Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

RELAY LADDER LOGIC AND PETRI NETS FOR DISCRETE EVENT CONTROL DESIGN: A COMPARATIVE STUDY

by
Edward Twiss

In the 1960's and earlier discrete event systems (DES) were controlled by hard-wired electromechanical relay systems. In 1969 an electronic programmable logic controller (PLC) was introduced. PLC's have been programmed utilizing relay ladder logic (RLL). RLL is a graphical programming language with software "devices" used to emulate electromechanical devices. RLL programs, however, often become large and difficult to understand because its graphical representation of physical switching devices obscures the discrete event dynamics inherent in the process to be controlled. Petri nets are a methodology for modeling discrete event systems (DES). Using a Petri net based controller, a control strategy could be developed that captures the discrete event dynamics of the process. This should result in a control strategy that is much easier to understand, troubleshoot, modify and evaluate.

RELAY LADDER LOGIC AND PETRI NETS FOR DISCRETE EVENT CONTROL DESIGN: A COMPARATIVE STUDY

By
Edward Twiss

Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Masters of Science in Manufacturing Systems Engineering

Department of Industrial and Manufacturing Engineering

May 1996

APPROVAL PAGE

RELAY LADDER LOGIC AND PETRI NETS FOR DISCRETE EVENT CONTROL DESIGN: A COMPARATIVE STUDY

Edward J. Twiss

Dr. MengChu Zhou, Thesis Advisor — Date
Professor of Electrical and Computer Engineering, NJIT

Dr. Reggie J. Caudill, Committee Member — Date
Professor of Industrial and Manufacturing Engineering, NJIT

Dr. Xiulu Chao, Committee Member — Date
Associate Professor of Industrial and Manufacturing Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Edward J. Twiss

Degree: Master of Science in Manufacturing Engineering

Date: May 1996

Undergraduate and Graduate Education:

- Master of Science in Manufacturing Systems Engineering
  New Jersey Institute of Technology, Newark, NJ, 1996

- Bachelor of Science in Electrical Engineering
  New Jersey Institute of Technology, Newark, NJ, 1987

Major: Manufacturing Systems Engineering

Presentations and Publications:

M.C. Zhou and E. Twiss, "A Comparison of Relay Ladder Logic Programming and Petri Net Approach Sequential Industrial Control Systems", *Proc. of the 4th IEEE Conf. on Control Applications,* Albany, NY, September 1995, pp. 748-753.

M.C. Zhou and E. Twiss, "Discrete Event Control Design Methods: A Review", to appear in *Preprints of the 13th IFAC World Congress*, San Francisco, CA, July 1996.

M. C. Zhou and E. Twiss (1995), "A Comparison of Relay Ladder Logic Programming and Petri Net Synthesis for Control of Discrete Event Systems", Technical Report #9501, Discrete Event Systems Laboratory, ECE, New Jersey Institute of Technology.

ACKNOWLEDGMENT

I would like to express my sincere thanks to Dr. Zhou for his guidance and support throughout the progress of thesis. I would also like to express thanks to Dr. Reggie Caudill and Dr. Xiulu Chao for their participation in my committee. Special thanks also goes to my wife Noreen and Bernadette O'Connor for their support, many hours of proof reading and that always revitalizing cup of tea when most needed.

This thesis is dedicated to
my loving wife Noreen

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

For the past two decades programmable logic controllers (PLC's) using relay ladder logic (RLL) programming have been the workhorse for controlling sequential industrial systems. Petri nets are a methodology for modeling, evaluating and controlling discrete event systems (DES). This thesis uses a theoretical Petri net based controller for an industrial control application. RLL programming and Petri net methods are compared using an industrial design example. Comparisons are made on the ability to evaluate the programmed logic, flexibility of the logic, and the ability to troubleshoot/debug the system.

As product life cycles become shorter, factories are pushed to develop small batches of many different products. The need for highly flexible control systems has become a necessity. The majority of existing automated industrial systems are controlled by programmable logic controllers (PLC's). In most cases the control programs for PLC's are developed using relay ladder logic (RLL) [10]. RLL is a graphical programming language consisting of software devices (i.e. relays, timing relays, drum sequencers and programmable counting devices) to achieve a control strategy. RLL's graphical representation of physical switching devices does not capture the underlying sequential, asynchronous and concurrent events that drive the process to be controlled. It is difficult to determine the original design specification/sequence of operation from the completed RLL program. For this reason control software written utilizing RLL is often difficult to understand and lacks a high degree of flexibility.

Petri net theory was developed in 1962 by Carl Adam Petri. They are a graph theoretic as well as a visually graphical tool specifically designed for modeling, analysis, performance evaluation and control of discrete event systems (DES) [7]. Petri nets are capable of modeling sequential, asynchronous and concurrent events that drive an industrial process. Since Petri nets are a proven tool for modeling these systems, they should prove to be a valuable tool for controlling these processes [7, 14, 8].

Using a Petri net model, a control strategy can be developed which captures the discrete event dynamics of the process being controlled. This should result in a control strategy that is easier to understand, troubleshoot, modify and evaluate system performance. This thesis discusses ladder logic programming and Petri net synthesis techniques for DES. A functional specification for a sequential industrial control problem is described. A control system is designed for the described system using both ladder logic and Petri nets. The thesis makes a comparison on the design/performance evaluation of the RLL-based and Petri net-based systems. Comparisons are made on the flexibility (number of changes required to meet a change in specification), maintainability and support documentation of the two design methodologies.

CHAPTER 2

PROGRAMMABLE LOGIC CONTROLLERS

2.1 Brief History

In the late 1960's electromechanical devices were the order of the day as far as industrial automated control was concerned. These devices, relays, electromechanical timers/counters and electromechanical sequencers were being used by the thousands to control many sequential industrial processes and stand alone machines. These devices installed in control cabinets used hundreds of wires and their interconnection to affect a control solution [4]. These control systems were able to sequence and synchronize the events required to manufacture a single product. However, if a change was made to the product, or a new product had to be produced, the control solution had to be changed by physically rewiring the panel. This required downtime to make the needed wiring changes. If the amount of wiring changes were excessive, it was not uncommon to discard the entire panel in favor of a new one.

In the late 1960's the Hydromatic division of General Motors Corporation wrote a specification for a new type of programmable controller [4]. This new programmable controller would eventually be programmed using relay ladder logic (RLL). The RLL programming language was developed to smooth the transition from relay control systems to PLC's. The initial intent of RLL was to allow plant maintenance personnel to troubleshoot, maintain and make minor modifications to the system. PLC's greatly improved the flexibility of industrial control systems and reduced the downtime required to make modifications. However, the larger the control system, the more difficult it is to determine the

initial design specifications (how the system operates) by examining the control logic. This makes troubleshooting these systems difficult.

Short RLL programs (less than 100 rungs) are not difficult to evaluate. However, as a system becomes more complex, the RLL program can easily grow to 1000 rungs or more. Adverse results of large RLL programs are as follows:

1. The complexity to validate and evaluate the ladder logic grows exponentially (in general).
2. It becomes more difficult to make program modifications to reflect changes in the systems functional specification.
3. Task of troubleshooting and maintaining the program becomes more difficult.

2.2 Methods for Developing RLL

Several methods exist for developing RLL programs, however, no method has been universally accepted by industry. Although industry has greatly benefited from the PLC, little effort has been put forth in the development of formal standards for RLL or other forms of discrete event control. The object of this section to discuss several methods for developing RLL. These methods are useful for developing RLL, and some could be directly implemented on a PLC or general purpose computer.

To help summarize these methods a control problem based on an industrial level control system will be described. A control solution using each design method will be applied to the system, and the resulting solutions will be evaluated. The system to be

controlled (Figure 2.1) is a level control system for a product storage tank. The tank is equipped with five level switches, an inlet solenoid valve and an outlet pump. The following design methods/tools for developing RLL programs shall be discussed and compared:

1. Direct implementation of RLL.
2. Instrumentation Society of America (ISA) Logic Diagrams (ISA standard S5.2-1976).
3. Timing/sequence diagrams.
4. State Diagrams.

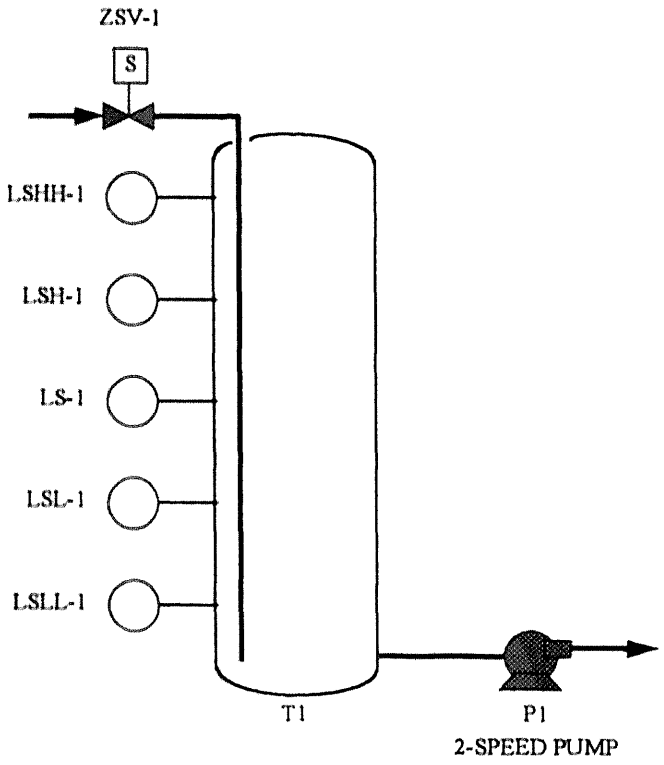


Figure 2.1 - Tank Level Control System

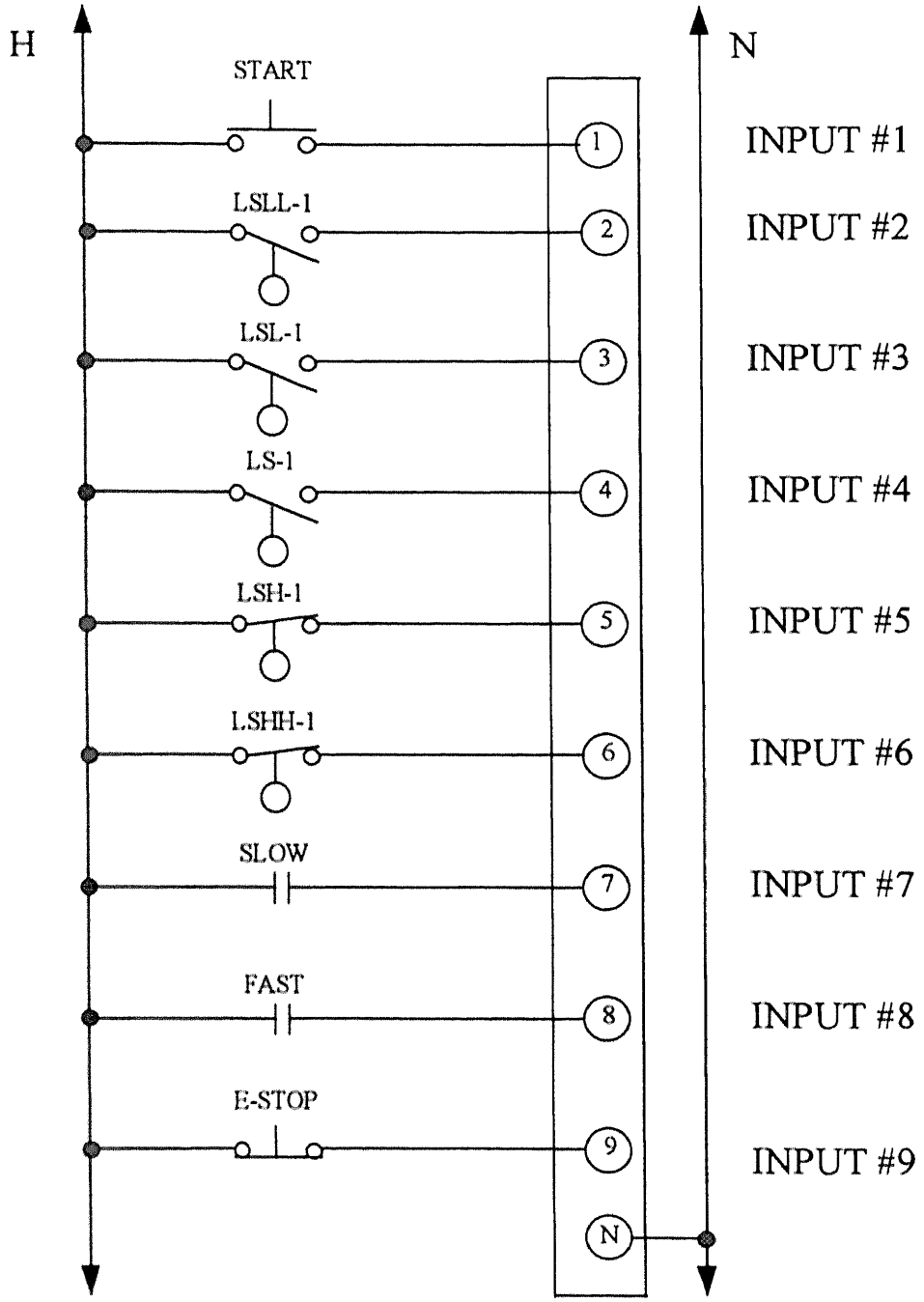


Figure 2.2 - Level Control System Inputs

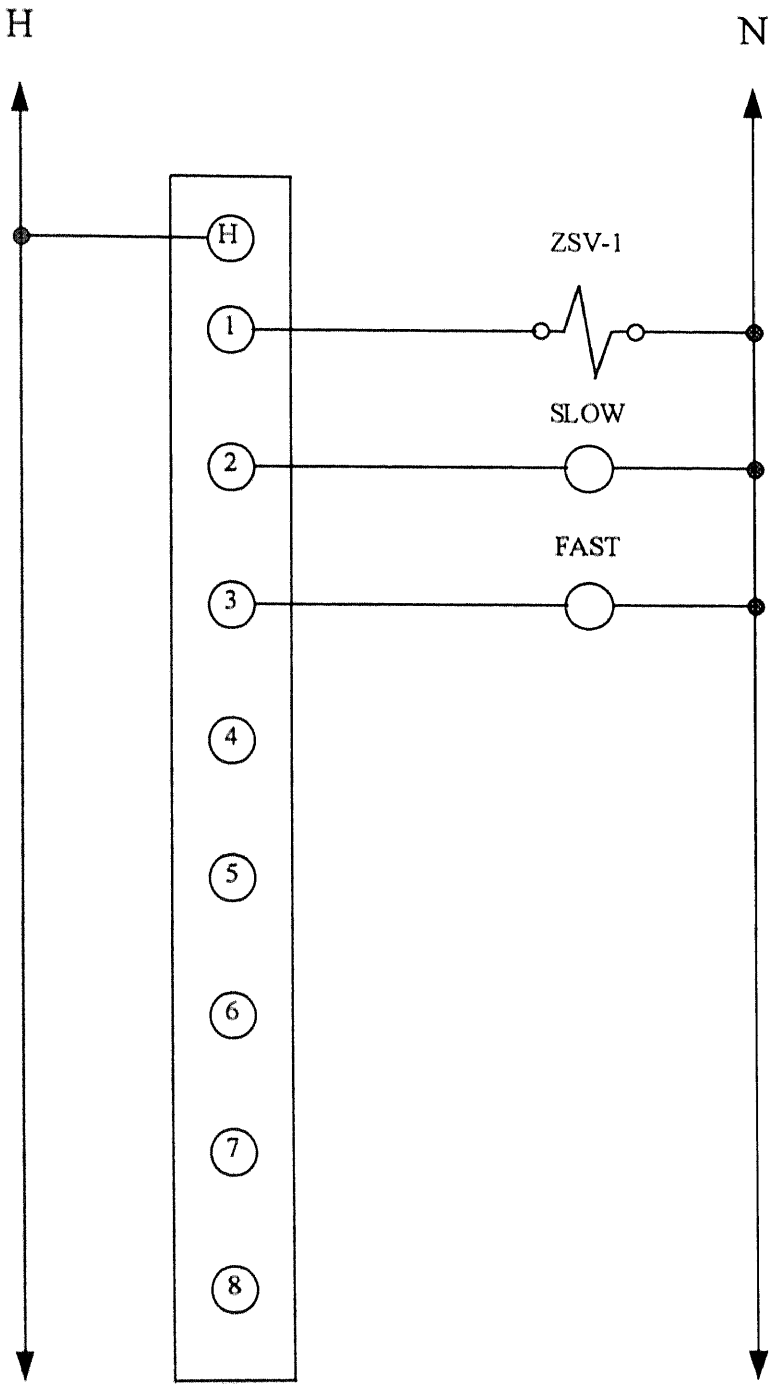


Figure 2.3 - Level Control System Outputs

2.3 Tank Level Control System Functional Description

1. The system is enabled by depressing the reset pushbutton, system operation is halted by depressing the emergency stop (e-stop) pushbutton.
2. Initially the tank is empty (LSLL-1, LSL-1, LS-1=0; LSH-1, LSHH-1=1), fill valve (ZSV-1) opens.
3. Level exceeds LSH-1 (LSLL-1, LSL-1, LS-1, LSHH-1=1; LSH-1=0). Fill valve (ZSV-1) closes and the discharge pump runs at high speed. If LSH-1 fails to actuate; the above occurs when LSHH-1 actuates (LSHH-1 = 0).
4. When LS-1 actuates on falling level (LSLL-1, LSL-1, LS-1=0; LSH-1, LSHH-1 = 1), the pump runs at low speed.
5. When LSL-1 actuates on falling level (LSL-1 = 0), the pump stops and ZSV-1 opens. If LSL-1 fails to actuate; the above occurs when LSLL-1 actuates (LSLL-1 = 0).
6. Level control system inputs and outputs are shown in Figures 2.2 and 2.3: Note high level switches and emergency stop pushbutton are wired for fail safe operation.

