

INDUSTRIAL SEQUENCERS FOR PROCESS CONTROL APPLICATIONS USING PETRI NETS AND LADDER LOGIC DIAGRAM

A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree*

of

MASTER OF TECHNOLOGY

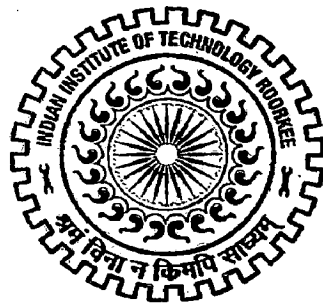
in

ELECTRICAL ENGINEERING

(with Specialization in System Engineering and Operations Research)

By

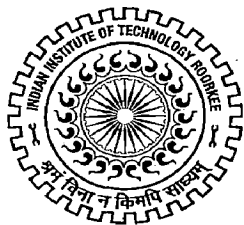
SOUMYOJYOTI MAITRA



**DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)
JUNE, 2007**

INDIAN INSTITUTE OF TECHNOLOGY, ROORKEE

ROORKEE



I.D. No.-M.T. 398/2007-31/IG

CANDIDATE'S DECLARATION


I hereby declare that the work, which is being presented in this dissertation entitled **INDUSTRIAL SEQUENCERS FOR PROCESS CONTROL APPLICATIONS USING PETRI NETS AND LADDER LOGIC DIAGRAM** in the partial fulfillment of the requirements for the award of the degree of **Master of Technology in Electrical Engineering** with specialization in **System Engineering and Operations Research**, submitted in the **Department of Electrical Engineering**, Indian Institute of Technology Roorkee, Roorkee is an authentic record of my own work carried out during a period from May 2006 to June 2007 under the supervision of **Dr. Indra Gupta**, Assoc. Professor, Electrical Engineering Department, Indian Institute of Technology Roorkee, Roorkee.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other Institute.


(Soumyojyoti Maitra)

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 28.6.07


Dr. Indra Gupta
Assoc. Professor
Department of Electrical engineering
Indian Institute of Technology
Roorkee.

ABSTRACT

Earlier hardwired electromechanical relays were used to realize and control discrete event systems, then a steep rise of the PLC was observed and today sequencing is normally implemented in software instead of relays. PLC's are industrially hardened microcomputer that performs discrete and continuous control functions in a variety of processing plants and control environments.

Originally PLC's were introduced as a relay replacement tool but in present days they find usage in all sorts of industrial control applications. From their onset in market one of the languages that gained prominence for programming PLC's is ladder logic diagrams of IEC 1131-3, recently flexible industrial automation are challenging the use of LLD's and other high level programming language methods. LLD's grow too large and difficult to understand and troubleshoot once complexity in sequence control increases.

Petri Nets originated from the field of computer science and is used as a tool for modeling asynchronous and concurrent activity, lately it is gaining popularity as an effective technique for sequence control of industrial automation systems. The popularity of Petri Nets is due to the fact that unlike LLD's not only they successfully control the system but they also help in system analysis, evaluation and simulation thereby making control strategies more easier to understand, troubleshoot, modify and evaluate.

In this dissertation complexity of sequence controllers designed using both LLD and PN for process control applications are compared based on number of basic elements used to model strategies and number of rules and logic transformations required to implement them. It is shown that with increasing complexity in applications Petri Nets (PN) supercedes its LLD counterpart in both measures thereby establishing PN as a better solution for designing controllers for discrete event systems.

Real Time Petri Net Controllers for different process control applications are synthesized on FPGA using VHDL, the syntactic and semantic compatibility of Petri Net descriptions with VHDL is observed for considered applications. Further it is observed that Field Programmable Logic for controller design results in creation of near-optimal controller implementations in terms of performance.

Acknowledgements

It is my proud privilege to express my deep sense of gratitude and indebtedness towards my thesis supervisor Dr. Indra Gupta, Assoc.Professor, Department of Electrical Engineering, for her invaluable guidance & criticism, and kind and continuous encouragement, which were the vital factors in successful completion of the present work. I am heartily thanking her for deep concern towards my academics and personal welfare.

Her painstaking support and involvement in preparation of manuscript, theoretical analysis and simulation studies are gratefully acknowledged. I humbly acknowledge a lifetime's gratitude to her and hope for a continued interaction even in the future.

I express my deep sense of gratitude to the Prof. S. P. Gupta, Head, Electrical Engineering Department, Indian Institute of Technology, Roorkee, for providing excellent facilities and nice working atmosphere in the department for the research work.

I am thankful to Prof. M. K. Vasantha in so many ways. His constant encouragement and willingness to listen to and help with my academic queries or personal problems are some of the things I benefited with. I found him a complete teacher, who taught me that it is equally important to be a good human being as much as to be a good technocrat. The education given by him will help me all my life.

My sincere thanks are also due to Dr. Surendra Kumar, Dr. Rajendra Prasad, Dr. H.O. Gupta, and Dr. G. N. Pillai for extending moral support and technical discussions as and when required during the work.

I thankfully appreciate and acknowledge my indebtedness to research scholars Dr. Vishal Kumar, Dr. Vijayender Singh, Mr. Rohit Bhakar, Mrs. Nidhi Singh, Mrs. Nidhi Kulkarni and Mr. S. K. Tomar for their help, cooperation, and moral support from time to time. My special thanks goes to the fellow classmates Mr. Shashank Saran Rai, Mr. Rohith Kumar H.C, Ms. Sowmya Kollipara, Mr. Patel Vinod Kumar, Mr. Siluveru Karunakar, Mr. Srikanth Reddy, Mr. Anil Kumar, Mr. Kalyan Ayyagiri, Mr. M. Chandrashekar, Mr. Praveen Rangiseti for their cooperation and moral support during my stay.

Besides the enjoyment of doing research work, this time has also given me an opportunity to make great friends. I shall never be able to forget the time, which I spent with Mr.

V.S.Sriram and Mr. Sheri Sundeep and Mr.Abhishek Banick and Mr. Suman Chatterjee and this writing space is not enough to write about the memories I have in my mind.

I am thankful to the technical staff of the SEOR Lab Shri Kalyan Singh, Shri C.M. Joshi, and other staff members for their timely cooperation and needful help.

Thanks are also due to all those who helped me directly and indirectly for the completion of the work.

My heartiest gratitude goes to my parents and other members of family, for constantly supporting me, which allowed me to concentrate on my work.

I would like to dedicate this research work to my family and my well wishers.

If what you are working for really matters, you will give it all you have got. Efforts may fail but don't fail to make efforts.

(Soumyojyoti Maitra)

Contents

	Page Number
ABSTRACT	i
ACKNOWLEDGEMENTS	ii
CONTENTS	iv
LIST OF FIGURES	vii
LIST OF TABLES	x
LIST OF ABBREVIATIONS	xi
1) INTRODUCTION	1
1.1 General.....	1
1.2 Formal Methods in Controller Design.....	3
1.2.1 Informal Specification.....	3
1.2.1.1 Problem specific Functional Properties.....	3
1.2.1.2 Standard Functional Properties.....	5
1.2.1.3 Non-functional Properties.....	5
1.2.2 Formalization.....	5
1.2.2.1 Formalization of Problem specific functional Properties	5
1.2.2.2 Formalization of standard functional properties	5
1.2.2.3 Formalization of non-functional properties.....	5
1.2.3 Verification and Validation.....	6
1.2.3.1 Verification.....	6
1.2.3.2 Validation.....	6
1.2.4 Evaluation.....	6
1.2.5 Implementation.....	6
1.3 Objective of Thesis.....	6
1.4 Outline of Thesis.....	7
 2) PETRI NETS	 9
2.1 Introduction.....	9
2.2 Brief History of Petri Nets.....	9
2.3 Overview of Petri Nets.....	10
2.4 Structure of Petri Nets.....	15
2.4.1 Marking and Execution rule of Petri Net	16

	Page Number
2.4.2 The Reachability Set of Petri Net.....	18
2.5 Behavioral Properties of Petri Net.....	18
2.5.1 Reachability.....	18
2.5.2 Boundedness.....	18
2.5.3 Liveness.....	18
2.5.4 Reversibility and Home State.....	20
2.5.5 Persistence.....	20
2.6 Analysis of Petri Net Model based on Reachability Criteria.....	20
2.7 Petri Nets and Industrial Automation.....	23
2.8 The Real Time Petri Net Algorithm.....	24
2.9 Conclusion.....	25
3) THE PROGRAMMABLE LOGIC CONTROLLERS	27
3.1 Introduction.....	27
3.2 Evolution of PLC.....	27
3.3 PLC Architecture.....	28
3.4 Benefits of PLC.....	31
3.5 Programming Languages.....	31
3.5.1 Graphical.....	32
3.5.1.1 Ladder diagram (LD).....	32
3.5.1.2 Functional Block Diagram (FBD).....	33
3.5.2 Textual.....	33
3.5.2.1 Instruction List.....	33
3.5.2.2 Structured Text.....	35
3.6 Conclusion.....	35
4) EVALUATION CRITERIA	35
4.1 Introduction.....	37
4.2 Criteria for Evaluation.....	39
4.2.1 Design Complexity.....	39
4.2.1.1 Graphical Complexity.....	39
4.2.1.2 Adaptability for change of Specifications.....	40
4.2.2 Response Time.....	41
4.3 Rule based Comparison.....	42

	Page Number
4.4 Example.....	44
4.5 Conclusion.....	52
5) CASE STUDIES	53
5.1 Introduction.....	53
5.2 Case Study 1:Two Level Elevator Controller System.....	54
5.3 Case Study 2:Three Level Elevator Controller System.....	62
5.4 Case Study 3:Four Level Elevator Controller System.....	71
5.5 Conclusion.....	80
6) RESULTS AND DISCUSSIONS	81
6.1 Results and Discussion.....	81
7) CONCLUSION AND SCOPE FOR FUTURE WORK	85
7.1 Conclusion.....	85
7.2 Scope for Future Work.....	85
BIBLIOGRAPHY	87
APPENDIX A	91
APPENDIX B	109
APPENDIX C	115

LIST OF FIGURES

Number	Figure Number	Figure Description	Page Number
1	Fig 1.1	Development of Standards for PLC Programming	1
2	Fig 1.2	Design Process for a Logic Control System	2
3	Fig 1.3	Formal Methods in Design Process	4
4	Fig 2.1	A Simple Graph Representation of a PN Structure	11
5	Fig 2.2	Marked Graph Representation of PN structure in Fig 2.1	12
6	Fig 2.3	A Petri Net graph after firing t2	13
7	Fig 2.4	A Petri Net graph after firing t1	14
8	Fig 2.5	A Petri Net graph after firing t3	14
9	Fig 2.6	A Petri Net graph after firing t5	15
10	Fig 2.7	The Simple PN Graph	16
11	Fig 2.8	Marked PN graph of fig 2.7 with initial marking $\mu(p1) = (1, 0, 1, 0, 2)$	17
12	Fig 2.9	Basic Firing Rules	19
13	Fig 2.10	A Petri Net with initial marking (1,0,1,0) and infinite reachable state space	22
14	Fig 2.11	Reachability Tree for PN of Fig 2.10	22
15	Fig 3.1	Architecture of a Programmable Logic Controller	29
16	Fig 3.2	An example of Ladder Logic	34
17	Fig 3.3	An example of FBD	34
18	Fig 3.4	An example of IL program	34
19	Fig 3.5	An example of ST program	34
20	Fig 4.1	Various methods of Petri Net based sequence Control	38
21	Fig 4.2	Control logic representation in PN and LLD	40
22	Fig 4.3	Basic Elements representation in LLD and PN	41
23	Fig 4.4	IF-THEN Rules for LLD and PN	43
24	Fig 4.5	Bottle Filling System	44
25	Fig 4.6	The Petri Net structure of sequence controller for bottle filling system	48
26	Fig 4.7	The Logic Diagram of the LLD algorithm for the bottle filling system	49
27	Fig 4.8	The comparison of LLD and PN algorithms for the sequence controller of bottle filling system	52
28	Fig 5.1	The Elevator Setup from Vinytics Peripherals Ltd	55
29	Fig 5.2	The RTPN structure of two level elevator sequence controller (SM Approach)	56
30	Fig 5.3	The RTPN structure of two level elevator sequence controller (Place Oriented Conditionals Approach)	57

Number	Figure Number	Figure Description	Page Number
31	Fig 5.4.1	The comparison of PN (SM) and LLD based sequence controllers for two level elevator system	59
32	Fig 5.4.2	The comparison of PN (Place Oriented conditionals Approach) and LLD based sequence controllers two level elevator system	59
33	Fig 5.4.3	The comparison of PN(SM Approach) and PN(Place Oriented Conditionals Approach) based sequence controllers for two level elevator system	60
34	Fig 5.4.4	The Comparison of LLD (Directly Obtained) and LLD (Obtained from Formalization) based sequence controllers for two level elevator system.	60
35	Fig 5.4.5	The Reachability Graph for RTPN structure of two level elevator sequence controller	61
36	Fig 5.5	The RTPN structure of a three level elevator sequence controller (SM Approach)	63
37	Fig 5.6	The RTPN structure of a three level elevator sequence controller (Place Oriented Conditionals Approach)	64
38	Fig 5.7.1	The comparison of PN (SM Approach) and LLD based sequence controllers for three level elevator system	69
39	Fig 5.7.2	The comparison of PN (SM Approach) and LLD based sequence controllers for three level elevator system	69
40	Fig 5.7.3	The comparison of PN (SM Approach) and PN (Place Oriented Conditionals Approach) based sequence controllers for three level elevator system	70
41	Fig 5.7.4	The Comparison of LLD (Directly Obtained) and LLD (Obtained from Formalization) based sequence controllers for three level elevator system.	70
42	Fig 5.8	The RTPN structure of four level Elevator Sequence Controller (SM Approach)	72
43	Fig 5.9	The RTPN Algorithm of Four Level Elevator Sequence Controller (Place Oriented Conditionals Approach)	73
44	Fig 5.10.1	The comparison of PN (SM Approach) and LLD based sequence controller for four level elevator system	78
45	Fig 5.10.2	The comparison of PN (Place Oriented Conditionals Approach) and LLD based sequence controllers for four level elevator system	78
46	Fig 5.10.3	The comparison of PN (SM Approach) and PN (Place Oriented Conditionals Approach) based sequence controllers for four level elevator system	79

Number	Figure Number	Figure Description	Page Number
47	Fig 5.10.4	The Comparison of LLD (Directly Obtained) and LLD (Obtained from Formalization) based sequence controllers for four level elevator system.	79
48	Fig 6.1	Complexity comparison of two, three and four level elevator sequence controller designed using LLD and PN Approach in terms of number of basic elements	79
49	Fig 6.2	Complexity comparison of two, three and four level elevator sequence controller designed using LLD and PN Approach in terms of sum of number of rules and logical operators	82
50	Fig 6.3	Complexity comparison of two, three and four level elevator sequence controller designed using LLD (directly implemented) and LLD (derived from formal methods) in terms of number of basic elements	83
51	Fig 6.4	Complexity comparison of two, three and four level elevator sequence controller designed using LLD (directly implemented) and LLD (derived from formal methods) in terms of sum of number of rules and logical operators	83
52	Fig A.1	Front View of Hand Held Programmer	92
53	Fig A.2	The Elevator Controller Card	95
54	Fig B.1	Simple Logic Block Structure	111
55	Fig 7.6	FPGA design flow	112

LIST OF TABLES

Number	Table Number	Table Description	Page Number
1	Table 3.1	Advantages of PLC Programming	31
2	Table 4.1	Hardware platforms used	37
3	Table 4.2	The I/O specification for the RTPN based sequence controller of bottle filling system.	47
4	Table 4.3	Comparison of LLD and PN based Sequence controller for bottle filling system	51
5	Table 5.1	I/O Specifications for two level elevator system	54
6	Table 5.2	Comparison of Sequence Controllers designed for two level elevator system	58
7	Table 5.3	I/O Specifications for three level elevator sequence controller	62
8	Table 5.4	Details description of transitions and places of RTPN structure for three level elevator sequence controller.	65
9	Table 5.5	I/O's used in RTPN structure of three level elevator sequence controller.	67
10	Table 5.6	Comparison of Sequence Controllers designed for three level elevator system	68
11	Table 5.7	I/O specifications of Four level elevator controller module	71
12	Table 5.8	The Place and transition descriptions of RTPN structures of sequence controllers for four level elevator system	74
13	Table 5.9	The I/O description for RTPN structures of four level sequence controller	76
14	Table 5.10	Comparison of sequence controllers designed for four level elevator system	77
15	Table A.1	The I/O Description of the Elevator Controller Card	97

LIST OF ABBREVIATIONS

1	IEC	International Electrotechnical Commission
2	PLC	Programmable Logic Controller
3	LLD	Ladder Logic Diagrams
4	PN	Petri Nets
5	VHDL	Very High Speed Integrated Circuit Hardware Description Language
6	FPGA	Field Programmable Gate Array
7	VPL	Vinytics Peripherals Ltd
8	FSM	Finite State Machine
9	RTPN	Real Time Petri Net
10	NEMA	National Electrical Manufacturers Association
11	CPU	Central Processing Unit
12	LD	Ladder Diagram
13	FBD	Functional Block Diagram
14	IL	Instruction List
15	ST	Structured Text
16	PLCT	PLC Trainer
17	PC	Personal Computer
18	LED	Light Emitting Diode
19	EPROM	Electronically Programmable Read Only Memory
20	HDL	Hardware Description Language
21	IP	Intellectual Property
22	DSP	Digital Signal Processing
23	VLSI	Very Large Scale Integration
24	JTAG	Joint Technology Action Group
25	IEEE	Institute of Electrical and Electronics Engineers
26	IDE	Integrated Drive Electronics
27	VGA	Video Graphic Adapter
28	PROM	Programmable Read Only Memory
29	SRAM	Static Random Access Memory
30	SDRRAM	Single Data Rate Random Access Memory
31	USB	Universal Serial Bus
32	LCD	Liquid Crystal Display
33	BST	Boundary Scan Test
34	PCB	Printed Circuit Board
35	CMOS	Complementary Metal Oxide Semiconductor
36	TTL	Transistor Transistor Logic
37	SM	State Machine

CHAPTER 1

INTRODUCTION

1.1: General

Since 1970's PLC have been the primary workhorse of industrial automation. [1] For the few last decades it has been providing a distinct field of research, development and application in the field of control engineering. [24]

The area of control engineering in particular has contributed greatly in terms of design methods and programming languages used for design of PLC's. Due to the importance of PLC's for industrial applications many of these methods have been standardized internationally.

Figure 1.1 shows an overview of the standardization, with IEC 1131 being one of the most influential one. [6]

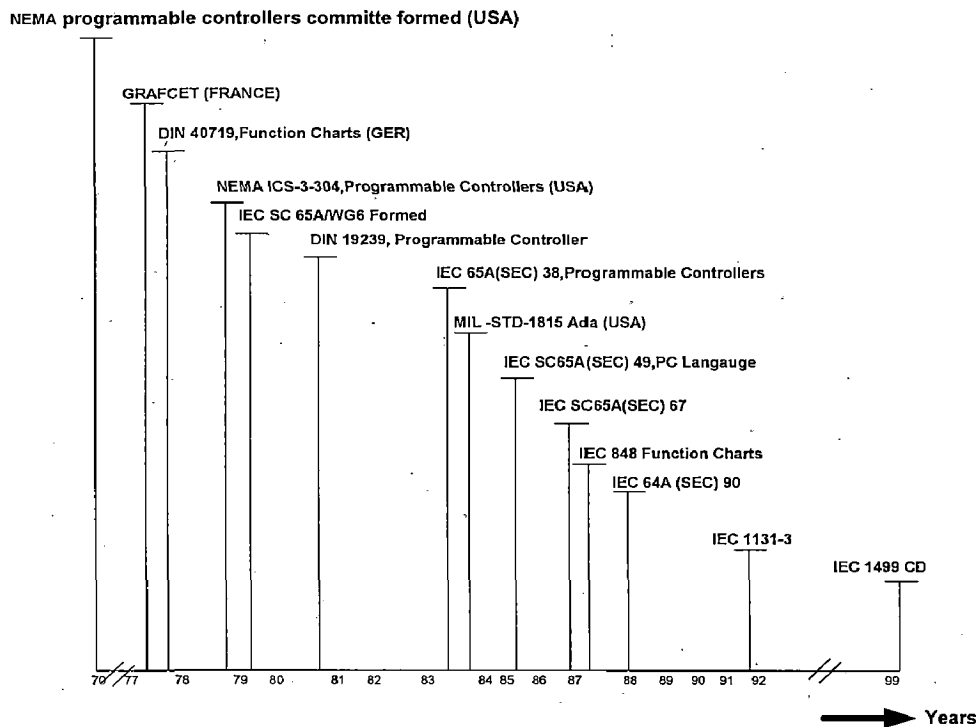


Fig 1.1 Development of Standards for PLC Programming

Although it has been a perceptive discipline for a long time, industrial PLC programming methods have been more and more influenced by formal methods. [2] [36]

There are several reasons for the application of formal methods with PLC programming:

1) The growing complexity of the control problem with the demand for reduced development time and the possible reuse of existing software modules requires the need for a formal approach in PLC programming.[1]

2) The demand for high quality solutions and especially the application of PLC's in safety critical processes need verification and validation procedures, hence there is a requirement of formal methods in the process of design to prove specific static and dynamic properties of the programs. For example properties such as liveness, boundedness and reachability. [1]

In figure 1.2 a general model of the logic control design process is shown, without the use of formal methods the controller design process only consists of the external ring. The controller is realized from the informal specification by direct implementation and afterwards it is informally validated against the informal specifications. [1]- [2]

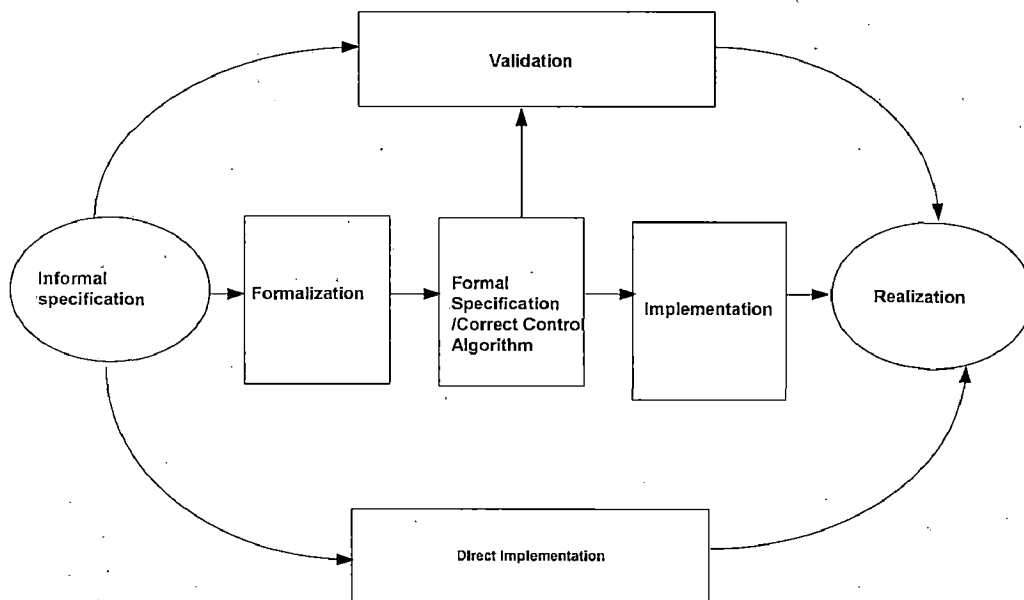


Fig 1.2 Design Process for a Logic Control System

In almost all design procedures, the designer of a logic control system starts with a given informal specification of the problem in hand. [36]

“The term informal refers to everything that is not based on strictly composed, syntactically and semantically well-defined form”. [1] In general the informal specification consists of details of the uncontrolled process and the requirements for control system.

The industrial standard approach to obtain the realization from the informal specification is direct implementation of controller using any one of the standard PLC programming languages and with standard hardware and well defined PLC functionality, the realization consists of the programmed control algorithm [1]. The non model based validation or in short the informal method of validation involves test of the implemented controller against the informal specification. The problems that are encountered with informal validation procedure is that it lacks completeness and is quite time consuming involving a lot of man-hour in work.

Recent years have seen a need amongst the PLC fraternity to use formal methods in programming and validation because of the advantages we have already discussed. Formalization refers to conversion of an informal specification to a formal specification and deriving the realization from formal specification is called Implementation. [34]

1.2: Formal methods in Controller Design

Figure 1.3 demonstrates the different parts of informal specification, how they are formalized and how the control algorithm obtained be checked against the formal specification.

1.2.1: Informal Specification

An informal specification contains different types of properties, the functional properties include standard and problem specific properties along with the non-functional ones the prominent one being quality requirements to be met by the controller.

1.2.1.1 Problem specific Functional Properties

The problem specific functional properties include those aspects of informal specification that describes the expected behavior of the controller under design.

Ex: turn on/off a motor when a certain switch is pressed.

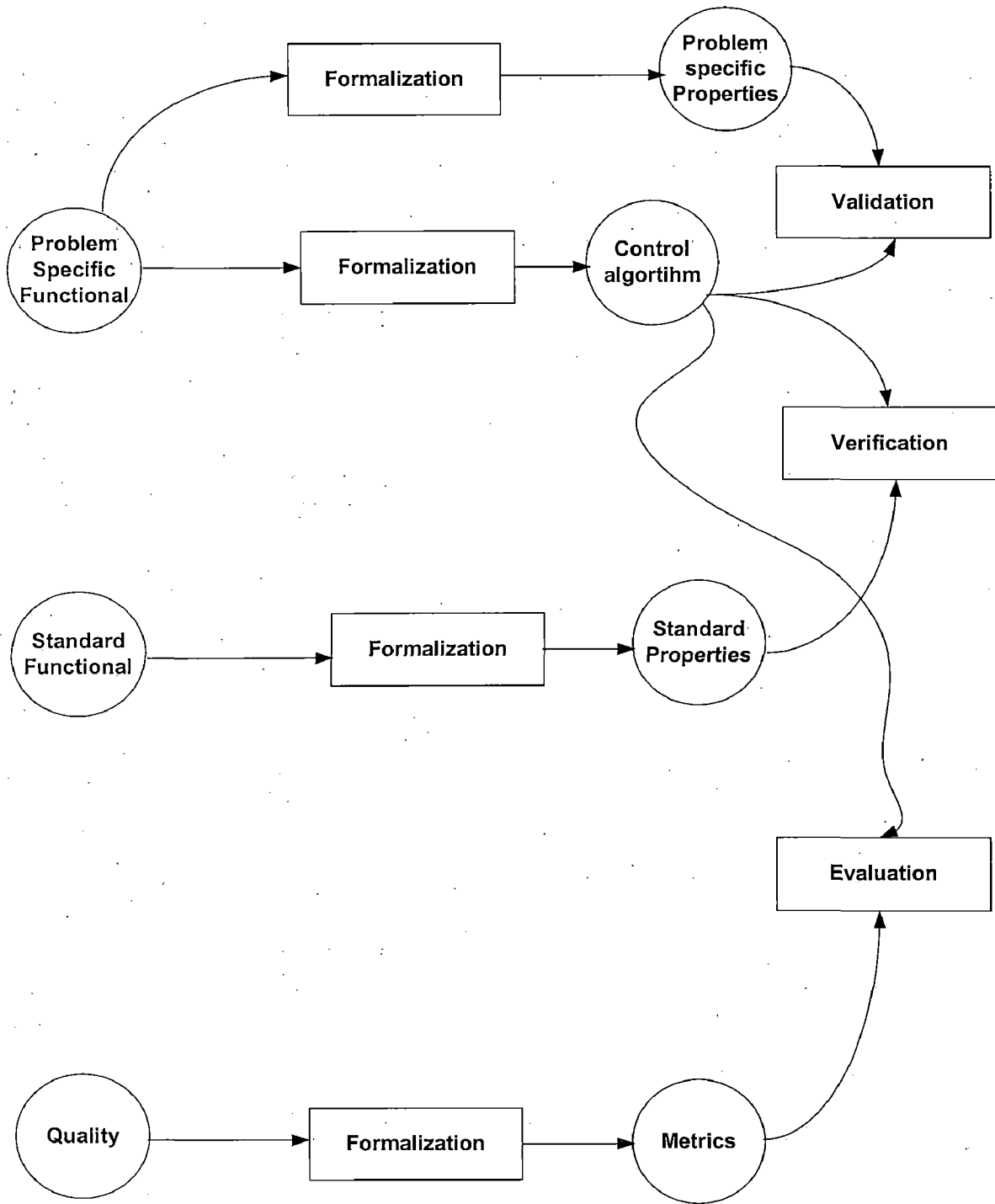


Fig 1.3 Formal Methods in Design Process

1.2.1.2 Standard Functional Properties

Standard functional properties are not dependent on the control objective but formalization of these properties does depend on the formalism used to describe the controller.

Ex: A controller should never get stuck in an infinite loop.

1.2.1.3 Non-Functional Properties

These are the properties that do not affect the behavior of controller, mainly they include details about software quality.

1.2.2: Formalization

In order to be used in a formal design approach all the informal properties need to be formalized.

1.2.2.1 Formalization of Problem Specific Functional Properties

It consists of two parts: first, the formalization of the control algorithm and then the formal description of problem specific properties to be fulfilled by the algorithm. The formalization of problem specific properties is done manually and is the main effort taking assignment during controller development cycle.

1.2.2.2 Formalization of Standard Functional Properties

“Standard properties include first and foremost that the algorithm respects the precepts of his formalism”. [2]

1.2.2.3 Formalization of Non-Functional Properties

All non-functional properties cannot be formalized, however it is possible to connect a non-functional property to some measurable properties of algorithm and hence, obtain a corresponding measure.

Ex: Readability of an algorithm can be connected to number of comments.

1.2.3: Verification and Validation

As, both the terms are not disjoint so often one seems to confuse one term with another. The two terms as given in both reference [1] and [2] is

“<< Verification: Am I building the product right? >>

<< Validation: Am I building the right product? >>”

1.2.3.1 Verification

Refers to the correct functioning of control algorithm while adhering to the standard functional properties. It is the first step after all formalizations and if result of verification is negative, the control algorithm has to be redesigned again and again until a satisfactory result is obtained.

1.2.3.2 Validation

Refers to the correct functioning of designed control algorithm while adhering to the problem specific functional properties. If the result of validation is negative, then first, the control algorithm is checked for failure and if there is no failure in control algorithm, next it is checked if informal property is formalized properly. If error is found in the process of formalization then the informal properties are rechecked and formalized again if required with additional assumptions to meet the correct validation requirements. Care must be taken that if during validation process the control algorithm is modified then verification of the modified control algorithm is must before proceeding to the next step.

1.2.4: Evaluation

In this process the non-functional properties of the controller are verified and measured.

1.2.5: Implementation

It is the last and final step in design process where a correct realization of the controller under design is obtained from the correct formal description so formulated.

1.3: Objective of the Thesis

In this thesis complexity of sequence controllers designed using both ladder logic diagrams and Petri Nets for the following process control applications:

- a) A bottle filling system.
- b) A two level elevator system.

- c) A three level elevator system.
- d) A four level elevator system.

are compared based on number of basic elements used to model strategies and number of rules and logic transformations used to implement them. It has been shown that with increasing complexity in applications Petri Nets supercedes its LLD counterpart in both the measures thereby establishing PN as a better tool for designing controllers for process control applications.

Further, real time Petri Net controllers for all the above applications are synthesized on FPGA using VHDL and ladder logic codes are also derived from PN structures and it is observed that the LLD codes obtained by using the formal methods like PN instead of direct realization from informal specification gives more compact and readable codes than the ones obtained from direct implementation.

1.4: Outline of the Thesis

Chapter two discusses the Petri Nets concepts along with the extension of RTPN.

Chapter three gives a brief idea about the PLC and the languages that are used for the programming of PLC based applications.

Chapter four gives a short description of the criteria of comparison which are explained in detail by considering a controller designed for a bottle filling system by the method of direct implementation along with the use of formal methods.

Chapter five discusses all the case studies used in this thesis and also compares and tabulates both PN and LLD structure of controllers developed for these applications.

Chapter six discusses the results derived from the experiments carried out during the thesis and finally chapter seven concludes the thesis and lays down the future work that could be done using formal methods in field of sequencer designs.

CHAPTER 2

PETRI NETS

2.1: Introduction

Petri Net is an abstract, formal model of information flow. The properties, concepts and techniques of Petri nets have been developed with an aim to obtain a natural, simple and powerful tool for analysis and study of flow of information and control in systems with asynchronous and concurrent activities. [3] [18]

Petri Nets are widely used as a graphical and mathematical modeling tool for studying and describing information systems. As a graphical tool it is very similar in form to flowcharts and block diagrams. Unlike flowcharts and block diagrams PN possess the concept of tokens which are used to simulate the dynamic and concurrent behavior of a system. As a mathematical tool it is used to derive state equations, algebraic equations and other mathematical properties governing the behavior of systems.

