Implementing supervisory control maps with PLC

Mohammad Moniruzzaman

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science
in Electrical and Computer Engineering at
Concordia University
Montréal, Québec, Canada

August 2006

Your file Votre référence
ISBN: 978-0-494-20753-6
Our file Notre référence
ISBN: 978-0-494-20753-6

Abstract

Implementing supervisory control maps with PLC

Mohammad Moniruzzaman

The supervisory control theory of Discrete-Event Systems (DES) can be used to construct a supervisor for any event-driven system in which the state space is discrete. To implement supervisors we propose to use Programmable Logic Controllers (PLC), which are widely used in industrial applications. In our work, we develop a new conversion algorithm which directly transforms a supervisor represented by a finite automaton to a Ladder Logic Diagram (LLD). To demonstrate the correctness of our proposed approach and discuss the issues that may arise in modeling and development of DES supervisors, we design supervisors for a boiler control system using supervisory control theory of Ramadge and Wonham, convert DES supervisors to PLC controllers using our conversion technique, and verify by setting a virtual plant setup using a PLC simulation software that the converted LLD can be executed by the PLC and that the original behavior of the DES supervisors under PLC implementation can be achieved.

Acknowledgments

In the first aspect I would like to express my sincere gratitude and thanks to my supervisor Dr. Peyman Gohari for his valuable suggestions, guidance and patience throughout the thesis period. Without his precious privilege and teaching, I would never have acquainted with the realm of the research. Much of this work has been established by his insightful knowledge as an exceptional researcher.

Many thanks to Mr. Liang Du and other members of the group for their friendly co-operation and information. Thanks to Famic Technologies, Canada for their technical support and information about the Automation Studio software. I would like to thank the members of my defense committee Dr. Rabin Raut, Dr. Shahin Hashtrudi Zad and Dr. Ali Dolatabadi for their valuable comments and evaluation.

My roommate M.M.A. Hayder, a Ph.D. student, is the only friend with whom I shared my research experiences and feelings; I do appreciate his friendliness and would like to thank him. Special thanks to my well-wisher Junaid Mohaimin. I would also like to thank those who co-operated and helped me in many ways during my study period.

Finally, I would like to convey my heartiest appreciations and thanks to my family members, especially my parents Mohammad Muklesur Rahman and Kazi Momtaz Begum, and my wife Afsana Siddique, for their continuous support, patience and inspiration, and of course for their long cherished dream about me.

Mohammad Moniruzzaman

August 2006.

To my loving parents, wife and son Ridwan Zaman Roddur .....

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of the thesis

The present thesis follows two objectives. The main objective is to develop a general conversion algorithm which can be used to implement theoretical Discrete-Event System (DES) supervisors using physical devices called Programmable Logic Controllers (PLC). To verify the conversion technique and discuss the issues that may arise in modeling and development of DES supervisors, the real-field application to a boiler control system is investigated.

The Supervisory Control Theory (SCT) of discrete-event systems, introduced by Ramadge and Wonham (RW) [1] is a general theory to design supervisors for a wide range of discrete-event systems found in real-life applications. A (often minimally restrictive) supervisor, modeled by a finite automaton, is designed so that the system under supervision satisfies the specification of some desired behaviors. The supervisor can be implemented using PLC or other specialized hardware.

A PLC is a microprocessor-based specialized computer that performs many types of complex logic-based control functions [2], [3], [4]. PLC programs can be written by LLDs (Ladder Logic Diagrams) or Sequential Function Charts (SFCs) or other PLC programming languages [2], [3], [4]. In our work we choose LLD to program PLC.

The conversion of supervisor's automaton to LLD has already been addressed by many researchers [5], [6], [7], [8]. We observe that the implementation of RW supervisor using PLC lies intuitively on the theory itself, i.e. how controllable and uncontrollable events are translated by the designer, and mapped the actual event interpretations to the real systems using associated devices. We introduce a simple

conversion technique which directly converts supervisor's automaton to LLD. We assume that controllable events can only appear in the form of $(n+1)$-state switches, which are then easily mapped to PLC output signals. A supervisor is described by a state machine which consists of a set of events and states. Supervisor moves from one state to the next in response to the occurrence of events, and specifies which controllable events should be *disabled* in the new state [9], [10]. PLC's operation can be seen from an input-output perspective [11] where the values of output signals are updated in response to input signals. In contrast to SCT, the PLC plays an "active" role by *generating* those controllable events that in SCT are generated by the plant and are enabled by the supervisor, and thus the notion of *disabling* events by a supervisor is replaced with *generating* enabled controllable events by the PLC controller.

We propose to partition states and events of the supervisor as PLC's input and output signals. As PLC handles signals only, we have to interpret supervisor's states and events as PLC signals. Generally, events occur instantaneously and causing transition from one state to another. We represent the plant events with the rising and falling edges of the input/output signals. By comparing signals between two ladder scan cycles [2], [3], [4], rising or falling edges can be easily detected. Our method clearly portrays how a user can define controllable and uncontrollable events as LLD signals. To test our approach, we design petrochemical boiler controllers using supervisory control theory, convert DES supervisors to LLD using our conversion technique, and finally test our design by setting a virtual plant setup using a PLC simulation software Automation Studio 5.2 [12].

1.2 Review of related works

The subject of conversion of RW supervisors to ladder logic diagrams has been well studied and already been addressed by many researchers [5], [6], [7], [8]. The implementation issue of supervisor using PLC has been first investigated by Brandin [5]. Brandin considers that supervisory control consists of three tasks and suggests how these tasks can be implemented by the PLC. These are: 1) Monitoring of the plant behavior: during every program scan, the status of the PLC inputs representing plant behaviors is evaluated. 2) Control evaluation: every state of the supervisor is

represented by an internal label. One such internal label is energized at any time and the supervisor's current state is represented by this internal label. Upon occurrence of events (where the corresponding PLC inputs are energized), these internals are latched and unlatched. Whenever supervisor reaches a state, outputs corresponding to the supervisory control map are energized. 3) Control enforcement: the enforcement of control commands can be carried out by the PLC output modules to energize the given outputs, which in turn enforces the corresponding events and thus their corresponding output devices.

Fabian and Hellgren [6] point out that physical implementation of a supervisor is not straightforward. They identify few problems that immediately arise when implementing the event-based asynchronous automata by the synchronous signal-based PLC. Few problems and the associated solutions they offered are listed below.

Events and signals. When associating events with rising and falling edges of signals, there might be a risk to introduce *avalanche effect* because of the sequential evaluation of the PLC program, which could be problematic when several states are sequentially stepped through by the occurrence of a single event. Due to this effect, PLC program might skip over a number of states during the same scan cycle. They suggests the problem may be solved by placing of rungs in an 'intelligent' order.

Causality. The SCT states that the plant generates all events, and the supervisor dynamically disables events if plant generates undesirable events. In other words, supervisor controls the plant to stay within the specifications. However, in PLC implementation the signals in the plant change in response to the signal changes generated by the PLC. Thus, the events that are not naturally generated by the plant can be generated by the PLC, and not more than one event should be generated by the PLC at the same time.

Choice. A supervisor is generally nondeterministic: in every state it offers the plant alternative events to choose from. However because of its deterministic nature PLC can only generate one event in every state—any subsequent events will be ignored as the system state may already be left as a result of the occurrence of the first event. If the choice is not explicitly made by the programmer, PLC will make the choice according to the rung order.

Inexact synchronization. When PLC executes its program it does not observe the plant. In general, program scan time is shorter than the plant response time. However, there is no guarantee that the PLC scan cycle time and plant response time are in phase; therefore, there might be a possibility that plant can change its state while the program is being executed. The changed state of the plant may invalidate the command issued by the program. This problem can be solved by the introduction of *delay insensitive* languages. The supervisor is delay insensitive if its choice is not invalidated by a plant event that can occur while the program is executed by the supervisor. The supervisor can always control the plant if it generates the command, and then wait for the plant response. Indeed, this is natural in PLC implementation.

Most of the suggestions described above are incorporated in our conversion method.

The first generalized conversion method for converting supervisors to LLDs (in the work termed as Relay Ladder Logic (RLL)) was addressed by Leduc and Wonham [8]. The method introduced by Leduc and Wonham first translates RW supervisors to equivalent Clocked Moore Synchronous State Machines (CMSSM), then expresses the boolean logic defining a CMSSM in Relay Ladder Logic (RLL), and finally RLL is run on PLCs. To facilitate the conversion method, a PLC-based testbed [13] is designed to simulate a manufacturing workcell and emphasize the problems of routing and collision. The method illustrates how CMSSM can be used to implement control functions in digital hardware. A CMSSM is represented by a 7-tuple, where all components are encoded by binary numbers implemented using boolean variables: inputs to the state machine, outputs from the state machine, state vector, next state function, output map, the initial or reset state, and period of the clock pulse that drives the state machine. The method introduces few functions to define each of the components of CMSSM. The inputs to the state machine are the DES events (Σ) and the outputs from the state machine are the DES controllable events (Σ_c). The next state vector is determined by a number of boolean functions in input variables and current state variables; outputs from the state machine are also determined using boolean functions. In both cases, there is no general formula to define next state vectors and outputs.

A method closely related to our work is carried out in [7] by Liu and Darabi. By viewing a manufacturing system as a set of activities and their required resources,

a new event classification is used to formulate the conversion problem. In order to perform a set of activities, at least one resource is required by the manufacturing system. During each activity, a *starting event* happens as soon as all required resources are obtained by the activity; conversely, when the activity is complete, an activity *ending event* occurs. When a resource is allotted to an activity, a resource-activity *latch event* occurs while a resource-activity *unlatch event* occurs when a resource is freed from an activity. Other than resource latch/unlatch or activity starting/ending events, the events that change manufacturing system's state can be classified as *system events* i.e a machine breakdown event. The method proposes an overview of the implementation process in three steps. First, by identifying the activities and resources, RW supervisor event set is partitioned into activity and system events. Next, for each activity, missing events are added to complete the *activity execution cycle* (assigning resources - starting activity - ending activity - freeing resources) resulting in an extended model of RW supervisor. In extended supervisor, activity-resource and state-resource vectors are defined, where *activity-resource vector* is used to show the utilization of resources by activities, and *state-resource vector* is used to show the resource utilization status in each state. Finally, a conversion rule is addressed to convert the extended RW supervisor model to an equivalent LLD, where resource latch and unlatch events are identified as outputs, while ending and system events are inputs.

We develop a new conversion method taking our clue from [7]. We identify a given RW supervisor event set with the corresponding PLC Input/Output (I/O) signal sets, and introduce a simpler conversion technique which directly converts supervisor's automaton to LLD.

1.3 Implementation procedures and tools

There are different discrete-event controller construction techniques available in academia based on formalisms such as Petri nets and finite automata [7]. We choose finite automata-based controller construction technique, introduced by Ramadge and Wonham (RW) [1], to design our controllers. The constructed controllers are mathematically guaranteed to be controllable and maximally permissive for a given DES plant and specification. Before the synthesis of a PLC-based solution in the DES

framework, we first need to identify the control problem wherein a controller is to be designed so that the controlled system satisfies some desired specifications.

Discrete-event systems cover a wide variety of physical systems such as manufacturing systems, traffic systems, communication protocols, logistic management systems and data communication networks. We design boiler control systems which are used in petrochemical industries to demonstrate our PLC-based implementation. Once plants and legal specifications are specified, one can use XPTCT software (we use version 114 for Windows 95/98/XP) [14] to design DES supervisors. The DES supervisor is then converted into LLD using our conversion technique. Finally the translated LLD is uploaded and simulated using PLC simulation software Automation Studio [12]. Necessary adjustments in the design procedures are carried out until the simulation results meet the specification requirements. If the simulation results satisfy user's requirements, PLC-based solution of the control problem can be developed using required hardware. The step-by-step guide for implementing DES supervisor using PLC is shown in Figure 1.1.

1.4 Outline of the thesis

The rest of this thesis is organized as follows. An overview of supervisory control theory of discrete-event systems and programmable logic controllers is presented in Chapter 2. After elaborately discussing a motivating example, and making some key observations and assumptions, the conversion technique is proposed in Chapter 3. Chapter 4 studies the real-field application of boiler control systems in DES framework. It also includes details of the boiler control plants, discussions of modeling I/O devices as DES, specifications of boiler controllers, construction of the boiler control supervisors using XPTCT software and implementation of the boiler controllers using PLC. Finally, Chapter 5 concludes the thesis and points out directions for future research.

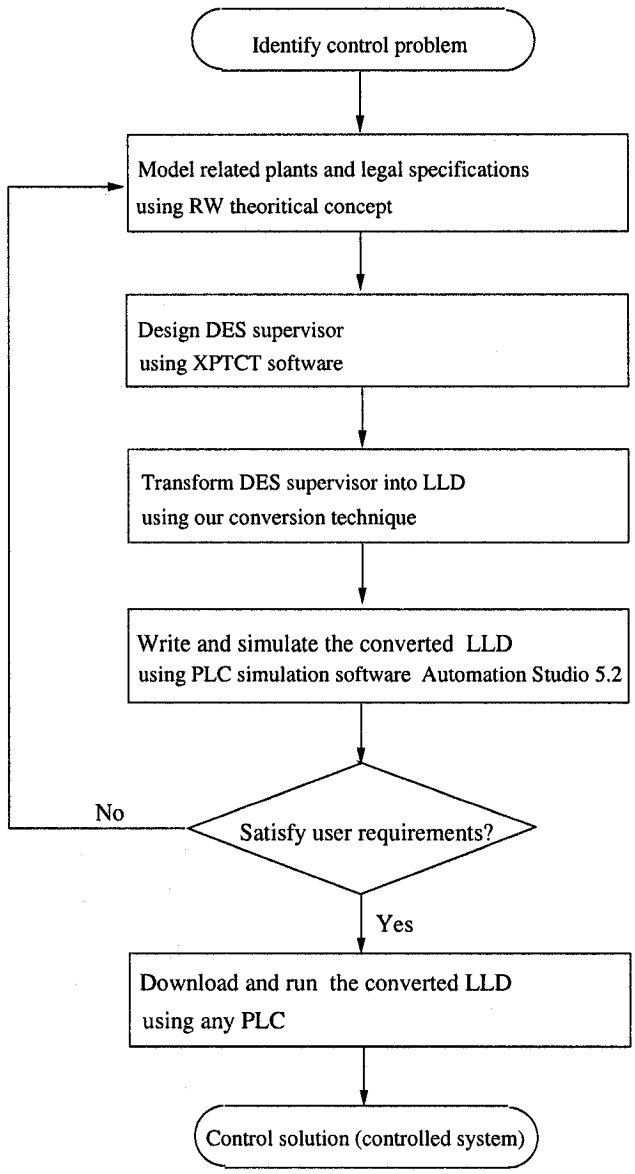


Figure 1.1: Implementation sequence.

Chapter 2

Preliminaries

2.1 Supervisory control of DES

2.1.1 DES plant, specification and supervisory control

The supervisory control theory provides an optimal solution to control a wide variety of physical systems that are discrete in time and state space, asynchronous, and nondeterministic. In Ramadge and Wonham (RW) theory [1], a plant can be thought of as the generator of a formal language and can be modeled as an automaton, which can be graphically represented by a transition graph. Let $G = (Q, \Sigma, \delta, q_0, Q_m)$ be an automaton representing a DES plant where Q is the set of states, Σ is the events set, $\delta : Q \times \Sigma \rightarrow Q$ is the partial state transition function, $q_0 \in Q$ is the initial state and $Q_m \subseteq Q$ is the subset of marker states commonly used to "mark" some states where a certain task is complete. The disjoint subsets Σ_c and Σ_u, where $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$, consist of *controllable* and *uncontrollable* events, respectively. A *language* L over Σ is any subset of Σ^*, where Σ^* denotes the set of all finite strings over Σ, including the empty string ϵ. The behaviors of G are distinguished as closed and marked. The closed behavior of DES G is the set of all possible event sequences which the plant may generate and is formally defined as

$$L(G) = \{s \in \Sigma^* \mid \delta(q_0, s)!\}$$

The marked behavior of DES G is comprised of those sequences in the closed behavior which reach some marker states and is formally defined as

$$L_m(G) = \{s \in \Sigma^* \mid \delta(q_0, s) \in Q_m\}$$

The DES G is said to be *nonblocking* if every reachable state is coreachable (reachable and coreachable states are defined below). When G is nonblocking we have:

$$\overline{L_m(G)} = L(G)$$

A language $K \subseteq \Sigma^*$ is *controllable* with respect to G iff

$$\bar{K}\Sigma_u \cap L(G) \subseteq \bar{K}$$

When K is controllable the prefix closure $\bar{K}$ is invariant under the occurrence of uncontrollable events in $L(G)$. By specifying a subset of Σ_c, a particular subset of events to be enabled can be selected. Since uncontrollable events are always enabled, all Σ_u events are adjoined to this set. Each such subset of events is called a *control pattern*, and the set of all control patterns is denoted by $\Gamma = \{\gamma \subseteq \Sigma | \gamma \supseteq \Sigma_u\}$. A *supervisory control* for G is any map $V : L(G) \rightarrow \Gamma$. The pair (G, V) will be denoted as V/G, to suggest 'G under the supervision of V'. The closed behavior of V/G is defined to be the language $L(V/G) \subseteq L(G)$ defined below:

i. $\epsilon \in L(V/G)$,

ii. If $s \in L(V/G) \wedge \sigma \in V(s) \wedge s\sigma \in L(G) \Rightarrow s\sigma \in L(V/G)$,

iii. No other strings belong to $L(V/G)$.

The map V is *nonblocking* for G if

$$\overline{L_m(V/G)} = L(V/G)$$

If $K \subseteq L_m(G)$ is controllable with respect to G and $L_m(G)$-closed, then there exists a non-blocking supervisory control map V for G such that $L(V/G) = K$ [1].

Let S be any automaton such that $L(V/G) = L(S) \cap L(G)$. Then S is said to *implement* V. If $L(S)$ is controllable with respect to G and $\overline{L_m(S) \cap L_m(G)} = L(S) \cap L(G)$, we say S is a *supervisor* for G.

Let $E \subseteq \Sigma^*$ be the specification of some desired behaviors. Then the supervisory control theory articulates how a supervisor S can be designed to control the plant so that the plant under supervision can enjoy maximum freedom while behaving within the specification E. When E is not controllable with respect to G it is shown that a supremal controllable and $L_m(G)$-closed sublanguage of E, denoted by $\hat{K}$ exists. It follows from the main theorem in supervisory control theory [1] that a minimally

restrictive (thus optimal) nonblocking supervisory control $\hat{V}$ can be designed such that $L_m(\hat{V}/G) = \hat{K}$.

Consider two automata $G_1 = (Q_1, \Sigma_1, \delta_1, q_{10}, Q_{1m})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_{20}, Q_{2m})$. The synchronous product of G_1 and G_2 is the *reachable* (denoted by 'Rch'; defined below) part of the automaton

$$G_1 \parallel G_2 := Rch(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{10}, q_{20}), Q_{1m} \times Q_{2m})$$

where for $(q_1, q_2) \in Q_1 \times Q_2$ and $\sigma \in \Sigma$,

$$\delta((q_1, q_2), \sigma) = \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) & \text{if } \delta_1(q_1, \sigma)! \text{ and } \delta_2(q_2, \sigma)! \\ (\delta_1(q_1, \sigma), q_2) & \text{if } \delta_1(q_1, \sigma)! \text{ and } \sigma \notin \Sigma_2 \\ (q_1, \delta_2(q_2, \sigma)) & \text{if } \delta_2(q_2, \sigma)! \text{ and } \sigma \notin \Sigma_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

2.1.2 Operations on DES

To design a supervisor using RW supervisory control theory, the following operations on languages are carried out.

Trim

DES G is *trim* if it is both *reachable* and *coreachable*, that is, all states can be reached from the initial state, and some marker states can be reached from every state. The set of reachable states is formally defined as

$$Q_r = \{q \in Q \mid (\exists s \in \Sigma^*)\delta(q_0, s) = q\};$$

G is reachable if $Q_r = Q$. The set of coreachable states is formally defined as

$$Q_{cr} = \{q \in Q \mid (\exists s \in \Sigma^*)\delta(q, s) \in Q_m\};$$

G is coreachable if $Q_{cr} = Q$.

Synchronous product

Synchronous product specifies how several plant components can function jointly, and is denoted by $\parallel$. If $L_1 = L_m(G_1)$ and $L_2 = L_m(G_2)$, then G_1 and G_2 can generate

$L_1 \parallel L_2$ only when common events are eligible to occur synchronously in both plants, while individual plant events are allowed to occur whenever they are eligible in their corresponding components. Let $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$, where $\Sigma = \Sigma_1 \cup \Sigma_2$. Define *natural projection*

$$P_i : \Sigma^* \to \Sigma_i^* \qquad (i = 1, 2)$$

according to

$$P_i(\epsilon) = \epsilon$$

$$P_i(\sigma) = \begin{cases} \epsilon & \text{if} \quad \sigma \notin \Sigma_i \\ \sigma & \text{if} \quad \sigma \in \Sigma_i \end{cases}$$

$$P_i(s\sigma) = P_i(s)P_i(\sigma) \qquad s \in \Sigma^*, \sigma \in \Sigma$$

The action of natural projection P_i on a string s is to eliminate all occurrences of σ in s such that $\sigma \notin \Sigma_i$. The synchronous product $L_1 \parallel L_2 \subseteq \Sigma^*$ of two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ is defined according to

$$L_1 \parallel L_2 = P_1^{-1}L_1 \cap P_2^{-1}L_2$$

That is, $s \in L_1 \parallel L_2$ iff $P_1(s) \in L_1$ and $P_2(s) \in L_2$. Thus synchronous product of two DES G_1 and G_2, denoted by $G = G_1 \parallel G_2$, was previously defined. We have:

$$L_m(G) = L_m(G_1) \parallel L_m(G_2), \qquad L(G) = L(G_1) \parallel L(G_2)$$

Meet

Meet is the special case of synchronous product corresponding to $\Sigma_1 = \Sigma_2$; specifically all events are considered shared and synchronization is total. The meet operation blocks any transition which cannot occur in both G_1 and G_2. Consider two automata $G_1 = (Q_1, \Sigma_1, \delta_1, q_{10}, Q_{1m})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_{20}, Q_{2m})$. The meet of G_1 and G_2 is the reachable part of the automaton

$$G_1 \wedge G_2 := Rch(Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, (q_{10}, q_{20}), Q_{1m} \times Q_{2m})$$

where for $(q_1, q_2) \in Q_1 \times Q_2$ and $\sigma \in \Sigma$,

$$\delta((q_1, q_2), \sigma) = \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) & \text{if } \delta_1(q_1, \sigma)! \text{ and } \delta_2(q_2, \sigma)! \\ \text{undefined} & \text{otherwise} \end{cases}$$

Given two DES G_1 and G_2, the meet operation produces a reachable DES $G = G_1 \wedge G_2$ such that

$$L_m(G) = L_m(G_1) \cap L_m(G_2), \qquad L(G) = L(G_1) \cap L(G_2)$$

2.1.3 DES Supervisor

A *supervisor* is an external agent responsible for controlling the system. It monitors and controls the behavior of the system (a set of plants) so that it complies with some specifications; if supervisor observes any possible exit to the illegal behavior, it will restrict the plant's behavior by disabling a subset of controllable events to achieve a maximally permissive control solution. A supervisor can be represented by an automaton:

$$S = (X, \Sigma, \xi, x_0, X_m)$$

The control action of S on G can be visualized as follows: upon arriving at a state $x \in X$, an event $\sigma \in \Sigma$ is enabled if $\xi(x, \sigma)!$; otherwise it is disabled. Note that the uncontrollable events cannot be controlled and thus their occurrence should not be disabled by the supervisor. The closed-loop behavior of the plant G under the supervision of S, denoted by S/G (also called the supervised system), can be formalized by the meet of S and G.

Consider a supervisor S for G which can be represented by an automaton $S = (X, \Sigma, \xi, x_0, X_m)$. The closed-loop behavior of the plant G under the supervision of S, denoted by S/G can be formalized by the meet of S and G, that is $S/G = G \wedge S$, and therefore:

$$L(S/G) = L(S) \cap L(G)$$

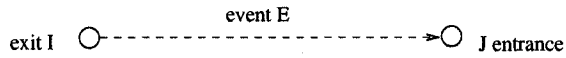
$$L_m(S/G) = L_m(S) \cap L_m(G)$$

2.1.4 TCT: DES supervisor design software

TCT[1] is a design software developed by W. M. Wonham's research team at University of Toronto [14]. TCT is used for the synthesis of supervisory control for untimed discrete-event systems. The descriptions thereafter about TCT procedures are reproduced from [9]. An automaton is represented by a 5-tuple [Size, Init, Mark, Voc,

[1]Where XPTCT refers to Windows version of the DES supervisor design software TCT.

Tran], where Size is the number of states and the state set is $\{0, \cdots\cdots, \text{Size} - 1\}$, Init is the initial state (always taken to be 0), Mark lists the marker states, Voc is a list of vocal states, and Tran is the transition table. The automata models in this thesis do not have output and thus Voc is always taken to be empty. A transition is a triple $[I, E, J]$ representing a transition from the exit state I to the entrance state J and having event label E. E is an odd or even nonnegative integer, depending on whether the corresponding event is controllable or uncontrollable, respectively.



In this work we use deterministic automata to model DES: distinct transitions from the same exit state must carry distinct labels.

The following is a list of procedures we use in system design. A complete list can be found in [9].

`create` prompts the user to define a new discrete-event system.

`selfloop` augments an existing DES by adjoining selfloops at each state with event labels in a list provided by the user.

`trim` applied to DES1 constructs the trim (reachable and coreachable) automaton DES2.

`sync` forms the reachable synchronous product of DES1 and DES2 to create DES3.

`meet` forms the meet (reachable cartesian product) of DES1 and DES2 to create DES3. Note that DES3 need not be coreachable.

`supcon` for a controlled generator DES1 and specification DES2 forms a trim recognizer for the supremal controllable sublanguage of the marked language generated by DES2 to create DES3. The resulting DES3 is a proper supervisor for DES1.

`minstate` reduces DES1 to a minimal state transition structure DES2 that generates the same closed and marked languages. DES2 is reachable but not necessarily conreachable.

`isomorph` tests whether DES1 and DES2 are identical up to renumbering of states; if so, their state correspondence is displayed.

The following is a list of utilities we use in system design:

`edit` allows the user to modify an existing DES.

`show` SE displays an existing DES, SA a data (condat) table, SX a TXT (text) file. Tables can be browsed with page keys. MAKEIT.TXT keeps a record of user files as they are generated.

`file` FE (resp. FA) converts a DES (resp. DAT) file to an ASCII PDS text file (resp. PDT) or Postscript PSS file (resp. PST) for printing. Some printers may only recognize a Postscript file with suffix .PS; in that case, rename the .PSS/.PST files with due care to avoid duplication.

`user file directory` lists the current user subdirectory.

2.1.5 Design of DES supervisor using TCT

Let the controlled DES G representing the plant be given together with an upper bound $E \subseteq \Sigma^*$ on admissible marked behavior. Assume that $EDES$ is the specification automaton such that $E = L_m(EDES)$. Our objective is to design an optimal supervisor $KDES$ for G. To compute a trim recognizer for the language $K := supC(E \cap L_m(G))$, the TCT procedure `supcon` computes a trim representation $KDES$ of K according to

$$KDES = \texttt{supcon}(G, EDES)$$

The set of controllable events that must be disabled by $KDES$ can be found using TCT procedure `condat`. Specifically, the `condat` operation returns the control pattern at each state of $KDES$ according to

$$KDAT = \texttt{condat}(G, KDES)$$

If there are n plant components and m specifications, G is the `sync` of all plant components while $EDES$ is the `meet` of all specifications. The supervisor is then computed following the above procedures.

2.2 Programmable Logic Controller (PLC)

A Programmable Logic Controller (PLC) is a microprocessor-based specialized computer that performs many types of complex logic-based control functions [2], [3], [4]. The main purpose of PLC is to continuously monitor process control parameters and manipulate plant operations by means of stored program set by the user. A PLC is designed to operate in a wide range of industrial environments such as high heat, humidity and vibrations. Their flexibility, reliability, maintainability, affordability and user friendliness make them immensely popular, and thus widely used in many industrial applications. The overall system layout of PLC and interconnection of each part are shown in Figure 2.1.

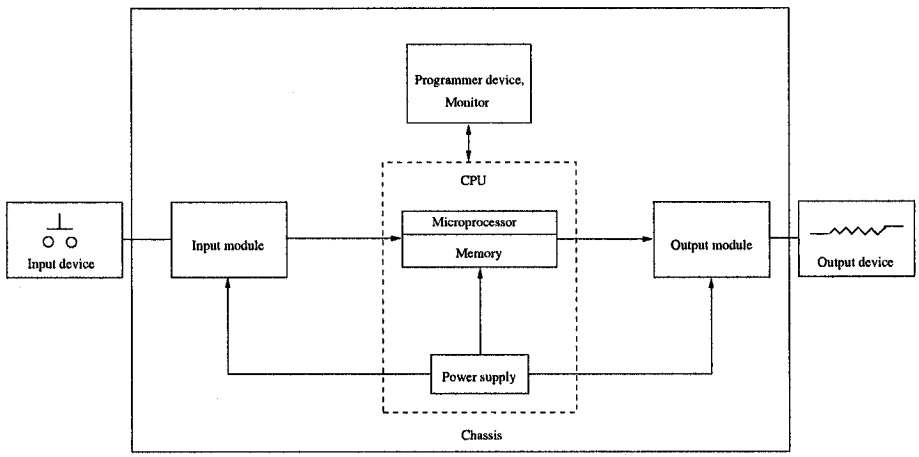


Figure 2.1: Layout of PLC.

The input module has terminals where plant electrical signals responded by input field devices are received. There are various types of input devices available to be connected to the input module which are mainly on/off switches. The common types of switches are toggle-type switches and pushbutton switches. When a pushbutton switch is pressed, the corresponding input contact becomes TRUE momentarily, while when the toggle-switch is pressed, the corresponding input contact becomes TRUE and remains TRUE until the switch is depressed. Other types of switches have either Normally[2] Open (NO) or Normally Close (NC) configuration. The most commonly used device in this category is the limit switch. When the limit switch is actuated, its electrical contact changes from NO to NC (action can be taken as on/off), or NC

[2]Normally means device is in unactuated state.

to NO (action can be taken as off/on). Some other types of input on/off devices are pressure switch, level switch, float switch, magnetic-sensitive switch and inductive-sensitive switch. Other types of input devices, such as potentiometer and transducer, can be used to produce a varying input electrical signal to PLC input module.

Similarly, the output module has terminals from where output signals are sent to activate output field devices. The most common output device is electrical solenoid which is used for on/off control of flow valves such as gas and liquid control valve. Other on/off output devices are relays, motors and various solid-state switching devices. A special type of output devices which can be controlled by PLC are analog output devices such as stepper motors, servomotors and servo valves.

The logic operations are carried out by microprocessor while data, user program and system software are stored and retrieved from memory.

PLC programs can be written by LLDs (Ladder Logic Diagrams) or Sequential Function Charts (SFCs) or other PLC programming languages [2], [3], [4]. A PLC program, often consisting of a set of boolean statements and linking inputs to outputs via their corresponding internal table, is evaluated sequentially from top to bottom and from left to right.

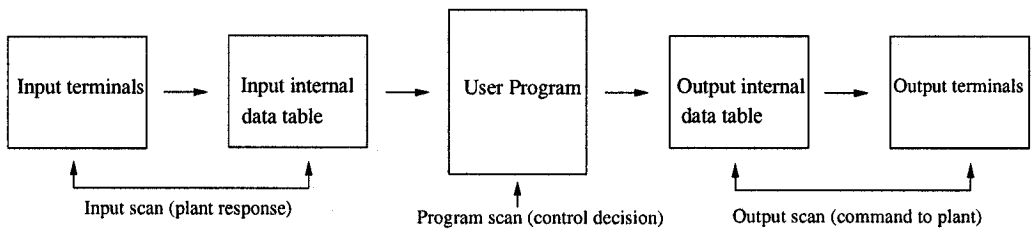


Figure 2.2: PLC scan cycle.

Each PLC operational cycle, called *scan cycle*, is made up of three parts, as shown in Figure 2.2: 1) Reading inputs (whether an input device is energized or de-energized) and updating the input status table. 2) Executing the control logic set by the user program, and updating the output status table. 3) Finally transferring the data associated with the output status table to the outputs (energize/de-energize output devices). Consequently, a PLC's operation can be seen from an input-output perspective where output signals update their states in response to the input signals, as shown in Figure 2.3 .

In the PLC, the input scan, program scan and output scan are separate functions.

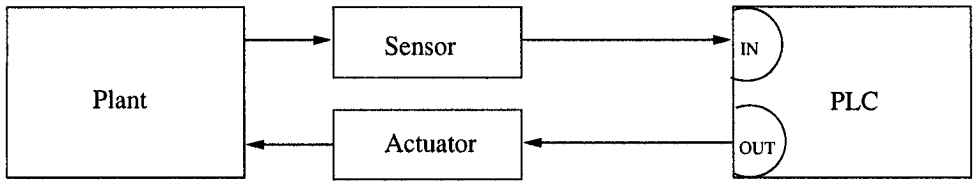


Figure 2.3: A simple PLC I/O view.