2.4 Review of RLL Development Methods

2.4.1 Direct Implementation of RLL

Direct implementation of RLL involves breaking the description of operation down to logical segments. Each segment is related to a separate control function or step in the process. These broken down segments are converted to RLL in an adhoc method. For the above example the system is broken down into the following segments; fill cycle, pump

running fast, pump running slow. Relays A,B,C in Figure 2.4A & B define these segments.

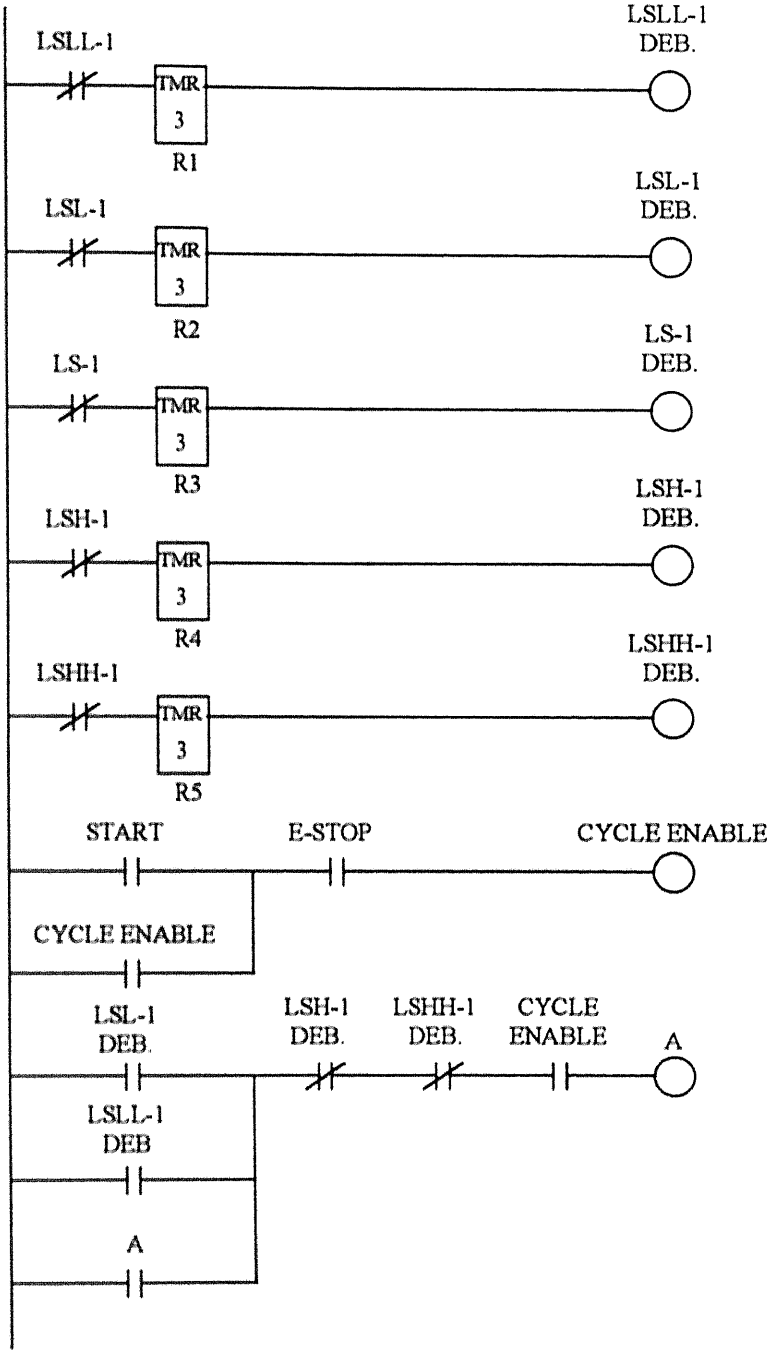


Figure 2.4A - Direct Implementation of RLL (Sheet 1)

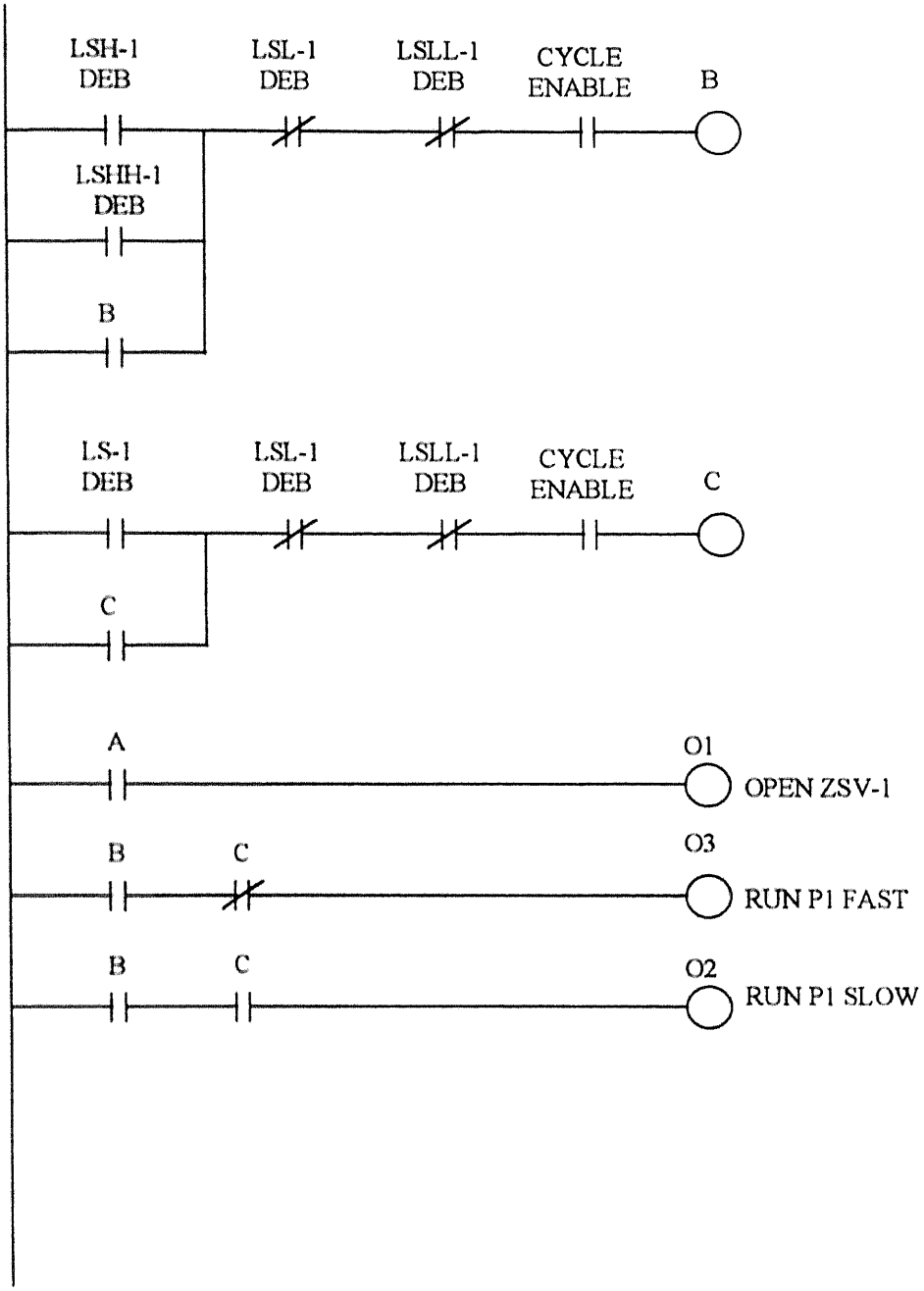


Figure 2.4B - Direct Implementation of RLL (Sheet 2)

Benefits of Direct Implementation of RLL

- RLL was initially developed to allow engineers with hardwired relay control systems experience to begin using PLC's with little or no learning curve.
- Simplifies system troubleshooting/debugging over conventional hardwired electromechanical control.
- Greatly improves system flexibility over hardwired electromechanical control.
- Many new instructions have been developed to allow complex math operations and data manipulation and messaging capabilities.

Drawbacks of Direct Implementation of RLL

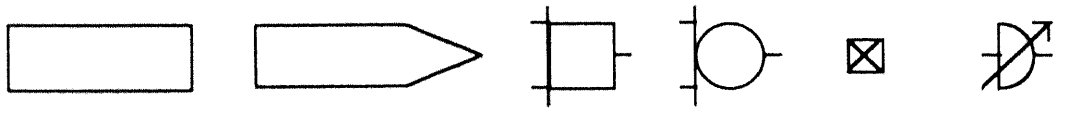
- Difficult to determine the initial design/operating sequences from the RLL program listing.
- Unable to simulate without physical hardware.
- RLL programs often become large and difficult to understand/modify.
- Formal rules do not exist.
- Is difficult to implement a hierarchical programming structure.

This technique can be employed by those who are very familiar with RLL. The major disadvantage of this method is that RLL by itself fails to capture the discrete event dynamics of the system. This makes it difficult to modify or troubleshoot the system. Another major disadvantage is that in order to test the final control system, I/O must be connected to simulate sensors and final control elements. This along with RLL's failure to

capture the discrete dynamics of the system makes programming errors more commonplace during system startup.

2.4.2 ISA Logic Diagrams

These diagrams use symbols similar to Boolean algebra symbols (i.e. AND/OR gates), and special symbols for timer functions, counter functions, and math/data manipulation functions. These diagrams are an intermediate step between the broken down description of operation and RLL coding. ISA logic diagrams tend to decrease the amount of code required by eliminating redundant interlocks. ISA logic diagrams also have the advantage of being converted into RLL of any PLC. The instruction sets of different PLC's vary, ISA logic diagrams are a portable form of logic for all types of PLC's. The logic diagrams tend to be a more compact representation of the system control strategy and can therefore help in system modifications/debugging. ISA logic diagrams could be directly implemented on a PLC. However, logic diagrams suffer from the same drawbacks of direct implementation of RLL. They fail to capture the discrete event dynamics of the system and cannot be simulated without connecting physical I/O to the system. The ISA logic diagrams are shown in Figures 2.5A & B. Following is a list of symbols used in the diagrams and their meaning. For more detail the reader should refer to ISA standard S5.2-1976.



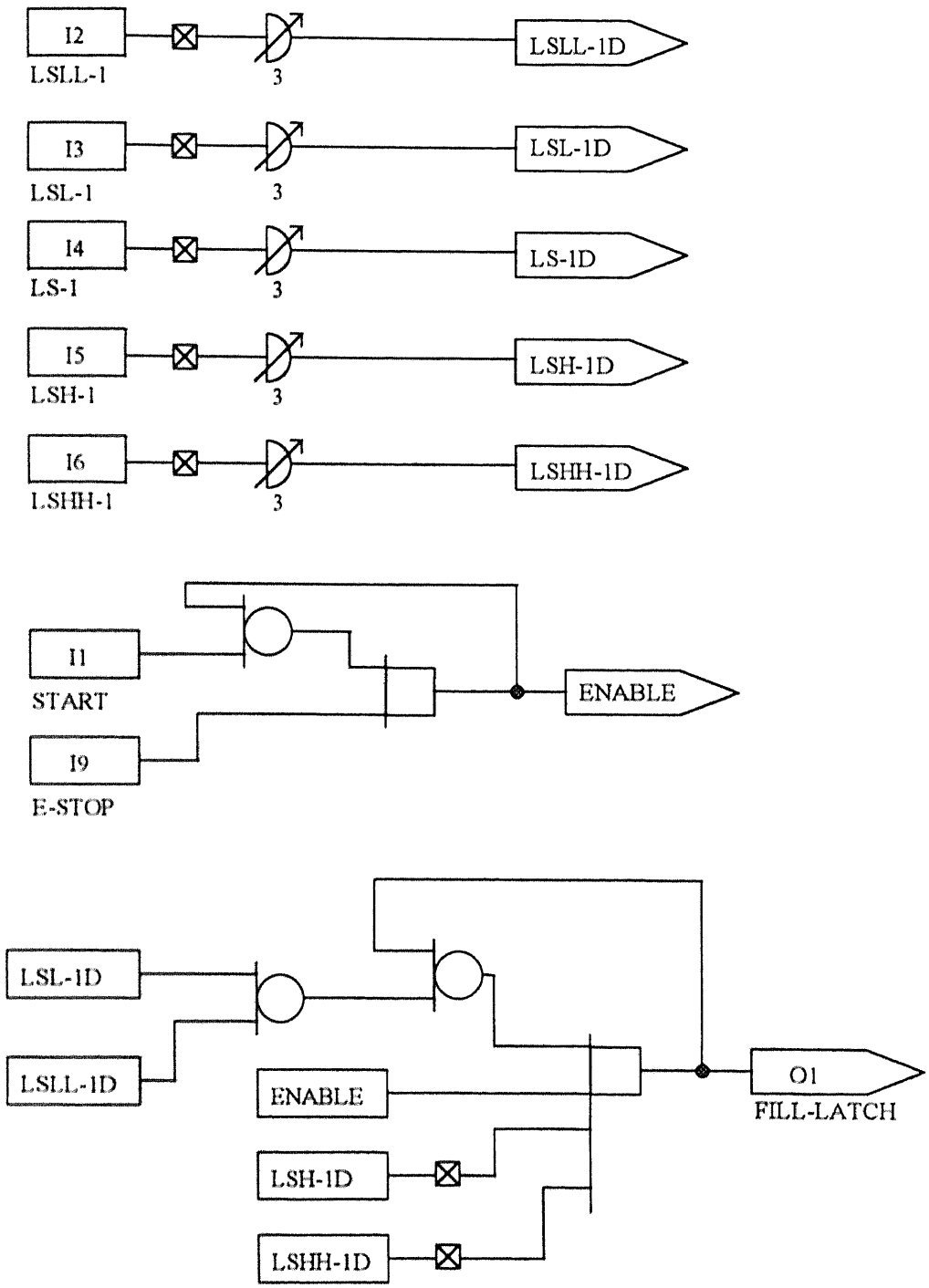


Figure 2.5A - ISA Logic Diagram (Sheet 1)

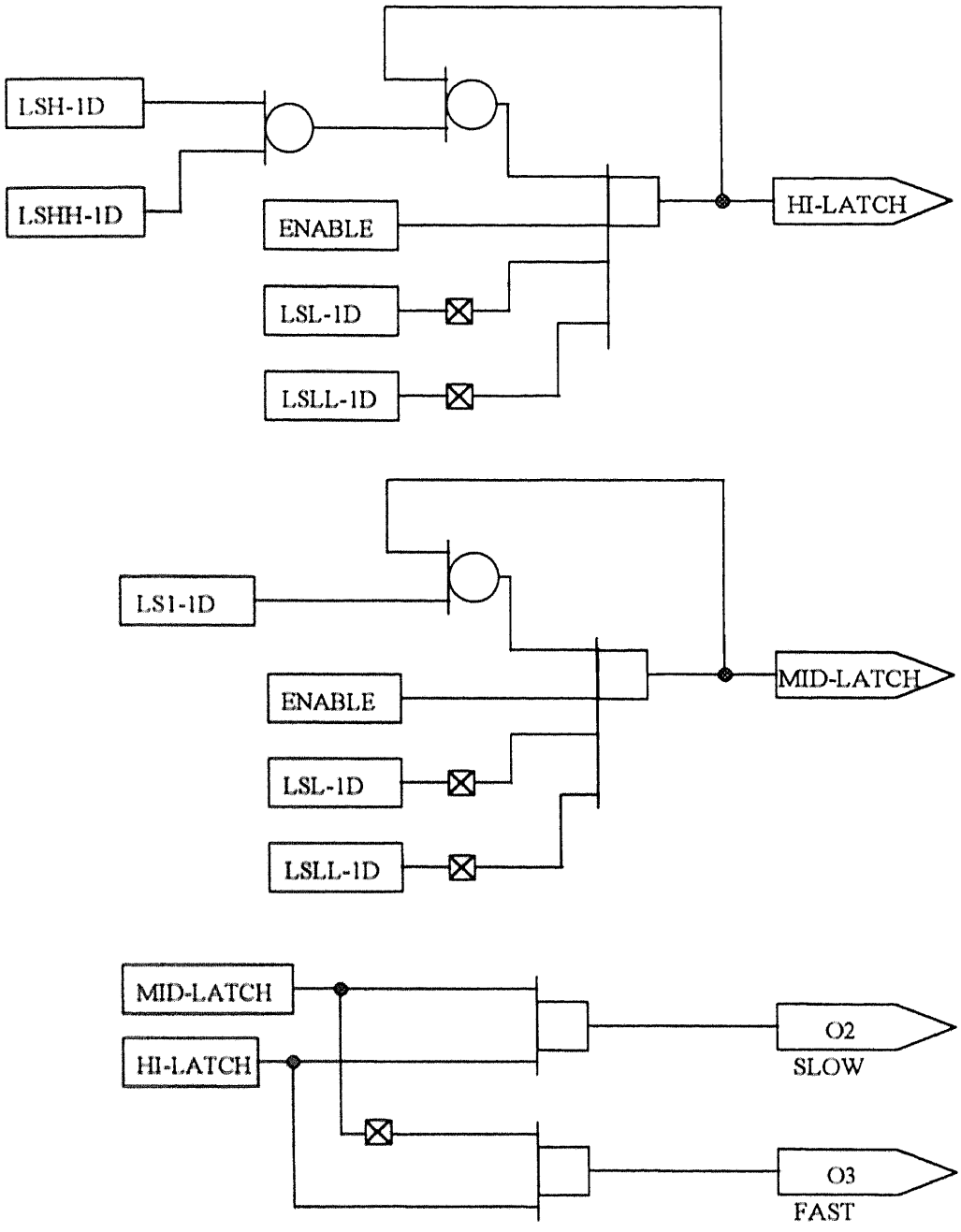


Figure 2.5B - ISA Logic Diagram (Sheet 2)

Benefits of ISA Logic Diagrams

- Helps to map out required PLC memory.
- Enables the control engineer to document most of the programmed instructions ahead of time.
- A portable form of logic.
- Graphical symbols are familiar to most engineers.
- Helps to eliminate repetitive interlocks.