The main advantage of PN as modeling tool is that it is popular with both theoreticians and practitioners. The practitioners can learn from theoreticians how to make a model more methodical in turn teaching the latter to make their model more realistic.

2.2: Brief History of Petri Nets

The theory of Petri Nets has developed from the work of Carl Adam Petri, A. W. Holt, Jack Dennis, and many others. Petri Nets originated in the early work of Petri, in Germany, who in his thesis, developed a new model of information flow in systems. This model was based on the concepts of asynchronous and concurrent operation by the parts of a system and the realization that relationships between the parts could be represented by a graph, or Net. [3]

The ideas communicated by Petri in his thesis drew attention from a group of researchers at Applied Data Research, Inc. working on the Information Systems Theory Project. This group led by Anatol Holt, developed the theory of systemic which was concerned with the representation and analysis of systems and their behavior. It was their work that led to the

development of early theory, notation, and representation of Petri Nets, and showed how Petri Nets could be applied to the modeling and analysis of systems of concurrent processes. Applied Data Research's associations with Project MAC at MIT, and particularly the Computation Structures Group under the direction of Jack Dennis, introduced the concepts of Petri Nets to this latter group. The Computation Structures Group has been a most productive source of research and literature in this field, publishing several PhD theses and numerous reports and memos on Petri Nets. Two pertinent conferences have been held by the Computation Structures Group: the Project MAC Conference on Concurrent Systems and Parallel Computation at Woods Hole in 1970, and the Conference on Petri Nets and Related Methods at MIT in 1975.[21]

Unlike the work of Petri, Holt, and many European researchers, who emphasized more on the fundamental concepts of systems, the work at MIT and many other American research centers concentrates on those mathematical aspects of Petri Nets that are more closely related to automata theory. This approach is motivated by a desire to analyze systems by modeling them as Petri Nets, and then manipulating the Petri Nets to derive Properties of the modelled systems. [22]

A large amount of research has been done on both the nature and the application of Petri Nets, and their use seems to be expanding. The simplicity and power of Petri nets make them excellent tools for working with asynchronous concurrent systems. [3]

2.3: Overview of Petri Nets

In this section, our primary aim is to study and understand the static and dynamic properties of systems modeled by means of Petri Nets by means of a simple example as shown in Fig 2.1. The Petri Net models the static properties of a system just like a flowchart represents the static properties of a computer program. The graph contains two types of nodes: circles (called *places*) and bars (called *transitions*). These nodes, places and transitions are connected by directed arcs from places to transitions and from transitions to places. If an arc is directed from node i to node j (either from a place to a transition or a transition to a place), then i is an *input to j* and j is an *output of i* . For example in the fig 2.1, place P_1 is an input to transition t_2 while places P_2 and P_3 are outputs of transition t_2 .

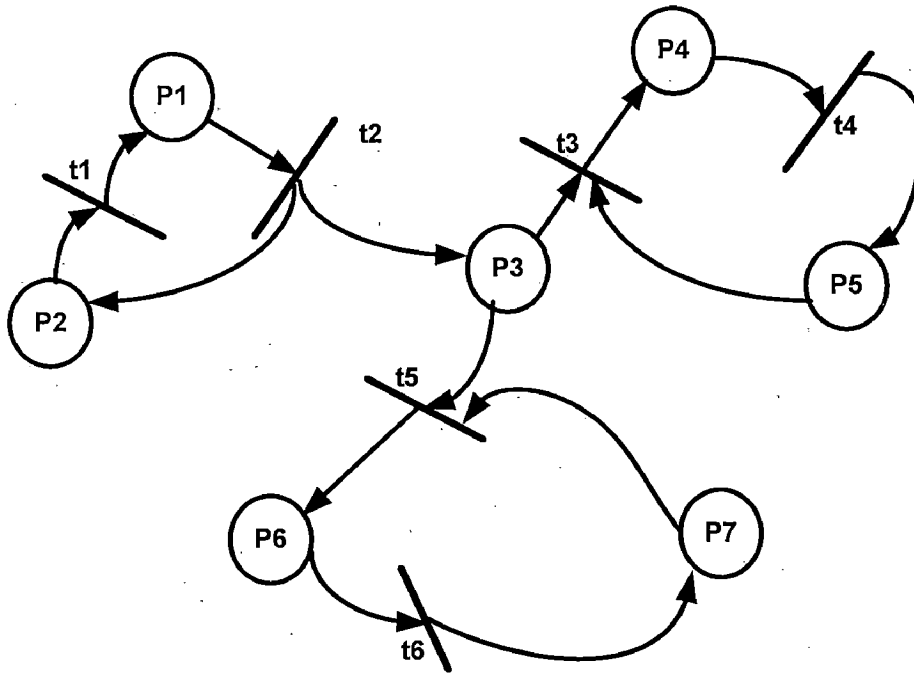


Fig 2.1 A Simple Graph Representation of a PN Structure

In addition to the static properties, a Petri Net also has dynamic properties that result from its execution. The execution of a Petri Net is controlled by the position and movement of **tokens** represented by a black dot that reside in circles representing places of the net. A Petri Net with marking is called a *marked Petri Net*.

One of the various marked Petri Net structures possible for the fig 2.1 example is as shown in fig 2.2. The position of tokens at various places in Petri Net graph is governed by a set of rules. They are

- 1) A transition t is said to be enabled if each input place p of t is marked with at least $w(p, t)$ tokens, where $w(p, t)$ is the weight of the arc from p to t .
- 2) An enabled transition may or may not fire (depending on whether or not the event actually takes place).
- 3) A firing of an enabled transition t removes $w(p, t)$ tokens from each input place p of t , and adds $w(t, p)$ tokens to each output place p of t , where $w(t, p)$ is the weight of the arc from t to p . [4]

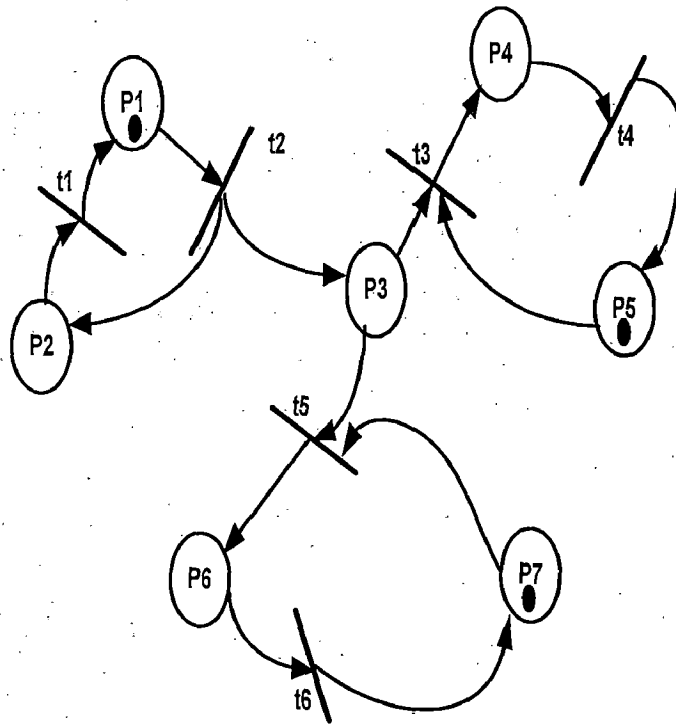


Fig 2.2 Marked Graph Representation of PN structure in fig 2.1

In short tokens are moved by firing of transitions of net, which are enabled when all the input places to transition have token in them. The transition fires by removing enabling tokens from their input places and by generating new tokens which are deposited in output places. In the marked graph example in fig 2.2 transition t2 is enabled since it has got a token in its input place t1, while t5 is not enabled since one of its input places p3 doesn't possess any tokens.

The Petri Net structure that results from firing of enabled transition t2 in fig 2.2 is as shown in fig 2.3. In fig 2.3 t2 fires thereby removing token from its input place P1 and depositing tokens one each at its output place P2 and P3. The distribution of tokens in a marked Petri Net signifies the state of net and is called its marking. In fig 2.3 three transitions t1, t3 and t5 are enabled due to firing of t2, the marked net structure that results due to firing of them single handedly is as shown in fig 2.4, fig 2.5 and fig 2.6 respectively.

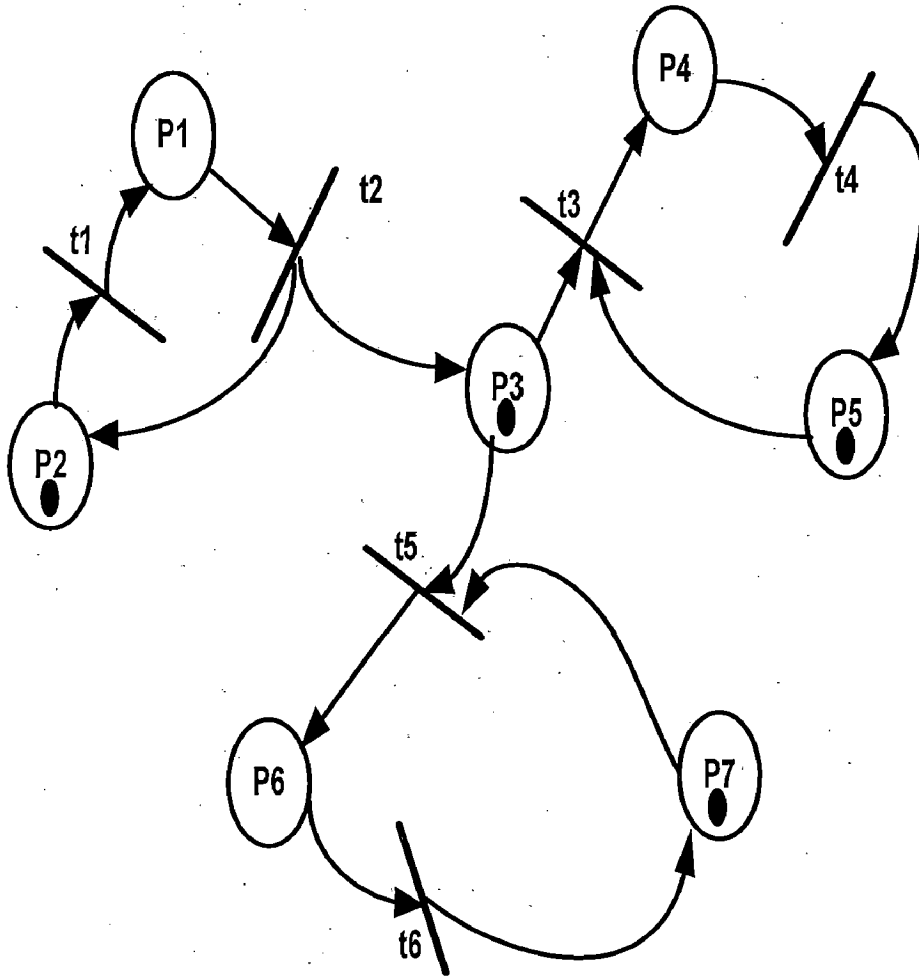


Fig 2.3 A Petri Net Graph after firing t2

Now this process of obtaining new marked net structure will continue as long as any enabled transition will be present.

The whole discussion about static and dynamic properties of systems modeled by using Petri Nets was to give an introduction to concepts and working of Petri Nets. With this knowledge gained from our discussion, we are now in a position to study the structure of Petri Net.

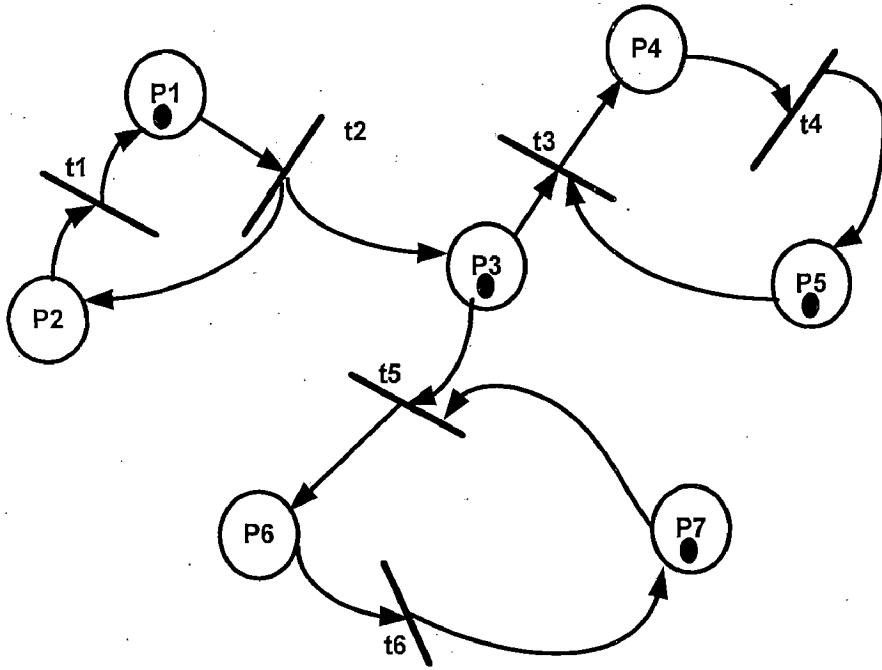


Fig 2.4 A Petri Net Graph after firing t1

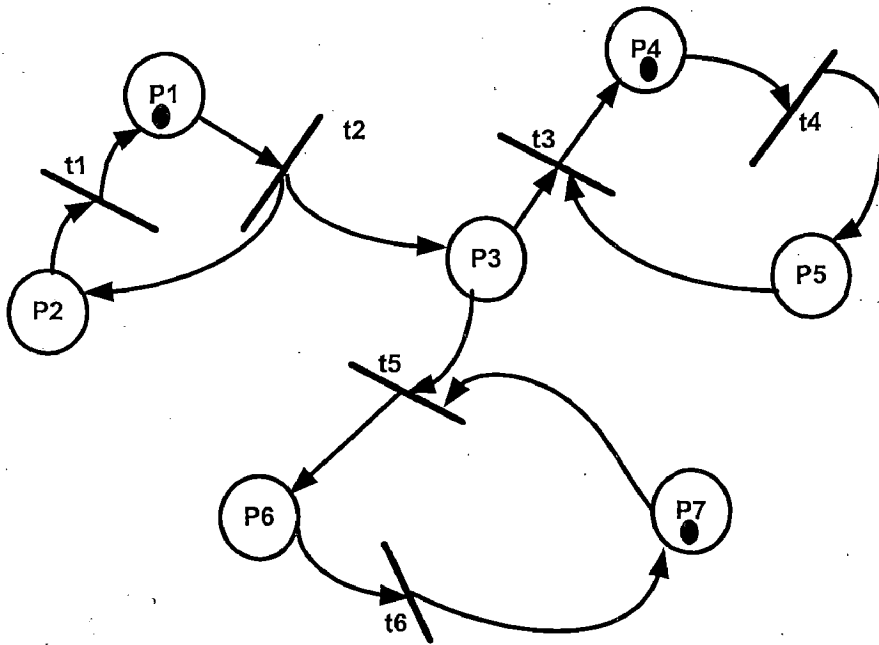


Fig 2.5 A Petri Net Graph after firing t3

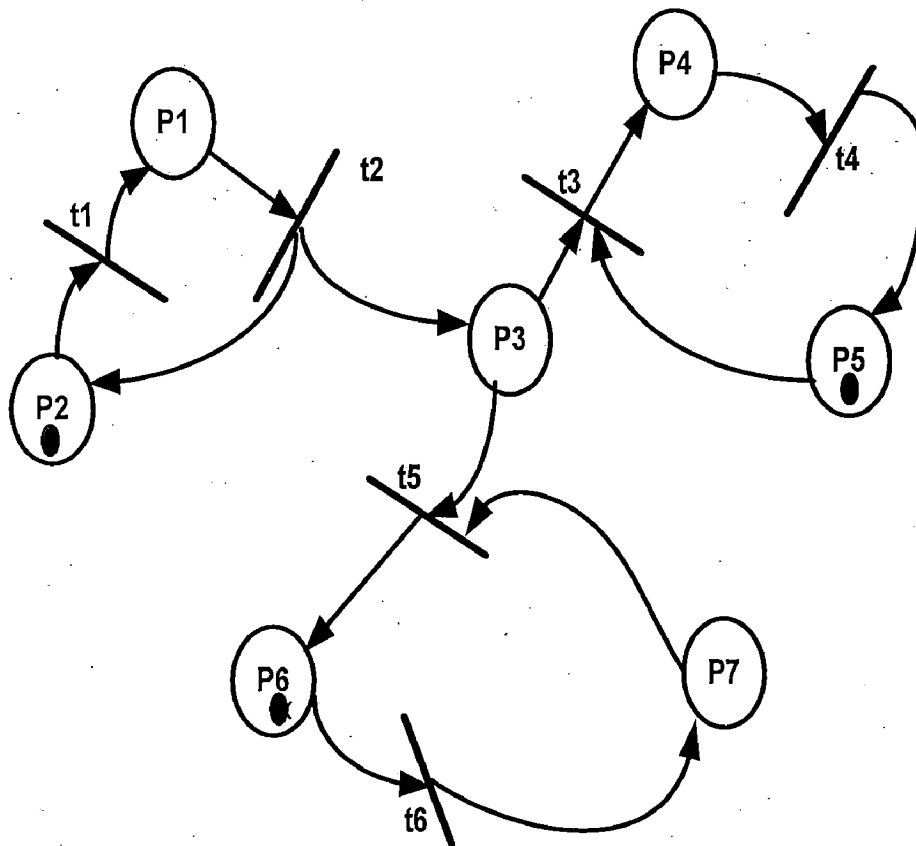


Fig 2.6 A Petri Net Graph after firing t_5

2.4: Structure of Petri Nets

The use of Petri Nets for the modeling of concurrent systems requires a good amount of knowledge and understanding about its properties. They are composed of two basic elements: a set of places P and a set of transitions T . To make the relationship more appropriate it is required to define a relationship between places and transitions. This can be done by using two functions connecting transitions to places, the input function I defines for each transition t_j , the set of input places for the transition $I(t_j)$. The output function O defines for each transition t_j , the set of output places $O(t_j)$.

Thus, these four structures model the primitive form a Petri Net. Formally a Petri Net C is defined as the four – tuple $C = (P, T, I, O)$.

Consider the following example as shown in fig 2.7 Petri Net structure, defined as a four-tuple structure.

Each component of the structure is given:

$$C = (P, T, I, O)$$

$$P = \{P1, P2, P3, P4, P5\}$$

$$T = \{t1, t2, t3, t4\}$$

$$I(t1) = \{P1\}$$

$$O(t1) = \{P2, P3, P5\}$$

$$I(t2) = \{P2, P3, P5\}$$

$$O(t2) = \{P5\}$$

$$I(t3) = \{P3\}$$

$$O(t3) = \{P4\}$$

$$I(t4) = \{P4\}$$

$$O(t4) = \{P2, P3\}$$

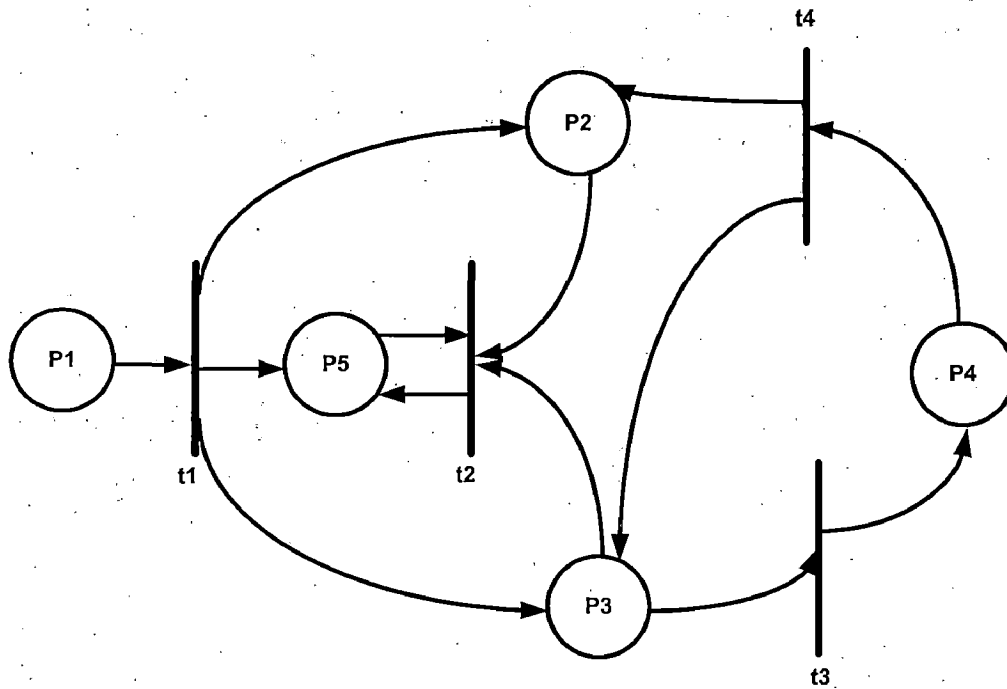


Fig 2.7 The Simple PN Graph

The similarity between Petri Net Graphs and Petri Net structures makes one to consider them as not different representations but as the same concept.[3] Hence, the term Petri Net is used to describe both PN Graphs and PN Structures.

2.4.1: Markings and Execution Rule of Petri Net

A marking μ of a Petri net is an assignment of tokens to places of that net. Tokens reside in the places of the net and the number and position of tokens in a net may change during its

execution. The vector $\mu = (\mu_1, \mu_2, \mu_3, \mu_4, \dots, \mu_n)$ depicts for each place in the Petri Net, the number of tokens in that place. The number of tokens in place P_i is $\mu_i = 1 \dots n$ also a marking function $\mu: P \rightarrow N$ from the set of places to the natural numbers, $N = \{0, 1, 2 \dots\}$ can be defined. This makes it appropriate to use the notation $\mu(P_i)$ to specify the number of tokens in place P_i . For a marking μ , $\mu(P_i) = \mu_i$. The marked Petri Net structure of figure 2.7 is shown in figure 2.8 with initial marking $\mu(p1) = (1, 0, 1, 0, 2)$.

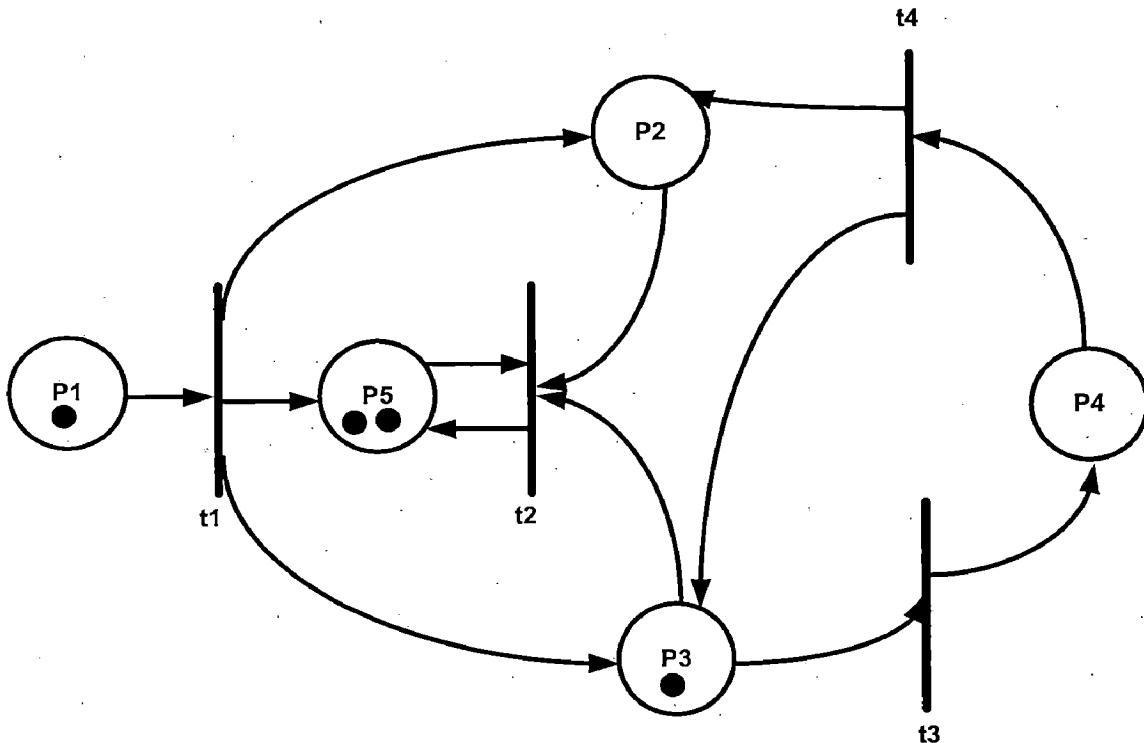


Fig 2.8 Marked PN Graph of fig 2.7 with initial marking $\mu(p1) = (1, 0, 1, 0, 2)$

A Petri net $C = (P, T, I, O)$ with a marking μ becomes the **Marked Petri Net** $M = (P, T, I, O, \mu)$. Since the number of tokens in a place is unbounded over the set of all markings, there are an infinite number of markings possible for a Petri Net. [3]

Firing of any transition will change the initial marking of Petri Net μ to μ' . It is observed that since only enabled transitions fire so the number of transitions at a place will always be non-negative even after a token is removed from it due to firing. Firing a transition can never take token from a place where there is no token in short if input places to a transition don't have any tokens the transition will never fire. The four basic rules involved in execution of transition are shown in fig 2.9.

2.4.2: The Reachability Set of a Petri Net

From a marking μ , it is possible to fire a set of transition. The result of firing a transition in a marking μ is a new marking μ' . So, it could be said that μ' is immediately reachable from μ if it is possible to fire some enabled transition in the marking μ resulting in the marking μ' . A marking μ' is reachable from μ if it is immediately reachable from μ or is reachable from any marking which is immediately reachable from μ . Thus the Reachability set $R(M)$ for a marked Petri net $M = (P, T, I, O, \mu)$ is defined as the set of all markings which can be reached from μ , so Reachability set of a marked Petri Net is the set of all states into which the Petri Net can enter by any possible execution.

2.5: Behavioral Properties of Petri Nets

A major strength of Petri Nets is their support for analysis of many properties and problems associated with concurrent systems. Those properties of concurrent systems modeled by Petri Nets which are marking dependent are called behavioral properties. Some of them are discussed in the following sections:

2.5.1: Reachability: Reachability is a fundamental property of a system that models its dynamic behaviour. With reference to our present discussion reachability in Petri Net is stated as follows a marking μ' is reachable from a marking μ_0 if there is a series of fireable transitions from μ_0 to μ' . The set of all possible markings reachable from a set μ_0 in net (N, μ_0) is denoted by $R(N, \mu_0)$.

2.5.2: Boundedness : A Petri Net is considered to be bounded or to be precise K -bounded if the number of tokens in each place does not exceed a finite number K for every marking reachable from μ_0 , i.e. $\mu(P) \leq K$ for every place P and every marking $\mu \in R(\mu_0)$. A Petri Net is said to be safe if it is 1-bounded.

2.5.3: Liveness: The concept of liveness could be related to absence of deadlock in operating system. A Petri Net is said to be live if it is possible to fire any transition of net from any new marking obtained from μ_0 by progressing through some further firing sequence.

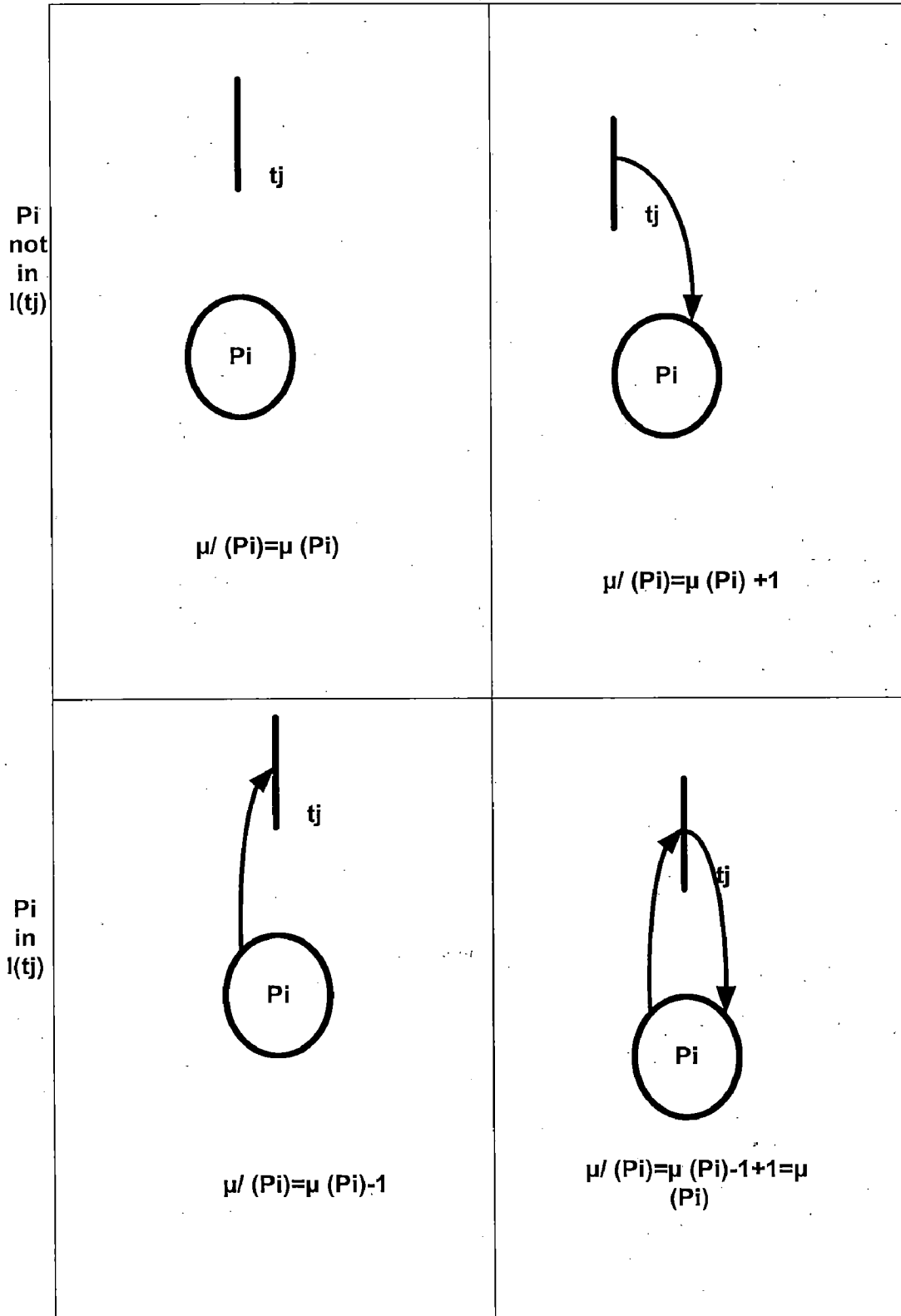


Fig 2.9 Basic Firing Rules

2.5.4: Reversibility and Home State: A Petri Net is reversible if one could reach initial marking μ_0 from a new marking μ' . In many physical applications once a system changes state hardly the system gets back to its initial state. In such systems a new state called a Home state is defined which is nothing but a marking μ' which is reachable from every making μ of $R(\mu_0)$. [20]

2.5.5: Persistence: A Petri Net is said to be persistent if for any two enabled transitions the firing of one will not disable the other. A transition in a persistent net once enabled will stay enabled until it fires.

2.6: Analysis of a PN model based on Reachability Criteria

The Reachability criteria states that given a marked Petri Net with a marking μ and μ' , is μ' reachable from μ . It is a very important problem to the analysis of Petri nets, where it is to be determined whether a set of all markings $S = \{\mu_1, \mu_2, \mu_3, \dots, \mu_k\}$ is a subset of Reachability set $R(\mu)$ of a marked Petri Net.

One of the approaches to analyze a Petri Net is to find the finite representation for the Reachability set of Petri Net as many properties of Petri Net could be studied from the properties of its Reachability set. The representation commonly known as the Reachability tree consists of a tree whose nodes represent markings of the net and the arcs represent the possible changes in the state resulting from firing of transitions.

The Reachability set of a marked Petri Net is often infinite, to make it finite many markings are mapped into the same node of the tree. This many to one mapping is accomplished by collapsing a set of nodes by ignoring the number of tokens in a place of net. This is represented by a special symbol ω for the number of tokens in this place. ω is said to represent an arbitrary large value.[3] As a result the operation of addition, subtraction and comparison is shown below:

$$\omega + n = \omega$$

$$\omega - n = \omega$$

$$n < \omega$$

For any natural number n

Each node in Reachability tree is marked with a marking and arcs are labeled with transitions. The initial node is labeled with an initial marking. For any given node x additional nodes are added to the tree for all markings that are reachable from marking of node x . For each transition t_j which is enabled in the marking of node x , a new node with marking $\delta(x, t_j)$ is created and an arc labeled t_j is directed from node x to this new node.[3] This process repeats itself until the whole state-space has been created. A marking from the root node to a node corresponds to a execution sequence. Two steps are taken to make it certain the state space if infinite models into finite Reachability tree.