Thus, any changes in the input device status during the program scan or output scan are not accepted by the PLC until the next input scan. Moreover, during the input and program scans, data changes in the output table are not forwarded to the output terminals. The new data which may change the states of the output devices is forwarded only during the output scan. The total response delay for one complete scan cycle, termed as scan time, is a function of the processor speed and length of the user program; typically, scan time may vary from 1 ms to 100 ms [5].

One of the main assumptions about DES is that events occur asynchronously. Due to scan time and the sequential operation of the PLC, the occurrence of events must be interpreted synchronously (one event before the other with at least some time gap) by the PLC. For the purpose of implementation of DES supervisor using PLC, the deviation of implementing the event-based asynchronous automata by the synchronous signal-based PLC could cause timing problems (the time gap between the occurrence of one event to another in PLC implementation, and the time gap between the PLC and plant response since PLCs are faster than plants), and we may refer to this timing problem as *transmission delay* or *response delay.* It is quite natural in any PLC implementation that once output data (command) is transferred to output devices by the PLC, PLC is needed to wait for plant response (input) to re-evaluate its output data. This necessity is because nominally PLCs are faster than plants, and therefore, plants response delay can be ignored.

2.2.1 Ladder Logic Diagrams (LLDs)

A ladder logic is made up of many rungs, each consisting of logical checkers (input contacts) which become true or false in response to plant outputs communicated via sensors, and output coils which drive some actuators or hold an internal storage bit that can be used in other parts of the ladder. If the logic statement is true, the contact

can "make" the circuit to energize output coils. If the logic statement is false, the contact can "break" the circuit to de-energize output coils. There are many types of contact and coil functions available for PLC programming. See [2], [3], [4], [5], [6] and [15] for details. In our work, we use normally open contacts and both latch and unlatch functions for output coils, as shown in Figure 2.4.

Component	Function
Y  Normally open contact	Examine if boolean variable Y is TRUE.
Y  Normally close contact	Examine if boolean variable Y is FALSE.
X  Latch	If logic statement on the left of the rung is TRUE, boolean variable X becomes TRUE and remains TRUE until it is unlatched.
X  Unlatch	If logic statement on the left of the rung is TRUE, boolean variable X becomes FALSE and remains FALSE until it is latched.

Figure 2.4: Some LLD components.

A normally open (NO) contact is used to examine if the boolean variable Y is TRUE. If the left state (TRUE/FALSE) of the contact of the boolean variable Y is TRUE, NO contact can "make" the circuit to energize output coils; otherwise, the right state of the contact is FALSE. Conversely, a normally close (NC) contact is used to examine if the boolean variable Y is FALSE. If the left state (TRUE/FALSE) of the contact of the boolean variable Y is TRUE, NC contact can "break" the circuit to de-energize output coils; otherwise, the right state of the contact is TRUE. After the evaluation of a logic statement, output of a rung is assigned by various kinds of coils such as latch and unlatch coil. These coils can be connected to physical output devices to be operated, or can be used to hold internal storage bits that can be referred to in other parts of the program for logic statements. When the logic statement on the left of the rung is TRUE, boolean variable X of the latch coil becomes TRUE (the associated output device is energized) and remains TRUE until it is unlatched. On the other hand, if the logic statement on the left of the rung is TRUE, boolean variable X of the unlatch coil becomes FALSE (the associated output device is de-energized) and remains FALSE until it is latched.

Other than contacts and coils, the most commonly used control device in the

PLC is the timer. Although industrial and digital electronic timers are available to use with PLC, all these timers can be replaced by using various kinds of PLC timer functions such as TIMER ON-DELAY and TIMER OFF-DELAY. An example of TIMER ON-DELAY function in the LLD, which we used in our implementations, is explained in Figure 2.5. TIMER-OFF DELAY is also included in the same figure for completeness.

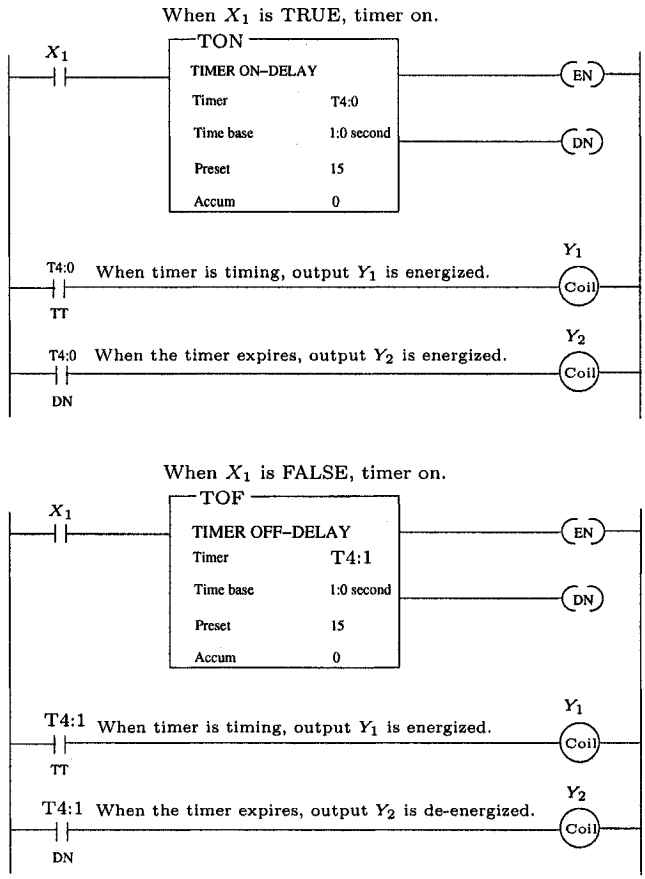


Figure 2.5: TIMER ON-DELAY and TIMER OFF-DELAY in LLD.

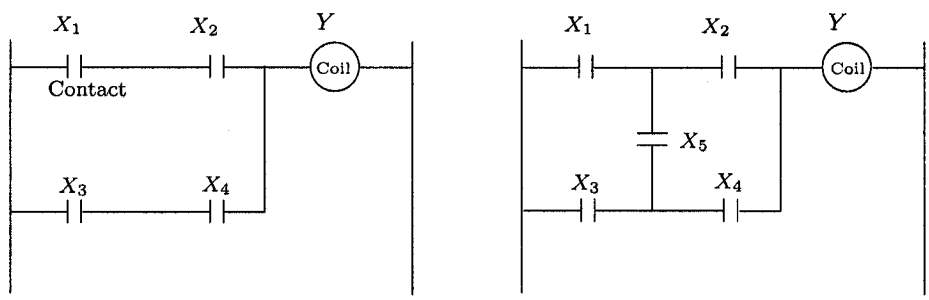
TIMER ON-DELAY works as follows: when X_1 is TRUE, the PLC starts the timer (addressed as T4:0) and the accumulated value of the timer is incremented in 1 second intervals. While the timer is accumulating time, T4:0 TT (timer internal contact) becomes TRUE as well as the output coil Y_1 is energized. When the timer expires, T4:0 TT becomes FALSE as well as the output coil Y_1 is de-energized and T4:0 DN becomes TRUE by which the output coil Y_2 is energized. If the accumulated value reaches 15 seconds (user can preset this value as desired) or X_1 becomes FALSE, the timer will be reset.

TIMER OFF-DELAY works as follows: when X_1 is FALSE, the PLC starts the timer (T4:1). The accumulated value of the timer is incremented in 1 second intervals. While the timer is accumulating time, T4:1 TT becomes TRUE as well as the output coil Y_1 is energized. When the timer expires, T4:1 TT becomes FALSE as well as the output coil Y_1 is de-energized and T4:1 DN becomes FALSE by which the output coil Y_2 is de-energized. If the accumulated value reaches 15 seconds (user can preset this value as desired) or the X_1 becomes TRUE, the timer will be stopped. When X_1 becomes FALSE again, the timer is reset and restarted.

When PLC is programmed using LLDs, the following ladder construction limitations must be taken into account. If incorrectly formatted LLDs are uploaded in the PLC, it will not accept the LLDs and report an error message. These limitations may vary depending on individual PLC manufacturers. It is recommended to consult with the individual operation manuals for proper programming information of a given PLC system. For proper construction of LLDs, we generally follow the guidelines mentioned in [2], [3] for LLD-based programming of Allen-Bradley and Siemens PLCs using PLC simulation software Automation Studio [12]. Since Automation Studio [12] supports only ladder libraries for Allen-Bradley and Siemens PLCs, we use both types of PLCs in our work. The common instructions are given below. The essence of the figures for the LLD instructions is taken from [3].

Contact placement.

1. A contact must always be placed in the upper left of the rung, as shown in all of the following figures.



a) All contacts are placed horizontally (correct).

b) X_5 is placed vertically (incorrect).

Figure 2.6: Contact placement in LLD.

2. All contacts must be placed horizontally; contacts are not allowed to be placed vertically. In Figure 2.6 (part b), contact X_5 is placed incorrectly, whereas in the same figure (part a) all contact are positioned correctly.

3. The number of contacts per matrix (per logic statement) must be within 11 across by 7 down, as shown in Figure 2.7 (part c).

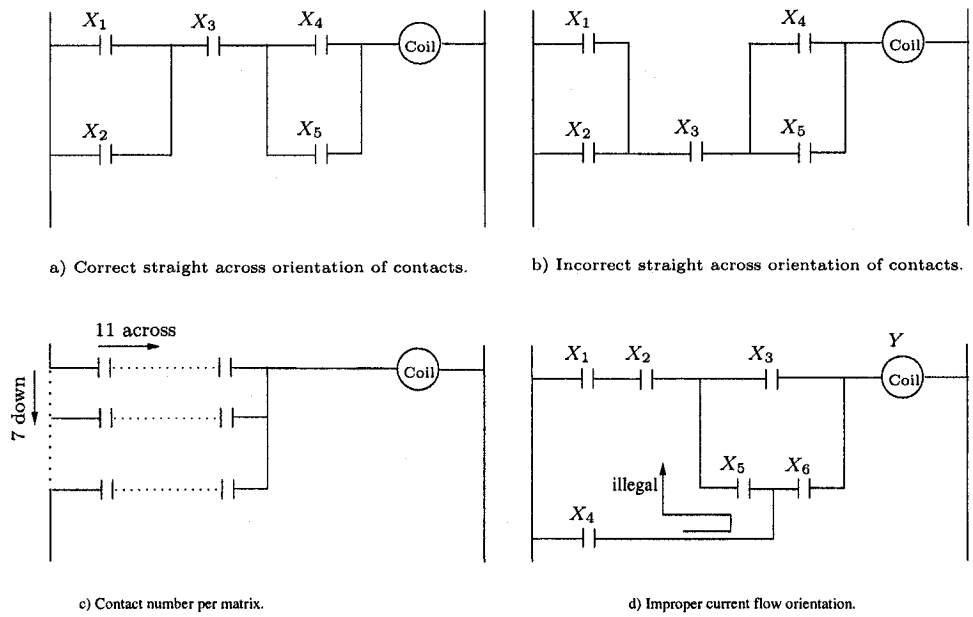
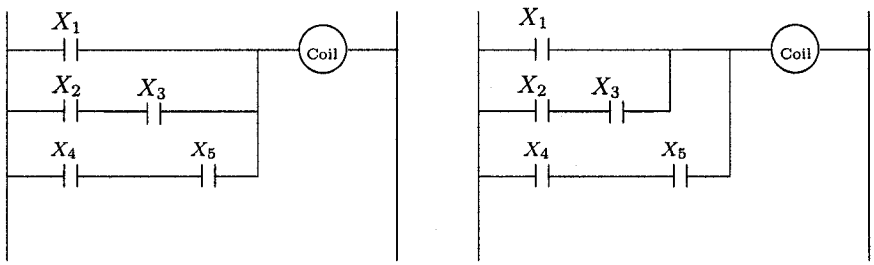


Figure 2.7: Contact placement in LLD.

4. Contacts must be nested (a branch circuit is placed within a branch circuit) properly, as described in Figure 2.8 (part a and b).

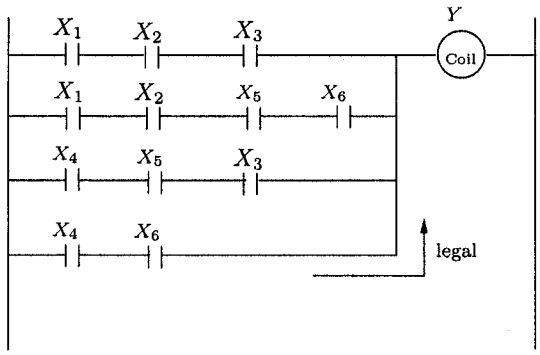
5. Contacts must be placed in straight across, as described in Figure 2.7 (part a and b).

6. Contacts must be inserted in the rung such that the current conduction through contacts is considered to flow from left to right. This is shown in Figure 2.7 (part d), Figure 2.8 (part c), and described as follows. When (refer Figure 2.7 (part d)) output for Y is considered as $Y = X_1 \cdot X_2 \cdot X_3 + X_1 \cdot X_2 \cdot X_5 \cdot X_6 + X_4 \cdot X_5 \cdot X_3 + X_4 \cdot X_6$, in this case current can flow from right to left through contact number 5 via 4 and 3; therefore this type of contacts placement is not legal. While, in Figure 2.8 (part c) the proper current flow for output Y is considered as $Y = X_1 \cdot X_2 \cdot X_3 + X_1 \cdot X_2 \cdot X_5 \cdot X_6 + X_4 \cdot X_5 \cdot X_3 + X_4 \cdot X_6$, in this case current can always flow from left to right; therefore this type of contacts placement is legal.



a) Correct nested program orientation.

b) Incorrect nested program orientation.



c) Proper current flow orientation.

Figure 2.8: Contact placement in LLD.

Coil placement.

1. A coil must always be placed at the end of a rung, as is the case in all of the figures above.
2. One output coil can be assigned to a group of contacts.
3. Coil can be used either to drive an output device or hold an internal storage bit that can be referred to in other parts of the rung for logic statements.

2.2.2 Automation Studio: A PLC simulation software

Automation Studio [12] is an ISO standard integrated software that can be used to design, simulate, document and animate circuits comprising of various automation technologies such as PLCs, pneumatics, hydraulics, sequential function charts

and electrical controls. Automation Studio allows users to design circuits or simulation platforms using its various libraries and modules such as PLC ladder logic and electrical controls. The PLC (Allen-Bradley and Siemens) ladder logic library includes all ladder logic functions such as contacts, coils and timers. The electrical controls library which comes complete with switches, relays, solenoids, push buttons and power supplies may be required to construct an experimental platform of any complex systems. The libraries of Automation Studio are listed in various groups of component category. The components' list can be searched easily, then the required component can be selected, and dragged and dropped onto the schematics or the program. It is allowed to name components following a suggested guideline. Users can simulate, experiment and implement possibly any virtual plants or control systems using Automation Studio ladder logic library and other appropriate libraries. During simulation, plant components become animated and electrical lines are color-coded according to their states from which the user can observe the simulation results. Users can further manipulate the simulation pace using available functions such as normal/slow motion and step-by-step. By using I/O interface hardware (with a limited number of I/O), users can directly connect Automation Studio to a real PLC I/O or to real equipments such as relays, valves and sensors. Automation Studio can then be used as a replacement of PLC to control the targeted equipments. More software details and sample videos can be found in [12].

Chapter 3

Conversion Algorithm

3.1 Event generation and supervisory control

In supervisory control of DES a supervisor plays a "passive" role in that it observes discrete evolution of the system and at any point it can disable a subset of controllable events. Thus plant plays the role of generating controllable and uncontrollable events alike, and controllable events can only be disabled by the supervisor. In contrast, in our proposed PLC implementation of supervisory control map represented by the supervisor's automaton, the PLC plays an "active" role by *generating* those controllable events that in SCT are generated by the plant and enabled by the supervisor, and that the notion of *disabling* events by a supervisor is replaced with *generating* enabled events by the PLC controller. This is similar to the idea addressed by Balemi [11] to indicate the fact that for most real systems plant events are not generated spontaneously, but only as the responses to given commands generated by the supervisor. In our method, this idea of identifying the sources of events (which events are generated by the plant, and which events are generated by the supervisor) is implemented.

We observe that this assumption is not restrictive at all: it is quite logical to envisage that a motor cannot be turned on arbitrarily, and something or someone (which is PLC or human operator) must generate such a relevant event according to control specifications to activate/de-activate a motor.

3.2 Event interpretation: DES to LLD

The SCT describes that plant events can have symbolic values and occur asynchronously at quasi-random time instants, while PLC deals with Boolean signals and updates their values synchronously. This deviation could cause timing problems. As considered in [16], it is quite realistic for two events to occur between consecutive clock cycles and for implementation purposes, supervisor must interpret these events as simultaneous. Generally, events occur instantaneously and cause transition from one state to another. We represent the plant events with the rising and falling edges of the input/output signals supplied by their corresponding devices. By comparing signals between two ladder scan cycles [2], [3], [4], rising or falling edges can be easily detected.

As usual we write $\Sigma = \Sigma_{c,e} \dot{\cup} \Sigma_u$. Further we write $\Sigma_u = \Sigma_{u,i} \dot{\cup} \Sigma_{u,p}$, where $\Sigma_{u,i}$ is the set of *input events* supplied by the operator, $\Sigma_{u,p}$ is the set of *plant response events* generated by the plant, and $\Sigma_{c,e}$ is the set of *external events* representing DES controllable event (Σ_c) generated by the supervisor. In the boiler control system example of chapter 4, we encounter two types of external controllable events.

1. Some controllable events have a binary status; for example, corresponding to a binary switch α we identify two events: the event of turning the switch on (α_{on}) and the event of turning the switch off (α_{off}). We implement both events by a single boolean signal $x_{\alpha_{on}}$: α_{on} occurs on the rising edge of $x_{\alpha_{on}}$ while α_{off} occurs on the falling edge of $x_{\alpha_{on}}$. This is shown in Figure 3.1.

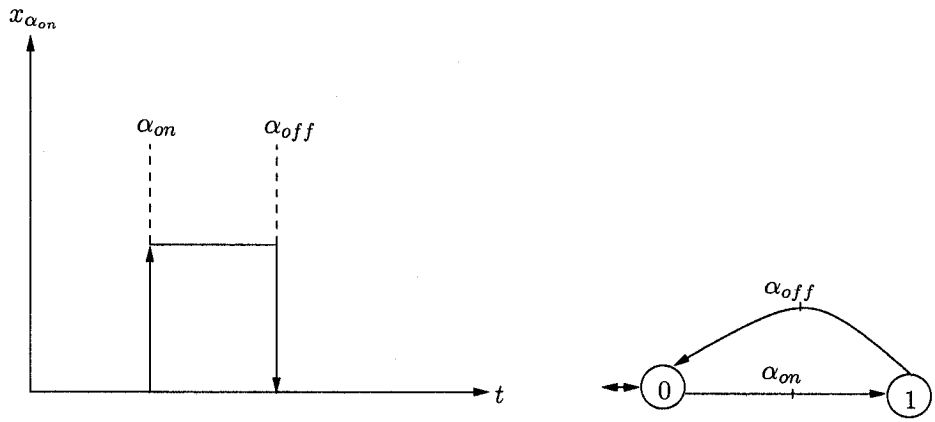


Figure 3.1: Representation of α_{on} and α_{off} by $x_{\alpha_{on}}$.

2. The other type of controllable events that we encounter have a ternary status; for example, a fuel valve β can be either set to high, low or closed, corresponding to controllable events β_{hi}, β_{lo} and β_{cl}, respectively. We implement these events by two boolean signals $x_{\beta_{lo}}$ and $x_{\beta_{hi}}$: β_{lo} and β_{hi} occur on the rising edge of $x_{\beta_{lo}}$ and $x_{\beta_{hi}}$, respectively, while β_{cl} occurs on the falling edge of $x_{\beta_{hi}} \vee x_{\beta_{lo}}$. This is shown in Figure 3.2.

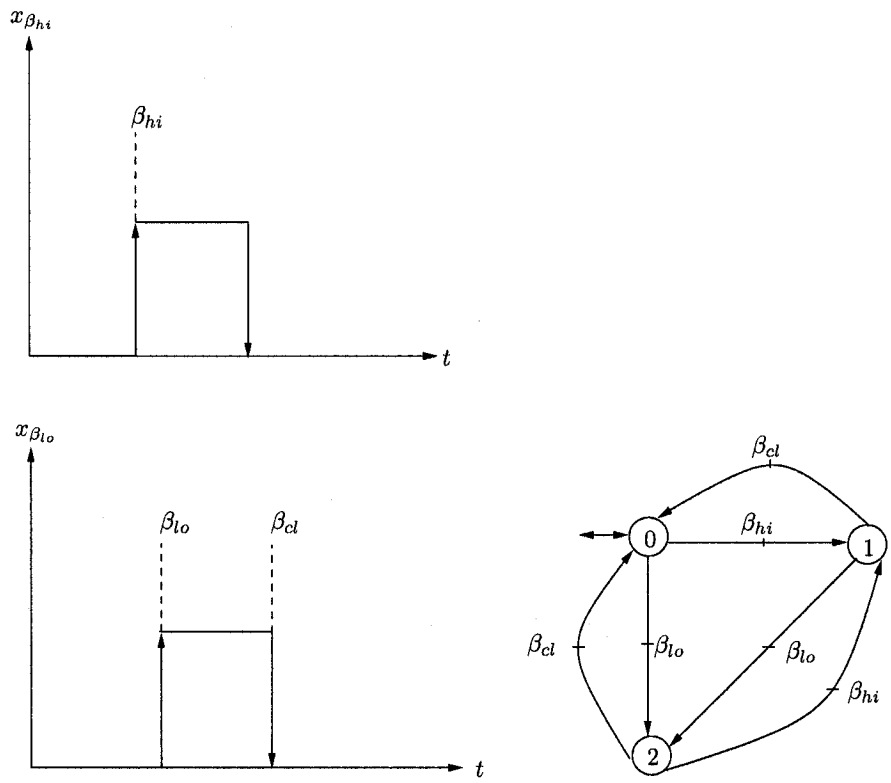


Figure 3.2: Representation of β_{hi}, β_{lo} and β_{cl} by $x_{\beta_{hi}}$ and $x_{\beta_{lo}}$.

The occurrence of an event γ in the set $\Sigma_{u,i} \cup \Sigma_{u,p}$ determines the set of actions to be followed: for example, if the temperature drops below a certain threshold (an uncontrollable event) the heater must be turned on. Thus, for such an event we introduce a boolean signal x_γ which needs to be 1 only for the duration of a scan cycle, and should drop back to 0 before the start of the next scan cycle to prevent what is called the *avalanche effect* in [6].

3.3 Motivating example

The following example takes the above considerations into account to implement a supervisor.

Example: Consider the supervisor of Figure 3.3 which we would like to implement using PLC. We have $\Sigma_{c,e} = \{c_{hi}, c_{lo}, c_{cl}, d_{on}, d_{off}\}$, $\Sigma_{u,i} = \{a\}$ and $\Sigma_{u,p} = \{b, e, f\}$.

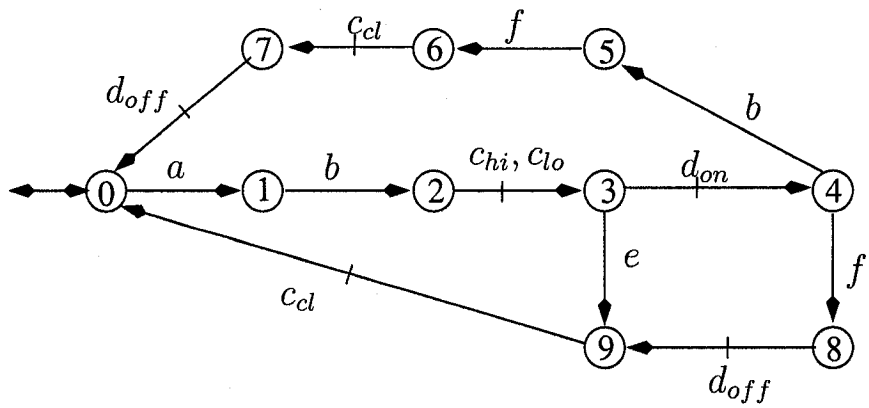


Figure 3.3: An example of a supervisor represented by an automaton.

Corresponding to external (controllable) events $\Sigma_{c,e}$ we define

$$X = \{x_{d_{on}}, x_{c_{hi}}, x_{c_{lo}}\}$$

The interpretation is that when $x_{d_{on}}$ rises from 0 to 1 d_{on} occurs, while when $x_{d_{on}}$ falls from 1 to 0 d_{off} occurs. Similarly, when $x_{c_{hi}}$ ($x_{c_{lo}}$) rises from 0 to 1 c_{hi} (c_{lo}) occurs, while when $x_{c_{hi}} \vee x_{c_{lo}}$ falls from 1 to 0 c_{cl} occurs.

For input and plant response events we define

$$Y = \{x^0, x_a, x_b, x_e, x_f\}$$

where x^0 is set by the operator to 1 to start the PLC controller.

Table 3.1 lists the rungs in LLD as a function

$$\Omega : (Q \cup \{\emptyset\}) \times (Y \cup \{\emptyset\}) \rightarrow Q \times (X \cup \{\emptyset\}) \times Pwr(X)$$
$$: (q, y) \mapsto (q', x, X')$$

The function Ω prescribes that at the current state q ($q = \emptyset$ when no state is arrived yet) if the event corresponding to $y \in Y$ occurs ($y = \emptyset$ when the move to the next state is unconditional) then move to state q', latch the variable x and unlatch all

Table 3.1: List of rungs in LLD defining latch/unlatch function

Rung #	Current state	Event	Next state	Latch	Unlatch
	$Q \cup \{\emptyset\}$	$(Y \cup \{\emptyset\})$	Q	$(X \cup \{\emptyset\})$	$Pwr(X)$
1		x^0	0		
2	0	x_a	1	$\emptyset$	$\emptyset$
3	1	x_b	2	$\emptyset$	$\emptyset$
4	2	$\emptyset$	3	$x_{c_{hi}}$	$\emptyset$
5	2	$\emptyset$	3	$x_{c_{lo}}$	$\emptyset$
6	3	x_e	9	$\emptyset$	$\emptyset$
7	3	$\emptyset$	4	$x_{d_{on}}$	$\emptyset$
8	4	x_b	5	$\emptyset$	$\emptyset$
9	4	x_f	8	$\emptyset$	$\emptyset$
10	5	x_f	6	$\emptyset$	$\emptyset$
11	6	$\emptyset$	7	$\emptyset$	$\{x_{c_{hi}}, x_{c_{lo}}\}$
12	7	$\emptyset$	0	$\emptyset$	$\{x_{d_{on}}\}$
13	8	$\emptyset$	9	$\emptyset$	$\{x_{d_{on}}\}$
14	9	$\emptyset$	0	$\emptyset$	$\{x_{c_{hi}}, x_{c_{lo}}\}$

variables in X' (x or X' are set to $\emptyset$ when no signal needs to be latched or unlatched, respectively).

In the LLD implementation of the given example, shown in Figure 3.4[1], rung 1 is devoted to *initialize* the initial state of the supervisor; in other words, the operator sets the variable x^0 to 1 when he wants to start the PLC controller. After initialization, the supervisor moves from state 0 to state 1 in response to the occurrence of the operator-supplied event a, which is why we place the corresponding variable x_a in rung 2 as input.

At state 2, two controllable events, namely c_{lo} and c_{hi}, can be generated by the supervisor; therefore the operator needs to tell the PLC which variable $x_{c_{hi}}$ or $x_{c_{lo}}$ must be latched. If the user does not pick one, the PLC will deterministically choose the one whose rung appears first in the ladder. If it is desired to give user the

[1]The devices in the input/output card are used for test purposes only.

Table 3.2: List of rungs in LLD defining operator input signals

Rung #	Current state	Event	Next state	Latch	Unlatch
	$Q \cup \{\emptyset\}$	$(Y \cup \{\emptyset\})$	Q	$(X \cup \{\emptyset\})$	$Pwr(X)$
$4'$	2	$x^{c_{hi}}$	3	$x_{c_{hi}}$	$\emptyset$
$5'$	2	$x^{c_{lo}}$	3	$x_{c_{lo}}$	$\emptyset$

freedom to choose the event to be generated, as shown in Table 3.2 we can add two new operator input signals $x^{c_{hi}}$ and $x^{c_{lo}}$ in rungs 4 and 5 respectively in order to let the user decide which event c_{hi} or c_{lo} is to be generated, respectively. In this case we say the PLC controller is in the *manual mode*. If there is only one outgoing *external controllable* event, PLC can automatically generate the event as in the case of rung 7.

Note that since PLC scan time is faster than the plant response time, the rungs corresponding to plant's (uncontrollable) response events are placed above the rungs corresponding to PLC generated (controllable) events. Thus, at state 3 the occurrence of the uncontrollable event e is checked *before* the controllable event d_{on} is generated (rungs 6 and 7). If we alter this arrangement, plant events may never get the opportunity to occur.

Rungs 8 and 9, corresponding to uncontrollable events b and f, can be arranged in any order. Here we assume that uncontrollable events do not occur simultaneously; if they do, the event whose rung appears first is selected deterministically by the PLC controller.

To execute LLD on PLC, all input contacts must receive their corresponding signals either from external devices or internal sources, and latch/unlatch coils must send their corresponding signals either to external devices or internal sources. It is evident from Figure 3.4 that all I/O signals have their own sources. We validated the given example by the PLC simulation software Automation Studio 5.2 [12] and verified by running a few test cases that the behavior of the DES supervisor is preserved.

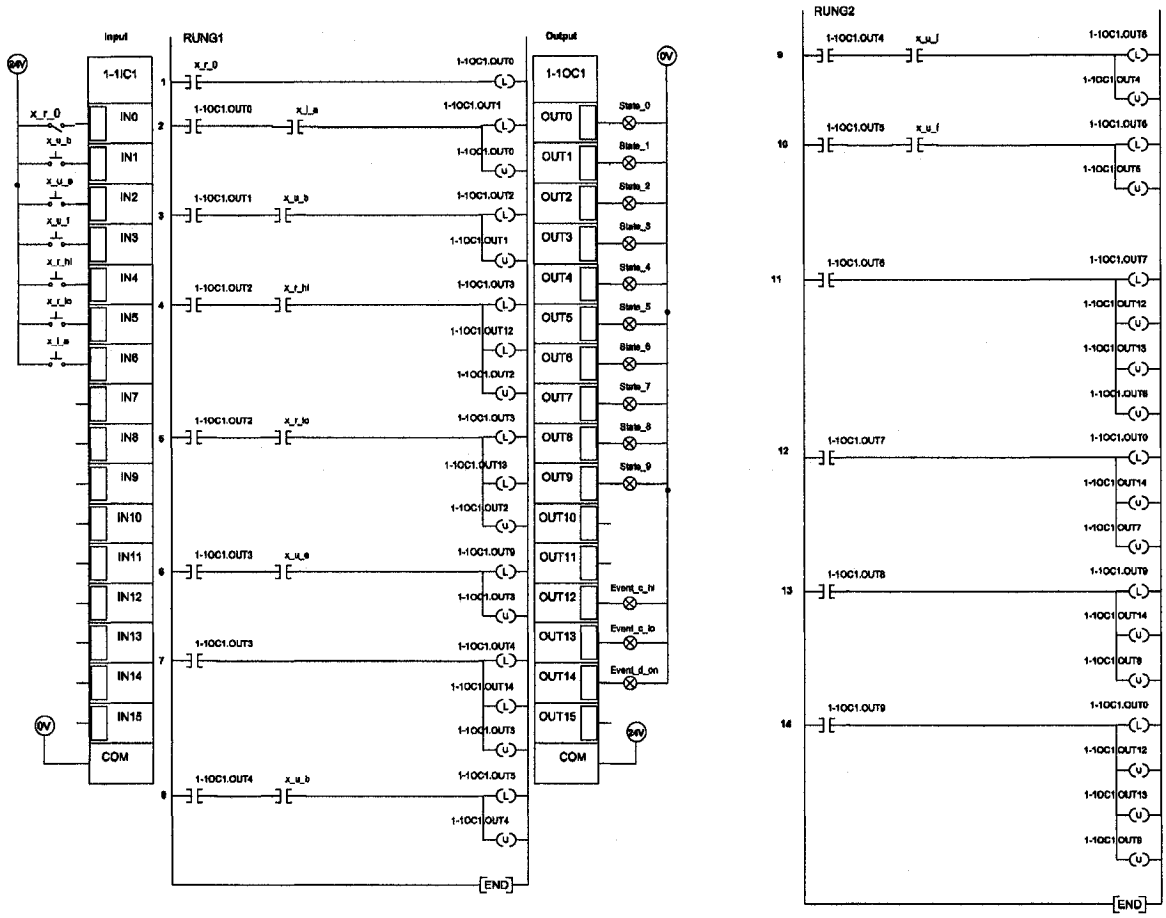


Figure 3.4: Motivating Example in LLD.