Drawbacks of ISA Logic Diagrams

- Is an adhoc approach.
- Difficult to determine the initial design/operating sequences from the final diagrams.
- Unable to simulate without physical hardware.
- Is difficult to implement a hierarchical programming structure.
- Diagrams often become large and difficult to understand.

2.4.3 Timing/Sequence Diagrams

A timing/sequence diagram is a horizontal bar chart similar to a Gantt chart. Timing/Sequence diagrams are an intermediate step between the description of operation and RLL development. The I/O addresses and major internal flags appear in a column on the left side of the chart and the major process events appear across the top of the chart. The events form vertical lines that determine the on/off sequencing of the I/O and internal flags. Internal flags are internal bits used to represent the broken down logical segments.

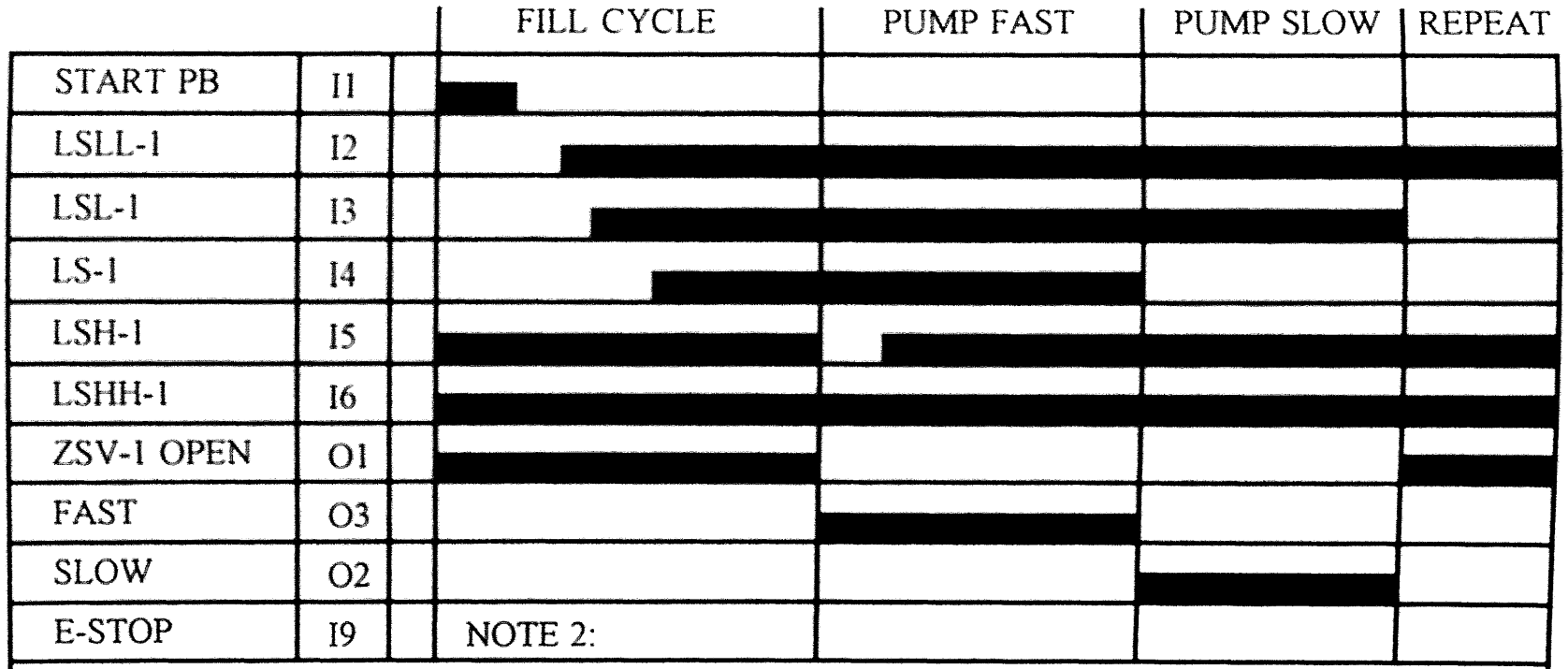
Timing/sequence diagrams succeed in capturing the discrete event dynamics of the system. This is the simplest representation of the on/off sequencing of a system. The only requirement to develop a sequence diagram is a full understanding of the system. The timing/sequence diagrams can be helpful in system modifications/debugging. They are helpful in developing programs for small sequential systems. However as systems grow in complexity and number of operating modes, the underlying logic that drives the system from one state/internal flag to the next becomes obscure. These diagrams can not be directly implemented on a PLC. These diagrams must be converted to RLL, and suffer from the same drawbacks as direct implementation of RLL.

Advantages of Timing/Sequence Diagrams

- Captures the systems sequence of operation.
- Aids in system troubleshooting/debugging.
- Little or no learning curve involved.

Disadvantages of Timing/Sequence Diagrams

- Unable to program without converting to RLL or some other form.
- A Sequence diagram must be developed for each subsystem.
- In a system with many subsystems it is often difficult to determine the interrelations between the different diagrams.
- Can be difficult to determine the underlying logic.



NOTE 1: All level switches are debounced for three seconds.

NOTE 2: E-stop will bring the system back to an idle state. The system must then be manually restarted by hitting the start push-button (I1).

NOTE 3: If the low level switch should fail then the low-low level switch will initiate stopping the pump and opening fill solenoid ZSV-1. If the high level switch should fail then the high -high level switch will initiate starting the pump at full speed.

Figure 2.6 - Timing/Sequence Diagram Approach

2.4.4 State Diagrams

State diagrams are an analytical method that provides a set of rules for developing RLL. By breaking the description of operation down to logical segments, the number of states and state variables can be determined. After these are determined the logic that takes the system from one state to the next can be determined. The state diagram can then be drawn by using circles with ones and zeroes inside (the ones and zeroes represent state variables). Directed arcs are used to interconnect the states; above each arc is the logical expression that translates the system from one state to the next. After the state diagram is complete, it can be converted to Boolean equations representing the setting and resetting of state variables [16]. These Boolean equations can then be converted to RLL or directly implemented on a computer. State diagrams help to eliminate the number of initial programming errors, capture the discrete event dynamics of the system, and could be directly implemented on a PLC or computer.

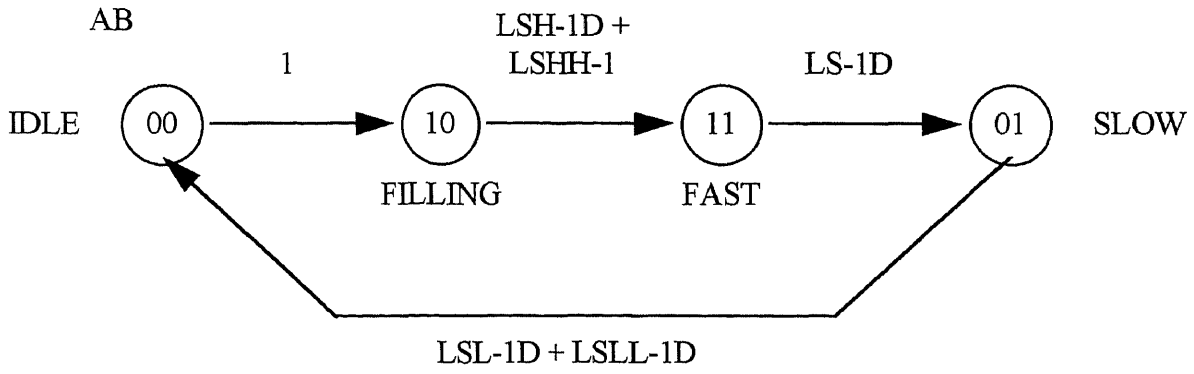


Figure 2.7 - State Diagram Method

Boolean Equations Derived from State Diagram:

$SA = B'$

$RA = B * LS\text{-}1D$

$SB = A * (LSH\text{-}1D + LSHH\text{-}1D)$

$RB = A' * (LSL\text{-}1D + LSLL\text{-}1D)$

Advantages of State Diagrams

- Is an analytical method that provides a set of rules for developing the RLL.
- Could be directly implemented by a PLC or computer.
- The systems sequence of operation can be determined from the model.
- Is a method taught and familiar to most engineers for developing sequential logic circuits.

Disadvantages of State Diagrams

- Difficult to implement when more than four state variables are involved.
- Difficult to model concurrent events.

The following table (Table 2.1) is a summary of the advantages and disadvantages of the different methods for developing RLL programs. Other techniques for developing RLL programs may exist. The techniques covered in this thesis are the techniques observed by the author through his experience in industry. As mentioned earlier industry has not invested much effort in standardizing on techniques for developing RLL programs.

Table 2.1 - Methods for Developing RLL Programs

METHOD	ADVANTAGES	DISADVANTAGES
Direct RLL Implementation.	1. Easily used by engineers with hardwired relay experience.	1. Fails to capture the system's discrete event dynamics, and is difficult to determine the initial design specification or operating sequence from the RLL program listing. 2. Unable to simulate without physical hardware. 3. Is difficult to implement a hierarchical programming structure.
ISA Logic Diagrams S5.2-1976.	1. A portable form of logic - Able to generate RLL code for different manufacturers' PLC's from a standard logic diagram. 2. Graphical symbols familiar to most engineers. 3. Helps to eliminate repetitive interlocks.	1-3. Same as direct RLL implementation.
Timing/Sequence Diagrams	1. Captures the system's discrete event dynamics. 2. Is easy to understand for a simple system. 3. Involves little or no learning curve involved. 4. Need only a complete understanding of the sequence of operation to develop the sequence diagram	1. For a large system with many subsystems it is necessary to develop a sequence diagram for each subsystem. It therefore becomes difficult to determine the interrelations between the different sequence diagrams. 2. Can be difficult to determine the underlying logic.
State Diagrams	1. An analytical method that provides a set of rules for developing RLL. 2. Captures the systems discrete event dynamics. 3. Is familiar to most engineers for developing sequential logic circuits.	1. Difficult to implement when more than four state variables are involved. 2. Difficult to model concurrent events.

CHAPTER 3

THEORY OF PETRI NETS

3.1 Basic Petri Nets

Carl A. Petri developed Petri net theory to analyze communication systems [8]. Petri nets have proven to be a useful tool for modeling, control and performance evaluation of manufacturing systems [14]. Their fundamental knowledge can be seen in [6, 13] and applications in manufacturing automation in [20]. Petri nets are useful for modeling systems with the following characteristics [7, 13]:

1. Concurrency - More than one operation taking place at a time.
2. Asynchronous Operations - Operations are completed at different times and in different amounts of time. A Petri net model can maintain the sequence of events under these conditions.
3. Deadlock - The system reaches a state in which no new processes can be started. This can happen when two or more processes share one resource.
4. Conflict/Choice - Two or more operations are enabled to begin, however, only one part is allowed to be processed. A choice must be made in this situation as to which process should be started.
5. Event Driven - A process coming to completion can be considered an event. The occurrence of this event starts a new process. The order/sequencing and timing of the events need not be unique. The order of the events are driven by the overall state of the system.

Petri nets like RLL are built up of several basic elements. Following is a description of these basic elements [13]:

1. Places (circles) are used to represent conditions (true/false), resource availability or a process status (i.e. a machine is processing a part).
2. Transitions (bars or boxes) are used for events, start and end of activities.
3. Input functions are defined as "arcs" from places to transitions.
4. Output functions are defined as "arcs" from transitions to places.

These four elements define the structure of a Petri net. The state of a Petri net is defined by its marking. The marking is the number of tokens (dots) in each place. Tokens travel from place to place along directed arcs. The flow of tokens is determined by the "firing" of transitions. A marking, m is denoted as an n-vector, where n is the total number of places. The pth component of m, denoted by $m(p)$, is the number of tokens in place p. Mathematically a marked Petri net is a five-tuple (P,T,I,O,m_0), where:

1. P is a finite set of places.
2. T is a finite set of transitions, with $P \cup T \neq 0$ and $P \cap T = 0$
3. I: $P \times T \rightarrow N$ is an input function that specifies arcs directed from places to transitions, where N is the set of all natural numbers.
4. O: $P \times T \rightarrow N$ is an output function that specifies arcs directed from transitions to places, where N is the set of all natural numbers.
5. $m_0 : P \rightarrow N$ is an initial marking whose i^{th} component represents the number of tokens represented by dots in the i^{th} place.

In order to simulate the dynamic behavior of a system, a state or marking in a Petri net is changed according to defined "firing" rules. These firing rules are also referred to as the token player game. They are defined as follows:

1. A transition t is said to be enabled if each input place p of t is marked with at least I(p,t) tokens, where I(p,t) is the weight of the arc from p to t. Mathematically, $t \in T$ is enabled iff $m(p) \geq I(p, t) \ \forall p \in P$.

2. An enabled transition may or may not fire (depending on whether or not the event takes place).

3. Firing of an enabled transition t removes I(p,t) tokens from each input place p of t, and adds O(p,t) tokens to each output place p of t, where O(p,t) is the weight of the arc from t to p [13]. Mathematically, t fires at marking m', yielding the new marking $m(p) = m'(p) + O(p,t) - I(p,t), \ \forall p \in P$.

The marking m is said to be reachable from m'. The *reachability* set is the set of all markings reachable from m_0 by a sequence of transition firings and is denoted by $R(Z,m_0)$. A place $p \in P$ is *k-bounded* iff $\exists\ k > 0$, $\ni\ m(p) \leq k$, $\forall\ m \in R(Z, m_0)$. Z is *k-bounded* iff p is *k-bounded*, $\forall\ p \in P$. Z is *safe* iff it is *1-bounded*. Z is *live* iff $\exists$ a fireable sequence whose firing results in a marking which enables t, $\forall t \in T$ and $m \in R(Z, m_0)$. The significance of *boundedness*, *liveness*, and other properties of Petri nets in manufac-

turing is discussed in [20]. Briefly, *boundedness* guarantees stability, *liveness* guarantees freedom from deadlocks and repeatability.

Deterministic time delays are often associated with transitions and/or places in Petri nets which model industrial automated systems. Random time variables can also be associated to a Petri net. By associating random time variables which follow an exponential distribution to some of the transitions, a generalized stochastic Petri net is developed (GSPN). These types of nets consist of both timed and immediate transitions. The timed transitions model the delays of a systems operation. GSPN's and Petri nets in general can be used to mathematically determine many of the properties of a system (i.e. throughput, resource utilization) [13, 7, 21].

3.2. Real-Time Petri Net Based Controller

A real-time Petri net (RTPN) based controller can be developed by assigning physical input/output (I/O) functions to places and assigning physical I/O and timing variables to transitions of the Petri net model. Formally the RTPN could be defined as follows [18]: A RTPN is eight tuple and defined as: RTPN = $\{P,T,I,O,m_0,D,Y,Z\}$ where:

- $\{P,T,I,O,m_0\}$ are as defined in 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.

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 a Boolean expressions of input addresses. When the function associated with Y(i) is true and all input places are marked, then the firing rule is executed (e.g. tokens are removed from input places and deposited in output places). Vector Y is the firing attribute of all transitions in the RTPN. The RTPN would have two (2) basic types of transitions:
    - Immediate - Represented as solid bars, these would always have a firing attribute of one and zero time delay.
    - Input - Represented by hollow bars, these transitions would fire when all input places are marked, the firing attribute is true, and the time delay d_i has expired. Transition t_i can have the form as follows: Y(i) = I# or a Boolean expression, e.g., (Y(i) = I1 * I2 + I(3); D(i) = x sec/min). Where i is the transition number, I# is an input address often represented by tag name, and x is a preset time delay. Y(i) and D(i) together are the firing attribute of t_i.

