MODEL CHECKING OF SIMULINK MODELS

A PROJECT REPORT

submitted by

JICSAH SAJU

IDK15CSCG10

to

the APJ Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the Degree

of

Master of Technology

In

Computer Science and Systems Engineering

Department of Computer Science and Engineering

Government Engineering College

Idukki

MAY 2017

DECLARATION

I undersigned hereby declare that the project report "Model Checking Of Simulink Models", submitted for partial fulfillment of the requirements for the award of degree of Master of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by me under supervision of Dr. Sumesh Divakaran and Prof. Sujit Kumar Chakrabarti. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and / or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Idukki

12-05-2017 Jicsah Saju

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GOVERNMENT ENGINEERING COLLEGE,IDUKKI

CERTIFICATE

This is to certify that the report entitled **'Model Checking Of Simulink Models'** submitted by **'Jicsah Saju'** to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Master of Technology in Computer Science and Engineering with specialization in Computer Science and Systems Engineering is a bonafide record of the project work carried out by her under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Internal Supervisor External Supervisor

PG Coordinator HEAD OF THE DEPT

ACKNOWLEDGEMENT

First and foremost I praise and thank GOD, the foundation of all wisdom from the depth of my heart for being the unfailing source of strength. Beyond good, there are dozens of individuals, who have helped me. I would like to add a few heartfelt that to these people who were a part of my report in numerous ways.

Next, I thank ***Dr. Vijayan P****, Principal and* ***Dr. Sumesh Divakaran****, Head of the Department of Computer Science and Engineering at Government Engineering College, Idukki, for providing new dimensions to my Engineering course.*

I am deeply indebted to Internal Supervisor ***Dr. Sumesh Divakaran****, for his careful attention and support to my work. I would like to express my deepest appreciation to my advisor for helping me to overcome several constraints.*

I deeply indebted to External Supervisor ***Prof. Sujit Kumar Chakrabarti****, Assistant Professor at IIIT Bangalore, for his continuous support and inspiration.*

I thank to PG coordinator ***Dr. Madhu K P****, Assistant Professor, for his support in my work.*

I also thank Project Coordinator ***Mr. Philumon Joseph****, Assistant Professor, for his support and supervision on the success of my work.*

I expand my special thanks to all my friends for their enthusiastic encouragement and full support. More than anybody else, I am grateful to my parents for their encouragement, support, and blessing.

ABSTRACT

Simulink is a graphical programming environment developed for modeling, simulating and analyzing multi-disk dynamic systems. Designers refer only to the graphical output for both the verification and validation (or simulation) of Simulink models. Model checking automatically verifies the correctness rules or properties of finite-state systems. Model checking methods have been developed for checking models for both hardware as well as software designs, such that the specification can be given in temporal logic formulas such as CTL and LTL formula. The basis of the work is to develop Simulink models and translating the models into a model checking tool, named NuSMV. Model Checker proves the properties of the models to check whether the system meets its required specifications.

Keywords

NuSMV, Temporal Logic, Model Checker

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

NuSMV	New Symbolic Model Verifier
LTL	Linear Temporal Logic
CTL	Computational Tree Logic
SPIN	Simple Promela Interpreter
G	Globally
F	Future
X	Next
U	Untill
A	All
SMV	Symbolic Model Verifier
BDD	Binary Decision Diagram
OBDD	Ordered Binary Decision Diagram
FSM	Finite State Machine
SLDV	Simulink Design Verifier
HTML	Hypertext Markup Language
MPS	Multiport Switch
EMC	Extended Model Checker
CNF	Conjunctive Normal Form
DNF	Destructive Normal Form

CHAPTER 1
INTRODUCTION

1.1 GENERAL BACKGROUND

Simulink is a tool developed by MathWorks. It is a block diagram environment widely used to solve real-world applications in many industries. It is used for analyzing multi-domain simulation and for Model-Based Design of dynamic systems and embedded systems. It can also be used to design, simulate, implement and test communications, controls, signal processing and other time-varying systems. It is customized with block libraries and provides solvers for modeling and simulating dynamic systems. Also provides the simulation environment for analysis, refinement, and validation. Designing a system in Simulink model uses a set of blocks that may be interconnected by lines and these lines are used to represent the connections of block inputs to block outputs [5]. Each block represents the dynamic system that produces an output either continuously by using continuous blocks or at specific points in time by using discrete blocks. Interconnected blocks are used to build sub-systems which in turn are put together to form a system model.

Simulink is integrated within MATLAB. To construct Simulink models, first launch library browser of Simulink from MATLAB section. Simulink contains a large set of libraries (having several blocks), that are used for creating models. Creating a new model can be performed by simply dragging the blocks that is needed to be designed the model into the model window. After adding blocks, connect them by using lines. Every block has angle brackets, by using this angle brackets, connect the blocks from input to output. Simulink blocks have attributes, which can be set according to the model.

Renaming a block is possible by clicking the block label and edit it. It is possible to label signals by double-clicking the signal line and enter the text. To use MATLAB variables as Simulink block parameters, variables must be defined in MATLAB work-space. Save the model with the · slx file. Model configuration parameters have several factors that affect the simulation. Description of each block and its parameters are provided within the blocks itself.

Some specific features of Simulink are: Code reuse is not exactly as in normal programming languages.

1.1.1 Property Proving in Simulink

A property is a requirement for the model to check whether the given specification satisfies or not. Property proving analysis needs to specify a functional requirement or a property that the model wants to prove. Simulink Design Verifier(SLDV)[6] uses formal methods for identifying hidden design errors such as dead logic, integers or fixed point data overflow, assertion violation and division by zero errors in models without extensive simulation runs. If any search completes without finding any path that violates the required specification, then the property is proven. Otherwise, if it finds any path that violates, then the output is falsified with some counter-examples.

Steps for proving properties in a model are following:[5]

1. Construct the Simulink Model: use blocks in Simulink library browser and connect them according to our design
2. Verify that our model is Compatible with SLDV: Compatibility checks result shows that the designed model is Compatible or Incompatible with the software

3. Add Proof Objective block to the model to prepare for its proof

4. Configure SLDV to prove properties

5. Prove properties to the designed model

6. Review analysis result: When the analysis completes, the log window displays the following options for reviewing the results:
   (a) Highlight the analysis results on the model
   (b) Generate a detailed HTML analysis report
   (c) Create a harness model

7. Add Proof Assumptions to specify analysis constraints

8. Prove a property of the customized model and interpret the results

1.1.2 Formal Methods

Formal methods are system design techniques comprises formal specification to specify the desired properties of the system using mathematical models to build both software as well as hardware systems. The syntax and semantics of formal specification language are formally defined. The syntax is used for specification representation; semantics, uses objects to describes the system; and a set of relations, uses rules to indicate the objects for satisfying the specification. Formal methods use mathematical proof to ensure correct behavior of the system by system testing.

The formal method contains two approaches, property based and model-based. The property specification describes the operations that are performed on the system. The model-based specification comprises the tools of set theory, function theory, and logic to develop an abstract model of the system. It also specifies the operations on an abstract

model. It describes, set of states of the system and the legal operations performed on the system. This indicates, how these legal operations change the current state.

By the use of formal verification schemes, formal methods differ from other design systems. Traditional system design uses extensive testing for verification, but testing is capable only for finite conclusions. Dijkstra and others have demonstrated such that the test cases can only show the situations where a system won't fail, but not able to say anything about the behavior of the system outside of the testing scenarios. In contrast, once a theorem is proven as true, it remains true.

Formal verification does not solve the needs for testing. Formal verification cannot fix bad assumptions in any designs, but it helps to identify errors in reasoning. While formal systems are attractive in theoretical basis, but practical implementations are wanting. It provides a compromise between the necessities of engineering and the goals of the formal design, it also provides a good compromise and is the most important route for future research.

1.1.3 Temporal Logic and Model Checking

Given a model of a system, it automatically checks whether this model meets a given specification, these refer to model checking. In most cases, hardware or software systems, whereas the specification may contain safety requirements such as the absence of deadlocks and similar critical states that can cause the system to crash. Model checking is a technique used for automatic verification of correctness properties of finite-state systems. Model checking is most commonly applied to hardware designs. And because of undecidability, the approach cannot be fully algorithmic for software's; usually, it may fail to prove or disprove a given property.

Instead of equivalence checking, property checking is used for verification when two descriptions are not functionally equivalent. During refinement, the specification is complemented because this is unnecessary in the higher level specification. It is not even possible to verify the newly introduced properties against the original specification. So, the strict bi-directional equivalence check is relaxed to one-way property checking. The implementation or design of a model is related to the circuit whereas the specifications are properties.

The structure is typically given as a source code description in an industrial hardware description language or a special-purpose language. Such a program corresponds to a finite state machine (FSM), is defined as a directed graph consisting of nodes (or vertices) and edges. A set of atomic propositions is associated with each node, stating which memory elements are one. States of a system are represented as nodes, the edges which represent possible transitions which may alter the state, and the atomic propositions represent for basic properties that hold at a point of execution.

Model checking methods have been developed for checking the models of both hardware as well as software designs where the specification is given by a temporal logic formula. For instance, whenever a request is made, access to a resource is eventually granted, but it may never granted to two requests simultaneously. Such a statement can be expressed in a temporal logic. Temporal logic contains two kinds of operators: logical operators and modal operators. Logical operators are usual truth-functional operators ($\neg, \vee, \wedge, \rightarrow$). The modal operators are used in Linear Temporal Logic and Computation Tree Logic are defined as follows:

1. Next (N ϕ): ϕ has to hold at the next state. (X is used synonymously)

2. Future (F ϕ): ϕ eventually has to hold (somewhere on the subsequent path)

3. Globally (G ϕ): ϕ has to hold on the entire subsequent path

4. Until ($\phi \bigcup \psi$): ϕ has to hold untilψ holds at the current or a future position. At that position ϕ does not have to hold any more

5. All (A ϕ): ϕ has to hold on all paths starting from the current state

6. Exists (E ϕ): there exists at least one path starting from the current state where ϕ holds

Different Model Checking tools are available, they are defined as follows:

1. **NuSMV**: [10]An extension of SMV, symbolic Model Verifier. It supports the analysis of specifications expressed in both CTL and LTL. NuSMV, a joint project between ITC-IRST (Istituto Trentino di Cultura in Trento, Italy).

2. **SPIN**: [9]Simple Promela Interpreter, a tool for verifying the correctness of distributed software models in a rigorous and an automated fashion. It was written by Gerard J. Holzmann and others in the original Unix group of the Computing Sciences Research Center at Bell Labs in 1980.

1.1.4 NuSMV

NuSMV is a reimplementation and extension of SMV symbolic model checker [4]. It is known as the first model checking tool based on Binary Decision Diagrams (BDDs). The tool is an open architecture for model checking. It may focus on reliable verification of industrially sized designs. It can be used as verification as well as research tools for formal verification methods. NuSMV2 is the version 2 of NuSMV, receives all the functionalities of NuSMV. It is combined with BDD-based model checking with

SAT-based model checking[4]. NuSMV supports the specifications expressed in both Linear Temporal Logic(LTL) as well as Computational Tree Logic(CTL). NuSMV can be broken down into modules and these modules can be reused conveniently.

1. To bring the interactive mode of NuSMV, use the following command:
   system_prompt> NuSMV -int <RET>
   NuSMV> go

2. NuSMV tries to read and execute commands from an initialization file if such a file exists and is readable. If [7] no such file exists, usually user's home directory and current directory will be checked. Commands in the initialization file are executed consecutively. When initialization phase is completed the NuSMV shell is displayed and the system is now ready to execute user commands. A NuSMV command usually consists of a command name and arguments to the invoked command. NuSMV read and execute commands as:
   NuSMV> read_model -i name of model

3. For getting the initial states:
   NuSMV>pick_sate -i

4. For Simulating:(transitions says that simulate the transition for n different steps)
   NuSMV>simulate -i -k n

5. For printing reachable states(says number of states and description of states)
   NuSMV> print_reachable_states -v

6. Checking for LTL specification or CTL specification:
   NuSMV>check_ltlspec -p "G(specification)"
   OR
   NuSMV>check_ctlspec -p "F(specification)"

1.2 OBJECTIVE

To demonstrate static verification of Siumulink models using NuSMV. Simulink models are easy to build in C or C++. It can also convert to XML format. In this project, the Simulink model is converted to a model checking tool called NuSMV. The NuSMV conversion is easy to represent by converting each Simulink blocks separately as different modules and finally combine all the modules together in a single file.

1.3 SCOPE

The property proving in Simulink design verifier is not much efficient. So proving the properties of a Simulink model using model checking. The NuSMV is the model checking tool used for proving these properties.

1.4 SCHEME OF PROJECT WORK

Development of an industrial strength case study in Simulink, Simulink is a tool in Mathworks. A model is created with the help of the blocks in Simulink library. Represent the created model into a model checking tool called NuSMV(New Symbolic Model Verifier). Prove different properties of the model using different model checking methods ie, LTL(Linear Temporal Logic) and CTL(Computational Tree Logic).

1.5 REPORT ORGANIZATION

The project is organized into seven chapters including the introduction. The second chapter gives the literature review made as the key findings and support to the work, while technical perspectives are related to the Simulink and model checking. The Elevator model in Simulink for converting it into a model checking tool is given in the third chapter. Different test cases of the elevator model in Simulink are provided in the fourth chapter. Translation of Simulink model to NuSMV model checking tool are described in the fifth chapter. The model checking outputs of the Elevator model are presented in the sixth chapter. Finally, the conclusion with future research is discussed in chapter seven.

CHAPTER 2
LITERATURE REVIEW

Meenakshi B et al. [1] present a tool that automatically translates Simulink models into a suitable model checker, named NuSMV. NuSMV is an open source verification tool and supports fully an automatic technique. Also, it is well-known to handle systems with large state space size. Symbolic model checker, NuSMV is based on Binary Decision Diagrams(BDDs). Specifications can be given either in CTL as well as LTL formulas. Model checking algorithms in NuSMV are used for checking whether the system meets its specification for verifying hardware designs. This paper mainly focused on translating the Simulink model of an avionics triplex sensor voter. The voter takes input from three sensors and produces a single genuine sensor output. NuSMV input language is designed for the specification of system models as finite state machines. According to get the complete system description, modules can be composed either synchronously or asynchronously. The translator algorithm of the Simulink model takes the MDL file format as NuSMV code. Translation retains the structure i.e, the hierarchy of the blocks, their names and interconnections of the input Simulink model. While running the translation algorithm, the system engineer has the option of bounding the ranges of all variables that occur in the model. If no ranges are specified, the translator assumes maximum range. Simulink models have both discrete and continuous blocks. But the scope of the discrete blocks of Simulink are restricted. One sample time in the Simulink model is equivalent to one execution step in the NuSMV. In order to debug the model, a reverse translation routine that takes NuSMV as input and translates it back into a textual notation that a Simulink designer can understand.

The research work by **Clarke et al.** [2] says that the Concurrent Program Verification was an important problem that needed to be solved. Finding concurrency errors are not easy to detect by program testing since they are hard to reproduce. This paper says that the Model checking is used for checking whether the structure M was a model for the formula f, such that M is the Kripke structure and f is the temporal formula. Petri Net Tools, Bochmann and Protocol Verification, Holzmann and Protocol Verification are some Verification Tools Before 1981. The EMC was the first Model Checker to implement fairness constraints. Fairness Constraints are formulas that must hold infinitely often on each fair path. Extended Model Checker algorithm was able to solve the Emptiness Problem for Non-deterministic Buchi Automata in time linear in the size of the automaton. EMC did not give counterexamples for universal CTL properties. In symbolic model checking, transition relations were represented explicitly by adjacency lists. The new symbolic representation was based on ordered binary decision diagrams. OBDDs provide a canonical form for boolean formulas that is more compactable than CNF or DNF. The Model Checker extracts a transition system represented as an OBDD from a program in the SMV language and uses an OBDD-based search algorithm to determine whether the system satisfies its specification. If the transition system does not meet its requirement, the verifier will create an execution trace that shows why the requirement is falsified. Using SMV it is easy to find several previously undetected errors and potential errors in the design. Verifying software may cause some problems for Model Checking because software tends to be less structured than hardware. Concurrent software is usually asynchronous, so the state explosion phenomenon is a serious problem. Model Checking has been used for hardware verification than software verification. The most successful method for dealing with asynchronous systems are based on the partial order reduction. Some basic techniques needed when symbolic methods and the partial order reduction don't work are Compositional Reasoning, Abstraction,

Symmetry Reduction, Induction and Parameterized Verification.

Kenneth et.al [3] use different techniques to reduce the state explosion problem by using a method called symbolic model checking(SMV). SMV can automatically verify programs with respect to temporal logic formulas with symbolic model checking technique. Simulation is the best tool for finding errors before the invention. Engineers using simulation are faced with two problems. The first is creating a set of input patterns that are enough to expose any incorrect way of the system. Second is to discover the correct output of the system. A formal verification framework has three basic components, a mathematical model of the system to be verified, a formal language to estimate the correctness problem and a procedure for proving the statement of correctness. The practical application of temporal logic in hardware verification is known as model checking. The symbolic model checking method is developed for state space search using OBDD. SMV system allows the automatic verification of programs that are specialized in a language for the finite state system. Induction method is applied for different process models with simple algebraic properties. Verification using occurrence nets is another technique used to prove an asynchronous distributed mutual exclusion circuit is hazard free.

CHAPTER 3
MODELLING IN SIMULINK

3.1 MODEL EXAMPLE: THE ELEVATOR MODEL

A simple elevator model with some basic properties. The model has a controller and plant, such that the inputs to the controller are coming from the plant and the output of the controller are given to the plant. Knowing the simple logic of elevator is easy to build the Simulink model of the elevator. Here, this elevator model is built for four floors. At a time the elevator can be in one of the floors. While moving it does not consider that elevator is in any of the floors. 3.1 says that the request is coming from the plant model and the controller has got the request and accordingly it works. Finally, the controller gives the output such that it will decide to move upwards or downwards or it will be in a stop mode. The output can see with scope block in Simulink for 150 as simulation stop time.

The plant model 3.2 contains three subsystems. Calculating position subsystem 3.3 contains controller output, based on this it will calculate the current floor where the elevator is. The positionId 3.4 says each floor of the elevator. 3.5 says the way to find the current position of the floor where the elevator is. LiveRequest 3.10 subsystem contains the live request to which the elevator should move next.

On receiving out = UP, the model starts incrementing the position, on receiving out = DOWN, the position is decremented. Depending on the value of the position one of the five P i lines is made high. Request(RI)are randomly generated to simulate passenger arrivals. Passenger arrival rates are external to the system.