3.4 Conversion method

3.4.1 Assumptions and key observations

We make the following assumptions and key observations about our conversion algorithm.

1. For our conversion method, we assume that all supervisors are modeled by deterministic automata [7], [8].

2. We ignore the effect of transmission delay in the PLC model. Thus we assume that our I/O devices respond/activate-deactivate instantly and they are completely error-free.

3. The self-looped $(\Sigma_{u,i} \cup \Sigma_{u,p})$ events in the supervisor are omitted in the PLC implementation as they do not change the supervisor's state and thus the set

of enabled events at that state.

4. The supervisor must require that an unlatching event be generated before the next corresponding latching event can be generated. This necessitates, for example, resetting all binary-state switches before returning to the initial state of the supervisor.

5. PLC can implement a supervisor in two modes: in *automatic mode* once the PLC starts, it will automatically perform its logic to drive the output devices. In *manual mode*, when the PLC starts, it executes its logic in response to the user input to drive the output devices.

6. We generally follow a specific rung order in our implementations. In manual mode, rung order does not matter as the user input determines the rung to be executed. In automatic mode, we place rungs corresponding to uncontrollable events first and then we place rungs corresponding to controllable events in arbitrary order. In either case, the rung order evolves according to supervisor states.

Assumption 4 mainly describes why it is important to add all necessary events in the supervisor. This assumption is also essential to directly convert DES supervisor into LLDs. If any event is missing in the DES supervisor, the user may need to construct an extended supervisor by attaching the missing events to the original supervisor as addressed by [7]. To avoid such complications we assume that all necessary events are already included in the supervisor. We now present our conversion algorithm in two steps. First, we partition supervisor event set into three alphabets: plant response alphabet, PLC command alphabet and input alphabet. Second, we convert supervisor's automaton into a ladder logic diagram. In our implementation we clearly define sets of boolean signals corresponding to PLC input contacts (signals) and output variables driving the output coils; we also define rung latch/unlatch function which describes how one can create LLD using the I/O variables. The resulting LLD can then be readily executed by the PLC.

3.4.2 First step: Event partition

The general architecture of the system is shown in Figure 3.5.

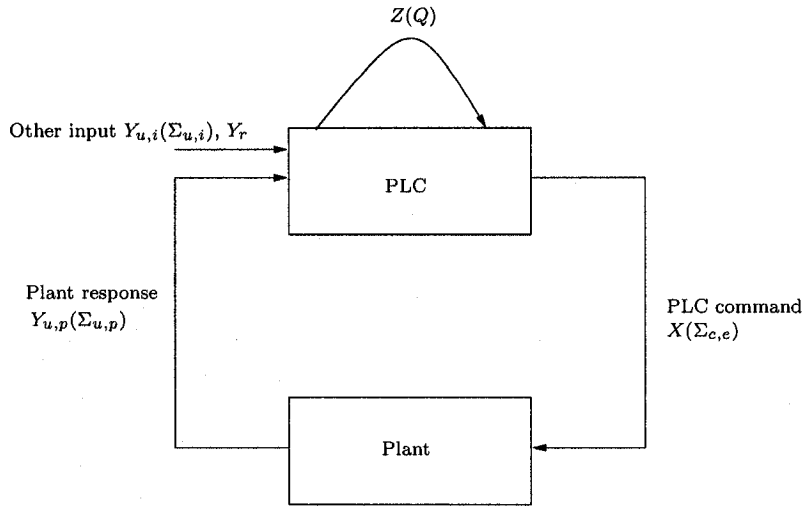


Figure 3.5: PLC-based supervisory control.

Let $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$ be the alphabet of events. We partition the set of uncontrollable events (Σ_u) into two alphabets $\Sigma_u = \Sigma_{u,i} \dot{\cup} \Sigma_{u,p}$.

The alphabet of *external events*, denoted by $\Sigma_{c,e}$, is the collection of controllable events (Σ_c) that PLC must generate by latching/unlatching output coils which drive their corresponding output devices. The alphabet of *input events*, denoted by $\Sigma_{u,i}$, is the collection of uncontrollable events that are generated by other agents, e.g. an operator. The alphabet of *plant response events*, denoted by $\Sigma_{u,p}$, is the collection of uncontrollable events that are generated by the plant.

3.4.3 Second step: I/O selection

We represent a ladder logic diagram by a quadruple:

$$G_c = (X, Y, \Omega, Z)$$

Description of each component as follows.

PLC commands/outputs. We assume that all external controllable events are to be generated by the PLC. Recall from the example that switch events with binary status are implemented with one boolean variable, and switch events with a ternary status are implemented using two boolean variables. Some events (such as α_{on} and β_{hi}) are triggered on the rising edges of their corresponding variables while other events (such as α_{off} and β_{cl}) are triggered on the falling edge of the logical OR of

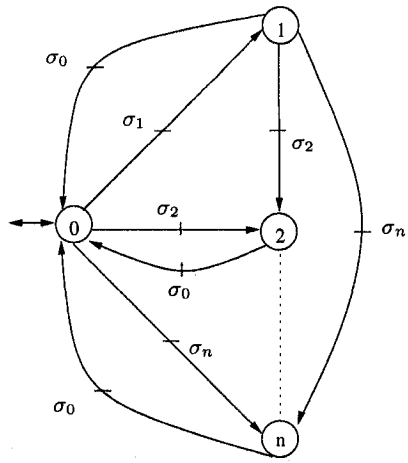


Figure 3.6: A $(n+1)$-state switch.

their corresponding variables. In general we model an output device as a $(n+1)$-state switch shown in Figure 3.6. The switch is deactivated when it is in state 0, while when it is activated it can be in any of the states $\{1, 2, \ldots, n\}$. When the switch is in state $0 \leq i \leq n$, it moves to state j, $0 \leq j \neq i \leq n$, upon the occurrence of σ_j.

For a $(n+1)$-state switch s define:

$$\Sigma_s := \{\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_n\}$$

Define the corresponding boolean variables as:

$$X_s := \{x_\sigma |\ \sigma \in \Sigma_s \wedge \sigma \neq \sigma_0\}$$

where

$$\sigma \text{ occurs} \Leftrightarrow \begin{cases} x_\sigma \uparrow & ;\sigma \in \{\sigma_1, \sigma_2, \ldots, \sigma_n\} \\ \bigvee_{\sigma' \in \{\sigma_1, \sigma_2, \ldots, \sigma_n\}} x_{\sigma'} \downarrow & ;\sigma = \sigma_0 \end{cases}$$

The set of PLC-generated external events is:

$$\Sigma_{c,e} := \bigcup_{s \in S} \Sigma_s$$

where S is the set of all switches. Finally, define the set of boolean signals representing the external events as:

$$X := \bigcup_{s \in S} X_s$$

PLC inputs. Corresponding to each $\sigma \in \Sigma_{u,i} \cup \Sigma_{u,p}$ define a signal x_σ that becomes momentarily true when $\sigma \in \Sigma_{u,p}$ is generated by the plant, or $\sigma \in \Sigma_{u,i}$ is generated

by the operator. Define:

$$Y_{u,p} := \{x_\sigma |\ \sigma \in \Sigma_{u,p}\},\ Y_{u,i} := \{x_\sigma |\ \sigma \in \Sigma_{u,i}\}$$

Also define

$$Y_r := \{x^0\} \cup \{x^\sigma |\ \sigma \in \Sigma_{c,e}\}$$

The user sets x^0 to 1 to initialize the PLC in the beginning of the program. When in the manual mode, or in the automatic mode when two or more external controllable events are possible in a state, an enabled event $\sigma \in \Sigma_{c,e}$ is generated by the PLC if x^σ is set to 1 by the user. Finally, let $Y := Y_{u,i} \cup Y_{u,p} \cup Y_r$.

PLC states. We define a set of boolean signals to represent the PLC states. Let

$$Z = \{x_q | q \in Q\}$$

The interpretation is that the supervisor is in state q iff $x_q = 1$. Thus at any point in time we require that

$$\exists! q \in Q.\ x_q = 1$$

When the PLC is initialized by the user, x_{q_0} is latched. When a transition from q to q' takes place, $x_{q'}$ is latched while x_q is unlatched.

Rung latch/unlatch function. We define a function Ω to model a LLD. It describes how events in $\Sigma_{c,e}$ are to be generated by latching/unlatching the variables in X. When the function prescribes a state change from q_i to q_j, latching of q_j and unlatching of q_i are implied.

$$\begin{aligned}\Omega : (Q \cup \{\emptyset\}) \times (Y \cup \{\emptyset\}) &\rightarrow Q \times (X \cup \{\emptyset\}) \times Pwr(X) \\ &: (q, y) \mapsto (q', x, X')\end{aligned}$$

The function Ω prescribes that at the current state q ($q = \emptyset$ when no state is arrived yet) if the event corresponding to $y \in Y$ occurs ($y = \emptyset$ when the move to the next state is unconditional) then move to state q', latch the variable x and unlatch all variables in X' (x and X' are set to $\emptyset$ when no signal needs to be latched and unlatched, respectively).

Chapter 4

Case study: boiler control system

4.1 Introduction

Discrete-event systems encompass a wide variety of physical systems such as manufacturing systems, traffic systems, communication protocols, logistic management systems and data communication networks. Due to its discrete-event nature, a petrochemical boiler control system can be characterized as a discrete-event system. Although boiler controller is commercially available in marketplace, the existing boiler controllers are not constructed within a formal framework. Since there are lots of safety issues and complexities involved in the design of a boiler control system, it is quite appropriate and necessary to synthesize a boiler controller based on formal methodologies such as Petri nets and Ramadge-Wonham Supervisory Control Theory (SCT).

The controllers, which can be synthesized automatically utilizing SCT [1] for a given DES plant and a specification's automaton representing control objectives, are mathematically guaranteed to be safe within the behavioral constraints; and are nonblocking. Therefore, for our test example we choose water bath boiler controller which is widely used in the petrochemical industry. It is now appropriate to provide a brief review of boiler and its control procedures.

4.1.1 A brief description of boiler

A boiler is a confined vessel and/or tubes where water or other high efficient heat transfer chemicals are heated at desired temperature and pressure by injected mixture of air and fuel to produce steam or hot chemical. This steam or the heat extracted

from the hot chemical (by heat exchanging method) is then transferred out of the combustion chamber for applying in a variety of heating applications such as industrial process heating, residential heating, space heating, etc. There are many types of boilers available in the market according to pressure range, temperature range, working fluid, fuel type, and other characteristics: fire-tube and water-tube boilers are very common types of boilers. We choose fire-tube boiler for our work, in which water or a mixture of water and glycol to be converted to steam is filled in a vessel (referred to as bath), and a fire tube and process coil are submerged in the bath, which transfers heat to the process stream in the coil as shown in Figure 4.1 (figure not in scale).

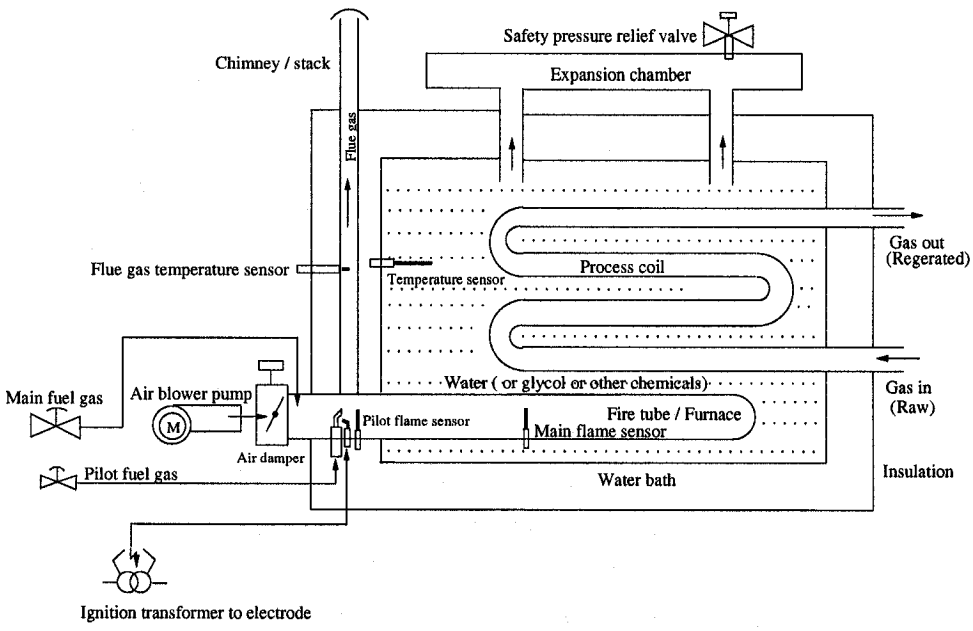


Figure 4.1: Water bath boiler.

As a combustion fuel, a boiler can use natural gas, oil, coal, etc. We use natural gas as a source of fuel. Generally, all boilers contain the following four components:

Burner: A mechanical arrangement allows injecting proper mixture of fuel and air into the boiler's combustion chamber. It also holds firing and sensor devices.

Tubes: In a fire-tube boiler, a steel conduit tube is submerged in water or other chemicals where fuel is burnt, and generated heat is transferred to the water or other chemicals. Water passes through the tubes in a water-tube boiler.

Stack: The stack is a route through which the combustion gases are exhausted to the outside atmosphere. The gases in the stack are called stack gases or flue gases.

Safety valve: To prevent pressurization and possible explosion, a safety valve is essentially adopted in a boiler which automatically open and release the pressure when the boiler vessel pressure exceeds a safe limit.

4.2 A typical boiler control system

Other than basic components, boilers have a set of combustion and safety control devices which together are called boiler controller. The main objectives of the controller are:

- To operate the boiler to maintain efficient and continuous supply of steam at the specified temperature and pressure.
- To allow safe start up, shut down, detect emergency conditions and take appropriate action for safe operation at all times.

Many control procedures are necessary for automatically ignited boilers to ensure safe, consistent, and efficient operation. Control procedures are categorized according to the functions they perform for a boiler. General control procedures are:

- Operating controls: These are the devices that control the operation of boiler to start, stop, and modulate the burner, maintain required water level, and maintain proper water pressure.
- Limit control: When operating limits of the boiler are reached (i.e. hot water temperature too high, water level too low, high stack temperature, no fuel, etc.), these control devices are responsible for shutting down the boiler.
- Safety control: These devices interrupt fuel flow to the burner and close the equipment when unsafe operation conditions such as ignition failure and main flame failure are detected.
- Programming controls: When all necessary conditions for a proper burner operation are met (i.e. pre-purge and post-purge cycles) these control procedures ensure proper sequencing of all of the above control systems.

A boiler control system can be implemented using relay logic controller, PID controller, and presently using PLC. A typical PLC-based boiler controller is shown in Figure 4.2.

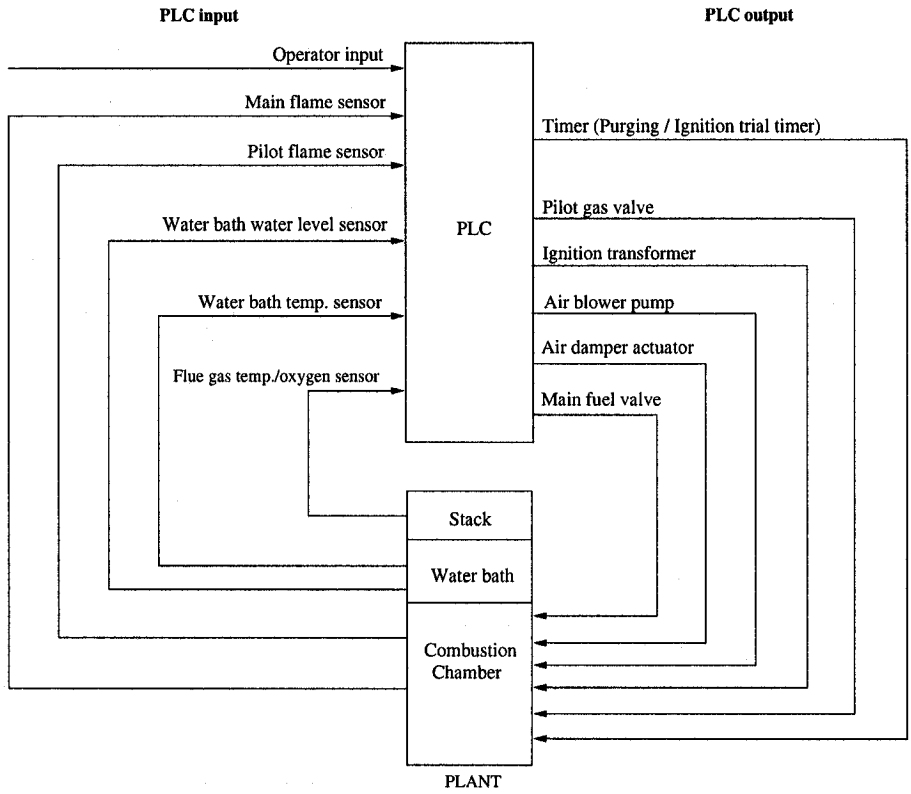


Figure 4.2: PLC-based boiler controller.

There are many kinds of sensors and switches available, which are employed to achieve boiler control objectives when implementing using PLCs.

Typical Input Devices: Operator selector switch (high stage, low stage, off), flame sensor, fuel-gas flow sensor, water bath temperature sensor, water bath water level sensor, flue gas oxygen sensor.

Typical Output Devices: Pilot gas valve (solenoid), ignition transformer, air blower pump, air damper actuator, main burner gas valve (solenoid), variable speed drive (optional). Here, air damper actuator is used to provide on/off control for air dampers, and act as VAV (Variable Air Volume) unit.

Details of the systems, objectives and I/O devices can be found in [15], [17], [18], [19], [20], [21].

4.2.1 Design objectives

We design basic integrated PLC-based controllers of the above control systems mentioned in section 4.2. We design two controllers to achieve the aforesaid control objectives. These are:

- Pilot flame controller is for controlling the pilot flame which must be formed within two ignition trials or within the time allowed by the user setting in ignition trial timer.
- Main flame controller is for controlling the main flame which starts its operation when it receives the appropriate signal from the pilot controller.

We design a complete boiler controller integrating both pilot and main flame controllers which can solely be used to achieve both pilot and main flame controllers objectives.

4.3 Modeling boiler control devices as DES

A plant can be modeled at different levels of abstraction. There is no definite rule to aid a designer to accurately model a plant. It depends only on the designer's discretion how to show all possible behaviors of the plant in an automaton model. Ryan Leduc mentions this problem in his thesis [22] by posing the question, "what conditions does a model need to meet to make it valid for a given plant?" He explained that it is important to include the events required to properly observe and control the plant. Further, with respect to these events, the plant model is valid if it correctly generates the sequence of events (strings) that the plant would generate. It is quite natural that events only occur in the plant when the plant model allows that event to be generated.

We adopt Balemi's [11], [23] notion about input/output perspective on control of discrete-event systems. Balemi mentioned that the controller commands can be seen as controllable events, while the plant responses can be seen as uncontrollable events. He pointed out that supervisor should be such that it cannot prevent the generation of plant responses, similarly the plant cannot prevent the generation of controller commands. As a plausible solution, the supervisor never generates any command which the plant cannot follow.

Our main approach of modeling plants is closely related to Balemi's concept; indeed, we observe that a plant model is required to include all of its relevant events (i.e. response and command events) regardless of whether they are to be generated by the plant or supervisor. Once PLC's input/output devices are identified they can be modeled in the DES framework as described in the next section.

4.3.1 Modeling input devices

These are mainly the various kinds of sensors (on/off switching devices) which communicate the current status of the output devices to PLC. Output devices can talk to PLC only through these devices. In other words, PLC observes the response of the output devices through the input devices. Since input devices are monitoring devices, PLC just listens to them but does not necessarily control them; therefore, we model input devices as shown in Figure 4.3 where for the ease of implementation, we include a sensor initialization event which is controllable because PLC initializes the sensor using this event to let the sensor communicate its status, while for obvious reasons, response events (yes-flame, no-flame, etc.) are uncontrollable.

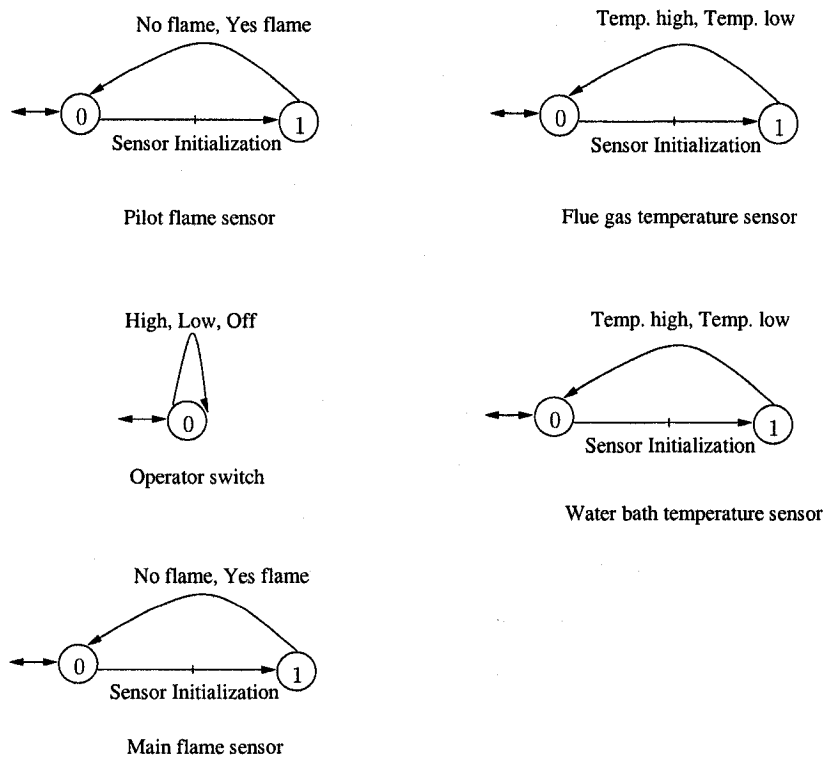


Figure 4.3: Some input devices.

The PLC receives another type of input signals that are generated by operator selector switch, thermostat call switch, user input signal switch, etc. An operator feeds such events to PLC, and consequently they are uncontrollable to the PLC, as shown in Figure 4.3.

4.3.2 Modeling output devices

Output modules of PLC are connected to different kinds of on/off electrical devices. Typical output devices are motor and electrical solenoid. All events associated with the output devices must be controllable. It is perceptible to assume that a motor cannot be turned on automatically, something (which is PLC) should generate on event so that the motor can run. Further, PLC should always control the output devices by generating relevant events in order to acquire them to behave within some specifications. Figure 4.4 shows the suggested output device models.

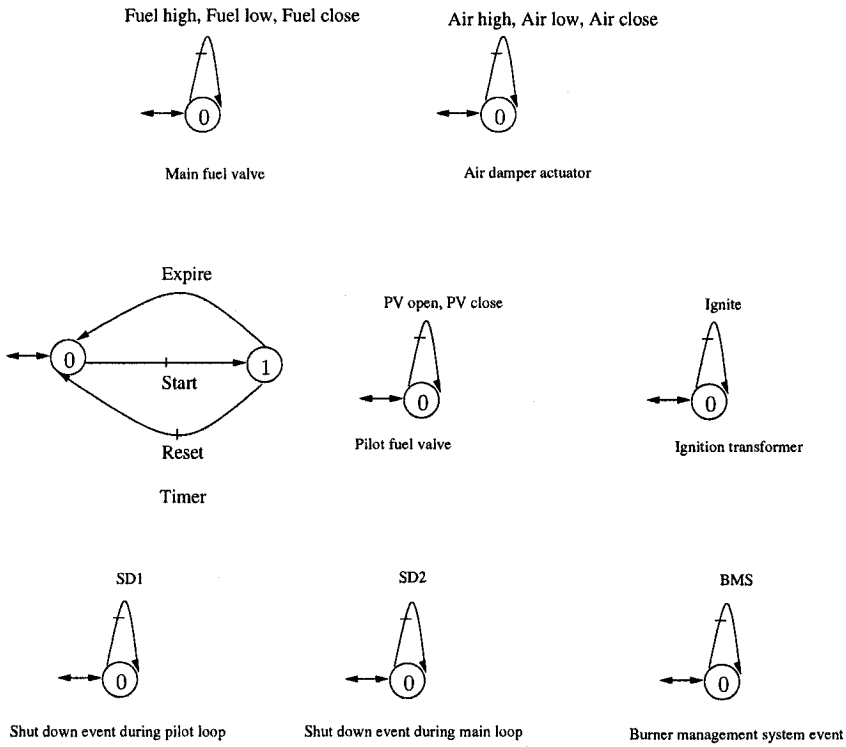


Figure 4.4: Some output devices.

The model shown in Figure 4.4 for air damper actuator and main fuel valve is an abstract model with some problems. First, this model suggests that any of its events can occur unlimited number of times without the occurrence of the closing event. In

such a case, even when the specification does not require it *explicitly*, the closing event must be generated by PLC when needed to start a new cycle. Second, since the model has only one state which is marked, if the specification requires to activate a high/low event, the switch will be activated to high/low, but the problem is that the switch can stay in high/low state forever because the state it is in is marked. It is not acceptable to have a switch on forever, and we would like the switch to eventually return to the close/off state. Again, in this case the device must be deactivated (closed) by PLC even when it is not required by the specification. Therefore, alternative models for an air damper actuator and main fuel valve adopted as shown in Figure 4.5.

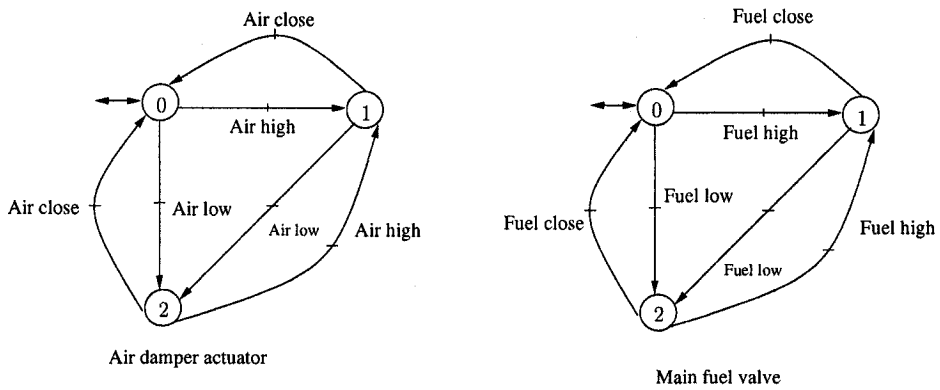


Figure 4.5: Alternative model for air damper and main fuel valve.

The new models do not pose any of the previous problems. We use both old and new models in our design work and demonstrate how they differ from each other, and finally we adopt the alternative model as a general representation for air damper actuator and main fuel valve.

Much attention is required when modeling a timer. Typical ignition trial timer model is shown in Figure 4.4. PLC can only turn the timer on and has to wait until it expires in order to allow a specified time for the flame to be formed. Therefore the *timer on* event is controllable while the *timer expiry* event is uncontrollable. In the operation specifications, we allow two ignition trials or a 15 seconds time window to ignite the flame. If the flame is formed within two trial periods, PLC needs to reset the ignition trial timer so that this timer is ready for the next cycle operation, hence we label timer reset event as controllable. Same arguments are applied to model other timers.

When modeling other output devices, we can follow all of the above observations.

The other notable device BMS describes the Burner Management System (BMS) event, which is controllable. The purpose of the BMS is to automatically, or after maintenance work starts, to place burners and igniters in service, to monitor flame conditions, to provide alarm in any abnormal conditions, and to remove burners and igniters from service when necessary. Since we design two controllers (pilot and main flame), we define two shut down events: one is for using in pilot operation period and the other is for main flame operation period. The purpose is the same: shut down the plant if any emergency conditions arise, and for obvious reasons these events are controllable.

4.4 Supervisor construction

4.4.1 General considerations

The following general considerations are important to our design work:

- Boiler can be controlled to supply two stage (high/low) output temperatures.
- Controller is intended to control a single burner unit (it is possible to design a controller which can control many burner units together).
- We consider few critical emergency conditions of the boiler in our controller design, while there are many other emergency conditions (water level low, no fuel flow, etc.) and many other design criteria (including variable motor speed drive with pump, flue gas oxygen analyzer with stack, etc.) can be included in the design.
- The controller is used to control the water bath boiler (sometimes called Regeneration Heater) in the petrochemical industry. Our example is adopted from the boiler used by [24].
- When any of the controllers move to a state called Burner Management System (BMS), we assume that BMS work can be done by one event as shown in Figure 4.4. In practice, BMS work is done by interfacing another controller which we leave for future works.

4.4.2 Pilot flame controller

A pilot flame controller is used to fire the pilot flame before the main burner ignition. Typical pilot ignition sequence is that a pilot flame is ignited within specified design criteria and is lit, a flame is detected by a flame sensor, and the main flame is then ignited. Pilot can be classified depending on the life time of a pilot with respect to the main burner. A pilot can be continuous, intermittent and interrupted. We choose to design continuous pilots which stays lit all the time during the system operation, and we use natural gas as a pilot fuel source. Overall, the objectives of pilot controller is to control the pilot flame which must be formed within two ignition trials or within the time allowed by the user.

DES model of pilot controller plants

A typical pilot flame controller is composed of the following sensors and switches: operator selector switch (high, low, off), flame sensor, pilot gas valve (solenoid), ignition transformer and timer. The corresponding DES model of the devices which we use to design pilot controller is shown in Figure 4.6.

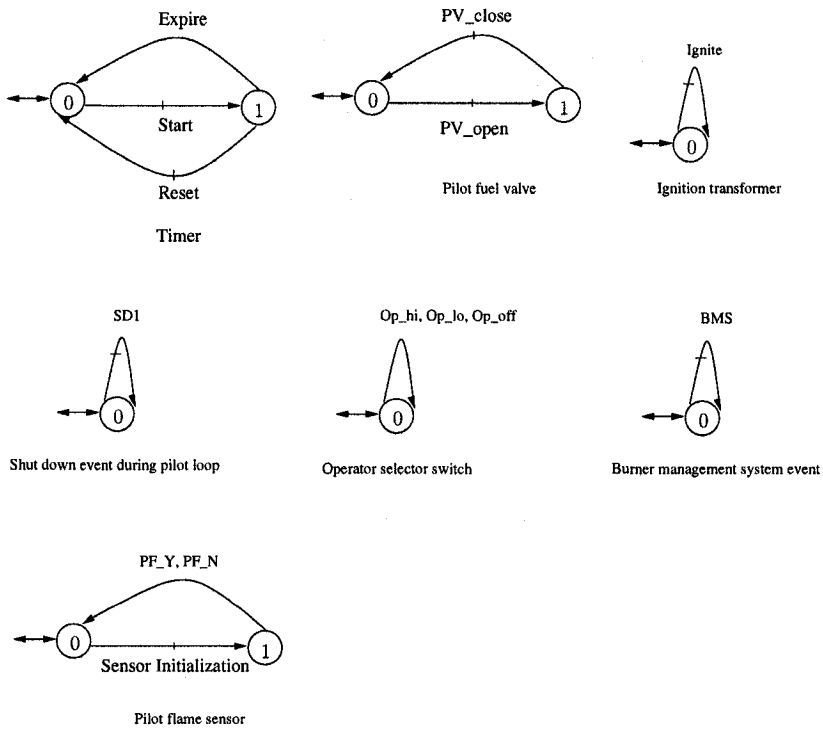


Figure 4.6: Pilot controller I/O devices.

Pilot flame control specification

The operation sequence of the pilot flame controller is shown in Figure 4.7.

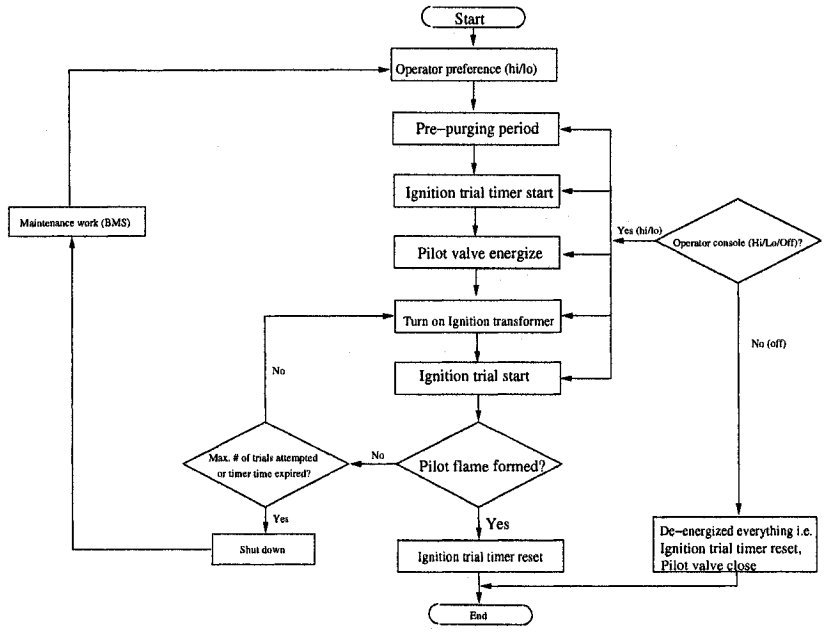


Figure 4.7: Operation sequence of Pilot controller.

