**DESIGN OF ELEVATOR CONTROLLER**

**USING VERILOG HDL**

A

Mini Project Report

Submitted in the Partial Fulfillment of the

Requirements

For the Award of the Degree of

**Bachelor of Technology**

**in**

**Electronics and Communication Engineering**

**Submitted**

**By**

**SRIKANTH KALEMLA 10881A0452**

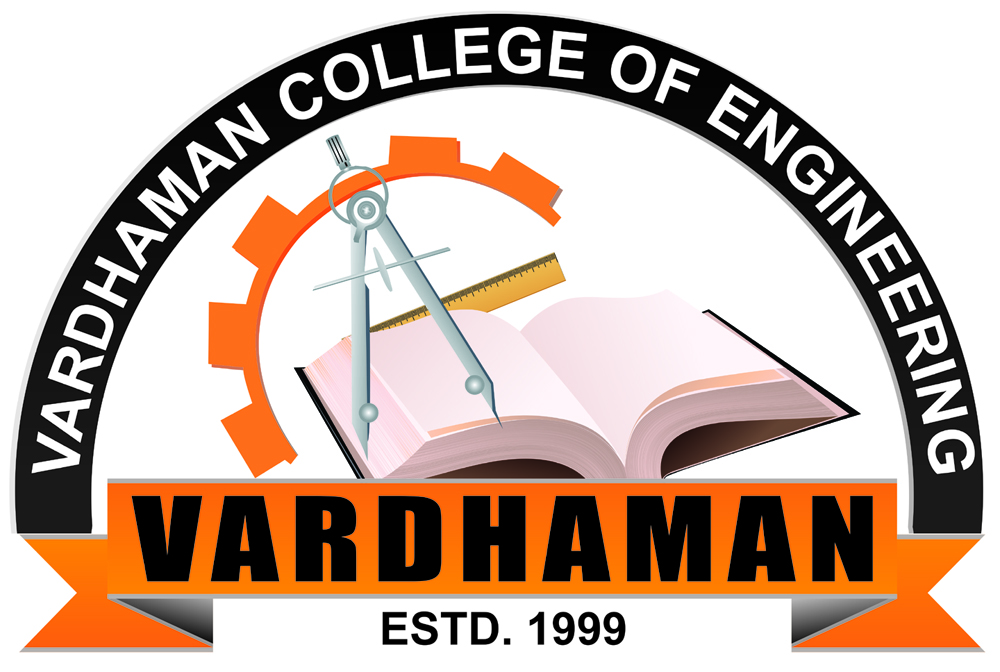
**VISHESH THAKUR SINGH 10881A0460**

**Under the Guidance of**

**MR. S. RAJENDAR**

**Associate Professor**

**Department of ECE**

****

**Department of Electronics and Communication Engineering**

**Vardhaman College of Engineering**

(AUTONOMOUS)

**(Approved by AICTE, Affiliated to JNTUH & Accredited by NBA)**

**2013 – 14**

**ACKNOWLEDGEMENT**

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

I express my heartfelt thanks to **Mr. S. Rajendar**, Associate Professor, technical seminar supervisor, for his suggestions in selecting and carrying out the in-depth study of the topic. His valuable guidance, encouragement and critical reviews really helped to shape this report to perfection.

I wish to express my deep sense of gratitude to **Dr. J. V. R. Ravindra**, Head of the Department for his able guidance and useful suggestions, which helped me in completing the technical seminar on time.

I also owe my special thanks to our Director **Prof. L. V. N. Prasad** for his intense support, encouragement and for having provided all the facilities and support.

Finally thanks to all my family members and friends for their continuous support and enthusiastic help.

Srikanth Kalemla

10881A0452

Vishesh Thakur Singh

10881A0460

**ABSTRACT**

The aim of the project is to design and implement an Elevator/Lift Controller using Verilog hardware descriptive language (HDL). The Elevator Controller is a device used to control a lift motion and to indicate the direction of motion, and the present floor level, etc. The device controls the lift motion by means of accepting the floor level as input and generate control signals (for control the lift motion) as output. The elevator controller is based on the concept of finite state machine technology. According to the FSM technology the elevator process can be defined with the help of different states. In the FSM technology there is a change from one state to another state likewise in the elevator there will be a change from one floor to another. Every possible way is assigned a path and the implemented based on FSM concept to write the program code for elevator controller. The whole program is designed in such a way that there are desirable switches in each floor and also inside the elevator to control the user commands. While the elevator is in the ground level in order to go upward direction we need only the up switch and nothing else. The same procedure we follow for the top floor. There is only one down switch there to move downward. But in between the ground floor and top floor all other floors contain two switches, one for moving up and another for moving down. Inside the elevator there must be at least ‘n’ switches for the implementation of an ‘n’ floor elevator controller. The elevator will move according to the desirable input that is given by the user. The design includes a simple scheme that aims at a good speed of response without requiring any extra logic circuitry.

# INTRODUCTION

The high growth of the semiconductor industry over the past two decades has put Very Large Scale Integration in demand all over the world. The basics of digital logic theory and techniques are easily understood by the design based on VLSI technology. These are the core fundamentals of the fast, high-speed complex digital circuits. As day to day the technology is gradually improving. So obviously the designs have to be made simpler for enjoying the benefits. To do that, an Elevator Controller is modeled. In the proposed design a VERILOG RTL code is developed to control the lift moment based on the request it will get. For that a finite state machine is developed to know from which state to state the controller is changing based on the requests from the end user. Lift is also called as Elevator or car. The design is based on the synchronous input which should be operating with a fixed sort of frequency. Finally the RTL is verified and implemented. In this work, the real-time three-lift controller will be modeled with Verilog HDL code using Finite-State machine (FSM) model to achieve the logic in optimized way.

## Defining Elevator

An elevator is a type of vertical [transport](http://en.wikipedia.org/wiki/Transport) equipment that efficiently moves people or goods between floors (levels, [decks](http://en.wikipedia.org/wiki/Deck_(building))) of a building, [vessel](http://en.wiktionary.org/wiki/vessel) or other structure. Elevators are generally powered by electric motors that either drive traction cables or counterweight systems like a [hoist](http://en.wikipedia.org/wiki/Hoist_(device)), or pump hydraulic fluid to raise a cylindrical piston like a [jack](http://en.wikipedia.org/wiki/Jack_(device)#Hydraulic_jack).

In agriculture and manufacturing, an elevator is any type of [conveyor](http://en.wikipedia.org/wiki/Conveyor) device used to lift materials in a continuous stream into bins or [silos](http://en.wikipedia.org/wiki/Silo). Several types exist, such as the chain and bucket [bucket elevator](http://en.wikipedia.org/wiki/Bucket_elevator" \o "Bucket elevator), grain auger [screw conveyor](http://en.wikipedia.org/wiki/Screw_conveyor) using the principle of [Archimedes' screw](http://en.wikipedia.org/wiki/Archimedes%27_screw), or the chain and paddles/forks of [hay elevators](http://en.wikipedia.org/wiki/Hay_elevator).