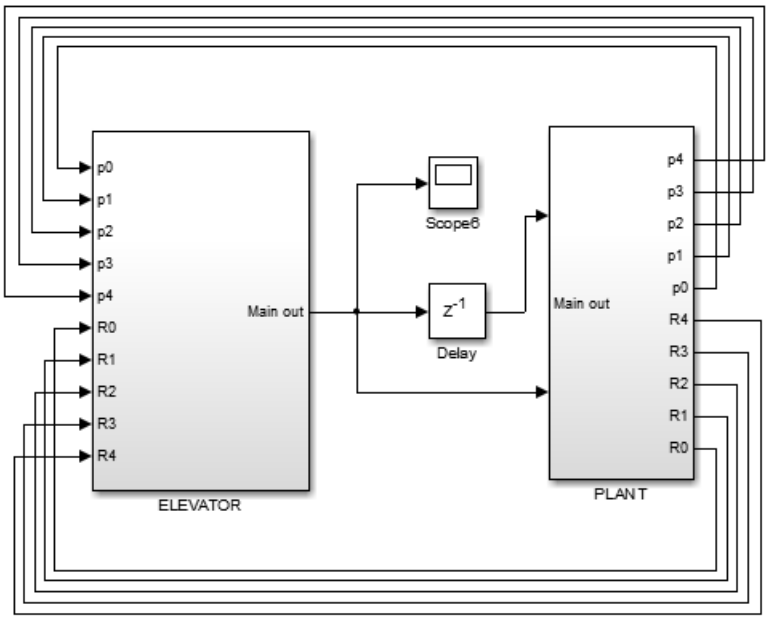

Figure 3.1: The elevator model

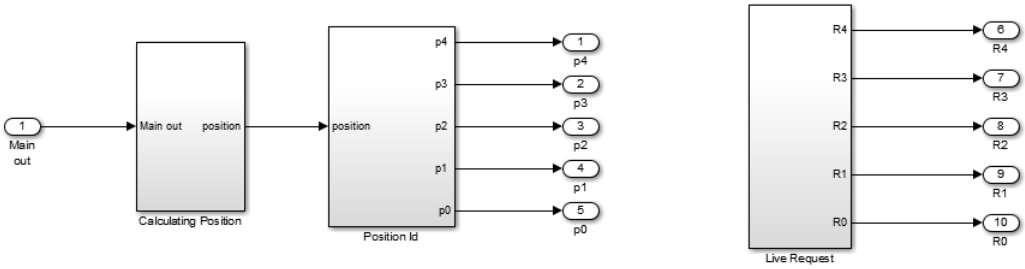

Figure 3.2: The plant model of elevator

The position of the elevator is calculated from controller output that is compared(relational operator block) with the direction to which it moves 3.3. The output is then multiplied by the integer either 1 or -1, then finally capturing the current position value with 'sum' block in Simulink.

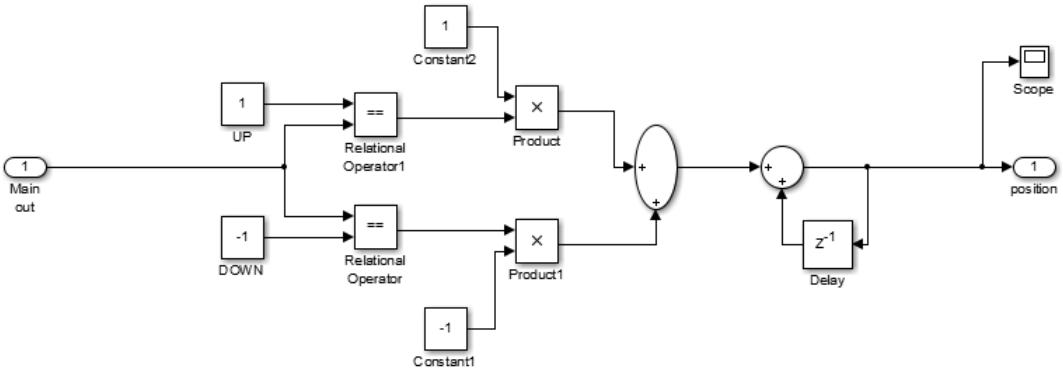

Figure 3.3: How calculating position of elevator model

PositionId subsystem will compare the position value with each floor(Pi)value, so that plant model tells the Pi value. This Pi value is then finally given to the controller as inputs 3.4.

The PositionId subsystem contains the five different subsystems to find the P i value by comparing the position with 10i, where i = 0,1,2,3 and 4 in 3.4. Constant value 10 indicates that it will take 10 second time for the movement of the elevator from one floor to the next floor.

Each subsystem in 3.4 is shown below figures such as 3.5 (Subsystem for P 4, comparing fourth floor with 40) , 3.6 (Subsystem1 for P 3 , comparing third floor with 30) , 3.7 (Subsystem2 for P 2 , comparing second floor with 20) , 3.8 (Subsystem3 for P 1, comparing first floor with 10), 3.9 (Subsystem4 for P 0 , comparing ground floor with 0) .

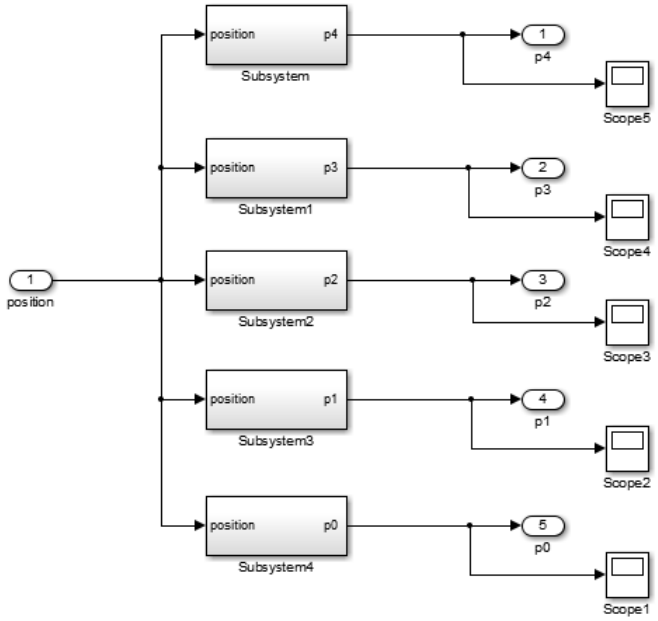

Figure 3.4: The positionid of elevator model

3.5 comparing the constant value of fourth floor value with the position value, if they are equal then it should give PositionId as the p4 = TRUE.

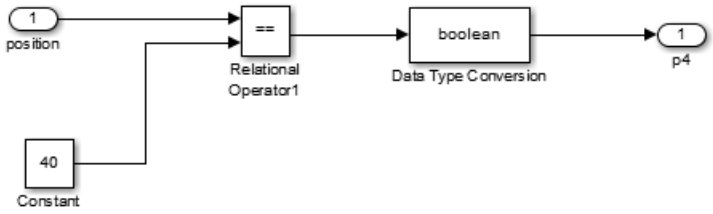

Figure 3.5: The subsystem model of positionid

3.6 comparing the constant value of third floor value with the position value, if they are equal then it should give PositionId as the p3 = TRUE.

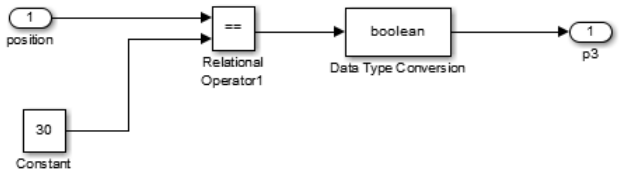

Figure 3.6: The subsystem1 model of positionid

3.7 comparing the constant value of second floor value with the position value, if they are equal then it should give PositionId as the p2 = TRUE.

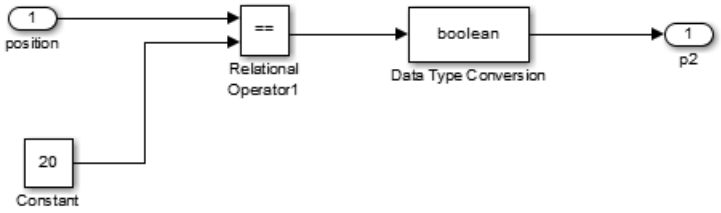

Figure 3.7: The subsystem2 model of positionid

3.8 comparing the constant value of first floor value with the position value, if they are equal then it should give PositionId as the p1 = TRUE.

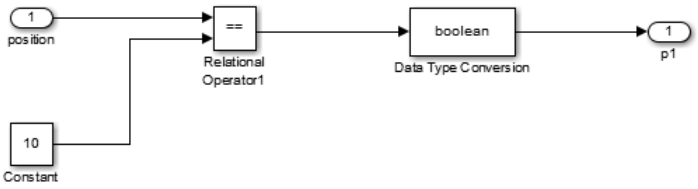

Figure 3.8: The subsystem3 model of positionid

3.9 comparing the constant value of ground floor value with the position value, if they are equal then it should give PositionId as the p0 = TRUE.

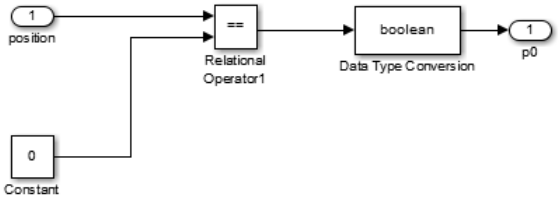

Figure 3.9: The subsystem4 model of positionid

The Live Request subsystem contains the subsystem of different test cases. Each test case is the origin of the request 3.10. The request origin is from the plant model of the elevator and this is given to the controller as input. Each test cases are explained in the next chapter.

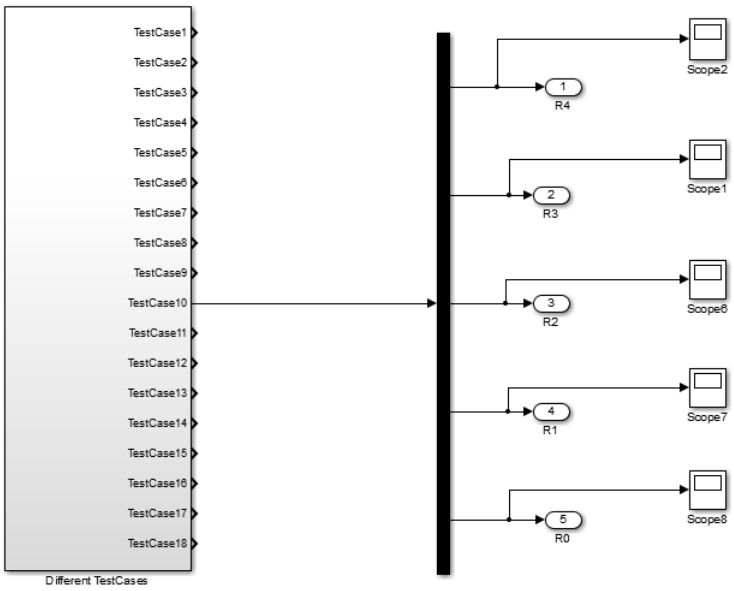

Figure 3.10: The live request from elevator model

The 3.11 contains different test cases such that how different requests has been generated and how it is served as per the given requests.

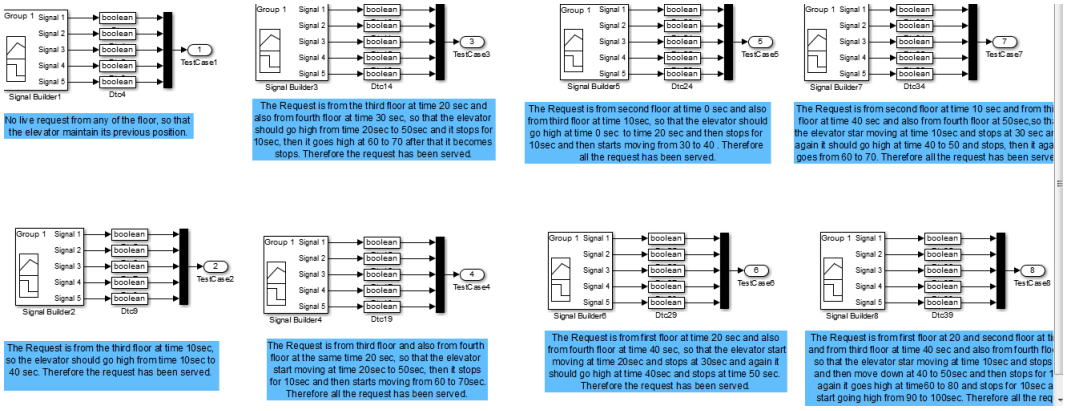

Figure 3.11: The different testcases for elevator model

The controller of Elevator model checks if the system arrives in any of the floors if so the timer goes high at some point in time. Then it will go and check the direction of elevator model to make the next movement. The direction subsystem will decide the next movement(UP, DOWN or STOP) of the elevator 3.12.

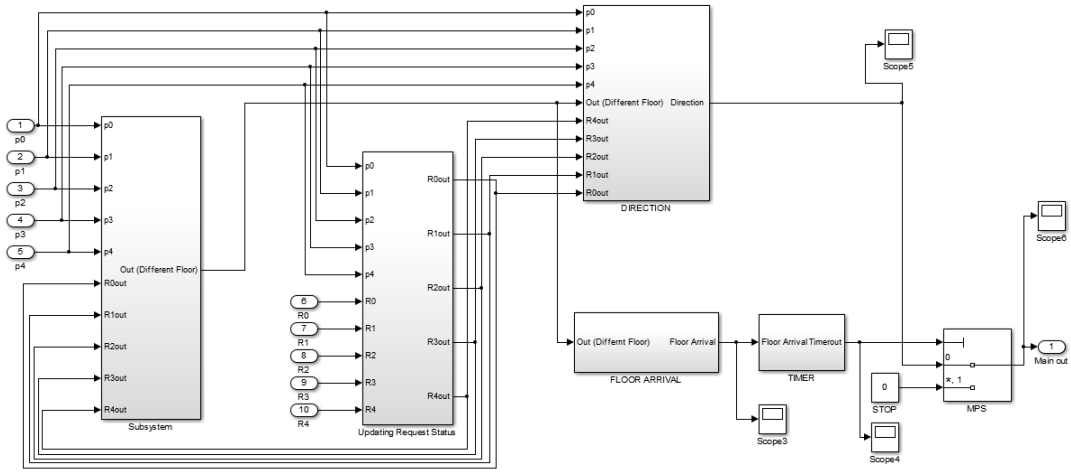

Figure 3.12: The controller of elevator model

The subsystem of controller contains the Live Request and Pi values from the plant. The delay block will delay for one-time step and that delayed output is combined(AND) with pi values respectively. Finally taking the OR of each 3.20 , according to the floor values. The output of the subsystem is given to the input for direction subsystem in controller 3.13. This subsystem block is to find whether there is any live request from any of one of the floor 3.14 .

Updating Request Status finds the live request from the request and pi values. It contains five different subsystems for each of the floor and requests from the plant. So first it will AND the NOT of each pi values with the request from plant if any. If so, the Multiport switch takes the TRUE value and again AND the output of MPS with NOT of each pi values. Finally, this subsystem should say whether if there is any live request or not.

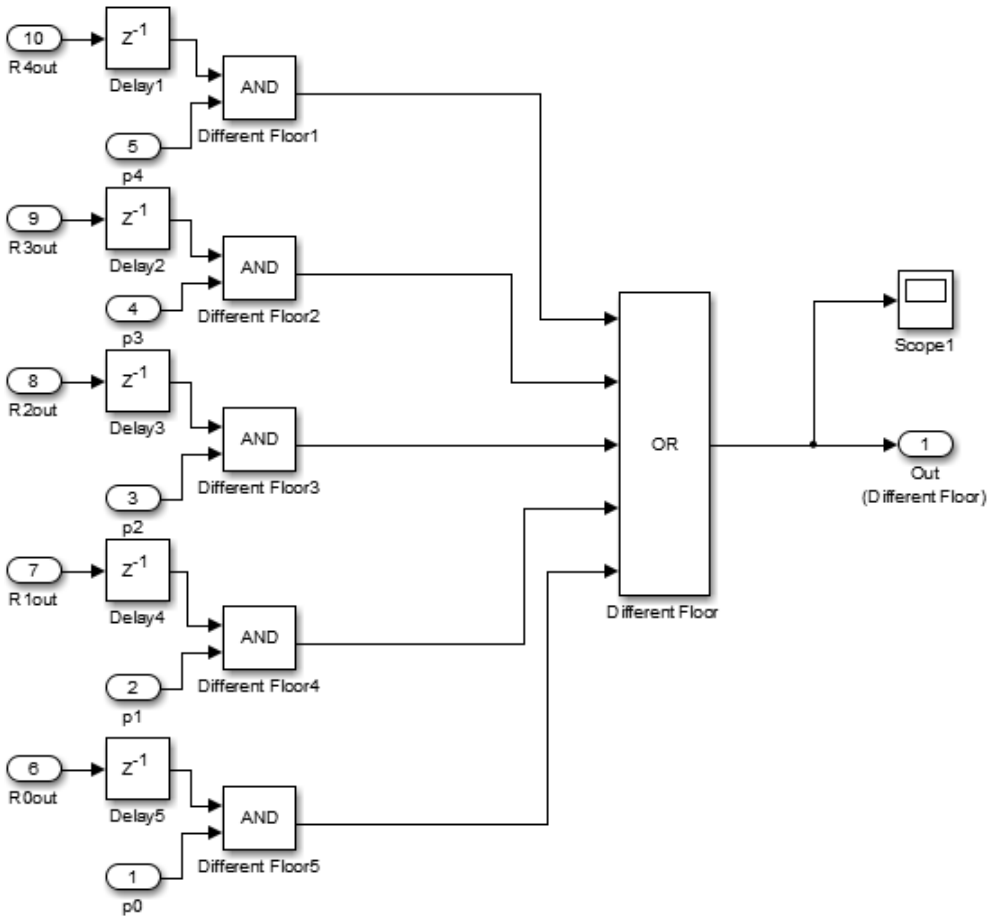

Figure 3.13: The subsystem model from controller

Each subsystem in 3.14 is shown below figures such as 3.15 (Subsystem for checking whether their is any live request from fourth floor), 3.16 (Subsystem1 for checking whether there is any live request from third floor), 3.17 (Subsystem2 for checking whether their is any live request from second floor), 3.18 (Subsystem3 for checking whether their is any live request from first floor), 3.19 (Subsystem4 for checking whether there is any live request from ground floor).