3. Vector Z writes commands to the digital output interface. When a place p_i is marked by a token, then some output function occurs (e.g. start motor). Place p_i would only be allowed to write to a single element x_i of vector Z, thus p_i would only be allowed to set or reset a single output.

It should be noted that in [18], input signal functions are associated with places and output with transitions. Our practice of PN modeling of industrial systems suggests that it is easier to associate input signal functions with transitions and output with places. This latter approach was adopted in [14, 21].

An emergency stop situation normally requires the process to be halted (all outputs reset) and for the system to return to an initial state or last state, upon the alarm condition being manually or automatically reset. To handle different emergency stop schemes and simplify modeling, special e-stop places would be used. These places have an additional attribute that disables all designated outputs when the e-stop place is marked. Once the e-stop place is marked all tokens will be removed from the RTPN. When the e-stop place loses its token, either the last state marking or the initial marking m_0 will be restored, based on the type of restart required. This place can also be used to execute a fault routine (i.e. execute a safe shutdown or alarm annunciation routine).

3.3. RTPN Model of Conveyor Start/Stop Control

Following is an example of how a RTPN can be used to model a conveyor system. The functional specification for the system is as follows:

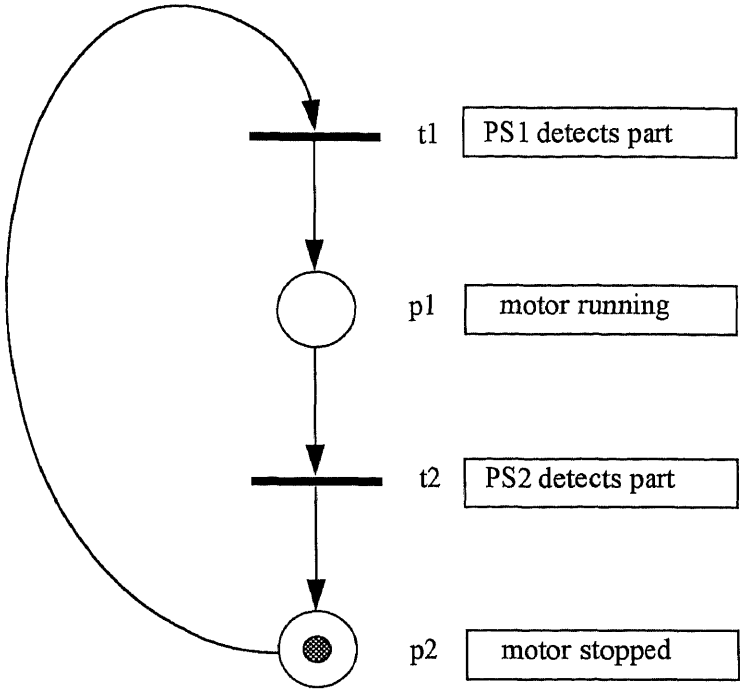
When parts are detected by a photo switch (PS1) in the conveyor receiving area, the conveyor motor is commanded to start. When the delivery area photo switch (PS2) detects a part the conveyor is commanded to stop. This DES can be broken down into the following events:

1. Parts are detected by the receiving area photo switch.
2. The motor is commanded to start.
3. Parts are detected by the delivery area photo switch.
4. The motor is commanded to stop.

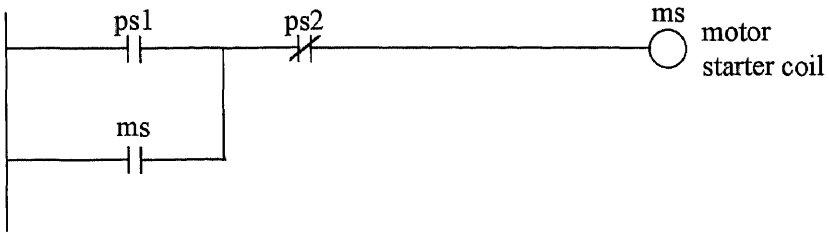
This system can be modeled using two places and two transitions. A place is required for each system state, i.e. conveyor running/conveyor stopped:

- p_1 sets the motor run output.
- p_2 resets the motor run output.
- t_1 has a firing attribute of PS1 = 1 and delay = 0.
- t_2 has a firing attribute of PS2 = 1 and delay = 0.

An RTPN model of the system is shown in Figure 3.1(A). The system has an initial marking, m_0 = [0,1], i.e., place p_1 is not marked and place p_2 is marked. Transition t_1 fires when p_2 is marked and PS1 detects a part. Place p_1 starts the conveyor motor by setting output signal x_1 on the digital output interface. Transition t_2 fires when the motor is running (p_1 marked) and a part is detected by the delivery photo switch PS2. When the token is removed from Place p_1 the motor is stopped by resetting output signal x_1. The system is now at its initial state and waits for another part to become available. The RLL equivalent to the RTPN appears in Figure 3.1(B).



(A.)



(B.)

Figure 3.1 - (A) RTPN Model of Conveyor System & (B) RLL Model.

CHAPTER 4

COMPARISON OF PROGRAMMING LANGUAGES

4.1 Basic Elements

Both RLL and Petri nets contain several basic elements [18]. PN's have basic elements to model conditions, status, activity, information/material flow, and resource avalability. RLL does not have corresponding explicit representation to model these conditions. Logical AND and logical OR can easily be modeled by both PN and RLL. Other concepts such as time delays, cycle counting, concurrency and synchronization can be modeled by both PN's and RLL [18].

4.2 Advanced Instructions

Since PLC's have been on the market, the instruction set has expanded far beyond the basic elements. The instruction set of todays typical PLC includes [4, 10, 12].

- Logical operations (AND, OR, NOT,etc..) on word length data.
- Integer and floating point math instructions.
- Data manipulation instructions (i.e. word moves, block moves, bit shift instructions, bit rotate, etc.)
- Data conversion (Binary to BCD, etc.)
- Program control functions (jump to subroutine, jump to label, etc.)

To include the expanded features of today's PLC's into a RTPN controller, special "places" could be developed. These special places could be built in functions, or user-defined functions. If these special places are part of the "built in" functions, programming software could be developed to allow the user to zoom into these places and observe their contents. If these special places are user defined subroutines, these routines could be written in "C" or Basic using the RTPN editor. These user defined subroutines could be stored in a library and used over and over again by changing the number and addresses of the input and output parameters. In RLL, data manipulation is performed by individual functions entered directly into the ladder logic editor. This allows the user to view the contents of the registers being manipulated. However, entering these functions directly in the ladder logic editor further obscures the systems underlying sequential logic. RLL allows the storage of blocks of ladder logic in library files. However, using these library files not only requires changing the number and addresses of the input and output parameters, but also requires modifying the addresses and tag descriptions of the internal elements of the library file.

To keep the PN both small and manageable, sub-PN's could be developed. These sub-PN's could be represented as a box containing two transitions and one place. The sub-PN's would consist of tested RTPN program modules representing the different components of the system to be controlled. This would allow the programmer to break the system down into small easily understandable RTPN models. These models could be designed and tested individually, allowing concurrent engineering activities to take place. To control program flow and execution, a supervisory PN could be used. The supervisory

PN would perform a function similar to the main program file in a "C" or Pascal program. The addition of the sub-PN and supervisory PN allows for a hierarchical top-down or bottom-up approach to developing a design algorithm. An example of this hierarchal structure is as follows:

A system consists of five automatic assembly lines, waste water treatment of the process effluent and an air pollution control and monitoring system. Each automatic assembly system has three assembly stations. The supervisory PN enables each of the main program sections (the five automatic conveyor systems, the waste water treatment process and the air pollution control system). Thus only enabled program sections are executed. If a particular line is not chosen for operation, its program logic is not solved. The waste water treatment system logic is solved if at least one of the lines is in operation. A RTPN is developed for a typical assembly line, the waste water and air pollution control processes. The typical assembly line PN is reused and modified for the remaining four conveyors. Each of these RTPN's is then broken down into groups of sub-PN's. A typical assembly line PN consists of a RTPN to sequence the transportation of materials and the initiation of work at each of the three assembly stations. The typical assembly line RTPN could contains three sub-PN's for each of the assembly stations.

At the present time Allen Bradley has developed a Sequential Function Chart (SFC) programming language. SFC is a Petri net like programming language used to coordinate large, complicated tasks into smaller, more maneagable tasks. The SFC itself is not used for direct process control. However, it controls the execution of independent

RLL files. Similar to the RTPN language, a RLL file is developed for subsystem and executed according to the supervisory SFC.

The RTPN would allow using places and transitions in more than one sub PN. In a formal PN, a place or transition is only allowed to appear once in the net. A cross reference index to places and transitions could be generated to show their usage. This would be used for simplifying different interlocking schemes and keeping multiple arcs from crossing over each other.

CHAPTER 5

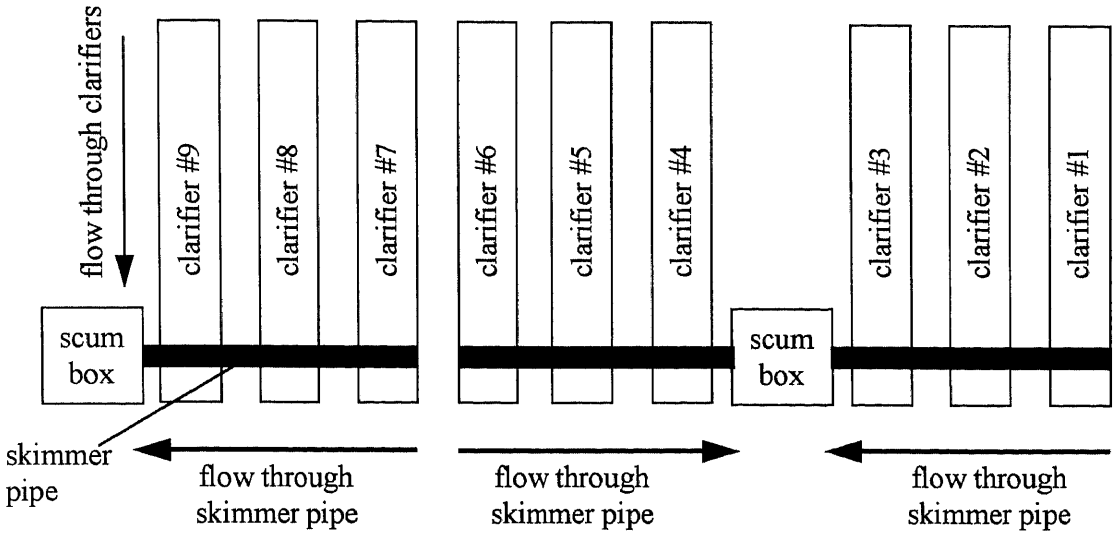
RLL AND PETRI NET DESIGN FOR AN INDUSTRIAL SYSTEM

5.1 Functional Specification of an Industrial System

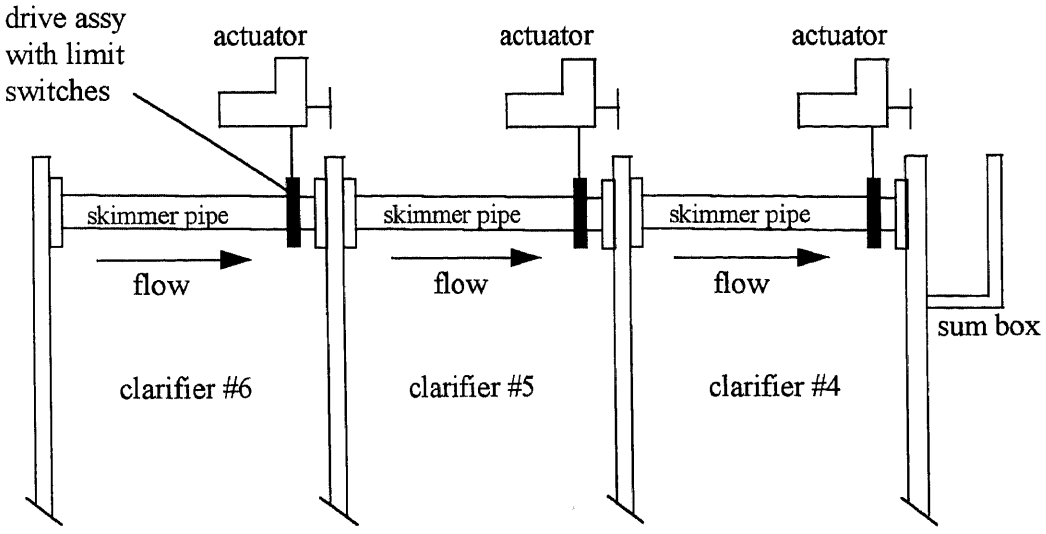
This functional description is based on a process used in water pollution treatment facilities. The control system is for a secondary clarifier scum removal system. The scum removal system is manufactured by Envirex Inc. Waukeshaw, Wisconson and is to be installed in the Deer Island water pollution treatment facility near Boston, Massachusettes. The system as shown in Figures 4.1(A) and 4.1(B) consists of nine (9) clarifier channels, nine (9) skimmer pipes (one for each channel), and two (2) scum boxes. Waste water flows through the clarifiers towards the skimmer pipe. As shown in Figure 4.1(B), the skimmer pipes are sloped in groups of three towards a scum box. The skimmers are rotated on a sequential basis to allow the froth on the surface of the water to be collected in a scum box. Figure 4.2 shows the skimmer pipe positions related to the flow of water through the clarifier.

5.1.1 Skimmer Pipes

As shown in Figure 5.1, each skimmer pipe is equipped with a motorized actuator and position limit switches. A local control panel at each skimmer pipe is provided with a local/off/remote selector switch. In the local mode the tubes are operated using switches located on the local panel. The local controls are provided for maintenance purposes and emergency system operation. In remote mode the skimmer pipes are controlled by the PLC.



(A)



(B)

Figure 5.1 - (A) Top View of Clarifiers & (B) Detail of Skimmers 4-6.

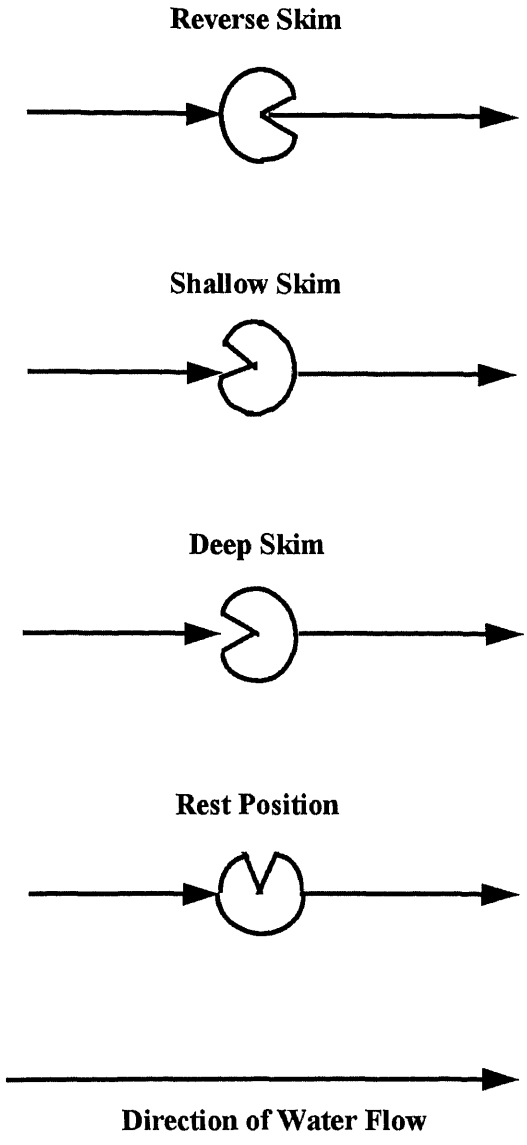


Figure 5.2 - Skimmer Pipe Rotation

5.1.2 Spray Water Valve

Each skimmer pipe is also equipped with a spray water solenoid valve (not shown in the Figures). Each spray valve has a local/off/remote selector switch at a local control panel. In remote mode the respective valve is controlled by the PLC. In local mode the respective valve is operated by local open/close switches. Similar to the skimmer pipes, the local controls are provided for maintenance purposes and emergency system operation.

5.1.3 Scum Flushing Valve

A scum flushing valve is provided to flush the scum boxes at the completion of the skimming cycle. The scum flushing valve is equipped with open and closed limit switches, and a torque switch. The scum valve is also provided with a local/off/remote selector switch at a local control panel. Local/remote operation is similar to the skimmer pipes and spray water valves.

5.1.4 Remote Sequence of Operation

The skimmer pipes are to be provided with a sequential controller to rotate each skimmer on a timed or continuous basis. In the timed mode, the sequence completes one cycle every 6-24 hours (adjustable). In the continuous mode, the sequence starts when continuous mode is selected. The sequence repeats until continuous mode is no longer selected. In both modes of operation the skimming sequence is the same. The sequence starts with skimmer A9 and finishes with A1 in a decreasing numerical order. The sequencing within each skimmer is as follows:

1. Rest position to reverse position for 10 to 60 seconds (adjustable).
2. Reverse to shallow or deep skim position for 10 to 120 seconds (adjustable).
3. Step to next skimmer.
4. The spray water valve opens during the skimming sequence of the associated skimmer pipe.
5. The scum flushing valve opens for 60 seconds (adjustable) at the completion of the skimming sequence.

