute and complete. After the pre-**run( )** global function phases complete (still at time-0), the global **run( )** phase starts. The **run ( )**phase is a task-based phase that executes the entire simulation, consuming all of the simulation time. When the **run( )** phase stops, the UVM post-**run( )** global function phases (**extract( )**, **check( )**, **report( )**) all run in the last time slot before simulation ends. By default, when **run_test( )** is done, **$finish** is called to terminate the simulation [11].

Phases are a synchronizing mechanism for the environment.The UVM provides the following predefined phases for all uvm_components [13].

- build - Depending on configuration and factory settings,create and configure additional component hierarchies.
- connect - Connect ports, exports, and implementations (imps).
- end_of_elaboration - Perform final configuration, topology, connection, and other integrity checks.
- start_of_simulation - Do pre-run activities such as printing banners, pre-loading memories, etc.
- run - Most verification is done in this time-consuming phase. May fork other processes. Phase ends when global_stop_request is called explicitly.
- extract - Collect information from the run in preparation for checking.
- check - Check simulation results against expected outcome
- report - Report simulation results.

4.2 UVM Verification Components

- **Design Under Test** - The design that is intended to be verified .This is generally RTL description in any of the HDL (Verilog, VHDL and System Verilog).This completely describes the functionality of the design as well the features to be verified.
- **Interface** - Interface serves as the actual link between the design- under- verification and the verification environment. It is a SystemVerilog interface. The interface describes the pin - level description of the DUT. An interface is basically a bundle of nets or wires.
- **Virtual Interfaces** - It provide a mechanism for separating abstract models from the actual signals of the design. A virtual interface allows the same instance or the subprogram to operate on different parts of the design. It dynamically controls the set of signals associated with the subprogram, this allows passing the same data over all the components.
- **Transactions** - Interfaces represent the input to the DUT. The fields and attributes of transactions are derived from the transaction's specification. In a test, many data items are generated and those are sent to the DUT via driver. Generally data item fields are randomized using System Verilog constraints many number of tests can be created.
- **Agents** - Most DUTs have a number of different signal interfaces, each of which have their own protocol. The UVM agent collects together a group of uvm_components focused around a specific pin-level interface. The purpose of the agent is to provide a verification component which allows users to generate and monitor pin level transactions. I2C agent used by the Testbench communicates with the DUT, and to create background traffic. Wishbone agent is used to drive the DUT via the DUT's wishbone interface
- **Sequence And Sequence -** A sequence is the series of transaction and sequencer is used to control the flow of transaction generation. A sequence is extended from uvm_sequence class. uvm_sequencer does the generation of this sequence of transaction. Driver (extension of uvm_driver) takes the transactions from Sequencer and processes the packets of data or drives them to other component or to the DUT. It allows the addition of constraints to the data item generated in the sequence, thus bringing forth the corner cases.
- **Driver -** Driver is defined by extending uvm_driver. Driver takes the transactions from the sequencer by using seq_item_port. These transactions will be driven to DUT as per the interface signal specifications. Then it sends the transaction to scoreboard using uvm_analysis_port. Task for resetting DUT and configuring the DUT are also declared here. An instance of the driver class is created in the environment class and the sequencer is connected to it.
- **Monitor** - A monitor is a passive entity that samples DUT signals but doesn't drive them. A monitor:
  - Collects transactions (data items).
  - Extracts events, performs checking and coverage.
  - Optionally prints trace information.

  Checking typically consists of protocol and data checkers to verify that the DUT Output meets the protocol specification. Coverage is collected in the monitor. It is implemented by extending the uvm_monitor class and an instance is created in the environment for hooking it up with DUT signals.
- **Scoreboard -** Scoreboard is implemented by extending uvm_scorboard. Scoreboard has 2 analysis imports. One is used to for getting the packets from the driver and other from the receiver. Then the packets are compared and if they don't match, then error is asserted. Compare function of transaction class is used for comparison.
- **Environment -** Environment class is used to implement verification environments in UVM. It is extension on uvm_env class. The testbench simulation needs some systematic flow like building the components, connection the components, starting the components etc. uvm_env base class has methods formalize the simulation steps. All the methods inside environment class are declared virtual. Virtual interface is created in the environment and all other virtual functions of environment class are extended. Our environment is the top level of the class based part of the testbench. It also contains the virtual sequencer that is used to run sequences to coordinate the wband i2cagents sequences [13].