The operation of pilot flame controller without operator's role is mainly defined as follows:

- Allow two ignition trials or allow the time set by the user (we use 15 seconds) to ignite the flame.
- Shut down the plant if the number of ignition trials exceeds 2, or ignition timer expires, whichever happens first.
- Ignition transformer is ignited only when pre-purging timer is expired, pilot valve is open and there is no flame, otherwise the transformer will not be ignited.
- We check in sequence. First pre-purging timer status, then pilot valve status and finally flame sensor status, before the ignition transformer is ignited.
- If pilot flame is formed, pilot valve will be stayed open and timer will be reset.
- After the occurrence of shut down event, plant will move to a state of burner management system, and it is re-initialized.

The initial time delay between operator call and trial for ignition is referred to as pre-purging period. It is required to allow purging time to clear out any incombustible gas from combustion chamber.

In the pilot operation sequence, operator can only turn the system on or off. Operator's role in pilot operation can be described as follows:

- Operator can select either high/low/off at any time (state). Initially (at state 0), when operator selects high/low, plant will follow pilot operation sequence. During the whole sequence, supervisor just carries operator's high/low signal to the main flame loop.
- At any state, when operator selects off, controller brings the plant to an initial state (state 0) by closing or resetting pilot valve and timer when they are on.

The resulting specification for the pilot flame controller is shown in Figure 4.8 and named as PILOTSPEC for TCT design procedure.

Pilot flame supervisor

We now synthesize the pilot flame supervisor [1] using supervisory control theory of Ramadge and Wonham [1] and XPTCT software [14]. We particularly follow the steps mentioned in [9] to design DES supervisors. For pilot flame supervisor, we first list the devices that are interacting with each other. These are flame sensor, pre-purging timer, trial for ignition loop timer, pilot valve, ignition transformer, shutdown event during pilot loop and burner management event. We then take synchronous product of the model of these devices.

- PLANT A = sync (Flame sensor, Pre-purging timer)
- PLANT B = sync (PLANT A, Trial for ignition timer)
- PLANT C = sync (PLANT B, Pilot valve)
- PLANT D = sync (PLANT C, Ignition transformer)

[1] We use *controller* to refer to PLC implementation, while we use *supervisor* to refer to DES design.

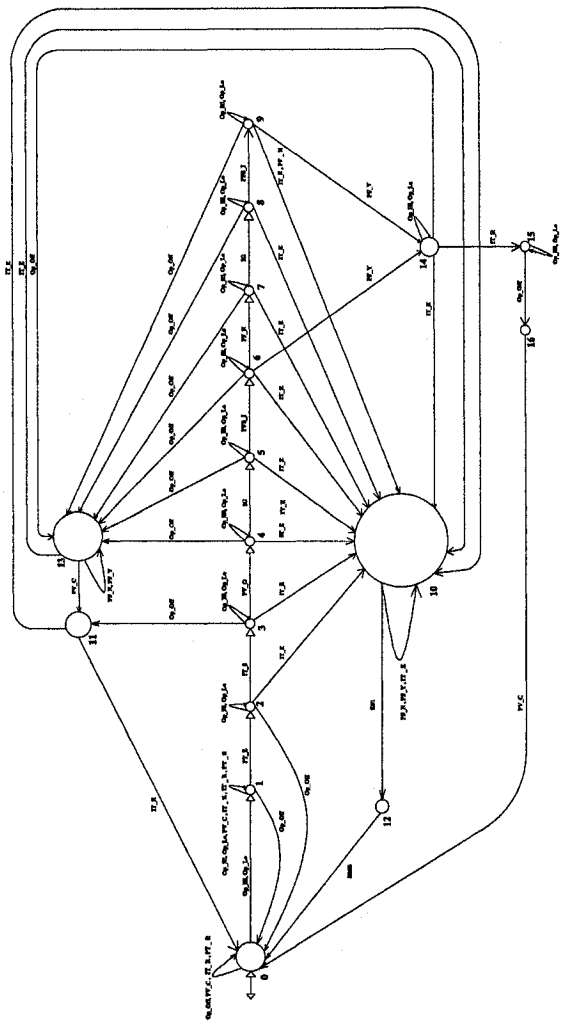


Figure 4.8: Specification for pilot flame supervisor (TCT: PILOTSPEC).

- PLANT E = sync (PLANT D, SD1)
- PLANT F = sync (PLANT E, BMS)
- PLANT G = sync (PLANT F, Operator)

Now we construct the pilot flame supervisor by taking the supcon of the pilot control specification (PILOTSPEC) with respect to all plant components involved in the pilot stage (PLANT G):

- PILOTSUPERVISOR = supcon (PLANT G, PILOTSPEC) (States: 21, Transitions: 75)

We use `minstate` to minimize the size of a supervisor whenever possible.

The resulting pilot supervisor's automaton is shown in Figure 4.9. It is clear from Figure 4.9 that the controlled behavior of the pilot supervisor is quite expected as described in the specification, and it is controllable.

4.4.3 Main flame controller

Upon receiving the appropriate signal from the pilot controller, main flame controller will start its operation. In this loop, controller mainly controls the devices to ignite the main flame and to control the main flame according to operator's preference. If any emergency conditions arise, controller will also take appropriate actions to achieve safe boiler operation. We use natural gas as a main burner fuel source.

DES model of Main controller plant components

A typical main flame controller is comprised of operator selector switch (high, low, off), flame sensor, main gas valve (solenoid), flue gas temperature sensor, water bath temperature sensor, air blower pump, air damper actuator and timer. The corresponding DES models of the devices for designing main flame controller is shown in Figure 4.10.

Main flame control specification

The overall operation sequence of the main flame controller is shown in Figure 4.11.

We describe the operation of main flame supervisor step-by-step. First consider the normal operation (no emergency conditions), which can be described as follows:

- Main flame loop should start if and only if pilot flame is formed
- Devices operate in sequence: First air damper actuator is energized, then air blower pump is turned on in order to avoid pump motor being overloaded. After that, main gas valve is energized which allows good combustion and less pollution [17], [18].
- Operator can select either high/low at any time (state). Initially, when operator chooses high/low, plant will follow main burner high/low path, respectively, as shown in the specification of Figure 4.12 and Figure 4.13.

- When the plant is running (main valve open, pump on, damper on) in a particular mode (high/low), if the operator changes his (low/high) selection, the controller will wait until the main flame is formed, then it will change only the opening volume of air and gas of air damper actuator and main gas valve accordingly.

- If the operator selects off at any state, the controller will move the plant to an initial state by closing main valve, pump and air damper, whichever other device which is on.

- If shut down occurs, plant will move to a state of burner management system, and it is re-initialized.

During normal operation, controller will continuously monitor safe operation of the main burner. At any stage, if emergency conditions (temperature too high, flame not detected, etc.) arise, controller will shut down the boiler and move the system to a repair state for maintenance work, and prepare it for a new operation cycle. We now include the emergency issues, listed below, in the main flame specification.

- Water bath temperature exceeds set limit.

- Flue gas temperature exceeds set limit.

- Main flame fails to form.

It is required that water bath temperature be maintained at a desired temperature level. To ensure complete combustion of fuel, proper mixture of air and fuel is required. But if the amount of excess air increases, flame temperature decreases and the boiler heat transfer rate goes down. As a result, flue gas temperature goes up; therefore regular monitoring of flue gas temperature can ensure proper proportion of excess air which greatly improves combustion efficiency [17], [18].

The proposed specification for the main flame controller is shown in Figures 4.12 and 4.13, and named MAINSPEC1 for TCT design procedure.

We introduce another specification MAINSPEC2 shown in Figure 4.14. This specification ensures that when BMS event occurs, air damper and fuel valve are closed, and plant is moved to the initial state to prepare for the next cycle. Also, it ensures that all devices are closed/off when plant is in initial state.

Main flame supervisor

For main burner control supervisor, first we list all devices that are interacting with each other. These are air damper actuator, main fuel valve, pilot flame sensor, main flame sensor, air blower pump, operator switch, shut down event during the main cycle and burner management event. We take the synchronous product of the above devices.

- PLANT H = `sync` (Air damper actuator, Main fuel valve)
- PLANT I = `sync` (PLANT H, Air blower pump)
- PLANT J = `sync` (PLANT I, Pilot flame sensor)
- PLANT K = `sync` (PLANT J, Main flame sensor)
- PLANT L = `sync` (PLANT K, SD2)
- PLANT M = `sync` (PLANT L, BMS)
- PLANT N = `sync` (PLANT M, Operator)
- PLANT O = `sync` (PLANT N, Flue gas temperature sensor)
- PLANT P = `sync` (PLANT O, Water bath temperature sensor)

We have two specifications for main flame supervisor; we can combine them using `meet` to obtain the design specification.

- SELF1 = `create` (SELF1, [mark 0])
- SELF1 = `selfloop` (SELF1, [list of all plants events in the main flame supervisor])
- MAINSPEC1 = `sync` (MAINSPEC1, SELF1)
- MAINSPEC2 = `sync` (MAINSPEC2, SELF1)

- MAINSPEC = `meet` (MAINSPEC1, MAINSPEC2)

Now we construct the main flame supervisor by taking the `supcon` of the main flame specification (MAINSPEC) with respect to all plants involved in the main flame supervisor (PLANT P):

- MAINFLAMESUPERVISOR = `supcon` (PLANT P, MAINSPEC) (States: 57, Transitions: 156)
- MMAINSUPERVISOR = `minstate` (MAINSUPERVISOR) (States: 42, Transitions: 136)

The resulting main flame supervisor is shown in Figure 4.15 and Figure 4.16. It is evident from the figures that the controlled behavior of the main flame supervisor is quite satisfactory and controllable.

4.4.4 Complete boiler controller

Now we reach a point to build the complete controller which alone can control the boiler. The purpose of this controller is to achieve the same control objectives as the pilot flame and main flame controllers individually do. For both pilot and main flame stages, we use natural gas as a fuel source.

DES model of plants

A typical boiler controller is composed of the following sensors and switches: operator selector switch (high, low, off), flame sensor, pilot gas valve (solenoid), ignition transformer, timer, main gas valve (solenoid), flue gas temperature sensor, water bath temperature sensor, air blower pump and air damper actuator. The corresponding DES model of the devices which we use to design the complete boiler controller is shown in Figure 4.17.

Boiler control specification: serial specifications

The purpose of the complete boiler controller is to integrate the operations of pilot flame and main flame controllers. We integrate both supervisors by taking their serial (as opposed to parallel) behavior as in the modular operation of DES. The architecture of the complete controller is shown in Figure 4.18. The modular approach to the synthesis of supervisors for DES describes that the overall supervisory task is divided into two or more subtasks and the resulting individual supervisors are run, in our case, sequentially to implement a solution for the original problem. The disjunction of two modular supervisors can achieve the same control objective as a centralized supervisor. Details of the theory will be developed in future work. The overall operation sequence of the complete boiler controller is shown in Figure 4.19, which combines both the pilot and main flame operation sequences.

Pilot and main flame loops are connected as follows: during the pilot execution, whenever pilot flame is formed, the controller will reset the ignition trial timer, usher the plant to the main flame loop (high or low sequence as required by the operator), and follow the main flame operation requirements.

The suggested specification automaton for the complete boiler controller is shown in Figure 4.20, Figure 4.21 and Figure 4.22, and named COMSPEC in TCT design procedure. It is easy to visualize from Figure 4.20, Figure 4.21 and Figure 4.22 that the complete boiler specification is simply the integration between pilot and main flame specifications.

Boiler control supervisor

As usual, we first list the devices that are interacting with each other. Operator selector switch (high, low, off), flame sensor, pilot gas valve (solenoid), ignition transformer, timer, main gas valve (solenoid), flue gas temperature sensor, water bath temperature sensor, air blower pump and air damper actuator. We then take the synchronous product of these devices. We use the same device models both in the

pilot (PLANT G) and main flame stage (PLANT P) as before, which are shown in Figure 4.6 and Figure 4.10, respectively. As a result, plant interactions for the complete boiler supervisor can be computed as:

- PLANT Q = sync (PLANT G, PLANT P)

Now we construct the complete boiler supervisor:

- COMSUPERVISOR = supcon (PLANT Q, COMSPEC) (States: 57, Transitions: 196)

The resulting complete boiler supervisor is shown in Figure 4.23, Figure 4.24 and Figure 4.25. The controlled behavior of the complete supervisor is quite satisfactory, and it is controllable.

4.5 PLC-based implementation of the controllers

We convert pilot flame, main flame and complete boiler supervisors to LLD using our conversion technique, and finally test our design by setting a virtual plant setup using a PLC simulation software Automation Studio 5.2 [12]. All of the resulting LLDs of the PLC-based implementation of the pilot flame, main flame and complete boiler controllers are given in the appendix sections. Here in Figure 4.26[2], we provide a part of LLDs of the pilot flame controller as an example view.

The PLC simulation software does not provide a 'runtime' simulated LLD as a simulation result. To verify the correctness of LLDs, after running a simulation one must verify the status of the output devices in response to the input signals supplied by the user. If all the output devices are activated/de-activated according to logic functions set by the user and in relation with the signals supplied by the input devices, then one can conclude that LLDs are functional; and hence, control objectives are achieved. Therefore, as an example a part of LLD of the pilot flame controller

[2]The devices in the input/output card are used for test purposes only.

is provided here so that one can compare this LLD with the pilot supervisor's automaton to examine whether the same sequence of events is generated by the given LLD as suggested by the supervisor's automaton. If the given LLD generates the same sequence of events, it can be claimed that the converted LLDs are functional. Moreover, it is required in any PLC implementation that all of its input contacts must receive their corresponding signals from some kinds of sources (external or internal) and latch/unlatch coils must send their corresponding signals either to external devices or internal sources. It can easily be verified from the virtual plant setup of boiler control system in the Automation Studio 5.2 that all I/O signals have their own sources, and therefore, all controllers can be executed by the PLC. By running a few test cases we observed that the boiler controllers function as outlined in the specifications.

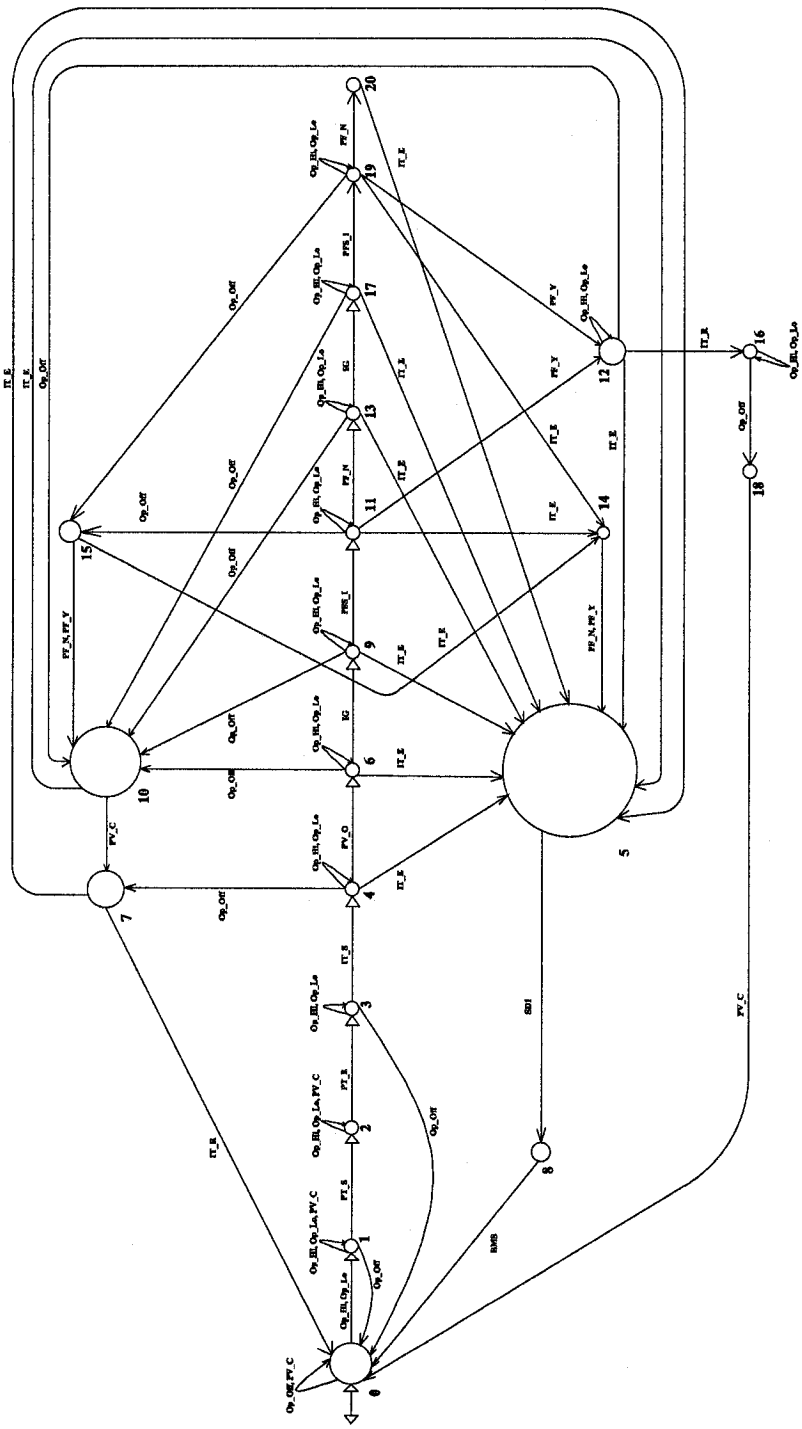


Figure 4.9: Pilot flame supervisor (TCT: PILOTSUPERVISOR).

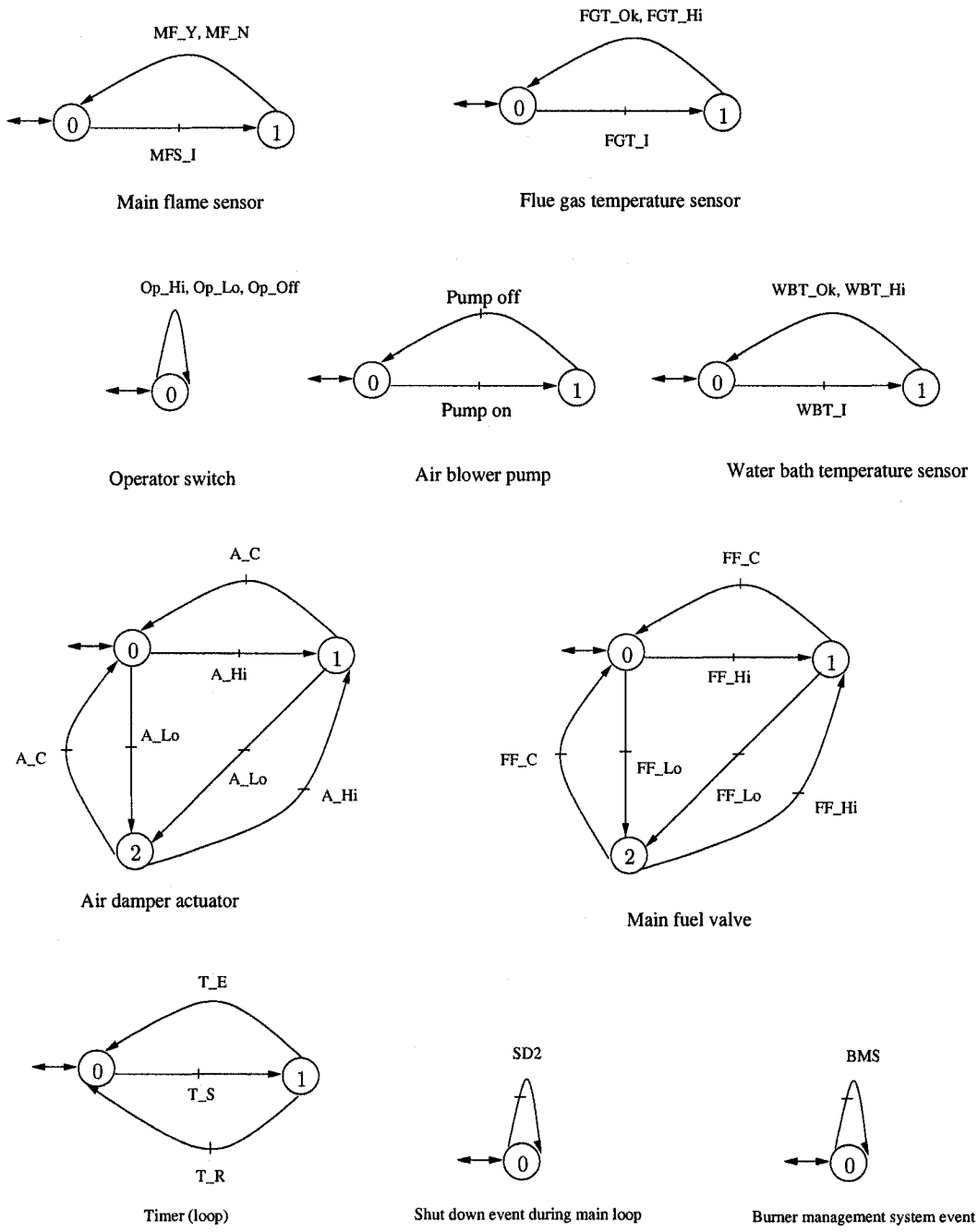


Figure 4.10: Main flame controller I/O devices.

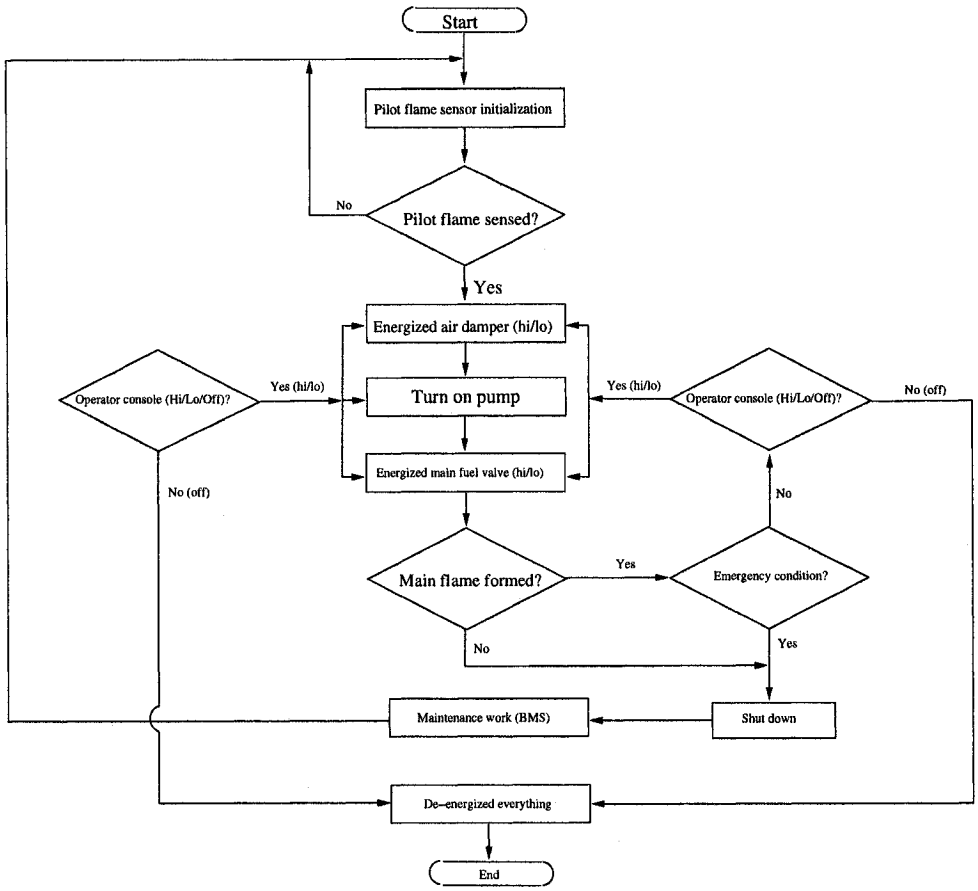


Figure 4.11: Operation sequence of main flame controller.

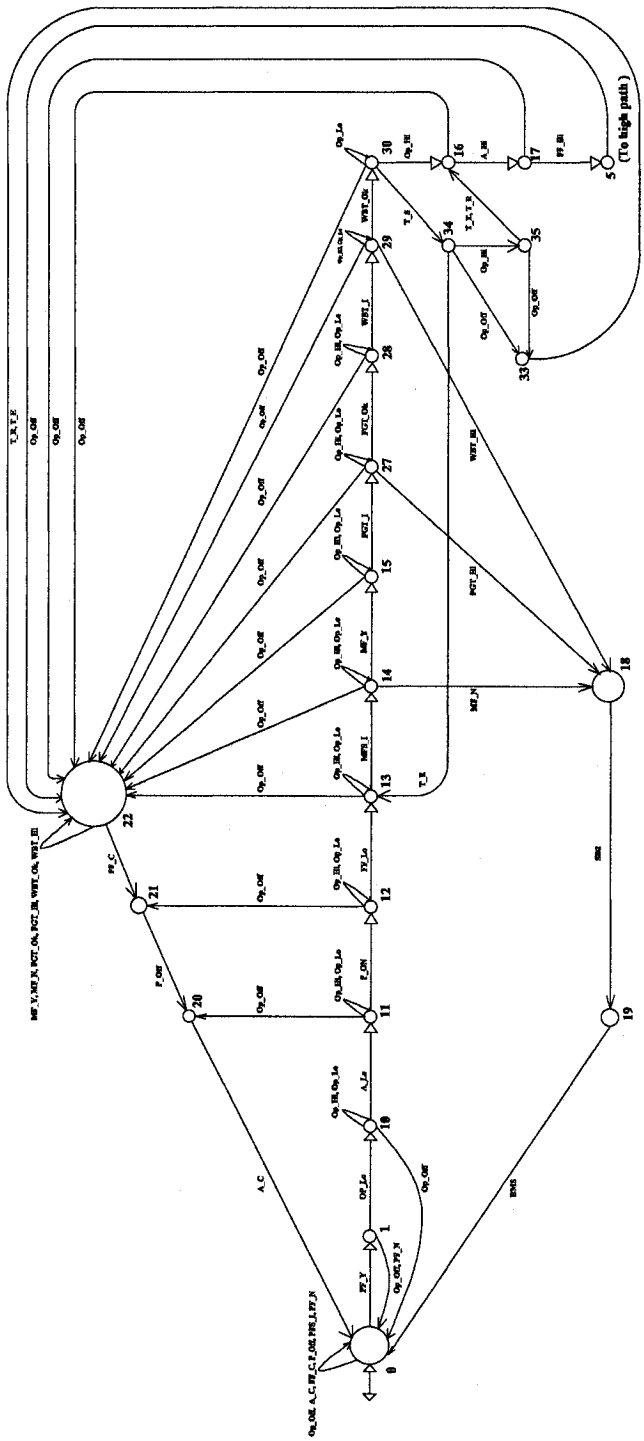


Figure 4.12: Specification for main flame supervisor: Low path (TCT: MAINSPEC1).

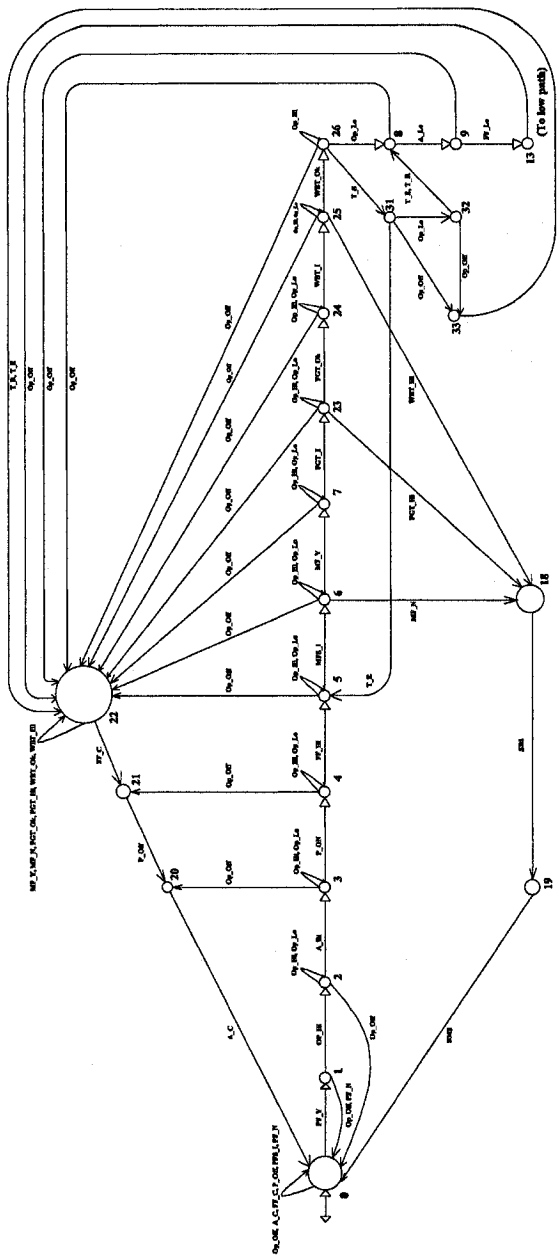


Figure 4.13: Specification for main flame supervisor: High path (TCT: MAINSPEC1).

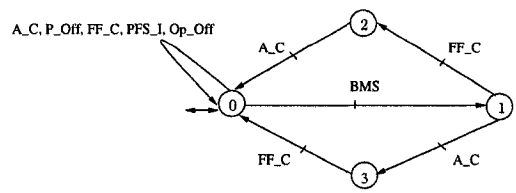


Figure 4.14: Specification for main flame supervisor (TCT: MAINSPEC2).

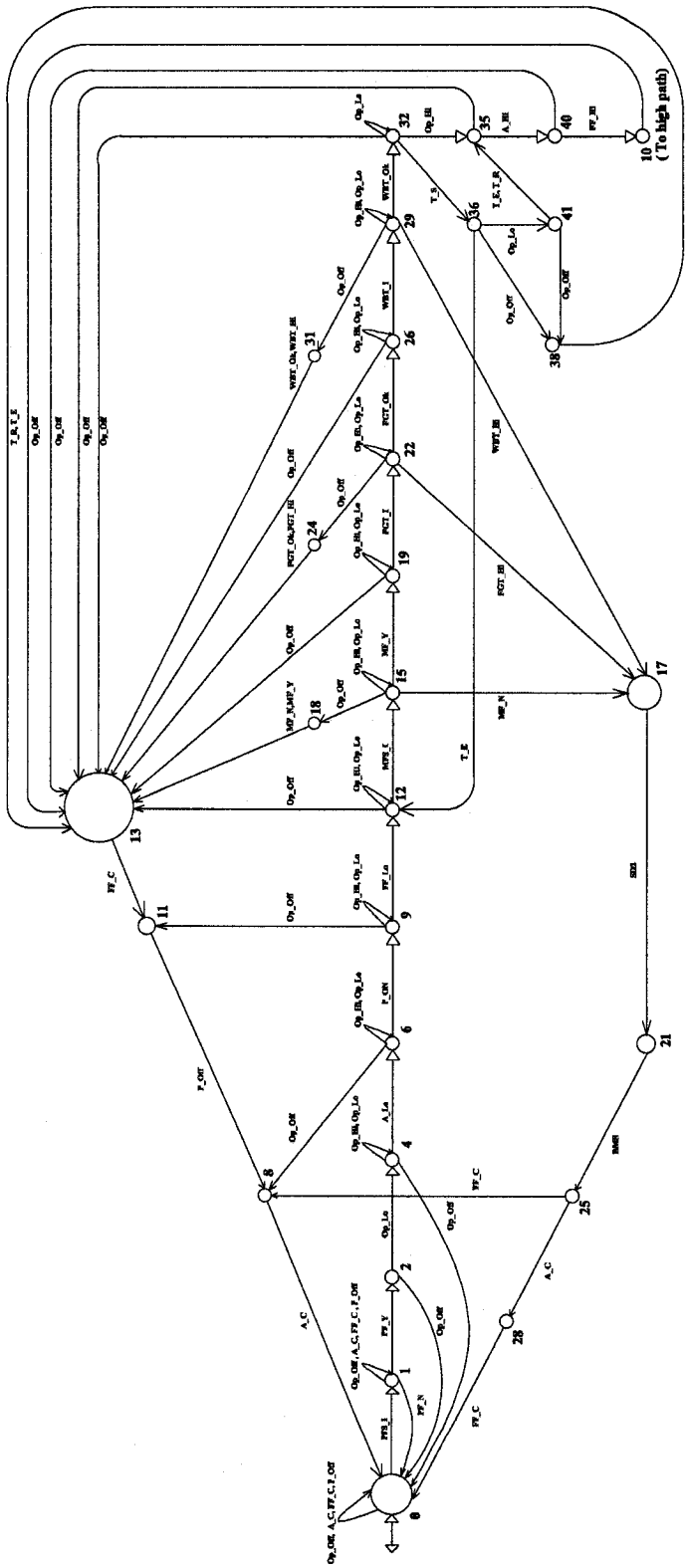


Figure 4.15: Supervisor for main flame: Low path.