First, if a new marking is obtained which is equal to an existing marking on the path from the root node to the new marking, the new (duplicate) marking becomes a terminal node. Since the new marking is equal to the previous marking, all markings reachable from it have already been added to the Reachability tree by the earlier identical marking. [3].

Second, if any new marking x is obtained which is greater than a marking y on the path from the root node to the marking x , then the components of marking x which are strictly greater than the corresponding components of marking y are replaced by the symbol ω . Since marking x is greater than marking y , any sequence of transition firings which is possible from marking y is also possible from marking x . In particular, the sequence that transformed marking y into marking x can be repeated as many number of times its possible thereby increasing the number of tokens in those places which have a ω . Thus the number of tokens in these places could have large values. [3]

The Reachability tree of Petri Net structure of fig 2.10 is as shown in fig 2.11. After discussing the basics of Reachability tree, one needs to answer the important question how will it be used to understand properties of nets? As per to safeness and boundedness it has already been stated that if a Petri Net is K (note $K=1$ for safeness) bounded then no more than K tokens can reside in any place. Now if the Reachability tree of a Petri Net contains ω then it violates both the condition of both boundedness and safeness. [3]

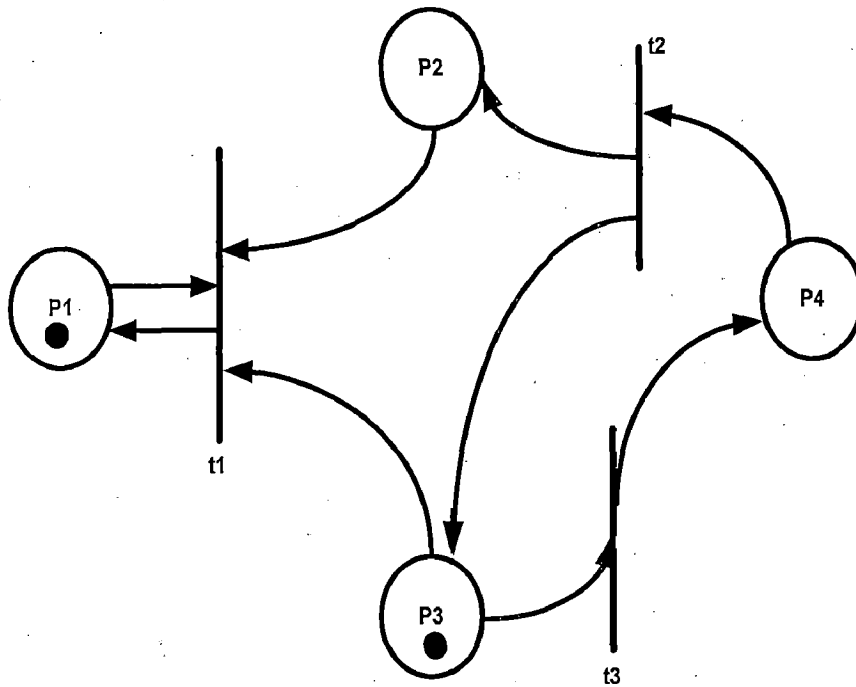


Fig 2.10 A Petri Net with initial marking $(1, 0, 1, 0)$ and infinite reachable state space

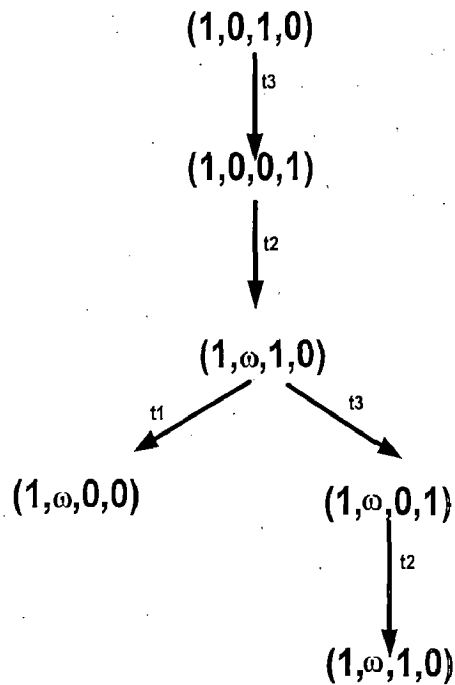


Fig 2.11 The Reachability Tree of Petri Net of Fig 2.10

2.7: Petri Nets and Industrial Automation

The ladder logic diagrams (LLD) have been widely used in manufacturing systems to design sequence controllers and is generally implemented on a PLC.

The I/O procedures of the PLC are specified by ladder logic diagrams (LLD) which makes industrial machines run repetitive operations easily. For most simple systems it is easy to program the PLC by heuristic methods, but modern manufacturing systems are more complex as a result the design of corresponding sequencers by LLD alone is increasingly difficult. Further, the usage of LLD is limited to control of the system rather than in analysis and evaluation of qualitative and performance characteristics.

Both finite state machines (FSM) and Petri nets (PN) effectively model sequential system but when it comes to complexity FSMs fail to model systems with increasing complexity since, they require the knowledge of all the states (global states) of the system. PN uses the local states while modeling a system and hence knowledge of all system states is not a prerequisite. The need for only the knowledge of local states while designing a PN based strategy makes them easily expandable i.e. addition of new components to the system only affects a PN's model locally, unlike in FSM's where it might affect every state in the model. Infact, it is possible to model a FSM using PN.

Petri Nets are capable of modeling sequential, asynchronous and concurrent events that drive an industrial process. Using PN's strategies are developed that are able to capture the discrete event dynamics of the process. The control strategies so developed are easier to understand, troubleshoot and modify and gives better evaluation of system performance.

The use of Petri Nets for modeling and analysis has received great importance because of several reasons

- 1) Graphical form of PN is easy to understand and has direct correspondence to discrete event control characteristics of industrial control systems especially in the field of manufacturing.
- 2) Petri Nets have a sound mathematical foundation and hence are amenable to the analysis for logical consistency, such as boundedness, liveness and Reachability.
- 3) The definition of PN makes quantitative performance evaluation possible.

- 4) PN can be employed to do modular designs because it exhibits natural decomposition properties.
- 5) PN makes it possible to translate itself into control code ready to be implemented on plant floor for desired execution.

One of the strategies for design of Industrial Sequencers could be to model the system using PN and then to analyze the model for its validity in terms of liveness, boundedness and reversibility and finally to implement the net structure by transforming it into equivalent LLD code for the sequencer.

2.8: The Real Time Petri Net Algorithm (RTPN)

A Real-Time Petri Net (RTPN) based controllers can be obtained by assigning physical (I/O) input/output functions to places and physical (I/O) and timing variables to transitions of the PN model.

A RTPN is eight tuple and is defined as

RTPN: (P, T, I, O, m, D, Y, Z) where

- ✦ (P, T, I, O, m) are the properties of the untimed Petri Net model.
- ✦ D is a firing time delay function, consisting of non-negative real numbers.
- ✦ Y is defined as the set of physical input signal functions mapped to transitions.
- ✦ Z is defined as the set of physical output signal functions mapped to places. [14]

- 1) Timing vector D assigns time delays to transitions. Timing vector D models the delays and synchronization of activities in the system.
- 2) Vector Y is used as an enable signal and determines when a transition is fired. Vector Y can be mapped to a single input address or can be Boolean expression of input addresses. When the function associated with Y (i) is true and all input places are marked, then the firing rule is executed i.e. tokens are removed from input places and deposited to output places. Vector Y is the firing attribute of all transitions in RTPN.
- 3) Vector Z write commands to the digital output interfaces. When a place P_i is marked by a token, then some output function occurs.

RTPN Algorithm has been used for obtaining Petri Net based sequence Controllers for all the process control applications discussed in this dissertation. The places of such sequence

controllers are assigned with actuator outputs; they remain valid as long as the places are active. The inputs are mapped to the transitions and they act as second condition for firing of transitions, i.e. if an input place to a transition is marked along with it being activated by its I/P sensors it fires.

The delay function is used along with transition as a third condition to delay firing of a transition if it is required in application.

2.9: Conclusion

In this chapter the basics of Petri Nets have been discussed along with the importance of Petri Nets in the field of Industrial Automation. Finally, a RTPN algorithm is given which have been extensively used for design of PN based sequence controllers for all the applications considered in this dissertation study. The popularity of Petri Nets for modeling system could be attributed to following factors

- 1) The same system model designed using PN could be used for analysis and performance evaluation of the system
- 2) Petri Nets are super class of state machines and unlike state machine could model system with concurrency, conflict and synchronization thereby making them as an obvious tool for modeling manufacturing systems.
- 3) Petri Nets as a graphical tool acts as a powerful medium between engineer and customer and its strong mathematical background enables it to do formal checking on behavior of modeled system.

Next chapter introduces the basics of Programmable Logic Controllers along with the discussion of standardized languages used for PLC programming.

CHAPTER 3

PROGRAMMABLE LOGIC CONTROLLERS

3.1: Introduction

The PLC's can be classified as a solid state member of computer family. A PLC is an industrial computer in which control devices such as limit switches, push buttons, proximity or photoelectric sensors, float switches or pressure switches, to name a few, provide incoming control signals into the unit. [9] [24]

Incoming control signals termed as inputs interacts with instructions specified in user ladder program, which tells the PLC how to react to incoming signals. The user program also directs the PLC on how to control field devices like motor starters, pilot lights and solenoids. A signal going out of PLC to control a field device is called an output. [8]

A formal definition of PLC comes from the National Electrical Manufacturers Association (NEMA):

“A Programmable controller is a digitally operated electronic system, designed for use in an industrial environment and uses a programmable memory for the internal storage of user oriented instructions for implementing specific functions such as logic, sequencing, timing, counting and arithmetic to control through digital or analog inputs and outputs, various types of machines or processes. Both the PLC and its associated peripherals are designed so that they can be easily integrated into an industrial control system and easily used in all their intended applications.” [5]

3.2: Evolution of PLC

The PLC was originally designed and developed by group of engineers of General Motors Corporation in 1968 to eliminate costly scrapping of assembly line relays during model changeover of cars. These PLC's had to be easily designed, programmed and reprogrammed, preferably in a plant, easily maintained and repaired, smaller than its relay equivalent and cost competitive with the solid state and relay panels then in use. This provoked great

interest from engineers of all disciplines using PLC for the industrial control. A microprocessor based PLC was introduced in 1977 by Allen-Bradley Corporation in USA, using INTEL 8080 microprocessor with circuitry to handle bit logic instructions at higher speeds. The early PLC's were designed using only for logic based sequencing operations (on/off signals).

Present day sees quite a lot of different brands in market differing in memory size and I/O capacity. The smaller ones serve as relay replacers with added timer and counter capability. The modern medium sized PLCs perform all the relay replacement functions expected of it but also add many other functions such as counting, timing and complex mathematical applications. In addition, new age medium and large size PLCs have been provided with data highway capabilities which enable them to function in liaison with Distributed Control System. [8]

3.3:PLC Architecture

A Programmable Logic Controller (PLC) manufactured by any company has several common functional units. The basic PLC architecture is as shown in fig 3.1.

The PLC architecture of fig 3.1 consists of

- a) Power supply
- b) Input/output (I/O) system
- c) Real-time central processing unit
- d) Memory unit
- e) Programmer unit

a) Power Supply

The power supply unit provides the isolation unit necessary to protect solid-state components from most high voltage line spikes. The power supply unit converts power line voltages to those required by the solid state components. In addition, the power supply is rated for heat dissipation requirements for plant floor operation. This dissipation capability allows PLC's to have high ambient temperature specifications and represent an important difference between programmable logic controllers and personnel computer for industrial applications.

The power supply unit drives the I/O logic signals, the central processing unit, the memory unit and some peripheral devices.

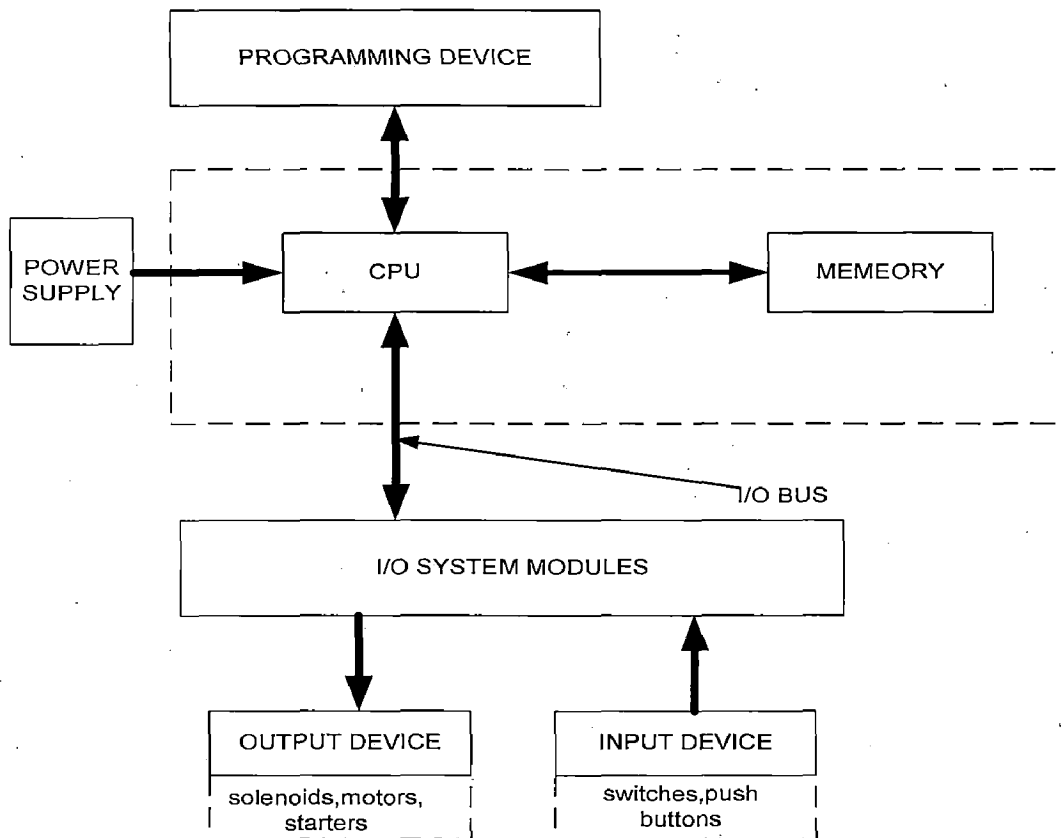


Fig 3.1 Architecture of a Programmable Logic Controller

b) Input/Output (I/O) System

Inputs are defined as real-world signals giving the controller real time status of process variables. These signals can be analog or digital, low or high frequency, maintained or momentary. Typically they are presented to PLC system as a varying voltage, current or resistance value. Signals from Thermocouples and RTD's are common examples of analog signals. Some flow meters and strain gauges provide variable frequency signals, while pushbutton, limit switches or even electromechanical relay contacts are examples of digital, contact closure type signals. Register input is another type of input signal that reflects the computer nature of PLC. The register input is only useful when the process condition is

represented by a collection of digital signals delivered to the PLC at the same time. There are three common categories of outputs: discrete, register and analog. The discrete outputs can be pilot lights, solenoid valves or annunciator windows. Register outputs can drive panel meters or displays. Analog outputs can drive signals to variable speed drives or to I/P (current to air) and thus to control valves.

c) Real-time Central Processing Unit

The Central processing unit (CPU) also called central control unit (CCU) performs the task necessary to fulfill the PLC function such scanning I/O bus traffic control program execution, peripheral and external device communications, special functions or data handling execution (enhancements), and self diagnostics.

d) Memory Unit

The memory unit of PLC serves several functions. It is the library, where the application program is stored. It is also where the PLC's executive program is stored. An executive table functions as the operating system of PLC. It is the program that interprets, manages and executes the user's application program. Finally, the memory module is the part of the programmable controller, where the process data from the input modules and control data from output modules are temporarily stored as data tables. Typically, an image of these data tables is used by the CPU and when appropriate is sent to output modules.

e) Programmer units

The programmer unit provides an interface between the PLC and the user during program development, start-up and troubleshooting. The instructions to be performed during each scan are coded and inserted into memory with the programmer unit. The programmer unit may vary from small hand held units to desktop stand-alone intelligent CRT screens. Programming units are the liaison between what the PLC understands and what the engineer desires to occur during the control sequence.

3.4: Benefits of PLC

Table 3.1: Advantages of PLC Programming

Features	Benefits
Solid state components	High reliability
Programmable memory	1) Flexible control 2) Simplifies changes
Microprocessor based	1) Communication capability 2) Multi-functional capability 3) High quality products 4) High level performance
Small size	Minimum space required
Software timers/counters	1) Eliminate hardware 2) Easily changed presets
Software control relays	1) Reduced cost 2) Reduced space requirements
Modular I/O interface	1) Easily maintained 2) Easily connected

3.5: PROGRAMMING LANGUAGES

As programmable controller manufacturers increased, so did the number of programming languages. The most disturbing fact was each manufacturer had its own language and protocol for using its PLC hardware; some were easier than other which left industry users with no choice of selecting a common platform.

IEC 1131-3 programming languages was introduced to take care of this menace and it addressed the problem successfully by establishing four interlinking languages to work together under the supervision of the fifth one. In addition it made the languages portable so that they could be used by any programmer, technician or PLC operator. The language for PLC Programming consists of two textual and two graphical versions. [6]

They are

GRAPHICAL

Ladder Diagram (LD)

Functional Block Diagram (FBD)

TEXTUAL

Instruction List (IL)

Structured Text (ST)

The choice of programming language is dependent on:

- ..The programmers' background.
- ..The problem at hand.
- ..The level of describing the problem.
- ..The structure of the control system.
- The interface to other people/departments. [10]

3.5.1 : GRAPHICAL LANGUAGES

They are employed as one of the most recognizable language type when one is programming PLC's. As engineers make the first hand drawn sketch of their plans, they actually provide the rough draft of what will finally be a detailed control scheme.

3.5.1.1 : LADDER DIAGRAM (LD): Ladder Diagram has its roots in the USA. It is based on the graphical presentation of Relay Ladder Logic. They are the probably the first thing to come to mind in a discussion of a PLC. Much credit goes to the electrical and electronics engineers for building a standardized set of symbols to represent the electrical sequence of an applications operation. The working of a ladder diagram is explained in fig 3.2. All equipments and devices used in the process or machine have an on/off state. Therefore, a ladder diagram can activate any of these devices based upon the state, or condition of another. All this is done in some predetermined sequence, from top-down, left-to-right running of the ladder diagram. This process continues till the end and then repeats itself, constantly checking the status of each contact as it scans the diagram. In fig 3.2 if A is on and B is off then LED is on, where A and B are Relays and LED ON represents the output coil.

3.5.1.2 : FUNCTIONAL BLOCK DIAGRAM (FBD): Function Block Diagram is very common to the process industry. It expresses the behavior of functions, function blocks and programs as a set of interconnected graphical blocks, like in electronic circuit diagrams. It looks at a system in terms of the flow of signals between processing elements. Program elements appear as blocks on the screen and are then wired together. The shell of the block receives inputs and sends output information all the while processing the I/O as necessary. In this way, the program elements can execute their respective algorithms and operations in the manner the wires dictate. The fig 3.3 shows the Functional Block Diagram of a PID Controller for Motor Control.

3.5.2: TEXTUAL LANGUAGES

As the name implies, textual languages do not use symbols or graphical representation to build and execute a program. Rather the code has to be developed line by line. The form may have high or low level character set, or byte-per-character form. This makes its usefulness less acceptable to operators and industry work force since the textual language has to relate to PLC devices and the person equipped with such knowledge is very few in number.

3.5.2.1 : INSTRUCTION LIST : Instruction List is the European counterpart of LD. As a form of textual language, it resembles assembler programming. It performs one operation per line, with the advantage that processing lines of code is much faster as it is in assembler form. Being in assembly language form its acceptance is limited in automation industry, but IL can be utilized for programming less complicated processes and machines as long as they could be interpreted without much effort. [6] Fig 3.4 gives an example of code written in IL.

In general, IL should not be used because this language:

- i) Provides no code structuring
- ii) Has weak semantics
- iii) Is machine dependent. [23]

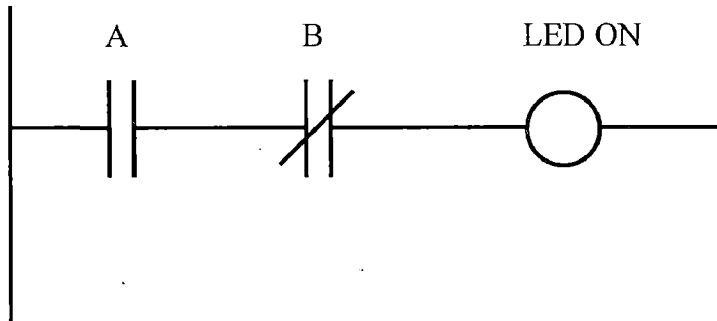


Fig 3.2 An example of ladder Logic.

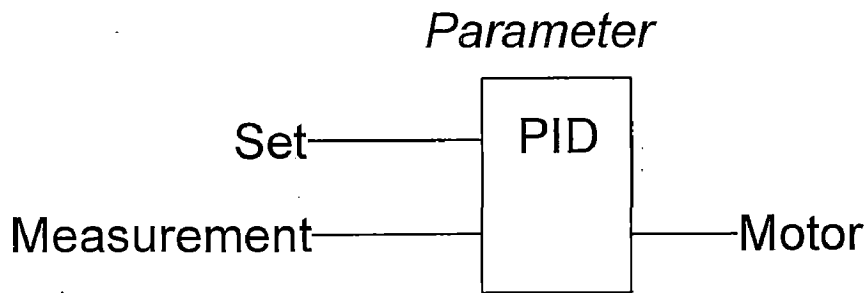


Fig 3.3 An example of FBD

Label	Operator	Operand	comment
	LD	Temp1	Load temp1 and
	GT	Temp2	Test if temp1>temp2
	JMPCN	Greater	Jump if not true to greater
	LD	Speed1	Load speed1
	ADD	200	Add constant 200
Greater	LD	Speed2	Load speed2

FIG 3.4 An example of IL Program

3.5.2.2 : STRUCTURED TEXT: Structured Text is a very powerful high-level language with its roots in Ada, Pascal and “C”. It contains all the essential elements of a modern programming language, including selection branches (IF-THEN-ELSE and CASE OF) and iteration loops (FOR, WHILE and REPEAT). These elements can also be nested. It can be used excellently for the definition of complex function blocks, which can be used within any of the other languages.[6] A Sample program in structured text is given in fig 3.5 .

```

I:=25;
WHILE J<5 DO
Z:= F (I+J);
END _WHILE
IF B_1 THEN
%QW100:= INT_TO _BCD (DISPLAY)
END IF
CASE TW OF
1.5: TEMP:=TEMP_1;
2: TEMP:=40;
4: TEMP:=FTMP( TEMP_2);
ELSE
TEMP:=0;
B-ERROR:=1;
END CASE;

```

Fig 3.5 An example of ST Program

3.6: Conclusion

In this chapter the hardware architecture of PLC and the programming languages used for coding them have been discussed. Though, PLC's offer various advantages as far as its architecture are considered, the difficulty is with the programming languages used to code them.

In this dissertation Ladder Logic Diagrams have been used to code the sequencer applications on PLC and it is observed that LLD code become more complex and difficult to analyze and debug with increasing complexity of applications. To quantify Petri Net based sequence controller designs over LLD based designs for similar process applications one needs to identify criteria's to compare both the algorithms. Details of criteria used to compare PN and LLD based sequence controller design are discussed in chapter 4.

CHAPTER 4

Evaluation Criteria

4.1: Introduction

The increasing complexity of control specification for design of present day flexible automation systems are challenging the use of ladder logic diagrams and higher programming languages. [27]

At Present, Petri Nets are increasingly used to design sequence controllers. In order to establish PN as a better alternative to LLD based design there is a need to justify the PN method. This can partially be accomplished by comparing PN and LLD based designs. Two of the most important factors for comparison of PN and LLD for discrete event control are

- a) Design Complexity
- b) Response Time.

Since, there are several ways some of which are shown in fig 4.1 to implement PN's and LLD's both in terms of hardware and software, hence, it is difficult to obtain a fair comparison solely in terms of response time criterion.

Table 4.1: Hardware platforms used

Figure	Hardware Used
a	Intel 8085 Assembler
b	A PLC based on ZILOG 80A microprocessor
c	16 bit microcomputer
d	VAX 11/780 and IBM- PC/XT
e	IBM- PC/XT
f	IBM- PC/XT
g	8086 CPU and DSP (TMS 32020)
h	IBM-PC/AT

Table 4.1 gives a few of the hardware platform used till now, for implementation of different schemes of PN Controller for sequence controller as shown in Fig 4.1 [14].

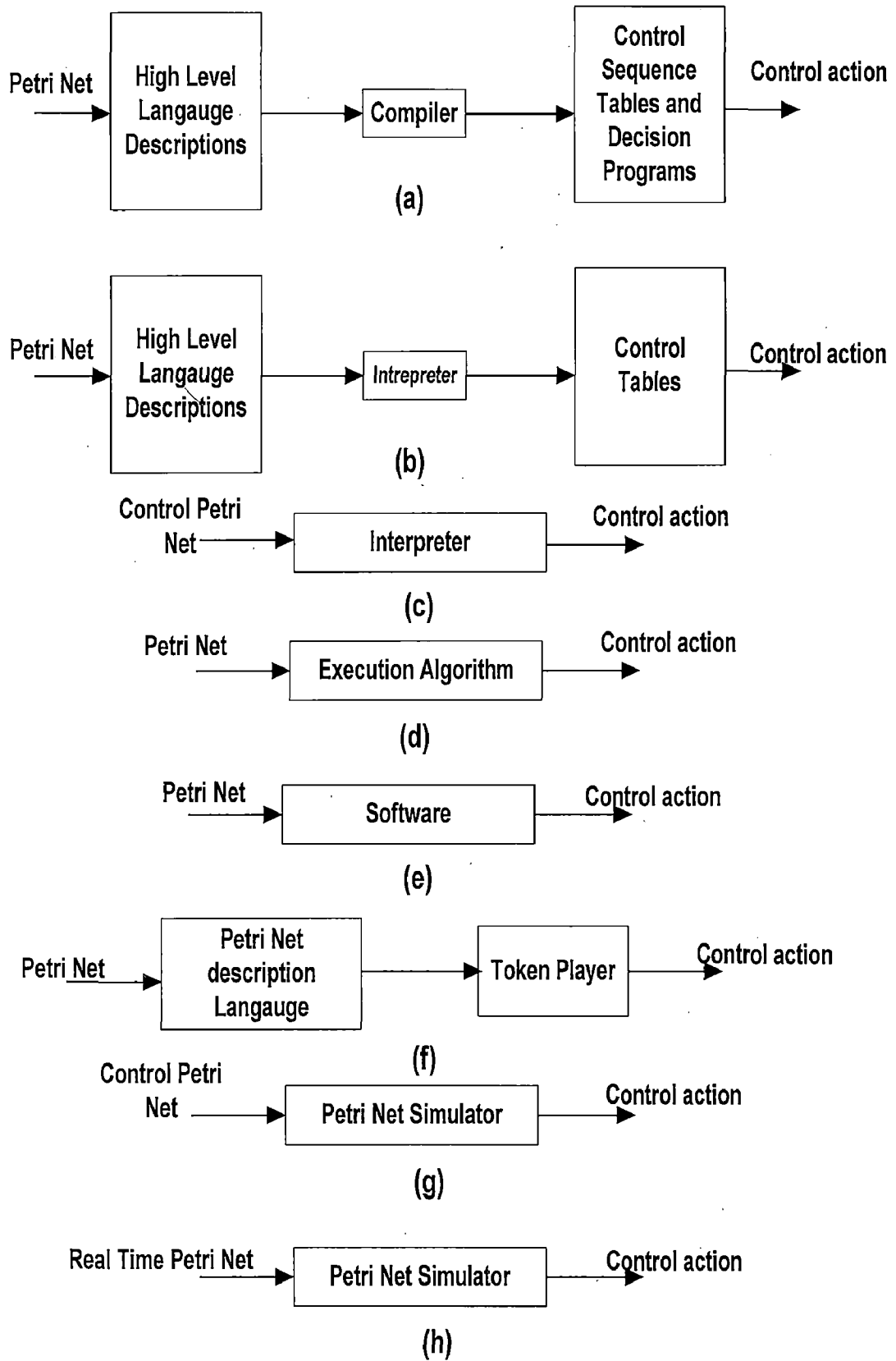


Fig 4.1 Various methods of Petri Net based Sequence Control

4.2: Criteria for Evaluation

4.2.1: Design Complexity

Design Complexity is stated as the complexity associated with the design of control logic for a given specification.

Now, design complexity is affected by many factors such as the experience of designers, size of control program, number of dynamic steps necessary for coding or changing the control program are a few amongst all, hence it is hard to quantify formally. However, design complexity can be characterized by two factors

- 1) Graphical Complexity
- 2) Adaptability for change in Specification. [14]

4.2.1.1: Graphical Complexity

It is mainly analyzed based on the number of nodes and links for a given graphical logic control design. It affects the understandability of control logic for people who don't have any prior understanding of PN or LLD. Hence, it is an important criterion in designing the logic initially and finally in debugging errors during implementation phase. The Graphical complexity in terms of net size is a major criteria in manufacturing systems and it is proven that the simpler the graphical complexity it becomes more easy to tract the controller [14].

Graphical Complexity has also its influence on the response time. For example, in the case of LLD's response time depends on size of LLD algorithm, hence a short LLD gives rise to a fast controller [14].

In case of PN's nodes are places (that represent preconditions and postconditions) and transitions (that model operations) and links are arcs. Whereas, in LLD's nodes are pushbuttons, normally opened/closed switches/ contacts, timers, counters, relays and solenoids, while, links are connections among nodes. The control logic representation by Petri Nets and Ladder Logic Diagrams are shown in fig 4.2 whereas the basic elements in LLD and PN are shown in fig 4.3.

4.2.1.2: Adaptability for change in Specifications

In recent years, this factor has gained due importance in the field of agile manufacturing where control sequences need to be changed very frequently in order to meet the changing requirements of market [14].

In short, the control software should be easily adaptable to change in specification so as to improve software productivity and further to keep minimum development time. A design is stated to be more adaptable if it needs fewer changes in order to meet a change in specification.





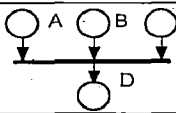
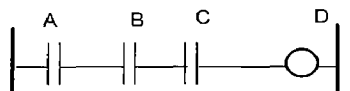
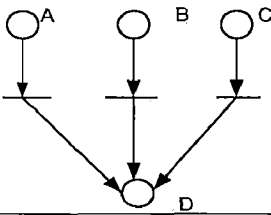
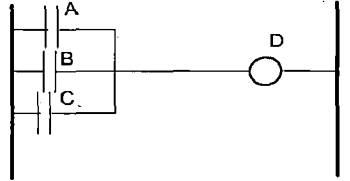
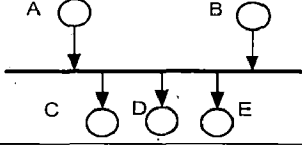
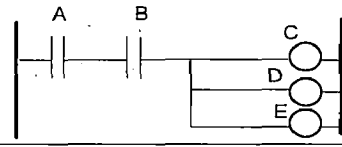
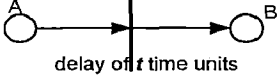
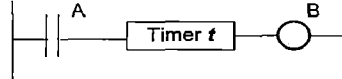
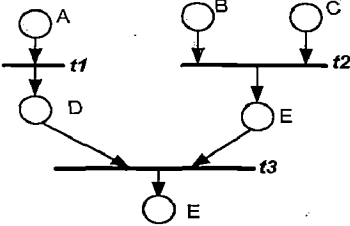
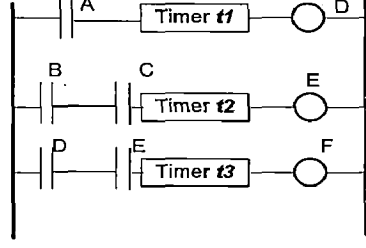
Logic Constructs	Petri Nets	Ladder Logic Diagrams
Condition or status of a system element	 Place	No explicit representation
An activity	 Transitions	No explicit representation
Flow of information or material	 Directed Arcs	No explicit representation
Objects such as machines, robots, pallets etc	 Token(s) in Place(s)	No explicit representation
Logical AND if A=1 and B=1 and C=1 then D=1		
Logical OR if A=1 or B=1 or C=1 then D=1		
Concurrency if A=1 and B=1 then C=1 and D=1 and E=1		
Time Delay if A=1 then after t time delay B=1		
Synchronization if A=1 then delay t1 time units: D=1 if B=1 and C=1 then delay t2 units: E=1 if D=1 and E=1 then delay t3 time units: F=1		

Fig 4.2 Control logic representation in PN and LLD

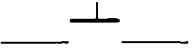

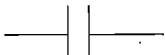
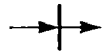







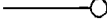
Basic Elements	LLD		PN	
Nodes	Push buttons		Place	
	Normally open Contact/switch		Transition	
	Normally closed Contact/switch			
	Relay Coil			
	Timer			
	Counter			
	Solenoid			
Links	Line		Normal Arc	
			Inhibitor Arc	

Fig 4.3 Basic Elements representation in LLD and PN

4.2.2: Response Time

Response time is termed as scan time in LLD literature and execution time in PN literature. The response time with respect to a control system means how fast a system under control can respond to an event in the system/ process under control. Unlike graphical complexity and adaptability for change in specification the response time is not only affected by physical appearance but also by the method of implementation [14].