Licensed Under Creative Commons Attribution CC BY

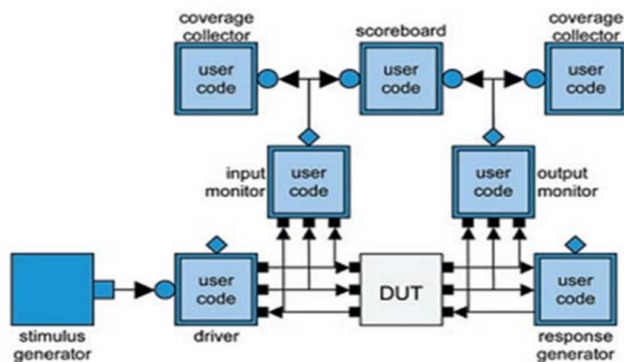

Figure 4: UVC in a generalized Verification Environment

- **Testcases -** The uvm_test class defines the test scenario for the testbench for the DUT and is specified in the test. Testcase contains the instance of the environment class. This testcase creates an Environment object and defines the required test specific functionality. Verification environment contains the declarations of the virtual interfaces. These virtual interfaces are pointed to the physical interfaces which are declared in the top module. These virtual interfaces are made to point to physical interface in the testcase.
- **Top Module -** SystemVerilog interface instance is created in this module. DUT instance is created and hooked up with the interface instance. Clock generator is implemented here. run_test method is called from here. The test name can be implicitly passed or can be passed as a command line argument during simulation. The command line argument takes greater precedence.

5. Verification Plan

The Verification Plan defines exactly what needs to be tested, and drives the coverage criteria. The completeness of a verification plan and its accurate implementation lead to success of the verification project in hand. Detailed goals using measurable metrics, along with optimal resource usage and realistic schedule estimates are the contents of a good plan. Feature extraction, Stimulus generation plan, Checker plan and Coverage plan are the important parts of a verification plan [13].

Feature Extraction
It contains the list of all the features to be verified. For the present DUT, it is the following.

- Response of the DUT in different states: idle, read, write.
- Generation of START and STOP condition.
- Clock Synchronization between the master and slave.
- 7- Bit addressing validity.
- All possible Master – Slave data transfer formats
- Generation of ACK and NACK.

Stimulus Generation Plan
- The type of the transfer (read/write).
- The length of the transfer.
- If arbitration occurs or not.
- Slave clock stretching.
- The speed of the transfer.
- The SCL frequency of each participant.
- The addresses addressed by the masters.
- The addresses of the slaves.
- The number of frames.
- The ACK/NACK probability.
- Whether the bus is released at theend of a transfer (STOP vs. Re-START).
- The delay between frames.

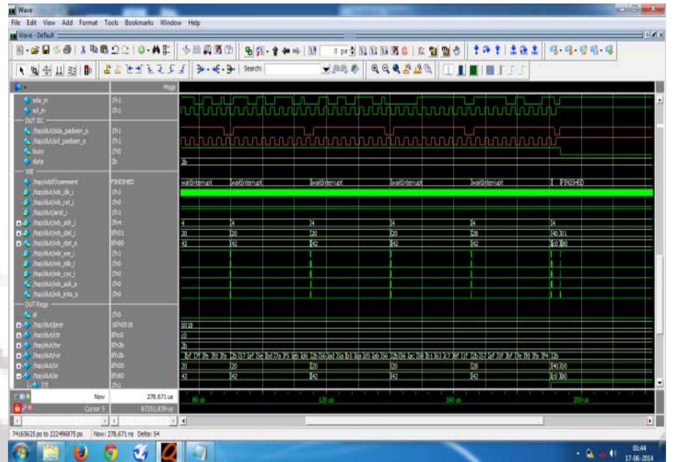
Figure 5(a): Simulation of the I2C for Read operation

Checker Plan is for checking expected results, implemented by monitors and scoreboards based on the protocol. Coverage Plan explains the functional coverage of the features. A functional coverage plan should be built to help implement coverage points in the verification environment.