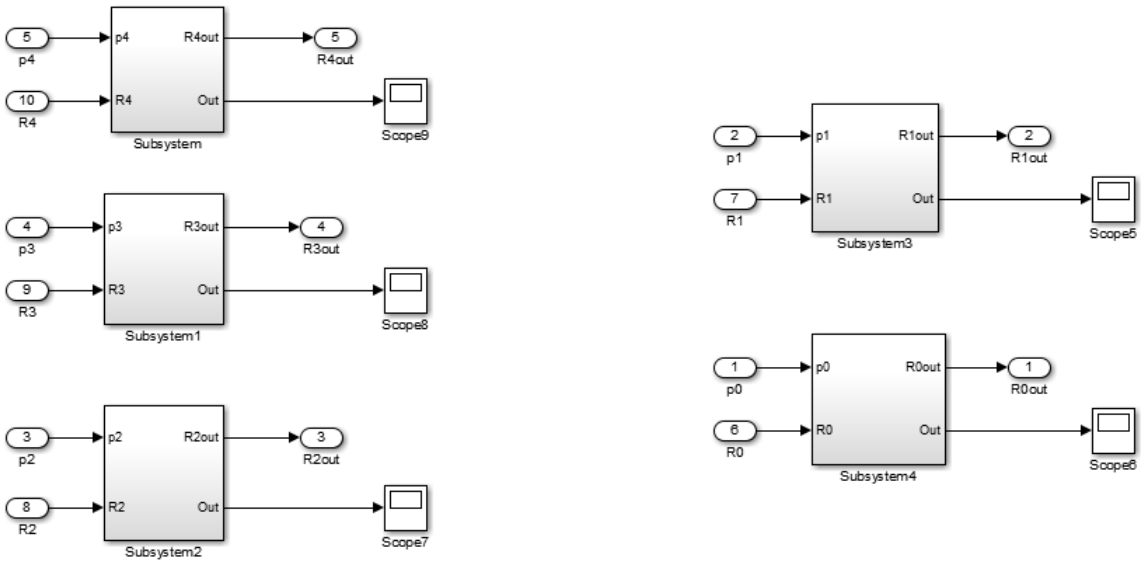

Figure 3.14: Updating request status subsystem from controller

3.15 will AND the NOT of p4 value with the request from plant. Next the Multiport switch takes the TRUE value based on it's control port. Finally, it will AND the output of MPS with NOT of p4. So it should calculates whether the request is from the fourth floor.

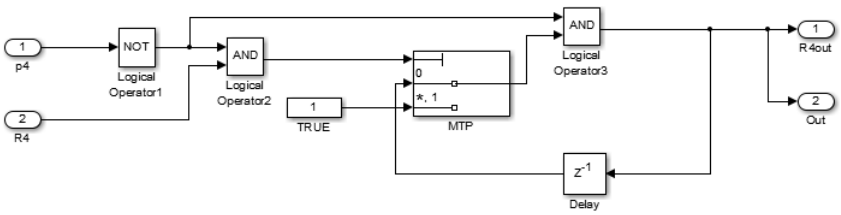

Figure 3.15: Subsystem of updating request status

3.16 will AND the NOT of p3 value with the request from plant. Next the Multiport switch takes the TRUE value based on it's control port. Finally, it will AND the output of MPS with NOT of p3. So it should calculate whether the request from the third floor.

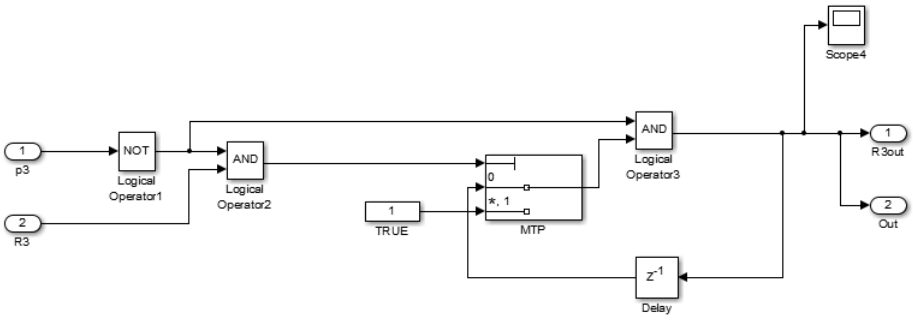

Figure 3.16: Subsystem1 of updating request status

3.17 will AND the NOT of p2 value with the request from plant. Next the Multiport switch takes the TRUE value based on it's control port. Finally, it will AND the output of MPS with NOT of p2.So it should calculate whether the request from the second floor.

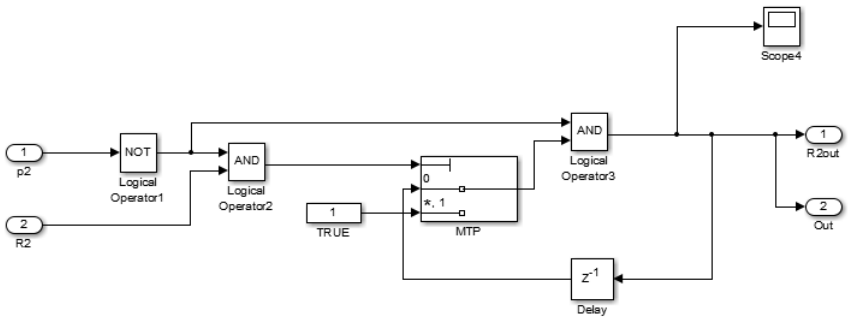

Figure 3.17: Subsystem2 of updating request status

3.18 will AND the NOT of p1 value with the request from plant. Next the Multiport switch takes the TRUE value based on it's control port. Finally, it will AND the output of MPS with NOT of p1. So it should calculate whether the request from the first floor.

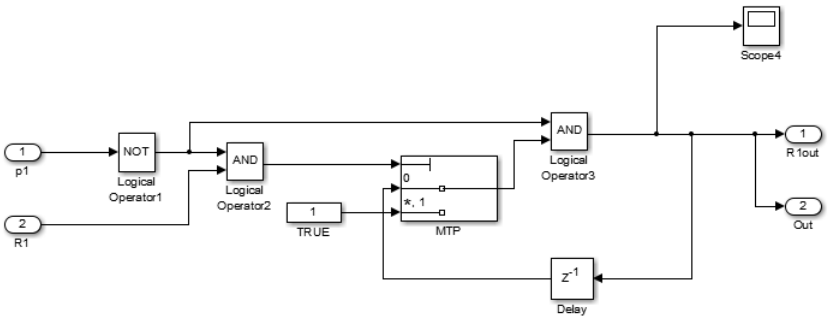

Figure 3.18: Subsystem3 of updating request status

3.19 will AND the NOT of p0 value with the request from plant. Next the Multiport switch takes the TRUE value based on it's control port. Finally, it will AND the output of MPS with NOT of p0. So it should calculate whether the request from ground floor.

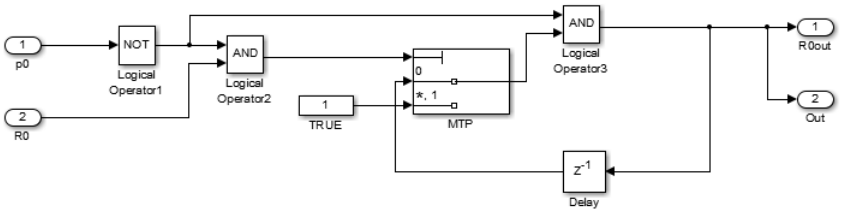

Figure 3.19: Subsystem4 of updating request status

If any live request from plant model and if the elevator is in any of the floors then the floor arrival should give a high pulse as output 3.20 . Here the floor arrival subsystem will check if there any high pulse from the subsystem of the controller then it will AND the arrived input with delay block in Simulink and then NOT with the delayed input. Finally, it will give the high pulse on different time steps.

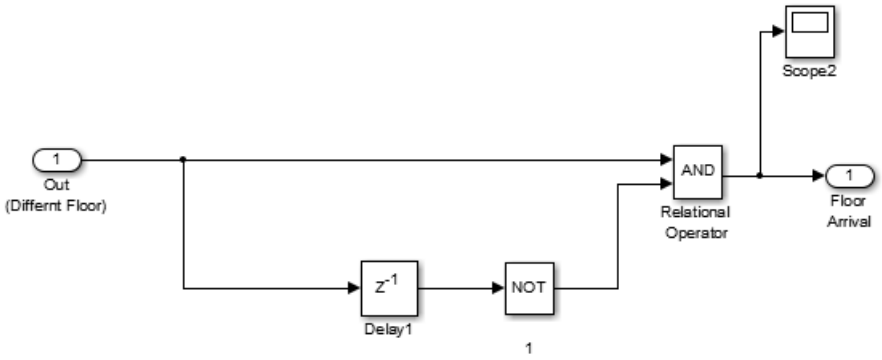

Figure 3.20: The floor arrival

Timer output is true if and only if the value is between the floor arrival time and constant 'n' added to the floor arrival time. Otherwise, the floor arrival value is false. Instead of getting the true value for timer, toggle the timer value as shown in 3.22 .

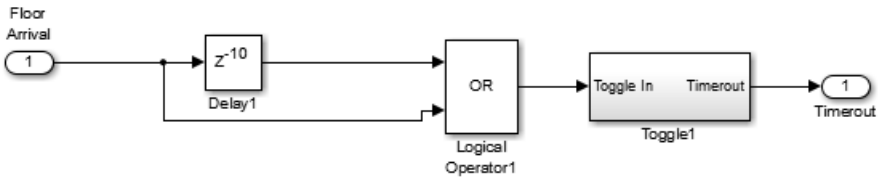

Figure 3.21: The timer

Figure 3.22 says that if the 'ToggleIn' is FALSE, the value will be its previous one, no changes happens. If it is TRUE, then the Timer out will be the NOT of the previous value.

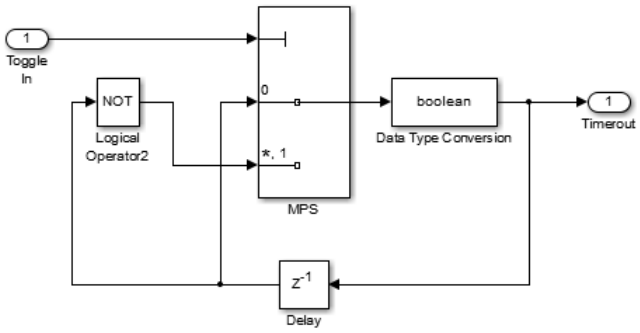

Figure 3.22: Toggle in timer

In Direction Subsystem, 3.23 tells that to which direction the elevator should move next. Based on that it will move either UP, DOWN or STOP. So that, first it calculates whether there is any live request, then calculate the current floor value. After calculating the current floor, current floor value is compared with the live requests. No request is allowed to come from the floor where the elevator is currently. After that, it calculates the floor value to which it moves. Finally, the subsystem tells the floor to which it moves.

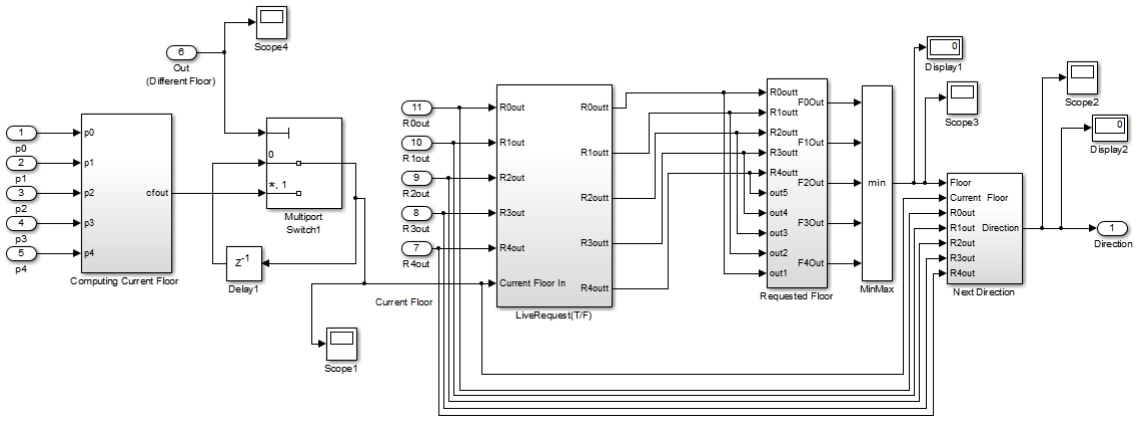

Figure 3.23: The direction

In the direction, the first subsystem 3.24 is for computing the current floor value. So that the elevator says the current floor where the elevator is. The Simulink blocks used for computing current floor are constant, inport, product, and sum. Multiply the inport port with the constant value for each of the floor and taking the sum, for computing the current floor value.

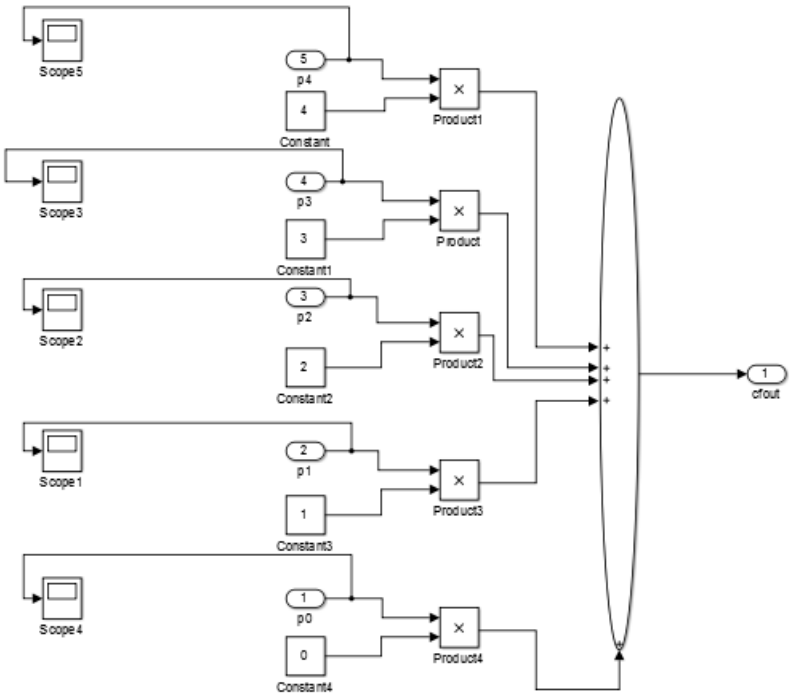

Figure 3.24: The computing current floor value

In LiveRequest(T/F) subsystem, 3.25 will calculate from which floor the live request comes, it will first calculate the absolute value (by absolute value block in Simulink) from the current floor value of each floor. If any live request value of any floor is TRUE then it will take the absolute value otherwise a constant value other than absolute value is taken (with multiport switch block) 3.26 .

3.26 tells each subsystems of 3.25 contains.

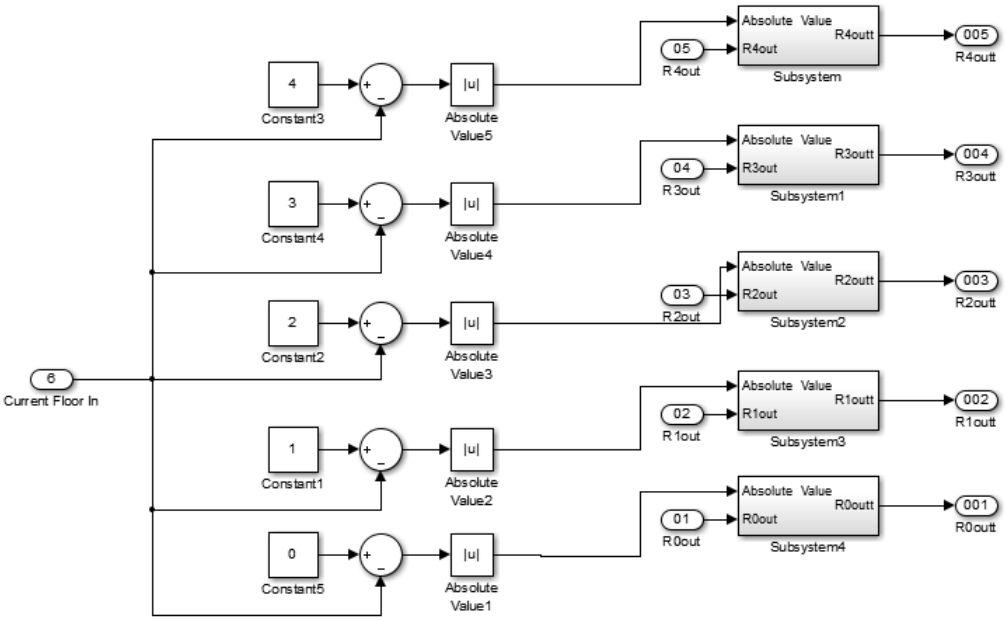

Figure 3.25: The liverequest(T/F)

Each subsystem in 3.25 is shown in below figures such as 3.26 (Subsystem for calculating the request status of fourth floor) , 3.27 (Subsystem1 for calculating the request status of third floor) , 3.28 (Subsystem2 for calculating the request status of second floor) , 3.29 (Subsystem3 for calculating the request status of first floor), 3.30 (Subsystem4 for calculating the request status of ground floor).

3.26 should take the absolute value only if R4out = TRUE. Otherwise the multiport switch gives a constatnt value other than absolute value.

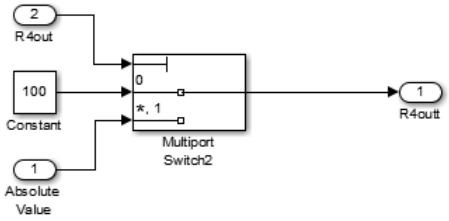

Figure 3.26: The subsystem from liverequest(T/F)

3.27 should take the absolute value only if R3out = TRUE. Otherwise the multiport switch gives a constatnt value other than absolute value.

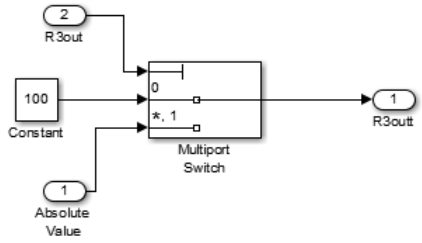

Figure 3.27: The subsystem1 from liverequest(T/F)

3.28 should take the absolute value only if R2out = TRUE. Otherwise the multiport switch gives a constatnt value other than absolute value.

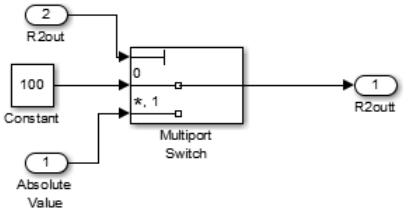

Figure 3.28: The subsystem2 from liverequest(T/F)

3.29 should take the absolute value only if R1out = TRUE. Otherwise the multiport switch gives a constatnt value other than absolute value.

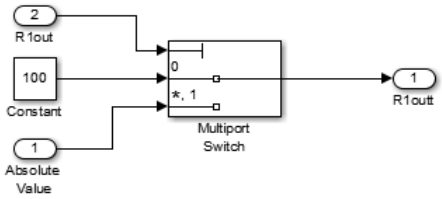

Figure 3.29: The subsystem3 from liverequest(T/F)

3.30 should take the absolute value only if R0out = TRUE. Otherwise the multiport switch gives a constatnt value other than absolute value.

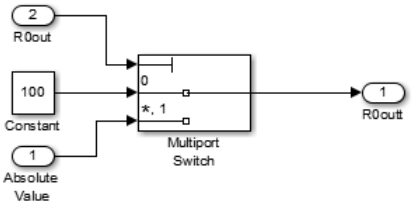

Figure 3.30: The subsystem4 from liverequest(T/F)

By taking the minimum from 3.31 and then comparing the minimum value with the live request. If they are equal then it gives the value of the floor (from 0 to 4) to which it should move, otherwise it will give a value other than floor value.