The following interlocks affect the skimming cycle as follows:

1. The skimming cycle is interrupted on a high wet well level, the skimming cycle is re-initialized at the interrupted skimmer on low wet well level.
2. The skimming cycle is halted and an alarm is generated on a skimmer malfunction. The skimming cycle is resumed at the next skimmer on activation of "resume" pushbutton. A skimmer malfunction is defined as failure to reach an operational mode (rest, reverse, shallow or deep), within a specified time.
3. If the "local/off/remote" selector switch of a particular skimmer is not in the remote position, then resume skimming at the next skimmer.

5.2 RLL Design

The RLL program was developed with the assistance of ISA logic diagrams. Samples of the logic diagrams appear in Zhou & Twiss [25]. As can be seen the logic diagrams map out many of the internal status bits and registers used by timers and counters. This enables

the programmer to document many of these internal instruction before any rungs are entered with the RLL editor. This helps to insure the programmer the correct addresses are entered when editing instructions. The PLC used for this project was an Allen Bradley PLC-5, model 5/40. The PLC software was written using Wintelligent software by ICOM Inc. A copy of the documented ladder logic program appears in Zhou and Twiss [25].

The RLL program was created by entering one RLL file for skimmer #9. This file was tested with a test stand physically connected to the PLC I/O modules. The test stand consisted of maintained and momentary switches to simulate input conditions, and lights to indicate output signals. After the first program file was completely tested, it was copied and stored in a library file. The addresses I/O and internal status bits of the library were then *searched* and *replaced* with the addresses for skimmers 8 through 1. When the *search* and *replace* was completed the files were inserted into the RLL program with the logic for skimmer #9. To index from one skimmer subroutine to another, a sequential function chart (SFC) was used [12, 6]. SFC's are similar to Petri nets. SFC's consist of steps (similar to Petri net places) and transitions. A SFC repeatedly scans a step (program file subroutine) until its output transition(s) become true. Once the output transition(s) become true, all output conditions in the step are reset and the next enabled step (program file subroutine) is scanned. After the SFC was entered, the system was again tested using the test stand. A printout of the SFC appears in Zhou & Twiss [25]. The logic for the scum valves and spray water valves was entered and a final test was then performed. A portion of the ladder logic as shown in Appendix B is described as follows:

6-24 Hour Timer

Rungs 1-3 define the 6-24 hour timer. The minute timer on rung #1 is a 1 minute self reseting timer, whose done bit pulses and increments the minute counter on rung #2. The minute counter has a maximum preset value of 32,767 minutes. The minute counter resets when rung #3 is executed.

Timer Mode

Rung #4 output instruction latches when the minute counter done bit is set. The auto start timer latch bit remains set until skimmer #1 completes its skim cycle or timer mode is no longer selected. This output instruction is used to remember that the system was initiated in timer mode and is used on rung #5 to automatically restart skimming after a high to low wet well transition. Rung #5 output instruction (auto start tmr) is a "one shot". The ONS instruction only enables the post-conditions for one program scan while the pre-conditions are set. This bit is set by either the minute counter done bit "or" a high to low wet well transition when the system is in timer mode. The logic for the high wet well level latch and low wet well reset bit is not shown.

Continuous Mode

Rung #6 output instruction is also a "one shot". This bit is initially set when the resume pushbutton is depressed. It is then repeatedly set by the previous cycle complete bit. This bit is also set on high to low wet well transitions while continuous mode is selected.

Step to Next Skimmer

Rung #7 enables the next skimmer when any of the following ocurrs:

1. A skimmer fault ocurrs and the "resume" pushbutton is hit.
2. The skimmer is in rest mode and the rest limit switch is activated.
3. Remote mode is not selected and the skimmer is in any cycle.

Auto Start

Rung #8 output instruction (auto start) is set when either the timer OR continuous auto start bits are set AND remote mode is selected. This is also a "one shot" instruction since the auto start timer and continuous instructions are "one shots".

Reverse Skim

Rung #9 output instruction (reverse skim) is set and latched when the auto start bit is set. This bit remains set until either of the following ocurrs:

1. Actuator fault going to the reverse position.
2. Deep or shallow skim mode is initiated.
3. A high wet well level is reached.

Once in reverse skim mode, an actuator fault timer is started (rung #10). If the input for the reverse limit switch does not become true within a preset time, an actuator fault is initiated (rung #11). Also in the reverse skim mode, the PLC has a choice of starting two timers depending on the position of the "deep/shallow" selector switch.

When one of these timers expires the reverse skim mode is terminated and shallow or deep skim is initiated.

Deep or Shallow Skim

Deep or shallow skim cycles are identical to the reverse skim cycle. When the respective timer expires the deep or shallow skim mode is terminated and the skimmer returns to the rest position.

Rest Position

Once the skimmer is commanded to go to the rest position and the rest limit switch is actuated, the next skimmer is enabled (rung #7). If the rest limit switch is not actuated within a preset time, an actuator fault is initiated.

Spray Water Valve

The spray water valve is an energize to open and energize close valve. The output instruction on rung #27 determines when the skimmer is in any cycle. A N.O. contact of the output instruction on rung 27 opens the spray water valve (rung #28). A N.C. contact of the output instruction on rung 27 closes the spray water valve (rung #27).

End of Transition

The output instruction on rung #30 (EOT) disables this program file and steps to the next skimmer file in the SFC. In general when an EOT output instruction is true, the

receptivity of the output transition of the respective program step (RLL file) is true. Once the output transition of a program step is true, then all the output conditions of the step are reset and program execution begins at the next enabled step. [6, 11].

5.2.1 Summary of Ladder Logic Listing (Appendix B)

As can be seen in Appendix B and in Zhou & Twiss, 1995 [25], it is fairly simple to determine the output of an individual rung. However, to determine the complete sequence of operation from the ladder logic takes a concentrated effort. The difficulty in determining the sequence of operation is due to the fact that the ladder logic listing fails to capture the discrete event dynamics of the system. The components shown in dotted boxes are for design modifications and will be explained later in chapter 6.

5.3 RTPN Design

5.3.1 Top Level Design

A top down approach is used in the RTPN design. Detail on top down Petri net synthesis is discussed in Zhou & Dicesare, 1993 [20] and a laboratory demonstrated example using four pneumatic pistons can be seen in Zhou, et. al., 1994 [18]. The top level design is shown in Figure 5.3. Figure 5.3 is a cross between a Petri net and a flow chart. It models the sequencing of the major elements of the system. Each of the large rectangles represents a model of the major elements of the system. The major elements of the system are: a model for the continuous mode of operation, a model for the timer mode of operation, a model of each skimmer and a model of the scum valve. The model of each

skimmer is broken down into four (4) sub-models. The four sb-models are: the skim cycle model, spray water valve model, high wet well e-stop model, and a model for loss of the remote input when the skimmer is in cycle.

The system can be initiated in either continuous or timer mode. When the continuous mode is selected, transition t_a fires and enables the continuous mode model. The output transition t_b of the continuous mode model enables skimmers 9 - 1 in a decreasing numerical order. Transition t_a has a firing attribute (Y(a) = continuous: D(a) = 0) and transition t_b is an immediate transition, which fires once all input places are marked. Once skimmer #1 completes its operation the scum valve is enabled. If continuous mode is still selected at the completion of the scum valve operation then the cycle repeats until the continuous mode is no longer selected.

When timer mode is selected the timer model output transition t_c fires once every 6-12 hours after completion of the scum valve operation. Transition t_c has a firing attribute (Y(c) = timer mode: D(c) = 6-12hrs). The output transition t_c of the timer mode model enables skimmers 9 - 1 in a decreasing numerical order. Once skimmer #1 completes its operation the scum valve is enabled. If timer mode is still selected at the completion of the scum valve operation then the system continues to repeat one cycle every 6-12 hours, until timer mode is no longer selected.

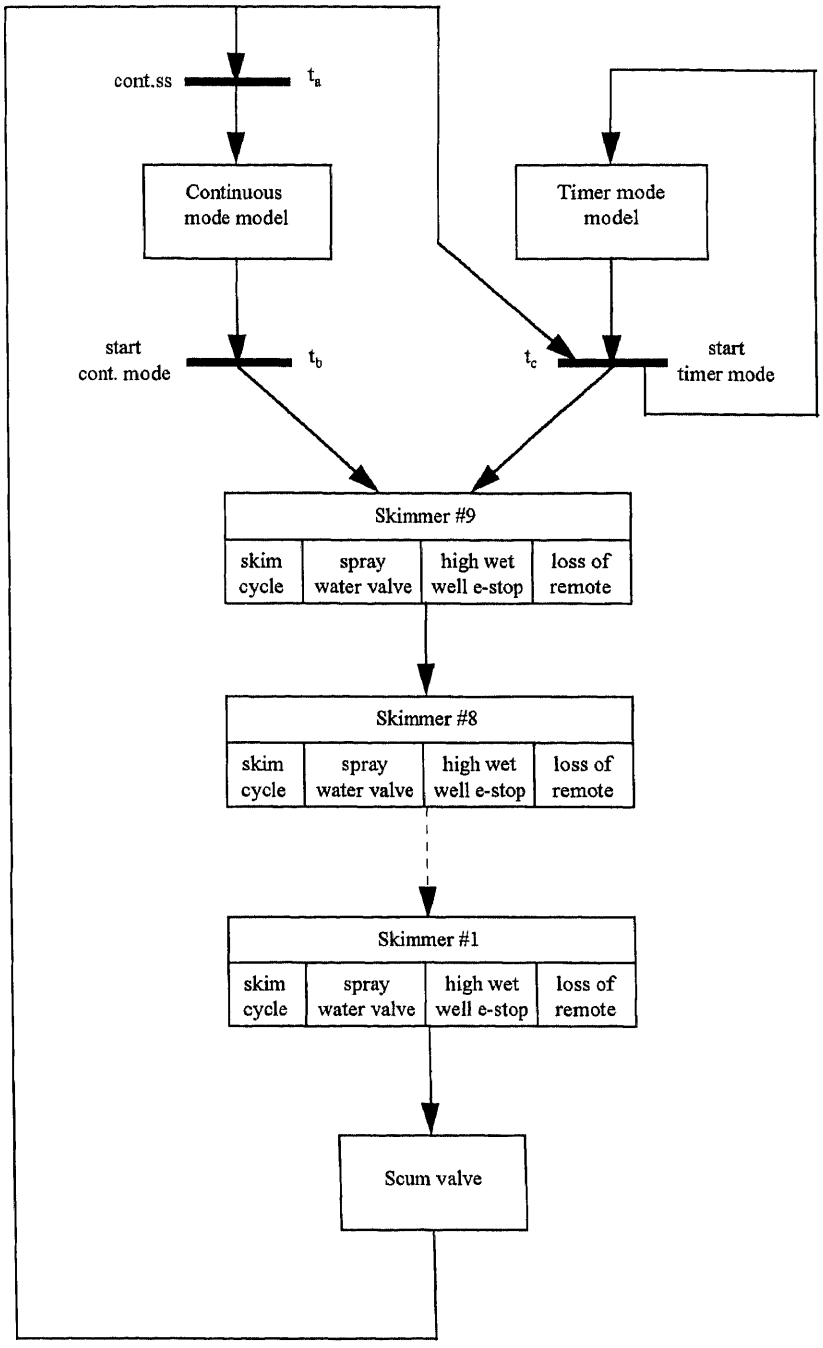


Figure 5.3 - First Level PN Model

5.3.2 Second Level Design

In the second level design (Figure 5.4) the model for skimmer #9 is expanded, and the details of the continuous and timer models are shown.

Skimmer #9 Model Expansion

The model for skimmer #9 is expanded to show the relationship between the different components of the skimmer model. As shown in the first level design the the skimmer model is made up of four components. Three of the components (skimmer seq/high wet well/spray water valve) are synchronized by transitions t_1 and t_{16}. When p_1 is marked by either the continuous or timer mode of operation, immediate transition t_1 fires passing tokens to the skimmer sequence model, the high wet well e-stop model and the spray water valve model. If the remote input from a skimmer is lost while the skimmer is in cycle, the skimmer stops and the next available skimmer (skimmer with remote selected) is enabled. Thus the loss of remote model is shown with connections to the reverse skim, shallow skim and deep skim places of the skimmer sequence model.

Continuous Mode Model Description

When the continuos mode of operation is selected, transition t_{29} fires. Transition t_{29} has a firing attribute of (Y(29) = continuous: D(29) = 0). When transition t_{29} fires, a token is removed from place p_{24} (enable skimmer #9) and deposited into place p_{20} (continuous mode enabled). Place p_{24} is originally marked by the initial marking m_0. When p_{20} is marked, transitions t_{30} and t_{31} are enabled. Transition t_{31} has a firing attribute of (Y(31) =

skimmer #9 remote': D(31) = 0), t_{30} has a firing attribute of (Y(30) = skimmer #9 remote: D(30) = 0). Therefore, if skimmer #9 is selected for remote operation, then t_{30} fires and passes a token to p_1 (skim start request) of skimmer #9. If p_{20} is marked and if skimmer #9 is not selected for remote operation then the skimming sequence is started at skimmer #8. When skimmer #9 has completed its cycle, skimmer #8 is enabled by place p_{25} (enable skimmer #8) being marked. When place p_{25} is marked, and if skimmer #8 is selected for remote operation, then skimmer #8 is enabled. If skimmer #8 is not selected for remote operation then transition t_{37} fires bypassing skimmer #8. Transition t_{37} has a firing attribute of (Y(37) = skimmer #8 remote': D(37) = 0). Once skimmers 8 through 1 and the scum valve have completed their sequences, or have been bypassed, then place p_{24} is marked. When p_{24} is marked and if continuous mode is selected, then the cycle repeats until continuous mode is no longer selected. The dashed arcs in the upper portion of Figure 6, are for design modifications and will be explained later.

Timer Mode Model Description

When the timer mode of operation is selected, transition t_{32} fires. Transition t_{32} has a firing attribute of (Y(32) = timer: D(32) = 0). When transition t_{32} fires a token is removed from place p_{24} (enable skimmer #9) and deposited into place p_{23} (timer mode enabled). Place p_{24} is originally marked by the initial marking m_0. When p_{23} is marked, transitions t_{33} and t_{34} are enabled. Transition t_{34} has a firing attribute of (Y(34) = timer': D(34) = 0), transition t_{33} has a firing attribute of (Y(33) = 1: D(33) = 6-12 hrs), therefore if p_{23} is marked for 6-12 hours then t_{33} fires and passes a token to p_{22} (timer mode remote check).

If timer mode is de-selected while p_{23} is marked then, transition t_{34} fires and p_{24} is re-marked. When p_{22} is marked, t_{35} and t_{36} are enabled. Transition t_{36} has a firing attribute of (Y(36) = skimmer #9 remote': D(36) = 0), and t_{35} has a firing attribute of (Y(35) = skimmer #9 remote: D(35) = 0). Therefore, if p_{22} is marked and if skimmer #9 is selected for remote operation, then t_{35} fires and passes a token to p_1 of skimmer #9. If p_{22} is marked and if skimmer #9 is not selected for remote operation then the skimming sequence is started at skimmer #8. When skimmer #9 has completed its cycle, skimmer #8 is enabled by place p_{25} (enable skimmer #8) being marked. The skimming cycle will sequence through skimmers 8 to 1, similar to the continuous mode of operation. Once skimmers 8 through 1 and the scum valve complete their sequences, or have been bypassed, then place p_{24} is marked. When p_{24} is marked and if timer mode is enabled, then t_{32} fires and the timer cycle is re-initiated.

Evaluation of Second Level Design

The RTPN models for the second level design was evaluated by playing the "token game". The skimmer function boxes were replaced with "normal" places to allow the token game to be played. To play the token game, the player places tokens according to the systems initial marking. The player then looks for any enabled transitions and manipulates the tokens according to the "firing rules" [13]. Incorporating the "token game" into the RTPN programming package would facilitate program development and evaluation. For this example the token player should find this model to be a *safe* and *live* PN. A *safe* and *live* PN is both deadlock free and each place can contain no more than one token (*1-*

bounded). A *safe* and *live* PN also means that no single transition can be fired twice in a row. After this PN was found to be *safe*, *live*, and token flow matched the functional description.