## Design

Some people argue that elevators began as simple rope or [chain](http://en.wikipedia.org/wiki/Link_chain) [hoists](http://en.wikipedia.org/wiki/Hoist_(device)). An elevator is essentially a platform that is either pulled or pushed up by a mechanical means. A modern day elevator consists of a cab (also called a "cage" or "car") mounted on a platform within an enclosed space called a shaft or sometimes a "hoistway". In the past, elevator drive mechanisms were powered by steam and water hydraulic pistons or by hand. In a "traction" elevator, cars are pulled up by means of rolling steel ropes over a deeply grooved [pulley](http://en.wikipedia.org/wiki/Pulley), commonly called a sheave in the industry. The weight of the car is balanced by a [counterweight](http://en.wikipedia.org/wiki/Counterweight). Sometimes two elevators are built so that their cars always move synchronously in opposite directions, and are each other's counterweight.

The friction between the ropes and the pulley furnishes the traction which gives this type of elevator its name. Hydraulic elevators use the principles of [hydraulics](http://en.wikipedia.org/wiki/Hydraulics) (in the sense of [hydraulic power](http://en.wikipedia.org/wiki/Hydraulic_power)) to pressurize an above ground or in-ground piston to raise and lower the car. Roped hydraulics uses a combination of both ropes and hydraulic power to raise and lower cars. Recent innovations include permanent magnet motors, machine room-less rail mounted gearless machines, and microprocessor controls.

The technology used in new installations depends on a variety of factors. Hydraulic elevators are cheaper, but installing cylinders greater than a certain length becomes impractical for very-high lift hoistways. For buildings of much over seven storys, traction elevators must be employed instead. Hydraulic elevators are usually slower than traction elevators.

Elevators are a candidate for [mass customization](http://en.wikipedia.org/wiki/Mass_customization). There are economies to be made from [mass production](http://en.wikipedia.org/wiki/Mass_production) of the components, but each building comes with its own requirements like different number of floors, dimensions of the well and usage patterns.

## What is an Elevator Controller?

An elevator is a device designed as a convenience appliance that has evolved to become an unavoidable feature of modern day urban life. An elevator is defined as, “A machine that carries people or goods up and down to different levels in a building or mine”. While a standalone elevator Isa simple electro-mechanical device, an elevator system may consist of multiple standalone elevator units whose operations are controlled and coordinated by a master controller. Such controllers are designed to operate with maximum efficiency in terms of service as well as resource utilization. This project details the design of an elevator controller using VERILOG. The Elevators/Lifts are used in multi store buildings as a means of transport between various floors. Elevator is a device designed as a convenience appliance that has evolved to become an unavoidable features of modern day in urban life normally .The lifts is controlled by Microprocessor based systems, which are costlier. It is proposed to design a low cost and compact dedicated controller. The Elevator Controller is a device used to control a lift motion and to indicate the direction of motion, and the present floor level, etc. The device control the lift motion by means of accepting the floor level as input and generate control signals (for control the lift motion) as output. We developed a VERILOG code for 3-story elevator control system for the cases of elevator moving up and down. The design and simulation of the Elevator controller can be performed using VERILOG. Also the Timings of various signals can be verified. VERILOG is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits.

The key advantage of VERILOG when used for systems design is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis calculation block can be used in many other projects. However, many formational and functional block parameters can be tuned that are capacity parameters, memory size, element base, block composition and interconnection structure.

## Organization of the Report

This report presents an overview of the Elevator/Lift Controller and its design. We analyze how it is correlated to very large scale integration technology. The report basically starts with the introduction of elevator and its evolution also giving primary importance to finite state machines. Then it explains implementation of elevator controller using Verilog HDL and Cadence tools. The report is organized as follows:

Chapter 1: Introduction - This chapter briefly explains the overview of the report.

Chapter 2: Evolution and Description of Elevator - This chapter describes the brief history of elevator and also analyses its objective and working.

Chapter 3: Hardware Descriptive Language-Verilog - This chapter gives the detailed introduction of hardware descriptive language Verilog.

Chapter 4: The Principle of Elevator controller - This chapter describes the basic principle involved in the working of elevator controller that is the finite state machines.

Chapter 5: Implementation using Cadence Tools – This chapter lists the tools used

and the simulation steps involved in designing. It also includes the Verilog code of Elevator Controller with relevant waveforms

Chapter 6: Conclusions - This chapter summarizes the major accomplishments of this report.

# Evolution and Description OF ELEVATOR

The first reference elevator was invented by Archimedes in 312. From some literacy source, elevator were developed as cable on a hemp rope and powered by hand or by through animals. This type of elevator was installed in the Sinai Monastery of Egypt. In the 17th century, the very small type elevators were placed in the building of England and France. In 1793, Lvan Kuliben created an elevator with the screw lifting mechanism for the winter place of Saint Petersburg. In 1816, an elevator was established in the main building of Sub-moscow village called Arkhamgelskoye. In the middle 1800’s, there were many types of curd elevators that - 2 -carried freight. Most of them ran hydraulically. The first hydraulic elevators used a plunger below the car to raise or lower the elevator. A pump applied water pressure to a plunger, or steel column, inside a vertical cylinder. In 1852, Elisha Otis introduced the safety elevator, which prevented the fall of the cab, if the cable broke. In 1857 March 23rd, the first Otis passenger elevator was installed in New York City. The first electric elevator was built by Werner Von Siemens in 1880.

In 1874, J.W. Meaker patented a method which permitted elevator doors to open and close safely. In 1882, when hydraulic power was a well established technology, a company later named the London Hydraulic Power Company was formed. In 1929, Clarence Conrad Crispen, with Inclinator Company of America, created the first residential elevator.

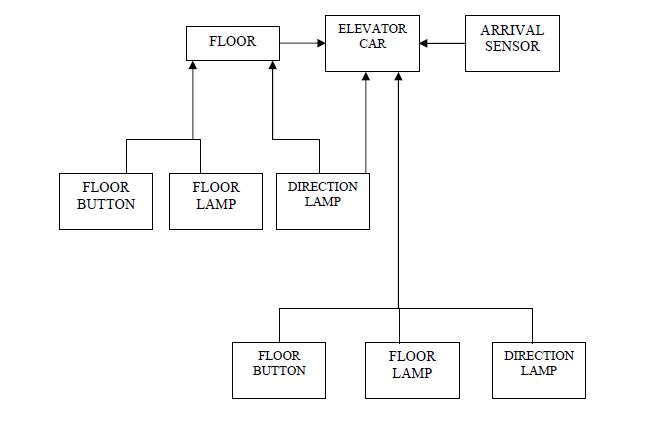


Figure 2.1 Elevator System Overview

Figure 2.1 shows the elevator system overview. This figure consists of floor where passenger wants to visit. Elevator car moves it either upward or downward direction. The arrival sensor detected the arrival of the elevator to the respective floor. Floor button is used to take the elevator to the respective floor. Floor lamp shows the indication of floor and direction lamp shows the direction of elevator movement, whether it is upward or downward direction. Elevator button is used for moving the elevator car either in upward and downward direction. Based on the elevator switch pressed, the elevator car is moved either in upward and downward direction. D.C. Motor is another important component of elevator system. Based on the switch pressed, the D.C. Motor either moves in forward and reverse direction to move the elevator in either upward or downward direction. Door of the elevator system is one of the important factors of elevator system. When elevator car stops in particular floor, the door of the elevator is opened for passenger to be come out and come in to the elevator car. When a particular car is reached to the particular floor, the elevator controller stops that car.