Method of implementation corresponds to the hardware and software used to control the system using either PN or LLD. Now, graphical complexity and adaptability cannot be quantified whereas, response time can be measured accurately given a logic design and implementation. Since, there are several ways to implement both PN and LLD in terms of hardware and software it is difficult to compare both in terms of response time criteria. Therefore, a common measure to obtain information about graphical complexity, adaptability and response time is the number of nodes and links used to design control logic. If more nodes and links are used in a code it is said to be graphically more complex and thus may require more response time. [14]

Hence, number of basic elements decides the length of the control model and a smaller model with smaller number of basic elements in terms of nodes and links are easier to understand, check, diagnose and maintain.

4.3: Rule Based Comparison

In rule based comparison technique both LLD and PN based design algorithms are converted into the IF-THEN formats and a comparison is then achieved based on

- a) Sum of the number of IF-THEN rules.
- b) The number of logical operators required to obtain both the logic developed in PN and LLD. [13]

Compound IF-THEN rules which include both conjunctive and disjunctive connective in antecedent and conclusion part can be categorized into the four following basic rules

TYPE 1: IF (A AND B) THEN C, or expressed as $(A \cap B) \rightarrow C$.

TYPE 2: IF A THEN (C AND D) or expressed as $A \rightarrow (C \cap D)$.

TYPE 3: IF (A OR B) THEN C, or expressed as $(A \cup B) \rightarrow C$.

TYPE 4: IF A THEN (C OR D), or expressed as $A \rightarrow (C \cup D)$.

The Type 2 rule can be broken into two simple rules $A \rightarrow C$ and $A \rightarrow D$. Similarly, the Type 3 rule is equivalent to two simple rules $A \rightarrow C$ and $B \rightarrow C$ because truth of either A or B or both implies the truth of C. Since, the Type 4 rule doesn't achieve the specific implication and often causes conflict problems, it is generally not suitable for real applications in sequence

control. The IF-THEN rules excluding the Type 4 rule for both LLD and PN transformations are shown in Fig 4.4:

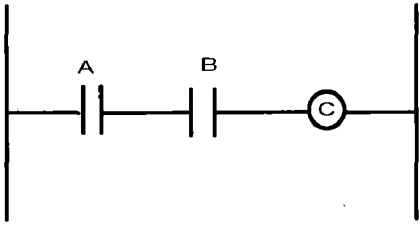
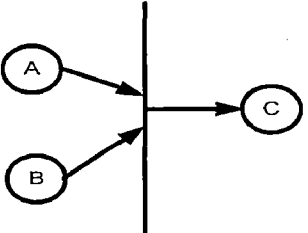
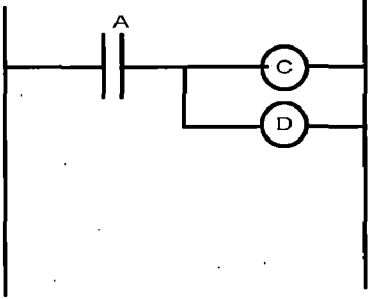
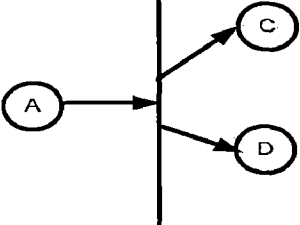
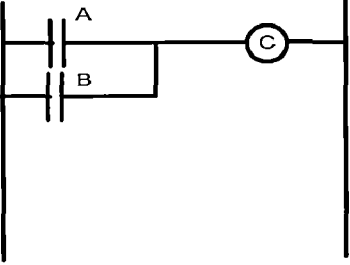
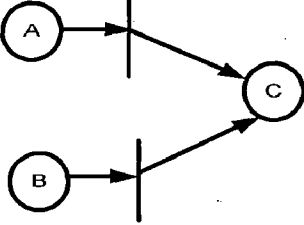
IF-THEN RULES	LLD	PN
IF A and B THEN C		
IF A, THEN C and D		
IF A or B THEN C		

Fig 4.4 IF-THEN rules for LLD and PN

Based on the IF-THEN rules two measures are derived to further evaluate LLD and PN

- 1) The number of IF-THEN rules.
- 2) The number of logical operators, including the AND / OR block.

The summation of the above two measures could be used as another evaluation parameter for comparison, as a model with less number of IF-THEN rules and operators are easier to understand, debug , check and maintain, it also have shorter response time. [15]

4.4: Example

A bottle filling system shown in fig 4.5 is considered for comparison of LLD and PN based designs. First of all a sequence controller based on informal specifications of bottle filling system is implemented directly using ladder logic diagrams. Next, the informal specifications are formalized and a PN based sequence controller is implemented for the same application. Finally, both the sequence controllers are compared in terms of the evaluation criteria discussed in preceding sections.

The main objective of the bottle filling system as shown in Fig 4.5 below is to fill bottles moving on a conveyor belt and the informal specifications of this system are detailed in two phases as given below

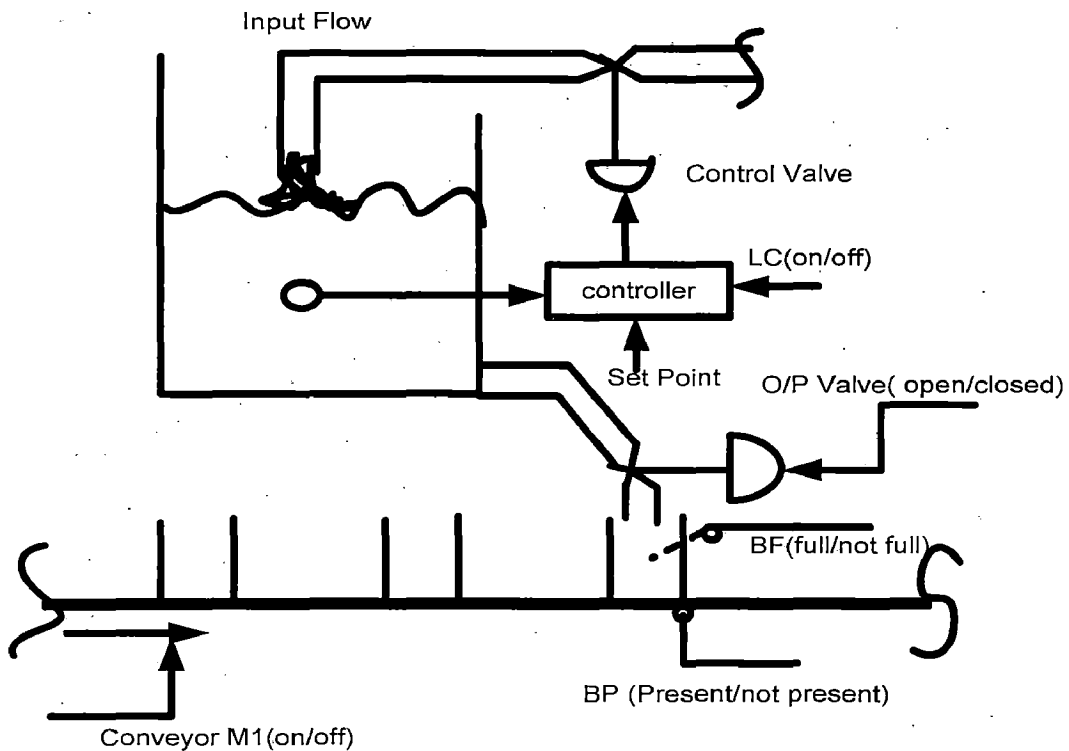


Fig 4.5 Bottle Filling System

1) INITIAL PHASE (Prefill the tank)

- a) Conveyor is stopped.
- b) The output valve is closed.
- c) Start the level control system and operate for prespecified time to fill the bottle to desired level.
- d) When the level is reached stop the level control.

e) Goto running phase.

2) RUNNING PHASE

a) Start the bottle conveyor.

b) When a bottle is in position

1) Stop the conveyor.

2) Open the output valve.

3) Turn on the level control system to keep the level constant during bottle fill.

c) When the bottle is full,

1) Close the output valve.

2) Stop the level control system.

d) Goto step a and repeat the sequence. [7]

The details of the formalization procedure discussed already in chapter 1 wrt example 4.5 are as follows:

A) Informal Specifications

1) **Problem Specific functional Properties:** Some of the problem specific functional properties for example 4.5 are listed below

a) When the bottle is full,

1) Close the output valve.

2) Stop the level control system

b) When a bottle is in position

1) Stop the conveyor.

2) Open the output valve.

2) **Standard functional Property:** With reference to example 4.5 the standard functional property is the sequence controller should be tractable i.e. under no circumstance it should enter into deadlock and neither two disjoint states should operate concurrently. Moreover, no outputs signals should be in conflict i.e. in each state of the sequence controller an unambiguous value of each and every o/p should be stated.

3) **Non-functional Property:** The algorithm should be well structured and it should be well documented with comments to facilitate clarity.

B) Formal Specifications

In order to obtain a formal algorithm of the sequence controller for bottle filling system, a formalizing tool is used. Further, the informal specifications are formalized to meet the formal controller requirements.

1) Formalization of the Algorithm: For the formalization of control algorithm a RTPN is used, where

- i) Every transition is associated with a Boolean function of input signals and time delays (if any) and constitutes the firing condition.
- ii) Every place is associated with an o/p function, which assigns values to subsets of o/p signals while it is marked.

The dynamic behavior of a RTPN is given by flow of tokens in net and is governed by the following four rules [36]

- i. A transition is enabled if all its pre-places are marked and all its post-places are not marked.
- ii. A transition fires as soon as it gets enabled and its firing condition exists.
- iii. All transitions which are enabled and are not in conflict fire simultaneously.
- iv. The firing process continues until a stable marking is reached.
- v. After a stable marking has been obtained o/p's corresponding to that place are enabled whereas, other o/p's corresponding to other places are deactivated.

2) Formalization of Specific Properties: The Problem specific properties are formalized using Boolean logic i.e. sequence controller is converted into RTPN algorithm with transitions having Boolean equations as firing rule.

3) Formalization of standard Properties: The standard properties such as O/P conditions at stable markings are formalized by assigning ones or zeroes to o/p variables in each marking.

4) Formalization of non-functional properties: The non-functional property is number of comments to enhance understanding and is quantified as

Comments: $\#comments/\#places+\#transitions$.

An algorithm is said to be more clear if it is associated with more number of comments, which makes it more easier to understand.

C) Verification and Validation

In this dissertation the RTPN structure of sequence controller for bottle filling system is developed using VHDL and is implemented in real-time on Altera Cyclone family of FPGA. The structure obtained is checked for all possible combinations of I/P conditions by means of simulation. Further, a reachability tree is drawn manually to search for any deadlock present in sequence controller of bottle filling system.

D) Implementation and Realization

The algorithm is implemented using VHDL and realized on Altera cyclone family FPGA kit.[11] [12] [27] [33]

Table 4.2: The I/O specification for the RTPPN based sequence controller of bottle filling system.

Switch	Status	Description
Resetrn	i/p	It is used to reset the system in case of deadlocks.
en	i/p	It is used as an enabling signal for starting the timer for level control system
start	i/p	It is used as an input to start the initialization phase.
bpin	i/p	External push button used to emulate the condition of bottle present.
bfin	i/p	External push button used to emulate the condition of bottle full.
lc	o/p	LED indicator used to emulate the signal to start the level control system.
cm	o/p	LED indicator used to emulate the signal that turns the conveyor motor on.
bp	o/p	LED indicator used to emulate the signal that turns the o/p valve open and also to show the Bottle present indicator.
bf	o/p	LED indicator used to emulate the signal to show the Bottle full indicator.

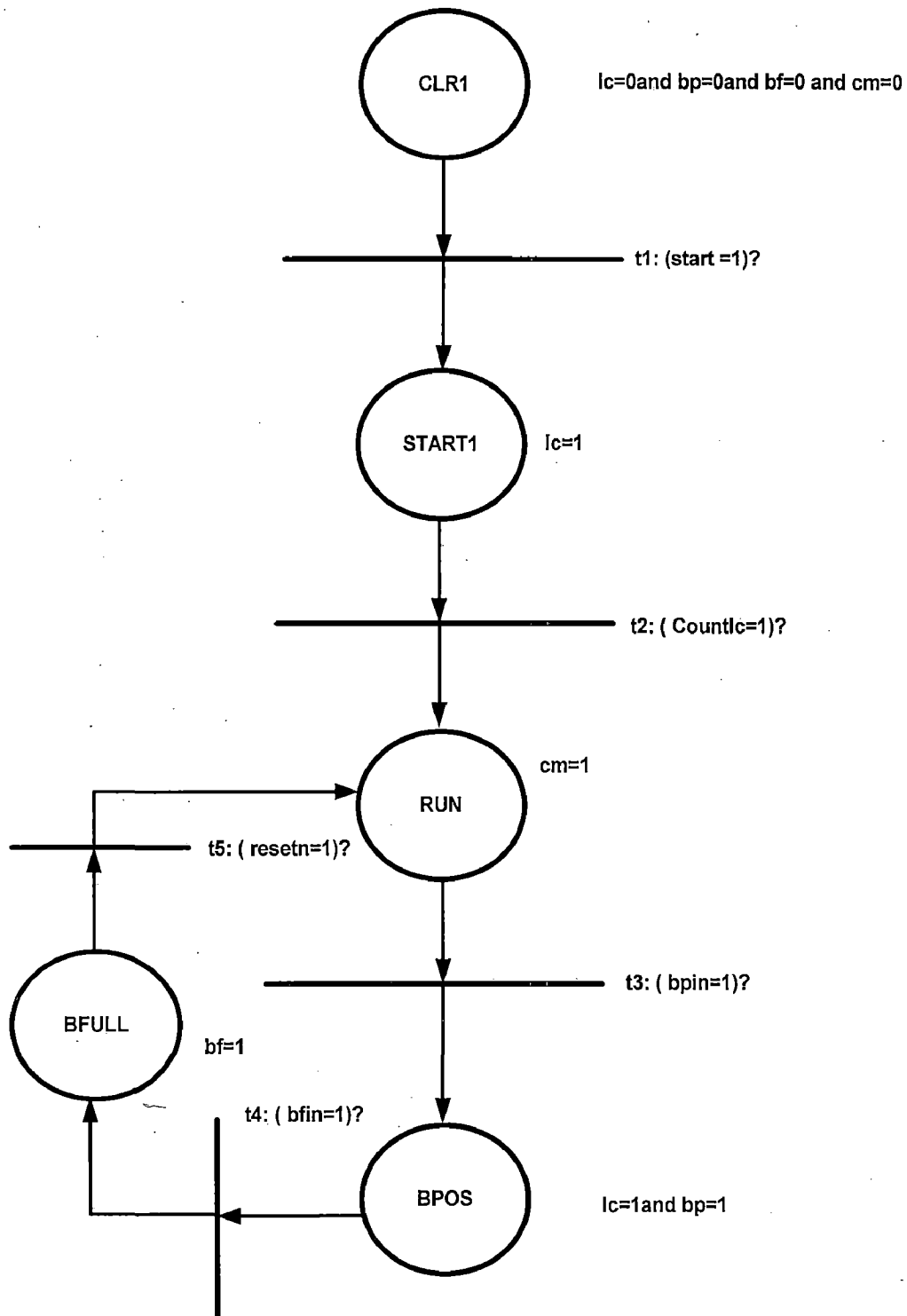


Fig 4.6 The Petri Net structure of the sequence controller for bottle filling system

The logic diagram of the sequence controller for the bottle filling system as derived from LLD Algorithm is presented in fig 4.7

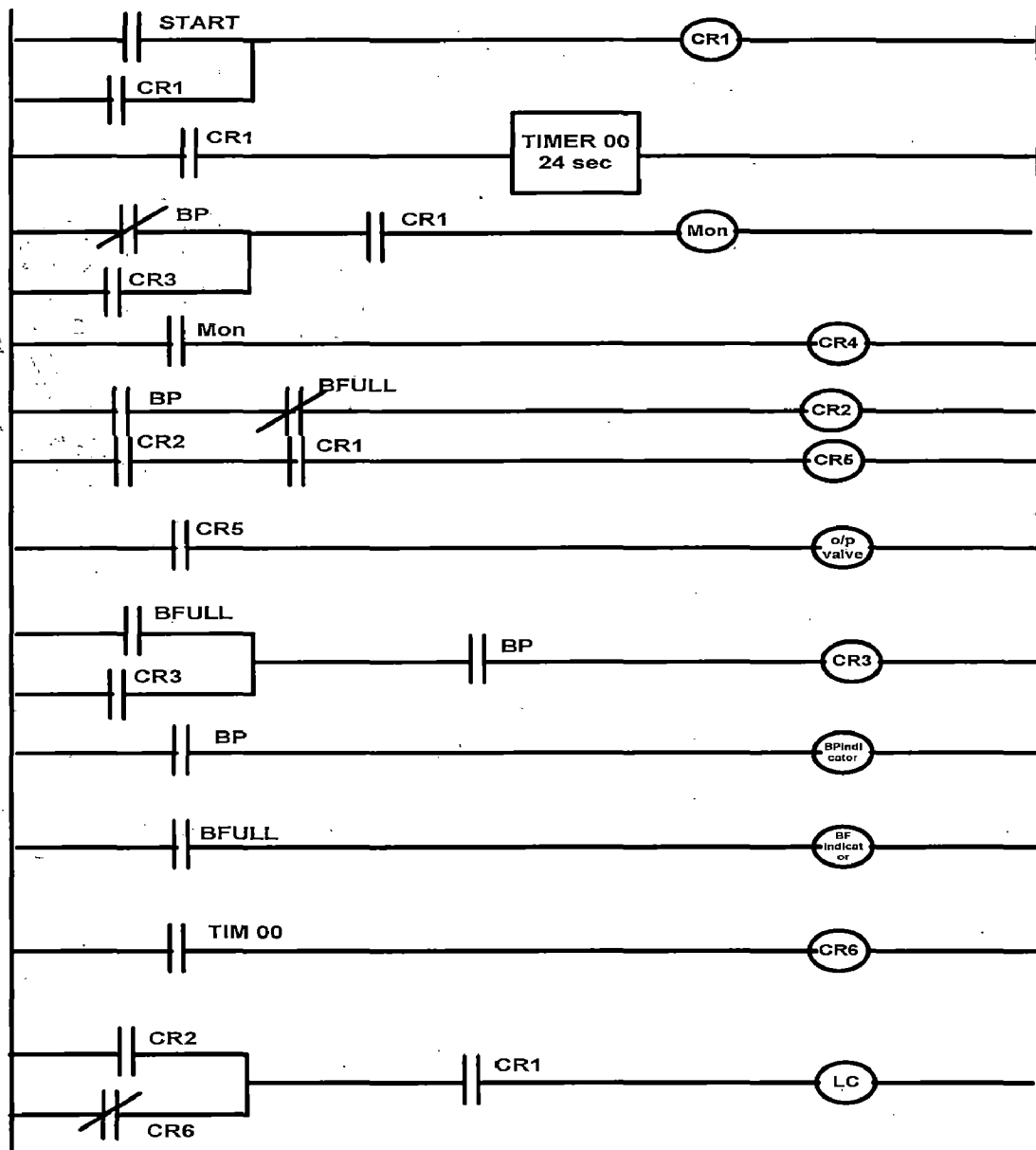


Fig 4.7 The Logic Diagram of the LLD algorithm for sequence controller of bottle filling system.

The rules derived from both the PN and LLD based algorithms of sequence controller for bottle filling system are listed below

LLD Algorithm

- 1) Internal relay CR1 remains activated if START switch is pressed and stays turned on even when START has turned off.

START OR CR1→CR1.

2) The TIM 00 is activated 24 sec after internal relay CR1 is turned on.

CR1→TIM 00.

3) Once TIM 00 is activated it turns on internal relay CR6.

TIM 00→CR6.

4) The conveyor motor is started as long as CR1 and CR3 are enabled and bottle present sensor is not high.

a) **((NOT BP) AND (CR1)) →Mon.**

b) **(CR3 AND CR1) →Mon.**

5) The internal relay CR4 is turned on as long as conveyor motor is on

Mon→CR4.

6) The internal relay CR2 is activated as long as bottle present signal is high and bottle full signal is low.

(BP AND (NOT BFULL)) →CR2.

7) The internal relay CR5 is turned on as long as CR2 and CR1 are active.

CR2 AND CR1→CR5.

8) The o/p valve of bottle filling system is turned on as soon as CR5 is activated and remains high as long as CR5 remains activated.

CR5→ o/p valve.

9) The internal relay CR3 is turned on as soon as both bottle full sensor and bottle present sensor goes high.

a) **BFULL AND BP→CR3.**

b) **CR3 AND BP→CR3.**

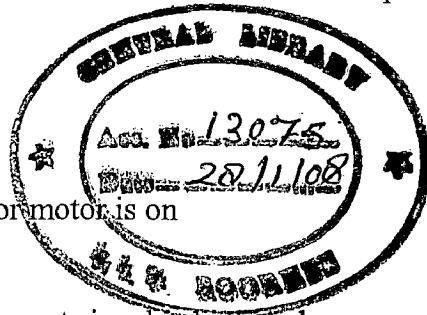
10) The bottle present indicator gets activated as long as bottle present signal is present.

BP→BP indicator.

11) The bottle full indicator gets activated as long as bottle full signal is present.

BFULL→BFULL indicator.

12) The Level control signal is activated if CR2 and CR1 are activated or else if CR6 is not activated and CR1 is activated.



a) **CR2 AND CR1 → LC**

b) **((NOT CR6) AND (CR1)) → LC.**

PN Algorithm

1) If CLR1 place has a marking and start signal is present the sequence controller enters into START1 phase.

CLR1 AND start → START1.

2) When START1 place is marked and after level control signal has remained activated for 24 sec, the sequence controller enters into RUN phase.

START1 AND countlc → RUN.

3) When RUN place is marked, if bottle present is sensed on moving conveyer, the sequence controller enters BPOS state.

RUN AND bpin → BPOS.

4) When BPOS place is marked, if bottle full signal is sensed the sequence controller enters into BFULL state.

BPOS AND bfin → BFULL.

5) The sequence controller goes to the run phase as soon as it senses the resetn signal high in o/p transition of BFULL state.

BFULL AND resetn → RUN.

The sequence controller derived both from PN and LLD Algorithm are compared in terms of number of basic elements and rules, the results so obtained is presented in table 4.3.

Table 4.3: Comparison of LLD and PN based Sequence controller for bottle filling system

Design Algorithm	Basic Elements		IF-Then Transform	
	Nodes	Links	Rules	Operators
Ladder Logic Diagrams (LLD)	34	50	14	12
Petri Nets(PN)	10	09	05	05

It is observed from Table 4.3 that PN based algorithm requires less number of basic elements (19 instead of 84) and less number of rules (5 instead of 14) in comparison to LLD

algorithm. Further the comparison results are represented with the help of bar chart representation in Fig 4.8

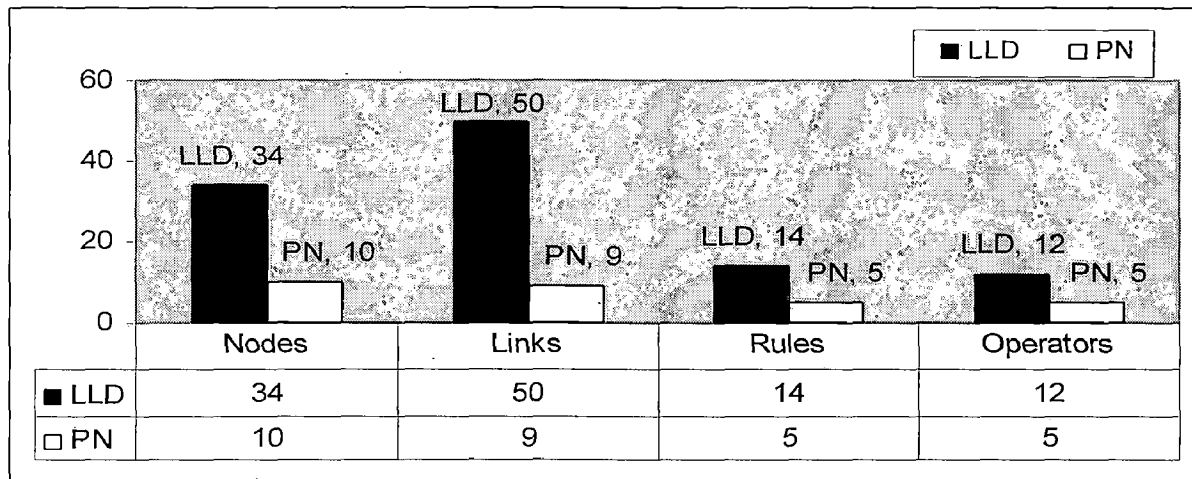


Fig 4.8: The Comparison of LLD and PN algorithms for the sequence controller of bottle filling system.

4.5: Conclusion

In this chapter criteria's on the basis of which PN and LLD based sequence controllers could be compared are discussed. Further, the formalization procedure is described by using the bottle filling system as an example. It is seen that LLD based sequence controller requires more basic elements and rules to implement than its PN counterpart. In next chapter sequence controller for three applications in increasing order of complexity are designed using both LLD and PN approaches. Further, LLD algorithms are obtained from the PN structures and a comparative study is done of all algorithms to find out the corresponding increase in complexity of sequence controller designed using LLD and PN approaches.

CHAPTER 5

Case Studies

5.1: Introduction

In this chapter sequence controllers designed using both LLD and PN for two level, three level and four level elevator systems are compared in terms of number of basic elements and number of rules. The setup on which the experiments are performed is shown in fig 5.1. The set up is obtained from Vinytics Peripherals Ltd, New Delhi. Originally, the set up was designed for four level elevator sequence controller but in this dissertation it is used to design two and three level elevator sequence controllers along with the four level sequence controller in order to find out the corresponding increase in complexity in both PN and LLD algorithms as one moves from two to three and finally to four level elevator sequence control.

For, the two level elevator sequence controller, the floor I/O's beyond first floor are assumed to be redundant and similarly, for three level elevator sequence controller the floor I/O's beyond second floor are considered to be redundant while designing algorithms for two level and three level elevator sequence controller using PN and LLD.

The details of elevator controller card and hardwire interconnections (I/O) has been discussed in appendix A.

The RTPN algorithms for two level, three level and four level sequence controllers are obtained by using

- 1) State Machine Approach.
- 2) Place Oriented Conditionals Approach.

In State Machine based design technique each place of RTPN structure is considered as a global state of the system whereas in Place Oriented Conditionals Approach each place of same RTPN structure is considered as local states. The main difference in both techniques is observed when a system exhibits concurrency; design of sequence controllers for such systems using State Machine approach is more complex in comparison to design using Place Oriented Conditionals approach. The case studies considered in this dissertation doesn't exhibit any concurrency and hence the global states and local states are no different in both

design approaches. Further, when the RTPN structures are implemented using VHDL it is seen that the number of logic elements required to implement sequence controllers using both the design techniques are same. Hence, it is inferred that if a system doesn't exhibit concurrency then sequence controllers for such systems could be designed either by SM approach or Place Oriented Conditionals approach as both lead to same amount of complexity. [12] [38]

5.2:Case Study 1:Two Level Elevator Sequence Controller

For the two level elevator sequence controller system the setup in fig 5.1 which corresponds to four level elevator system is modified in terms of I/O specifications as given in table 5.1

Table 5.1: I/O Specifications for two level elevator system

DESCRIPTION	INPUT	OUTPUT
GROUND	I2	O9
FIRST	I3	O8
MOTOR ON	NIL	O2
MOTOR UP	NIL	O3 ON
MOTOR DOWN	NIL	O3 OFF

Note: The I/O's corresponding to second and third level are not used in design algorithm, so the four level setup operates as a two level setup, with I/O status for upper two levels being redundant.

The coding derived for sequence controller based on LLD Algorithm is given as file elevtwo.asm in CD attached along with this report. The RTPN structure of two level elevator sequence controller obtained using State Machine Approach is shown in Fig 5.2. The RTPN

structure of two level elevator sequence controller using Place Oriented Conditional Approach is shown in Fig 5.3.