Each subsystem of 3.31 checks whether the live request and minimum of each value is true or false and finally, it gives the value of the floor with the help of multiport switch block in Simulink 3.32.

Each subsystem in 3.31 is shown below figures such as 3.32 (Subsystem for calculating the fourth floor value), 3.33 (Subsystem1 for calculating the request status of third floor), 3.34 (Subsystem2 for calculating the second floor value), 3.35 (Subsystem3 for

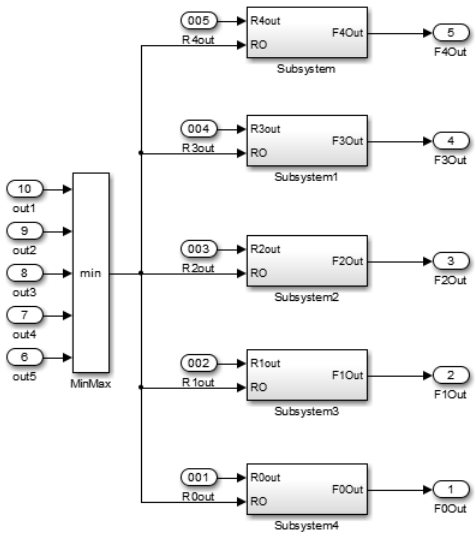

Figure 3.31: The requested floor

calculating the first floor value), 3.36 (Subsystem4 for calculating the ground floor value).

3.32 should give the current floor value(f4) as the direction to which the next movement occurs. Otherwise it gives a constant value other than the floor value.

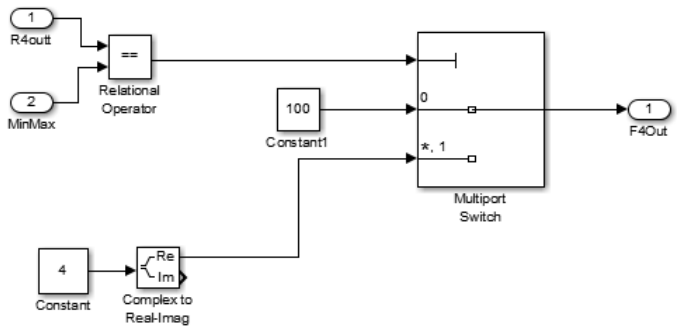

Figure 3.32: The subsystem from requested floor

3.33 should give the current floor value(f3) as the direction to which the next movement occurs. Otherwise it gives a constant value other than the floor value.

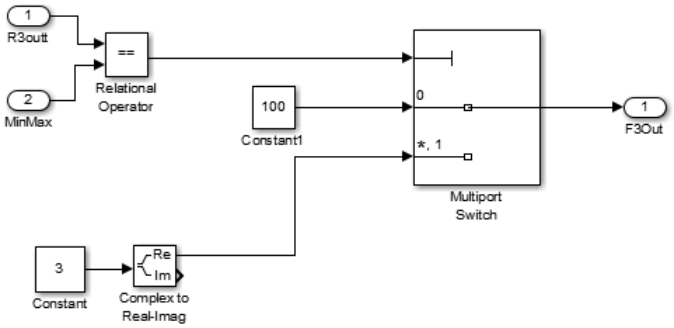

Figure 3.33: The subsystem1 from requested floor

3.34 should give the current floor value(f2) as the direction to which the next movement occurs. Otherwise it gives a constant value other than the floor value.

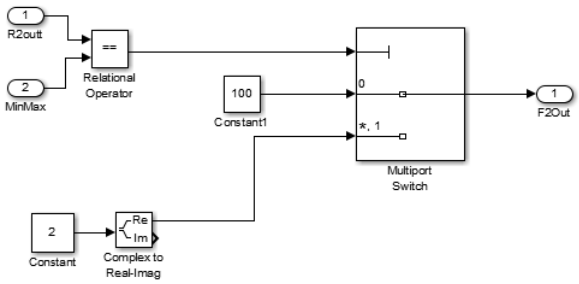

Figure 3.34: The subsystem2 from requested floor

3.35 should give the current floor value(f1) as the direction to which the next movement occurs. Otherwise it gives a constant value other than the floor value.

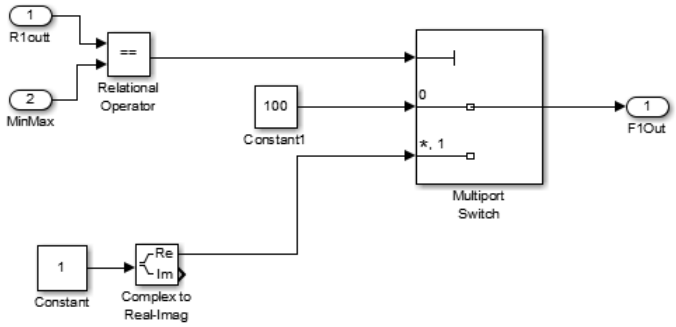

Figure 3.35: The subsystem3 from requested floor

3.36 should give the current floor value(f0) as the direction to which the next movement occurs. Otherwise it gives a constant value other than the floor value.

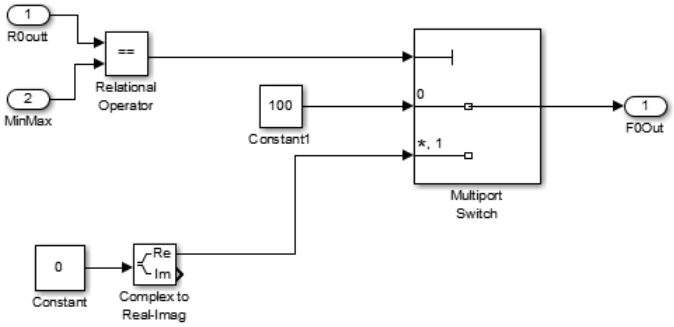

Figure 3.36: The subsystem4 from requested floor

The Next Direction subsystem 3.37 will checks whether the floor value and the current floor (where the elevator is) are equal or greater than or less than(with relational operator block). If it is true then it takes the product with 0, if it is less than then it takes the product with 1, if it is greater than then it takes the product with -1. The output of sum block is multiplied with the live request and gives the final output such that the elevator moves into the direction according to the live request.

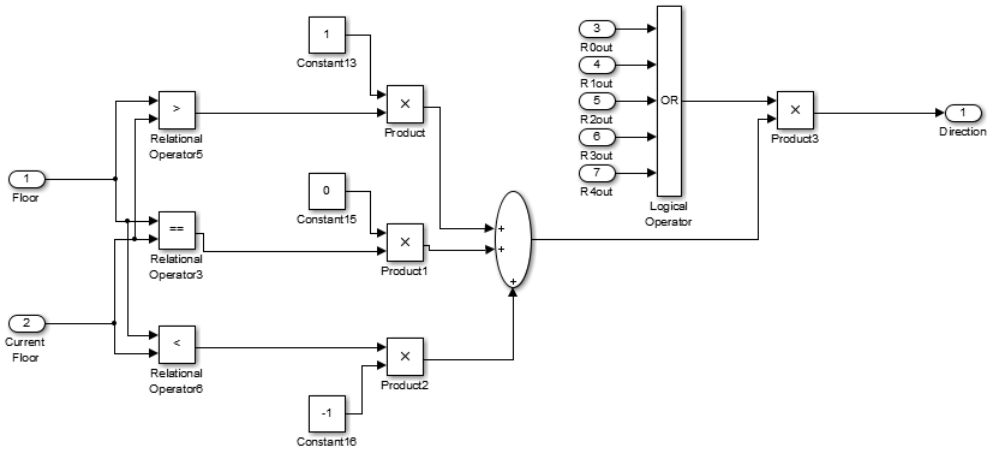

Figure 3.37: The next direction

CHAPTER 4
TESTING

4.1 TEST CASES

Testing is an important aspect in every field. There are several different methods for testing. Testing is mainly used to find the bugs in our model. Main requirements for testing is gathered from both users and also the system. Testing is one of the levels in software development life cycle or can be done module-wise. Software testing comprises with both Validation and Verification.

Testing can be done manually or automatically. Manual testing can be performed without the help of any automated testing tools. In software testing, details regarding different test cases can be reported to the manager. The tester should know whether they are using right test cases. But Automated testing is done with automated testing tools. By using automated testing, we can overcome the limitations of manual testing.

We tested the elevator model against many different test cases. The test cases were designed to exercise various salient aspects of the model.

The first test case is when there is no request from any of the floor 4.1 . The elevator will maintain its previous position.

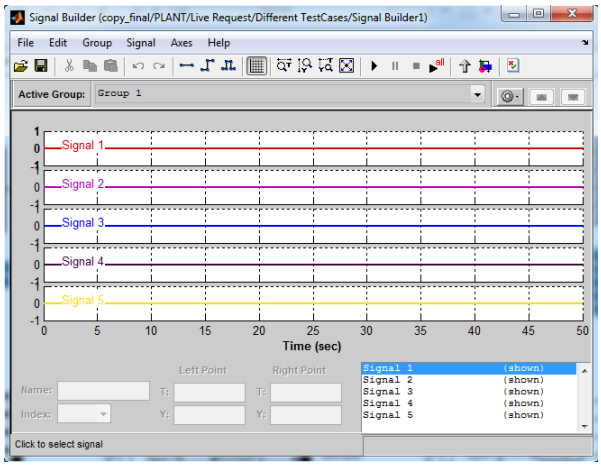

Figure 4.1: The testcase1

Output of Testcase 1 :

Figure 4.2: Scope1

In the Second test case, the request is on the third floor at time 10sec. The elevator should go up from time 10 to 40 then it will be reached on the floor three. As shown in

4.3, the model does indeed show the expected behavior.

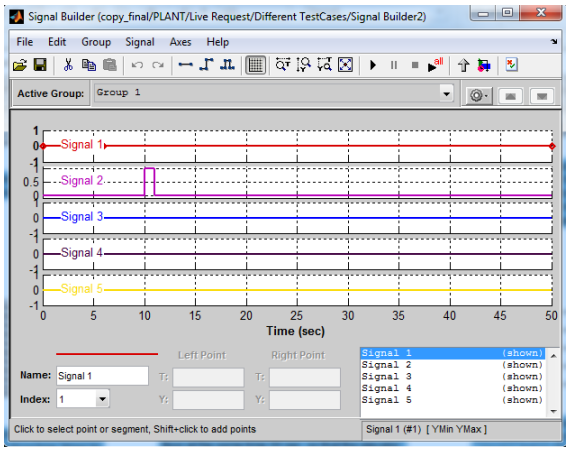

Figure 4.3: The testcase2

Output of Testcase 2 :

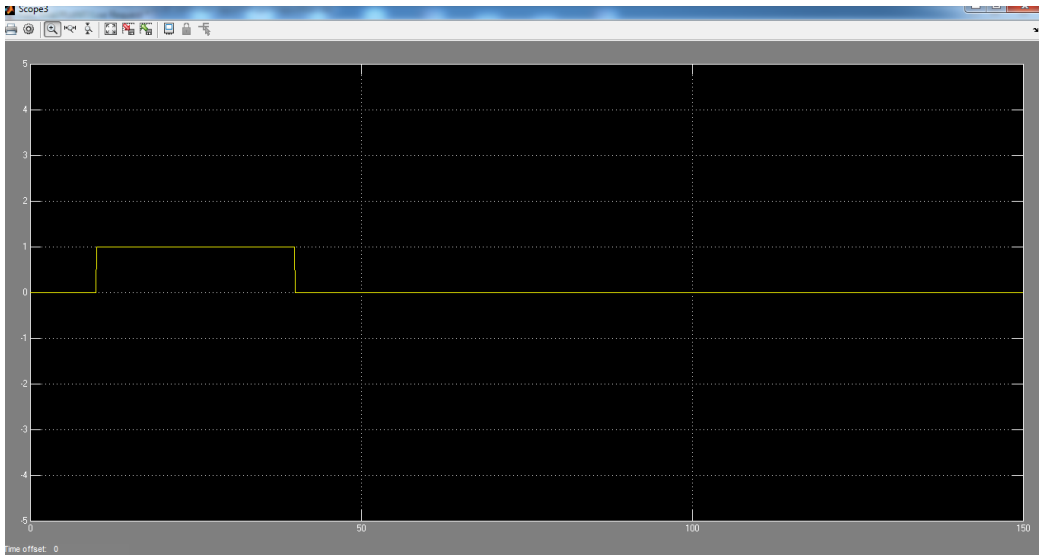

Figure 4.4: Scope2

Third test case says that the Request is from the third floor at time 20 sec and also from the fourth floor at time 30 sec, so that the elevator should go high from time 20sec to 50sec and it stops for 10sec, then it goes high at 60 to 70 after that it becomes stops. As

shown in 4.5, the models serves the request as expected.

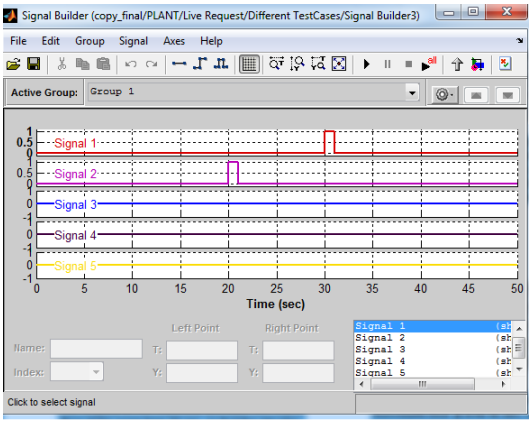

Figure 4.5: The testcase3

Output of Testcase 3 :

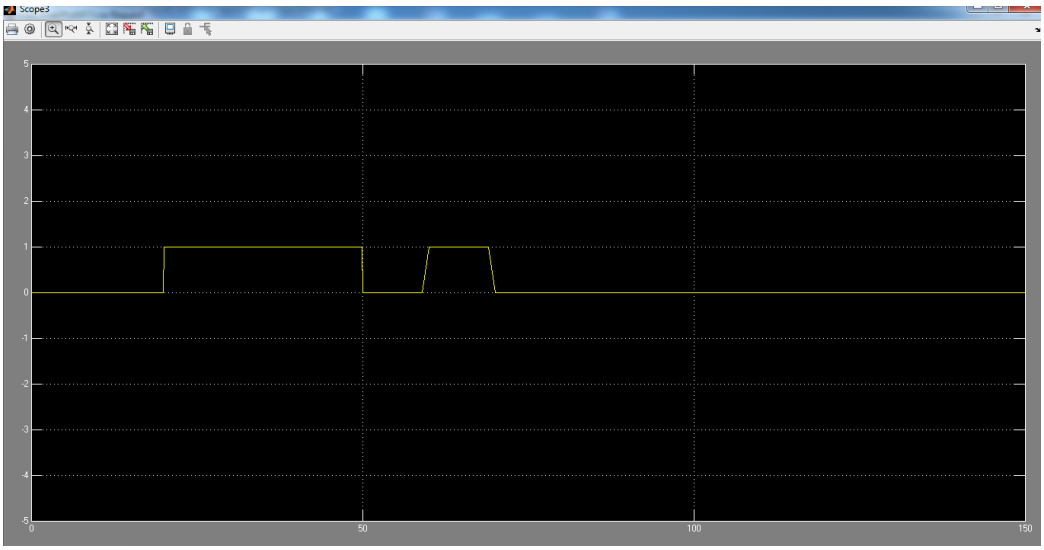

Figure 4.6: Scope3

In the fourth test case, the Request is from the third floor and also from the fourth floor at the same time 20 sec, so that the elevator start moving at time 20sec to 50sec, then it stops for 10sec and then starts moving from 60 to 70sec. As shown in 4.7, the model serves the request as expected.

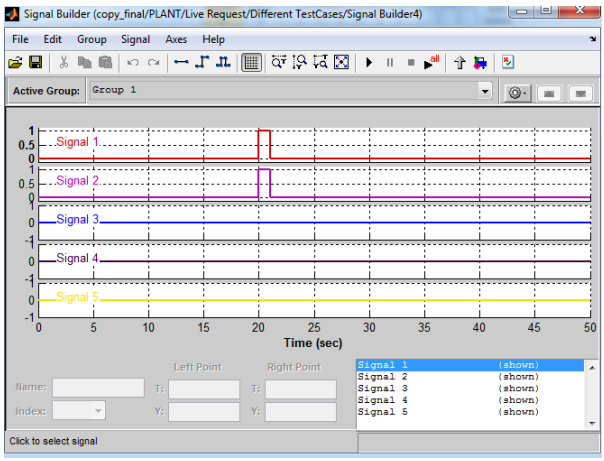

Figure 4.7: The testcase4

Output of Testcase 4 :

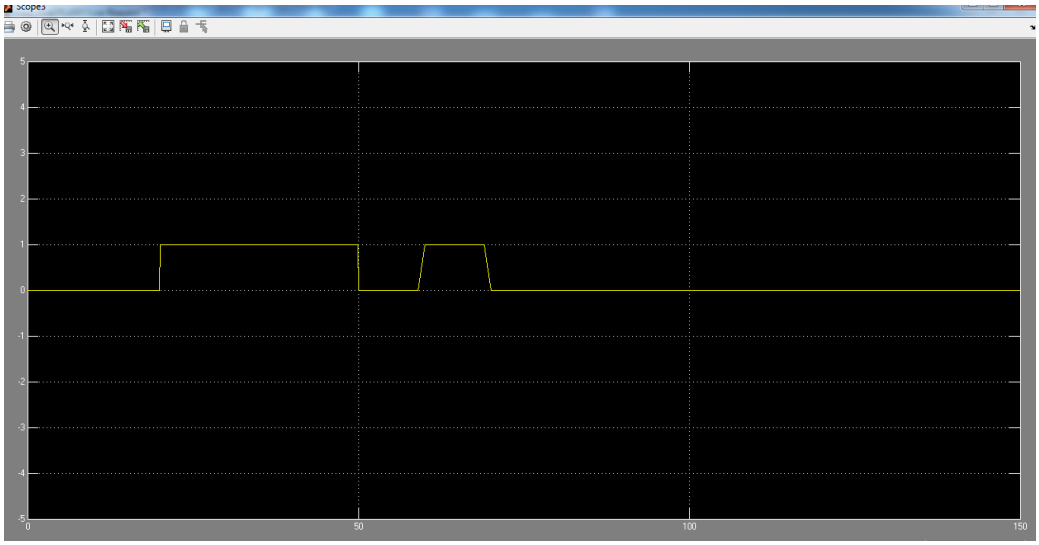

Figure 4.8: Scope4

In fifth test case, the Request is from the second floor at time 10 sec and from third floor at time 40 sec and also from fourth floor at 50sec,so that the elevator star moving at time 10sec and stops at 30 sec and again it should go high at time 40 to 50 and stops, then it again goes from 60 to 70. Rest all time it will be in the stop position. As shown in 4.9, the models serves the request as expected.