## Objective

An elevator system is a vertical transport vehicle that efficiently moves people or goods between floors of a building. They are generally powered by electric motors. The most popular elevator is the rope elevator. In the rope elevator, the car is raised and lowered by transaction with steel rope. Elevators also have electromagnetic brakes that engage, when the car comes to a stop. The electromagnetic actually keeps the brakes in the open position. Instead of closing them with the design, the brakes will automatically clamp shut if the elevator loses power. The whole program is designed in such a way that there are desirable switches in each floor and also inside the elevator to control the user commands. While the elevator is in the ground level in order to go upward direction we need only the up switch and nothing else. The same procedure we follow for the top floor. There is only one down switch there to move downward. But in between the ground floor and top floor all other floors contain two switches, one for moving up and another for moving down. The elevator will move according to the desirable input that is given by the user. The design includes a simple scheme that aims at a good speed of response without requiring any extra logic circuitry. Elevators also have automatic braking systems near the top and the bottom of the elevator shaft. Many modern elevators are controlled by a computer. The computers job is to process all of the relevant information about the elevator and turn the motor correct amount to move the elevator car in correct position. In order to do this the computer needs to know at least three things those are

i) Where people want to go

ii) Where each floor is

iii) Where the elevator car is

Finding out where people want to go is very easy. The buttons in the elevator car and the buttons in each floor are all wired to the computer, when anyone presses these buttons, the computer logs this request.

## Description

When User presses an elevator button, the elevator button sensor sends the elevator button request to the system, identifying the destination floor the user wishes to visit. When any new request comes, this new request is added to the list of floors to visit. If the elevator is stationary, the system determines in which direction the system should move in order to service the next request. The system 1commands the elevator door to close, when user presses the elevator door closed button. When the door has closed, the system commands the motor to start moving the elevator, either in up and down direction, based on switch pressed. When the elevator moves between floors, the arrival sensor detects that the elevator is approaching a floor and notifies the system to stop the elevator and open the door of the elevator system. Figures 2.2 show the elevator dispatching strategy.

## 3.jpg

Figure 2.2 Elevator Dispatching Strategy

## Controlling Elevators

Elevators are typically controlled from the inside/outside by a call box, which has up and down buttons, at each stop. When pressed at a certain floor, the button calls the elevator to pick up more passengers. If the particular elevator is currently serving traffic in a certain direction, it will only answer calls in the same direction unless there are no more calls beyond that floor.

In a group of two or more elevators, the call buttons may be linked to a central dispatch computer, such that they illuminate and cancel together. This is done to ensure that only one car is called at one time.

Key switches may be installed on the ground floor so that the elevator can be remotely switched on or off from the outside.

In destination control systems, one selects the intended destination floor (in lieu of pressing *"up"* or *"down"*) and is then notified which elevator will serve their request.

## The Elevator Algorithm

The [elevator algorithm](http://en.wikipedia.org/wiki/Elevator_algorithm), a simple [algorithm](http://en.wikipedia.org/wiki/Algorithm) by which a single elevator can decide where to stop, is summarized as follows:

* Continue traveling in the same direction while there are remaining requests in that same direction.
* If there are no further requests in that direction, then stop and become idle, or change direction if there are requests in the opposite direction.

The elevator algorithm has found an application in computer [operating systems](http://en.wikipedia.org/wiki/Operating_system) as an algorithm for scheduling [hard disk](http://en.wikipedia.org/wiki/Hard_disk) requests. Modern elevators use more complex [heuristic algorithms](http://en.wikipedia.org/wiki/Heuristic_(computer_science)) to decide which request to service next.

# hardware descriptive language-verilog

Verilog HDL is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from the algorithmic-level to the gate-level to the switch-level. The complexity of the digital system being modeled could vary from that of a simple gate to a complete electronic digital system, or anything in between. The digital system can be described hierarchically and timing can be explicitly modeled within the same description.

The Verilog HDL language includes capabilities to describe the behavior-al nature of a design, the dataflow nature of a design, a design's structural composition, delays and a waveform generation mechanism including aspects of response monitoring and verification, all modeled using one single language. In addition, the language provides a programming language interface through which the internals of a design can be accessed during simulation including the control of a simulation run.

The language not only defines the syntax but also defines very clear simulation semantics for each language construct. Therefore, models written in this language can be verified using a Verilog simulator. The language inherits many of its operator symbols and constructs from the C programming language. Verilog HDL provides an extensive range of modeling capabilities, some of which are quite difficult to comprehend initially. However, a core subset of the language is quite easy to learn and use. This is sufficient to model most applications.

The Verilog HDL language was first developed by Gateway Design Automation in 1983 as hardware are modeling language for their simulator product, At that time ,it was a proprietary language. Because of the popularity of the, simulator product, Verilog HDL gained acceptance as a usable and practical language by a number of designers. In an effort to increase the popularity of the language, the language was placed in the public domain in 1990. Verilog HDL provides an extensive range of modeling capabilities, some of which are quite difficult to comprehend initially.

Open Verilog International (OVI) was formed to promote Verilog. In 1992 OVI decided to pursue standardization of Verilog HDL as an IEEE standard. This effort was successful and the language became an IEEE standard in 1995. The complete standard is described in the Verilog hardware description language reference manual. The standard is called std. 1364-1995.

## Overview

Hardware description languages such as Verilog differ from software [programming languages](http://en.wikipedia.org/wiki/Programming_language) because they include ways of describing the propagation of time and signal dependencies (sensitivity). There are two types of assignment operators; a blocking assignment (=), and a non-blocking (<=) assignment. The non-blocking assignment allows designers to describe a state-machine update without needing to declare and use temporary storage variables. Since these concepts are part of Verilog's language semantics, designers could quickly write descriptions of large circuits in a relatively compact and concise form. At the time of Verilog's introduction (1984), Verilog represented a tremendous productivity improvement for circuit designers who were already using graphical schematic software and specially written software programs to document and [simulate electronic circuits](http://en.wikipedia.org/wiki/Electronic_circuit_simulation).

The designers of Verilog wanted a language with syntax similar to the [C programming language](http://en.wikipedia.org/wiki/C_(programming_language)), which was already widely used in engineering software development. Like C, Verilog is [case-sensitive](http://en.wikipedia.org/wiki/Case-sensitive) and has a basic [preprocessor](http://en.wikipedia.org/wiki/Preprocessor) (though less sophisticated than that of ANSI C/C++). Its [control flow](http://en.wikipedia.org/wiki/Control_flow) [keywords](http://en.wikipedia.org/wiki/Keyword_(computer_programming)) (if/else, for, while, case, etc.) are equivalent, and its [operator precedence](http://en.wikipedia.org/wiki/Operator_precedence) is compatible. Syntactic differences include variable declaration (Verilog requires bit-widths on net/reg types), demarcation of procedural blocks (begin/end instead of curly braces {}), and many other minor differences.