6. Results

The I2C is implemented using the Verilog with full duplex mode which allows the communication between the master and the salve through the handshaking protocol. When a slow slave is attached to the bus then problems may occur. This mechanism works on the SCL line only. The slave that wants the master to wait simply pulls the SCL low as long as needed. If the SCL gets stuck due to an electrical failure of a circuit, the master can go into deadlock and this can be handled by timeout counters.

Another drawback is speed. The bus is locked at that moment. Other masters cannot use the bus at that time either. This technique does not interfere with the previously introduced arbitration mechanism because the low SCL line will lead to back-off situations in other devices which possibly would want to "claim" the bus. So there is no real drawback to this technique except the loss of speed/bandwidth and some software overhead in the masters. We can use this mechanism between masters in a multi-master environment. So the implementation is carried with three slaves and two masters. The functional verification of the I2C is carried using the Questasim10.0c [14]. The verification is carried for both the READ and WRITE operation cycles and waveforms are generated as shown in figure 5(a) and (b). Coverage metrics such as Code coverage, Block coverage, Expression coverage, Toggle coverage and FSM coverage is also covered here. Covergroups and Assertions are used here to estimate functional coverage. The coverage report obtained is 100% for assertion based

Licensed Under Creative Commons Attribution CC BY

coverage and 90.15% functional coverage using SV (SystemVerilog). The overall average coverage so obtained is 95.07% as shown in figure 6. The FSM is generated using Cadence IMC showing various states covered and state transitions as shown in figure 7.