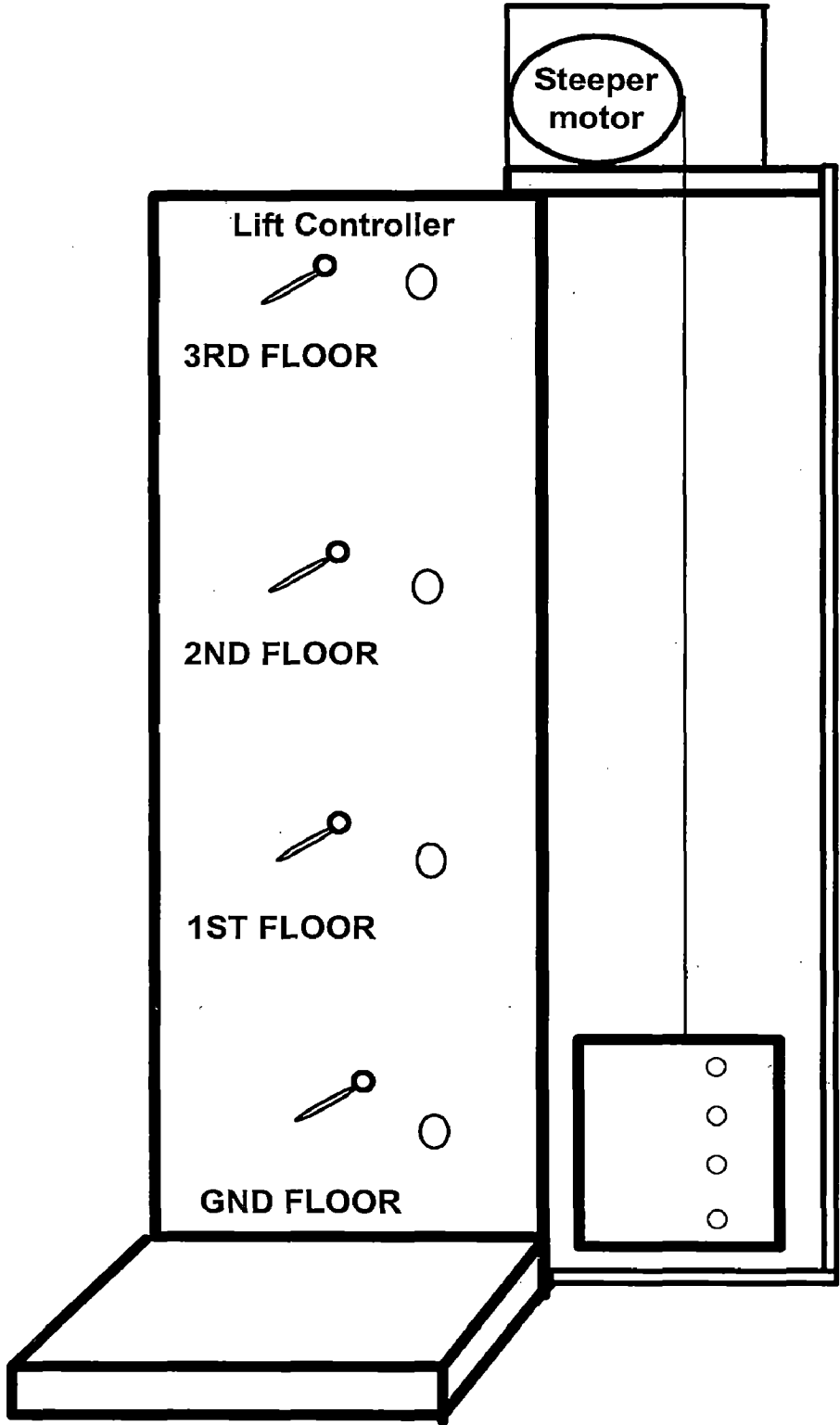


Fig 5.1 The Elevator Setup from Vinytics Peripherals Ltd

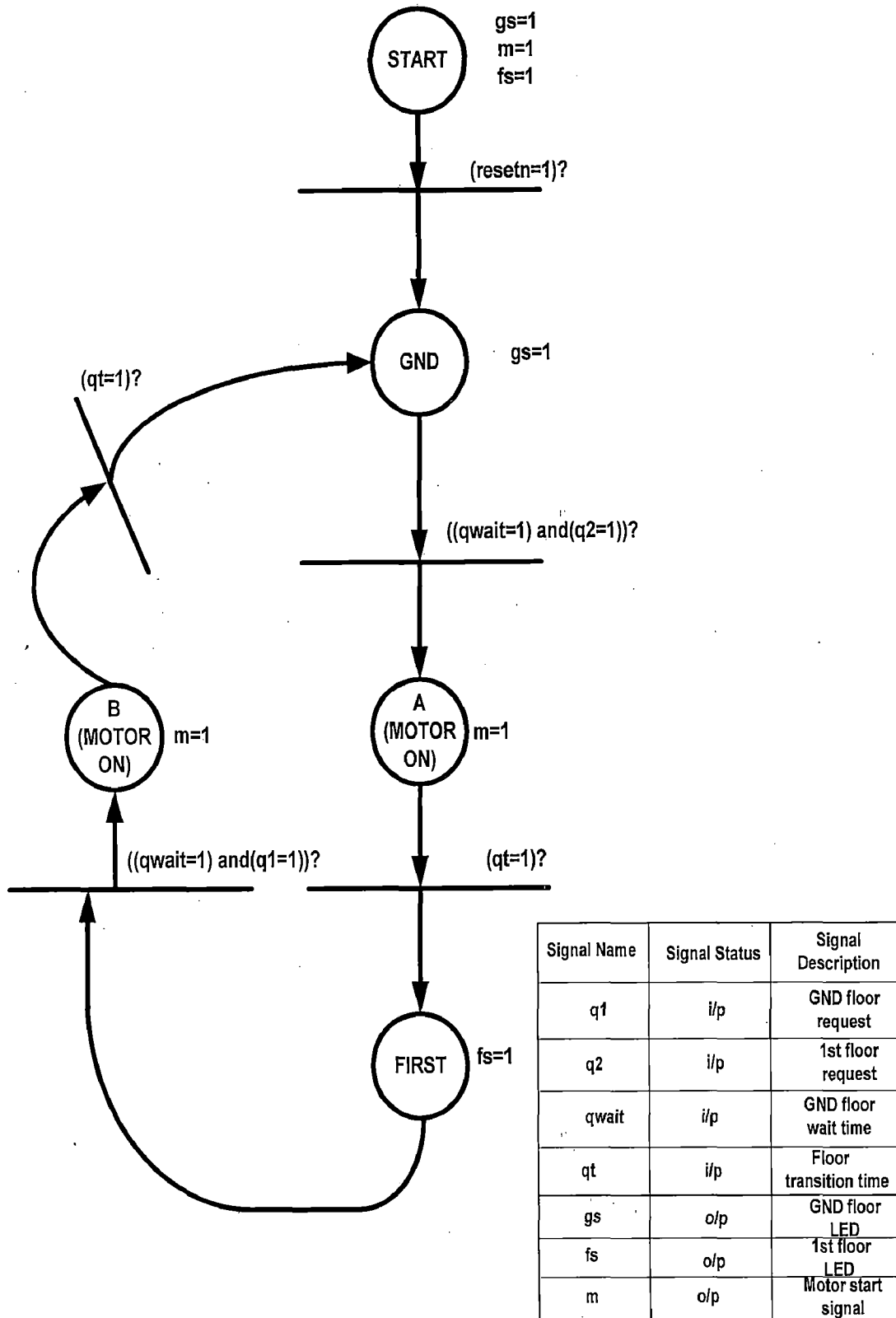


Fig 5.2 The RTPN structure of two level elevator sequence controller (SM Approach)

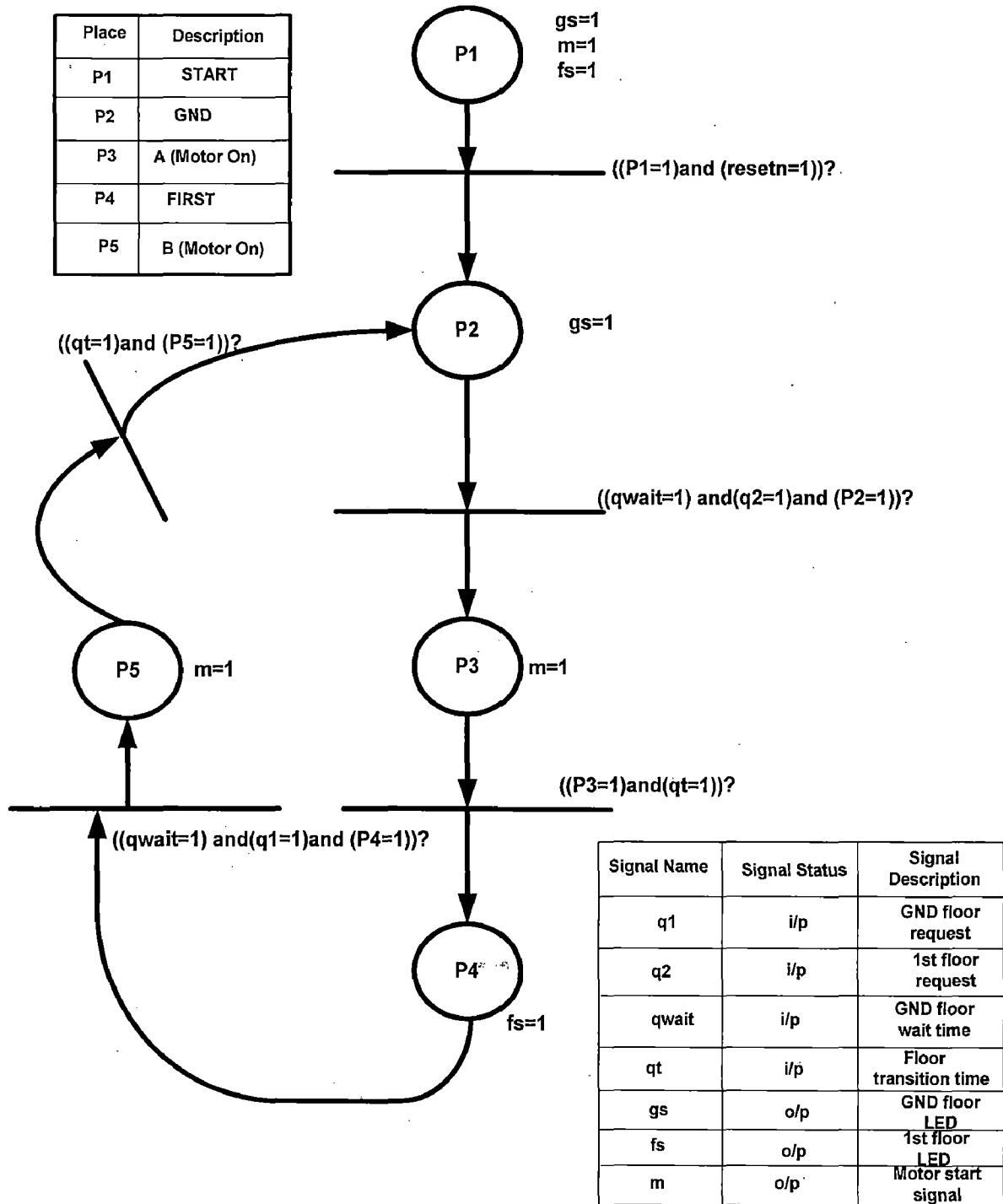


Fig 5.3 The RTPN structure of two level elevator sequence controller (Place Oriented Conditionals Approach)

The coding derived for two level sequence controller based on LLD algorithm obtained by means of formalization from the RTPN structure(SM Approach) of two level sequence controller is given as elev2p.asm on CD attached with this report.

Both, the algorithms are compared in terms of number of basic elements and number of rules and operators and the results so obtained are tabulated in table 5.2

Table 5.2: Comparison of Sequence Controllers designed for two level elevator system

DESIGN ALGORITHMS FOR TWO LEVEL ELEVATOR SYSTEM	BASIC ELEMENTS		IF-THEN ELSE TRANSFORMATION	
	NODES	LINKS	RULES	OPERATORS
LADDER LOGIC DIAGRAMS (LLD) DIRECT IMPLEMENTATION	61	100	37	20
PETRI NETS (PN) STATE MACHINE APPROACH	10	10	05	02
PETRI NETS (PN) PLACE CONDITIONAL APPROACH	10	10	05	07
LADDER LOGIC DIAGRAMS (LLD) OBTAINED AFTER FORMALIZATION	40	61	25	09

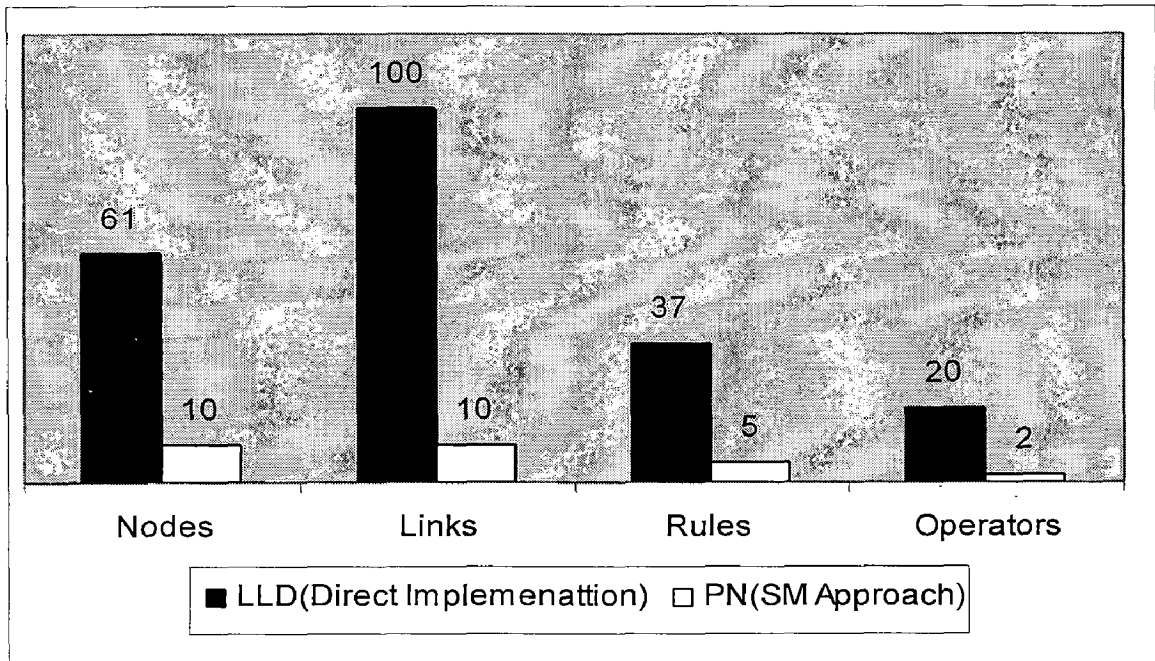


Fig 5.4.1 The comparison of PN (SM) and LLD based sequence controllers for two level elevator system

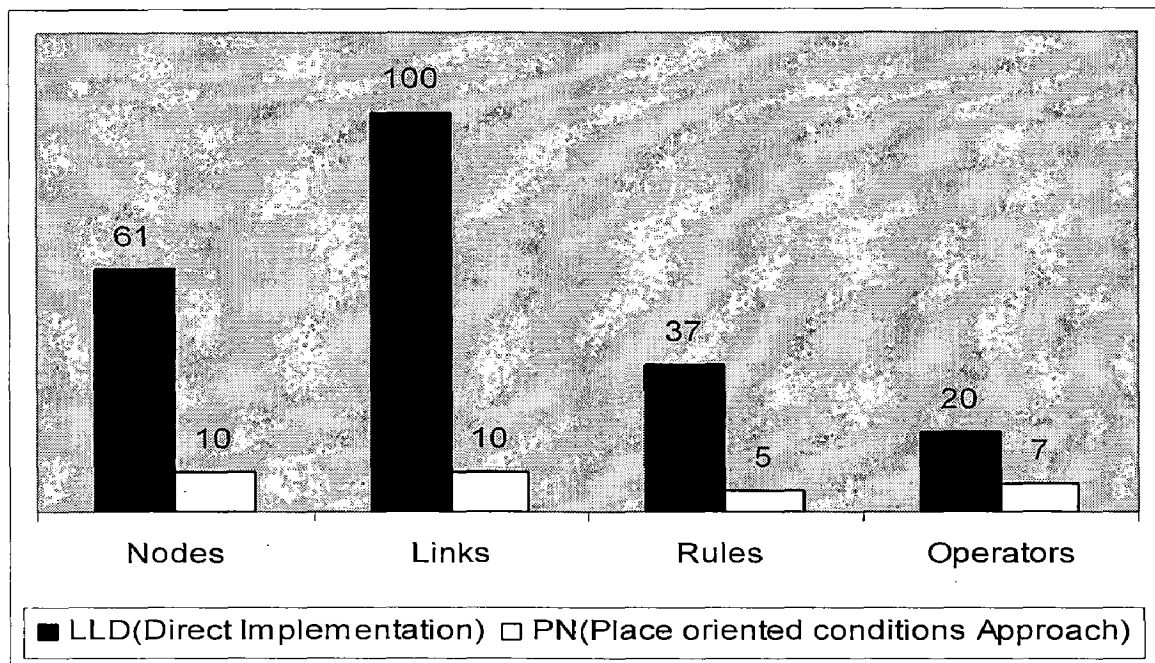


Fig 5.4.2 The comparison of PN (Place Oriented Conditionals Approach) and LLD based sequence controllers for two level elevator system

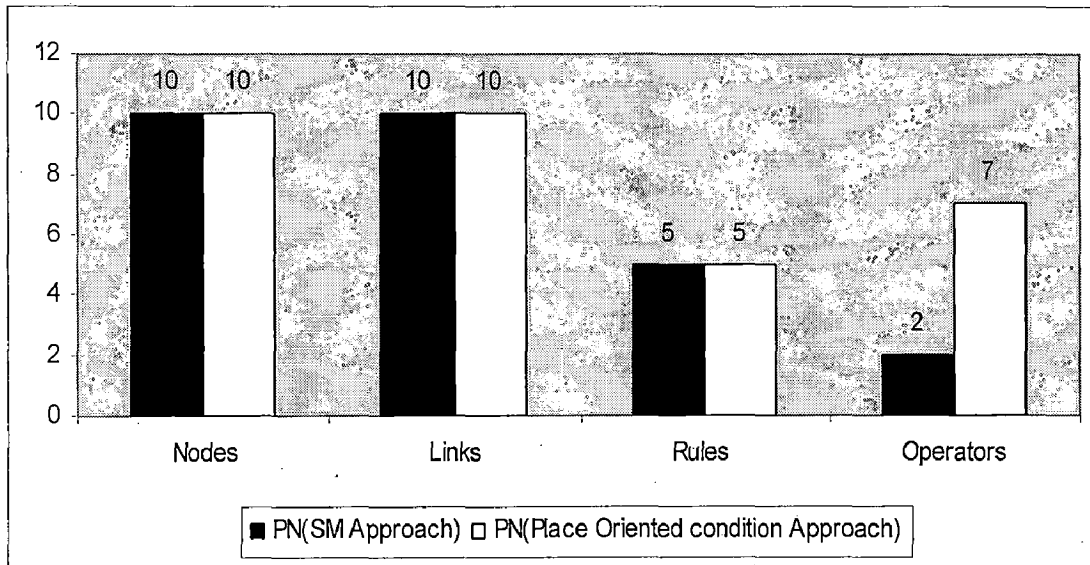


Fig 5.4.3 The Comparison of PN (SM Approach) and PN (Place Oriented Conditionals Approach) based sequence controllers for two level elevator system

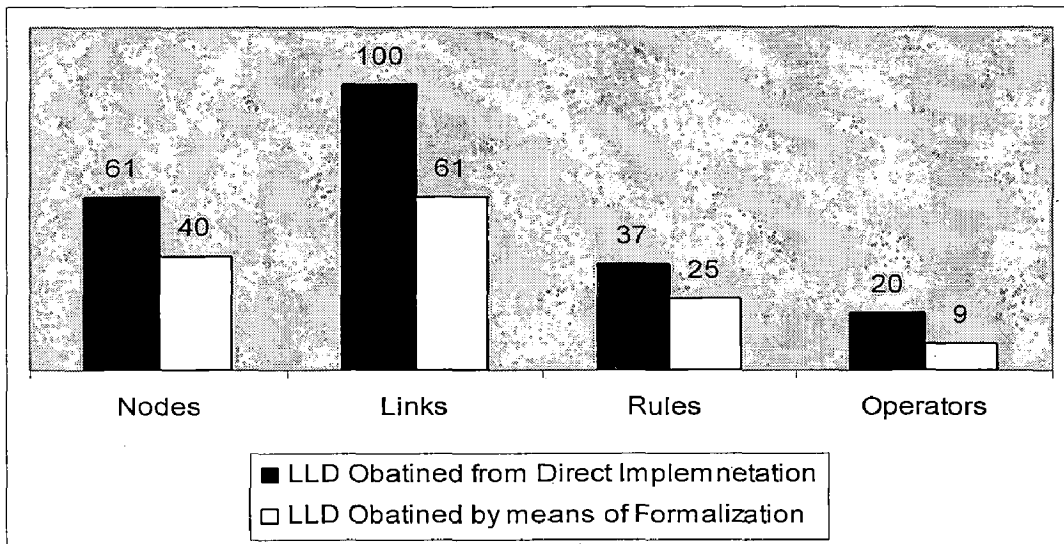


Fig 5.4.4 The Comparison of LLD (Directly Obtained) and LLD (Obtained from Formalization) based sequence controllers for two level elevator system.

It is observed from fig 5.4.1 and 5.4.2 RTPN structures of sequence controller for two level elevator system requires less number of basic elements and rules than a LLD based sequence controller obtained without formalization for the same system.

In fig 5.4.3 both the RTPN structures are compared to show that SM approach is a better solution than Place Oriented Conditionals Approach in terms of number of operators used.

Finally, in fig 5.4.4 the LLD based sequence controller obtained from PN model (SM Approach) is compared with directly implemented LLD algorithm to show that LLD based sequence controllers obtained by means of Formalization require less elements and rules than the ones that are implemented directly.

Fig 5.4.5 shows the Reachability analysis done on the RTPN algorithm for two level elevator controller. It is observed from the Reachability tree that the sequence controller is safe and free from any deadlock.

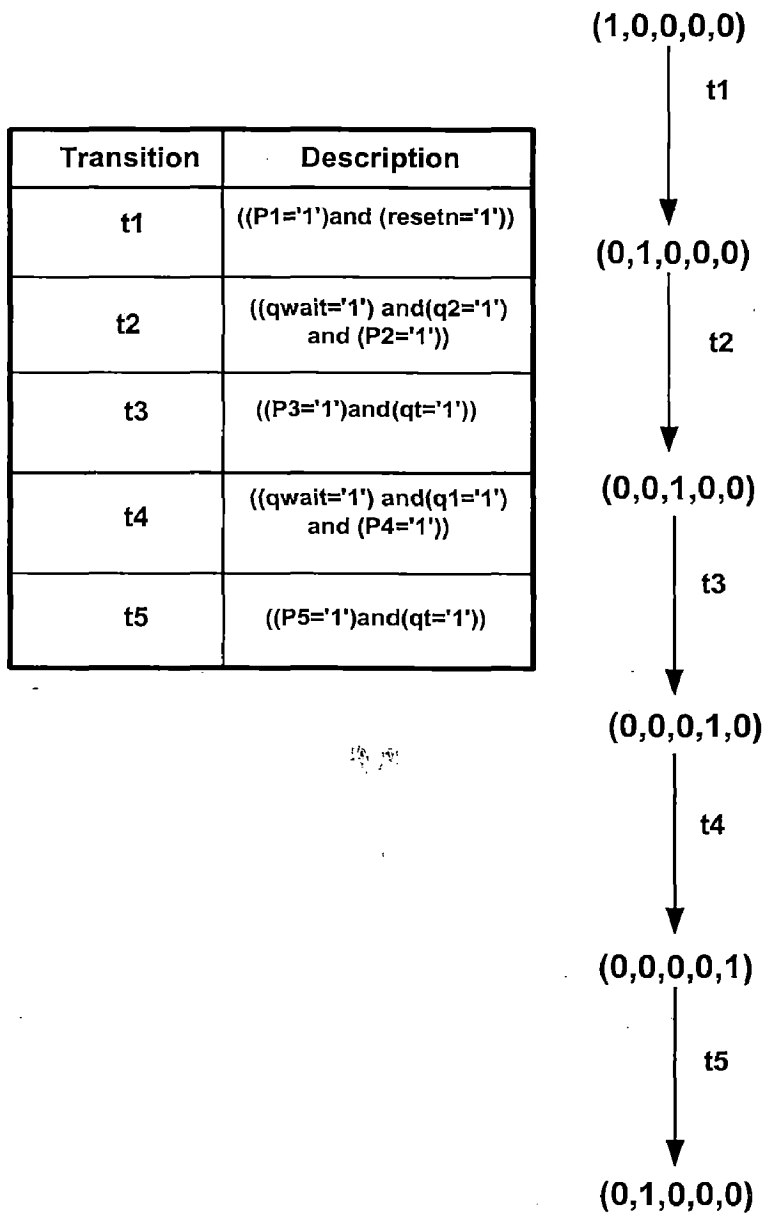


Fig 5.4.5 The Reachability Graph for RTPN structure of two level elevator sequence controller

5.3:Case Study 2:Three Level Elevator Sequence Controller

For the Three level elevator sequence controller system the setup in fig 5.1 which corresponds to four level elevator controller is modified in terms of I/O specifications as given in table 5.3

Table 5.3: I/O Specifications for three level elevator sequence controller

DESCRIPTION	INPUT	OUTPUT
GROUND	I2	O9
FIRST	I3	O8
SECOND	I4	O14
MOTOR ON	NIL	O2
MOTOR UP	NIL	O3 ON
MOTOR DOWN	NIL	O3 OFF

Note: The I/O's corresponding to third level are not used in design algorithm, so the four level setup operates as a three level setup, with I/O status for upper level being redundant.

The coding derived from the LLD Algorithm for three level elevator sequence controller is given as file elevthree.asm on CD attached with this report. The RTPN structure of three level elevator sequence controller obtained using State Machine Approach is given in fig 5.5. The RTPN structure of three level elevator sequence controller using Place Oriented Conditionals Approach is shown in Fig 5.6

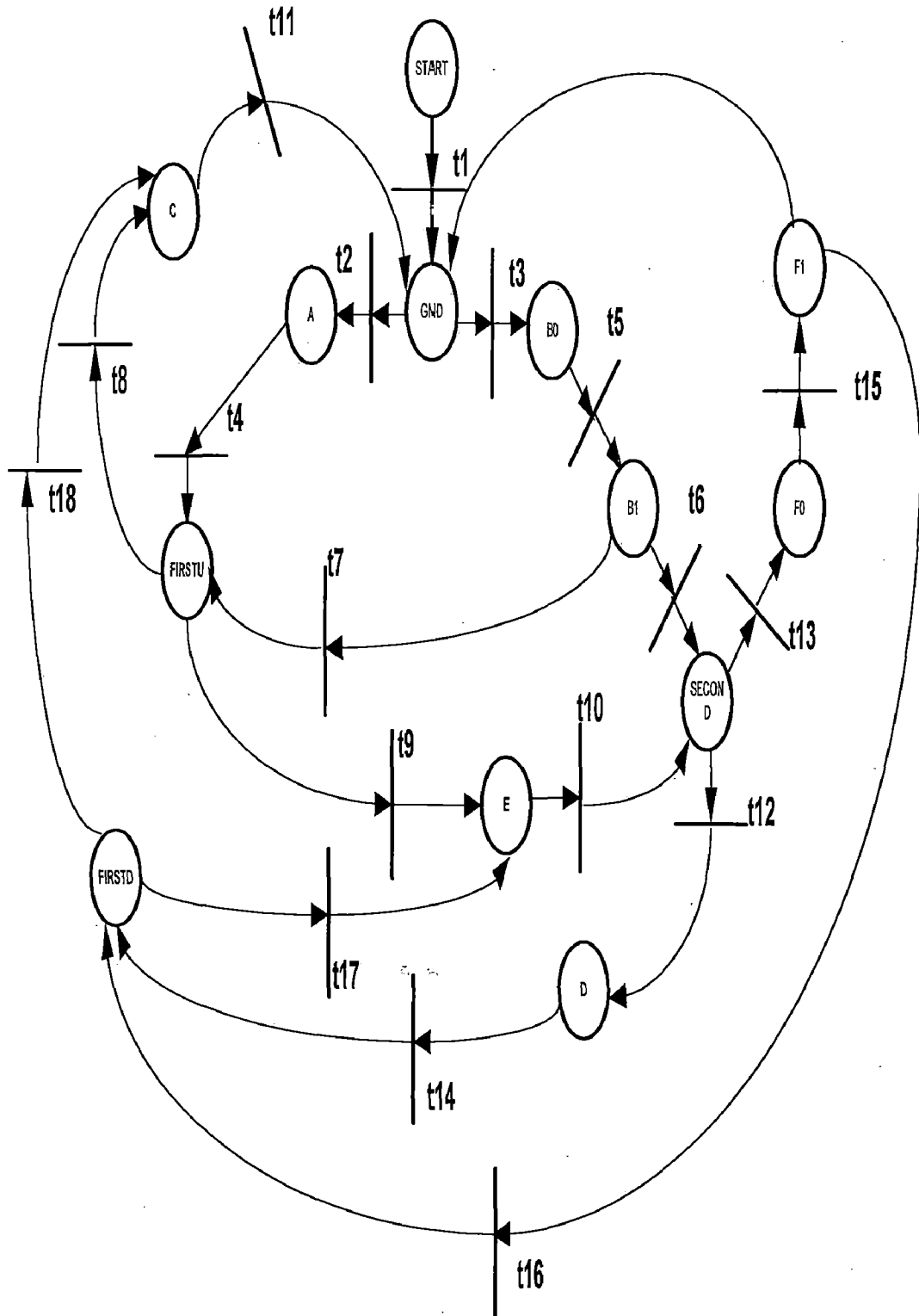


Fig 5.5 The RTPN structure of a three level elevator sequence controller (SM Approach)

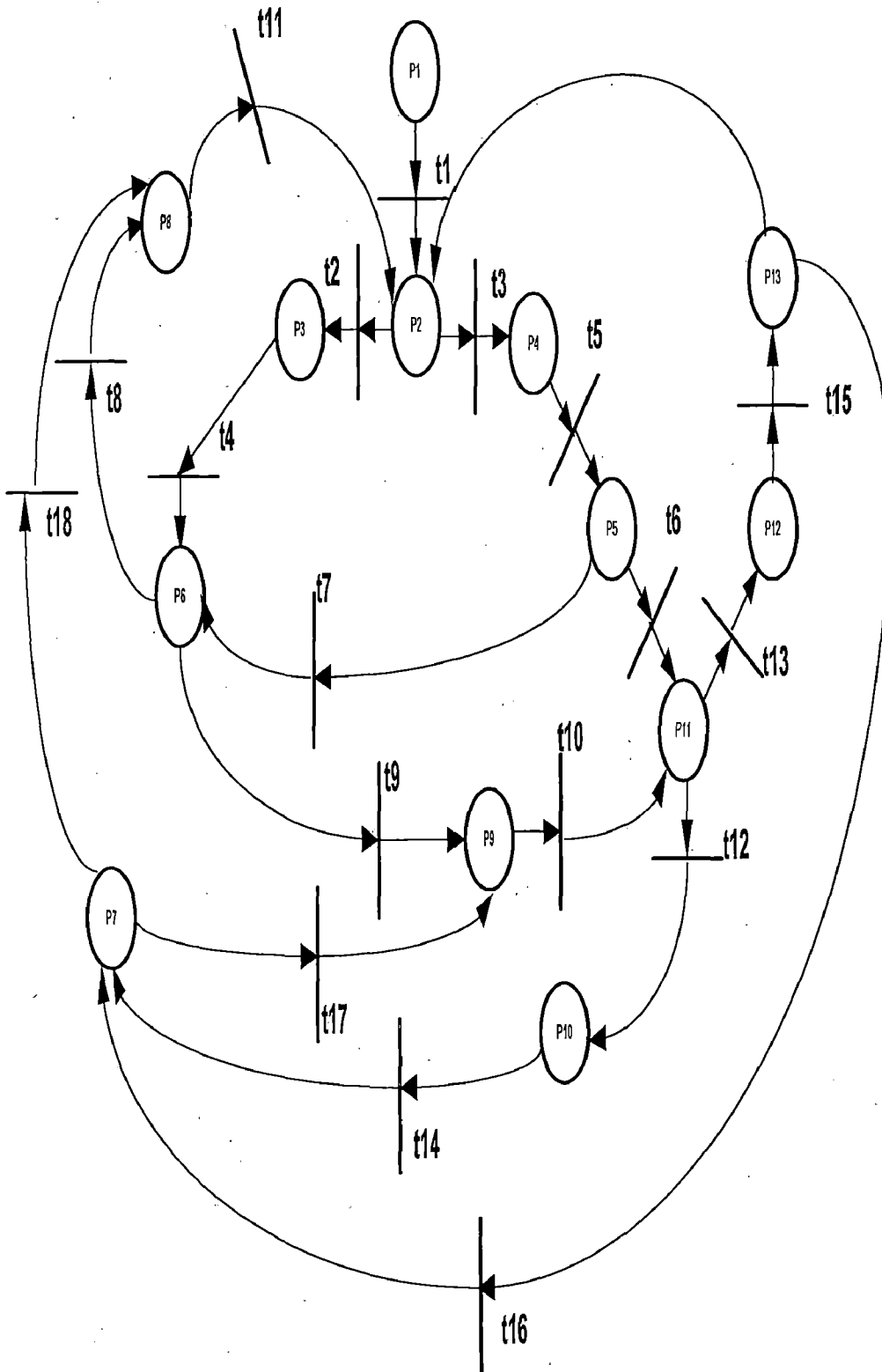


Fig 5.6 The RTPN Structure of three level elevator sequence controller (Place Oriented Conditionals Approach)

Table 5.4: Details description of transitions and places of RTPN structure for three level elevator sequence controller.

Name	Description	SM Approach	Place Oriented Conditional Approach	Outputs
START	PLACE	--	P1	gs=1 fs=1 ss=1 m=1
GND	same	--	P2	gs=1
A	same	--	P3	m=1
B0	same	--	P4	m=1
B1	same	--	P5	m=1
FIRSTU	same	--	P6	fs=1
FIRSTD	same	--	P7	fs=1
C	same	--	P8	m=0
E	same	--	P9	m=1
D	same	--	P10	m=0
SECOND	same	--	P11	ss=1
FO	same	--	P12	m=0
F1	same	--	P13	m=0
t1	TRANSITION	(resetrn='1')	((P1='1')and (resetrn='1'))	nil
t2	same	((q2='1')and (qwait='1'))	((P2='1')and(q2='1') and (qwait='1'))	do
t3	same	((q3='1')and (qwait='1'))	((P2='1')and(q3='1') and (qwait='1'))	do
t4	same	(qt='1')	((P3='1')and(qt='1'))	do
t5	same	(qt='1')	((P4='1')and(qt='1'))	do
t6	same	(qt1='1')	((P5='1')and(qt1='1'))	do
t7	same	(q2='1')	((P5='1')and(q2='1'))	do

Table 5.4 Contd.....

Name	Description	SM Approach	Place Oriented Conditional Approach	Outputs
t8	same	((q1='1')and(qwait='1'))	((P6='1')and(q1='1') and (qwait='1'))	do
t9	same	((q3='1')and(qwait='1'))	((P6='1')and(q1='1')and(qwait='1'))	do
t10	same	(qt='1')	((P9='1')and(qt='1'))	do
t11	same	(qt='1')	((P8='1')and(qt='1'))	do
t12	same	((q2='1')and(qwait='1'))	((P11='1')and(q2='1')and(qwait='1'))	do
t13	same	((q1='1')and(qwait='1'))	((P11='1')and(q1='1')and(qwait='1'))	do
t14	same	(qt='1')	((P10='1')and(qt='1'))	do
t15	same	(qt='1')	((P12='1')and(qt='1'))	do
t16	same	(q2='1')	((P13='1')and(q2='1'))	do
t17	same	((q3='1')and(qwait='1'))	((P7='1')and(q3='1')and(qwait='1'))	do
t18	same	((q1='1')and(qwait='1'))	((P7='1')and(q1='1')and(qwait='1'))	do

The description of places and firing rules used in transitions of RTPN structure of Three Level elevator Sequence controller is given in table 5.4. It is observed that RTPN structure realization by both SM and Place Oriented Conditionals Approach require same number of places and transitions. The details of I/O used in describing the RTPN based controller for three level elevator sequence controller is given in table 5.5.