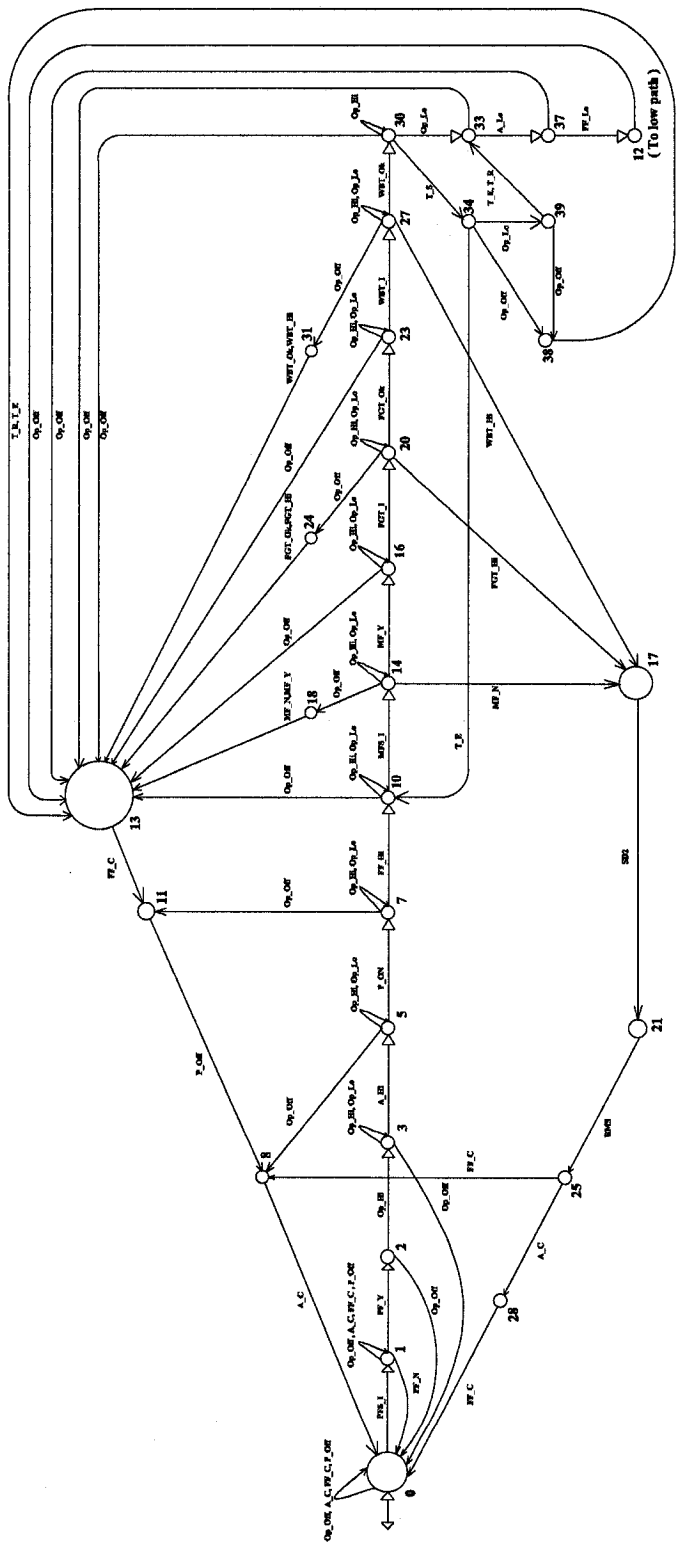


Figure 4.16: Supervisor for main flame: High path.

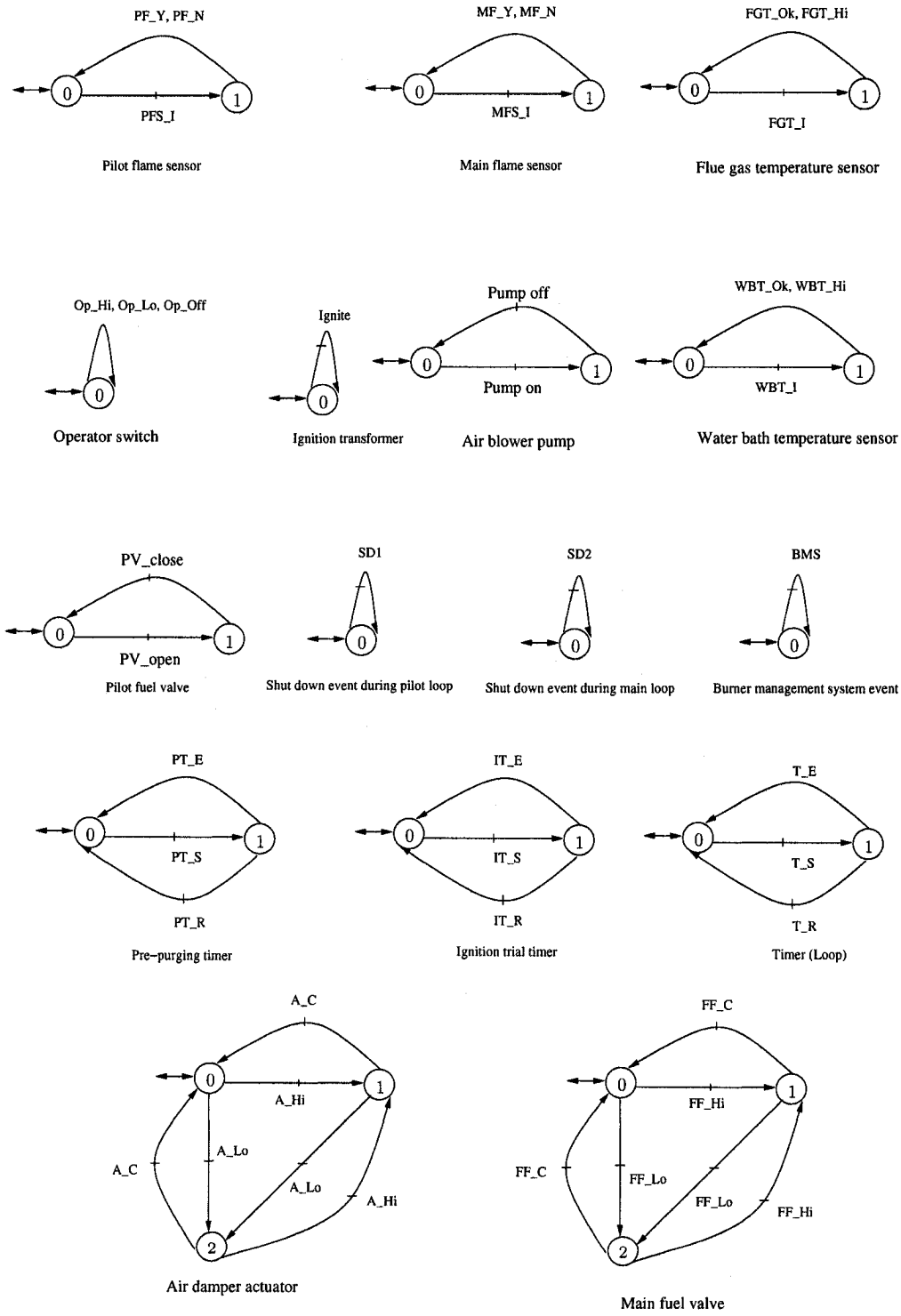


Figure 4.17: Complete boiler controller I/O devices.

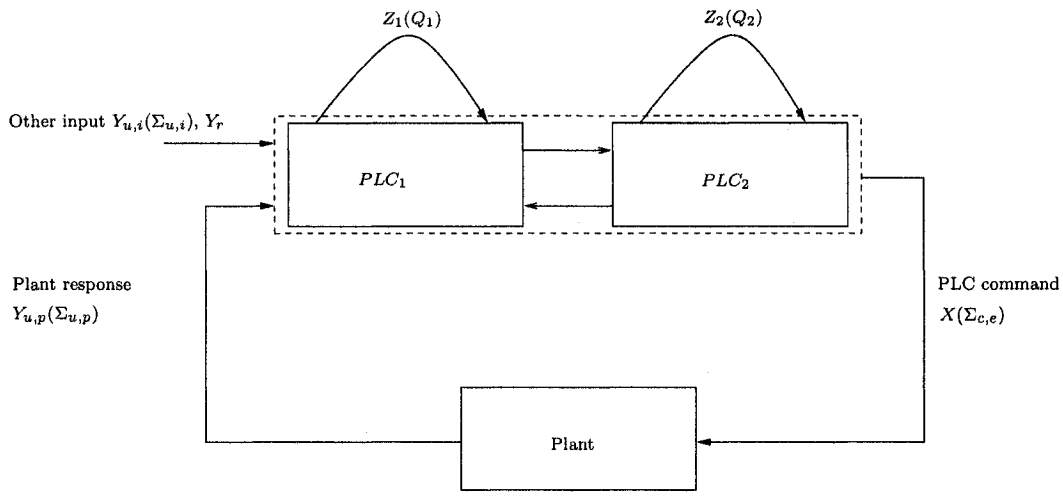


Figure 4.18: Complete boiler controller architecture.

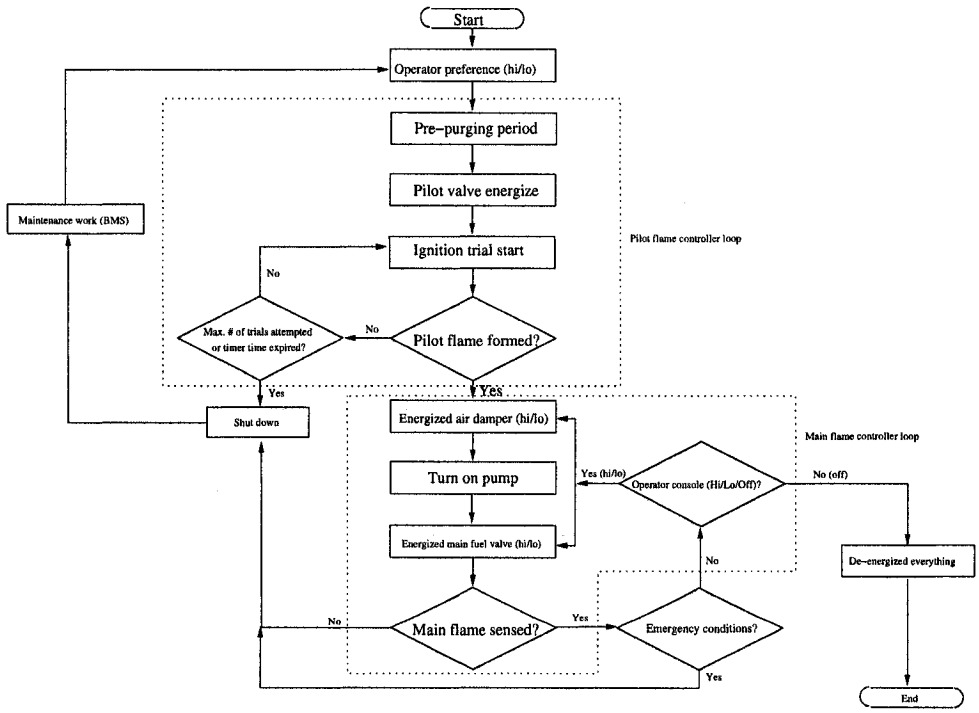


Figure 4.19: Typical operation sequence of boiler controller.

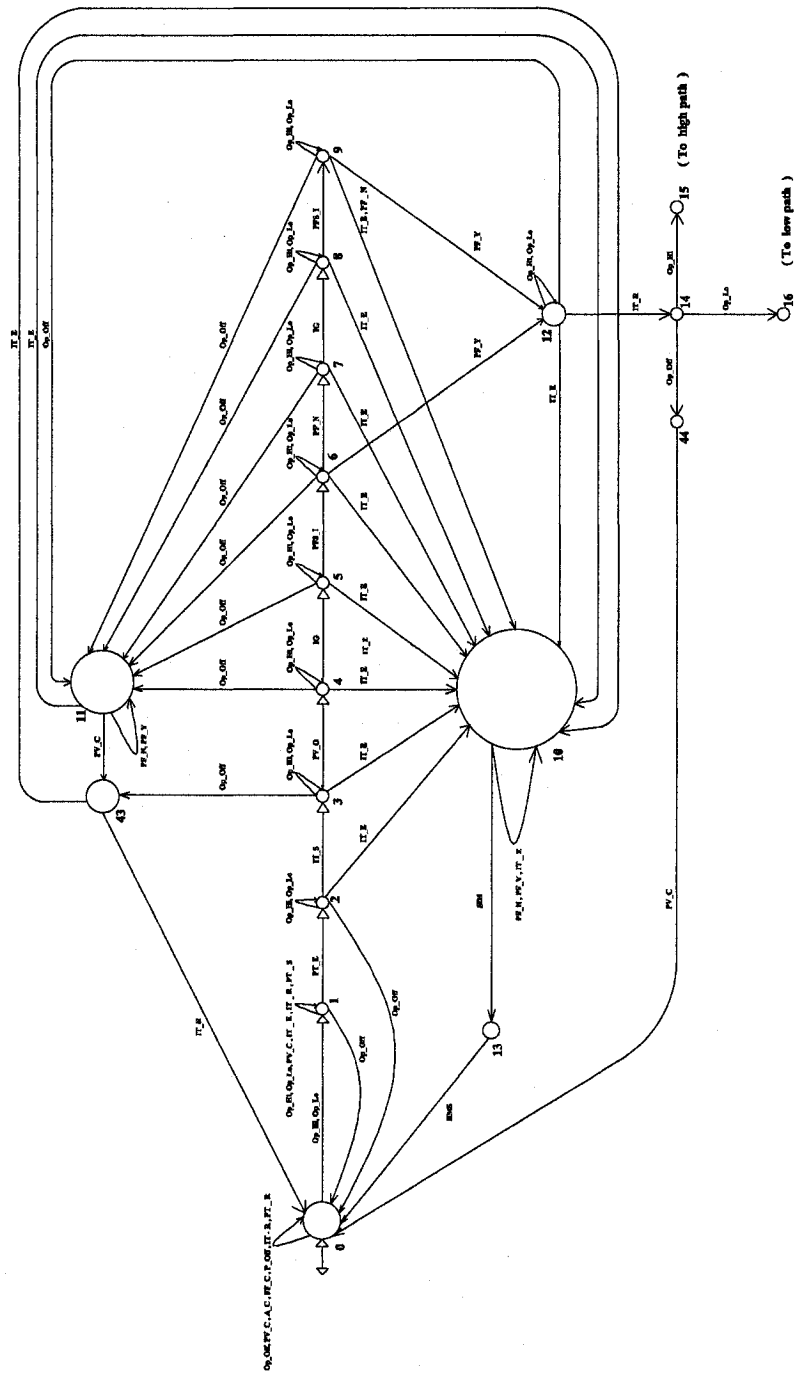


Figure 4.20: Specification for complete boiler supervisor: Pilot path (TCT: COMSPEC).

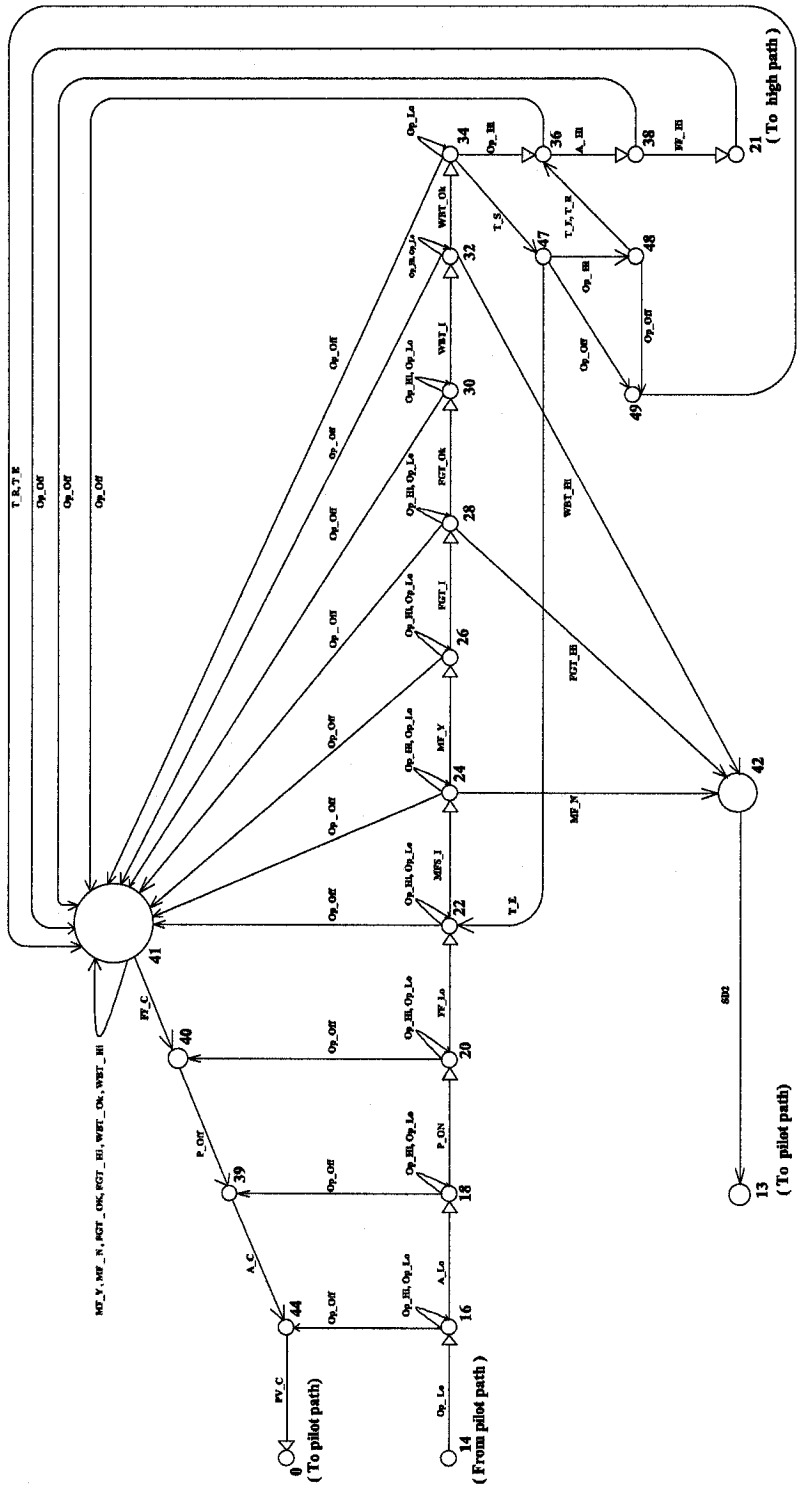


Figure 4.21: Specification for complete boiler supervisor: Low path (TCT: COMSPEC).

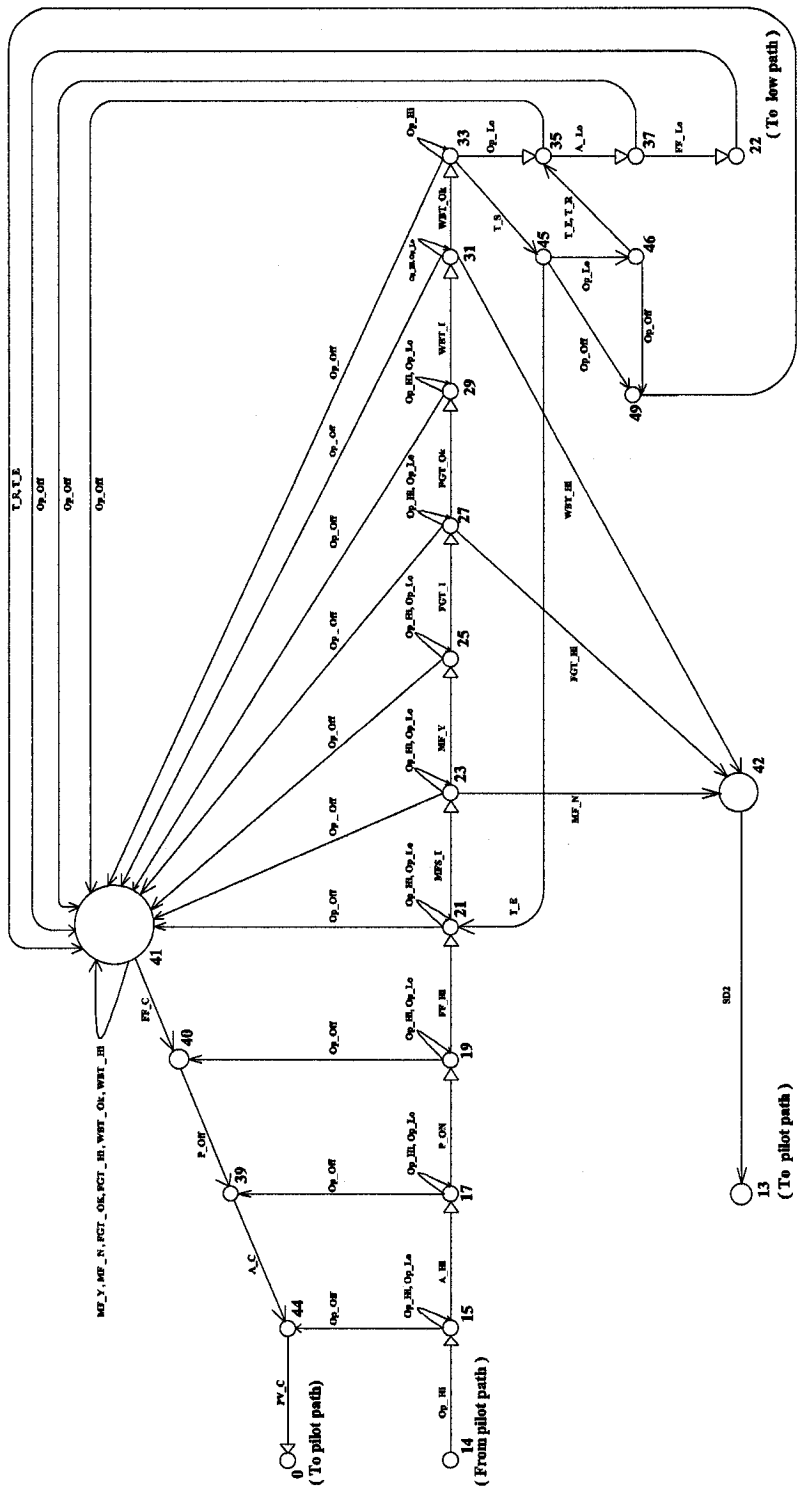


Figure 4.22: Specification for complete boiler supervisor: High path (TCT: COMSPEC).

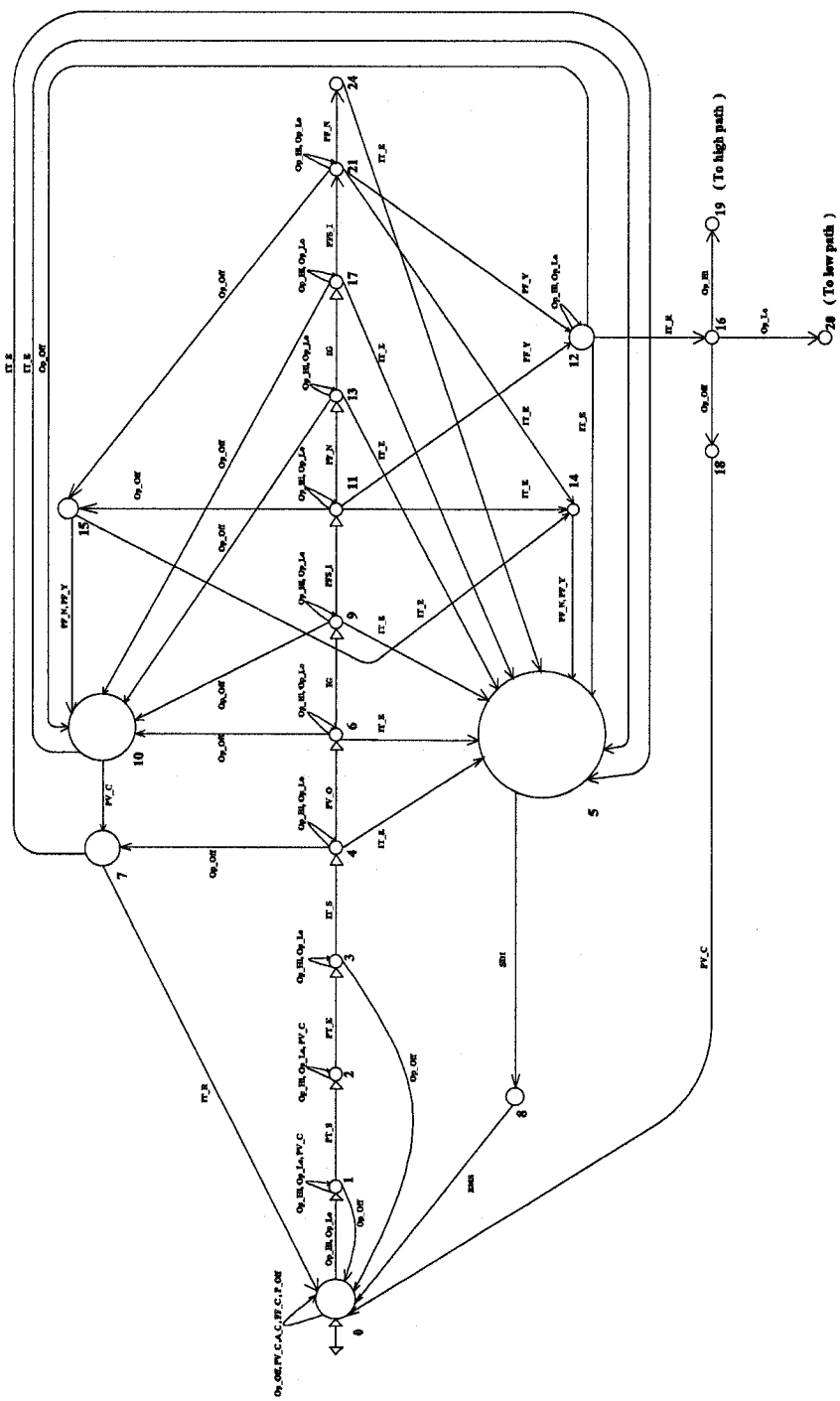


Figure 4.23: Complete boiler supervisor: Pilot path.

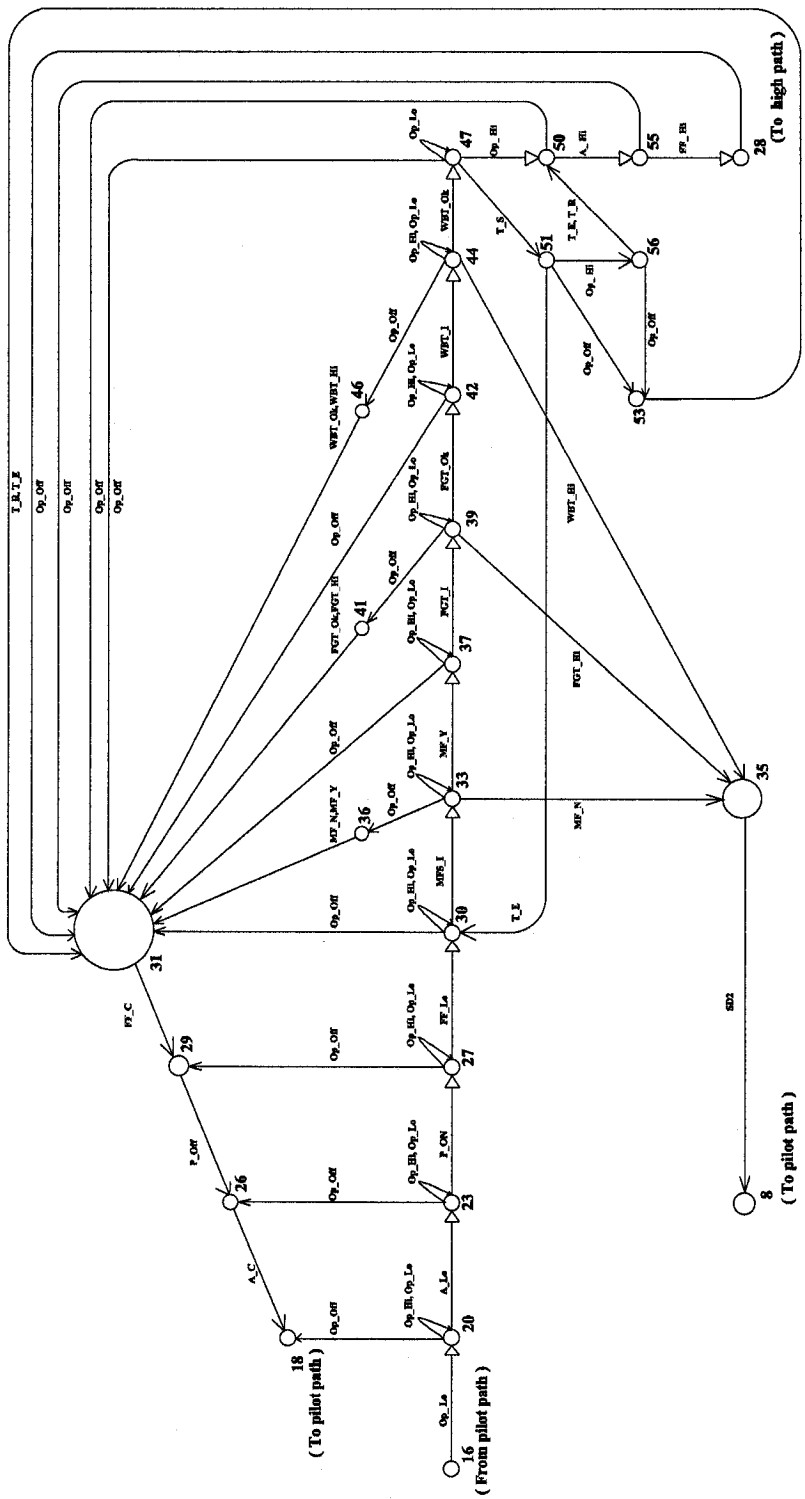


Figure 4.24: Complete boiler supervisor: Low path.

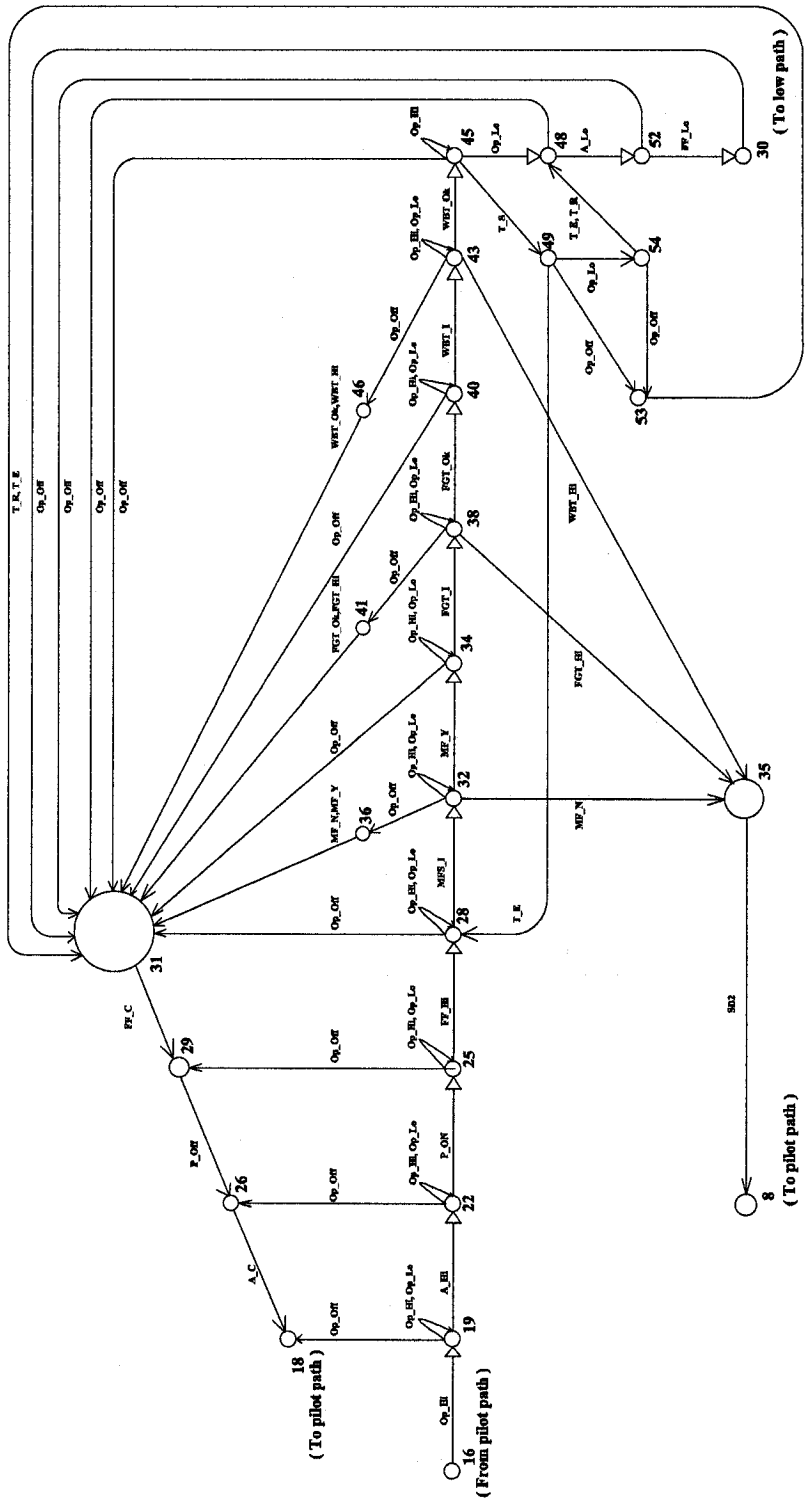


Figure 4.25: Complete boiler supervisor: High path.