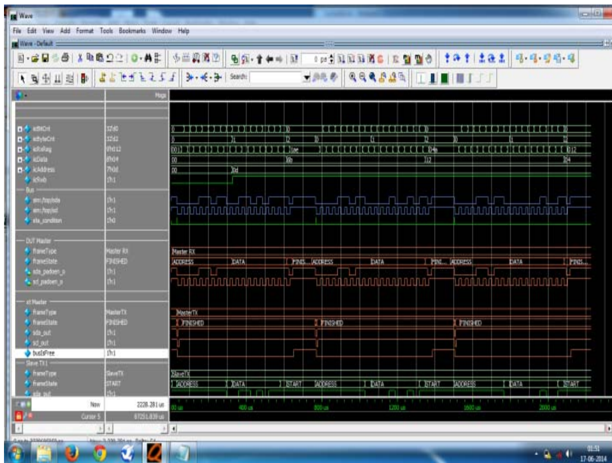

Figure 5(b): Simulation of the I2C for write operation

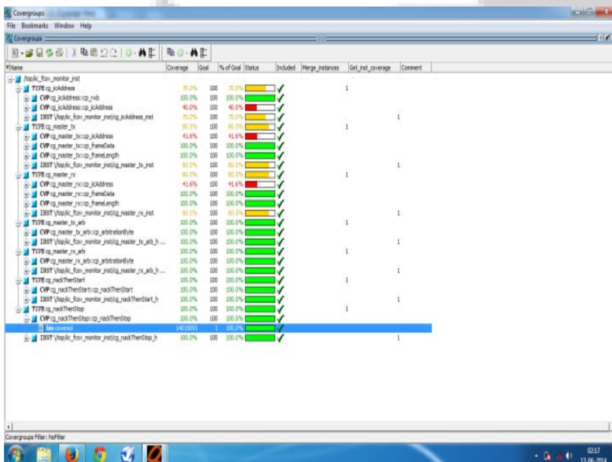

Figure 6: Functional Coverage of I2C

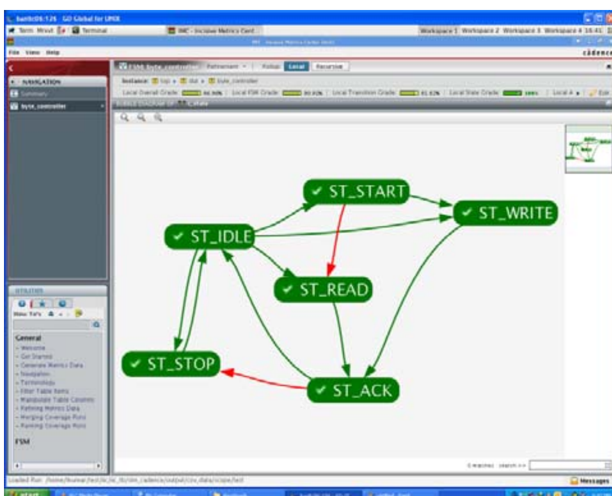

Figure 7: FSM Coverage of various states in I2C

7. Conclusions

The I2C IP core for intercommunication bus is designed using the Verilog for multi-master slave communication in full duplex mode. The verification environment is created using SystemVerilog constructors and UVM base classes.UVM Agents are crated for I2C.Sequences are generated using randomized constraints which cover all the necessary cases along with some corner cases. Functional coverage requires a detailed verification plan and much time creating the cover groups, analyzing the results, and modifying tests to create the proper stimulus. This may seem like a lot of work, but is less effort than would be required to write the equivalent directed tests. Additionally, the time spent in gathering coverage helps us better track our progress in verifying design. This methodology provides the coverage of the RTL design so as to acquire the fault free Protocol design of I2C and its reusable test environment, so that can be implemented in real time systems.

References

[1] IEEE Computer Society and the IEEE Standards Association Corporate Advisory Group, "IEEE Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language", IEEE Std 1800™-2012, 21 February 2013
[2] IEEE Computer Society "IEEE Standard Verilog® Hardware Description Language", IEEEStd 1364™-2005, 07 April 2006.
[3] Chris Spear, "SystemVerilog for Verification", Second Edition, 2008.
[4] Janick Bergeron, "Writing Testbenches: Functional Verification of HDL Models", Springer US, 28-Feb-2003
[5] Accellera Organization, "Universal Verification Methodology (UVM) 1.1 Class Reference", June 2011
[6] Philips Semiconductor, "I2C-bus specification and user manual", Rev. 6 - 4 April 2014
[7] Datasheet for Microchip 24LC256 – 2K I2C Serial EEPROM.
[8] [Online] http://www.testbench.in
[9] R. Herveille. I2C-Master Core Specification, Rev. 0.9, 2003
[10] P. Venkateswaran, "FPGA Based Efficient Interface Model for Scalefree Computer Network using I2C Bus Protocol"; Spl. Issue – Advances in Computer Sci. &Engg., National Polytechnic Institute, Mexico, Vol.23, pp. 191- 198, Nov. 21-24, 2006.
[11] Clifford E. Cummings, Tom Fitzpatrick, "OVM & UVM Techniques for Terminating Tests" DVCon 2011
[12] Prakash Rashinkar, Peter Paterson, Leena Singh, "SoC VerificationMethodology and Techniques" 2002
[13] Mentor Graphics® Verification Academy "Cookbook Online Methodology Documentation"
[14] Mentor Graphics Corporation, "Questa® SIM User's Manual", © 1991-2011

Author Profile

T Tarun Kumar received his B.E. degree in Electronics & Comm. Engineering from Atria Institute of Technology in 2012 and M. Tech degree in VLSI Design and Embedded System from Bangalore Institute of Technology in 2014.

C.Y. Gopinath did B.E. in Electronics & Communication from UVCE in 1985 and M. Tech in Industrial Electronics from SJCE, Mysore in 1989. He is currently Associate Professor in Bangalore Institute of Technology.

Licensed Under Creative Commons Attribution CC BY