The coding derived from the LLD algorithm obtained from RTPN structure by Formalization procedure is given as elev3p.asm on CD attached with this report. [17][29][25]

The algorithms for three level elevator sequence controller are compared in terms of number of basic elements and number of rules and operators, the results so obtained are tabulated in table 5.6.

Table 5.5 : I/O's used in RTPN structure of three level elevator sequence controller.

Signal	Signal Status	Signal Description
q1	i/p	GND floor request
q2	i/p	FIRST floor request
q3	i/p	SECOND floor request
qt,qt1	i/p	transition time between two floors
qwait	i/p	floor wait time.
gs	o/p	GND floor LED indicator
fs	o/p	FIRST floor LED indicator
ss	o/p	SECOND floor LED indicator
m	o/p	LED indicating Motor on/off

Table 5.6: Comparison of Sequence Controllers designed for three level elevator system

DESIGN ALGORITHMS FOR THREE LEVEL ELEVATOR SYSTEM	BASIC ELEMENTS		IF-THEN ELSE TRANSFORMATION	
	NODES	LINKS	RULES	OPERATORS
LADDER LOGIC DIAGRAMS (LLD) DIRECT IMPLEMENTATION	157	257	107	59
PETRI NETS (PN) STATE MACHINE APPROACH	31	39	18	8
PETRI NETS (PN) PLACE CONDITIONAL APPROACH	31	39	18	27
LADDER LOGIC DIAGRAMS (LLD) OBTAINED AFTER FORMALIZATION	128	204	76	30

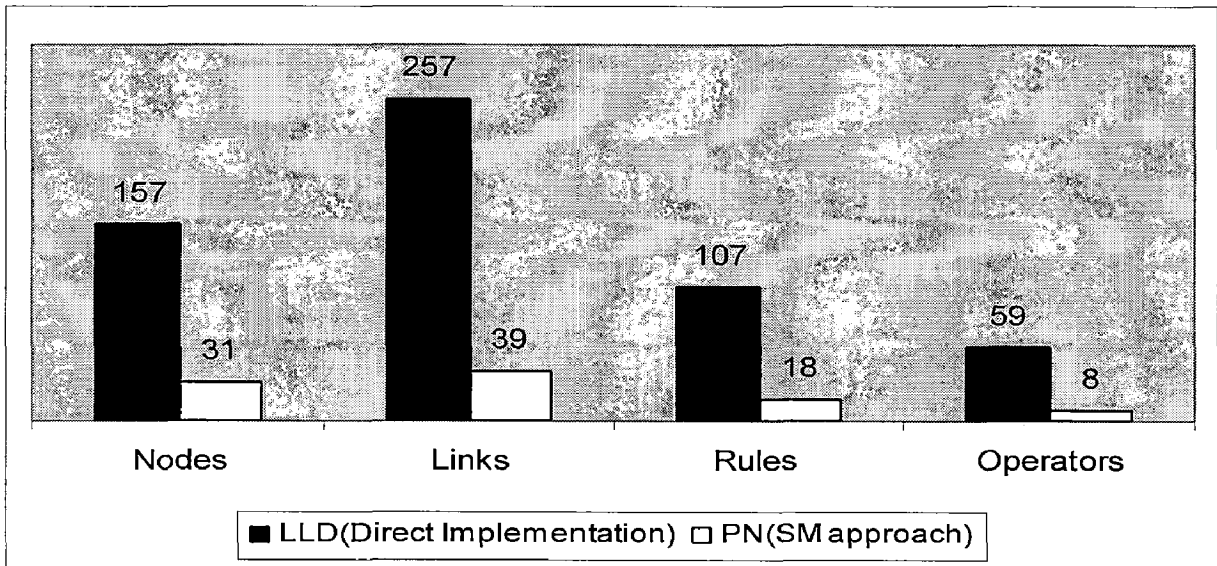


Fig 5.7.1 The comparison of PN (SM Approach) and LLD based sequence controllers three level elevator system

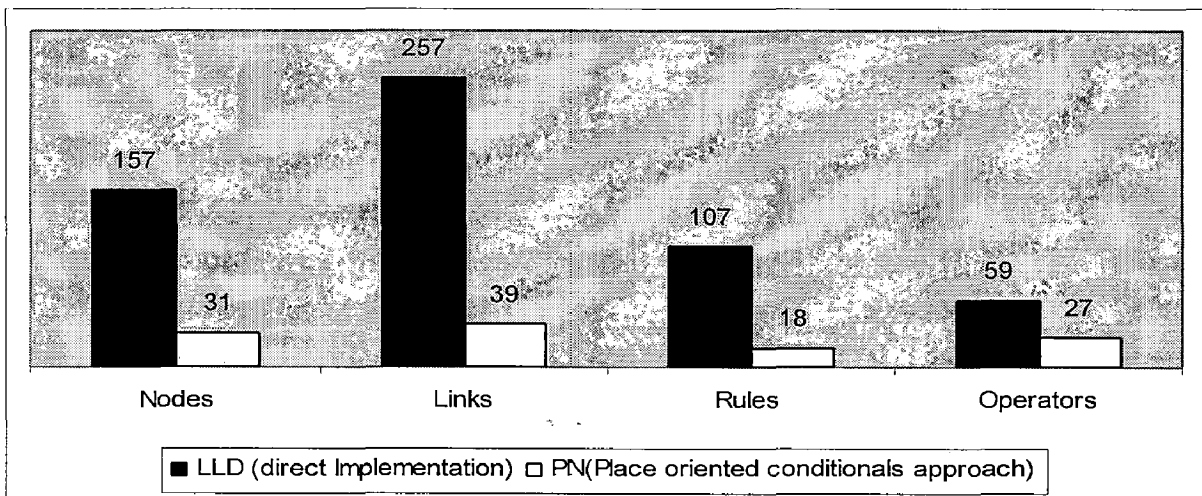


Fig 5.7.2 The comparison of PN (Place Oriented Conditionals Approach) and LLD based sequence controllers for three level elevator system

Fig 5.7.1 shows the comparison of algorithms developed for three level elevator controller using PN (SM Approach) and LLD (Direct Implementation).

Fig 5.7.2 shows the comparison of algorithms developed for three level elevator controller using PN (Place Oriented Conditionals Approach) and LLD (Direct Implementation).

It is seen that in both cases of PN structures it supercedes its LLD counterpart both in terms of Basic Elements and IF-THEN rules approach.

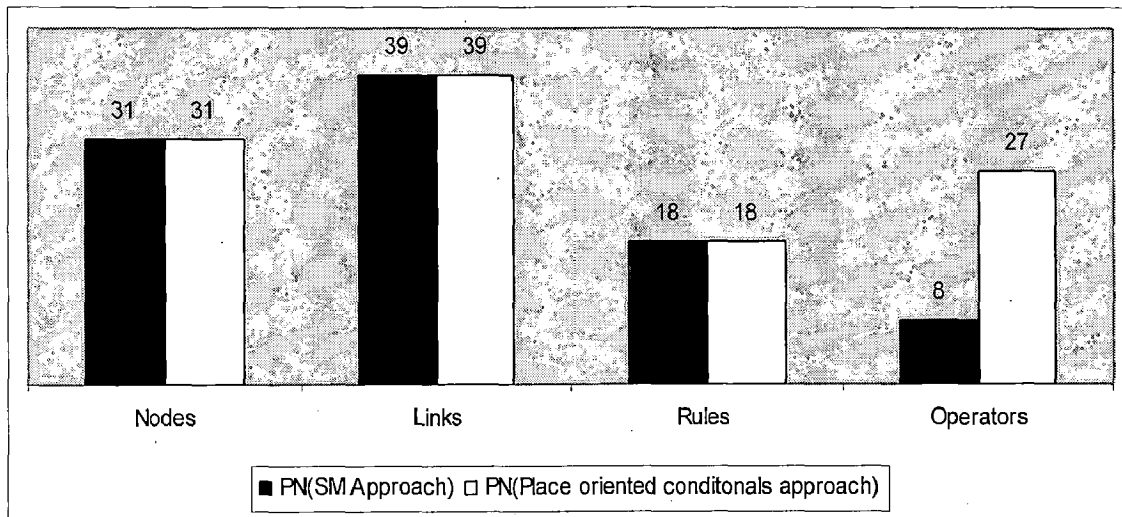


Fig 5.7.3 The comparison of PN (SM Approach) and PN (Place Oriented Conditionals Approach) based sequence controllers for three level elevator system

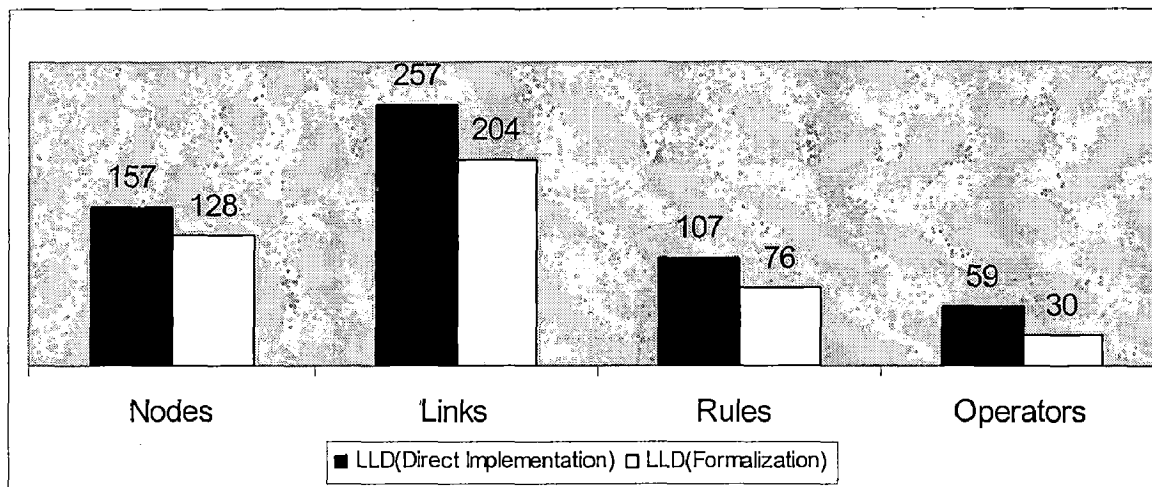


Fig 5.7.4 The Comparison of LLD (Directly Obtained) and LLD (Obtained from Formalization) based sequence controllers for three level elevator system.

In fig 5.7.3 both the PN algorithms are compared to show that SM approach is a better solution than Place Oriented Conditionals Approach in terms of number of operators used. Finally, in fig 5.7.4 the LLD algorithm from PN model (SM Approach) is compared with directly implemented LLD algorithm to show that LLD algorithm obtained by means of Formalization require less elements and rules than the ones that are implemented directly. Here, the LLD algorithm obtained by means of formalization is better in decision making than the one directly implemented, as the PN structure has FIRSTU and FIRSTD places it helps the LLD algorithm to decide to service upper floors of First floor while going up

owing to the module FIRSTD and lower floors of first floor while coming down owing to the module FIRSTD both of which are obtained from places FIRSTD and FIRSTD of PN algorithm.

5.4:Case Study 3:Four Level Elevator Controller System

For the four level elevator controller there is no modifications made to the I/O specifications of setup of fig 5.1. The I/O specifications for four level controller setup are shown in table 5.7

Table 5.7: The I/O specifications of four level elevator system

Description	Input	Output
GROUND	I2	O9
FIRST	I3	O8
SECOND	I4	O14
THIRD	I5	O15
MOTOR ON	NIL	O2
MOTOR UP	NIL	O3 ON
MOTOR DOWN	NIL	O3 OFF

The coding of the LLD Algorithm for the four level elevator sequence controller is given as file elevfour.asm in the CD attached with this report. The RTPN structure of four level elevator sequence controller designed using State Machine Approach is given in fig 5.8 and that using Place Oriented Conditionals Approach is shown in fig 5.9

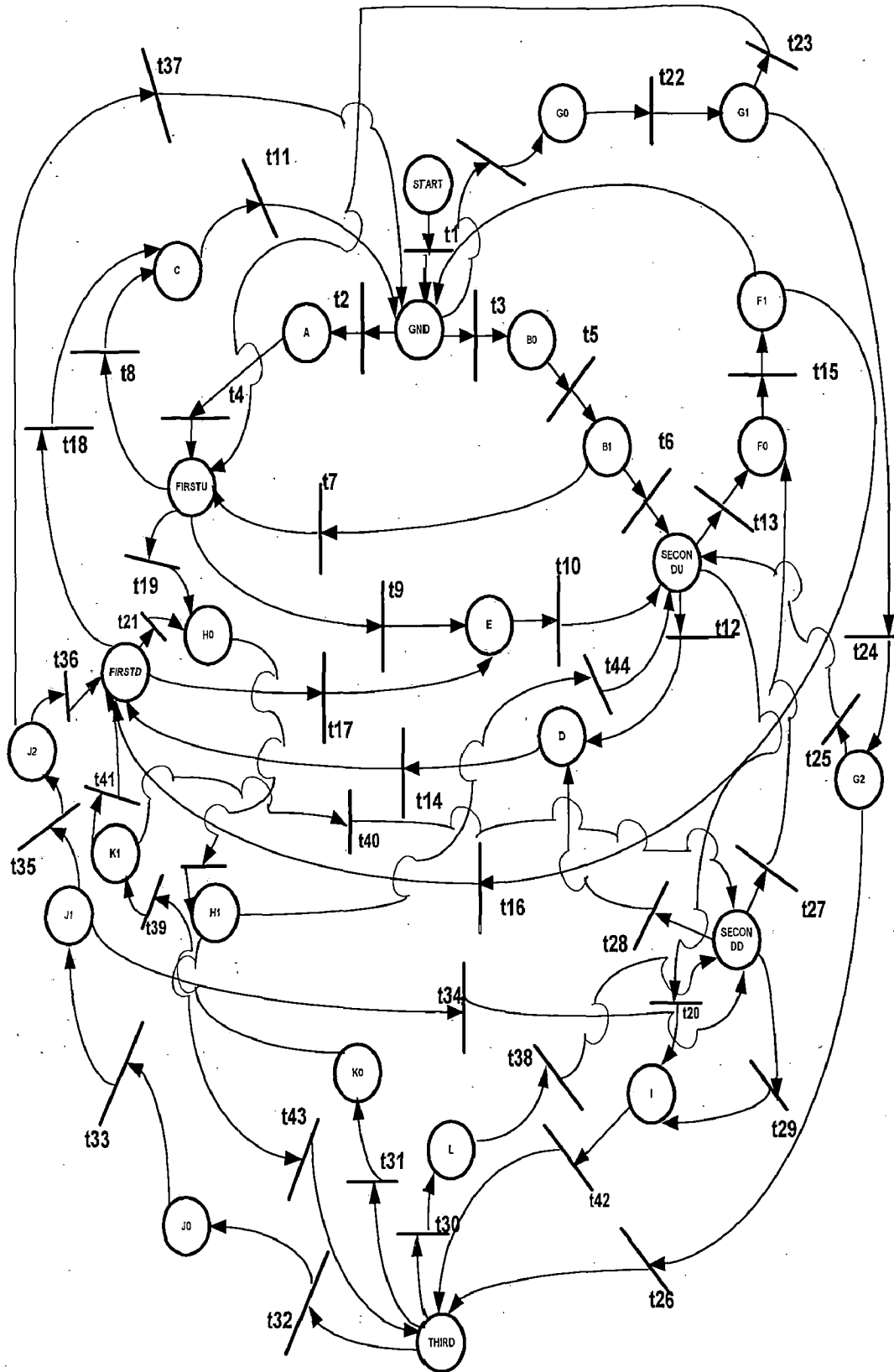
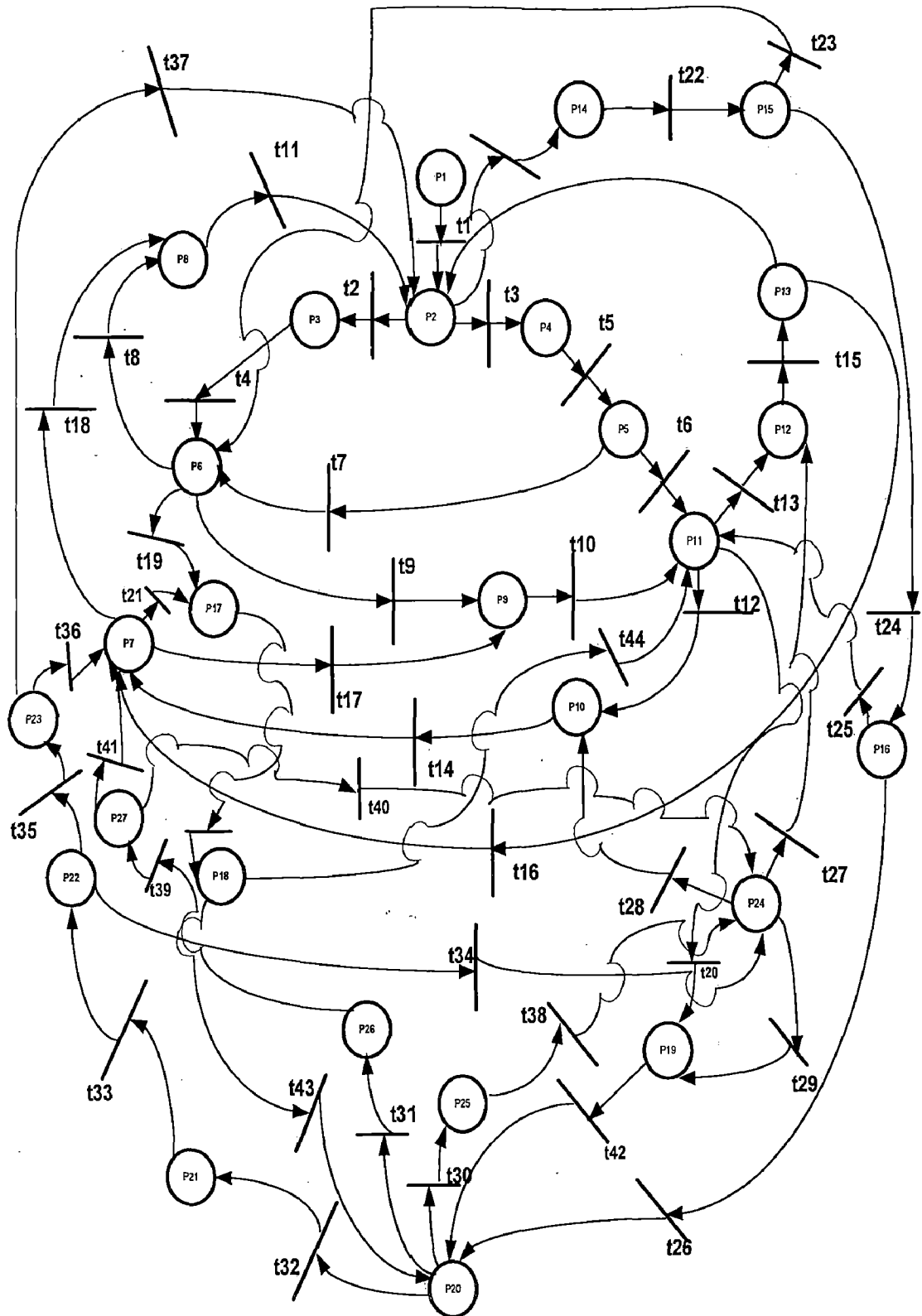


Fig 5.8: The RTPN structure of four level Elevator Sequence Controller (SM Approach)



**Fig 5.9: The RTPN Algorithm of Four-Level Elevator Sequence Controller
(Place Oriented Conditionals Approach)**

Table 5.8: The Place and transition descriptions of RTPN structures of sequence controllers for four level elevator system

Name	Description	SM Approach	Place Oriented Conditional Approach	Outputs
START	PLACE	--	P1	gs=1 fs=1 ss=1 m=1
GND	same	--	P2	gs=1
A	same	--	P3	m=1
B0	same	--	P4	m=1
B1	same	--	P5	m=1
FIRSTU	same	--	P6	fs=1
FIRSTD	same	--	P7	fs=1
C	same	--	P8	m=0
E	same	--	P9	m=1
D	same	--	P10	m=0
SECOND	same	--	P11	ss=1
FO	same	--	P12	m=0
F1	same	--	P13	m=0
G0	same	--	P14	m=1
G1	same	--	P15	m=1
G2	same	--	P16	m=1
H0	same	--	P17	m=1
H1	same	--	P18	m=1
I	same	--	P19	m=1
THIRD	same	--	P20	ts=1
J0	same	--	P21	m<='0'
J1	same	--	P22	m<='0'
J2	same	--	P23	m<='0'
SECONDD	same	--	P24	ss=1
K0	same	--	P25	m=0
K1	same	--	P26	m=0
L	same	--	P27	m=0
t1	TRANSITION	(resetrn='1')	((P1='1')and (resetrn='1'))	nil
t2	same	((q2='1')and(qwait='1'))	((P2='1')and(q2='1')and (qwait='1'))	do
t3	same	((q3='1')and(qwait='1'))	((P2='1')and(q3='1')and (qwait='1'))	do
t4	same	(qt='1')	((P3='1')and(qt='1'))	do

Table 5.8 Contd...

Name	Description	SM Approach	Place Oriented Conditional Approach	Outputs
t5	same	(qt='1')	((P4='1')and(qt='1'))	do
t6	same	(qt1='1')	((P5='1')and(qt1='1'))	do
t7	same	(q2='1')	((P5='1')and(q2='1'))	do
t8	same	((q1='1')and(qwait='1'))	((P6='1')and(q1='1')and(qwait='1'))	do
t9	same	((q3='1')and(qwait='1'))	((P6='1')and(q1='1')and(qwait='1'))	do
t10	same	(qt='1')	((P9='1')and(qt='1'))	do
t11	same	(qt='1')	((P8='1')and(qt='1'))	do
t12	same	((q2='1')and(qwait='1'))	((P11='1')and(q2='1')and(qwait='1'))	do
t13	same	((q1='1')and(qwait='1'))	((P11='1')and(q1='1')and(qwait='1'))	do
t14	same	(qt='1')	((P10='1')and(qt='1'))	do
t15	same	(qt='1')	((P12='1')and(qt='1'))	do
t16	same	(q2='1')	((P13='1')and(q2='1'))	do
t17	same	((q3='1')and(qwait='1'))	((P7='1')and(q3='1')and(qwait='1'))	do
t18	same	((q1='1')and(qwait='1'))	((P7='1')and(q1='1')and(qwait='1'))	do
t19	same	((q4='1')and(qwait='1'))	((P6='1')and(q1='1')and(qwait='1'))	do
t20	same	((q4='1')and(qwait='1'))	((P11='1')and(q1='1')and(qwait='1'))	do
t21	same	((q4='1')and(qwait='1'))	((P7='1')and(q1='1')and(qwait='1'))	do
t22	same	(qt='1')	((P14='1')and(qt='1'))	do
t23	same	(q2='1')	((P15='1')and(q2='1'))	do
t24	same	(qt1='1')	((P15='1')and(qt1='1'))	do
t25	same	(q3='1')	((P16='1')and(q3='1'))	do
t26	same	(qt2='1')	((P16='1')and(qt2='1'))	do
t27	same	((q1='1')and(qwait='1'))	((P24='1')and(q1='1')and(qwait='1'))	do
t28	same	((q2='1')and(qwait='1'))	((P24='1')and(q2='1')and(qwait='1'))	do
t29	same	((q4='1')and(qwait='1'))	((P24='1')and(q1='1')and(qwait='1'))	do
t30	same	((q3='1')and(qwait='1'))	((P20='1')and(q3='1')and(qwait='1'))	do
t31	same	((q2='1')and(qwait='1'))	((P20='1')and(q2='1')and(qwait='1'))	do
t32	same	((q1='1')and(qwait='1'))	((P20='1')and(q1='1')and(qwait='1'))	do
t33	same	(qt='1')	((P21='1')and(qt='1'))	do
t34	same	(q3='1')	((P22='1')and(q3='1'))	do
t35	same	(qt1='1')	((P22='1')and(qt1='1'))	do
t36	same	(q2='1')	((P23='1')and(q2='1'))	do
t37	same	(qt2='1')	((P23='1')and(qt2='1'))	do
t38	same	(qt='1')	((P25='1')and(qt='1'))	do
t39	same	(qt='1')	((P26='1')and(qt='1'))	do

Table 5.8 Contd...

Name	Description	SM Approach	Place Oriented Conditional Approach	Outputs
t40	same	(q3='1')	((P27='1')and(q3='1'))	do
t41	same	(qt1='1')	((P27='1')and(qt1='1'))	do
t42	same	(qt='1')	((P19='1')and(qt='1'))	do
t43	same	(qt1='1')	((P18='1')and(qt1='1'))	do
t44	same	(q3='1')	((P18='1')and(q3='1'))	do

The details of transitions and place descriptions and outputs associated with places are given in table 5.8. The details of I/O used in describing PN based controller for four level elevator system is given in table 5.9

Table 5.9: The I/O description for RTPN structures of four level sequence controller

Signal	Signal Status	Signal Description
q1	i/p	GND floor request
q2	i/p	FIRST floor request
q3	i/p	SECOND floor request
q4	i/p	THIRD floor request
qt,qt1,qt2	i/p	transition time between two floors
qwait	i/p	Floor waits time.
gs	o/p	GND floor LED indicator
fs	o/p	FIRST floor LED indicator
ss	o/p	SECOND floor LED indicator
ts	o/p	THIRD floor LED indicator
m	o/p	LED indicating Motor on/off

The coding obtained from the LLD Algorithm designed by means of formalization from the RTPN structure of four level elevator sequence controller is given as elev4p.asm in CD attached with this report. [16] [17] [25]

Now, the algorithms for four level elevator controller are compared in terms of number basic elements, number of rules and operators and the results so obtained are tabulated in table 5.10

Table 5.10: Comparison of sequence controllers designed for four level elevator system

DESIGN ALGORITHMS FOR FOUR LEVEL ELEVATOR SYSTEM	BASIC ELEMENTS		IF-THEN ELSE TRANSFORMATION	
	NODES	LINKS	RULES	OPERATORS
LADDER LOGIC DIAGRAMS (LLD) DIRECT IMPLEMENTATION	294	458	216	118
PETRI NETS (PN) STATE MACHINE APPROACH	71	93	44	18
PETRI NETS (PN) PLACE CONDITIONAL APPROACH	71	93	44	61
LADDER LOGIC DIAGRAMS (LLD) OBTAINED AFTER FORMALIZATION	266	419	163	58

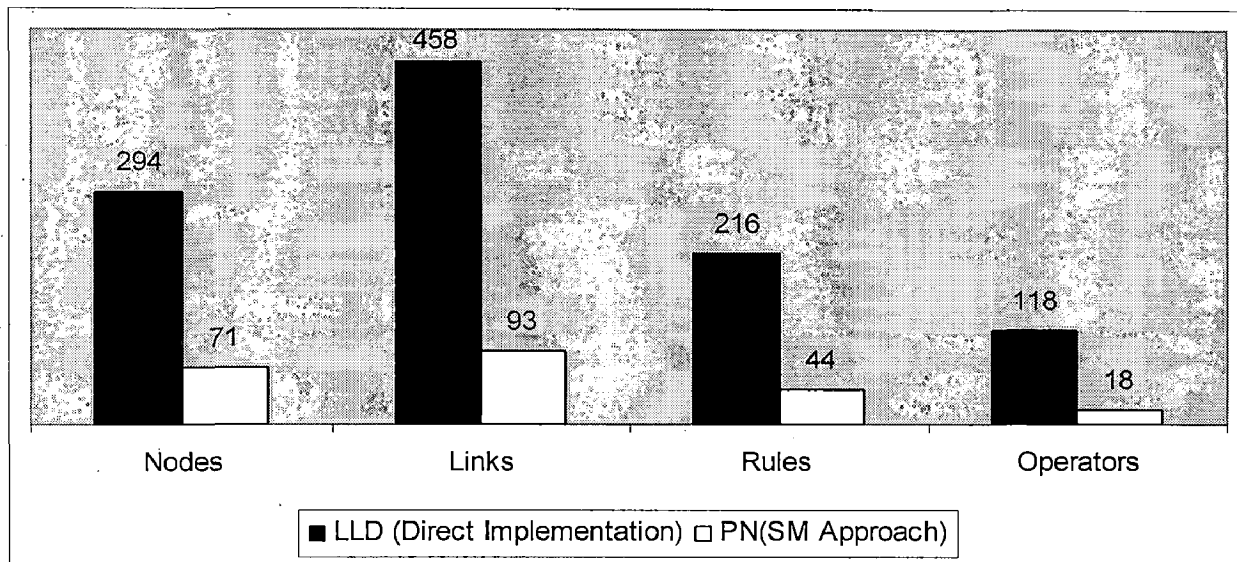


Fig 5.10.1 The comparison of PN (SM Approach) and LLD based sequence controllers for four level elevator system

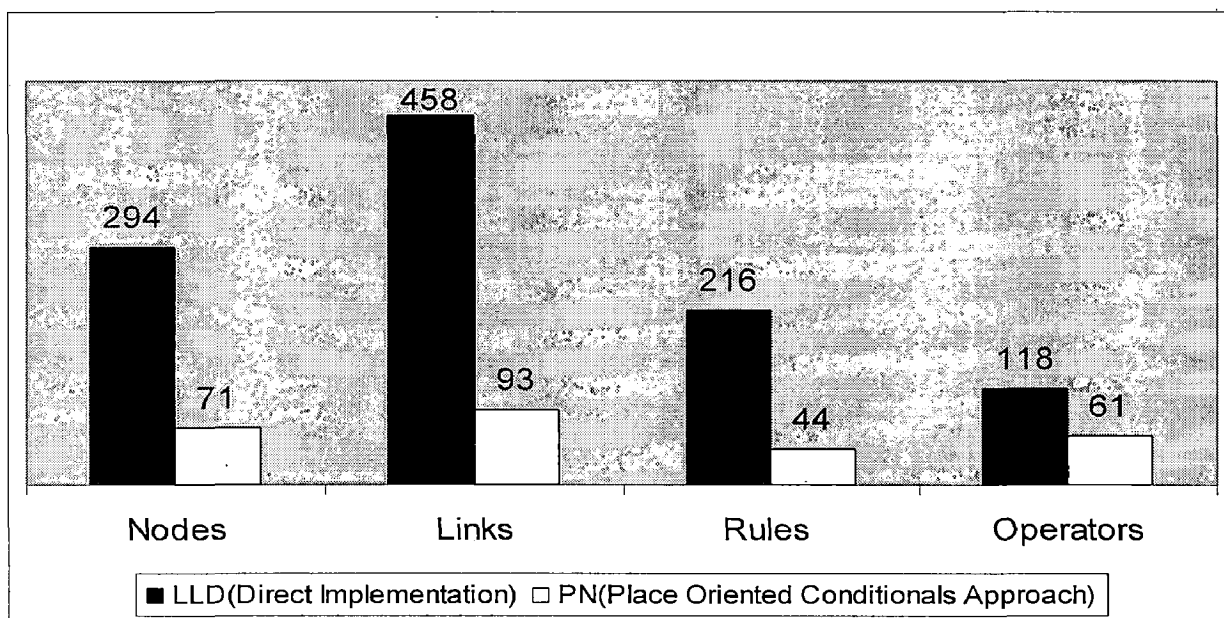


Fig 5.10.2 The comparison of PN (Place Oriented Conditionals Approach) and LLD based sequence controllers for four level elevator system

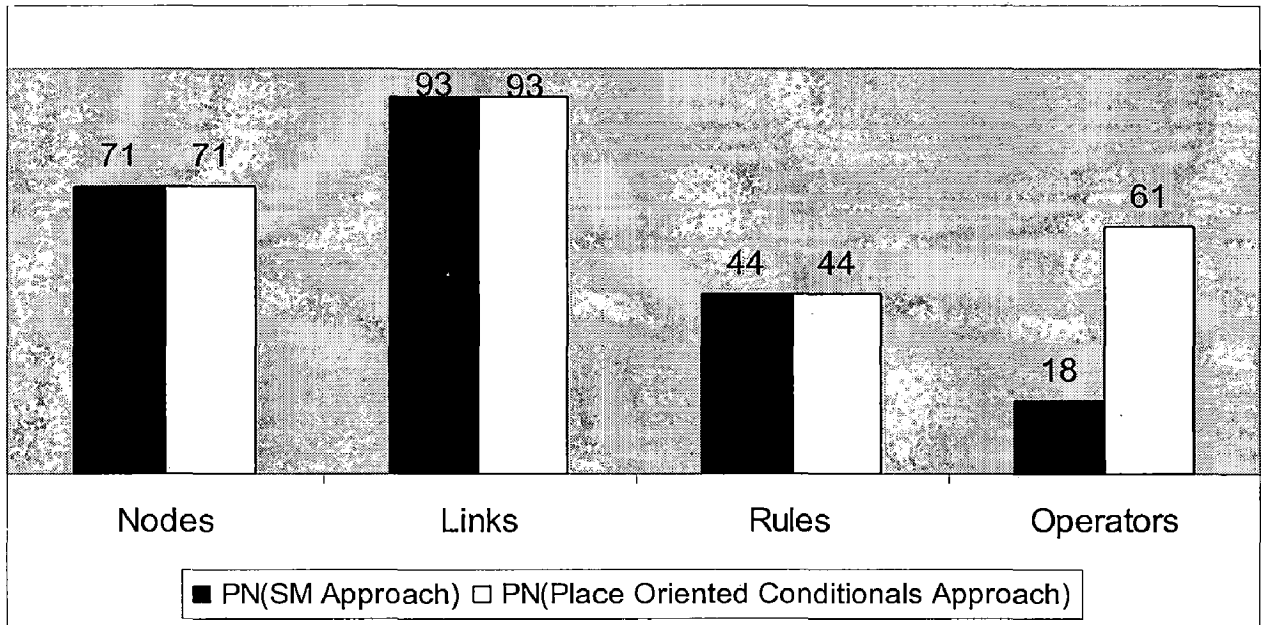


Fig 5.10.3 The comparison of PN (SM Approach) and PN (Place Oriented Conditionals Approach) based sequence controllers for four level elevator system

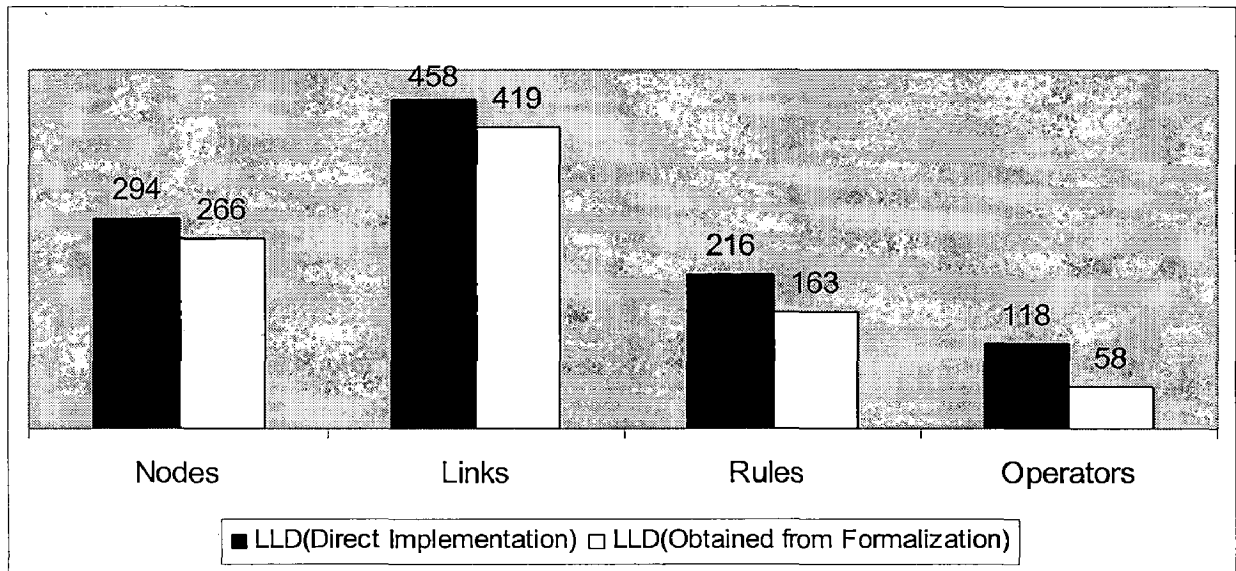


Fig 5.10.4 The Comparison of LLD (Directly Obtained) and LLD (Obtained from Formalization) based sequence controllers for four level elevator system.