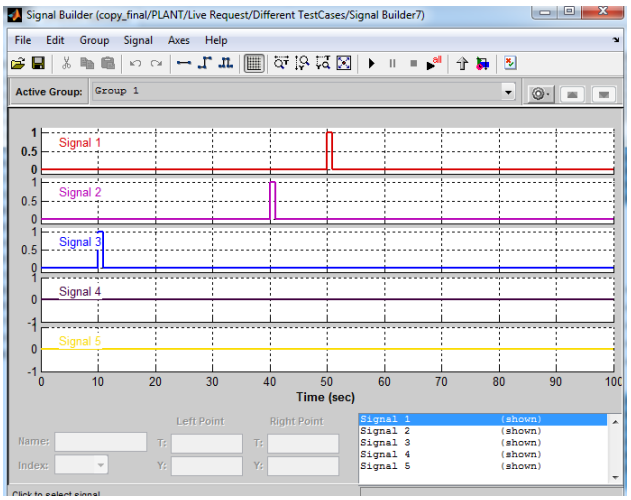

Figure 4.9: The testcase5

Output of Testcase 5 :

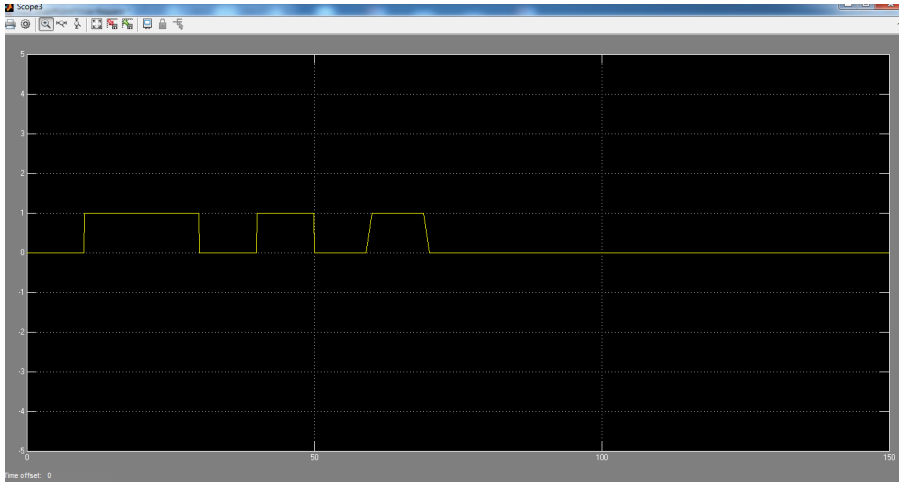

Figure 4.10: Scope5

Next test case, the Request is on the first floor at 20 and second floor at time 10 sec and from the third floor at time 40 sec and also from the fourth floor at 50sec, so that the elevator star moving at time 10sec and stops at 30sec and then move down at 40 to 50sec and then stops at 10sec and again it goes high at time60 to 80 and stops for 10sec and again start going high from 90 to 100sec. As shown in 4.11, the models serves all the

request as expected.

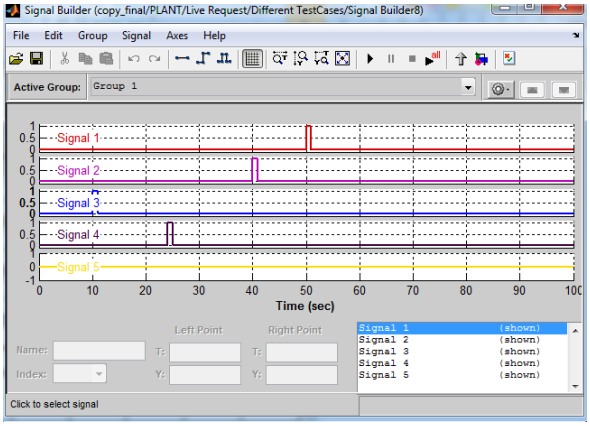

Figure 4.11: The testcase6

Output of Testcase 6 :

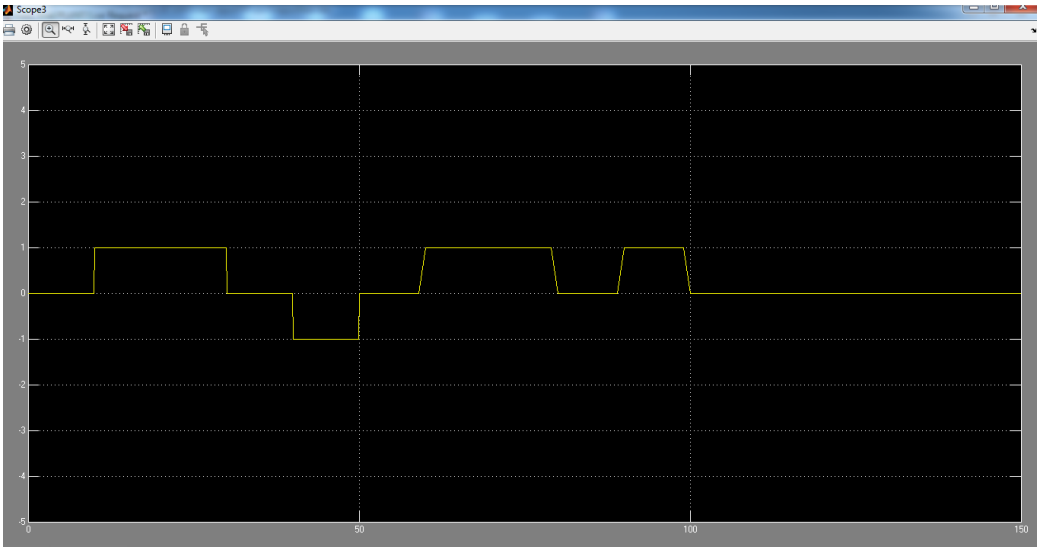

Figure 4.12: Scope6

This test case says that the Request is from the third floor at time 10sec and another request from the first floor at time 35, so the elevator should go high from time 10 to 40 and stops, then goes low from 50 to 70, after that it should stop. As shown in 4.13, the model serves all the request as expected.

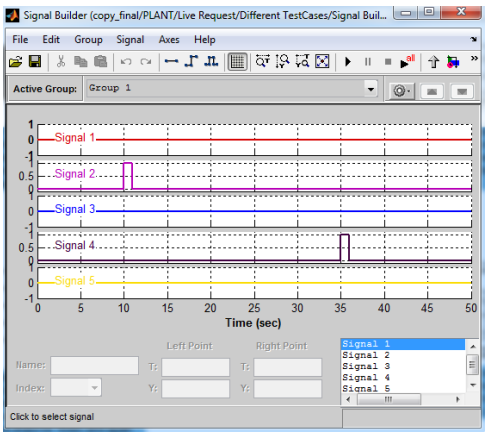

Figure 4.13: The testcase7

Output of Testcase 7 :

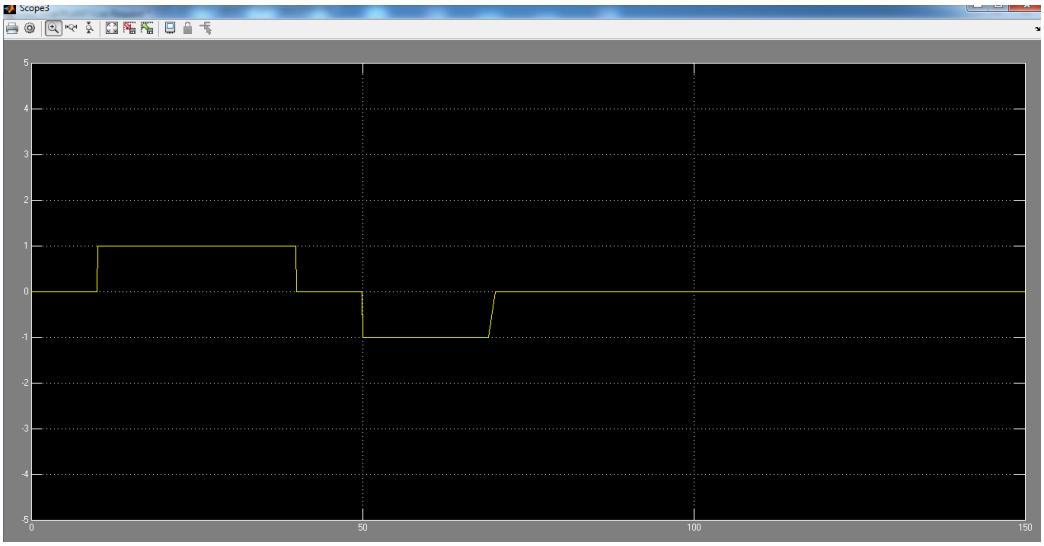

Figure 4.14: Scope7

This test cases, the Request is from the fourth floor at time 10sec and another request from the second floor at time 20 and final request from ground floor at 40sec, so the elevator should go high from time 10 to 40 and stops, then goes low from 50 to 70, after that it should stop. As shown in 4.15, the model serves the request as expected.

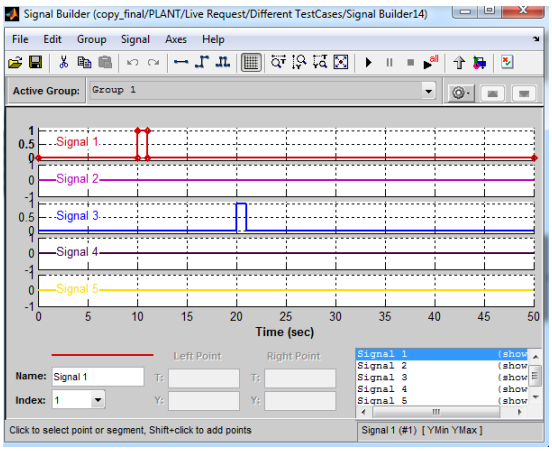

Figure 4.15: The testcase8

Output of Testcase 8 :

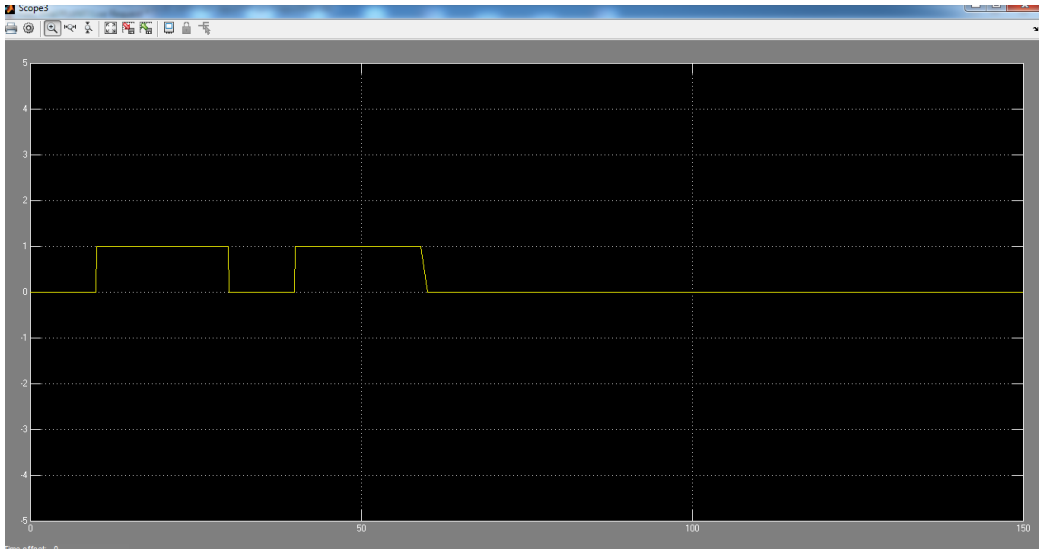

Figure 4.16: Scope8

In the below test case, the request is from four floors of the building on different time. First request is from the fourth floor at 10, so the elevator starts moving to fourth floor from time 10 to 50 and stops, then at 50 another request from third floor at 52 and move downwards and stops, then at 71 another request from second floor and again move to down and stops, again there is a request from the first floor at 92, so it start moving down

and stops. As shown in 4.17, the model serves all the request as expected.

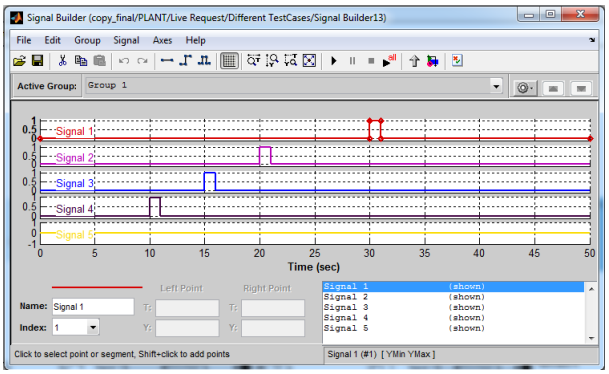

Figure 4.17: The testcase9

Output of Testcase 9 :

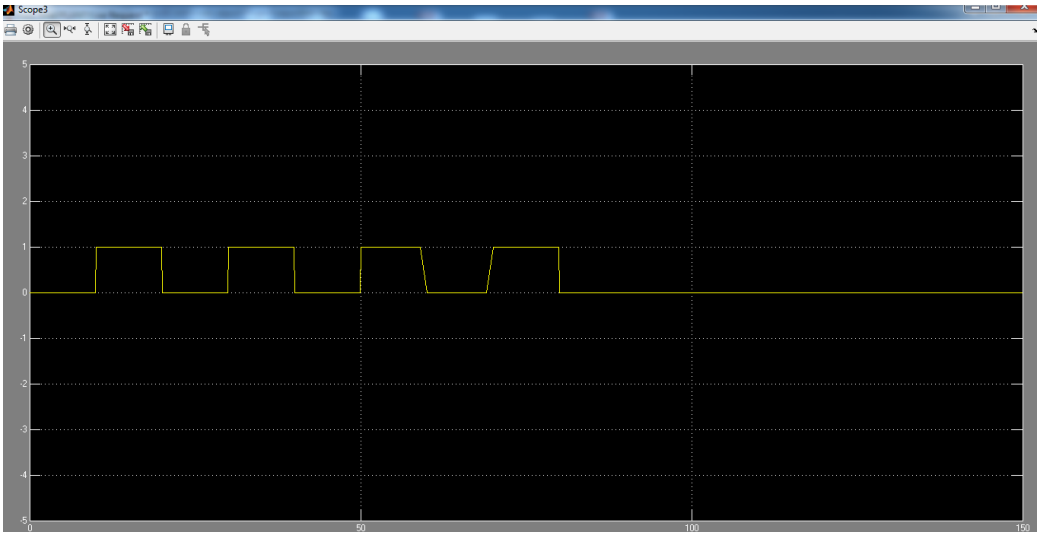

Figure 4.18: Scope9

In the tenth test case, the Request is on the first floor at time 10 to 15sec ie, somebody presses the button for a long time, so the elevator should go high from time 10sec to 20 sec. As shown in 4.19, the model serves all the request as expected.

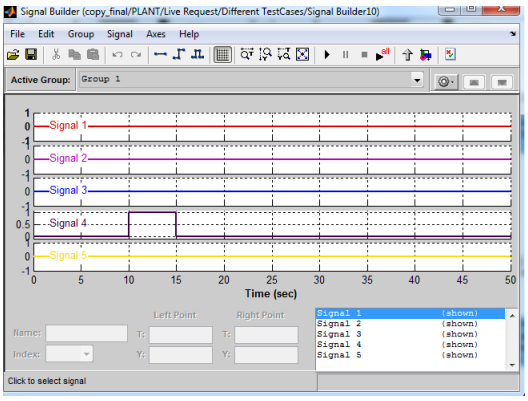

Figure 4.19: The testcase10

Output of Testcase 10 :

Figure 4.20: Scope10

In the eleventh test case, The Request is on the first floor at 10 and next request from the second floor at 15 and next from the third floor at 20 and final request from the fourth floor at 30sec. So the elevator starts moving from time 10 to serve the first request and then to the second floor then to the third floor and finally move towards the fourth floor. As shown in 4.21, the model serves all the request as expected.

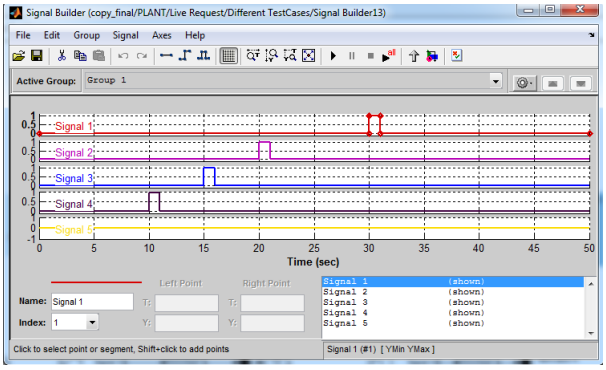

Figure 4.21: The testcase11

Output of Testcase 11 :

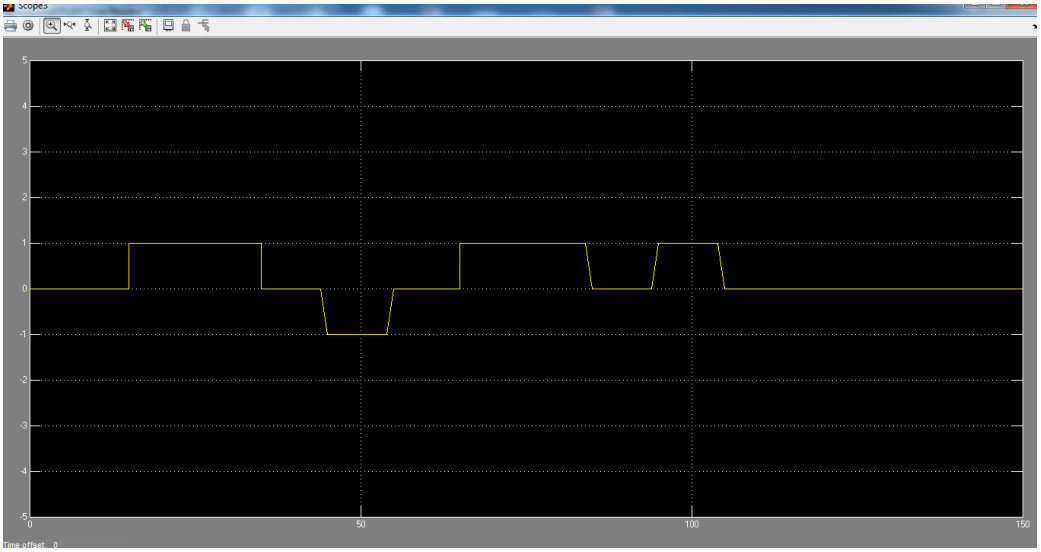

Figure 4.22: Scope11

4.1.1 Modularity

Testing a large Simulink models can be done easily by making small modules. ie, In Simulink model there can be many subsystems, each subsystem can test separately. Each subsystem contains different blocks, try to make the large subsystem as small subsystem so that it is easy to find any small bugs in each of the blocks. In this elevator model, there are many blocks with different subsystems. Finding a bug in the whole model is a difficult task. So split each block, so that it is easy to find even a very small bug.