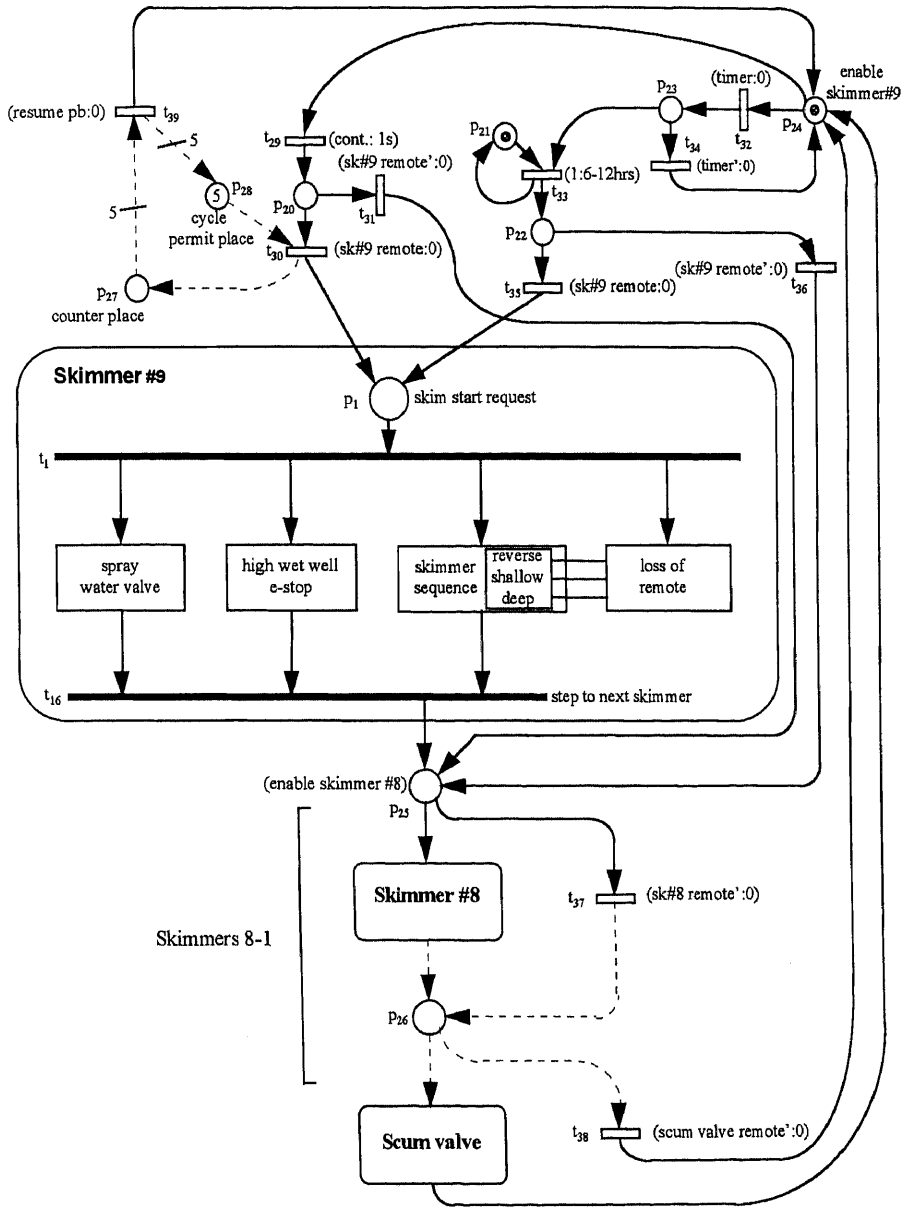


Figure 5.4 - Second Level PN Model

5.3.3 RTPN Model of Skimmer

The RTPN of a skimmer is composed of three components; the skimmer sequence, the high wet well e-stop and spray water valve. Transition t_1 is used to initialze the three sub-nets. Transition t_{16} is used to synchronize the three sub-nets at the end of the cycle. Normal token flow through the typical skimmer RTPN model (Figure 5.5) is as follows: Place p_1 (skim start request) receives a token from the continuous or timer mode of operation (see second level design). When t_1 fires p_2, p_{12} and p_{16} are marked. Place p_{16} enables the high wet well e-stop RTPN. Place p_{12} enables the spray water valve RTPN. When p_2 (skimmer to reverse) is marked, the output address that drives the skimmer to the reverse position is set, (Z(2) = skimmer to reverse). When p_2 is marked, transitions t_2 or t_{14} are enabled to fire. Transition t_2 has a firing attribute of (Y(2) = the reverse limit switch: D(2) = 0), when this transition fires a token is removed from p_2 and deposited into p_3 (reverse skim), reseting Z(2). Transition t_{14} is a pure time delay transition, firing attribute = (1: x seconds), where x is a variable delay preset by an operator. Transition t_{14} fires if the reverse limit switch is not actuated before D(14) time delay expires. When t_{14} fires, place p_{10} (skimmer fault) is marked. This is fault loop is typical for "deep", "shallow" and "rest" actuator faults. Place p_3 sets the output for the reverse skim indicator (Z(3) = reverse skim indicator). When p_3 is marked transitions t_3 or t_6 are enabled to fire, t_3 has a firing attribute of (Y(3) = shallow skim: D(3) = x sec), t_6 has a firing attribute of (Y(6) = deep skim: D(6) = x sec). Depending on the mode selected (shallow or deep), the skimmer will stay in *"reverse"* until the delay associated with t_3 or t_6 expires. The skimmer then enters *"shallow"* or *"deep"* skim mode. When p_4 (skimmer

When p_4 or p_6 is marked, then t_4/t_{12} or t_7/t_{13} are enabled to fire. Transitions t_4/t_7 have a firing attribute of (Y(4) = the shallow limit switch: D(4) = 0/ Y(6) = the deep limit switch: D(6) = 0), when one of these transitions fire a token is removed from p_4/p_6, and deposited into p_5/p_7 (shallow/deep skim skim), thus reseting Z(4)/Z(6). Similar to the reverse skim mode, if the skimmer does not reach the desired position within a preset delay (t_{12} or t_{13}), then t_{12} or t_{13} fires and place p_{10} (skimmer fault) is marked. Places p_5 and p_7 indicate that the skimmer is in the reverse skim position and set the outputs for those respective indicators. The system stays in the deep or shallow skim mode until the delay associated with t_5 or t_8 expires. Place p_8 converges the two paths (shallow/deep). Transition t_9 is an immediate transition and fires as soon as p_8 is marked. When p_9 (skimmer to rest) is marked the output address that drives the skimmer to the rest position is set, (Z(9) = skimmer to rest). When p_9 is marked, transitions t_{10} and t_{11} are enabled to fire. Transition t_{10} has a firing attribute of (Y(10) = the rest limit switch: D(10) = 0), when this transition fires a token is removed from p_9 and deposited into p_{11} (cycle complete), thus reseting Z(9). Similar to the reverse skim mode, if the skimmer does not reach the desired position within a preset delay (t_{11}). Then t_{11} fires and place p_{10} (skimmer fault) is marked.

When an actuator fault occurs during the skim cyclce, place p_{10} is marked. To reset the fault and continue skimming at the next skimmer the operator must hit the resume pushbutton. In the RTPN transition t_{15} has a firing attribute of (Y(15) = resume

pushbutton: D(15) = 0). When p_{10} is marked and the resume pushbutton is hit, then a token is removed from p_{10} (skimmer fault) and depsoited to p_{11} (cycle complete).

Spray Water Valve

The spray water valve is enabled when place p_{12} is marked. When p_{12} is marked, transitions t_{17} and t_{20} are enabled to fire. Place p_{14} is marked by the initial marking m_0. Transition t_{17} has a firing attribute of (Y(17) = spray water valve remote: D(17) = 0), transition t_{20} has a firing attribute of (Y(20) = spray water valve remote': D(20) = 0). If the spray water valve is not selected for remote operation, then t_{20} fires and place p_{15} is marked. If the spray water valve is selected for remote operation, then t_{17} fires and p_{13} is marked. Place p_{13} sets the spray water valve open output. When p_{13} is marked transitions t_{18} and t_{19} are enabled. Transition t_{18} is an immediate transition and fires when p_{11} (skimmer cycle complete) is marked. When t_{18} fires a token is removed and re-deposited into p_{11} by the bi-directional arc, a token is removed from p_{13} (resetting the spray water valve open outpt), and a token is deposited to p_{14}. Place p_{14} sets the spray water valve close output. If p_{13} is marked and the remote mode of operation is de-selected, then t_{19} fires, removing the token from p_{13} and depositing a token into p_{14} and p_{15}. This allows an operator to take local control of the spray water valve while the automatic skimming cycle continues.

High Wet Well E-Stop Description

When the high wet well level e-stop place p_{17} is marked, skimmer operation is disabled by removing all tokens from all places. Place p_{17} is marked when the particular skimmer is in any cycle and a high wet well condition ocurrs. The high wet well condition is automatically reset on a low wet well level. Transitions t_{23} and t_{24} are used to debounce the high and low level switches. These transitions have the following firing attributes (Y(23) = low level switch: D(23) = 3 sec) and (Y(24) = high level switch: D(24) = 3 sec). When the high wet well condition is reset, immediate transition t_{22} fires and the token is removed from the e-stop place p_{17}. When the e-stop place loses its marking, then the initial marking is returned to the particular skimmer were the high wet well condition ocurred. The initial marking for the skimmer is for p_1, p_{12} and p_{18} to be marked. The skimming cycle then re-initializes and continues forward from this point.

Loss of Remote PN Description

Place p_{11} (cycle complete) is marked when the skimmer is in any cycle (reverse/shallow/deep) and the remote input is lost. Transitions t_{25}, t_{26}, t_{27} have a firing attribute of (Y(25-26) = skimmer remote'; D(25-26) = 0).

The following figure (Figure 5.5) is the RTPN model for a typical skimmer. This model includes the skimmer operation, spray water valve operation, high wet well e-stop interlock, and the action of skipping to the next skimmer when remote mode is no longer at the skimmer local control panel.

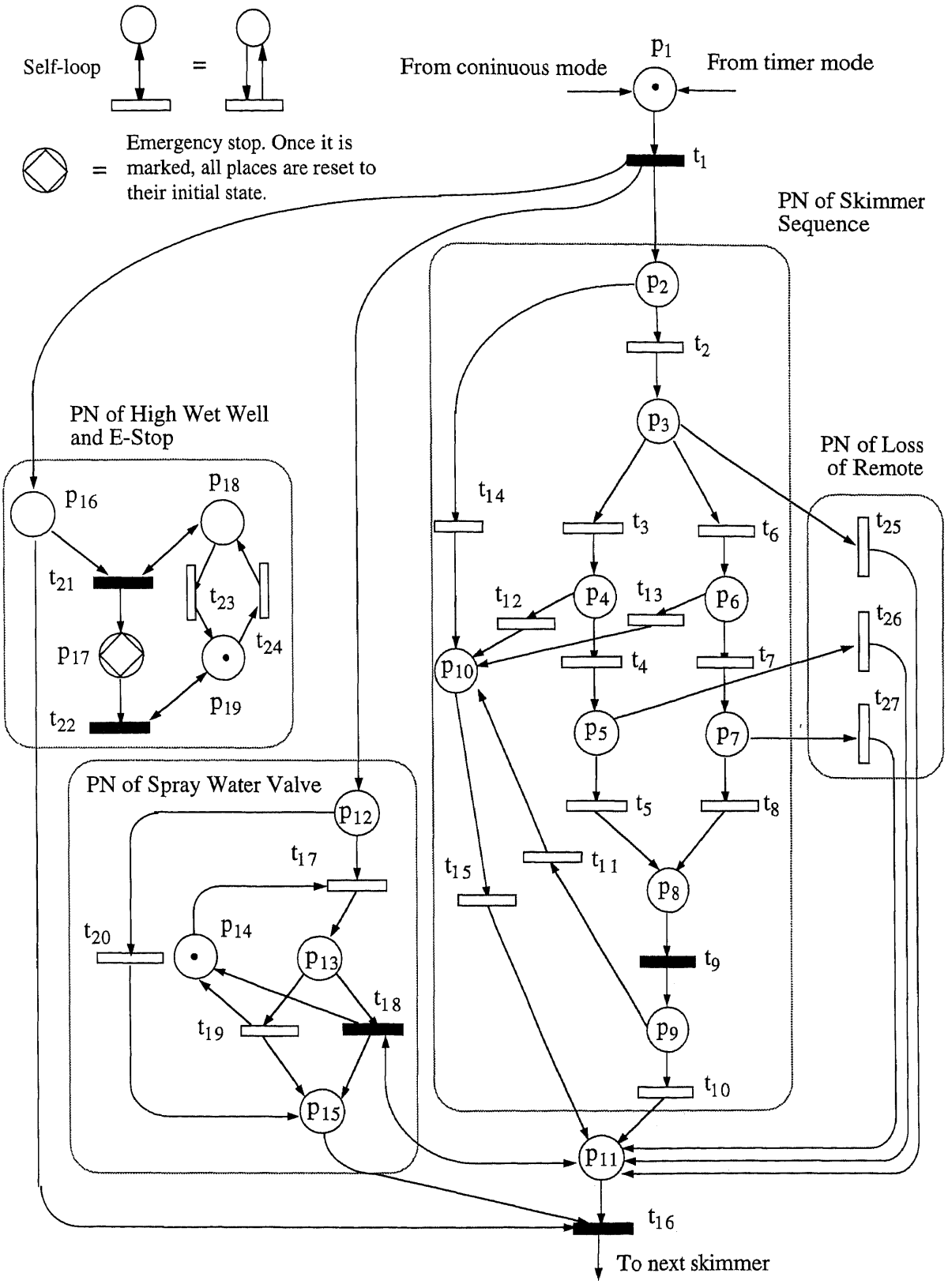


Figure 5.5 - Petri net model of a skimmer

5.3.4 Scum Valve Model

The RTPN model of the scum valve is shown in Figure 5.6. Normal token flow through the scum valve RTPN is as follows: Place p_{30} enables receives a token when skimmer #1 completes its skim cycle. Transition t_{41} is an immediate transition, when t_{41} fires a token is removed from p_{30} and deposited to p_{31}. Place p_{31} sets the output that opens the scum valve, (Z(31) = open scum vave). When p_{31} is marked transitions t_{42} and t_{43} are enabled. Transition t_{43} fires when the scum valve open limit switch is actuated, (Y(43) = scum valve open limit switch; D(43) = 0). When t_{43} fires place p_{32} is marked. Place p_{32} indicates the scum valve is open (Z(32) = scum valve open indicator). Place p_{32} will remain marked until the delay associated with t_{44} expires. Transition t_{44} has a firing attribute of (Y(44) = 1; D(44) = x sec). When t_{44} fires p_{33} is marked, p_{33} sets the output that closes the scum valve. When the scum valve closes, t_{45} (Y(45) = scum valve closed limit switch; D(45) = 0), fires and marks place p_{34}. Place p_{34} indicates that the scum valve has completed its cycle. When p_{34} is marked immediate transition t_{46} fires and passes a token to skimmer #9.

Scum valve actuator fault is initiated when the valve is commanded to open or close and does not reach its full open or closed position within a predetermined time. Transitions t_{42} and t_{48} are pure time delay transitions. These transitions fire and mark p_{45} (skimmer fault) when the scum valve does not reach its open or closed position within a predetermined time. To reset the fault condition the resume pushbutton must be depressed. Transition t_{47} has a firing attribute of (Y(47) = resume pushbutton; D(47) = 0). If the valve repeatedly fails to close the fault must be manually corrected.

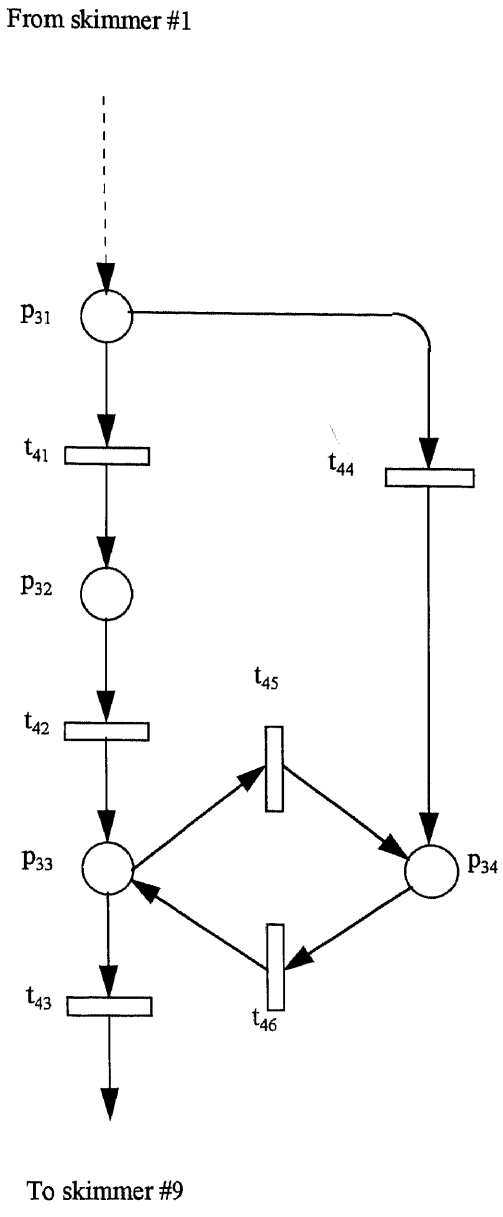


Figure 5.6 - PN Model of Scum Valve

CHAPTER 6

DESIGN COMPARISON

6.1 Understandability

The RLL program listing produces approximately 450 rungs of code, or 90 pages, see Zhou & Twiss [25], while the RTPN program produces four Petri net designs structures (Figures 5.3, 5.4, 5.5 and 5.6) plus a table (Appendix A) to interpret the meanings of places and transitions. RTPN's make it easier to implement a hierarchal structure. This results in a more compact design algorithm. One could argue, the ladder logic program consists of 9 subroutines. The only difference between these subroutines is the addresses and tag numbers. Therefore, once one subroutine is understood, all are understood. Each RLL subroutine is approximately 30 rungs, or 7 pages, roughly equivilant to the RTPN. As mentioned earlier the RLL program was written with the assistance of a supervisory SFC. The SFC simplifies the sequencing of each RLL subroutine. However, when many subroutines are used in a pure RLL file it becomes difficult to determine how and in what order they are executed. This is because RLL is similar to a line number oriented programming language. Every rung is scanned in sequential order wether the conditions for the rung output instruction are true or false (jump to subroutine is considered a rung output instruction). RTPN is similar to a procedural programming language, e.g., C or Pascal and can be used to outline the execution of a program. In a RTPN, only sections of logic that are enabled are executed. The order of execution is determined by the structure of the RTPN itself. As mentioned earlier RLL fails to capture the time/event dependency

of the different events taking place in the system. The RTPN model, however, explicitly models the skimmer sequencing.