A Verilog design consists of a hierarchy of modules. Modules encapsulate design hierarchy, and communicate with other modules through a set of declared input, output, and bidirectional ports. Internally, a module can contain any combination of the following: net/variable declarations (wire, reg, integer, etc.), concurrent and sequential statement blocks, and instances of other modules (sub-hierarchies). Sequential statements are placed inside a begin/end block and executed in sequential order within the block. However, the blocks themselves are executed concurrently, making Verilog a [dataflow language](http://en.wikipedia.org/wiki/Dataflow_language).

Verilog's concept of 'wire' consists of both signal values (4-state: "1, 0, floating, undefined") and strengths (strong, weak, etc.). This system allows abstract modeling of shared signal lines, where multiple sources drive a common net. When a wire has multiple drivers, the wire's (readable) value is resolved by a function of the source drivers and their strengths.

A subset of statements in the Verilog language are [synthesizable](http://en.wikipedia.org/wiki/Logic_synthesis). Verilog modules that conform to a synthesizable coding style, known as RTL (register-transfer level), can be physically realized by synthesis software. Synthesis software algorithmically transforms the (abstract) Verilog source into a [netlist](http://en.wikipedia.org/wiki/Netlist), a logically equivalent description consisting only of elementary logic primitives (AND, OR, NOT, flip-flops, etc.) that are available in a specific [FPGA](http://en.wikipedia.org/wiki/FPGA) or [VLSI](http://en.wikipedia.org/wiki/VLSI) technology. Further manipulations to the netlist ultimately lead to a circuit fabrication blueprint (such as a [photo mask set](http://en.wikipedia.org/wiki/Mask_set) for an [ASIC](http://en.wikipedia.org/wiki/Application-specific_integrated_circuit) or a [bitstream](http://en.wikipedia.org/wiki/Bitstream" \o "Bitstream) file for an [FPGA](http://en.wikipedia.org/wiki/FPGA)).

## Capabilities of Verilog HDL

Listed below are the major capabilities of the Verilog hardware description:

* Primitive logic gates, such as and, or and nand, are built-in into the language.
* Flexibility of creating a user-defined primitive (UDP). Such a primitive could either be a combinational logic primitive or a sequential logic primitive.
* Switch-level modeling primitive gates, such as pmos and nmos, are also built- in into the language.
* A design can be modeled in three different styles or in a mixed style. These styles are: behavioral style modeled using procedural constructs; dataflow style - modeled using continuous assignments; and structural style modeled using gate and module instantiations.
* There are two data types in Verilog HDL; the net data type and the register data type. The net type represents a physical connection between structural elements while a register type represents an abstract data storage element.
* Figure.3-1 shows the mixed-level modeling capability of Verilog HDL, that is, in one design; each module may be modeled at a different level.

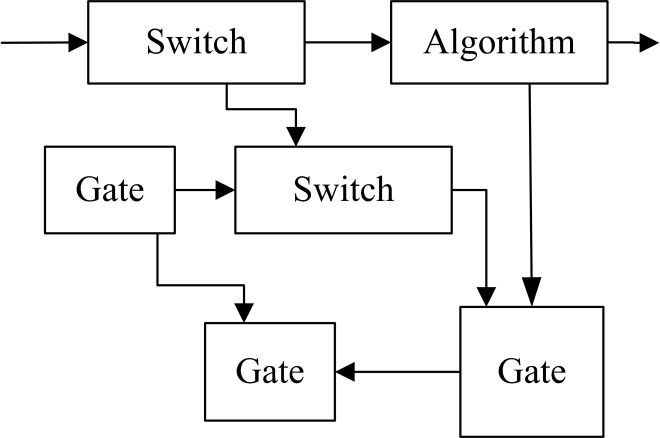


Figure 3.1 Mixed level modeling

* Verilog HDL also has built-in logic functions such as & (bitwise-and) and I (bitwise-or).
* High-level programming language constructs such as conditionals, case statements, and loops are available in the language.
* Notion of concurrency and time can be explicitly modeled.
* Powerful file read and write capabilities fare provided.
* The language is non-deterministic under certain situations, that is, a model may produce different results on different simulators; for example, and the ordering of events on an event queue is not defined by the standard.

## Design Abstraction Levels of Verilog

Verilog supports designing at many different levels of abstraction. Three of them are very important:

* Behavioral level
* Register-Transfer level
* Gate level

### Behavioral level

This level describes a system by concurrent algorithms (Behavioral). Each algorithm itself is sequential, that means it consists of a set of instructions that are executed one after the other. Functions, Tasks and Always blocks are the main elements. There is no regard to the structural realization of the design.

### Register-Transfer level

Designs using the Register-Transfer Level specify the characteristics of a circuit by operations and the transfer of data between the registers. An explicit clock is used. RTL design contains exact timing bounds: operations are scheduled to occur at certain times. Modern RTL code definition is "Any code that is synthesizable is called RTL code".

### Gate level

Within the logic level the characteristics of a system are described by logical links and their timing properties. All signals are discrete signals. They can only have definite logical values (`0', `1', `X', `Z`). The usable operations are predefined logic primitives (AND, OR, NOT etc gates). Using gate level modeling might not be a good idea for any level of logic design. Gate level code is generated by tools like synthesis tools and this netlist is used for gate level simulation and for backend.

## Simulation

Simulation is the process of verifying the functional characteristics of models at any level of abstraction. We use simulators to simulate the Hardware models. To test if the RTL code meets the functional requirements of the specification, we must see if all the RTL blocks are functionally correct. To achieve this we need to write a testbench, which generates clk, reset and the required test vectors. A sample testbench for a counter is shown below. Normally we spend 60-70% of time in design verification.

We use the waveform output from the simulator to see if the DUT (Device Under Test) is functionally correct. Most of the simulators come with a waveform viewer. As design becomes complex, we write self checking testbench, where testbench applies the test vector, then compares the output of DUT with expected values.

There is another kind of simulation, called **timing simulation**, which is done after synthesis or after P&R (Place and Route). Here we include the gate delays and wire delays and see if DUT works at rated clock speed. This is also called as **gate level simulation**.

## Synthesis

Synthesis is the process of constructing a gate level netlist from a register- transfer level model of a circuit described in Verilog HDL. A synthesis system may as an intermediate step, generate a netlist that is comprised of register-transfer level blocks such as flip-flops, arithmetic-logic-units, and multiplexers, interconnected by wires. In such a case, a second program called the RTL module builder is necessary. The purpose of this builder is to build, or acquire from a library of predefined components, each of the required RTL blocks in the user- specified target technology.

A mapping mechanism or a construction mechanism has to be provided that translates the Verilog HDL elements into their corresponding hardware elements. Synthesis is the process in which synthesis tools like design compiler take RTL in Verilog or VHDL, target technology, and constrains as input and maps the RTL to target technology primitives. Synthesis tool, after mapping the RTL to gates, also do the minimal amount of timing analysis to see if the mapped design is meeting the timing requirements. (Important thing to note is, synthesis tools are not aware of wire delays, they only know of gate delays). After the synthesis there are a couple of things that are normally done before passing the netlist to backend (Place and Route).