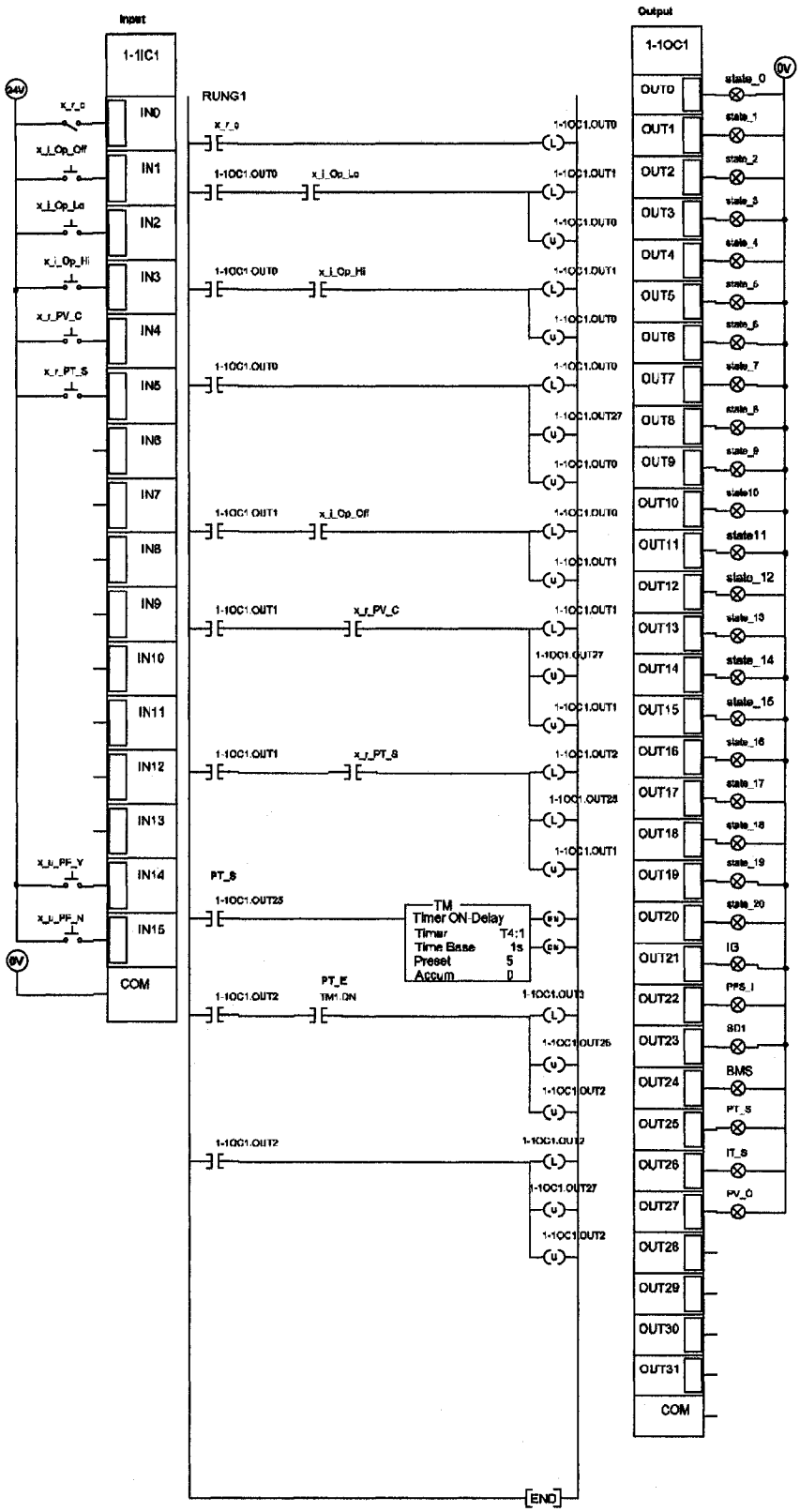


Figure 4.26: A sample LLD of Pilot flame controller.

Chapter 5

Conclusions and future research

5.1 Conclusions

In this thesis we present an algorithm to implement DES supervisors using PLCs, and we show how the user can implement converted LLDs either manually or automatically. After presenting a motivating example, the conversion method is introduced, and then we propose an overview of the implementation process in two steps.

First, we partition the event set into three event subsets: plant response event set, external event set and input event set. For a given DES supervisor uncontrollable event set is partitioned to plant response events, which are events that can be generated by the plant, and input events which are events that can be generated by other agents. DES supervisor controllable event set is considered as external events, which are events that can be generated by the PLC to drive output devices.

Second, a conversion method is introduced to convert the DES supervisor to an equivalent LLD by selecting PLC's inputs and outputs from among the system events. We assume that external controllable events can only appear in the form of $(n+1)$-state switches, which are then mapped to PLC output signals. We propose that external controllable events can be represented as PLC outputs using latch or unlatch coils, while plant response and input events can be represented as PLC inputs using normally open contacts. Finally we model a DES state transition function by a rung latch/unlatch function in LLD.

We distinguish PLC implementation of DES supervisor in manual and automatic mode. When two or more external controllable events are eligible to occur in a state of the plant, user needs to tell the PLC which events among them are to be

generated, otherwise PLC deterministically choose the one whose rung appears first in the ladder. We call this implementation in manual mode. When there is only one outgoing external controllable event, PLC can automatically generate the event, provided the plant response and input events (if there is any eligible to occur in the current state) are placed above the rung associated with the external controllable event. We call this implementation in automatic mode.

We discuss several key observations and assumptions about our conversion algorithm. The self-looped $(\Sigma_{u,i} \cup \Sigma_{u,p})$ events in the supervisor are omitted in the PLC implementation since they do not change the supervisor's state, and thus the set of enabled events at that state. The incorporation of this observation in the implementation has reduced the number of PLC inputs and outputs. We observe that it is quite important to add all necessary events in the supervisor to directly transform DES supervisors into LLDs.

Since PLC program is executed sequentially, several input and output events might occur in one scan cycle and there might be a possibility to skip over a number of events when several controllable events are to be generated in a state of the plant; we address this problem by suggesting that input variables can be set by the user manually to tell the PLC which events must be generated first among several controllable events.

To demonstrate the correctness of our approach, boiler control systems have been designed using supervisory control theory. In the steps of supervisor design using SCT, we tackle several issues that might arise in the real applications i.e modeling of input and output devices as DES. We design three controllers to achieve the boiler control objectives. Pilot flame controller is used for controlling the pilot flame which must be formed within two ignition trials or within the time allowed by the user setting in ignition trial timer. Main flame controller is responsible for controlling the main flame which starts its operation when it receives the appropriate signal from the pilot controller. By taking the serial composition of all controllers, we design a complete boiler controller integrating both pilot and main flame controllers which can be used to perform both pilot and main flame control objectives.

We then convert DES supervisors to LLDs using our conversion technique; the converted LLDs have been copied and simulated using PLC simulation software Automation Studio 5.2 [12]. We observe from the simulation test that our conversion

method results in LLDs that generate the same sequences of events as the supervisors, and thus they preserve the behavior of the DES supervisor. In another point of view, our work demonstrates how SCT can be used to design controllers for real field control problems and how these controllers can be implemented using PLCs.

5.2 Future research

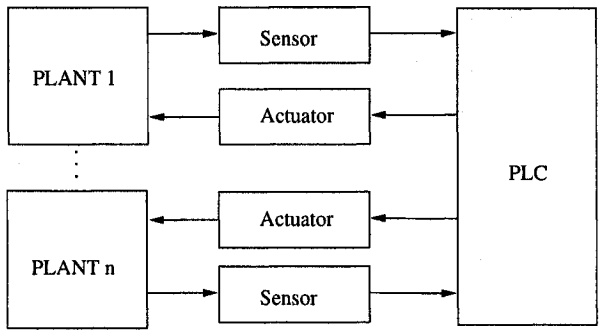
There are few directions in which our work can be extended. For future research, the following projects are significant and relevant to our work.

- It will be quite useful in industrial applications if a software can be developed which integrates the functionality of TCT [14] with our conversion algorithm. This integrated software can be used as a complete tool to design PLC controllers directly from DES plant and specification models.

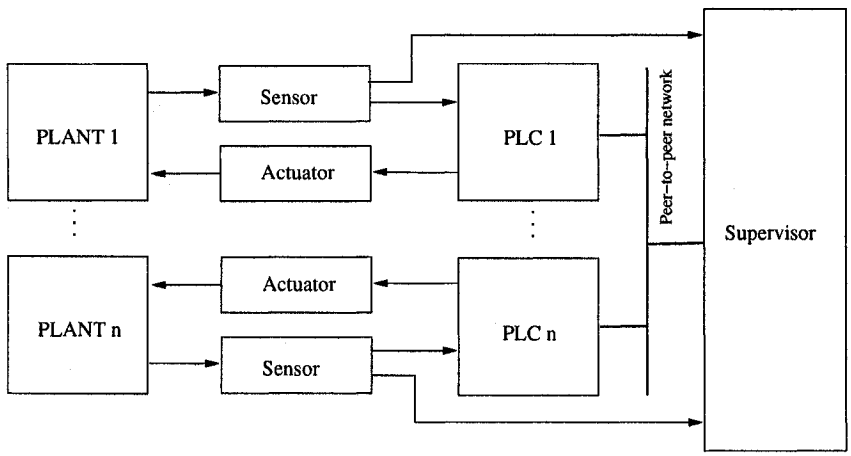
- We take the disjunction (serial) of two supervisors to develop a complete supervisor which is used to achieve the same control objective as a centralized supervisor. The development of a general theory for combining several supervisors serially is an interesting problem to investigate in future. This method can be used to implement several controllers using only one PLC.

- Nowadays Distributed Control Systems (DCS) [25], [26] are increasingly used to control large and complex processes. A DCS is comprised of a supervisory controller and one or more distributed (subordinate) controllers, which are used to control their associated plant components in the same process unit. The supervisory controller and distributed controllers are connected via a peer-to-peer network. Necessary data from distributed controllers are requested by the supervisor and based on the received data, control commands are generated and sent by the supervisor to the distributed controllers. The associated plant components (motor, valves, switches, etc.) are then controlled by their distributed controllers based on the supervisor commands. If we assume that supervisor and other distributed controllers in a DCS can be implemented by PLCs, it might be possible to develop a methodology to investigate how DES supervisors can be implemented in a DCS approach using PLCs. A view of PLC and DCS implementation is shown in Figure 5.1, where in a DCS implementation

one PLC controller can be used for controlling each plant component, and a supervisor can be used for coordinating all PLCs. A local PLC may not need help from the coordinator for all its decisions; in other words, some control decisions can be made locally and some by the help of coordinator.



a) A PLC implementation.



b) A DCS implementation.

Figure 5.1: A typical operation of PLC and DCS.

- It would be worthwhile to build a prototype boiler control system based on the results of this work. This prototype model can include other design criteria (adding variable motor speed drive with pump, flue gas oxygen analyzer with stack, etc.) and emergency conditions (water level low, no fuel flow, etc.).
- In our implementation, we ignore the effect of plant response delay. To enhance our algorithm it can be taken as a future work to develop a methodology where such a response delay may be possible to handle.

Bibliography

[1] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[2] J. Stenerson, *Fundamentals of programmable logic controllers, sensors, and communications.* Prentice Hall, 1993.

[3] J. W. Webb and R. A. Reis, *Programmable logic controllers: principles and applications*, 5th ed. Prentice Hall, 2003.

[4] F. D. Petruzella, *Programmable logic controllers*, 2nd ed. McGraw-Hill, 1997.

[5] B. A. Brandin, "The real-time supervisory control of an experimental manufacturing cell," *IEEE Trans. Robotics and Automat*, vol. 12, No. 1, pp. 1–14, February 1996.

[6] M. Fabian and A. Hellgren, "PLC based implementation of supervisory control for discrete event systems," *Proceedings of the 37th IEEE conference on Decision and Control, Tampa, Florida, USA*, vol. 3, pp. 3305 – 3310, Dec. 1998.

[7] J. Liu and H. Darabi, "Ladder logic implementation of Ramadge-Wonham supervisory controller," *Proceedings of the Sixth International Workshop on Discrete-event systems (WODES02), IEEE*, 2002.

[8] R. Leduc and W. Wonham, "PLC implementation of a DES supervisor for a manufacturing testbed," *Proc. of Thirty-third Annual Allerton Conference on Communication, Control, and Computing*, pp. 519–528, Oct. 1995.

[9] W. Wonham, *Supervisory control of Discrete-Event Systems.* Systems control group, University of Toronto, 2006.

[10] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems.* Kluwer, 1999.

[11] S. Balemi, "Control of discrete event systems: theory and application," *Ph.D thesis, SUPSI-DTI, Switzerland*, pp. 25–31, 1992.

[12] "Automation studio 5.2, Educational edition," http://www.automationstudio.com/, 2006.

[13] R. Leduc and W. Wonham, "Discrete-event systems modeling and control of a manufacturing testbed," *Proc. of Canadian Conference on Electrical and Computer Engineering*, vol. II, pp. 793–796, Sep. 1995.

[14] "Design software: XPTCT," W. M. Wonham's homepage, http://www.control.utoronto.ca/~wonham/, updated July 2006.

[15] "Programmable logic controller, boiler," http://www.answers.com/, 2006.

[16] M. Wood, "Application, implementation and integration of DES control theory," *M.A.Sc. thesis, Queen's University, Canada*, pp. 26–29, 2005.

[17] M. A. Malek, *Power boiler design, inspection, and repair.* McGraw-Hill, 2005.

[18] S. G. Dukelow, *The Control of Boilers*, 2nd ed. Instrument Society of America, 1991.

[19] "G779 universal replacement intermittent pilot ignition control," http://www.johnsoncontrols.com/cg-heating/ignition.htm, 2006.

[20] "Burner and boiler control," http://europe.hbc.honeywell.com/products, 2006.

[21] "Burner, BGN 50," http://www.baltur.it/en/home.php, 2006.

[22] R. Leduc, "PLC implementation of a DES supervisor for a manufacturing testbed: an implementation perspective," *M.A.Sc. thesis, University of Toronto, Canada*, pp. 13–19, 1996.

[23] S. Balemi and U. Brunner, "Supervision of discrete event systems with communication," *Proc. of American Control Conference*, pp. 2794–2798, June 1992.

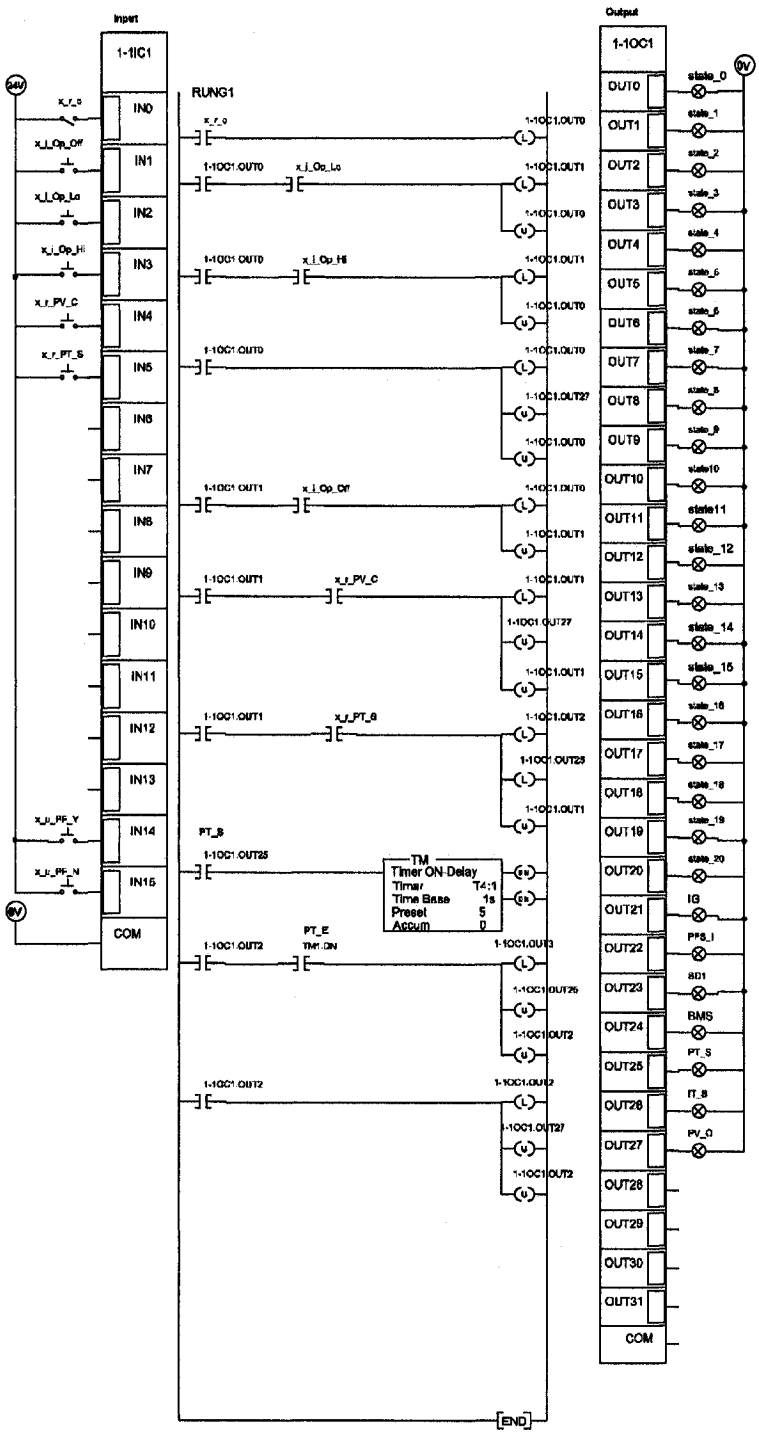
[24] "Water bath heater," http://www.natcogroup.com/, 2006.

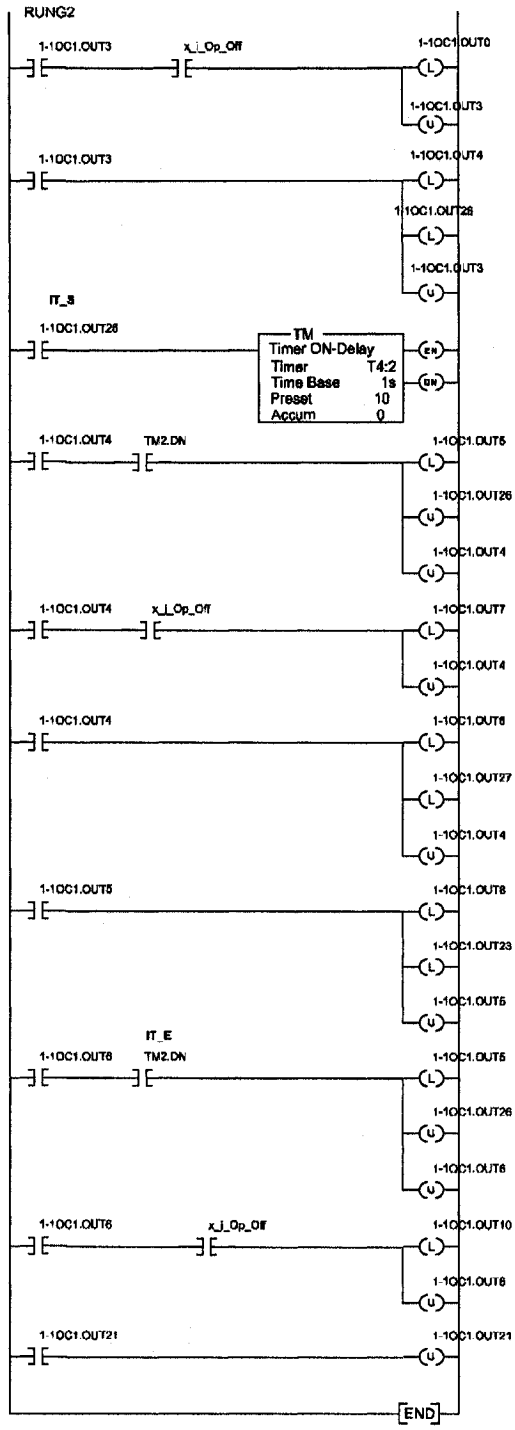
[25] N. Mahalik, *Fieldbus Technology.* Springer, 2003.

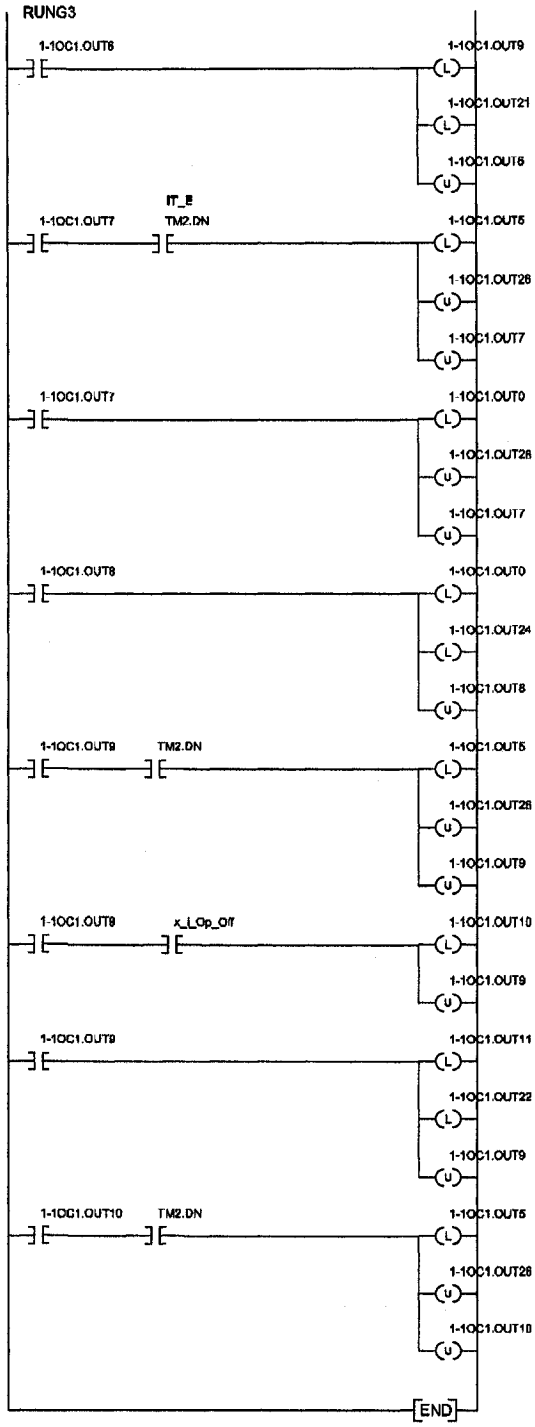
[26] "The foxboro A^2 automation system," http://www.foxboro.com/, 2006.

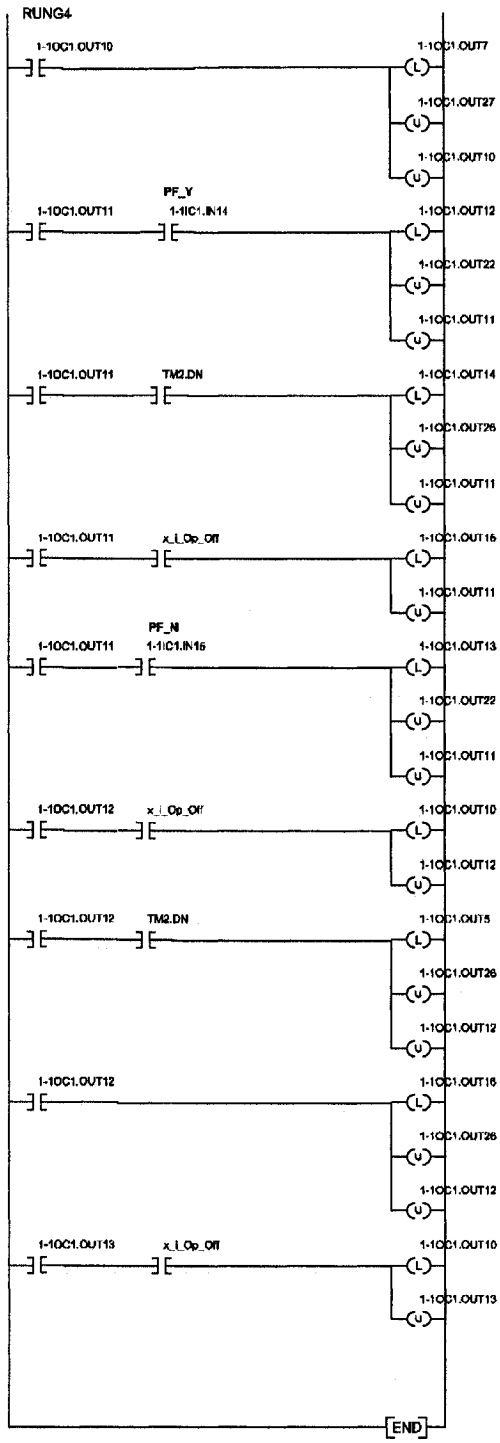
Appendix 1

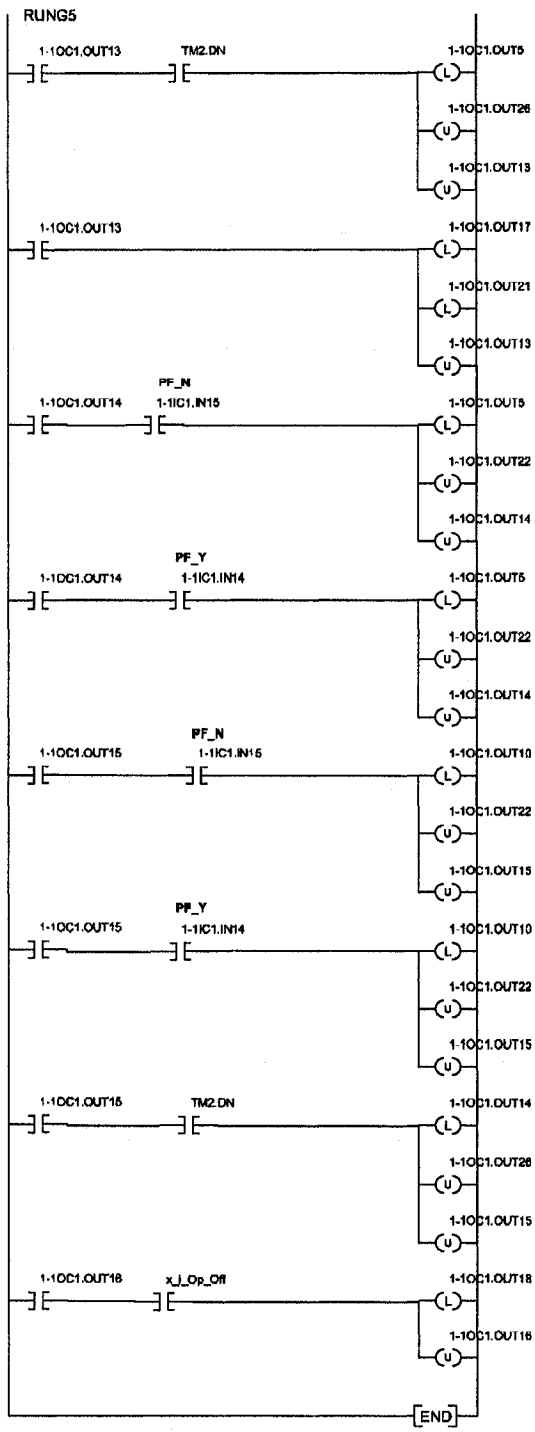
PLC-based implementation of the pilot flame controller

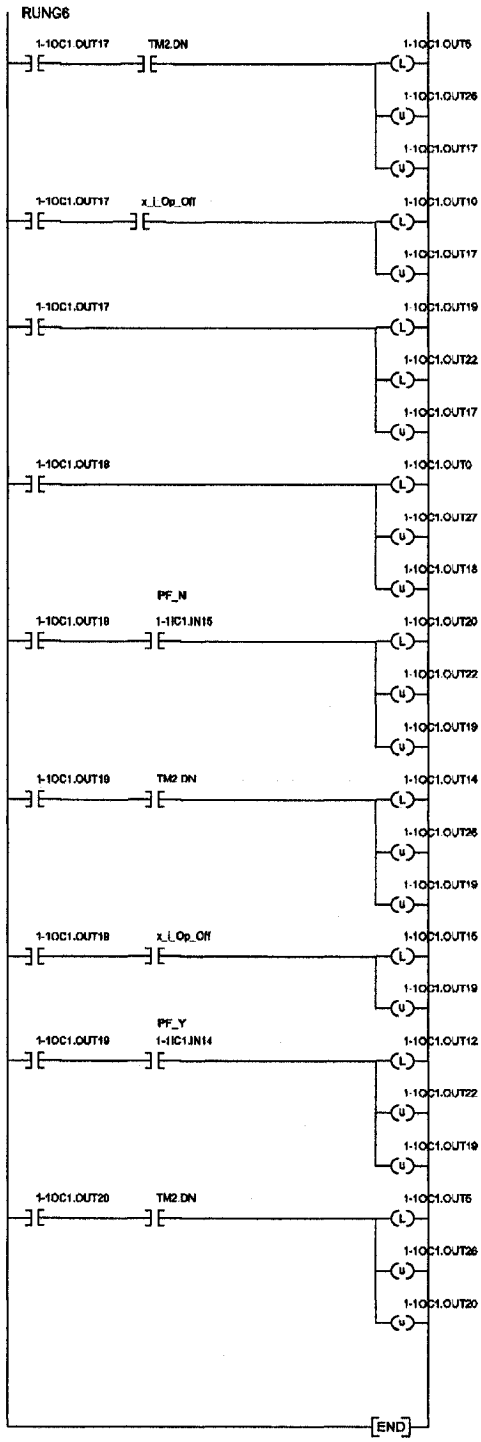

Input
1-1IC1
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
IN8
IN9
IN10
IN11
IN12
IN13
IN14
IN15
COM
RUNG1
TM
Timer ON Delay
Timer T4:1
Time Base 1s
Preset 5
Accum 0
END
Output
1-1OC1
OUT0
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
OUT8
OUT9
OUT10
OUT11
OUT12
OUT13
OUT14
OUT15
OUT16
OUT17
OUT18
OUT19
OUT20
OUT21
OUT22
OUT23
OUT24
OUT25
OUT26
OUT27
OUT28
OUT29
OUT30
OUT31
COM


RUNG2
1-10C1.OUT3
x_i_Op_Off
1-10C1.OUT0
1-10C1.OUT3
1-10C1.OUT3
1-10C1.OUT4
1-10C1.OUT26
1-10C1.OUT3
IT_S
1-10C1.OUT26
TM
Timer ON-Delay
Timer T4:2
Time Base 1s
Preset 10
Accum 0
1-10C1.OUT4
TM2.DN
1-10C1.OUT5
1-10C1.OUT26
1-10C1.OUT4
1-10C1.OUT4
x_i_Op_Off
1-10C1.OUT7
1-10C1.OUT4
1-10C1.OUT4
1-10C1.OUT8
1-10C1.OUT27
1-10C1.OUT4
1-10C1.OUT5
1-10C1.OUT8
1-10C1.OUT23
1-10C1.OUT5
IT_E
1-10C1.OUT8
TM2.DN
1-10C1.OUT5
1-10C1.OUT26
1-10C1.OUT8
1-10C1.OUT6
x_i_Op_Off
1-10C1.OUT10
1-10C1.OUT8
1-10C1.OUT21
1-10C1.OUT21
END