6.2 Simulation

Simulation is inherently built into RTPN's. By playing the "token game," a system designer could follow the program logic step by step before the software is loaded into the controller. Simulation is not inherently built into RLL. Several software manufacturers have recently added the ability to create "historgrams," or bar charts that show the timing sequence of several input and output conditions. A single historgram, however, cannot be used to show the timing sequence of every input and output instruction in a RLL program of any measurable size. The average size of a histogram is 5 to 10 horizontal timing bars. This requires the designer to build many histograms to monitor the RLL program.

6.3 Flexibility

To determine the flexibility of each algorithm, the following changes will be made to the original design specifications for the waste water treatment system described above:

- A time delay is to be inserted before a skimmer begins the reverse skim operation.
- Continuous mode is to be modified to allow only five complete skimming cycles.
- The RTPN sequence is to be modified as follows: rest to reverse, reverse to shallow, shallow to deep, deep to rest, all with adjustable time delays as before.

6.3.1 Time Delay

The insertion of a time delay before the reverse skim operation begins has no effect on the structure of the RTPN. The immediate transition t_1 that follows p_1 in Figure 5.5, is changed from an immediate transition to a timed transition. Changing a transition from immediate to timed has no effect on the RTPN structure.

The insertion of a time delay in the RLL program requires an additional rung to include the timer function, an additional rung output instruction (coil), and modification to existing rungs of ladder logic. The original output instruction (coil) is replaced by the new output instruction (coil). A normally open contact of the new output instruction was used to enable the new timer. The done bit of the timer was then used to set the original output instruction (coil) that started the reverse skim mode operation. See Appendix B for RLL modifications.

6.3.2 Maximum Cycle Counter

The insertion of a counter requires modification to the basic structures of both designs. The RTPN requires two additional places p_{27} (counter place), p_{28} (cycle permit place) and an additional transitions t_{39}. Transition t_{39} has a firing attribute of (Y(39) = resume PB: D(39) = 0). The "counter place" has an output arc of weight 5 to the new transition and an input arc of weight 1 from transition t_{30}. When the system is in continuous mode, each time transition t_{30} fires, a token is removed from p_{28} and deposited to p_{27}. After t_{30} fires five times, p_{28} has 0 tokens, t_{30} is disabled, p_{27} has five tokens, and t_{39} is enabled if the resume pushbutton is hit. When transition t_{39} fires all five tokens are removed from p_{27}, five

tokens are deposited to p_{28} and one to p_{24}. Place p_{28} is initially marked with five tokens by m_0. The required modifications are shown in Figure 5.4 using dashed arcs.

The insertion of a cycle counter in the RLL program requires modifications to existing rungs and two new rungs. One rung to include the counter function and another to reset the counter. The modifications are as follows (see Appendix B for the modifications):

- A normally open contact of the original rung output instruction (coil) used to start the system in continuous mode was used on a new rung to increment the counter each time a new cycle is started.
- To reset the counter the resume PB input was used.
- To prevent more than five cycles from occurring at once the "not" (normally closed contact) of the counter done bit was placed in series with the original output instruction that started the system when continuous is selected.

6.3.3 Sequence Modification

Modification of the sequence to eliminate the choice structure of the system requires modifications to the structures of both algorithms. The elimination of the choice structure simplifies the RTPN structure by eliminating the parallel paths after p_3 . The elimination of the choice structure in the RLL program does not simplify this algorithm. See [25] for a comparison of the old and new RTPN's and RLL.

6.4 Diagnostics

The diagnostic features of RLL programming packages have been consistently improved over the past 20 years. With today's RLL packages the systems engineer can search for a certain output statement, observe the precondition logic and backtrack to the source of the problem. RLL packages also offer the capability to "force" override the precondition logic for an output statement. This allows engineers to determine if the fault is in the hardware or software. A RTPN software package would have to have at minimum the above capabilities. However the main advantage the RTPN could have over RLL would be on-line monitoring of token flow.

6.5 Documentation

As can be seen in Zhou & Twiss [25] the RLL program is built around an extensive data-base. Also at the bottom of each rung is a cross-reference of where the output instruction is used throughout the program. Programmers are required to describe (for their own benefit) each and every instruction used in the program. For an average program they spend approximately 25% to 50% of their time editing the database. A RTPN package would require the description and I/O mapping of each place and transition. This should result in a smaller database due to the elimination of internal flags required in RLL, which are not be needed in the RTPN model. Another advantage to an RTPN package would be that documentation is "built into" the control program by virtue of the model itself.

CHAPTER 7

CONCLUSION

ISA Logic diagrams and Timing/Sequence diagrams are valuable tools in the development and maintenance of relay ladder logic control systems. Graphical modeling approaches such as Petri nets and Sequential Function Charts tend to be more compact than their corresponding relay ladder logic programs. Graphical modeling techniques capture the discrete event dynamics of a system. These approaches show how the controller operates instead of how it is implemented [1]. This enables quicker understanding, easier maintenance, and reduced programming errors.

Petri nets hold a promise as a solution to modern industrial control problems. They display greater flexibility and understandability of the process than today's standard RLL programming techniques. To speed up their applications to industrial discrete event control problems, more standard software tools have to be developed. It should be noted that several companies in Japan, e.g., Hitachi, Kobe Steel, have invested in this area and achieved a significant savings in system development time compared with traditional RLL and general-purpose programming approaches [14, 19]. A benchmark study on a variety of methodologies should be helpful in convincing engineers to accept new approaches such as Petri nets. Future work includes benchmark studies among sequential function charts, Petri nets, structured text, and state machine methods.

APPENDIX A

PLACE/TRANSITION DESCRIPTION TABLE

Place/Transition Table

Place	Description	Attributes
P1	Skimmer #9 start request	status only
P2	Skimmer #9 moving to reverse position	(Z(2) = skimmer to reverse)
P3	Skimmer #9 in reverse skim mode	(Z(3) = reverse skim indicator)
P4	Skimmer #9 moving to shallow position	(Z(4) = skimmer to shallow)
P5	Skimmer #9 in shallow skim mode	(Z(5) = shallow skim indicator)
P6	Skimmer #9 moving to deep position	(Z(6) = skimmer to deep)
P7	Skimmer #9 in deep skim mode	(Z(7) = deep skim indicator)
P8	Skimming complete	status only
P9	Skimmer #9 moving to rest position	(Z(9) = skimmer to rest)
P10	Skimmer #9 fault	status only
P11	Skimmer #9 cycle complete	status only
P12	Enable spray water valve	status only
P13	Spray water valve open	(Z(13) = spray water valve open)
P14	Spray water valve closed	(Z(14) = spray water valve closed)
P15	Spray water valve loss of remote	status only
P16	Enable high wet well e-stop	status only
P17	High wet well e-stop	Disables all outputs when marked/resets RTPN by placing a single token in P(1)when t(22) fires.
P18	Hi wet well level	status only
P19	Low wet well level	status only
P20	Continuous mode start request	status only
P21	6-12 hour timer enable	status only
P22	Timer mode start request	status only
P23	Timer mode selected check	status only
P24	Enable skimmer #9	status only

P25	Enable skimmer #8	status only
P26	Enable scum valve	status only
P27	Not used	
P28	Not used	
P29	Not used	
P30	Not used	
P31	Open scum valve	(Z(31) = open scum valve)
P32	Scum valve open	(Z(32) = scum valve open indicator)
P33	Close scum valve	(Z(33) = close scum valve)
P34	Scum valve fault	status only

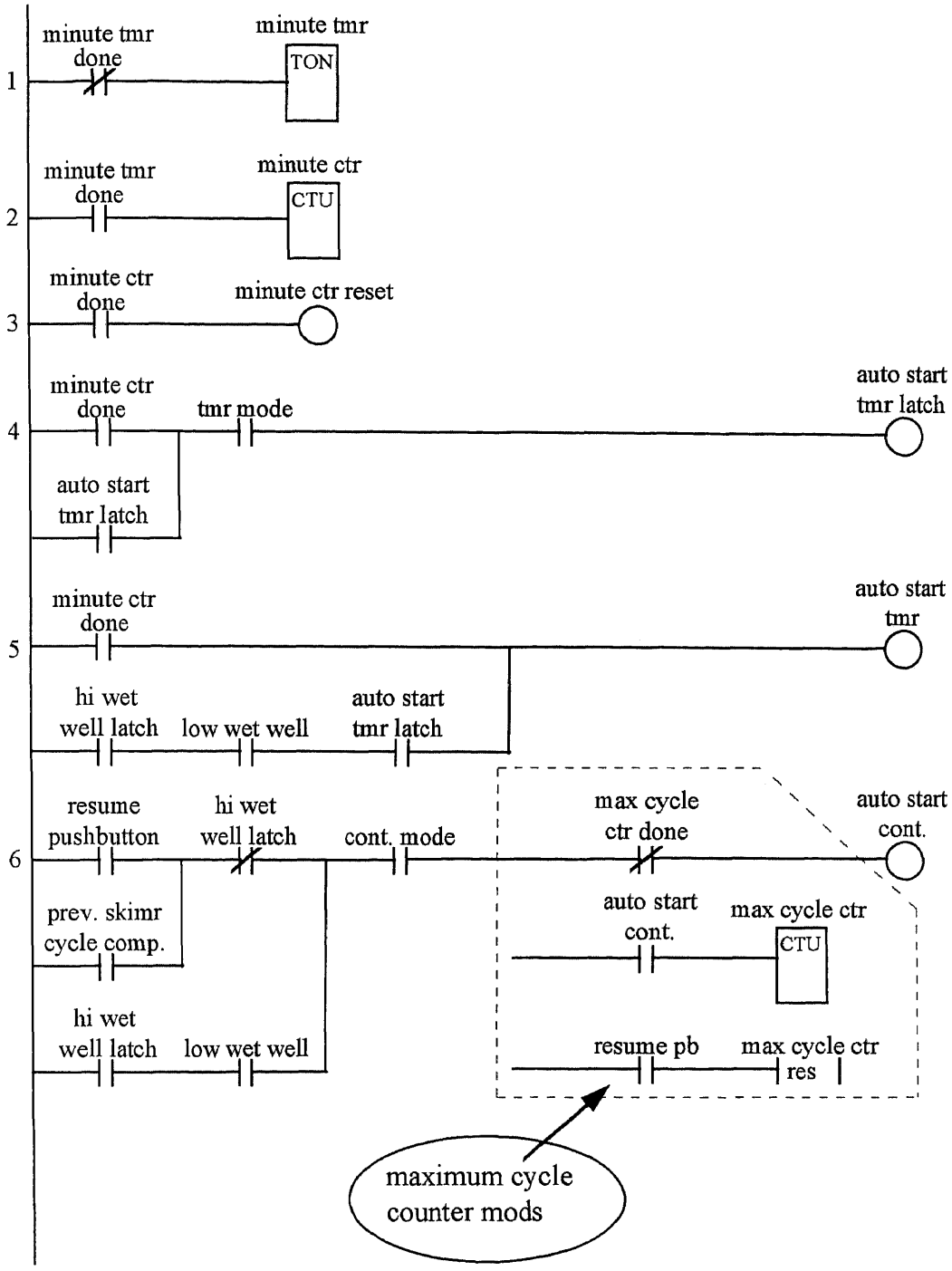
Trans	**Description**	**Attributes**
T1	Initiate skim cycle	Immediate transition
T2	Reverse position limit switch	(Y(2) = reverse skim limit switch; D(2) = 0)
T3	Reverse skim duration 1	(Y(3) = 1; D(3) = 10-60 seconds)
T4	Shallow position limit switch	(Y(4) = shallow skim limit switch; D(4) = 0)
T5	Shallow skim duration	(Y(5) = 1; D(5) = 10-120 seconds)
T6	Reverse skim duration 2	(Y(6) = 1; D(6) = 10-60 seconds)
T7	Deep position limit switch	(Y(7) = deep skim limit switch; D(7) = 0)
T8	Deep skim duration	(Y(8) = 1; D(8) = 10-120 seconds)
T9	Skimming complete	Immediate transition
T10	Rest position limit switch	(Y(10) = rest position limit switch; D(10) = 0)
T11	Skimmer to rest fault check	(Y(11) = "not" rest position limit switch; D(11) = 45 seconds)
T12	Skimmer to shallow fault check	(Y(12) = "not" shallow skim limit switch; D(12) = 45 seconds)
T13	Skimmer to deep fault check	(Y(13) = "not" deep skim limit switch; D(13) = 45 seconds)

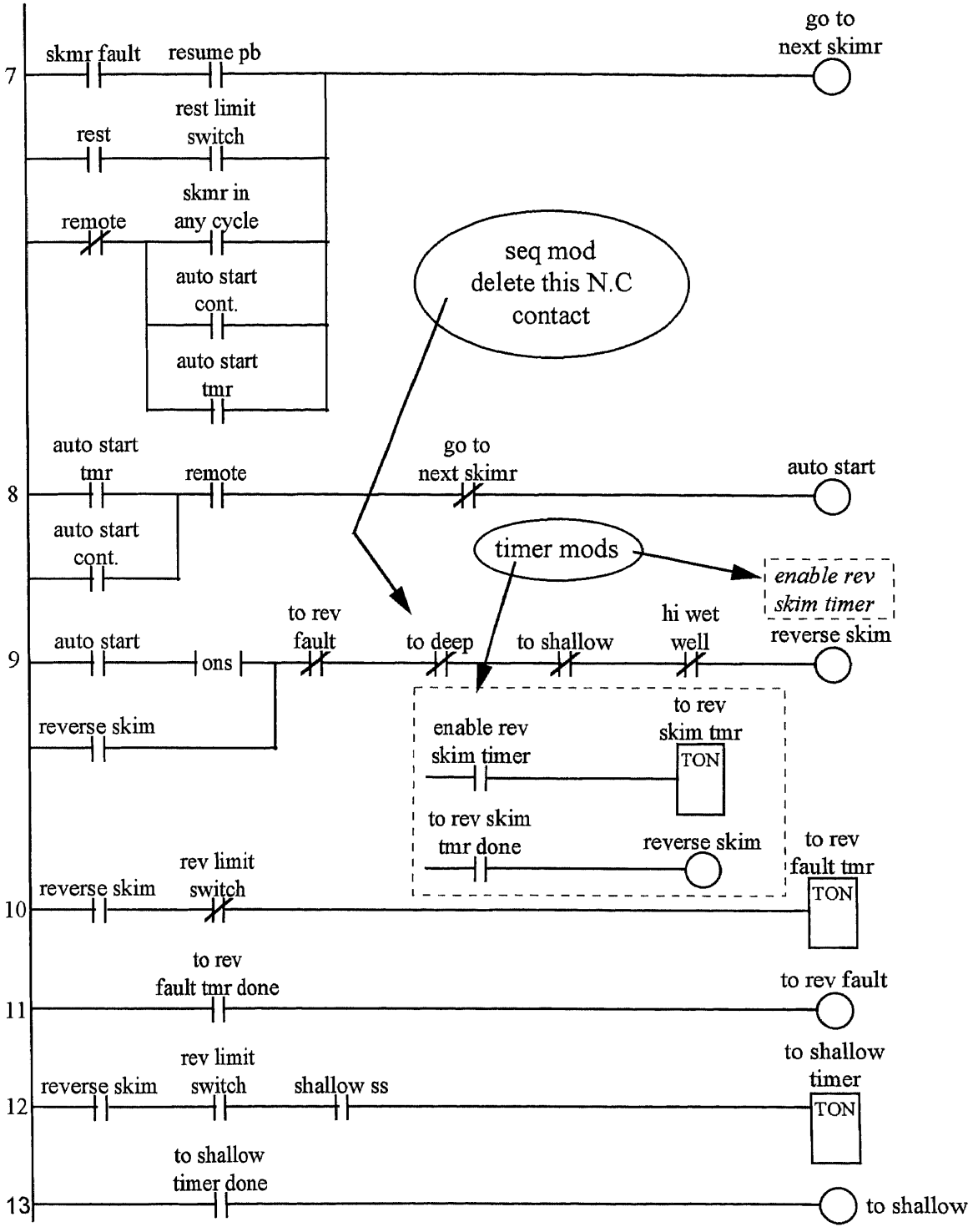
T14	Skimmer to reverse fault check	(Y(14) = "not" reverse skim limit switch; D(14) = 45 seconds)
T16	Step to next skimmer	Immediate transition
T17	Spray water valve remote check 1	(Y(17) = spray water valve remote; D(17) = 0)
T18	Initiate close spray water valve	Immediate transition
T19	Spray water valve remote check 2	(Y(19) = spray water valve "not" remote; D(19) = 0)
T20	Spray water valve remote check 3	(Y(20) = spray water valve "not" remote; D(20) = 0)
T21	Initiate high wet well e-stop	Immediate transition
T22	Reset high wet well e-stop	Immediate transition
T23	Low wet well level switch debounce	(Y(23) = wet well low level switch; D(23) = 3 s)
T24	High wet well level switch debounce	(Y(24) = wet well high level switch; D(23) = 3 s)
T25	Skimmer #9 "not" in remote	(Y(25) = skimmer#9 "not" in remote; D(25) = 1s)
T26	Skimmer #9 "not" in remote	(Y(26) = skimmer#9 "not" in remote; D(26) = 1 s)
T27	Skimmer #9 "not" in remote	(Y(27) = skimmer#9 "not" in remote; D(27) = 1 s)
T28	Not used	
T29	Initiate continuous operation	(Y(29) = skimmer#9 continuos mode selected; D(29) = 0)
T30	Continuous mode remote check 1	(Y(30) = skimmer#9 remote mode selected; D(30) = 0)
T31	Continuous mode remote check 2	(Y(30) = skimmer#9 "not" in remote mode; D(30) = 0)