The figure shown below lists the basic elements of Verilog HDL and the elements used in hardware. A mapping mechanism or a construction mechanism has to be provided that translates the Verilog HDL elements into their corresponding hardware elements as shown in figure.

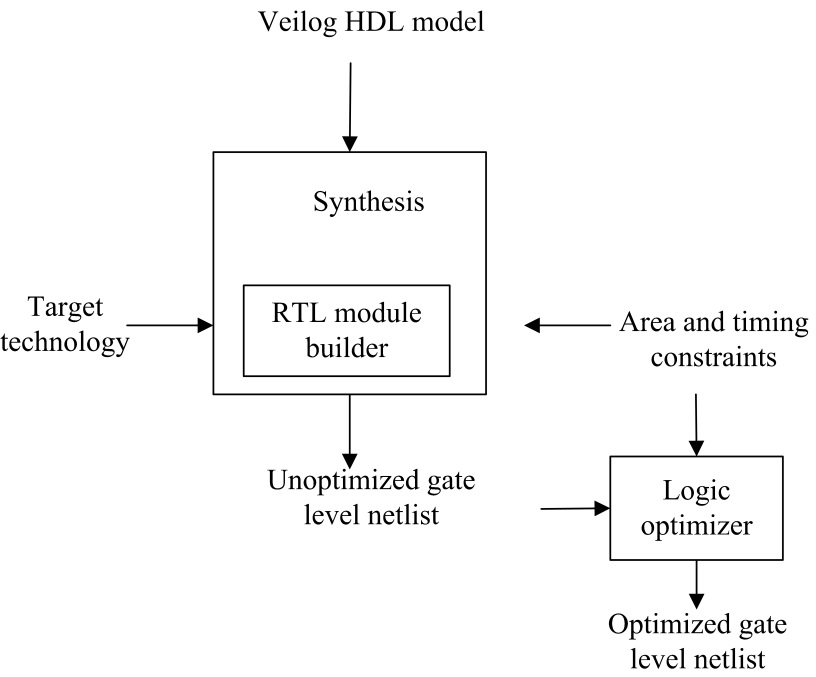


Figure 3.2 Description of Synthesis process

## Summary

The Verilog HDL language includes capabilities to describe the behavioral nature of a design, the dataflow nature of a design, a design's structural composition, delays and a waveform generation mechanism including aspects of response monitoring and verification, all modeled using one single language. This project details the design of an elevator controller using Verilog HDL. The Elevators/Lifts are used in multi store buildings as a means of transport between various floors. Elevator is a device designed as a convenience appliance that has evolved to become an unavoidable features of modern day in urban life normally .The lifts is controlled by Microprocessor based systems, which are costlier. It is proposed to design a low cost and compact dedicated controller. The Elevator Controller is a device used to control a lift motion and to indicate the direction of motion, and the present floor level, etc. The device control the lift motion by means of accepting the floor level as input and generate control signals (for control the lift motion) as output. We developed a Verilog HDL code for 3-story elevator control system for the cases of elevator moving up and down. The design and simulation of the Elevator controller can be performed using Verilog HDL. Also the Timings of various signals can be verified. Verilog HDL is a hardware description language used in electronic design automation to describe digital and mixed-signal systems such as field-programmable gate arrays and integrated circuits.

The key advantage of Verilog HDL when used for systems design is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis calculation block can be used in many other projects. However, many formational and functional block parameters can be tuned that are capacity parameters, memory size, element base, block composition and interconnection structure.

# 

Figure 3.3 Typical Design Flow

# THE PRINCIPLE OF ELEVATOR CONTROLLER

Elevator controller is an elementary system consisting of elevator serving 3 floors. The elevator car has a pair of control buttons (up /down) for moving the elevator up and down. The floors also have call buttons to call for the service of the elevator system. The following principles have been applied during the design of the elevator controller: The floors are defined as first floor and second etc.

1. A floor call is serviced using the elevator.
2. Upon arrival at a floor, the doors open immediately.
3. Doors remain open before closure.
4. If an obstruction is detected when door is about to close, it remains open
5. Each elevator car is treated as a sub-system controlled by the controller.
6. Elevator Up / Down buttons are connected to elevator units.
7. Each door unit is treated as a subsystem controlled by the respective elevator car.

The elevator controller is based on the concept of finite state machine technology. According to the FSM technology the elevator process can be defined with the help of different states. In the FSM technology there is a change from one state to another state likewise in the elevator there will be a change from one floor to another. Every possible way is assigned a path and the implemented based on FSM concept to write the program code for elevator controller. The whole program is designed in such a way that there are desirable switches in each floor and also inside the elevator to control the user commands. While the elevator is in the ground level in order to go upward direction we need only the up switch and nothing else. The same procedure we follow for the top floor. There is only one down switch there to move downward. But in between the ground floor and top floor all other floors contain two switches, one for moving up and another for moving down. Inside the elevator there must be at least ‘n’ switches for the implementation of an ‘n’ floor elevator controller. The elevator will move according to the desirable input that is given by the user. The design includes a simple scheme that aims at a good speed of response without requiring any extra logic circuitry.

## Basic Principle

Elevators themselves are simple devices, and the basic lifting systems have not changed much in over 50 years. The control systems, however, have changed substantially to improve safety and speed of operation. Elevators are designed for a specific building, taking into account such factors as the height of the building, the number of people traveling to each floor, and the expected periods of high usage.   
Controllers can also be programmed to respond differently at different times of the day. For example, the elevator controller in a busy office building will receive a preponderance of calls from the ground floor in the morning, when workers are arriving and need to go to their workplaces on the upper floors. In that case, the controller will be programmed to send all unassigned cars to the ground floor, rather than have them return to a home floor in their sector. Later in the day, a different set of instructions can be used to send unassigned elevators to different sectors, since passengers leaving the building will be much more evenly distributed among the floors than in the morning.

## Introduction

The elevator controller designing in HDL has been a great challenge. The Moore model did simplify the approach, but in order to operate the system in optimized way and accept user request by the nearest elevator, highlighted the complexities involved in it.

The elevator algorithm respects the constraints defined and works in alignment to the assumptions made.

The timer implementation as a part of the code remained an unresolved issue, as the coding should also have been synthesizable. The other issues that could not be addressed were to store the request and provide to the elevator at a later stage if the elevator is in busy state at that time. As the elevator system in for only two floors, the same could also be automated that once user gets into the lift, he does not need to request for the destination.

In general, the elevator controller has to control a larger group in a multi-floor building. Then, the capabilities of designing the algorithm of the designer decide the functionality of the elevator system.

Every possible way is assigned a path and the implemented based on FSM concept to write the program code for elevator controller. The whole program is designed in such a way that there are desirable switches in each floor and also inside the elevator to control the user commands.

## Assumptions of an Elevator System

The Elevator System does face some conflict in the operation that which car should take the request when both are at the same positions. So, some assumptions are implemented in the elevator algorithm. The assumptions considered are:   
**Elevator Priority:**

Elevators are prioritized for the requests. The elevator1 has a priority for the first floor request and the elevator2 for the request from second floor.  
**Default State:**

Elevator1 on first floor with closed door and the elevator2 at second floor with closed door. This default position provides quick response to the request coming at any of the two floors.