RUNG3
1-1OC1.OUT8
1-1OC1.OUT9
1-1OC1.OUT21
1-1OC1.OUT6
IT_E
1-1OC1.OUT7
TM2.DN
1-1OC1.OUT5
1-1OC1.OUT26
1-1OC1.OUT7
1-1OC1.OUT7
1-1OC1.OUT0
1-1OC1.OUT26
1-1OC1.OUT7
1-1OC1.OUT8
1-1OC1.OUT0
1-1OC1.OUT24
1-1OC1.OUT8
1-1OC1.OUT9
TM2.DN
1-1OC1.OUT5
1-1OC1.OUT26
1-1OC1.OUT9
1-1OC1.OUT8
x_LOp_Off
1-1OC1.OUT10
1-1OC1.OUT9
1-1OC1.OUT9
1-1OC1.OUT11
1-1OC1.OUT22
1-1OC1.OUT9
1-1OC1.OUT10
TM2.DN
1-1OC1.OUT5
1-1OC1.OUT26
1-1OC1.OUT10
END

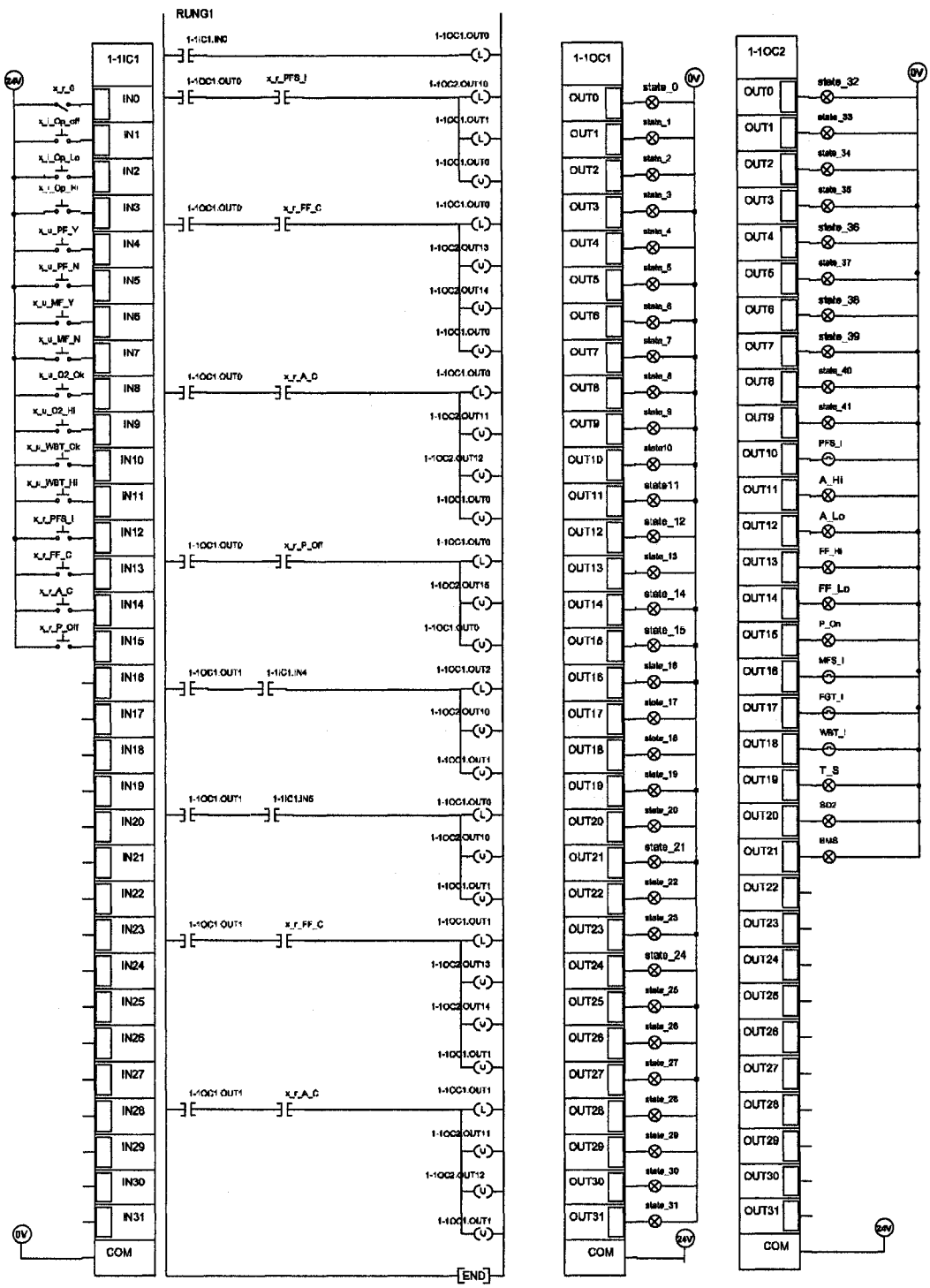

RUNG4
1-10C1.OUT10
1-10C1.OUT7
1-10C1.OUT27
1-10C1.OUT10
PF_Y
1-10C1.OUT11
1-1IC1.IN14
1-10C1.OUT12
1-10C1.OUT22
1-10C1.OUT11
1-10C1.OUT11
TM2.DN
1-10C1.OUT14
1-10C1.OUT26
1-10C1.OUT11
1-10C1.OUT11
x_L_Op_Off
1-10C1.OUT16
1-10C1.OUT11
PF_N
1-10C1.OUT11
1-1IC1.IN16
1-10C1.OUT13
1-10C1.OUT22
1-10C1.OUT11
1-10C1.OUT12
x_L_Op_Off
1-10C1.OUT10
1-10C1.OUT12
1-10C1.OUT12
TM2.DN
1-10C1.OUT5
1-10C1.OUT26
1-10C1.OUT12
1-10C1.OUT12
1-10C1.OUT16
1-10C1.OUT26
1-10C1.OUT12
1-10C1.OUT13
x_L_Op_Off
1-10C1.OUT10
1-10C1.OUT13
END

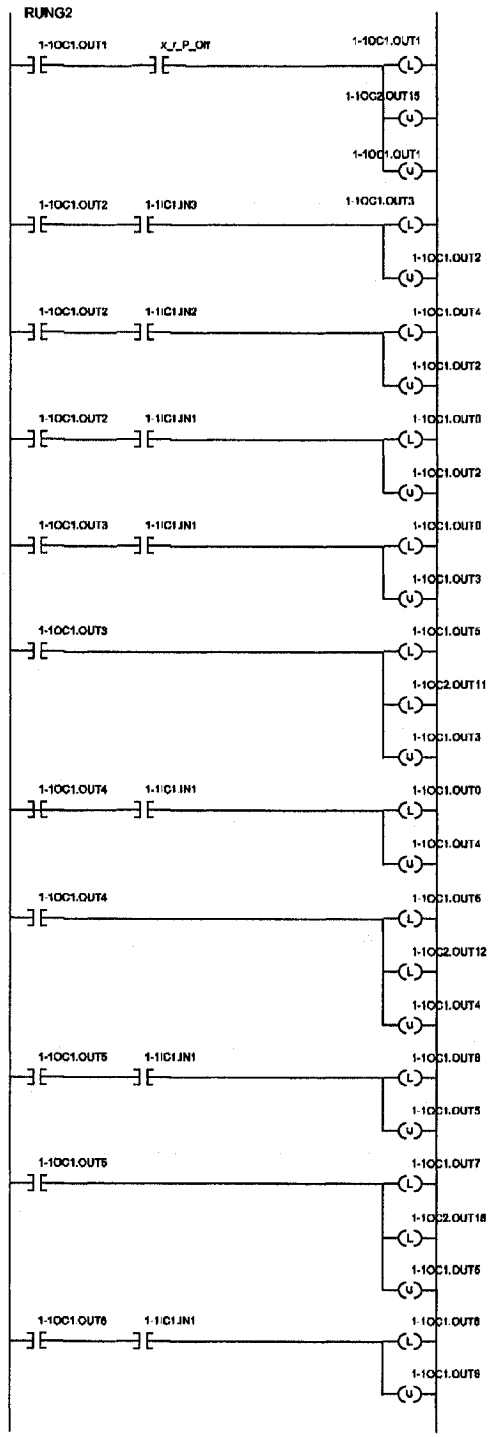

RUNG5
1-1OC1.OUT13
TM2.DN
1-1OC1.OUT5
1-1OC1.OUT26
1-1OC1.OUT13
1-1OC1.OUT13
1-1OC1.OUT17
1-1OC1.OUT21
1-1OC1.OUT13
PF_N
1-1OC1.OUT14
1-1IC1.IN15
1-1OC1.OUT5
1-1OC1.OUT22
1-1OC1.OUT14
PF_Y
1-1OC1.OUT14
1-1IC1.IN14
1-1OC1.OUT5
1-1OC1.OUT22
1-1OC1.OUT14
PF_N
1-1OC1.OUT15
1-1IC1.IN15
1-1OC1.OUT10
1-1OC1.OUT22
1-1OC1.OUT15
PF_Y
1-1OC1.OUT15
1-1IC1.IN14
1-1OC1.OUT10
1-1OC1.OUT22
1-1OC1.OUT15
1-1OC1.OUT15
TM2.DN
1-1OC1.OUT14
1-1OC1.OUT26
1-1OC1.OUT15
1-1OC1.OUT16
x_I_Op_Off
1-1OC1.OUT16
1-1OC1.OUT16
END

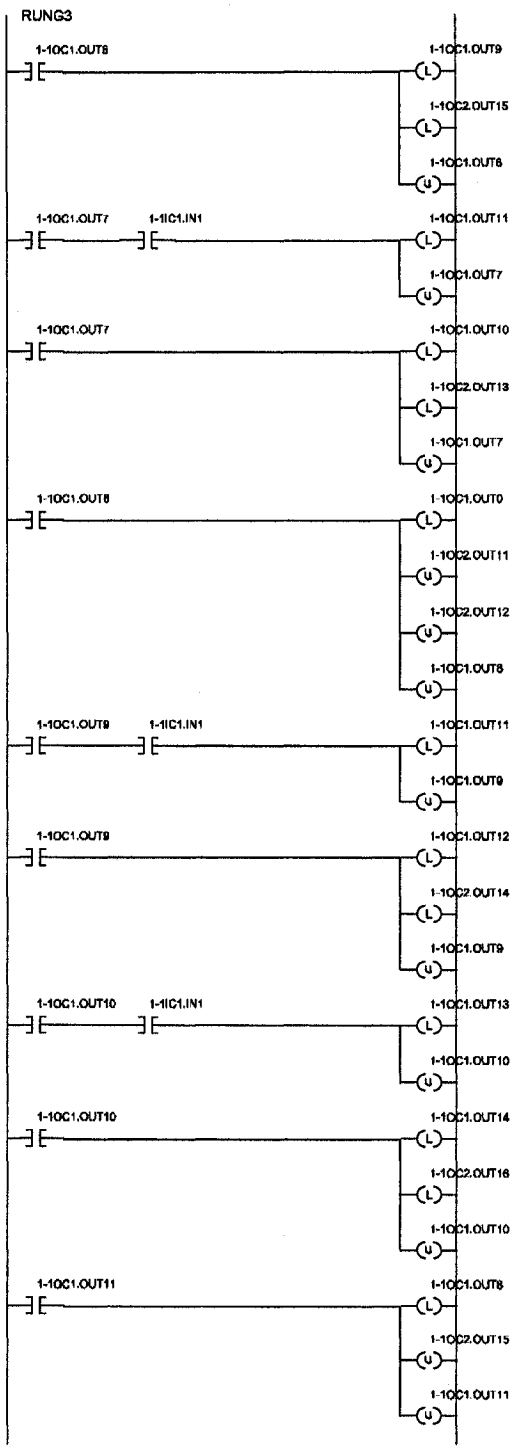

RUNG6
1-1OC1.OUT17
TM2.DN
1-1OC1.OUT5
1-1OC1.OUT26
1-1OC1.OUT17
1-1OC1.OUT17
x_I_Op_Off
1-1OC1.OUT10
1-1OC1.OUT17
1-1OC1.OUT17
1-1OC1.OUT19
1-1OC1.OUT22
1-1OC1.OUT17
1-1OC1.OUT18
1-1OC1.OUT0
1-1OC1.OUT27
1-1OC1.OUT18
PF_N
1-1OC1.OUT18
1-1IC1.IN15
1-1OC1.OUT20
1-1OC1.OUT22
1-1OC1.OUT19
1-1OC1.OUT19
TM2.DN
1-1OC1.OUT14
1-1OC1.OUT26
1-1OC1.OUT19
1-1OC1.OUT19
x_I_Op_Off
1-1OC1.OUT15
1-1OC1.OUT19
PF_Y
1-1OC1.OUT19
1-1IC1.IN14
1-1OC1.OUT12
1-1OC1.OUT22
1-1OC1.OUT19
1-1OC1.OUT20
TM2.DN
1-1OC1.OUT5
1-1OC1.OUT26
1-1OC1.OUT20
END

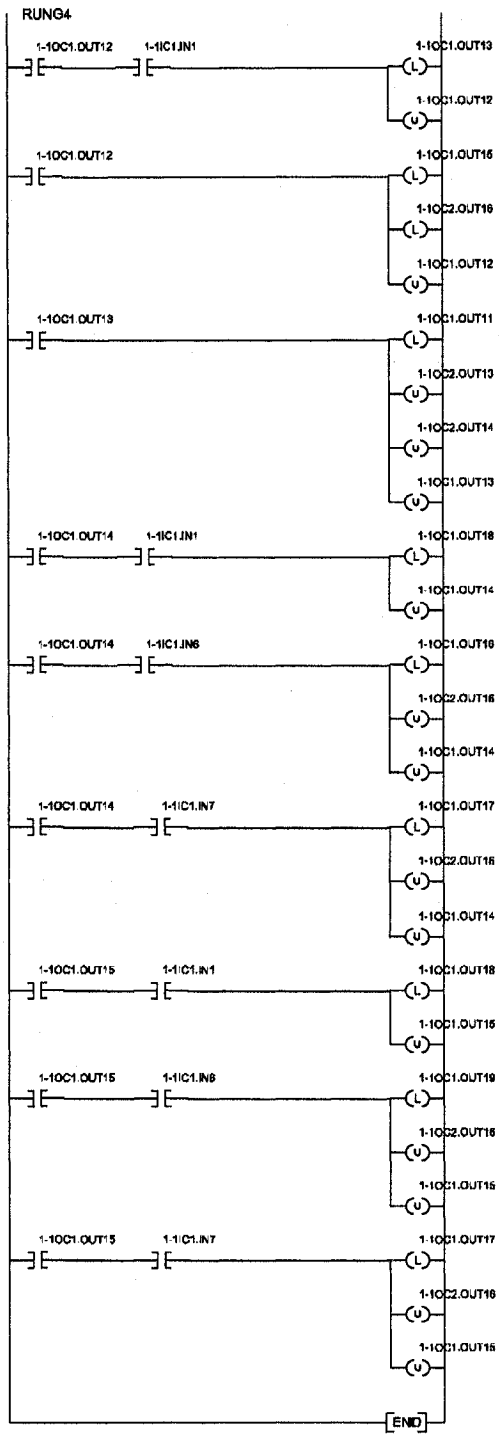
Appendix II

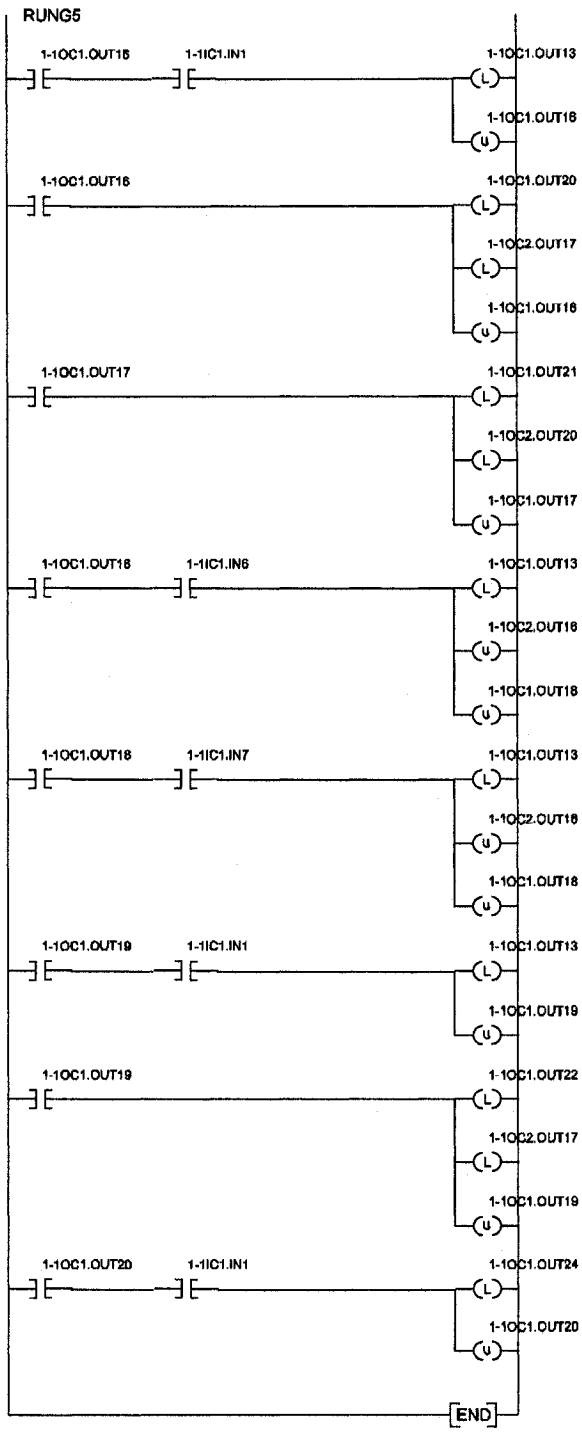
PLC-based implementation of the main flame controller

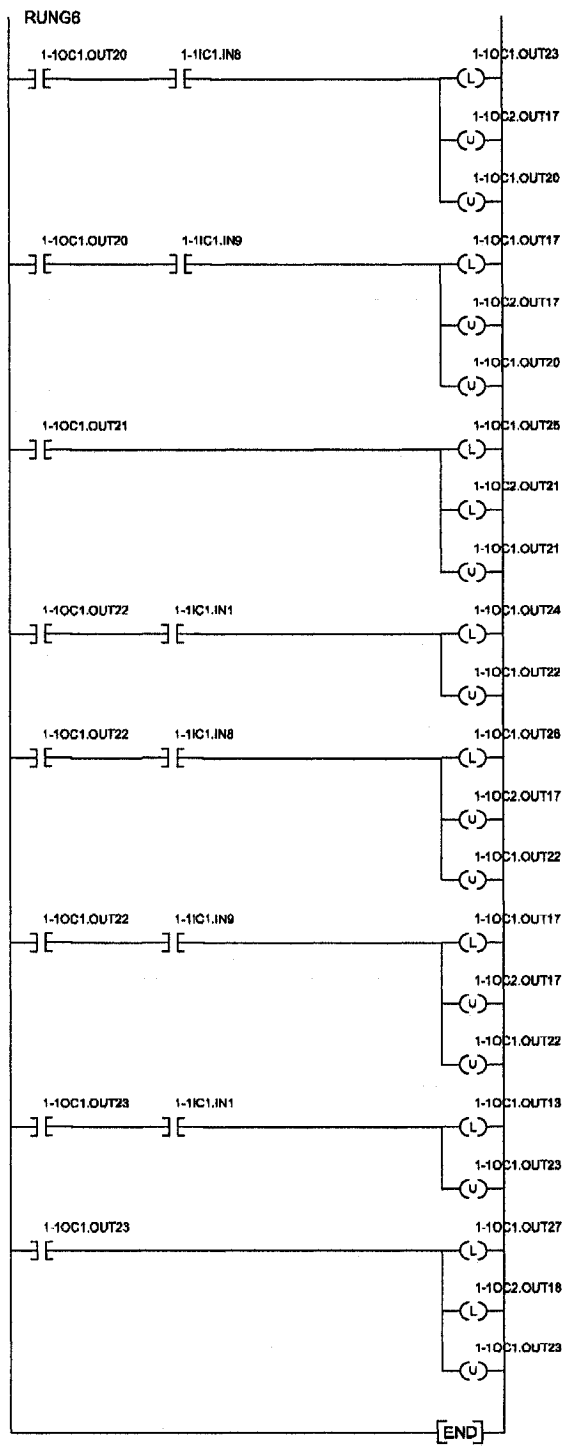

RUNG1
1-1IC1
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
IN8
IN9
IN10
IN11
IN12
IN13
IN14
IN15
IN16
IN17
IN18
IN19
IN20
IN21
IN22
IN23
IN24
IN25
IN26
IN27
IN28
IN29
IN30
IN31
COM
1-1OC1
OUT0
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
OUT8
OUT9
OUT10
OUT11
OUT12
OUT13
OUT14
OUT15
OUT16
OUT17
OUT18
OUT19
OUT20
OUT21
OUT22
OUT23
OUT24
OUT25
OUT26
OUT27
OUT28
OUT29
OUT30
OUT31
COM
1-1OC2
OUT0
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
OUT8
OUT9
OUT10
OUT11
OUT12
OUT13
OUT14
OUT15
OUT16
OUT17
OUT18
OUT19
OUT20
OUT21
OUT22
OUT23
OUT24
OUT25
OUT26
OUT27
OUT28
OUT29
OUT30
OUT31
COM
[END]

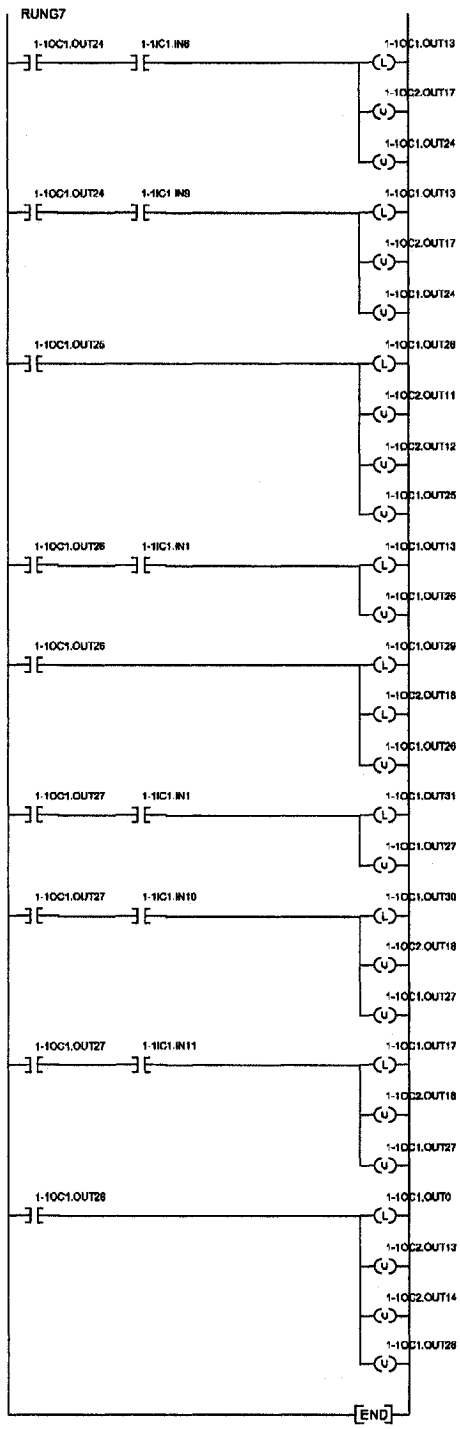

RUNG2
1-1OC1.OUT1
X_r_P_Off
1-1OC1.OUT1
1-1OC2.OUT15
1-1OC1.OUT1
1-1OC1.OUT2
1-1IC1.IN3
1-1OC1.OUT3
1-1OC1.OUT2
1-1OC1.OUT2
1-1IC1.IN2
1-1OC1.OUT4
1-1OC1.OUT2
1-1OC1.OUT2
1-1IC1.IN1
1-1OC1.OUT0
1-1OC1.OUT2
1-1OC1.OUT3
1-1IC1.IN1
1-1OC1.OUT0
1-1OC1.OUT3
1-1OC1.OUT5
1-1OC2.OUT11
1-1OC1.OUT3
1-1OC1.OUT4
1-1IC1.IN1
1-1OC1.OUT0
1-1OC1.OUT4
1-1OC1.OUT4
1-1OC1.OUT6
1-1OC2.OUT12
1-1OC1.OUT4
1-1OC1.OUT5
1-1IC1.IN1
1-1OC1.OUT8
1-1OC1.OUT5
1-1OC1.OUT5
1-1OC1.OUT7
1-1OC2.OUT16
1-1OC1.OUT5
1-1OC1.OUT6
1-1IC1.IN1
1-1OC1.OUT8
1-1OC1.OUT6

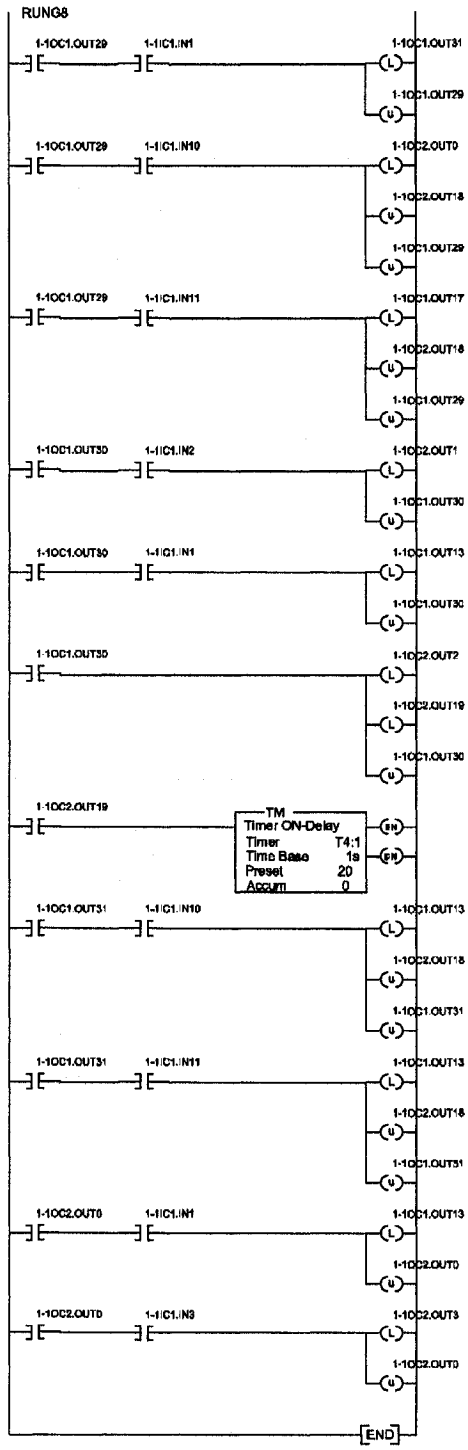

RUNG3
1-10C1.OUT8
1-10C1.OUT9
1-10C2.OUT15
1-10C1.OUT8
1-10C1.OUT7
1-1IC1.IN1
1-10C1.OUT11
1-10C1.OUT7
1-10C1.OUT7
1-10C1.OUT10
1-10C2.OUT13
1-10C1.OUT7
1-10C1.OUT8
1-10C1.OUT0
1-10C2.OUT11
1-10C2.OUT12
1-10C1.OUT8
1-10C1.OUT9
1-1IC1.IN1
1-10C1.OUT11
1-10C1.OUT9
1-10C1.OUT9
1-10C1.OUT12
1-10C2.OUT14
1-10C1.OUT9
1-10C1.OUT10
1-1IC1.IN1
1-10C1.OUT13
1-10C1.OUT10
1-10C1.OUT10
1-10C1.OUT14
1-10C2.OUT16
1-10C1.OUT10
1-10C1.OUT11
1-10C1.OUT8
1-10C2.OUT15
1-10C1.OUT11

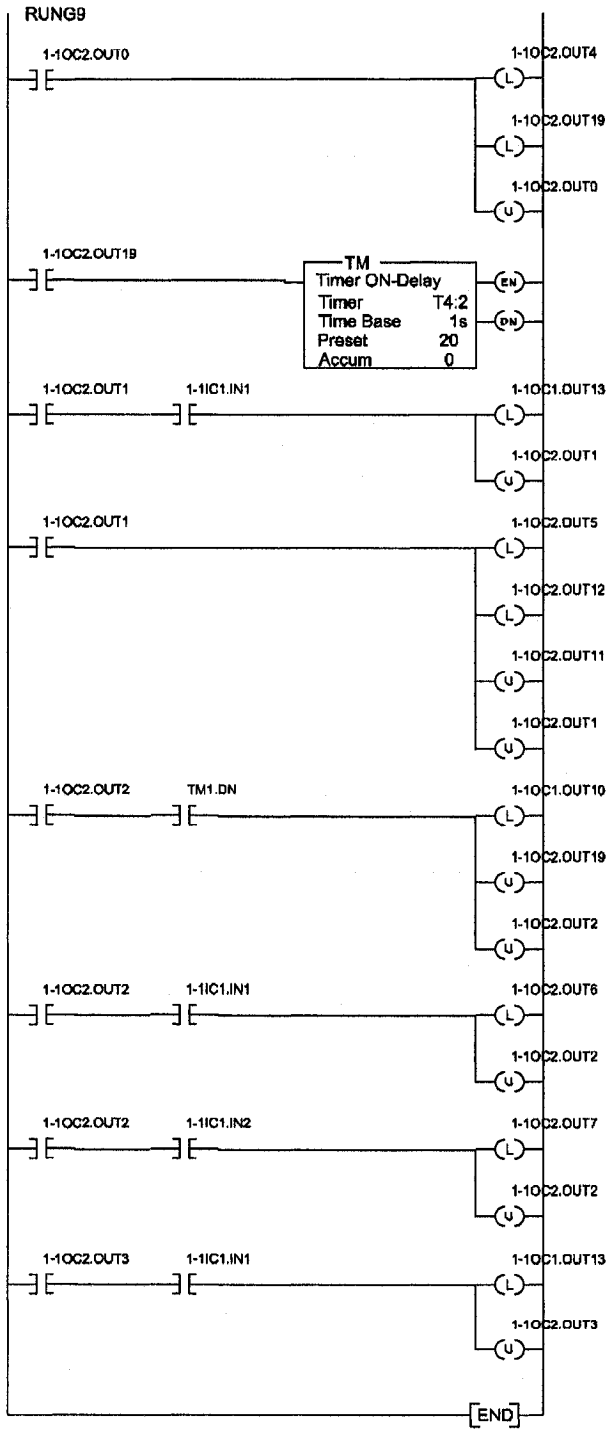

RUNG4
1-10C1.OUT12
1-1IC1.IN1
1-10C1.OUT13
1-10C1.OUT12
1-10C1.OUT12
1-10C1.OUT16
1-10C2.OUT16
1-10C1.OUT12
1-10C1.OUT13
1-10C1.OUT11
1-10C2.OUT13
1-10C2.OUT14
1-10C1.OUT13
1-10C1.OUT14
1-1IC1.IN1
1-10C1.OUT18
1-10C1.OUT14
1-10C1.OUT14
1-1IC1.IN6
1-10C1.OUT16
1-10C2.OUT16
1-10C1.OUT14
1-10C1.OUT14
1-1IC1.IN7
1-10C1.OUT17
1-10C2.OUT16
1-10C1.OUT14
1-10C1.OUT15
1-1IC1.IN1
1-10C1.OUT18
1-10C1.OUT15
1-10C1.OUT15
1-1IC1.IN6
1-10C1.OUT19
1-10C2.OUT16
1-10C1.OUT15
1-10C1.OUT15
1-1IC1.IN7
1-10C1.OUT17
1-10C2.OUT16
1-10C1.OUT15
END