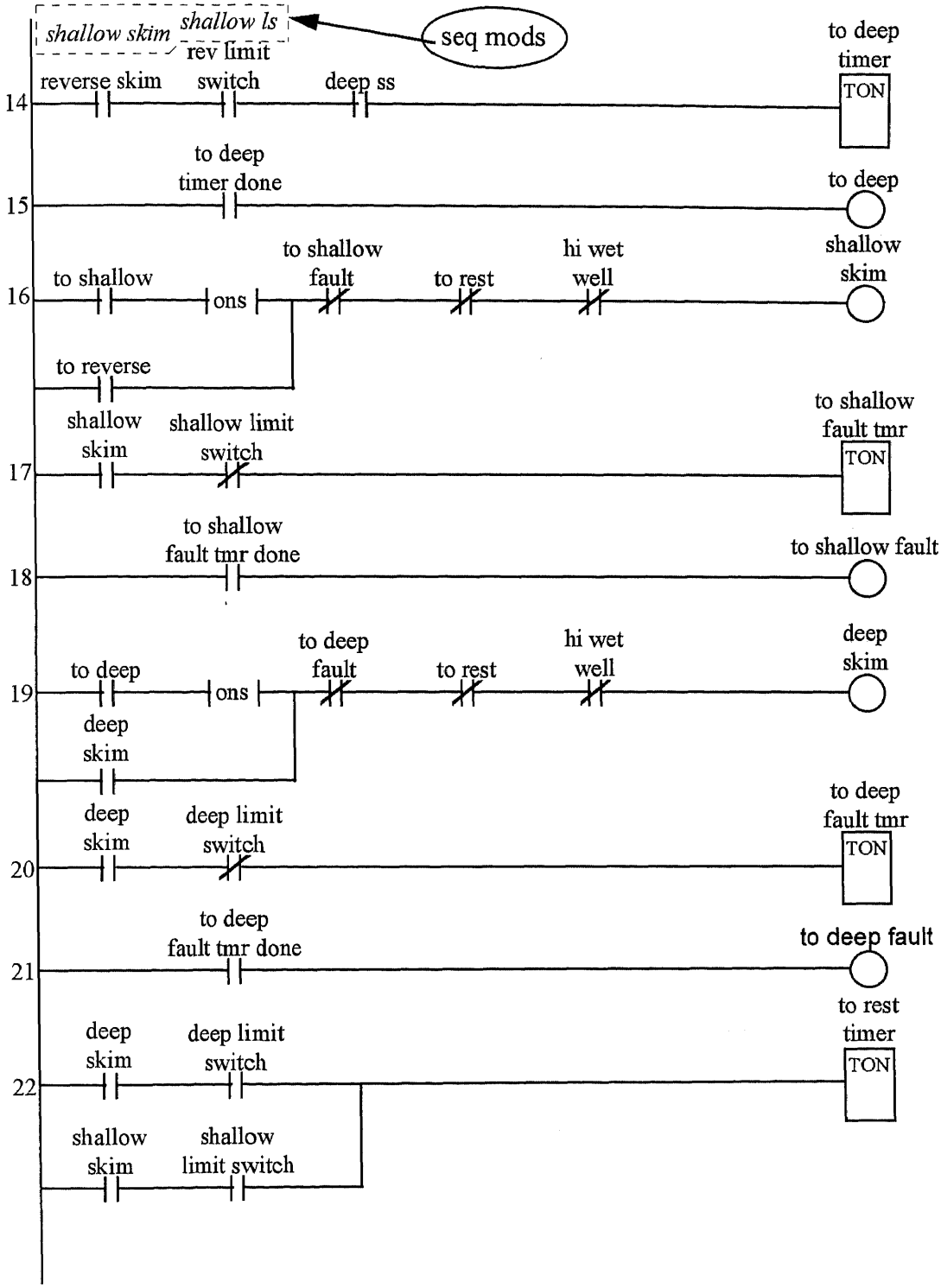
T32	Initiate timer operation	(Y(32) = skimmer#9 timer mode selected; D(32) = 0)
T33	Timer mode initiate delay	(Y(33) = 1; D(33) = 6-12 hours)
T34	Timer mode check	(Y(34) = skimmer#9 timer mode selected; D(34) = 0)
T35	Timer mode remote check 1	(Y(35) = skimmer#9 remote mode selected; D(35) = 0)
T36	Timer mode remote check 2	(Y(36) = skimmer#9 "not" in remote mode; D(36) = 0)
T37	Skimmer #8 remote mode check	(Y(37) = skimmer#8 "not" in remote mode; D(37) = 0)
T38	Scum valve remote mode check	(Y(38) = scum valve "not" in remote mode; D(38) = 0)
T39	Not used	
T40	Not used	
T41	Scum valve open limit switch	(Y(41) = scum valve open limit switch; D(41) = 0)
T42	Scum valve open duration	(Y(42) = 1; D(42) = 30 - 120 seconds)
T43	Scum valve closed limit switch	(Y(43) = scum valve closed limit switch; D(43) = 0)
T44	Scum valve fail to open fault	(Y(44) = "not" scum valve open limit switch; D(44) = 45 seconds)
T45	Scum valve fail to close fault	(Y(45) = "not" scum valve closed limit switch; D(45) = 45 seconds)
T46	Reset scum valve fault	(Y(46) = resume pushbutton; D(46) = 0)

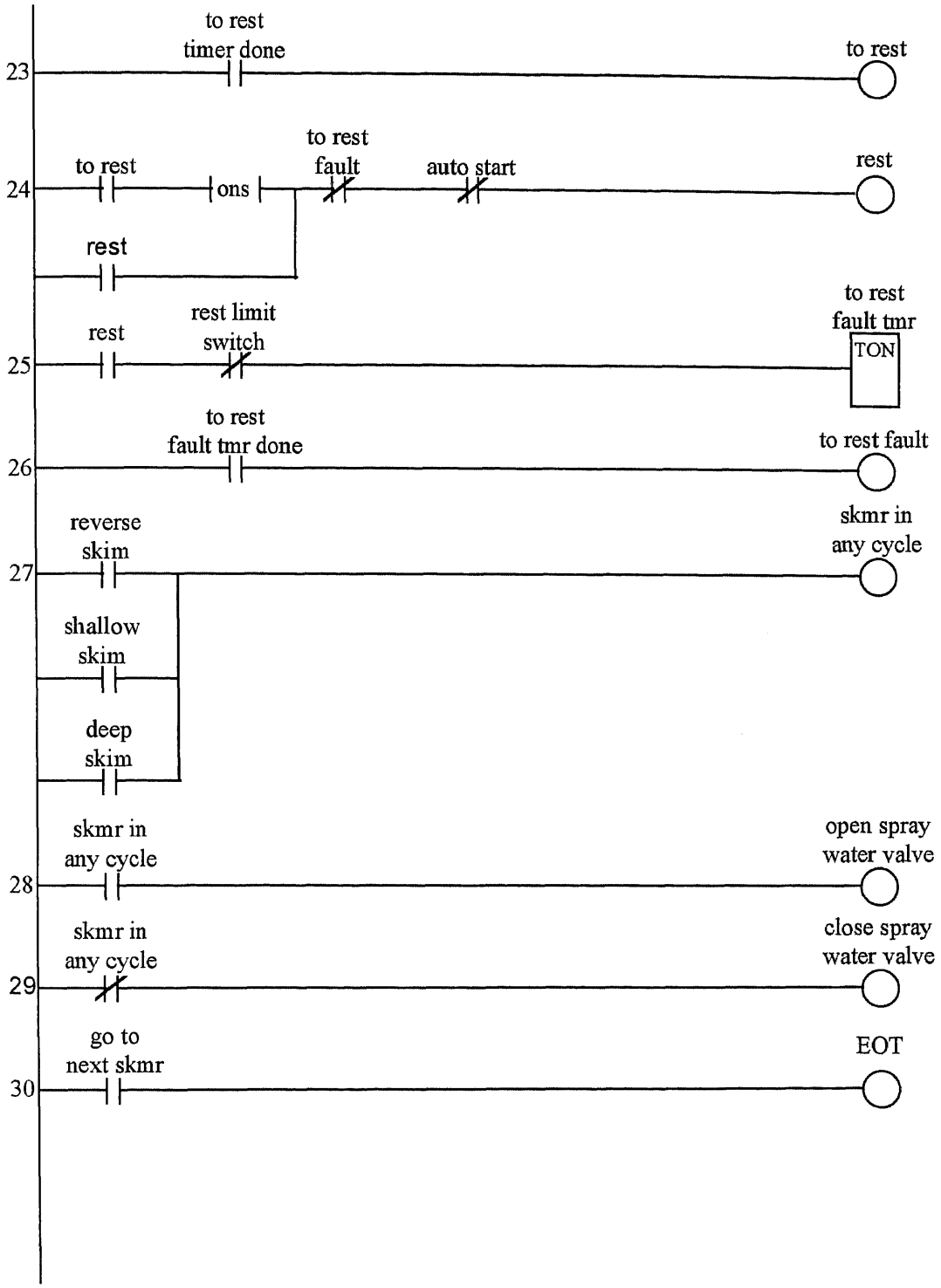
APPENDIX B

PLC LADDER LISTING OF TYPICAL SKIMMER WITH SEQUENCE MODIFICATIONS SHOWN


1
minute tmr done
minute tmr
TON
2
minute tmr done
minute ctr
CTU
3
minute ctr done
minute ctr reset
4
minute ctr done
tmr mode
auto start tmr latch
auto start tmr latch
5
minute ctr done
auto start tmr
hi wet well latch
low wet well
auto start tmr latch
6
resume pushbutton
hi wet well latch
cont. mode
max cycle ctr done
auto start cont.
prev. skimr cycle comp.
auto start cont.
max cycle ctr
CTU
hi wet well latch
low wet well
resume pb
max cycle ctr res
maximum cycle counter mods


7
skmr fault
resume pb
go to next skimr
rest
rest limit switch
remote
skmr in any cycle
auto start cont.
auto start tmr
seq mod
delete this N.C contact
8
auto start tmr
remote
go to next skimr
auto start
auto start cont.
timer mods
enable rev skim timer
9
auto start
ons
to rev fault
to deep
to shallow
hi wet well
reverse skim
reverse skim
enable rev skim timer
to rev skim tmr
TON
to rev skim tmr done
reverse skim
10
reverse skim
rev limit switch
to rev fault tmr
TON
11
to rev fault tmr done
to rev fault
12
reverse skim
rev limit switch
shallow ss
to shallow timer
TON
13
to shallow timer done
to shallow


shallow skim
shallow ls
rev limit
seq mods
14
reverse skim
switch
deep ss
to deep timer
TON
15
to deep timer done
to deep
16
to shallow
ons
to shallow fault
to rest
hi wet well
shallow skim
to reverse
17
shallow skim
shallow limit switch
to shallow fault tmr
TON
18
to shallow fault tmr done
to shallow fault
19
to deep
ons
to deep fault
to rest
hi wet well
deep skim
deep skim
20
deep skim
deep limit switch
to deep fault tmr
TON
21
to deep fault tmr done
to deep fault
22
deep skim
deep limit switch
to rest timer
TON
shallow skim
shallow limit switch


23
to rest
timer done
to rest
24
to rest
ons
to rest
fault
auto start
rest
rest
25
rest
rest limit
switch
to rest
fault tmr
TON
26
to rest
fault tmr done
to rest fault
27
reverse
skim
skmr in
any cycle
shallow
skim
deep
skim
28
skmr in
any cycle
open spray
water valve
29
skmr in
any cycle
close spray
water valve
30
go to
next skmr
EOT

REFERENCES

1. A.D. Baker, T.L. Johnson, D.I. Kerpelman, and H.A. Sutherland, Grafcet and SFC as Factory Automation Standards Advantages and Limitations", *Proc. of the 1987 American Control Conference*, Minn, MN, pp. 1725-1730, June 1987.

2. T.O. Boucher, M.A. Jafari, and A.G. Meredith, "Petri Net Control of an Automated Manufacturing Cell," *Computers and Indu. Eng.*, 17(1-4), pp. 459-463, 1989.

3. G. Bruno, and G. Marchetto, "Process translatable Petri Nets for the Rapid Prototyping of Process Control Systems," *IEEE Trans. on Software Eng.*, 12 (2), pp. 346-356, 1986.

4. L. A. Bryan, Programmable Controllers - Theory and Implementation, Industrial Text, Chicago, IL., 1988.

5. D. Crockett, A.A. Desrochers, F. DiCesare, and T. Ward, "implementation of a Petri Net Controller for a Machining Workstation," *Proc. of IEEE Int. Conf. Robotics and Automation*, Raleigh, NC, pp. 1861-1867, 1987.

6. R. David and H. Alla, Petri Nets and Grafcet, Tools for Modeling Discrete Event Systems, Prentice Hall, N.Y., N.Y., 1992.

7. A. A. Desrochers and F. DiCesare, "Modeling, Control, and Performance Analysis of Automated Manufacturing Systems Using Petri Nets", Control and Dynamic Systems, C.T. Leondes, Ed., N.Y., N.Y.: Academic, Vol. 47, pp. 121-172, 1991.

8. A. A. Desrochers, Modeling and Control of Automated Manufacturing Systems, IEEE Computer Society Press, Washington, D.C., 1990.

9. A. Falcoine and B. H. Krogh, "Design Recovery for Relay Ladder Logic," *IEEE Control Systems*, 13(2), pp. 90-98, 1993.

10. D.G. Johnson, Programmable Controllers for Factory Automation, Dekker, N.Y., N.Y., 1987.

11. International Electrotechnical Commission, Technical Committee 65: Industrial Process Measurement and Control, Subcommittee 65A, Working Group 6 (1990). Part 3: *Programming Languages*, March 15, 1990.

12. G. Michel, Programmable Logic Controllers, Architecture and Applications, John Wiley & Sons, N.Y., N.Y., 1991.

13. T. Murata, "Petri Nets: Properties, Analysis and Applications", Proceedings of the IEEE, 77(4), pp.541-580, 1989.

14. T. Murata, N. Komoda, K. Matsumoto and K. Haruna, "A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation", *IEEE Trans. on Industrial Electronics*, pp. 1-8, 1986.

15. T. Murata and I. Suzuki "A Method for Stepwise Refinements and Abstractions of Petri Nets", *Journal of Comp. and Syst. Science*, 27, pp. 51-76, 1983.

16. L. Ready, "Programming PLC's with Sequential Logic", *Control Engineering*, pp. 101-107, November 1991.

17. R. Vallette, "Analysis of Petri Nets by Stepwise Refinements", *Journal of Comp. and Syst. Science*, 18, pp. 35-46, 1979.

18. K. Venkatesh, M.C. Zhou, and R. J. Caudill, "Comparing Ladder Logic and Petri Nets for Sequence Controller Design through a Discrete Manufacturing System", IEEE Trans. on Industrial Electronics, 41(6), pp. 611-619, 1994.

19. M.C. Zhou, Petri Nets in Flexible and Agile Automation, Kluwer Academic, Boston, MA, 1995

20. M.C. Zhou and F. DiCesare, Petri Nets Synthesis for Discrete Control of Manufacturing Systems, Kluwer Academic, Boston, MA, 1993.

21. M.C. Zhou, K. McDermott and P.A. Patel, "Petri Net Synthesis and Analysis of a Flexible Manufacturing System Cell", *IEEE Trans. on Man & Cybernetics*, 23(2), pp. 523-531, March/April 1993.

22. M.C. Zhou, F. DiCesare, and D. Rudolph, "Design and Implementation of a Petri Net Based Supervisor for a Flexible Manufacturing System", *Automatica*, 28(6), pp. 1999-2008, 1992.

23. M.C. Zhou and E. Twiss, "A Comparison of Relay Ladder Logic Programming and Petri Net Approach Sequential Industrial Control Systems", *Proc. of the 4th IEEE Conf. on Control Applications*, Albany, NY, pp. 748-753, September 1995.

24. M.C. Zhou and E. Twiss, "Discrete Event Control Design Methods: A Review", to appear in *Preprints of the 13th IFAC World Congress*, San Francisco, CA, July 1996.

25. M. C. Zhou and E. Twiss, "A Comparison of Relay Ladder Logic Programming and Petri Net Synthesis for Control of Discrete Event Systems", Technical Report #9501, Discrete Event Systems Laboratory, ECE, New Jersey Institute of Technology, Newark, N.J., 1995.