**Closing the Elevator Door:**

Door of the elevator close after some time duration, defined by the timer. By default the timer should be 0 and when to close the door, it should be 1. The system also checks for any obstruction if present between the doors. Both, the timer and obstructions, are implemented using switch.

Thus, to close the door the door of an elevator, the respective timer needs to be triggered to high state and the obstruction should be 0.

## Finite State Machine

A finite-state machine (FSM) is a mathematical abstraction sometimes used to design digital logic or computer programs. It is a behavior model composed of a finite number of states, transitions between those states, and actions. The finite-state machine is similar to a flow graph in which one can inspect the way logic runs when certain conditions are met. It has finite internal memory, an input feature (reading symbols in a sequence, one at a time without going backward), and an output feature, which may be in the form of a user interface, once the model is implemented. The number and names of the states typically depend on the different possible states of the memory, e.g. if the memory is three bits long, there are 8 possible states. The state that reads the first symbol of the input sequence is called the start state and the state which signifies the successful operation of the machine is called the accept state. Every state may lead to a different one depending on the next input symbol and output can be provided during the transition. State machines can also have probabilistic transitions, fuzzy states and other oddities.

This chapter introduces finite-state machines, a primitive, but useful computational model for both hardware and certain types of software. We also discuss regular expressions, the correspondence between non-deterministic and deterministic machines, and more on grammars. Finally, we describe typical hardware components that are essentially physical realizations of finite-state machines.

Finite-state machines provide a simple computational model with many applications. Recall the definition of a Turing machine: a finite-state controller with a movable read/write head on an unbounded storage tape. If we restrict the head to move in only one direction, we have the general case of a finite-state machine. The sequence of symbols being read can be thought to constitute the input, while the sequence of symbols being written could be thought to constitute the output. We can also derive output by looking at the internal state of the controller after the input has been read.

In general, a state machine is any device that stores the status of something at a given time and can operate on input to change the status and/or cause an action or output to take place for any given change. A [computer](http://searchwinit.techtarget.com/definition/computer) is basically a state machine and each machine instruction is input that changes one or more states and may cause other actions to take place. Each computer's data register stores a state. The read-only memory from which boot program is loaded stores a state (the boot program itself is an initial state). The operating system is itself a state and each [application](http://searchsoftwarequality.techtarget.com/definition/application) that runs begins with some initial state that may change as it begins to handle input. Thus, at any moment in time, a computer system can be seen as a very complex set of states and each program in it as a state machine. In practice, however, state machines are used to develop and describe specific device or program interactions.

To summarize it, a state machine can be described as:

* An initial state or record of something stored someplace
* A set of possible input events
* A set of new states that may result from the input
* A set of possible actions or output events that result from a new state

A finite state machine is one that has a limited or finite number of possible states. (An infinite state machine can be conceived but is not practical.) A finite state machine can be used both as a development tool for approaching and solving problems and as a formal way of describing the solution for later developers and system maintainers.

## Types of finite state machines

Finite state machines are broadly classified into two broad categories as follows:

* Mealy state machine
* Moore state machine

### Mealy State Machine

A **Mealy machine** is a [finite-state machine](http://en.wikipedia.org/wiki/Finite-state_machine) whose output values are determined both by its current [state](http://en.wikipedia.org/wiki/State_(computer_science)) and the current inputs.

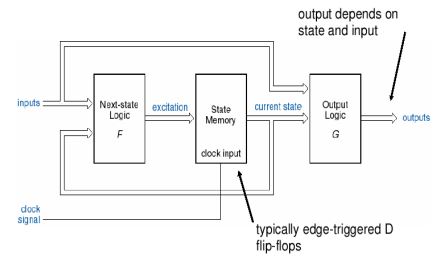


Figure 4.1 Mealy State Machine

### Moore State Machine

 A **Moore machine** is a [finite-state machine](http://en.wikipedia.org/wiki/Finite-state_machine) whose output values are determined solely by its current [state](http://en.wikipedia.org/wiki/State_(computer_science)).

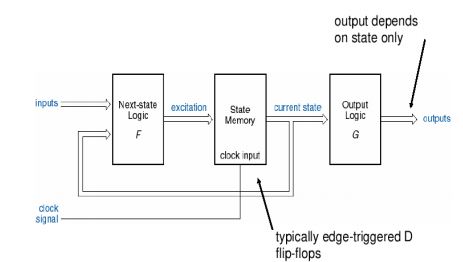


Figure 4.2 Moore State Machine

## Design Steps of Finite State Machine

* Determine the inputs /outputs. Determine the states and give them mnemonic names.
* Draw up a state diagram and a next state table.
* Render the inputs, outputs and states in binary format.
* Draw an excitation table - a truth table showing the inputs and current state
* binary values as inputs and the desired next state binary values as the
* outputs.
* Use K-maps to obtain produce minimal next state and output
* combinational logic.
* Use the standard VERILOG formulation to simulate your design and check for correct operation. Revise as appropriate.

# Implementation using cadence tools

## Tools Used

1. Pc installed with Linux operating system
2. Installed cadence tools:

* Ncvlog – For checking errors
* Ncverilog – For execution of code
* Simvision – To View waveforms

## Coding Steps

1. Create directory structure for the project as below



Figure 5.1 Project Directory Structure

1. Write RTL code in a text file and save it as .v extension in RTL directory
2. Write code for testbench and store in TB directory

## Simulation steps

The Commands that are used in cadence for the execution are

1. Initially we should mount the server using “mount -a”.
2. Go to the C environment with the command “csh” //c shell.
3. The source file should be opened by the command “source /root/cshrc”.
4. The next command is to go to the directory of cadence\_dgital\_labs

#cd .../../cadence\_digital\_labs/

1. Then check the file for errors by the command “ncvlog ../rtl/filename.v -mess”.
2. Then execute the file using “ncverilog +access +rwc ../rtl/filename.v ../tb/file\_tb.v +nctimescale +1ns/1ps

Rwc –read write command Gui- graphical unit interface

1. After running the program we open simulation window by command “Simvision &".

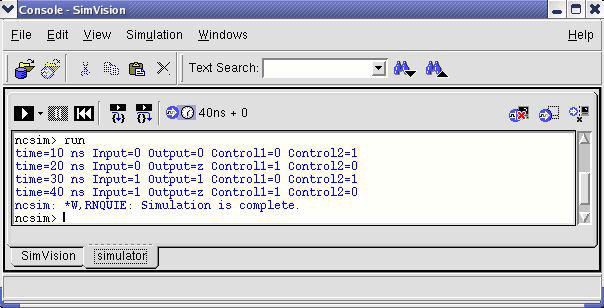


Figure 5.2 Simulation Window

1. After the simulation the waveforms are shown in the other window.



Figure 5.3 Waveform Window

## Elevator controller code

**module** elevator**(input** clk**,**rst**,**inG**,**in1**,**in2**,**in3**,**in4**,**in5**,**in6**,**in7**,**inopen**,**inclose**,** **output** open**,**close**,**up**,**down**);**