CHAPTER 5
MODEL CHEKING ELEVATOR MODEL

5.1 TRANSLATION INTO NuSMV

Model Checking tool NuSMV is used for the translation of the elevator model in Simulink. First conversion can be done for each individual blocks. Each basic blocks in Simulink are translated into its equivalent NuSMV code separately as MODULES. The NuSMV model that is output by the translator varies with the type of input ports of the Simulink model. In Simulink model, the basic block for addition can add two scalars or two vector inputs, the type matching and conversions take place automatically. But the case with NuSMV is different from Simulink models, the module that adds two scalar inputs and two vector inputs are different. For example, consider the relational operator block in Simulink given in 5.1. Assume that the first input (in1) and the second input (in2)and the operation is checked for are =. Similarly, it is possible to check for $\leq, \geq, <, >$ in the relational operator block. The NuSMV module equivalent to the relational operator Simulink block is given below:

```
1 MODULE relational_operator
2 VAR
3   output : boolean;
4   in1    : boolean;
5   in2    : boolean;
6 ASSIGN
7   output := in1 = in2;
```

Listing 5.1: Relational Operator

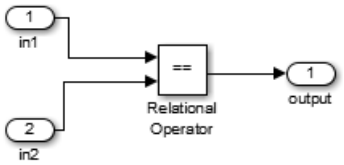

Figure 5.1: Relational operator

Similarly, it is also possible to translate every individual blocks of Simulink model into NuSMV code. The below example figure 5.2 shows the translation of a logical operator block (AND) of Simulink models. Similarly, conversion of OR, NOT, NAND, XOR, NOR, XNOR.

```
MODULE logical_operator
VAR
  output : boolean;
  in     : boolean;
  in1    : boolean;
ASSIGN
  output := in & in1;
```

Listing 5.2: Logical Operator

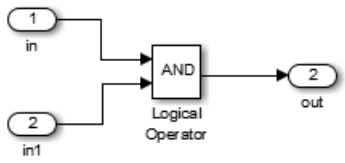

Figure 5.2: Logical operator

Cascading block may contain more than one Simulink blocks and each block are connected. This can be translated with the help of more than one MODULES in NuSMV. For example 5.3 shows the first MODULE with relational operator block of Simulink related to 5.1. Next MODULE is the product (i.e,) the output of relational operator is being multiplied with a constant value 1. Final is the main MODULE, from which both the above modules had called.

```
MODULE r_op(p1)
VAR
  output : boolean;
  in     : boolean;
  in1    : boolean;
ASSIGN
  output := in = in1;
--------------------------
MODULE product(p2)
VAR
  out        : boolean;
  output     : boolean;
ASSIGN
  out := output * 1;
--------------------------
MODULE main
VAR
rop : r_op(p1);
pdct: product(rop.p2);
```

Listing 5.3: Cascading Block

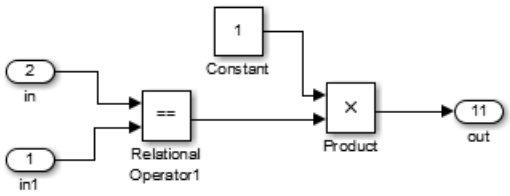

Figure 5.3: Cascading block

Figure 5.4 shows how the case statement working in NuSMV code. So that, here consider the Multiport Switch block having one control port and rest all are data port and also notice that the data port order is zero based contiguous in signal attributes (i.e,) each data port value is based on the controller part.

```
MODULE mps
VAR
  abs    : 0..1;
  in     : 0..4;
  output : 0..5;
ASSIGN
  init(abs) := 0;
  output  := case
    abs  = 0 : 5;
    abs  = 1 : in;
    esac;
```

Listing 5.4: Multiport Switch

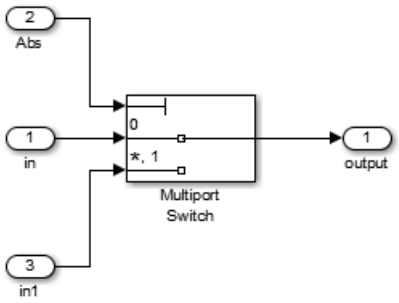

Figure 5.4: Multiport switch

Translation of delay block is shown in 5.5. If the initial value of the input to delay block is FALSE then the value of delay block on next time step is the previous input.

```
MODULE delay_bool(input)
VAR
  out   : boolean;
  in    : boolean;
ASSIGN
  init(out) := FALSE;
  next(out) := in;
--------------------------------------
MODULE main
VAR
  d : delay_bool(input);
--------------------------------------
```

Listing 5.5: Delay Block

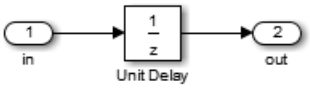

Figure 5.5: Delay

Module pdt says what is the value of ro1 then it is multiplied with 1 and ro2 is multiplied with -1. Module sum says that it takes the sum of pd1 and pd2 5.6. The counter module performs its counting after every time step. Calling of each module is done in the main module.

```
MODULE pdt(m,p)
VAR
  pd1 : {0, 1};
  pd2 : {-1, 0};
  ro1 : 0..1;
  ro2 : 0..1;
ASSIGN
  init(ro1) := 0;
  pd1 := case
    ro1 = 1 : 1 * 1;
    ro1 = 0 : 0 * 1;
         esac;
  pd2 := case
    ro2 = 1 : 1 *-1;
    ro2 = 0 : 0 * -1;
         esac;
-----------------------------------
MODULE sum(n,q)
VAR
  pos : -1..1;
```

```
21   pd1 : {0, 1};
22   pd2 : {-1, 0};
23 ASSIGN
24   pos := pd1 + pd2;
25 -------------------------------------
26 MODULE counter(q)
27 VAR
28   pos : -1 .. 10;
29 ASSIGN
30   init(pos) := -1;
31   next(pos) :=
32     case
33       pos = 10 : 1;
34       TRUE     : pos + 1;
35     esac;
36 ---------------------------------------
37 MODULE main
38 VAR
39   b : pdt(ro.ro1,ro.ro2);
40   c : sum(b.pd1,b.pd2);
41   d : counter(c.pos);
42 ---------------------------------------
```

Listing 5.6: Counter

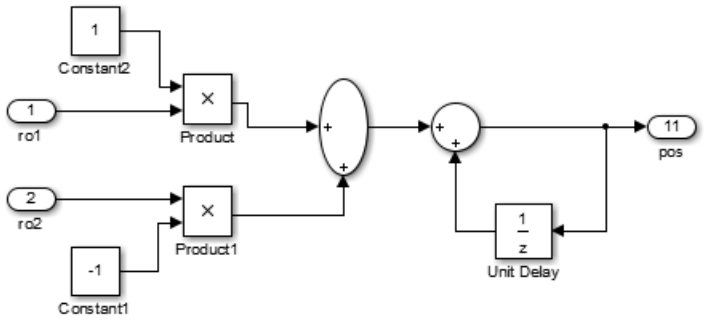

Figure 5.6: Counter

Figure 5.7 says how two subsystems can be connected. The output of the first subsystem is taken as the input for the second subsystem. Initial input is the input to the first subsystem and final out is the output from the second subsystem.

```
MODULE rop(p1)
VAR
  sub1 : boolean;
  in1  : boolean;
  in2  : boolean;
ASSIGN
  sub1 := in1 = in2;
--------------------------
MODULE and(p2)
VAR
  sub1   : boolean;
  in3    : boolean;
  sub2   : boolean;
ASSIGN
  sub2 := sub1 & in3;
--------------------------
MODULE main
```

```
VAR
r : rop(p1);
a : and(r.p2);
-------------------------------------
```

Listing 5.7: Subsystems

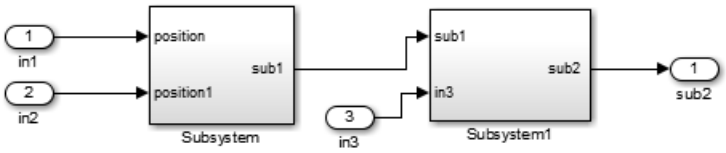

Figure 5.7: Subsystems

5.2 MODEL CHECKING USING NuSMV

Model checking using NuSMV can be done with the help of temporal logic formulas such as both LTL as well as CTL. With temporal logic formula, the model should say whether it meets the required specification. Plant Model NuSMV code shown below. First property checks whether always one of the floor value is TRUE. Specification given that it is FALSE and gives some counter examples.

```
NuSMV > check_ltlspec -p "G(e.p4 = TRUE)"
-- specification  G e.p4 = TRUE  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  b.pd1 = 0
  c.pos = 0
  c.pd1 = 1
```

```
  c.pd2 = -1
  d.pos = -1
  e.cnt = 30
  e.p4 = FALSE
  e.p3 = TRUE
-> State: 1.2 <-
  b.pd1 = 1
  b.ro1 = 1
  d.pos = 0
  e.cnt = 40
  e.p4 = TRUE
  e.p3 = FALSE
-- Loop starts here
-> State: 1.3 <-
  d.pos = 1
-> State: 1.4 <-
  d.pos = 2
-> State: 1.5 <-
  d.pos = 3
-> State: 1.6 <-
  d.pos = 4
-> State: 1.7 <-
  d.pos = 5
-> State: 1.8 <-
  d.pos = 6
-> State: 1.9 <-
  d.pos = 7
-> State: 1.10 <-
  d.pos = 8
-> State: 1.11 <-
  d.pos = 9
-> State: 1.12 <-
```

```
  d.pos = 10
-> State: 1.13 <-
  d.pos = 1

\setlength{\parskip}{12pt}
\noindent
Sometimes in plant model it checks whether the value of position is
    always 10.
NuSMV > check_ltlspec -p "F(d.pos=10)"
-- specification  F d.pos = 10  is true
```

Listing 5.8: Subsystems

Direction says to which the next movement of the model. Also check whether all the time, the direction is equal to -1. Specification tells that it is FALSE with some counter examples.

```
NuSMV > check_ltlspec -p "G(multt4.dout=2)"
-- specification  G multt4.dout = 2  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 1.1 <-
  mult4.p4 = 0
  mult3.p3 = 0
  mult2.p2 = 0
  mult1.p1 = 0
  mult0.p0 = 0
  d.out = FALSE
  d.input = FALSE
  q.Rout = 0
  q.abs = 0
```

```
  q.Routt = 5
  multt1.fd = 0
  multt2.fd1 = 0
  multt3.fd2 = 0
  sm.out = -1
  multt4.sout = -1
  multt4.lout = 0
  multt5.dout = -1
  mult4.out = 0
  mult3.out = 0
  s.cfout = 0
  multt1.out = 0
  multt2.out = 0
  multt3.out = 0
  sm.sout = -3
  multt4.dout = 0
  multt5.direction = -1

```

The direction is checking all the possibilities. It checks sometimes in future the direction may go UP or DOWN or STOPS. The last specification tells that it is FALSE with some counter examples because it will not go any other position.

```
-> State: 1.2 <-
NuSMV > check_ctlspec -p "EF(multt4.dout=1)"
-- specification EF multt4.dout = 1  is true
NuSMV > check_ctlspec -p "EF(multt4.dout=-1)"
-- specification EF multt4.dout = -1  is true
NuSMV > check_ctlspec -p "EF(multt4.dout=0)"
-- specification EF multt4.dout = 0  is true
NuSMV > check_ctlspec -p "EF(multt4.dout = 2)"
-- specification EF multt4.dout = 2  is false
```

```
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 2.1 <-
  mult4.p4 = 0
  mult3.p3 = 0
  mult2.p2 = 0
  mult1.p1 = 0
  mult0.p0 = 0
  d.out = FALSE
  d.input = FALSE
  m.dfout = 1
  m.cfout = 3
  m.cfoutt = 3
  m.out = 0
  p.cf = 3
  p.const = 0
  p.abs = 3
  p.output = 3
  q.Rout = 0
  q.abs = 0
  q.Routt = 5
  q2.Rout = 1
  q2.rout = 0
  q2.RO = 0
  q3.Rout = 0
  q3.constant = 0
  q3.Fout = 5
  f.Fout1 = 3
  f.Fout2 = 3
  f.Fout3 = 3
  f.Fout4 = 3
```

```
  f.Fout5 = 3
  f.Floor = 3
  multt1.fd = 0
  multt2.fd1 = 0
  multt3.fd2 = 0
  sm.out = -1
  multt4.sout = -1
  multt4.lout = 0
  multt5.dout = -1
  mult4.out = 0
  mult3.out = 0
  mult2.out = 0
  mult1.out = 0
  mult0.out = 0
  s.cfout = 0
  multt1.out = 0
  multt2.out = 0
  multt3.out = 0
  sm.sout = -3
  multt4.dout = 0
  multt5.direction = -1
```

Floor subsystem gives whether their any floor arrives. It Checks whether the floor value is always FALSE, the specification says that it is not TRUE and also gives some counter example.

```
NuSMV > check_ltlspec -p "G(a.fa_out =FALSE)"
-- specification  G a.fa_out = FALSE  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
```

```
  d.out = FALSE
  d.input = FALSE
  a.out = TRUE
  a.not_out = TRUE
  t.fa_out = FALSE
  n.not_out = TRUE
  a.fa_out = TRUE
  t.cc = FALSE
-- Loop starts here
-> State: 1.2 <-
  a.not_out = FALSE
  a.fa_out = FALSE
-> State: 1.3 <-
NuSMV > check_ctlspec -p "EF(a.fa_out = TRUE)"
-- specification EF a.fa_out = TRUE  is true
```

Timer subsystem checks whether there is any floor arrived, if so, check that always floor value is TRUE, then it says that no and gives some counter examples. Also, checks whether sometimes in future is there the value of the timer is high or TRUE. The specification says that it is correct so that there are no such counter-examples.

```
NuSMV > check_ltlspec -p "G(o.fa_out = TRUE)"
-- specification  G o.fa_out = TRUE  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  t.Toggle_in = TRUE
  t.Toggle_out = TRUE
  t.out = FALSE
  d.out = FALSE
  d.input = FALSE
```

```
  o.fa_out = FALSE
  o.output = FALSE
  o.Toggle_in = FALSE
-- Loop starts here
-> State: 1.2 <-
  o.fa_out = TRUE
  o.Toggle_in = TRUE
-> State: 1.3 <-
NuSMV > check_ltlspec -p "G(o.Toggle_in =TRUE)"
-- specification  G o.Toggle_in = TRUE  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 2.1 <-
  t.Toggle_in = TRUE
  t.Toggle_out = TRUE
  t.out = FALSE
  d.out = FALSE
  d.input = FALSE
  o.fa_out = TRUE
  o.output = FALSE
  o.Toggle_in = TRUE
-> State: 2.2 <-
  o.fa_out = FALSE
  o.Toggle_in = FALSE
-- Loop starts here
-> State: 2.3 <-
  o.fa_out = TRUE
  o.Toggle_in = TRUE
-> State: 2.4 <-
NuSMV > check_ctlspec -p "EF(o.Toggle_in =TRUE)"
```

```
-- specification EF o.Toggle_in = TRUE  is true
```

The controller checks whether the elevator always moves upwards or down-words or in its stop position. The specification says FALSE with counter examples. Also check sometimes in future whether it moves in all directions(UP, DOWN and STOP), these specification says TRUE.

```
NuSMV > check_ltlspec -p "G(multt5.direction=-1)"
-- specification  G multt5.direction = -1  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  sub4.r4_out = FALSE
  sub4.P4 = TRUE
  sub4.input = TRUE
  sub4.final_out4 = TRUE
  sub3.r3_out = FALSE
  sub3.P3 = TRUE
  sub3.input = TRUE
  sub3.final_out3 = TRUE
  sub2.r2_out = FALSE
  sub2.P2 = TRUE
  sub2.input = TRUE
  sub2.final_out2 = TRUE
  sub1.r1_out = FALSE
  sub1.P1 = TRUE
  sub1.input = TRUE
  sub1.final_out1 = TRUE
  sub0.r0_out = FALSE
  sub0.P0 = TRUE
  sub0.input = TRUE
```

```
sub0.final_out0 = TRUE
summ.final_out4 = FALSE
summ.final_out3 = FALSE
summ.final_out2 = FALSE
summ.final_out1 = FALSE
summ.final_out0 = FALSE
qw.f_or = FALSE
nt.p4 = FALSE
ad.p4 = TRUE
ad.r4 = TRUE
ad.fandout = TRUE
d.out = FALSE
d.input = FALSE
m.true = TRUE
m.fandout = TRUE
m.mpsout = TRUE
m.out = FALSE
fd.and_out = FALSE
fd.mpsout = FALSE
fd.p4 = FALSE
gg.r4_out = FALSE
nt3.p3 = FALSE
ad3.p3 = TRUE
ad3.r3 = TRUE
ad3.fandout = TRUE
m3.true = TRUE
m3.fandout = TRUE
m3.mpsout = TRUE
m3.out = FALSE
fd3.and_out = FALSE
fd3.mpsout = FALSE
fd3.p3 = FALSE
```

```
gg3.r3_out = FALSE
nt2.p2 = FALSE
ad2.p2 = TRUE
ad2.r2 = TRUE
ad2.fandout = TRUE
m2.true = TRUE
m2.fandout = TRUE
m2.mpsout = TRUE
m2.out = FALSE
fd2.and_out = FALSE
fd2.mpsout = FALSE
fd2.p2 = FALSE
gg2.r2_out = FALSE
nt1.p1 = FALSE
ad1.p1 = TRUE
ad1.r1 = TRUE
ad1.fandout = TRUE
m1.true = TRUE
m1.fandout = TRUE
m1.mpsout = TRUE
m1.out = FALSE
fd1.and_out = FALSE
fd1.mpsout = FALSE
fd1.p1 = FALSE
gg1.r1_out = FALSE
nt0.p0 = FALSE
ad0.p0 = TRUE
ad0.r0 = TRUE
ad0.fandout = TRUE
m0.true = TRUE
m0.fandout = TRUE
m0.mpsout = TRUE
```

```
m0.out = FALSE
fd0.and_out = FALSE
fd0.mpsout = FALSE
fd0.p0 = FALSE
gg0.r0_out = FALSE
mult4.p4 = 0
mult3.p3 = 0
mult2.p2 = 0
mult1.p1 = 0
mult0.p0 = 0
mpp.dfout = 1
mpp.cfout = 3
mpp.cfoutt = 3
mpp.out = 0
p.cf = 3
p.const = 0
p.abs = 3
p.output = 3
q.abs = 3
q.r4_out = 1
q.R4outt = 3
q.r3_out = 1
q.R3outt = 3
q.r2_out = 1
q.R2outt = 3
q.r1_out = 1
q.R1outt = 3
q.r0_out = 1
q.R0outt = 3
q2.Rout = 1
q2.rout = 0
q2.RO = 0
```

```
122 q3.Rout = 0
123 q3.constant = 0
124 q3.Fout = 5
125 f.Fout1 = 3
126 f.Fout2 = 3
127 f.Fout3 = 3
128 f.Fout4 = 3
129 f.Fout5 = 3
130 f.Floor = 3
131 g.fd = 1
132 g.floor = 3
133 g.cfv = 1
134 e.fd1 = 1
135 e.floor = 3
136 e.cfv = 3
137 l.fd2 = 1
138 l.floor = 3
139 l.cfv = 4
140 multt1.fd = 0
141 multt2.fd1 = 0
142 multt3.fd2 = 0
143 sm.out = -1
144 multt4.sout = -1
145 multt4.lout = 0
146 multt5.dout = 0
147 dq.out = FALSE
148 dq.input = FALSE
149 a.out = FALSE
150 a.not_out = FALSE
151 tt.fa_out = FALSE
152 t.Toggle_in = TRUE
153 t.Toggle_out = TRUE
```

```
154 t.out = FALSE
155 dd.v1 = TRUE
156 dd.v2 = TRUE
157 dd.v3 = TRUE
158 dd.v4 = TRUE
159 dd.v5 = TRUE
160 dd.v6 = TRUE
161 dd.v7 = TRUE
162 dd.v8 = TRUE
163 dd.v9 = TRUE
164 dd.output = TRUE
165 dd.input = FALSE
166 o.fa_out = FALSE
167 o.output = FALSE
168 cout.Toggle_out = 1
169 cout.c_out = 0
170 cout.direction = -1
171 summ.f_or = FALSE
172 qw.addition = FALSE
173 fd.r4_out = FALSE
174 gg.final_s = FALSE
175 fd3.r3_out = FALSE
176 gg3.final_s = FALSE
177 fd2.r2_out = FALSE
178 gg2.final_s = FALSE
179 fd1.r1_out = FALSE
180 gg1.final_s = FALSE
181 fd0.r0_out = FALSE
182 gg0.final_s = FALSE
183 mult4.out = 0
184 mult3.out = 0
185 mult2.out = 0
```

```
  mult1.out = 0
  mult0.out = 0
  s.cfout = 0
  multt1.out = 0
  multt2.out = 0
  multt3.out = 0
  sm.sout = -3
  multt4.dout = 0
  multt5.direction = 0
  n.not_out = TRUE
  a.fa_out = FALSE
  tt.cc = FALSE
  o.Toggle_in = FALSE
-- Loop starts here
-> State: 1.2 <-
  sub4.r4_out = TRUE
  sub3.r3_out = TRUE
  sub2.r2_out = TRUE
  sub1.r1_out = TRUE
  sub0.r0_out = TRUE
  nt.p4 = TRUE
  nt3.p3 = TRUE
  nt2.p2 = TRUE
  nt1.p1 = TRUE
  nt0.p0 = TRUE
  q2.rout = 3
  q2.RO = 3
  q3.Rout = 1
  q3.constant = 3
  q3.Fout = 3
  multt5.dout = -1
  dd.v1 = FALSE
```

```
  multt5.direction = -1
-> State: 1.3 <-
  sub4.P4 = FALSE
  sub4.input = FALSE
  sub4.final_out4 = FALSE
  sub3.P3 = FALSE
  sub3.input = FALSE
  sub3.final_out3 = FALSE
  sub2.P2 = FALSE
  sub2.input = FALSE
  sub2.final_out2 = FALSE
  sub1.P1 = FALSE
  sub1.input = FALSE
  sub1.final_out1 = FALSE
  sub0.P0 = FALSE
  sub0.input = FALSE
  sub0.final_out0 = FALSE
  nt.p4 = FALSE
  nt3.p3 = FALSE
  nt2.p2 = FALSE
  nt1.p1 = FALSE
  nt0.p0 = FALSE
  dd.v2 = FALSE
NuSMV > check_ctlspec -p "EF(multt5.direction=-1)"
-- specification EF multt5.direction = -1  is true

NuSMV > check_ctlspec -p "EF(multt5.direction=1)"
-- specification EF multt5.direction = 1  is true

NuSMV > check_ctlspec -p "EF(multt5.direction =0)"
-- specification EF multt5.direction = 0  is true