Fig 5.10.1 shows the comparison of algorithms developed for four level elevator controller using PN(SM Approach) and LLD (Direct Implementation).

Fig 5.10.2 shows the comparison of algorithms developed for four level elevator controller using PN (Place Oriented Conditionals Approach) and LLD (Direct Implementation).

It is seen that in both cases of PN structures is simpler than its LLD counterpart both in terms of Basic Elements and IF-THEN rules approach.

In fig 5.10.3 both the PN algorithms are compared to show that SM approach is a better solution than Place Oriented Conditionals Approach in terms of number of operators used.

Finally, in fig 5.10.4 the LLD algorithm from PN model (SM Approach) is compared with directly implemented LLD algorithm to show that LLD algorithm obtained by means of Formalization require less elements and rules than the ones that are implemented directly.

Here, the LLD algorithm obtained by means of formalization is better in decision making than the one directly implemented, as the PN structure has FIRSTU and FIRSTD along with SECONDU and SECONDD places, it helps the LLD algorithm to decide to service upper floors of First and Second floors while going up owing to the module FIRSTU and SECONDU and lower floors of first and second floor while coming down owing to the module FIRSTD and SECONDD both of which are obtained from places FIRSTU, FIRSTD, SECONDU and SECONDD of PN algorithm.

5.5: Conclusion

It is observed that as one moves from two to three and finally to four level elevator sequence controller design, unlike LLD's the RTPN structures and LLD algorithm obtained from the RTPN structure are more adaptable to changing specifications as they require less number of basic elements and rules than the former one for implementation.

In the next chapter the increase in complexity in sequence controller design using LLD's directly and formal methods (RTPN) are discussed as one moves from two level to three level and finally to a four level elevator system.

CHAPTER 6

Results and Discussion

6.1: Results and Discussion

As per the discussion on algorithm complexity for LLD and PN in terms of graphical complexity and adaptability for change in specification along with the number of rules to implement an algorithm, fig 6.1 and 6.2 shows the complexity comparison of LLD and PN based algorithms in terms of basic elements and rules.

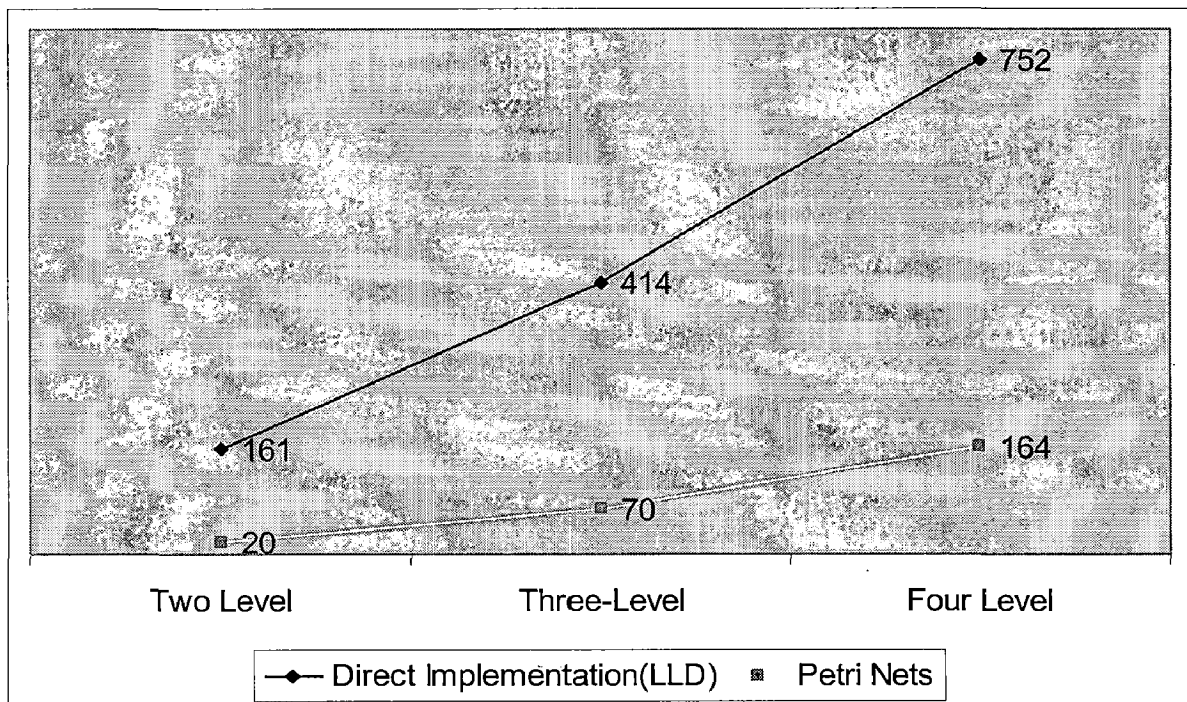


Fig 6.1 Complexity comparison of two, three and four level elevator sequence controller designed using LLD and PN Approach in terms of number of basic elements

It is observed that with change in specifications the complexity of LLD algorithms increases considerably both in terms of basic elements and sum of number of rules and operators when compared to PN based algorithms.

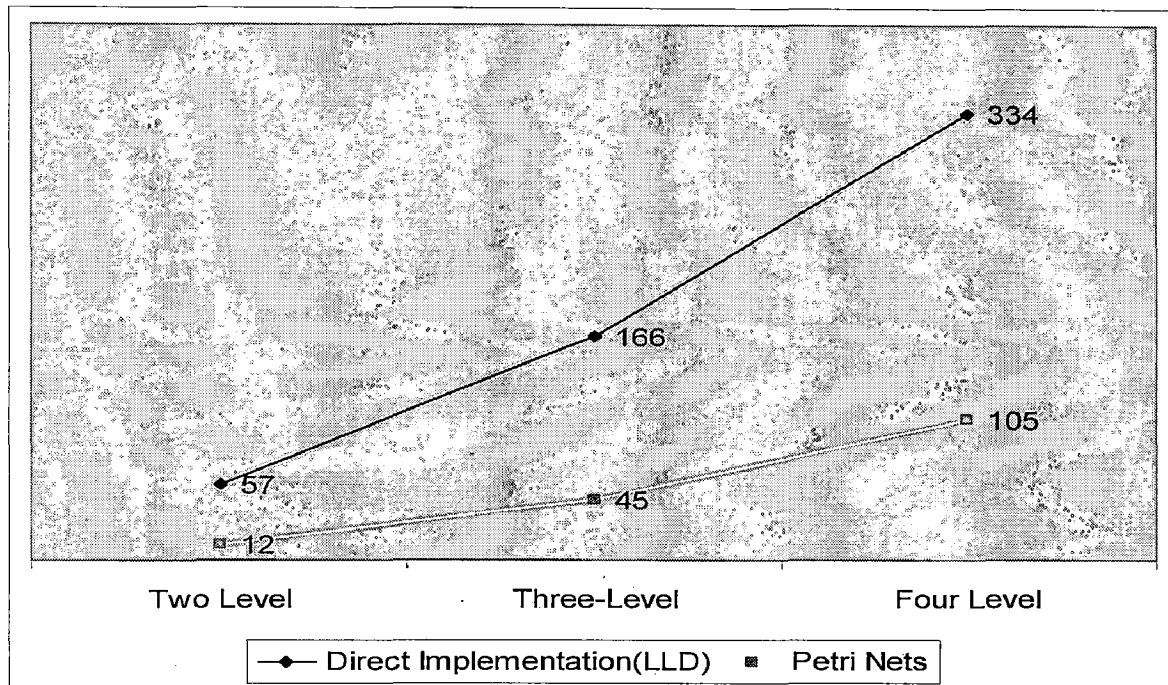


Fig 6.2 Complexity comparison of two, three and four level elevator sequence controller designed using LLD and PN Approach in terms of sum of number of rules and logical operators

Finally, the LLD algorithm obtained by means of formalization is compared with the LLD algorithm obtained directly from informal specifications and it is observed the increase in complexity in later with changing specifications is far less than the one obtained directly. Hence, it is easier to maintain, debug and read a formalized LLD algorithm as it is well structured due to its formalized design. The complexity comparison of both the algorithms is shown in fig 6.3 and fig 6.4

Note: Fig 6.3 shows comparison of complexity of both the LLD algorithms in terms number of basic elements. It is observed that the complexity of both the algorithms is quite same in terms of basic elements but the formalized LLD algorithm is superior than the one obtained directly, as it has the ability to take decisions to service floors in increasing order while shaft moves in upward direction and to service the floors in decreasing order as the shaft moves downwards due to presence of places FIRSTU, SECONDU and FIRSTD and SECONDD in their corresponding RTPN structure.

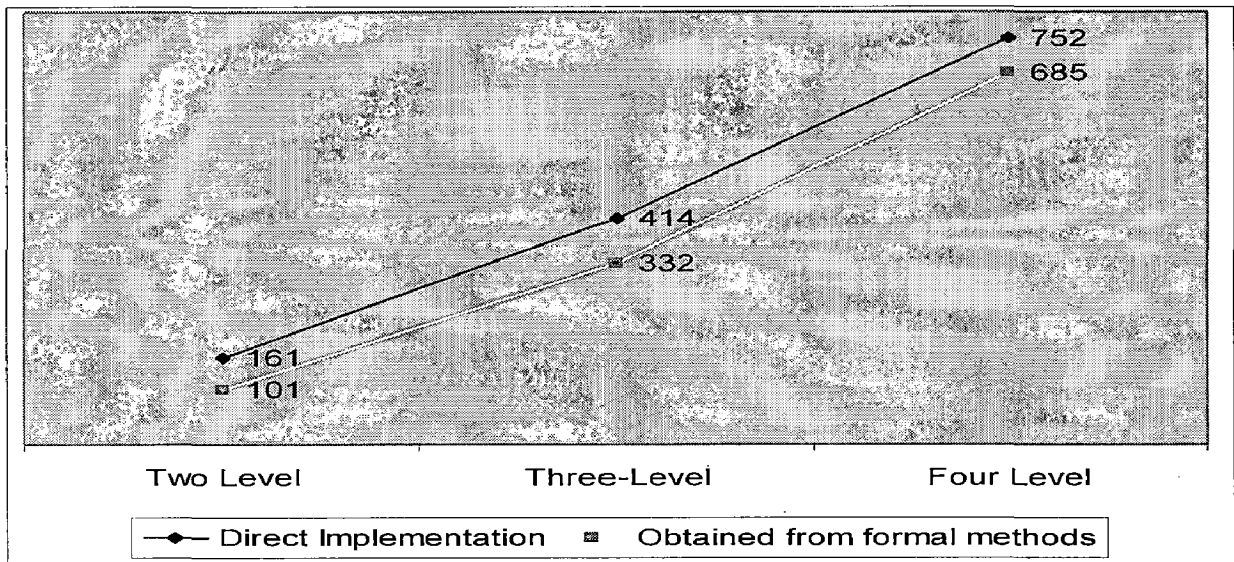


Fig 6.3 Complexity comparison of two, three and four level elevator sequence controller designed using LLD (directly implemented) and LLD (derived from formal methods) in terms of number of basic elements

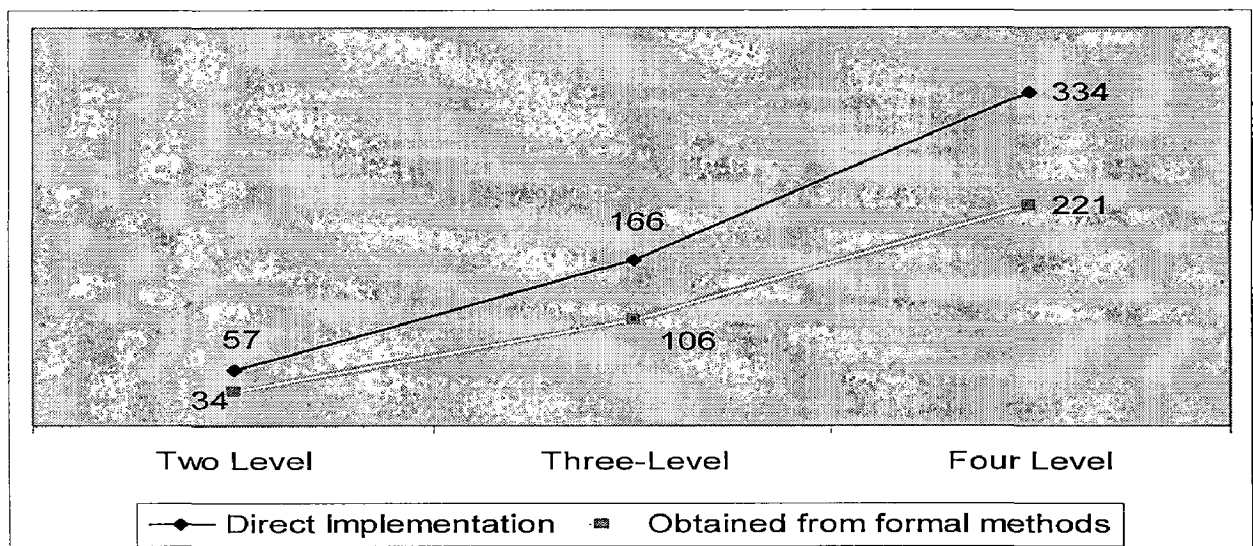


Fig 6.4 Complexity comparison of two, three and four level elevator sequence controller designed using LLD (directly implemented) and LLD (derived from formal methods) in terms of sum of number of rules and logical operators

CHAPTER 7

Conclusion and Scope for Future work

7.1: Conclusion

In this dissertation, PN based sequence controllers for discrete event systems are designed and implemented on FPGA, comparison with their LLD counterpart in terms of design complexity and rules required to implement them.

It is observed that PN based sequence controllers are less complex compared to their LLD counterpart and requires less number of rules for implementation. They are more flexible and adaptable to changing specifications and their syntactic and semantic compatibility with VHDL makes them an obvious choice for designing fast and reliable controllers on FPGA.

The difficulty with PN is that its use is limited to educational and research interests and lack of proper design tools have reduced its acceptability to the industrial arena. Further, people who are comfortable with LLD programming are not willing to shift over to PN based designs because of lack of knowledge in the field of formal methods. So, another approach is to obtain the LLD algorithm from PN specifications, which constitutes a considerable part of the work done during this dissertation. LLD algorithms are obtained by means of formalization from PN(SM approach) and it is seen that LLD algorithms so obtained are less complex, more flexible to changing specifications and are easier to debug and maintain than the ones obtained directly.

7.2: Scope for Future Work

In this dissertation PN based controllers for discrete event systems are designed and are compared with LLD based designs.

Future scope of work in this area could be development of PN based controllers for hybrid and continuous time control systems

State Machine approach of PN is used to obtain formalized LLD algorithms, one may use token pass logic to obtain the same algorithms.

Another study could be comparison of PN based algorithm designed using SM approach and Place oriented conditionals approach for systems that exhibit concurrency and then to obtain and compare LLD algorithms derived from such formalized structures.

Bibliography

- 1) G.Frey and L.Litz, "Formal methods in PLC programming", Proc. Of the IEEE SMC 2000, Vol 4, page 2431-36, 2000.
- 2) Stephane Klein, George Frey and Lothar Litz, "A Petri Net based Approach to the Development of correct Logic Controllers" <http://www.eit.uni-kl.de/litz>.
- 3) James. L. Peterson, "Petri Nets **", Proc. Of Computing Surveys Sep 1977, Vol 9, No.3, page 223-52, 1977.
- 4) Murata, Tadao, "Petri Nets: Properties, analysis and applications" Proceedings of the IEEE 1989, Vol 77, No. 4, April 1989
- 5) User's manual for VPC-PLCT Programmable Logic Controller Trainer, Vinytics Peripherals Ltd, New Delhi
- 6) *IEC 61131-3: A Standard Programming Resource*, International Electrotechnical Commission, 1993.
- 7) Johnson CD, "Process Control Instrumentation Technology New York" John Wiley and Sons, 1977.
- 8) Singh, S.K., "Computer Aided Process Control" Prentice Hall India, 2003.
- 9) Gary Dunning, "Introduction to Programmable Logic Controllers" Thomson Asia Pvt Ltd, Singapore, 2003.
- 10) Robert Carrow, "Soft Logic." Tata McGraw Hill , New York, 1998
- 11) M.Adamski, "SFC, Petri Nets and Application Specific Logic Controllers", in Proc. Of IEEE Int.Conf. On systems, Man and Cybernatics, San Diego, USA, 11-14.10.1998, pp 728-733.
- 12) M.Adamski and J.L. Monterio, " From Interpreted Petri Net Specification to Reprogrammable Logic Controller Design", in the Proc. of IEEE International symposium on Industrial Electronics, Vol 1, 2000, pp 13-19.
- 13) J.-S. Lee and P.-L. Hsu, "An Improved Evaluation of Ladder Logic Elements and Petri Nets for the Sequence Controller Design in Manufacturing Systems", International Journal of Advance Manufacturing Technology, Vol 24, no 3-4, 2004, pp 279-87.

- 14) K.Venkatesh, M.C.Zhou and R.J.Caudhill, "Comparing Ladder Logic diagrams and Petri Nets for Sequence Controller Design through a Discrete Manufacturing System" IEEE Transaction on Industrial Electronics, Vol 41, no 6, Dec 1994, pp 611-19.
- 15) M.C. Zhou and Edward Twiss, "A Comparison of Relay Ladder Logic Programming and Petri Net approach for Sequential Industrial Control Systems ", IEEE Conference on Control Applications-Proc 1985, pp 748-753.
- 16) M.Uzam, A.H. Jones and N. Ajlovni, "Conversion of Petri Net Controllers for Manufacturing Systems into Ladder Logic Diagrams ", IEEE Symposium on Emerging Technologies and Factory Automation, ETFA 96, 19996, Vol 2, pp 649-655.
- 17) M.Uzam and A.h. Jones, " Discrete Event System design using Automation Petri Net and their Ladder Design Implementation ", International Journal of Advanced Manufacturing Technology, Vol 14, no 10, 1998, pp 716-28.
- 18) Katalin Hangos and Ian Cameron, "Process Modeling and Model Analysis" Process System engineering, Vol 4, Academic Press, New York ,2001.
- 19) I.Miyazawa, H.Tanaka and T.sekiguchi, " Verification of the Behavior of Sequential Function Chart Based on its Petri Net Model", IEEE Symposium on Emerging Technologies and Factory Automation, ETFA 1997, pp 532-37.
- 20) R.Zurawski and M.C.Zhou,"Petri Nets and Industrial Applications: A Tutorial", IEEE Transaction on Industrial Electronics, Vol1.41, N0.6. December 1994, pp567-83.
- 21) J.L. Peterson," Petri net theory and the modeling of systems", Prentice Hall Inc, 1981.
- 22) W. Reisig , "Petri Nets", Springer Verlag ,1985.
- 23) www.plcopen.org
- 24) I.G Warnock,"Programmable Controllers: Operation and Application", Prentice Hall International (UK), 1988.
- 25) J-L Chirn and D.C. McFarlene," Petri Net based Design of Ladder Logic Diagrams", UKACC International Conference on control 2000, 2000, pp 1-6.
- 26) E.Soto and M.Pereira, "Implementing a Petri Net Specification on FPGA using VHDL", Proc. of the International workshop on Discrete-Event System design, 2001, pp 19-24.

- 27) K.Venkatesh, M.C. Zhou and R.Caudill, "Design of sequence Controllers using Petri Net Models", IEEE International Conference on System, Man and Cybernetics. Intelligent systems for the 21st Century,(Cat No.95ch3576-1),pt.4, Vol 4,1994, pp3469-74.
- 28) J-S Lee and P-s Hsu, "A PLC -Based Design of Sequence Controllers using Petri Net Models", Proc. of the 2000 IEEE International Conference on Control Applications, Vol 1, 2000, pp 929-34.
- 29) S.G.Tzafestas, A.G. Pantelesis, and D.L. Kostis, "Design and Implementation of a Logic Controller using Petri Nets and Ladder Logic Diagrams", Manufacturing, Modeling, Management and Control (MMM 2000) Proc. Volume from the IFAC Symposium, 2001, pp 333-38.
- 30) D. Crockett, A. Desrochers, F.Dicesare and T.Ward, "Implementation of a Petri Net Controller for Machining Workstation", Proc. of the 1987 IEEE International Conference on Robotics and Automation(Cat. No.87CH2413-3), Vol 3, 1987, pp 1861-67.
- 31) T.O. Boucher, M.A. Jafari and G.A. Meredith, "Petri Net Control of an Automated Manufacturing Cell", Computers and Industrial Engineering, Vol 17, 1989,pp 459-63.
- 32) M.C. Zhou and Shishen Peng, "A Synopsis of Petri Net Classes for Manufacturing Systems", 2002 IEEE International Conference on Systems, Man and Cybernetics conference Proceedings(Cat No. 02CH37349), Vol 6, pt.6,2002,pp 203-208.
- 33) J.M. Fernandes, M.Adamski and A.J. Porenca, " VHDL Generation from Hierarchical Specification of Parallel Controllers", IEEE Proceedings- Computers and Digital Techniques, Vol 144, No.2, March 1992, pp 461-473..
- 34) M.A. Adamski and J.L. Monteiro, "Rule-Based Formal specification and Implementation of Logic Controller Programs", IEEE International Symposium on Industrial Electronics, Vol 2, 1995, pp 700-705.
- 35) M.A. Adamski and J.L. Monterio, "PLD Implementation of Logic controllers", Proc. of the IEEE International Symposium on Industrial Electronics, Vol 2, pt.2, 1995, pp 706-11.

- 36) S.Brown and Z.Vranesic, "Fundamentals of Digital Logic with VHDL DEsign", Tata Mcgraw Hill Publishing Ltd., New Delhi, 2005.
- 37) G.Frey and L.Litz, "Verification and Validation of Control Algorithms by coupling of Interpreted Petri Nets", Proc. of the IEEE International Conference on Systems, Man and Cybernatics, Vol 1, 1998, pp 7-12.
- 38) www.altera.com

APPENDIX A

Hardware and Software Tools Used

A1: Vinytics Peripherals Ltd- An Introduction

Vinytics Peripherals was founded by a team of young engineers in 1981 has since then, been developing microprocessor, PC based products for Educational Training, Measurement, Testing and Automation Applications .Celebrating a decade of impeccable service to customers, *Vinytics* continues their tradition of consistently providing diversified and market rich solutions to a wide array of Educational, Industrial and R & D Applications. With the know-how and the expertise gained, they have designed & produced a comprehensive range of State-of-the-Art products to solution to meet the various demanding Industrial Applications. *Vinytics* Microprocessors & PC based products are deployed in many different environments: Engineering, Science Colleges, Polytechnics, Test & Measurement, Process Control, Industrial Automation, Medical, Instrumentation, Intelligent Graphic Systems, Research Laboratories, Supervisory Control & Data Acquisition, Telecommunication & Production facilities.

A2: VPL-PLCT: PLC Demonstration Trainer

PLC trainer has been designed in a different way than conventional PLC suitable to impart training and use of PLC in process industry for students at all levels with:

- 16 inputs switches and 16 output LED's to simulate the ladder program.
- PLC assembler to generate ladder programs.
- ZIF on front panel to load programs from EPROM and adaptor.
- Descriptive user manual containing description and fundamentals of PLC.

A.2.1: Specification

- Microcontroller based modular PLC trainer.
- PLC trainer consisting of main unit, hand held programming unit.
- 16 input signals (24V) and 16 outputs for controlling the process on D-type sub connector.
- Programming Unit: Hand held model with 28 keys and 6 digit display.

- Static process control simulator board consisting of switches and 16 LED's for input simulation and 16 LED's for output indications respectively.
- RAM module with battery backup to store the programs.
- RS232C port available for uploading and downloading of files from PC.
- Powerful commands like AND, OR, ANI, ORI,SET, RST,LD, LDI, OUT, ANB, ORB, IL,ILC etc directly executable from hand held programmer.
- Execution of programmes possible even without the hand held unit.
- Standard EPROM module which is affixed on ZIF socket to store programmes.
- Main unit and hand held key pad are housed in attractive metallic unit.
- Operating voltage 220V, 50 Hz AC, with drift of 10% at 50 deg centigrade.

A.2.2:The Hand Held Programmer Unit

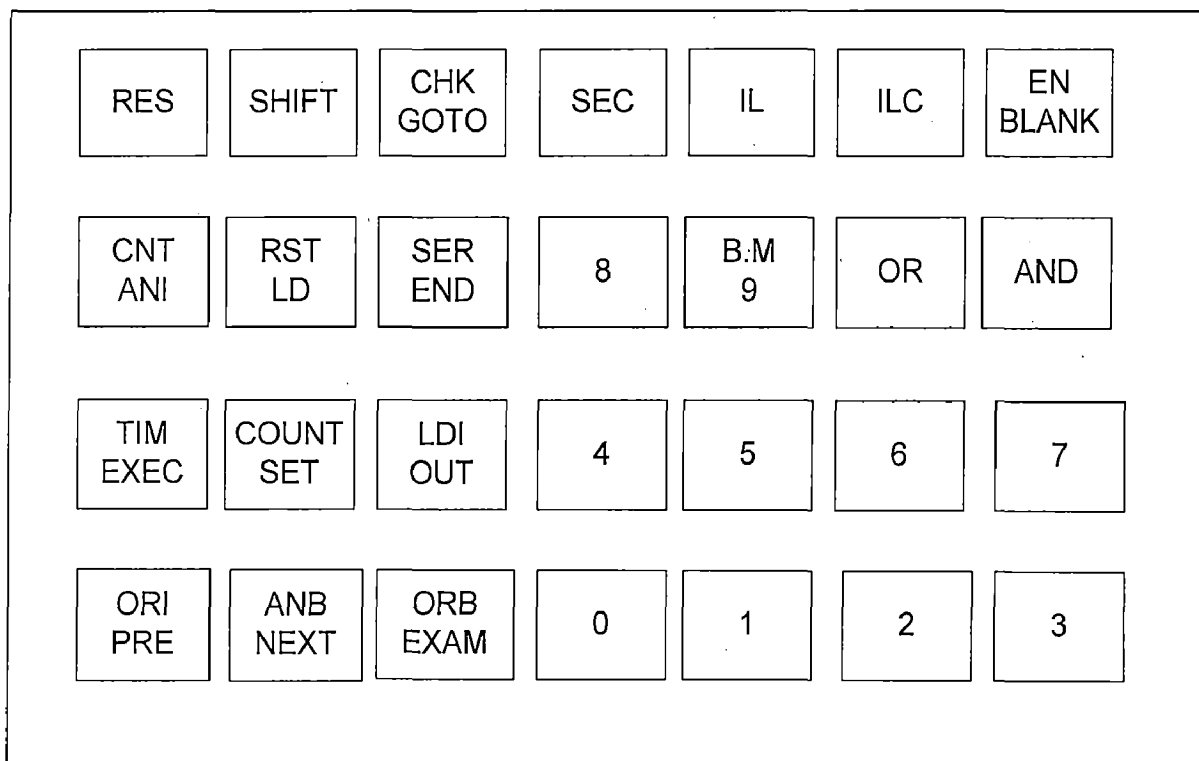


Fig A.1 Front View of Hand Held Programmer

Fig A.1 shows the 28 keys available for PLC trainer programming on VPLC hand held programmer.

A.2.3:Command Description

RESET: This key initializes the PLC Trainer.

CHK: This key checks the status of port address only for high stage and jumps to the line number as state in example below, and if it is in the lower stage it goes to the next line.

```
Ex: LD 01h
    chk 01h
    goto rite
    goto wrong
```

In the above example if port 01 status is high, the program jumps to that part of execution starting with label rite else it goes to the part starting with label wrong.

GOTO: This key is used to goto line number as required.

CNT: This key enables the counter and defines the port address. Sixteen counters can be defined from port addresses 80 to 87 and 90 to 97.

ANI: This key is used to implement the function AND inverse.

RST: This key resets the input.

LD: This key loads the inputs. PLC trainer has 16 digital inputs which can be defined from port addresses 00 to 07 and 10 to 17.

SER: This key is used to interface the PLC trainer with RS-232C port (serial port) of the computer for serial communication at a baud rate of 4800.

END: This key used in the end of every logic implementation.

TIM: This key is pressed when timer is required in the logic implementation and the port addresses are defined from 60 to 67 and 70 to 77.

EXEC: This command is used to execute the program/logic written at memory locations.

COUNT: This key is pressed to define the delay when timer (TIM) and counter (CNT) are used. The symbolic representation of count is #. In case of timer the number following the # is the multiplier. The number should be multiplied by 0.1 to get the delay after which the time relay goes high.

SET: This key is used to set the output as required in the logic. The output which is set by pressing this key will remain in the same state till reset key is pressed.

LDI: This command key is used to load the inverse of inputs.

OUT: This command key is used when outputs are required in the logics. IN VPLC-PLCT we have 16 logical outputs defined from port addresses 20 to 27 and 30 to 37.

ORI: This command key is used to OR inverse which is required in most of the logic circuits.

ANB: This key is pressed when the function AND block is required.

ORB: This key is pressed when the function command OR block is required.

EXAM/NEXT/PRE: This command key is pressed to examine the contents of any memory location. After pressing this key the contents of first memory location will be displayed this can further be examined through the NEXT key and the instructions in the previous locations through PRE key.

IL: This command key is pressed when the logic needs to be inter-locked which means that the instructions which are further written are not executed as long as this key is defined for particular line number.