**parameter** IDLE**=4'd8,**OPEN**=4'd9,**CLOSE**=4'd10,**UP**=4'd11,**DOWN**=4'd12,**WAIT**=4'd13;**

**parameter** G**=3'd0,**F1**=3'd1,**F2**=3'd2,**F3**=3'd3,**F4**=3'd4,**F5**=3'd5,**F6**=3'd6,**F7**=3'd7;**

**reg[3:0]** pstate**,**nstate**;**

**reg[2:0]** pfloor**,**nfloor**;**

**reg[6:0]** count**;**

**reg** reached**,**overload**=0,**dir**=1;**

//present state

**always@(posedge** clk **or** **negedge** rst**)**

**begin**

pstate**<=**rst**?**nstate**:**IDLE**;**

**if(!**rst**)** pfloor**=**G**;**

**else** **if(**pstate**==**OPEN**)**

pfloor**=**nfloor**;**

**end**

//next state

**always@(\*)begin**

nstate**=**IDLE**;**

**case(**pstate**)**

IDLE**:begin**

**if(**inopen)

nstate**=**OPEN**;**

**else** **if(**pfloor**<**nfloor**)**

nstate**=**UP**;**

**else** **if(**pfloor**>**nfloor**)**

nstate**=**DOWN**;**

**else** **if(**pfloor**==**nfloor**)**

**begin**

**if(**inG**|**in1**|**in2**|**in3**|**in4**|**in5**|**in6**|**in7**)**

nstate**=**OPEN**;**

**end**

**else** nstate**=**IDLE**;**

**end**

OPEN**:begin**

**if(**inclose**)**

nstate**=**CLOSE**;**

**else** nstate**=**WAIT**;**

**end**

CLOSE**:begin**

**if(**inopen**)**

nstate**=**OPEN**;**

**else** nstate**=**IDLE**;**

**end**

UP**:begin**

**if(**reached**)**

nstate**=**OPEN**;**

**else** nstate**=**UP**;**

**end**

DOWN**:begin**

**if(**reached**)**

nstate**=**OPEN**;**

**else** nstate**=**DOWN**;**

**end**

WAIT**:begin**

**if(**overload**)**

nstate**=**WAIT**;**

**else** nstate**=**CLOSE**;**

**end**

**default:** nstate**=**IDLE**;**

**endcase**

**end**

//next floor

**always** **@(\*)** **begin**

**if(**pstate**==**IDLE**)**

**begin**

**repeat(2)**

**if(**dir**)**

**begin**

**case(**pfloor**)**

G**:**nfloor**=**in1**?**F1**:**in2**?**F2**:**in3**?**F3**:**in4**?**F4**:**in5**?**F5**:**in6**?**F6**:**in7**?**F7**:**G**;**

F1**:begin** nfloor**=**in2**?**F2**:**in3**?**F3**:**in4**?**F4**:**in5**?**F5**:**in6**?**F6**:**in7**?**F7**:**F1**;**

**if(!(**in2**||**in3**||**in4**||**in5**||**in6**||**in7**))** dir**=0;end**

F2**:begin** nfloor**=**in3**?**F3**:**in4**?**F4**:**in5**?**F5**:**in6**?**F6**:**in7**?**F7**:**F2**;**

**if(!(**in3**||**in4**||**in5**||**in6**||**in7**))** dir**=0;end**

F3**:begin** nfloor**=**in4**?**F4**:**in5**?**F5**:**in6**?**F6**:**in7**?**F7**:**F3**;**

**if(!(**in4**||**in5**||**in6**||**in7**))** dir**=0;end**

F4**:begin** nfloor**=**in5**?**F5**:**in6**?**F6**:**in7**?**F7**:**F4**;**

**if(!(**in5**||**in6**||**in7**))** dir**=0;end**

F5**:begin** nfloor**=**in6**?**F6**:**in7**?**F7**:**F5**;**

**if(!(**in6**||**in7**))** dir**=0;end**

F6**:begin** nfloor**=**in7**?**F7**:**F6**;**

**if(!**in7**)** dir**=0;end**

F7**:begin** nfloor**=**F7**;**

**if(!**in7**)**dir**=0;end**

**default:**nfloor**=**G**;**

**endcase**

**end**

**else**

**begin**

**case(**pfloor**)**

F7**:**nfloor**=**in6**?**F6**:**in5**?**F5**:**in4**?**F4**:**in3**?**F3**:**in2**?**F2**:**in1**?**F1**:**inG**?**G**:**F7**;**

F6**:begin** nfloor**=**in5**?**F5**:**in4**?**F4**:**in3**?**F3**:**in2**?**F2**:**in1**?**F1**:**inG**?**G**:**F6**;**

**if(!(**inG**||**in1**||**in2**||**in3**||**in4**||**in5**))** dir**=1;end**

F5**:begin** nfloor**=**in4**?**F4**:**in3**?**F3**:**in2**?**F2**:**in1**?**F1**:**inG**?**G**:**F5**;**

**if(!(**inG**||**in1**||**in2**||**in3**||**in4**))** dir**=1;end**

F4**:begin** nfloor**=**in3**?**F3**:**in2**?**F2**:**in1**?**F1**:**inG**?**G**:**F4**;**

**if(!(**inG**||**in1**||**in2**||**in3**))** dir**=1;end**

F3**:begin** nfloor**=**in2**?**F2**:**in1**?**F1**:**inG**?**G**:**F3**;**

**if(!(**inG**||**in1**||**in2**))** dir**=1;end**

F2**:begin** nfloor**=**in1**?**F1**:**inG**?**G**:**F2**;**

**if(!(**inG**||**in1**))** dir**=1;end**

F1**:begin** nfloor**=**inG**?**G**:**F1**;**

**if(!**inG**)** dir**=1;end**

G**:begin** nfloor**=**G**;**

**if(!**inG**)**dir**=1;end**

**default:**nfloor**=**G**;**

**endcase**

**end**

**end**

**end**

//counter

**always@(posedge** clk**)**

**begin**

**if(**nstate**==**UP**)**

**begin**

count**<=**count**+1;**

**if(**count**==(**nfloor**-**pfloor**)\*1)**

**begin**

count**<=0;**reached**=1;**

**end**

**end**

**else** **if(**nstate**==**DOWN**)**

**begin**

count**<=**count**+1;**

**if(**count**==(**pfloor**-**nfloor**)\*1)begin**

count**<=0;**reached**=1;**

**end**

**end**

**else**

**begin**

count**<=0;**

reached**<=0;**

**end**

**end**

//output

**assign**

open**=**pstate**==**OPEN**;**

**assign** close**=**pstate**==**CLOSE**;**

**assign** up**=**pstate**==**UP**;**

**assign** down**=**pstate**==**DOWN**;**

**endmodule**

## Elevator Testbench

**module** elevator\_tb**();**

**reg** clk**,**rst**,**inG**,**in1**,**in2**,**in3**,**in4**,**in5**,**in6**,**in7**,**inopen**,**inclose**;**

**wire** open**,**close**,**up**,**down**;**

**parameter** period**=10;**

**reg[7:0]** ins**;**

elevator DUT**(**clk**,**rst**,**inG**,**in1**,**in2**,**in3**,**in4**,**in5**,**in6**,**in7**,**inopen**,**inclose**,**open**,**close**,**up**,**down**);**