```

```
NuSMV > check_ctlspec -p "EF(cout.c_out=1)"
-- specification EF cout.c_out = 1  is true

NuSMV > check_ctlspec -p "EF(cout.c_out=-1)"
-- specification EF cout.c_out = -1  is true

NuSMV > check_ctlspec -p "EF(cout.c_out=0)"
-- specification EF cout.c_out = 0  is true
```

CHAPTER 6
RESULT

Below listing says that whether both the elevator and the request is always from the fourth floor, it checks and gives that the specification is FALSE with some counter examples.

```
NuSMV > check_ltlspec -p "G(ep.p4=TRUE & sp.R4= TRUE)"
-- specification  G (ep.p4 = TRUE & sp.R4 = TRUE)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  sub4.r4_out = FALSE
  sub4.P4 = TRUE
  sub4.input = TRUE
  sub4.final_out4 = TRUE
  qw.f_or = FALSE
  nt.p4 = FALSE
  ad.p4 = TRUE
  fd.p4 = FALSE
  gg.r4_out = FALSE
  ep.p4 = FALSE
  ep.p3 = TRUE
  sp.R3 = FALSE
  sp.R4 = FALSE
-- Loop starts here
-> State: 1.3 <-
  nt.p4 = FALSE
  nt3.p3 = FALSE
  nt2.p2 = FALSE
  nt1.p1 = FALSE
```

```
  nt0.p0 = FALSE
  dd.v2 = FALSE
  dp.pos = 1
-> State: 1.4 <-
  sub4.P4 = FALSE
  sub4.input = FALSE
  sub4.final_out4 = FALSE
  sub3.P3 = FALSE
  sub3.input = FALSE
  sub3.final_out3 = FALSE
  nt.p4 = TRUE
  nt3.p3 = TRUE
  nt2.p2 = TRUE
  nt1.p1 = TRUE
  nt0.p0 = TRUE
  dd.v3 = FALSE
  dp.pos = 2
  t1p.cnt = 10
-> State: 1.5 <-
  sub4.r4_out = FALSE
  sub3.r3_out = FALSE
  sub2.r2_out = FALSE
  sub1.r1_out = FALSE
  sub0.r0_out = FALSE
  nt.p4 = FALSE
  nt3.p3 = FALSE
  nt2.p2 = FALSE
  nt1.p1 = FALSE
  nt0.p0 = FALSE
  dd.v4 = FALSE
  dp.pos = 3
-> State: 1.23 <-
```

```
  sub4.r4_out = TRUE
  sub3.r3_out = TRUE
  sub2.r2_out = TRUE
  sub1.r1_out = TRUE
  sub0.r0_out = TRUE
  nt.p4 = FALSE
  nt3.p3 = FALSE
  nt2.p2 = FALSE
  nt1.p1 = FALSE
  nt0.p0 = FALSE
  dd.v2 = FALSE
  dd.output = TRUE
  dp.pos = 1
  t1p.cnt = 0

```

Next specification is that there should be no live request from the floor where the elevator currently is. Below two listing says that the specification is TRUE.

```
NuSMV > check_ctlspec -p "EF(ep.p4=TRUE & sub4.r4_out =FALSE)"
-- specification EF (ep.p4 = TRUE & sub4.r4_out = FALSE)  is true
```

```
NuSMV > check_ctlspec -p "EF((ep.p4=TRUE & sub4.r4_out=TRUE)-> (sub4.
    r4_out=FALSE))"
-- specification EF ((ep.p4 = TRUE & sub4.r4_out = TRUE) -> sub4.
    r4_out = FALSE)  is true
```

Next listing says that always the elevator should be in any of the floors. The below specification says that it is TRUE.

```
NuSMV > check_ctlspec -p "AG((ep.p0|ep.p1|ep.p2|ep.p3|ep.p4)=TRUE)"
-- specification AG ((((ep.p0 | ep.p1) | ep.p2) | ep.p3) | ep.p4) =
    TRUE  is true
```

The below specification says that the elevator is in the ground floor then if there is a request from the third floor, then the request should be served in future.

```
NuSMV > check_ctlspec -p "EF((sub3.r3_out=TRUE & ep.p0=TRUE)-> (sub3.
    r3_out)=TRUE)"
-- specification EF ((sub3.r3_out = TRUE & ep.p0 = TRUE) -> sub3.
    r3_out = TRUE)  is true
```

CHAPTER 7
CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

Model Checking is used for checking whether the model reaches its specification or not. Basically, several tools are used for model checking. In this project, NuSMV model checking tool is used to check whether the simple large elevator model satisfies the basic requirements. The Elevator Model contains both the controller part as well as plant part. Connecting both makes the complete elevator model. The entire elevator model is made in Simulink tool of Mathworks. Translating the model to the model checking tool, NuSMV. Finally, several properties had proved using both LTL and CTL such that the model satisfies the required specifications.

7.2 FUTURE WORK

There are several scopes for this work by extending the model with some more specifications so that the elevator model is large enough to work similar to the elevator that now we are using in large buildings or shopping malls. Generation of an automatic iterative abstraction refinement methodology (CEGAR) can also be considered in future. The abstract model may generate spurious counterexamples, analyze these counter example with new symbolic techniques, so that the refinement algorithm may reduce state space explosion problem.

REFERENCES

[1] **Meenakshi B, Abhishek Bhatnagar, and Sudeepa Roy** (2006), Tool for Translating Simulink Models into Input Language of a Model Checker, *ICFEM*

[2] **Edmund M. Clarke**, The Birth of Model Checking

[3] **Kenneth L. McMillan**, Symbolic Model Checking

[4] **Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella** (2002), NuSMV 2: An OpenSource Tool for Symbolic Model Checking, *UNIVERSITY OF TRENTO*