ILC: This command key is pressed when the interlocks needs to be cleared which means that the instructions which are written are executed totally as the interlock is cleared.

EN: This key is used to enable the counter.

BM: This key is used to move a memory block from the EPROM to the RAM area.

OR: This function key is used when ORing is required in logic.

AND: This function key is used when ANDing is required in the logic.

The fig A.2 shows the Elevator controller card used to interface the four level Elevator setup to the VPL-PLC trainer kit.

A.3: ELEVATOR CONTROLLER MODULE

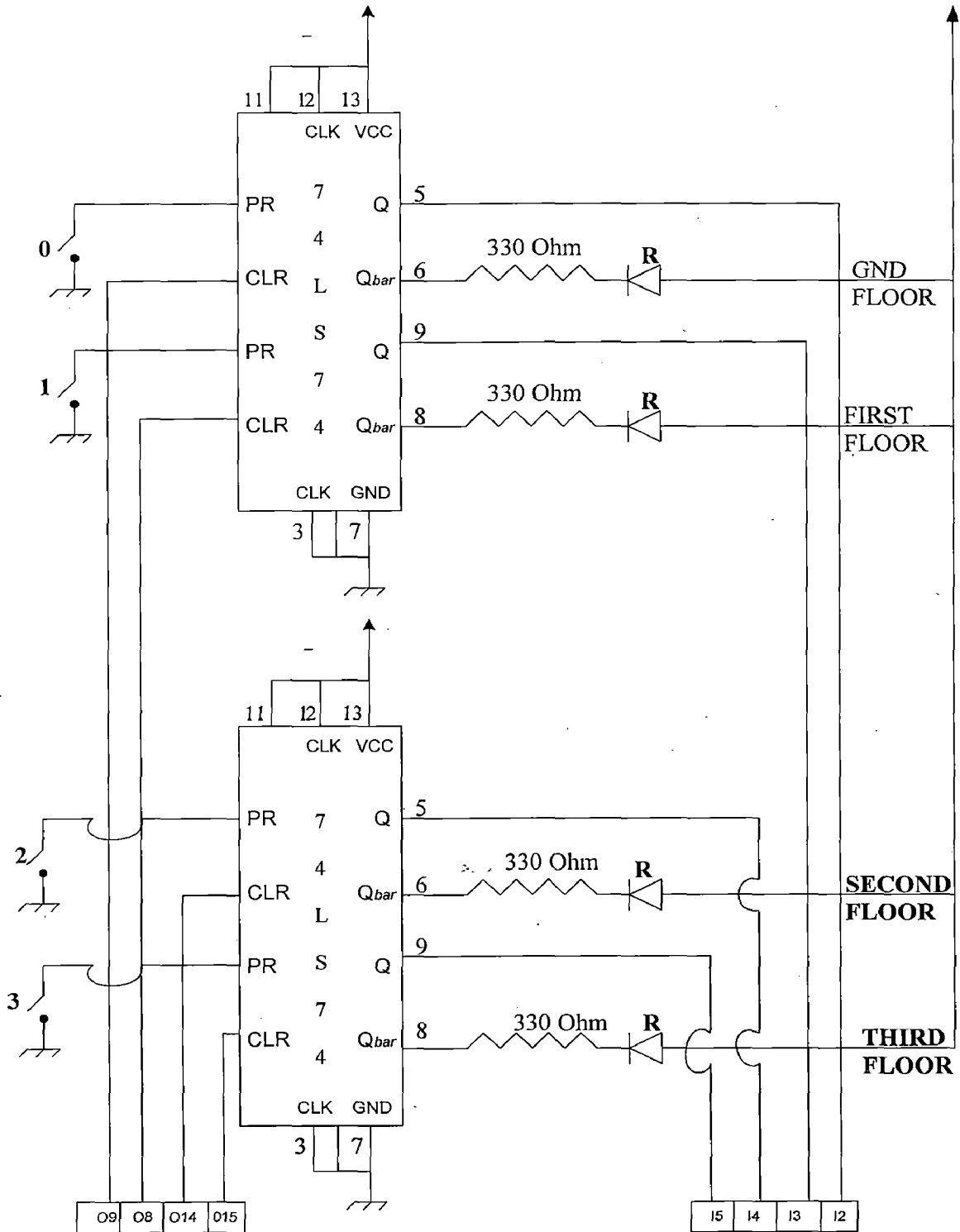
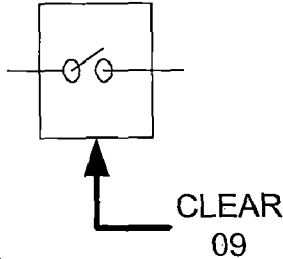
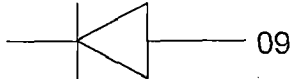
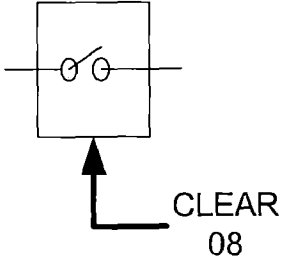

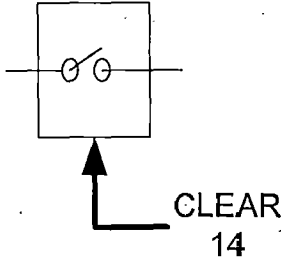

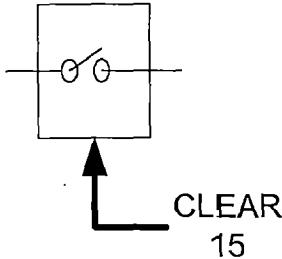
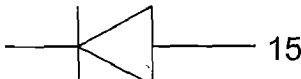


Fig A.2 The Elevator Controller Card

A.3.1: Elevator Controller Module Reference

Floor	Request Switch	Lift Position	Condition of PLC switch
GROUND FLOOR			ALL
FIRST FLOOR			INPUT
SECOND FLOOR			SWITCHES
THIRD FLOOR			HIGH

A.3.2: Module Description

Elevator controller has one motor which is controlled by output port 02 of PLC trainer kit. The UP/DOWN motion is controlled by output port 03 i.e. when 03 is ON the stepper motor carries the lift towards upward direction and when OFF it carries the elevator towards downward direction.

The request for each floor is received on input ports I2, I3, I4 and I5, where the lift position and clear is generated by O9, O8, O14 and O15. The details of the interfacing are tabulated below in table A.1:

Table A.1: The I/O Description of the Elevator Controller Card

FLOOR POSITION	INPUT	OUTPUT
GROUND	I2	O9
FIRST	I3	O8
SECOND	I4	O14
THIRD	I5	O15
MOTOR ON	NIL	O2
MOTOR UP	NIL	O3 ON
MOTOR DOWN	NIL	O3 OFF

The two flip-flops shown above only work on their Preset and signal as tabulated below:

SET	PRESET	Q	Q _{bar}	CLOCK
0	0	H*	H*	X
0	1	1	0	X
1	0	0	1	X

* This condition is unstable, that it will not persist when either the preset and/or clear return to their inactive level.

In the elevator Controller module the request switches for different floors are wired to the Preset inputs of four flip-flops, two in each 74LS74 IC. The Q of each flip-flop is received at the input ports i.e. I2-I5 of VPLC trainer. The Clear Signal of the flip-flops is generated on output ports O8, O9, O14, O15 of VPLC trainer.

A.4:Details of OMRON PLC Simulator

The PLC simulator mimics the OMRON Industrial PLC with the following supported ladder code: LD, OR, AND, CNT, TIM, BLK, DIFU, SFT. It has 16 inputs, 16 outputs, 20000 internal relays, 8 counters and 8 timers.

Possible combination of ladder codes:

LD, LD NOT

LD TIM, LD NOT TIM

LD CNT, LD NOT CNT

OR, OR BLK, OR TIM, OR CNT

OR NOT, OR NOT BLK, OR NOT TIM, OR NOT CNT

AND, AND BLK, AND TIM, AND CNT

AND NOT, AND BLK, AND NOT TIM, AND NOT CNT

OUT, OUT NOT, DIFU, SFT

The features of the PLC Simulator are

- * Ease of ladder code editing.
- * Near real-time timing and counting capabilities.
- * User friendly interface.

- * Ladder code can be saved into files.
- * Easy to use 'click INPUT'.
- * LED indicator for both inputs and outputs.

The details of XTALK software for communication between VPLC kit and PC is given in Appendix A.

A.5:THE UP3 KIT

A.5.1:General description

The UP3 Education Kit provides a powerful educational support and also a low-cost solution for prototyping and rapidly developing products. The board serves as an excellent means for system prototyping, emulation and hardware as well as software development. The board comes with a powerful Altera Cyclone FPGA. It gives scope to a hardware design engineer to design, prototype and test IP cores or any hardware design using HDLs like Verilog or VHDL. The entire environment helps to quickly implement any processor as well as any real time operating system on the kit. Along with that one can simulate and test 'C' or assembly code also. The board has industry standard interconnections, Memory Subsystem, Multiple clocks for system design, JTAG Configuration, expansion headers for greater flexibility and capacity and additional user interface features. The board can be used for DSP applications by interfacing directly to a DSP processor or implementing DSP functions inside the FPGA. In short, it is a dual-purpose kit, which can be used for prototyping and developing VLSI designs as well as designing and developing microprocessor based embedded system designs. [38]

A.5.2:Features

The following are some of the features of the ESDK Board.

- ✓ Features an Altera EP1C6Q240 Device and EPCS1 configuration device
- ✓ Supports intellectual property based (IP-Based) design both with and without a microprocessor
- ✓ USB 1.1 (Full speed & Low speed)
- ✓ RS 232 Port
- ✓ Parallel port (IEEE 1284)
- ✓ PS/2 Port

- ✓ VGA port
- ✓ IDE (Integrated Drive Electronics)
- ✓ 2KBytes of I2C PROM (Expandable)
- ✓ 128KBytes of SRAM
- ✓ 2MBytes of FLASH
- ✓ 8MByte SDRAM
- ✓ Supports multiple clocks like CPU clock, USB clock, PCI clock, and IOAPIC clock.
- ✓ JTAG and Active Serial download capability
- ✓ 5V Santa Cruz long Expansion Card Header provides 72 I/O for the development of additional boards providing various functionalities
- ✓ One user definable 4-bit switch block
- ✓ Four user definable push button switches, and one global reset switch
- ✓ Four user definable LEDs
- ✓ I2C Real Time Clock
- ✓ One 16X2 character LCD Module

A.5.3: The Cyclone EP1C6Q240 Device

U11 is an ALTERA Cyclone EP1C6Q240 in a 240-pin in a PQFP.

FPGA uses SRAM cells to store configuration data. Since SRAM memory is volatile configuration data must be downloaded to Cyclone FPGA each time the device powers up. There are three methods to configure the device Active serial configuration, Passive serial configuration, JTAG- based configuration. The UP3 board supports two modes.

Active Serial Mode

Active serial configuration is carried out through serial configuration device EPCS1.

Serial configuration devices provide a serial interface to access configuration data. During device configuration, Cyclone FPGA read configuration data via the serial interface, decompresses data if necessary, and program their SRAM cells. This scheme is referred to as an AS configuration scheme because the FPGA controls the configuration interface.

JTAG Mode

JTAG (Joint Test Action Group) interface. JTAG has developed a specification for boundary-scan testing. This boundary-scan test (BST) architecture offers the capability to efficiently test components on printed circuit boards (PCBs) with tight lead spacing. The BST architecture can test pin connections without using physical test probes and capture functional data while a device is operating normally. The user can also use the JTAG circuitry to shift configuration data into Cyclone FPGA. The Quartus II software automatically generates .sof files that can be downloaded using Byte-Blaster II for JTAG configuration. Cyclone is designed such that JTAG instructions have precedence over any device operating modes. So JTAG configuration can take place without waiting for other configuration to complete (e.g., configuration with serial or enhanced configuration devices). If the user attempt JTAG configuration in Cyclone FPGA during non-JTAG configuration, non- JTAG configuration will be terminated and JTAG configuration will be initiated.

Flash Memory Device

U8 is a 2Mbyte of Flash memory connected to the Cyclone device. The U8 is a 16,777,216-bit, 3.0-V read-only electrically erasable and programmable flash memory organized as 2,097,152 words x 8 bits or as 1,048,576 words x 16 bits. The U8 features commands for Read, Program and Erase operations to allow easy interfacing with microprocessors. The Program and Erase operations are automatically executed in the chip.

SRAM Device

U7 is the 128KBytes asynchronous SRAM device. It is a high speed, 1,048,576-bit static RAM organized as 65,536 words by 16 bits. It is fabricated using the ISSI's high performance CMOS technology. This highly reliable process coupled with innovative circuit design techniques; yields access times as fast as 10 ns with low power consumption.

SDRAM Device

U6 is a 8MByte Synchronous Dynamic RAM. It is organized as 1,048,576 bits X 16-bit X 4-bank for improved performance. The synchronous DRAMs achieve high speed data transfer using pipeline architecture. All the input and output signals refer to the rising edge of the clock input..

Liquid Crystal Display

U1 is a 16X2 character Liquid Crystal Display. Here 16X2 represents 2 display lines with 16 characters per line. The display contains 2 internal byte wide registers, one for the command and second for characters to be displayed. It also contains user programmed RAM area that can be programmed to generate any desired character that can be formed using a dot matrix.

Serial Port Connector

SER2 is the standard DB-9 Serial connector. It has all 9-pin connections to the FPGA, a FULL Modem interface. This connector is typically used for communication with a host computer using a standard serial cable connected to (for example) a COM port. U21 is a level translator for interfacing the SER2, Full Modem serial port, with the FPGA.

USB

USB is a cable bus that supports data exchange between a host computer and a wide range of simultaneously accessible peripherals. The attached peripherals share USB bandwidth through a host detached while the host and other peripherals are in operation.

PS/2 Connector

JP1 is a PS/2 Connector. The PS/2 interface allows the connectivity to a PS/2 device. The connector is a female 6-pin mini din type.

Parallel Port

CON1 is a standard DB25 Female parallel port connector.

VGA Port

UP3 board has a standard VGA connector. It contains 5 active signals. Two signals compatible with TTL logic levels, horizontal sync and vertical sync, are used for synchronization of the video. Three analog signals with 0.7 to 1.0 volts peak-to-peak levels are used to control the color.

Push Button Switches

SW4, SW5, SW6 and SW7 are momentary-contact push-button switches and are used to provide stimulus to designs in the Cyclone device. Each switch is connected to the Cyclone general-purpose I/O pin with pull-up resistor. The Cyclone device pin will see logic '0' when each switch is pressed. SW8 is a global reset switch connected to the RESET IC. The RESET IC pin RESETIN# will see logic '0' when pressed.

Dip Switches

SW3 is a block of four switches. Each switch is connected to the Cyclone general-purpose I/O pin with pull-up resistor. The Cyclone device pin will see logic '1' when switch is in ON condition.

LEDs

D3, D4, D5 and D6 are four individual LEDs connected to the Cyclone device general purpose I/O with current limiting resistors. All of them are active low driven (Common Anode). The LED will glow when there is logic '0' at FPGA pin. D8 is Configuration Done LED that indicates successful completion of downloading process. The CONFIG_DONE pin (U11.145) of the Cyclone device controls this LED. D15 is INVALID indication LED that indicates faulty/no connection of the serial cable at the serial port (SER2). If invalid voltage appears at any receive lines of the MAX 3243 chip (U21) the LED will glow. U21.21 pin controls this LED.

Power Supply Circuitry

The UP3 board is powered with number of different regulated supply voltages.

1. +1.5 Volt is for Cyclone core supply.
2. +3.3 Volt is for Cyclone I/O ring supply.
3. +5 Volt is for 5volt operative devices on the board.

The board accepts +9 Volt unregulated/regulated supply from external source (with center-terminal positive supply).

A.5.4: Altera NIOS processor

The NIOS embedded processor is a soft-core processor designed and developed by Altera Corporation. The Nios soft-core processor is a general purpose 16/32 bit RISC processor that can be implemented on Altera PLDs and FPGAs. The NIOS processor is designed to incorporate both the scalable and flexible features of soft-core processor.

The general NIOS architecture is quite simple and could be used both as an embedded processor and as a complex controller. The maximum address range is of 4 G bytes and it has a five stage pipeline enabling one instruction to execute per clock cycle. There are upto 512 general purpose registers available with a window size of 32 registers. The NIOS processor is rated for a maximum speed of 80 MHz for its 32 bit core version. Its 16 bit

version is rated for a maximum speed of 50 MHz. NIOS has a fixed length 16 bit instruction set.

A.6: Altera Quartus II Design Software

The Altera Quartus II design software design software technology leadership gives us unmatched levels of performance and ease of use. It provides a complete multi-platform design environment that easily adapts to our specific design needs. The Quartus II software also allows us to use the Quartus II graphical user interface, EDA tool interface, or command-line interface for each phase of the design flow.

The basics of FPGA are provided in Appendix B.

A.7: XTALK AS A PC INTERFACE SOFTWARE

Vinytics PLC-trainer though is a stand alone model; it has an RS 232C port to communicate with PC for uploading and downloading files for further execution. The procedure to connect PC to VPLC-trainer is given below:

- Connect +5V, GND to power supply.
- Connect the serial cable to the PC.
- Switch ON the power supply. Press RESET key and then SER key of the key pad.

Serial Number	Device to be Connected	Interface at Connector	Connection Details
1	PC/XT/AT	RS-232C	PIN NO.2-TX+ PIN NO.3-RX+ PIN NO.5-GND

A.7.1: Introduction to XTALK Software

XTALK is simple emulator software for IBM-PC/XT/AT compatible computers which allows the user to communicate with the computer through serial port with the facility of downloading and uploading the data between the computer and other serial device. The various communication parameters like baud rate (speed), number of data bits, stop bits,

parity etc. can be changed. The package communicates through COM1: as well as COM2: port of the IBM-PC/XT/AT system.

A.7.2: Installation of XTALK Software

- i) Install the diskette containing XTALK in A-drive.
- ii) Make a directory in the name of XTALK in your hard drive.
- iii) Copy the diskette in given directory.
- iv) Execute XTALK as

XTALK<CR>

A.7.3:Section of Communication Parameters

For setting the parameters, press <HOME> key. The system will show a **COMMAND?** Prompt which means the system is asking users user for commands. The different parameters of XTALK are set as per the table given below for successful communication between PC and VPLC-Trainer.

Parameters to Set	Implication	Commands to give in XTALK mode
SPEED	4800 BAUD RATE	SP 4800
DUPLEX MODE	FULL	DU FULL
PARITY	NONE	PA NONE
STOP BITS	1	ST 1
DATA BITS	8	DA 8
CWAIT	DELAY 20	CW 20
LWAIT	DELAY 20	LW 20
EMULATION	NONE	EM NONE

Next write **GOLO** in **COMMAND** prompt and press. <CR>

Press **SER** key of VPLC-Trainer and press <CR> of the PC keyboard.

A dot will be displayed on the monitor of screen of PC which means that PC and PLC trainer are now linked for communication for uploading and downloading files.

A.7.3.1: Upload

- XTALK provides a feature by which the data coming in the serial link of PC/XT/AT can be stored in hard drive. This is achieved by the following instructions as shown below:
- Press 'U' of PC keyboard.

- Press Home Key, **COMMAND?** will be displayed in the 25th line.
- Write **CAON** and press <CR> and again press <CR> three times.
- Again press HOME key.
- Write **CA OFF** and press<CR>.
- The following message will be displayed on computer screen
- **“Information is still in the capture. Do you want to save it? (Y/N)”**
- After pressing ‘Y’ the command displayed will be **“Write capture buffer to what file?”** Please enter the file name with .dat extension.
- After pressing <CR> on the screen the command displayed will be **“File successfully written, Press <Enter>.”**
- Data captured will be stored in the file with given filename in hard disk.

A.7.3.2: Download

The following is procedure is to be adapted for downloading a file from PC/XT/AT to VPC-PLCT.

The ‘D’ command loads the data from PC to memory of the PLC trainer.

- Press ‘D’ key of the PC keyboard.
- Before pressing <CR> key press ‘HOME key’ will appear.
- Name the file that needs to be transferred from PC to memory of VPLC-trainer.
- Write **SEND ***.dat** in command prompt and press<CR> key twice.

The program stored in file will be stored in memory of PLC trainer as the address specified in program.

A.7.3.3:Aborting the Operation

- Press HOME key of computer keyboard **“COMMAND?”** will be displayed.
- Type **“QUIT”** and press ENTER to exit.

A.8: INTRODUCTION TO PLC 51

The **PLC 51** assembler enables the user to write programmes on any PC, PC/XT, PC/AT machines which can then be assembled into a relocatable code. This generates an Intel HEX format file and a documentation file.

A.8.1: ASSEMBLER DIRECTIVES

a) **ORIGIN:** Set the program assembly address:

ORG EXP

EXP- Has to be a decimal or hex format.

Example: **ORG 02000H**

The program will have origin 02000 hexadecimal, address starting with A, B, C, D, E, F are to be entered with 0 <address> just to be distinguished between label and address. The PLC trainer has fixed origin as 2000H.

b) **END**

End defines the end of the program. Every program has an END statement.

c) **LABEL**

At any line in the program, label could be entered. The label is followed by a colon and can have maximum length of seven characters.

d) **COMMENTS**

The comments line or comments within a line are to be started of with a semi colon; any entry made after this will be treated as comments.

e) **ASSEMBLY**

The program entered as per the directions given above can be assembled using the **PLC51.COM**. This file will generate the **XXX.PRN** and **XXX.HEX** files from the **XXX.ASM** file.

XXX.PRN is a list file in document form and contains all labels, comments, disassembly in proper format.

XXX.HEX is an **INTEL HEX** format file containing the address and code as per the program. This file has to be downloaded by programmer from **PLC** to **VPLC-Trainer**.



2002-00

W
O
L
F
I
N
F
A
N
T
M

FEED

2012/03

INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE - 247 667

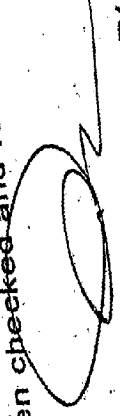
Date..... 28/6/07

No. ACD/ 09 / A-184

Prof. & Head FEED

..... 3 copies of dissertation
for the degree of M. Tech. / M. Arch. / M. U. R. P. / M. Phil. in the field of
Submitted by Shri/ Ms. S. SUMY JOYATI M. S. B. R.

Enclosed please find herewith
2605-09 for further necessary action at your end please.
(Batch)



Copies of dissertation of above student have been checked and found correct.

For ASSTT. REGISTRAR(ACD.)

Encl. : As Above

APPENDIX B

FPGA

The FPGA is an integrated circuit that contains numerous (over 10,000) identical logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. The individual cells are interconnected by a programmable interconnect (matrix of wires and programmable switches). A user's logic design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the programmable interconnect matrix. The cell's combinatorial logic is physically implemented as a small look-up table memory (LUT) or as a set of multiplexers and gates. LUT devices tend to be a bit more flexible and provide more inputs per cell than multiplexer cells at the expense of propagation delay. The array of logic cells and interconnects form a fabric of basic building blocks for logic circuits (also named as Logic elements - LE). Complex designs are formed by combining these Logic elements to build the desired circuit. Field Programmable means that the FPGA's function is defined by a user's program rather than by the manufacturer of the device. A typical integrated circuit performs a particular function defined at the time of manufacture. In contrast, a program written by someone other than the device manufacturer defines the FPGA's function. Depending on the particular device, the program is either 'burned' in permanently or semi-permanently as part of a board assembly process, or is loaded from an external memory each time the device is powered up. This user programmability gives the user access to complex integrated designs without the high engineering costs associated with application specific integrated circuits.

[38]

Basic Principle

Figure B-1 shows an example of a logic block consisting of a 3-LUT, and a flip-flop. An 8-to-1 multiplexer in a LUT is implemented using 2-to-1 multiplexers. Therefore, the propagation delay from inputs to the output is not the same for all the inputs. Input IN 1 experiences the shortest propagation delay, because the signal passes through fewer multiplexers than signals IN 2 and IN 3. Since a LUT can implement any function of its input variables, inputs to the LUTs should be mapped in such a way that the signals on a critical path pass through as few multiplexers as possible. Logic blocks also include a flip-

flop to allow the implementation of sequential logic. An additional multiplexer is used to select between the LUT and the flip-flop output. Logic blocks in modern FPGA's are usually more complex than the one presented here. Each logic block can implement only small functions of several variables. Programmable interconnection, also called *routing*, is used to connect logic blocks into larger circuits performing the required functionality. Routing consists of wires that span one or more logic blocks. Connections between logic blocks and routing, I/O blocks and routing, and among wires themselves is programmable, which allows for the flexibility of circuit implementation. Routing is a very important aspect of FPGA devices, because it dominates the chip area and most of the circuit delay is due to the routing delays. I/O blocks in an FPGA connect the internal logic to the outside pins. Depending on an actual device, most pins can be configured as input, output, or bidirectional.

Programmability of FPGAs is commonly achieved using one of three technologies: SRAM cells, antifuses, and floating gate devices. Most devices use SRAM cells. The SRAM cells drive pass transistors, multiplexers, and tri-state buffers, which in turn control the configurable routing, logic and I/O blocks. Since the content of SRAM cells is lost when the device is not powered, the configuration needs to be reloaded into the device on each power-up. This is done using a configuration device that loads the configuration stored in some form of non-volatile memory. Programmability of FPGAs comes at a price. Resources necessary for the programmability take up chip area and consume power. Therefore, circuits implemented in FPGAs take up more area and consume more power than in equivalent ASIC implementations.

Furthermore, since the routing in FPGAs is achieved using programmable switches, as opposed to metal wires in ASICs, circuit delays in FPGAs are higher. Because of that, care has to be taken to exploit the resources in an FPGA efficiently. Circuit speed is important for high-throughput applications like Digital Signal Processing (DSP), while power is important for Low power embedded processor applications. CAD tools [like Quartus II by Altera, Xilinx ISE by Xilinx, etc] are used by the designer to meet these requirements.

FPGA Design Flow

Designing a complex system targeting FPGAs would be virtually impossible without *CAD tools*. The CAD tools convert the user's specification into an FPGA configuration that implements the specified functionality, while optimizing one or more design parameters.

Common optimizations include reducing the chip area, increasing the speed, and reducing the power usage.

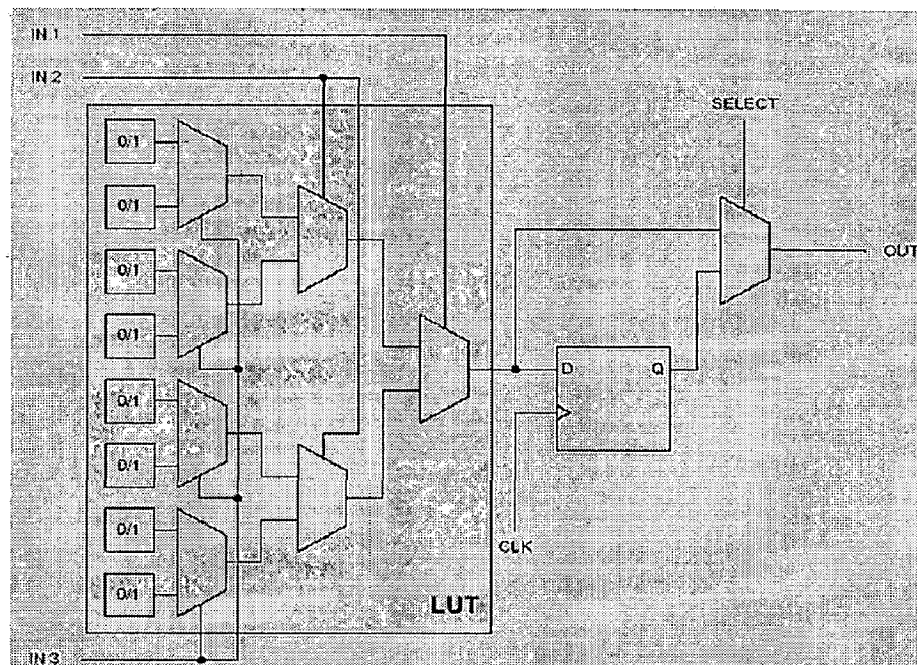


Fig B.1 Simple Logic Block Structure

The CAD tools perform a set of steps to map the design specification to an FPGA. Figure B-2 shows the design flow of typical CAD tools targeting FPGAs. Input to a CAD tool is a high-level circuit description, which is typically provided using a HDL. *VHDL* and *Verilog HDL* are the two most popular HDLs in use today. An HDL circuit description is converted into a netlist of basic gates in the *synthesis* step of the design flow. The netlist is optimized using *technology-independent logic minimization algorithms*. The optimized netlist is mapped to the target device using a *technology-mapping algorithm*. A minimization algorithm ensures that the circuit uses as few logic blocks as possible. Further optimizations that exploit the structure of the underlying FPGA are also performed. For instance, some FPGAs group logic blocks in *clusters*, with high connectivity among the blocks inside the cluster, and less routing resources connecting logic blocks in different clusters. This is usually referred to as *hierarchical routing*. The synthesis tool will use information on the cluster size and connectivity to map logic that requires many connections inside a cluster. This optimization is commonly known as *clustering*. The final result of synthesis is a netlist of logic blocks, a set of LUT programming bits, and possibly the clustering information.

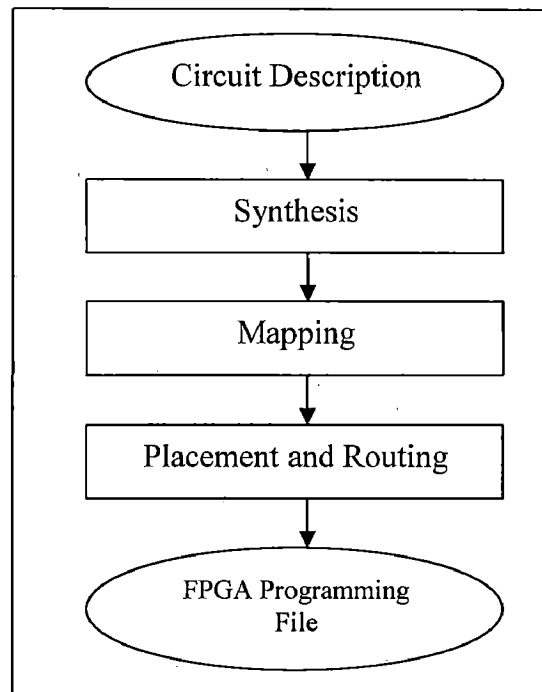


Fig B.2 FPGA design flow

The *placement algorithm* maps logic blocks from the netlist to physical location on an FPGA. If the clustering was performed during the synthesis step, clustered LUTs are mapped to physical clusters. Logic block placement directly influences the amount of routing resources required to implement the circuit. A placement configuration that requires more routing resources than is available in the corresponding portion of the device cannot be routed. Hence, the circuit cannot be implemented in an FPGA with that placement configuration, and a better placement must be found, if one exists. The algorithm starts with a random placement and incrementally tries to improve it. The quality of a particular placement is determined by the routing required to realize all the connections specified in the netlist. Since the routing problem is known to be NP-complete [18], a cost function approximating the routing area is used to estimate the quality of the placement. If the cost function is associated with routing resources only, the placement is said to be routability wire-length-driven. If the cost function also takes into account circuit speed, the placement is timing-driven. Although simulated annealing produces suboptimal results, a good choice of

the cost function yields average results that are reasonably close to optimal. Once the placement has been done, the *routing algorithm* determines how to interconnect the logic blocks using the available routing. The routing algorithm can also be *timing* or *routability-driven*. While a routability-driven algorithm only tries to allocate routing resources so that all signals can be routed, a timing-driven algorithm tries also to minimize the routing delays. The routing algorithm produces a set of programming bits determining the state of all the interconnection switches inside an FPGA. The final output the CAD tools produce is the *FPGA programming file*, which is a bit stream determining the state of every programmable element inside an FPGA. Design flow, including synthesis, placement and routing is sometimes referred to as the *design compilation*. Although the term synthesis is also commonly used, we will use the term design compilation to avoid confusion between the synthesis step of the design flow, and the complete design flow.

Although design compilation does not generally require the designer's assistance, modern tools allow the designer to direct the synthesis process by specifying various parameters. Even variations in the initial specification can influence the quality of the final result (examples of such behaviour will be shown later in the thesis). This suggests that the designer should understand the CAD tools and the underlying technology to fully exploit the capabilities of both tools and the device.

APPENDIX C

The CD along with this report contains the following codes

1. The VHDL code of PN controller(SM Approach) for two level elevator system.
2. The LLD code (Direct Implementation) for two level elevator system.
3. The LLD code obtained by means of formalization for two level elevator system
4. The VHDL code of PN controller(SM Approach) for three level elevator system.
5. The LLD code (Direct Implementation) for three level elevator system.
6. The LLD code obtained by means of formalization for three level elevator system.
7. The VHDL code of PN controller(SM Approach) for four level elevator system.
8. The LLD code (Direct Implementation) for four level elevator system.
9. The LLD code obtained by means of formalization for four level elevator system.