//clock

**initial** **begin**

clk**=1'b1;**

inG**=0;**

in1**=0;**

in2**=0;**

in3**=0;**

in4**=0;**

in5**=0;**

in6**=0;**

in7**=0;**

inopen**=0;**

inclose**=0;**

**end**

**always** **#(**period**/2)** clk**=~**clk**;**

//reset

**initial** **begin**

rst**=1'b1;**

**#**period rst**=~**rst**;**

**#**period rst**=~**rst**;**

**end**

//inputs

**initial** **begin**

//reg temp;

**#30** ins**=**$random**;**

$display**("inputs:%b , ins random: %b",{**inG**,**in1**,**in2**,**in3**,**in4**,**in5**,**in6**,**in7**},**ins**);**

**{**inG**,**in1**,**in2**,**in3**,**in4**,**in5**,**in6**,**in7**}=**ins**;**

**repeat(3)**

intest**;**

**wait(!{**inG**,**in1**,**in2**,**in3**,**in4**,**in5**,**in6**,**in7**})**

**#100** $finish**;**

**end**

//update inputs

**always@(posedge** open**)begin**

**case(**DUT**.**nfloor**)**

DUT**.**G**:**inG**=0;**

DUT**.**F1**:**in1**=0;**

DUT**.**F2**:**in2**=0;**

DUT**.**F3**:**in3**=0;**

DUT**.**F4**:**in4**=0;**

DUT**.**F5**:**in5**=0;**

DUT**.**F6**:**in6**=0;**

DUT**.**F7**:**in7**=0;**

**endcase**

**end**

//record

**initial** **begin**

$monitor**("pfloor:%d , inputs:%b , ins random: %b",**DUT**.**pfloor**,{**inG**,**in1**,**in2**,**in3**,**in4**,**in5**,**in6**,**in7**},**ins**);**

$recordfile**("./elevator.trn");**

$recordvars**("depth=0");**

**end**

**task** intest**;** **begin**

**wait(**open**)**

**wait(**close**)**

ins**=**$random**;**

ins**=(**ins**|{**inG**,**in1**,**in2**,**in3**,**in4**,**in5**,**in6**,**in7**});**

**{**inG**,**in1**,**in2**,**in3**,**in4**,**in5**,**in6**,**in7**}=**ins**;**

$display**("ins after:%b",**ins**);**

**end** **endtask**

**endmodule**

## Elevator Controller Waveform

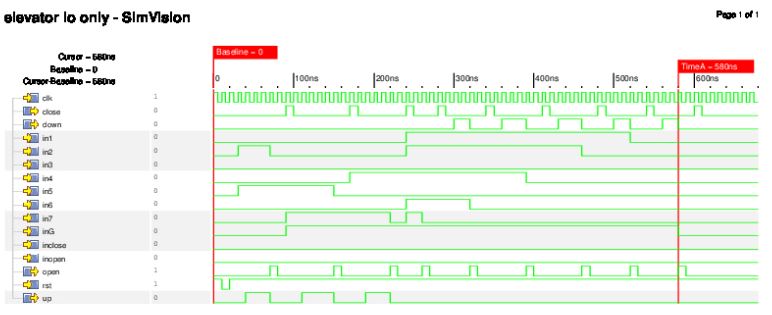


Figure 5.4 Elevator controller input/output waveform

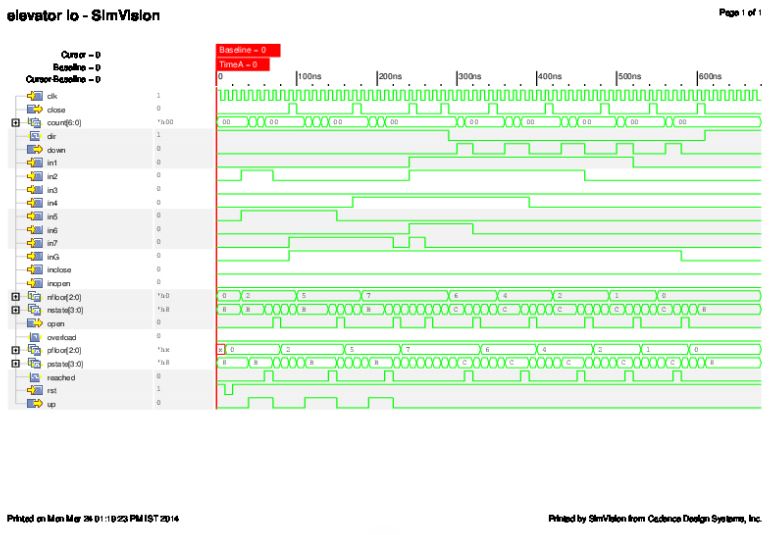


Figure 5.4 Elevator Controller with intermediate waveform

# CONCLUSIONS

The report explains the design of an Elevator Controller proposed using Verilog HDL and is verified by using SimVision tool. The tool helps to identify whether the design works properly or not. Verilog HDL was given primary importance in designing this project due to its vast range of advantages compared to traditional schematic-based design. Functional verification can be done early by optimizing to meet the desired functionality. The language is analogous to computer programming languages which also includes textual description with comments. Verilog HDL is rich in libraries provided by fabrication vendors for post logic simulation with powerful programming language interface. As day to day the technology is gradually improving. So obviously the designs have to be made simpler for enjoying the benefits. To do that, an Elevator Controller is modeled. In the proposed design a Verilog, RTL code is developed to control the lift moment based on the request it will get. The RTL is verified and implemented. In this work, the real-time three-lift controller will be modeled with Verilog HDL code using Finite-State machine (FSM) model to achieve the logic in optimized way. According to the FSM technology the elevator process can be defined with the help of different states. In the FSM technology there is a change from one state to another state likewise in the elevator there will be a change from one floor to another. Every possible way is assigned a path and the implemented based on FSM concept to write the program code for elevator controller. The whole program is designed in such a way that there are desirable switches in each floor and also inside the elevator to control the user commands.

A key challenge facing current and future computer designers is to reverse the trend by removing layer after layer of complexity, opting instead for clean, robust, and easily certifiable designs, while continuing to try to devise novel methods for gaining performance and ease-of-use benefits from simpler circuits that can be readily adapted to application requirements.

Finally the working of the elevator controller is been verified, tested and the results have been plotted.

# REFERENCES

1. Zhou, *PLC electronic control and configuration design*, Science Press, Beijing, 2003.
2. ZHAO Yuhang, MA Muyan, "Implementation of six-layer automatic elevator controller based on FPGA", Wireless Communications Networking and Mobile Computing (WiCOM), 2010.
3. Samir Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis*, Prentice Hall Professional, 2003.
4. Thomas D.E and P.R Moorby, *The Verilog Hardware Description Language*, 4th edition, Kluwer Academic Publications.