[5] Simulink web page: `http://www.mathworks.com/products/simulink/`

[6] Stateflow web page: `http://www.mathworks.com/products/stateflow/`

[7] NuSMV web page: `http://nusmv.irst.itc.it/`

[8] `http://nl.mathworks.com/help/simulink/`

[9] `https://en.wikipedia.org/wiki/SPIN_model_checker`

[10] `https://en.wikipedia.org/wiki/NuSMV_model_checker`

APPENDIX A
SAMPLE CODES

```
1  ----------------------------------------
2  --ELEVATOR MODEL--
3  ----------------------------------------
4  --[A]PLANT IS COMBINED WITH TWO MAIN MODULES
5  ----------------------------------------
6  -- (A.1)CALCULATING THE POSITION
7  MODULE ro
8  VAR
9    out : -1..1;
10   ro1 : 0..1;
11   ro2 : 0..1;
12 ASSIGN
13   init(out) := -1;
14   ro1 := case
15     out = -1 : 0;
16     out =  1 : 1;
17     out =  0 : 0;
18           esac;
19   ro2 := case
20     out = -1 : 1;
21     out =  1 : 0;
22     out =  0 : 0;
23           esac;
24 ------------------------------------
25 MODULE pdt(m,p)
26 VAR
27   pd1 : {0, 1};
28   pd2 : {-1, 0};
```

```
  ro1 : 0..1;
  ro2 : 0..1;
ASSIGN
  init(ro1) := 0;
  pd1 := case
    ro1 = 1 : 1 * 1;
    ro1 = 0 : 0 * 1;
         esac;
  pd2 := case
    ro2 = 1 : 1 *-1;
    ro2 = 0 : 0 * -1;
         esac;
-------------------------------------
MODULE sum(n,q)
VAR
  pos : -1..1;
  pd1 : {0, 1};
  pd2 : {-1, 0};
ASSIGN
  pos := pd1 + pd2;
-------------------------------------
MODULE counter(q)
VAR
  pos : -1 .. 10;
ASSIGN
  init(pos) := -1;
  next(pos) :=
    case
      pos = 10 : 1;
      TRUE     : pos + 1;
    esac;
-------------------------------------
```

```
61 --(A.2)POSITION ID
62 MODULE postn(r)
63 VAR
64   cnt : {0, 10, 20, 30, 40};
65   p4   : boolean;
66   p3   : boolean;
67   p2   : boolean;
68   p1   : boolean;
69   p0   : boolean;
70 ASSIGN
71   p4 := cnt = 40;
72   p3 := cnt = 30;
73   p2 := cnt = 20;
74   p1 := cnt = 10;
75   p0 := cnt = 0;
76 ----------------------------------------
77 --(A.3)TEST CASES
78 MODULE tc1
79 VAR
80   cnt   : 0..10;
81   out1  : boolean;
82   out2  : 0..11;
83 ASSIGN
84   init(out1) := FALSE;
85   init(out2) := 0;
86 --  init(cnt)  := 0;
87   next(out1) := case
88       cnt = 0 : TRUE;
89       cnt = 1 : TRUE;
90       cnt = 2 : TRUE;
91       cnt = 3 : TRUE;
92       cnt = 4 : TRUE;
```

```
93        cnt = 5 : TRUE;
94        cnt = 6 : TRUE;
95        cnt = 7 : TRUE;
96        cnt = 8 : TRUE;
97        cnt = 9 : TRUE;
98        cnt = 10 : TRUE;
99    --     TRUE : out1;
100     esac;
101   next(out2) := case
102       cnt = 0 : 2;
103       cnt = 1 : 3;
104       cnt = 2 : 4;
105       cnt = 3 : 5;
106       cnt = 4 : 6;
107       cnt = 5 : 7;
108       cnt = 6 : 8;
109       cnt = 7 : 9;
110       cnt = 8 : 10;
111       cnt = 9 : 11;
112       TRUE : out2;
113     esac;
114 --  next(cnt) :=cnt+1;
115 ------------------------------------------------
116 MODULE system(w,e)
117 VAR
118   R0:boolean;
119   R1:boolean;
120   R2:boolean;
121   R3:boolean;
122   R4:boolean;
123 ------------------------------------------------
124 --[B]CONTROLLER
```

```
125 ------------------------------------------------
126 --(B.1)SUBSYTEM IN CONTROLLER
127 ------------------------------------------------
128 MODULE sub_and4(inn,epp)
129 VAR
130 r4_out   : boolean;
131 P4       : boolean;
132 input    : boolean;
133 final_out4 : boolean;
134 ASSIGN
135 init(r4_out) := FALSE;
136 next(r4_out) := input;
137 final_out4   := input & P4;
138 ----------------------------------------
139 MODULE sub_and3(inn,epp)
140 VAR
141 r3_out   : boolean;
142 P3       : boolean;
143 input    : boolean;
144 final_out3 : boolean;
145 ASSIGN
146 init(r3_out) := FALSE;
147 next(r3_out) := input;
148 final_out3   := input & P3;
149 ----------------------------------------
150 MODULE sub_and2(inn,epp)
151 VAR
152 r2_out   : boolean;
153 P2       : boolean;
154 input    : boolean;
155 final_out2 : boolean;
156 ASSIGN
```

```
157 init(r2_out) := FALSE;
158 next(r2_out) := input;
159 final_out2   := input & P2;
160 ----------------------------------------
161 MODULE sub_and1(inn,epp)
162 VAR
163 r1_out   : boolean;
164 P1       : boolean;
165 input    : boolean;
166 final_out1 : boolean;
167 ASSIGN
168 init(r1_out) := FALSE;
169 next(r1_out) := input;
170 final_out1   := input & P1;
171 ----------------------------------------
172 MODULE sub_and0(inn,epp)
173 VAR
174 r0_out   : boolean;
175 P0       : boolean;
176 input    : boolean;
177 final_out0 : boolean;
178 ASSIGN
179 init(r0_out) := FALSE;
180 next(r0_out) := input;
181 final_out0   := input & P0;
182 ----------------------------------------
183 MODULE orr(a,b,c,d,e)
184 VAR
185 final_out4 : boolean;
186 final_out3 : boolean;
187 final_out2 : boolean;
188 final_out1 : boolean;
```

```
final_out0 : boolean;
DEFINE
f_or := final_out4 |final_out3 |final_out2 |final_out1 |final_out0;
-----------------------------------------------------------------
MODULE so(y)
VAR
  f_or : boolean;
DEFINE
addition := f_or;
-------------------------------------------------
--(B.2)UPDATING REQUEST STATUS
-------------------------------------------------
--ureqs_subsystem4
-----------------------------------------
MODULE not(q,epp)
VAR
  p4 : boolean;
ASSIGN
init(p4) := FALSE;
next(p4) := !(p4);
-----------------------------------------
MODULE and(ua,rp)
VAR
  p4 : boolean;
  r4 : boolean;
   fandout :boolean;
ASSIGN
fandout := p4 & r4;
-----------------------------------------
MODULE umps(ump,mp)
VAR
  true    : boolean;
```

```
221   fandout : boolean;
222   mpsout  : boolean;
223   out     : boolean;
224 ASSIGN
225   mpsout := case
226     fandout= FALSE  : out;
227     TRUE    : true;
228       esac;
229 ----------------------------------------
230 MODULE delay_bool(input1)
231 VAR
232   out   : boolean;
233   input: boolean;
234 ASSIGN
235   init(out) := FALSE;
236   next(out) := input;
237 ----------------------------------------
238 MODULE fiand(n,m,o)
239 VAR
240   and_out : boolean;
241   mpsout  : boolean;
242   p4      : boolean;
243 DEFINE
244   r4_out := mpsout & p4;
245 ----------------------------------------
246 MODULE s(d)
247 VAR
248   r4_out : boolean;
249 DEFINE
250   final_s := r4_out;
251 ----------------------------------------
252 --ureqs_subsystem3
```

```
253 ----------------------------------------
254 MODULE not3(w,epp)
255 VAR
256   p3 : boolean;
257 ASSIGN
258 init(p3) := FALSE;
259 next(p3) := !(p3);
260 ----------------------------------------
261 MODULE and3(ua,rp)
262 VAR
263   p3  : boolean;
264   r3  : boolean;
265    fandout :boolean;
266 ASSIGN
267 fandout := p3 & r3;
268 ----------------------------------------
269 MODULE umps3(ump,mp)
270 VAR
271   true    : boolean;
272   fandout : boolean;
273   mpsout  : boolean;
274   out     : boolean;
275 ASSIGN
276   mpsout := case
277     fandout= FALSE  : out;
278     TRUE   : true;
279       esac;
280 ----------------------------------------
281 MODULE fiand3(n,m,o)
282 VAR
283   and_out : boolean;
284   mpsout  : boolean;
```

```
285   p3       : boolean;
286 DEFINE
287   r3_out := mpsout & p3;
288 ----------------------------------------
289 MODULE s3(d)
290 VAR
291   r3_out : boolean;
292 DEFINE
293   final_s := r3_out;
294 ----------------------------------------
295 --ureqs_subsystem2
296 ----------------------------------------
297 MODULE not2(e,epp)
298 VAR
299   p2 : boolean;
300 ASSIGN
301 init(p2) := FALSE;
302 next(p2) := !(p2);
303 ----------------------------------------
304 MODULE and2(ua,rp)
305 VAR
306   p2   : boolean;
307   r2   : boolean;
308    fandout :boolean;
309 ASSIGN
310 fandout := p2 & r2;
311 ----------------------------------------
312 MODULE umps2(ump,mp)
313 VAR
314   true     : boolean;
315   fandout : boolean;
316   mpsout   : boolean;
```

```
317   out      : boolean;
318 ASSIGN
319   mpsout := case
320     fandout= FALSE   : out;
321     TRUE    : true;
322       esac;
323 -----------------------------------------
324 MODULE fiand2(n,m,o)
325 VAR
326   and_out : boolean;
327   mpsout   : boolean;
328   p2       : boolean;
329 DEFINE
330   r2_out := mpsout & p2;
331 -----------------------------------------
332 MODULE s2(d)
333 VAR
334   r2_out : boolean;
335 DEFINE
336   final_s := r2_out;
337 -----------------------------------------
338 --ureqs_subsystem1
339 -----------------------------------------
340 MODULE not1(r,epp)
341 VAR
342   p1 : boolean;
343 ASSIGN
344 init(p1) := FALSE;
345 next(p1) := !(p1);
346 -----------------------------------------
347 MODULE and1(ua,rp)
348 VAR
```

```
  p1 : boolean;
  r1 : boolean;
   fandout :boolean;
ASSIGN
fandout := p1 & r1;
----------------------------------------
MODULE umps1(ump,mp)
VAR
  true     : boolean;
  fandout : boolean;
  mpsout   : boolean;
  out      : boolean;
ASSIGN
  mpsout := case
    fandout= FALSE   : out;
    TRUE    : true;
      esac;
----------------------------------------
MODULE fiand1(n,m,o)
VAR
  and_out : boolean;
  mpsout   : boolean;
  p1       : boolean;
DEFINE
  r1_out := mpsout & p1;
----------------------------------------
MODULE s1(d)
VAR
  r1_out : boolean;
DEFINE
  final_s := r1_out;
----------------------------------------
```

```
--ureqs_subsystem0
-----------------------------------------
MODULE not0(t,epp)
VAR
  p0 : boolean;
ASSIGN
init(p0) := FALSE;
next(p0) := !(p0);
-----------------------------------------
MODULE and0(ua,rp)
VAR
  p0  : boolean;
  r0  : boolean;
   fandout : boolean;
ASSIGN
fandout := p0 & r0;
-----------------------------------------
MODULE umps0(ump,mp)
VAR
  true    : boolean;
  fandout : boolean;
  mpsout  : boolean;
  out     : boolean;
ASSIGN
  mpsout := case
    fandout= FALSE  : out;
    TRUE   : true;
      esac;
-----------------------------------------
MODULE fiand0(n,m,o)
VAR
  and_out : boolean;
```

```
413   mpsout   : boolean;
414   p0       : boolean;
415 DEFINE
416   r0_out := mpsout & p0;
417 ----------------------------------------
418 MODULE s0(d)
419 VAR
420   r0_out : boolean;
421 DEFINE
422   final_s := r0_out;
423 ----------------------------------------------------
424 --(B.3)DIRECTION
425 ----------------------------------------------------
426 --[B.3.a]COMPUTING CURRENT FLOOR
427 ----------------------------------------------------
428 MODULE multiplier4(f1,f2,f3,epp)
429 VAR
430   p4 : 0..1;
431 DEFINE
432   out := p4 * f2;
433 ------------------------------------
434 MODULE multiplier3(f1,f2,f3,epp)
435 VAR
436   p3 : 0..1;
437 DEFINE
438   out := p3 * f2;
439 ------------------------------------
440 MODULE multiplier2(f1,f2,f3,epp)
441 VAR
442   p2 : 0..1;
443 DEFINE
444   out := p2 * f2;
```

```
445 ------------------------------------
446 MODULE multiplier1(f1,f2,f3,epp)
447 VAR
448   p1 : 0..1;
449 DEFINE
450   out := p1 * f2;
451 ------------------------------------
452 MODULE multiplier0(f1,f2,f3,epp)
453 VAR
454   p0 : 0..1;
455 DEFINE
456   out := p0 * f2;
457 ------------------------------------
458 MODULE sum5(p,q,r,s,t)
459 DEFINE
460   cfout := p+q+r+s+t;
461 -------------------------------------------------
462 --MPS
463 -------------------------------------------------
464 MODULE mps(mp,mp1,mp2)
465 VAR
466   dfout : 0..1;
467   cfout : 0..5;
468   cfoutt: 0..5;
469   out   : 0..1;
470 ASSIGN
471   cfoutt := case
472     dfout = 0 : out;
473     TRUE      : cfout;
474       esac;
475 -------------------------------------------------
476 --[B.3.b]LIVE REQUEST
```

```
477 ------------------------------------------------------
478 MODULE absval(p)
479 VAR
480   cf     : 0..4;
481   const  : 0..4;
482   abs    : 0..4;
483   output: -4..4;
484 ASSIGN
485   output := cf - const;
486         abs := case
487     ((output) >= 0) : (output);
488     ((output) < 0)  : (0 -(output));
489     esac;
490 ----------------------------------------
491 MODULE outt(q,p,r,s,t,u)
492 VAR
493   abs           : 0..4;
494   r4_out      : 0..1;
495   R4outt      : 0..5;
496   r3_out      : 0..1;
497   R3outt      : 0..5;
498   r2_out      : 0..1;
499   R2outt      : 0..5;
500   r1_out      : 0..1;
501   R1outt      : 0..5;
502   r0_out      : 0..1;
503   R0outt      : 0..5;
504 ASSIGN
505   R4outt := case
506     r4_out = 0 : 5;
507     r4_out = 1 : abs;
508     esac;
```

```
509   R3outt := case
510     r3_out = 0 : 5;
511     r3_out = 1 : abs;
512     esac;
513   R2outt := case
514     r2_out = 0 : 5;
515     r2_out = 1 : abs;
516     esac;
517   R1outt := case
518     r1_out = 0 : 5;
519     r1_out = 1 : abs;
520     esac;
521   R0outt := case
522     r0_out = 0 : 5;
523     r0_out = 1 : abs;
524     esac;
525 ------------------------------------------------
526 --[B.3.c]REQUESTED FLOOR
527 ------------------------------------------------
528 MODULE min
529 VAR
530   out1 : 0..5;
531   out2 : 0..5;
532   out3 : 0..5;
533   out4 : 0..5;
534   out5 : 0..5;
535   RO   : 0..5;
536 ASSIGN
537   RO := case
538 (out1 <= out2) & (out1 <= out3) & (out1 <= out4) &(out1 <= out5) :
      out1;
```

```
539 (out2 <= out1) & (out2 <= out3) & (out2 <= out4) &(out2 <= out5) :
    out2;
540 (out3 <= out1) & (out3 <= out2) & (out3 <= out4) &(out3 <= out5) :
    out3;
541 (out4 <= out1) & (out4 <= out2) & (out4 <= out3) &(out4 <= out5) :
    out4;
542 (out5 <= out1) & (out5 <= out2) & (out5 <= out3) &(out5 <= out4) :
    out5;
543         esac;
544 -----------------------------------------
545 MODULE r(min1)
546 VAR
547   Rout    : 0..1;
548   rout    : 0..5;
549   RO      : 0..5;
550 ASSIGN
551    init(rout):= 0;
552    Rout    := case
553     rout = RO  : 1;
554     rout != RO : 0;
555     esac;
556 -----------------------------------------
557 MODULE floor(f)
558 VAR
559   Rout      : 0..1;
560   constant    : 0..4;
561   Fout      : 0..5;
562 ASSIGN
563   init(Rout) := 0;
564   Fout     := case
565     Rout = 0 : 5;
566     Rout = 1 : constant;
```

```
567         esac;
568 ------------------------------------------------------
569 --[B.3.d]MINIMUM
570 ------------------------------------------------------
571 MODULE min2(s,j)
572 VAR
573   Fout1 : 0..5;
574   Fout2 : 0..5;
575   Fout3 : 0..5;
576   Fout4 : 0..5;
577   Fout5 : 0..5;
578   Floor : 0..5;
579 ASSIGN
580   Floor := case
581 (Fout1 <= Fout2) & (Fout1 <= Fout3) & (Fout1 <= Fout4) &(Fout1 <=
      Fout5) : Fout1;
582 (Fout2 <= Fout1) & (Fout2 <= Fout3) & (Fout2 <= Fout4) &(Fout2 <=
      Fout5) : Fout2;
583 (Fout3 <= Fout1) & (Fout3 <= Fout2) & (Fout3 <= Fout4) &(Fout3 <=
      Fout5) : Fout3;
584 (Fout4 <= Fout1) & (Fout4 <= Fout2) & (Fout4 <= Fout3) &(Fout4 <=
      Fout5) : Fout4;
585 (Fout5 <= Fout1) & (Fout5 <= Fout2) & (Fout5 <= Fout3) &(Fout5 <=
      Fout4) : Fout5;
586         esac;
587 ------------------------------------------------------
588 --[B.3.e]NEXT FLOOR DIRECTION
589 ------------------------------------------------------
590 --The subsystem contains different modules, gretaer then,equal
591 --less than,or,sum and different multipliers, each functions
592 --are explained seperately in each subsystems
593 ------------------------------------------------------
```

```
--{B.3.e.1}GREATER THAN
-----------------------------------------------------
MODULE gt(l1,m1)
VAR
  fd    : 0..1;
  floor: 0..5;
  cfv   : 0..5;
ASSIGN
  fd := case
    floor > cfv : 1;
    !(floor > cfv) :0;
    esac;
-----------------------------------------------------
--{B.3.e.2}EQUAL
-----------------------------------------------------
MODULE eq(l2,m2)
VAR
  fd1   : 0..1;
  floor: 0..5;
  cfv   : 0..5;
ASSIGN
  fd1 := case
    floor = cfv   : 1;
    floor != cfv :0;
    esac;
-----------------------------------------------------
--{B.3.e.3}LESS THAN
-----------------------------------------------------
MODULE lt(l3,m3)
VAR
  fd2   : 0..1;
  floor: 0..5;
```

```
626   cfv   : 0..5;
627 ASSIGN
628   fd2 := case
629     floor    < cfv   : 1;
630     !(floor < cfv) : 0;
631     esac;
632 ------------------------------------------------
633 --IF GREATER THAN MODULE IS RIGHT, THE VALUE IS GETTING
634 --MULTIPLIED WITH 1 VALUE
635 -------------------------
636 MODULE multiple(f1,f2)
637 VAR
638   fd   : 0..1;
639 DEFINE
640   out := fd * f2;
641 ------------------------------------------------
642 --IF EQUAL MODULE IS RIGHT, THE VALUE IS GETTING
643 --MULTIPLIED WITH 0 VALUE
644 -------------------------
645 MODULE multiple1(f1,f2)
646 VAR
647   fd1   : 0..1;
648 DEFINE
649   out   := fd1 * f2;
650 ------------------------------------------------
651 --IF LESS THAN MODULE IS RIGHT, THE VALUE IS GETTING
652 --MULTIPLIED WITH -1 VALUE
653 -------------------------
654 MODULE multiple2(f1,f2)
655 VAR
656   fd2   : 0..1;
657 DEFINE
```

```
658   out   := fd2 * f2;
659 ------------------------------------------------
660 --TAKING THE SUM OF WHOLE THREE MULTIPLIERS
661 ----------------------------------------
662 MODULE sum3(x,y,z)
663 VAR
664   out   : -1..1;
665 DEFINE
666   sout := (out + out + out);
667 ------------------------------------------------
668 --TAKING OR OF ALL THE FLOORS SUCH THAT THE ELEVATOR IS ON ONE
669 --OF THE FLOOR AT THE SAME TIME
670 ------------------------------
671 MODULE or
672 VAR
673 ASSIGN
674   lout := (R4 |R3 |R2 |R1 |R0);
675 ------------------------------------------------
676 --FLOOR VALUE IS MULTIPLIED WITH THE SUM OF MULTIPLIERS
677 ------------------------------------------------
678 MODULE multiple3(f1,f2,f3)
679 VAR
680   sout : -1..1;
681   lout : 0..1;
682 DEFINE
683   dout := lout * sout;
684 ------------------------------------------------
685 MODULE multiple4(f1)
686 VAR
687   dout   : -1..1;
688 DEFINE
689   direction := dout * 1;
```

```
690 ------------------------------------------------
691 --(B.4)FLOOR ARRIVAL
692 ------------------------------------------------
693 MODULE delay_bools(aa,av)
694 VAR
695   out   : boolean;
696   input: boolean;
697 ASSIGN
698   init(out) := FALSE;
699   next(out) := input;
700 --------------------------------
701 MODULE not_module(input)
702 DEFINE
703   not_out := !input ;
704 --------------------------------
705 MODULE and_module(b,c)
706 VAR
707   out       : boolean;
708   not_out : boolean;
709 DEFINE
710   fa_out := out & not_out;
711 --------------------------------
712 MODULE crrt(inp)
713 VAR
714   fa_out : boolean;
715 DEFINE
716   cc := fa_out ;
717 ------------------------------------------------
718 --(B.5)TIMER SUBSYSTEM
719 ------------------------------------------------
720 --DELAY AFTER 10 SECONDS
721 MODULE delay(df)
```

```
722 VAR
723   v1: boolean;
724   v2: boolean;
725   v3: boolean;
726   v4: boolean;
727   v5: boolean;
728   v6: boolean;
729   v7: boolean;
730   v8: boolean;
731   v9: boolean;
732   output: boolean;
733   input : boolean;
734 ASSIGN
735   init(v1) := TRUE;
736   init(v2) := TRUE;
737   init(v3) := TRUE;
738   init(v4) := TRUE;
739   init(v5) := TRUE;
740   init(v6) := TRUE;
741   init(v7) := TRUE;
742   init(v8) := TRUE;
743   init(v9) := TRUE;
744   init(output) := TRUE;
745   next(v1) := input;
746   next(v2) := v1;
747   next(v3) := v2;
748   next(v4) := v3;
749   next(v5) := v4;
750   next(v6) := v5;
751   next(v7) := v6;
752   next(v8) := v7;
753   next(v9) := v8;
```

```
754   next(output) := v9;
755 ------------------------------------------------
756 --[B.5.a]TOGGLE IN TIMER
757 ------------------------------------------------
758 MODULE toggle(a,b)
759 VAR
760   Toggle_in : boolean;
761   Toggle_out: boolean;
762   out       : boolean;
763 ASSIGN
764   Toggle_out := case
765       Toggle_in = FALSE : out;
766       Toggle_in = TRUE  : !(out);
767       esac;
768 ------------------------------------------------
769 MODULE or_mod(p,q)
770 VAR
771   fa_out  : boolean;
772   output  : boolean;
773 DEFINE
774   Toggle_in := (fa_out | output);
775 ------------------------------------------------
776 --[B.5.b]MPS
777 ------------------------------------------------
778 MODULE ct_out(fg,gh)
779 VAR
780   Toggle_out: 0..1;
781   c_out     : -1..1;
782   direction : -1..1;
783 ASSIGN
784   c_out := case
785     Toggle_out = 0: direction;
```

```
    Toggle_out = 1   : 0;
    esac;
-------------------------------------------------
MODULE controller(co)
VAR
  c_out      : -1..1;
DEFINE
  controller_out := c_out;
-------------------------------------------------
--MAIN OF CONTROLLER
-------------------------------------------------
MODULE main
VAR
  sub4   : sub_and4(input,ep.p4);
  sub3   : sub_and3(input,ep.p3);
  sub2   : sub_and2(input,ep.p2);
  sub1   : sub_and1(input,ep.p1);
  sub0   : sub_and0(input,ep.p0);
  summ   : orr(sub4.final_out4,sub4.final_out4,sub4.final_out4,sub4.
    final_out4,sub4.final_out4);
  qw     : so(summ.f_or);
  nt     : not(sub4.p4,ep.p4);
  ad     : and(nt.p4,s.R4);
  d      : delay_bool(input);
  m      : umps(d.out, ad.fandout);
  fd     : fiand(nt.p4, m.mpsout,sub4.r4_out);
  gg     : s(fd.r4_out);
  nt3    : not3(sub3.p3,ep.p3);
  ad3    : and3(nt3.p3,s.R3);
  m3     : umps3(d.out, ad3.fandout);
  fd3    : fiand3(nt3.p3, m3.mpsout,sub3.r3_out);
  gg3    : s3(fd3.r3_out);
```

```
817  nt2   : not2(sub2.p2,ep.p2);
818  ad2   : and2(nt2.p2,s.R2);
819  m2    : umps2(d.out, ad2.fandout);
820  fd2   : fiand2(nt2.p2, m2.mpsout,sub2.r2_out);
821  gg2   : s2(fd2.r2_out);
822  nt1   : not1(sub1.p1,ep.p1);
823  ad1   : and1(nt1.p1,s.R1);
824  m1    : umps1(d.out, ad1.fandout);
825  fd1   : fiand1(nt1.p1, m1.mpsout,sub1.r1_out);
826  gg1   : s1(fd1.r1_out);
827  nt0   : not0(sub0.p0,ep.p0);
828  ad0   : and0(nt0.p0,s.R0);
829  m0    : umps0(d.out, ad0.fandout);
830  fd0   : fiand0(nt0.p0, m0.mpsout,sub0.r0_out);
831  gg0   : s0(fd0.r0_out);
832
833  mult4 : multiplier4(f1,4,sub4.p4,ep.p4);
834  mult3 : multiplier3(f1,3,sub3.p3,ep.p3);
835  mult2 : multiplier2(f1,2,sub2.p2,ep.p2);
836  mult1 : multiplier1(f1,1,sub1.p1,ep.p1);
837  mult0 : multiplier0(f1,0,sub0.p0,ep.p0);
838  s     : sum5(mult4.out,mult3.out,mult2.out,mult1.out,mult0.out);
839  mpp   : mps(d.out,s.cfout,qw.f_or);
840  p     : absval(mpp.cfoutt);
841  q     : outt(p.abs,fd0.r0_out,fd1.r1_out,fd2.r2,fd3.r3,fd.r4);
842  q2    : r(min.RO);
843  q3    : floor(q2.Rout);
844  f     : min2(floor,q3.Fout);
845  g     : gt(s.cfout,q3.Fout);
846  e     : eq(s.cfout,q3.Fout);
847  l     : lt(s.cfout,q3.Fout);
848  multt1: multiple(g.fd ,1);
```

```
849   multt2: multiple1(e.fd1 ,0);
850   multt3: multiple2(l.fd2 ,-1);
851   sm     : sum3(multt1.out,multt2.out,multt3.out);
852   multt4: multiple3(sm.sout, or.lout, dout);
853   multt5: multiple4(multt4.dout);
854
855   dq    : delay_bools(input,qw.f_or);
856     n     : not_module(dq.out);
857     a     : and_module(dq.out,n.not_out);
858     tt    : crrt(a.fa_out);
859
860 --CONTAINS TWO MAIN MODULES:TIMER WHICH CONTAINS TOGGLE SUBSYSTEM
861   t     : toggle(d.out,o.Toggle_in);
862   dd    : delay(a.fa_out);
863   o     : or_mod(output,fa_out );
864
865   cout    : ct_out(t.Toggle_out,multt5.direction);
866   ct    : controller(cout.c_out);
867 --PLANT MAIN
868 --  ap    : ro;
869   bp    : pdt(ro.ro1,ro.ro2);
870   cp    : sum(bp.pd1,bp.pd2);
871   dp    : counter(cp.pos);
872   ep    : postn(dp.cnt);
873   t1p     : tc1;
874   sp      : system(t1p.out1,t1p.out2);
875 ------------------------------------------------
```