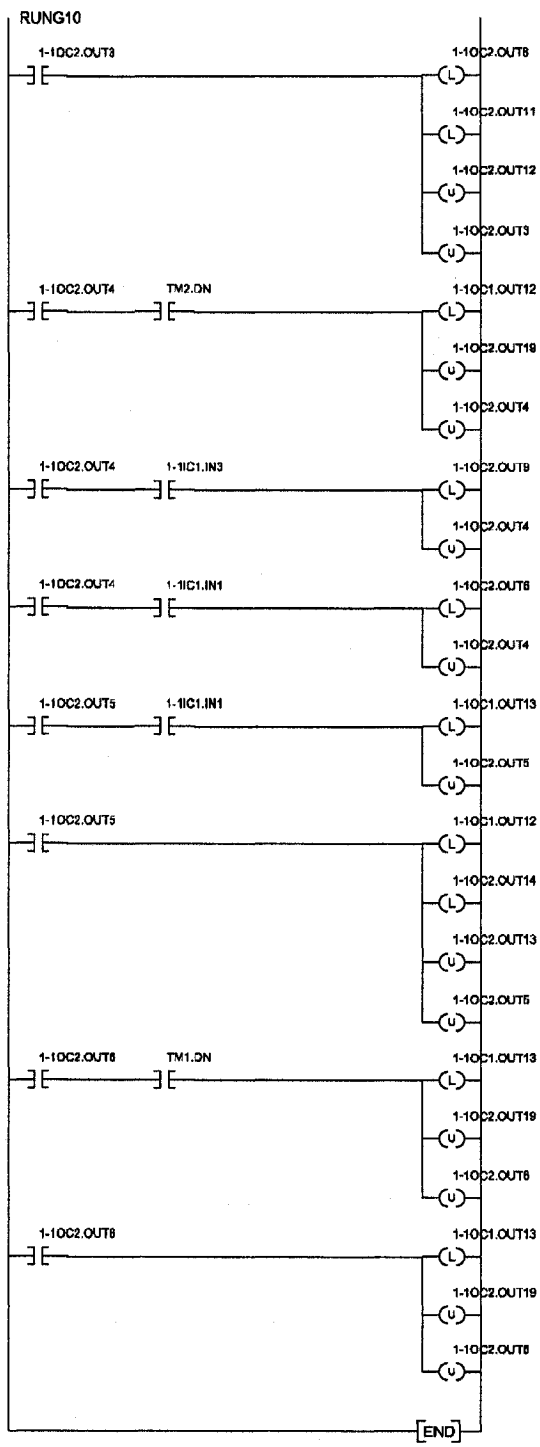

RUNG5
1-1OC1.OUT15
1-1IC1.IN1
1-1OC1.OUT13
1-1OC1.OUT15
1-1OC1.OUT16
1-1OC1.OUT20
1-1OC2.OUT17
1-1OC1.OUT16
1-1OC1.OUT17
1-1OC1.OUT21
1-1OC2.OUT20
1-1OC1.OUT17
1-1OC1.OUT18
1-1IC1.IN6
1-1OC1.OUT13
1-1OC2.OUT16
1-1OC1.OUT18
1-1OC1.OUT18
1-1IC1.IN7
1-1OC1.OUT13
1-1OC2.OUT16
1-1OC1.OUT18
1-1OC1.OUT19
1-1IC1.IN1
1-1OC1.OUT13
1-1OC1.OUT19
1-1OC1.OUT19
1-1OC1.OUT22
1-1OC2.OUT17
1-1OC1.OUT19
1-1OC1.OUT20
1-1IC1.IN1
1-1OC1.OUT24
1-1OC1.OUT20
END

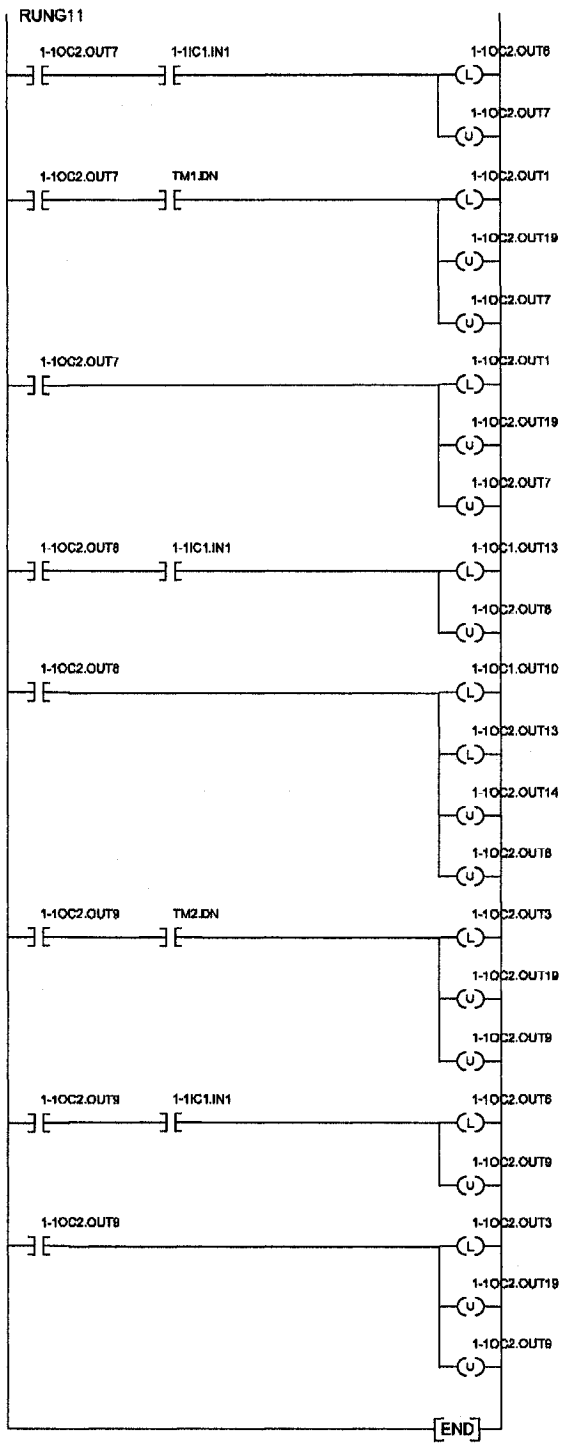

RUNG6
1-1OC1.OUT20
1-1IC1.IN8
1-1OC1.OUT23
1-1OC2.OUT17
1-1OC1.OUT20
1-1OC1.OUT20
1-1IC1.IN9
1-1OC1.OUT17
1-1OC2.OUT17
1-1OC1.OUT20
1-1OC1.OUT21
1-1OC1.OUT25
1-1OC2.OUT21
1-1OC1.OUT21
1-1OC1.OUT22
1-1IC1.IN1
1-1OC1.OUT24
1-1OC1.OUT22
1-1OC1.OUT22
1-1IC1.IN8
1-1OC1.OUT26
1-1OC2.OUT17
1-1OC1.OUT22
1-1OC1.OUT22
1-1IC1.IN9
1-1OC1.OUT17
1-1OC2.OUT17
1-1OC1.OUT22
1-1OC1.OUT23
1-1IC1.IN1
1-1OC1.OUT13
1-1OC1.OUT23
1-1OC1.OUT23
1-1OC1.OUT27
1-1OC2.OUT18
1-1OC1.OUT23
END


RUNG7
1-1OC1.OUT24
1-1IC1.IN8
1-1OC1.OUT13
1-1OC2.OUT17
1-1OC1.OUT24
1-1OC1.OUT24
1-1IC1.IN9
1-1OC1.OUT13
1-1OC2.OUT17
1-1OC1.OUT24
1-1OC1.OUT25
1-1OC1.OUT28
1-1OC2.OUT11
1-1OC2.OUT12
1-1OC1.OUT25
1-1OC1.OUT26
1-1IC1.IN1
1-1OC1.OUT13
1-1OC1.OUT26
1-1OC1.OUT26
1-1OC1.OUT29
1-1OC2.OUT18
1-1OC1.OUT26
1-1OC1.OUT27
1-1IC1.IN1
1-1OC1.OUT31
1-1OC1.OUT27
1-1OC1.OUT27
1-1IC1.IN10
1-1OC1.OUT30
1-1OC2.OUT18
1-1OC1.OUT27
1-1OC1.OUT27
1-1IC1.IN11
1-1OC1.OUT17
1-1OC2.OUT18
1-1OC1.OUT27
1-1OC1.OUT28
1-1OC1.OUT0
1-1OC2.OUT13
1-1OC2.OUT14
1-1OC1.OUT28
END


RUNG8
1-1OC1.OUT29
1-1IC1.IN1
1-1OC1.OUT31
1-1OC1.OUT29
1-1OC1.OUT29
1-1IC1.IN10
1-1OC2.OUT0
1-1OC2.OUT18
1-1OC1.OUT29
1-1OC1.OUT29
1-1IC1.IN11
1-1OC1.OUT17
1-1OC2.OUT18
1-1OC1.OUT29
1-1OC1.OUT30
1-1IC1.IN2
1-1OC2.OUT1
1-1OC1.OUT30
1-1OC1.OUT30
1-1IC1.IN1
1-1OC1.OUT13
1-1OC1.OUT30
1-1OC1.OUT30
1-1OC2.OUT2
1-1OC2.OUT19
1-1OC1.OUT30
1-1OC2.OUT19
TM
Timer ON-Delay
Timer T4:1
Time Base 1s
Preset 20
Accum 0
EN
DN
1-1OC1.OUT31
1-1IC1.IN10
1-1OC1.OUT13
1-1OC2.OUT18
1-1OC1.OUT31
1-1OC1.OUT31
1-1IC1.IN11
1-1OC1.OUT13
1-1OC2.OUT18
1-1OC1.OUT31
1-1OC2.OUT0
1-1IC1.IN1
1-1OC1.OUT13
1-1OC2.OUT0
1-1OC2.OUT0
1-1IC1.IN3
1-1OC2.OUT3
1-1OC2.OUT0
END

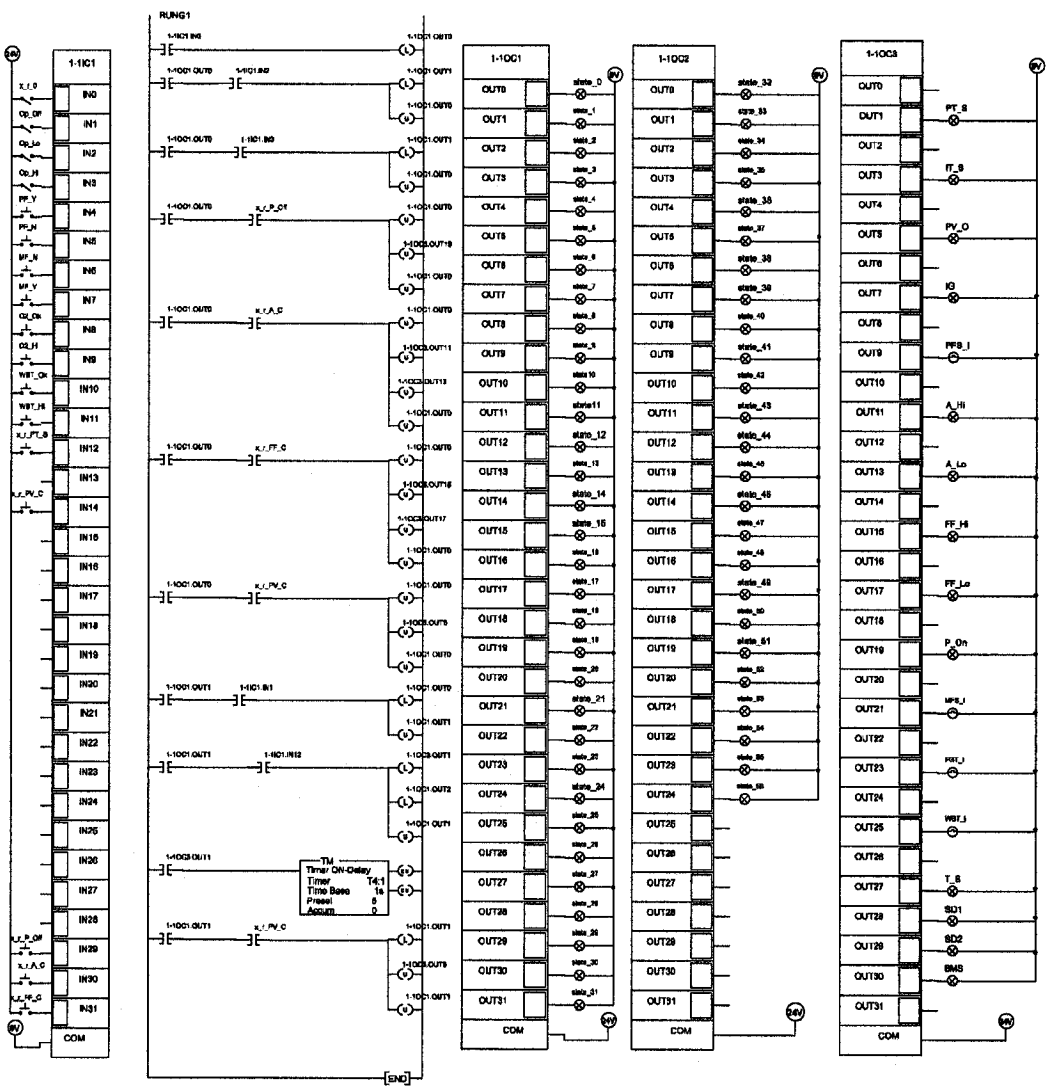

RUNG9
1-1OC2.OUT0
1-1OC2.OUT4
1-1OC2.OUT19
1-1OC2.OUT0
1-1OC2.OUT19
TM
Timer ON-Delay
Timer T4:2
Time Base 1s
Preset 20
Accum 0
EN
DN
1-1OC2.OUT1
1-1IC1.IN1
1-1OC1.OUT13
1-1OC2.OUT1
1-1OC2.OUT1
1-1OC2.OUT5
1-1OC2.OUT12
1-1OC2.OUT11
1-1OC2.OUT1
1-1OC2.OUT2
TM1.DN
1-1OC1.OUT10
1-1OC2.OUT19
1-1OC2.OUT2
1-1OC2.OUT2
1-1IC1.IN1
1-1OC2.OUT6
1-1OC2.OUT2
1-1OC2.OUT2
1-1IC1.IN2
1-1OC2.OUT7
1-1OC2.OUT2
1-1OC2.OUT3
1-1IC1.IN1
1-1OC1.OUT13
1-1OC2.OUT3
END

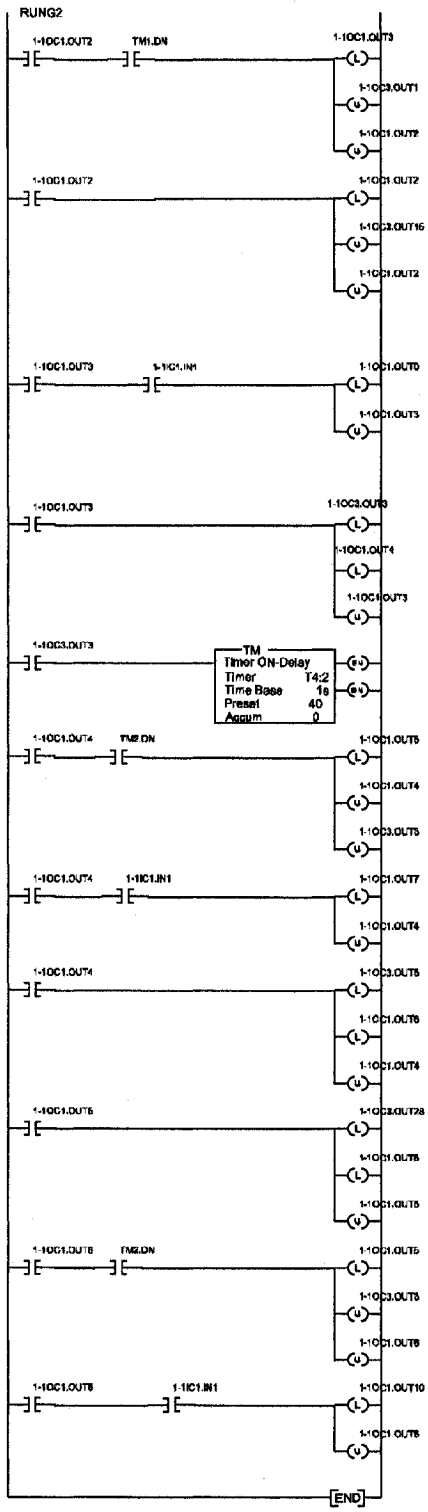

RUNG10
1-10C2.OUT3
1-10C2.OUT6
1-10C2.OUT11
1-10C2.OUT12
1-10C2.OUT3
1-10C2.OUT4
TM2.DN
1-10C1.OUT12
1-10C2.OUT19
1-10C2.OUT4
1-10C2.OUT4
1-1IC1.IN3
1-10C2.OUT9
1-10C2.OUT4
1-10C2.OUT4
1-1IC1.IN1
1-10C2.OUT6
1-10C2.OUT4
1-10C2.OUT5
1-1IC1.IN1
1-10C1.OUT13
1-10C2.OUT5
1-10C2.OUT5
1-10C1.OUT12
1-10C2.OUT14
1-10C2.OUT13
1-10C2.OUT5
1-10C2.OUT6
TM1.DN
1-10C1.OUT13
1-10C2.OUT19
1-10C2.OUT6
1-10C2.OUT6
1-10C1.OUT13
1-10C2.OUT19
1-10C2.OUT6
[END]

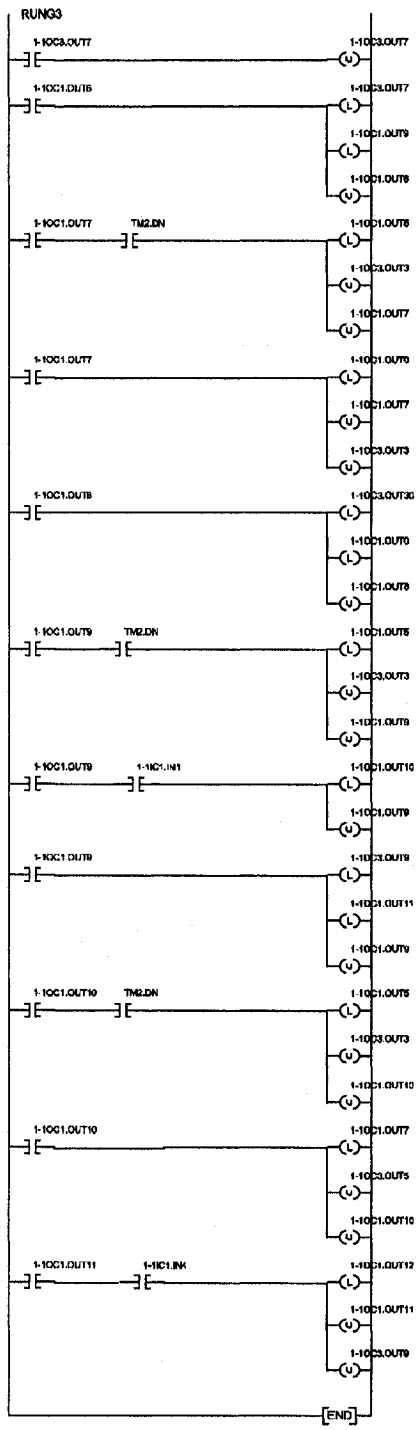

RUNG11
1-1OC2.OUT7
1-1IC1.IN1
1-1OC2.OUT6
1-1OC2.OUT7
1-1OC2.OUT7
TM1.IDN
1-1OC2.OUT1
1-1OC2.OUT19
1-1OC2.OUT7
1-1OC2.OUT7
1-1OC2.OUT1
1-1OC2.OUT19
1-1OC2.OUT7
1-1OC2.OUT8
1-1IC1.IN1
1-1OC1.OUT13
1-1OC2.OUT8
1-1OC2.OUT8
1-1OC1.OUT10
1-1OC2.OUT13
1-1OC2.OUT14
1-1OC2.OUT8
1-1OC2.OUT9
TM2.IDN
1-1OC2.OUT3
1-1OC2.OUT19
1-1OC2.OUT9
1-1OC2.OUT9
1-1IC1.IN1
1-1OC2.OUT6
1-1OC2.OUT9
1-1OC2.OUT9
1-1OC2.OUT3
1-1OC2.OUT19
1-1OC2.OUT9
END

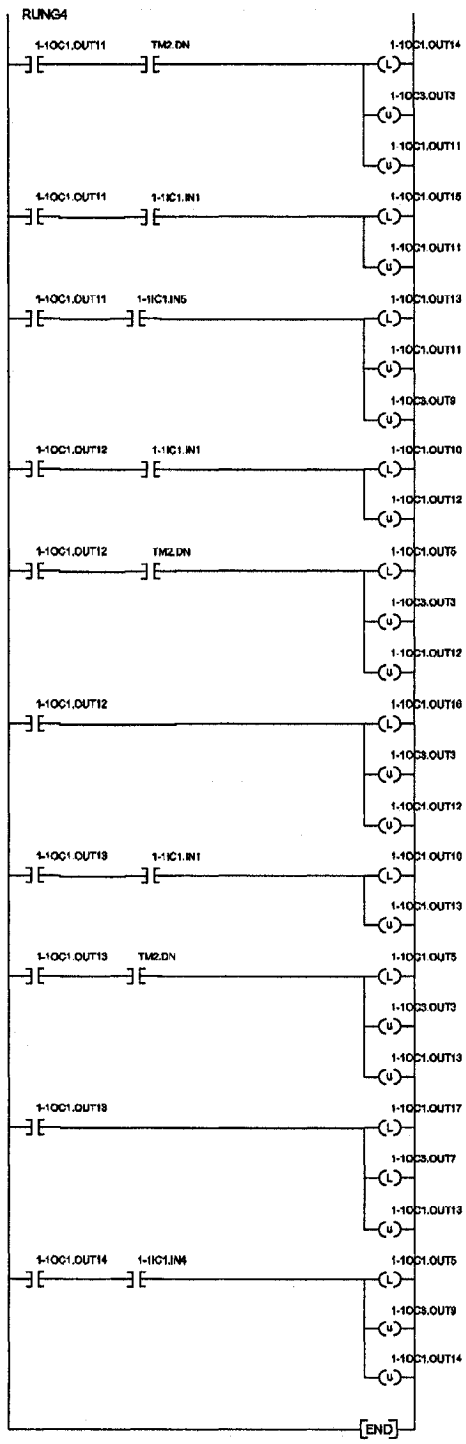
Appendix III

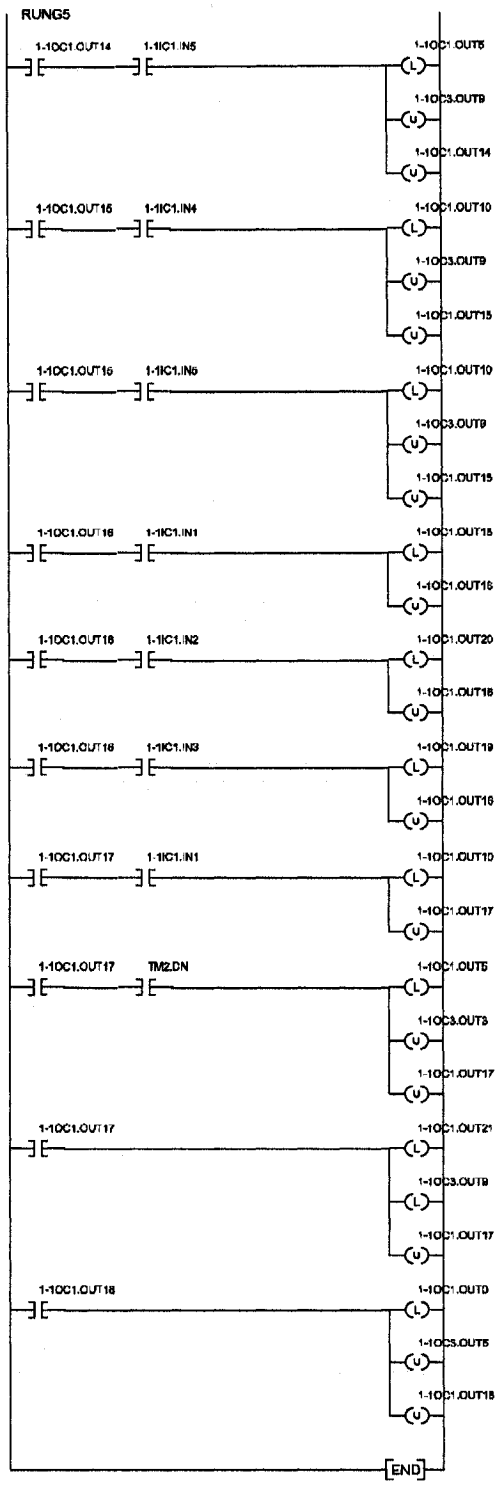
PLC-based implementation of the complete boiler controller

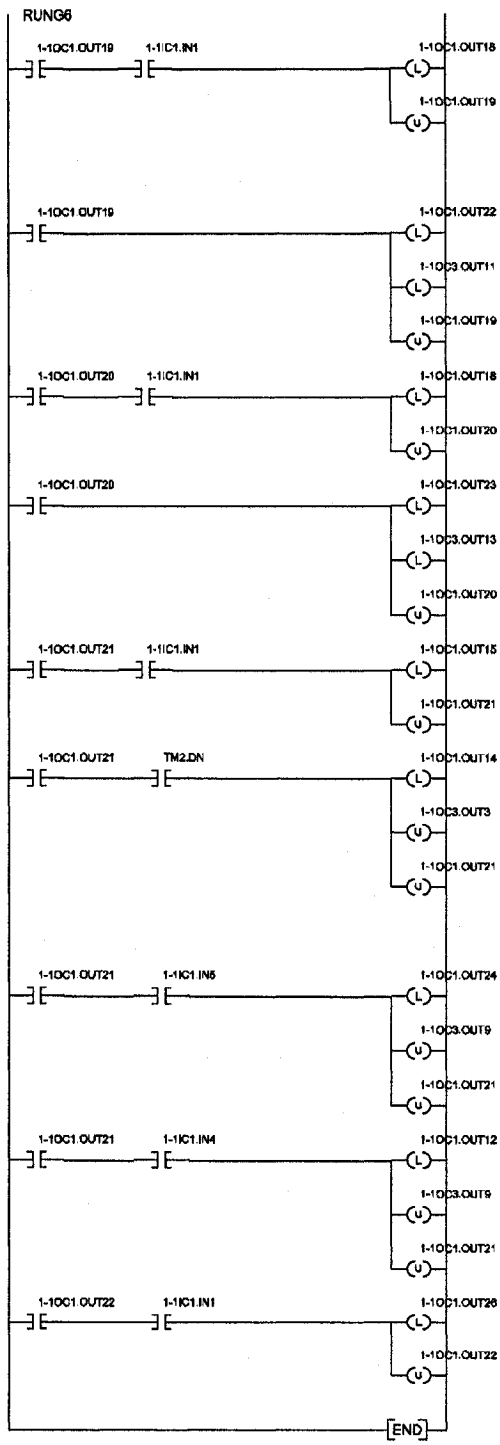

RUNG1
1-1IC1
1-1OC1
1-1OC2
1-1OC3
TM
Timer ON-Delay
Timer T4:1
Time Base 1s
Preset 5
Accum 0
[END]

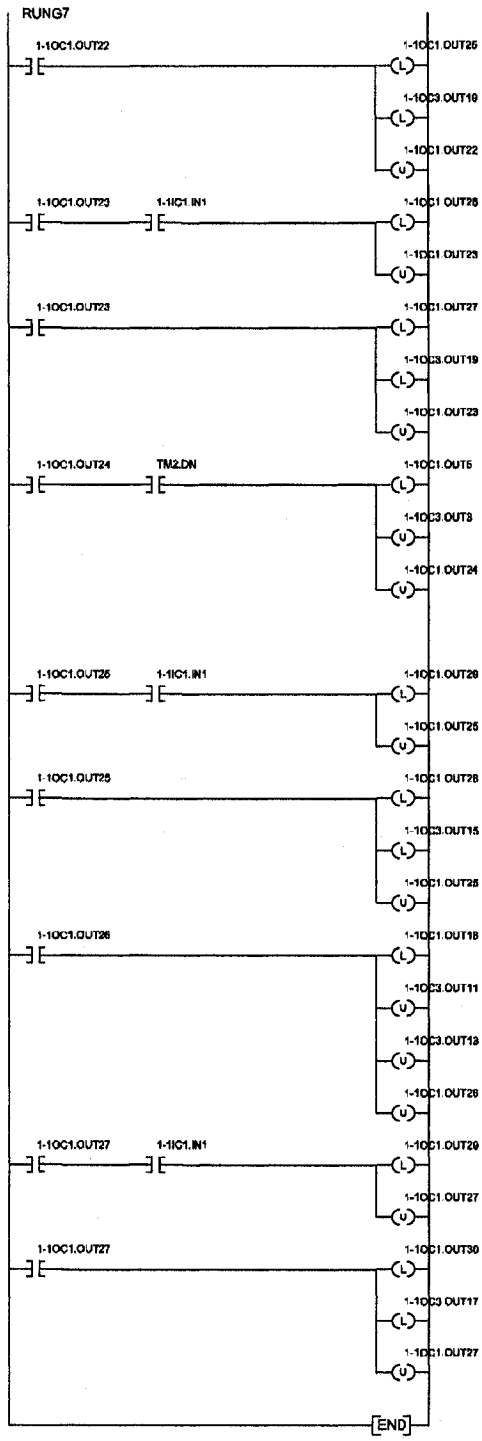

RUNG2
1-10C1.OUT2
TM1.DN
1-10C1.OUT3
1-10C3.OUT1
1-10C1.OUT2
1-10C1.OUT2
1-10C1.OUT2
1-10C3.OUT16
1-10C1.OUT2
1-10C1.OUT3
1-1IC1.IN1
1-10C1.OUT3
1-10C1.OUT3
1-10C1.OUT3
1-10C3.OUT3
1-10C1.OUT4
1-10C1.OUT3
1-10C3.OUT3
TM
Timer ON-Delay
Timer T4:2
Time Base 1s
Preset 40
Accum 0
1-10C1.OUT4
TM2.DN
1-10C1.OUT5
1-10C1.OUT4
1-10C3.OUT5
1-10C1.OUT4
1-1IC1.IN1
1-10C1.OUT7
1-10C1.OUT4
1-10C1.OUT4
1-10C3.OUT5
1-10C1.OUT6
1-10C1.OUT4
1-10C1.OUT5
1-10C3.OUT28
1-10C1.OUT5
1-10C1.OUT5
1-10C1.OUT6
TM3.DN
1-10C1.OUT5
1-10C3.OUT6
1-10C1.OUT6
1-10C1.OUT6
1-1IC1.IN1
1-10C1.OUT10
1-10C1.OUT6
END

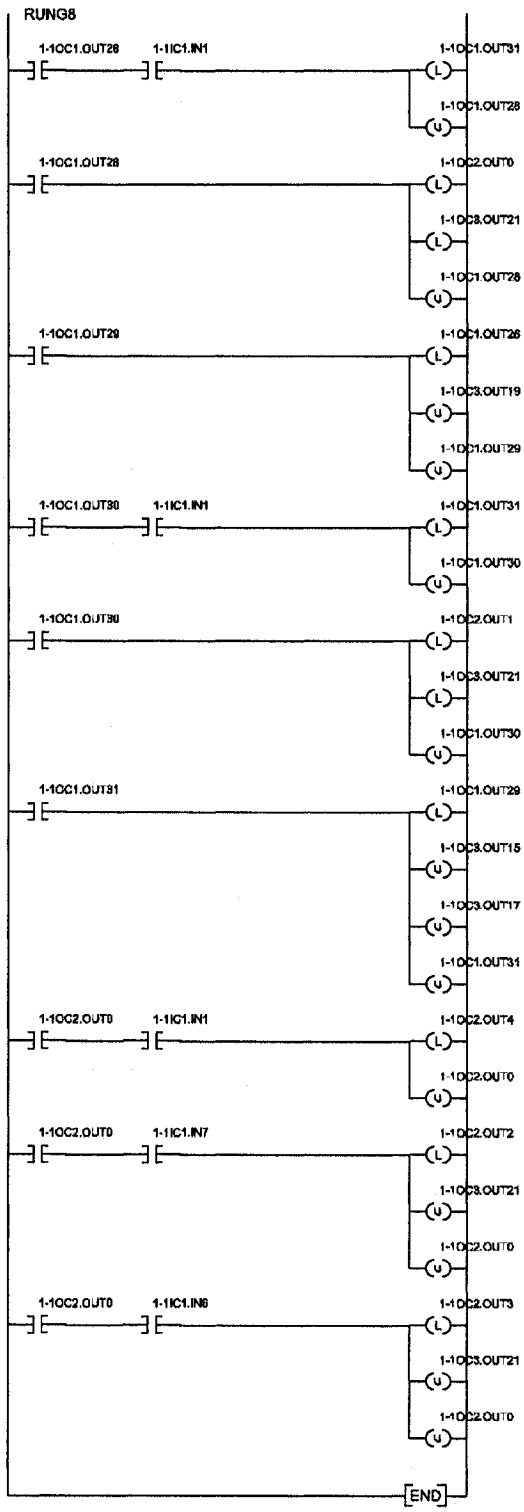

RUNG3
1-10C3.OUT7
1-10C3.OUT7
1-10C1.OUT16
1-10C3.OUT7
1-10C1.OUT9
1-10C1.OUT9
1-10C1.OUT7
TM2.DN
1-10C1.OUT8
1-10C3.OUT3
1-10C1.OUT7
1-10C1.OUT7
1-10C1.OUT0
1-10C1.OUT7
1-10C3.OUT3
1-10C1.OUT8
1-10C3.OUT30
1-10C1.OUT0
1-10C1.OUT8
1-10C1.OUT9
TM2.DN
1-10C1.OUT5
1-10C3.OUT3
1-10C1.OUT9
1-10C1.OUT9
1-11C1.IN1
1-10C1.OUT10
1-10C1.OUT9
1-10C1.OUT9
1-10C3.OUT9
1-10C1.OUT11
1-10C1.OUT9
1-10C1.OUT10
TM2.DN
1-10C1.OUT5
1-10C3.OUT3
1-10C1.OUT10
1-10C1.OUT10
1-10C1.OUT7
1-10C3.OUT5
1-10C1.OUT10
1-10C1.OUT11
1-11C1.IN4
1-10C1.OUT12
1-10C1.OUT11
1-10C3.OUT9
[END]

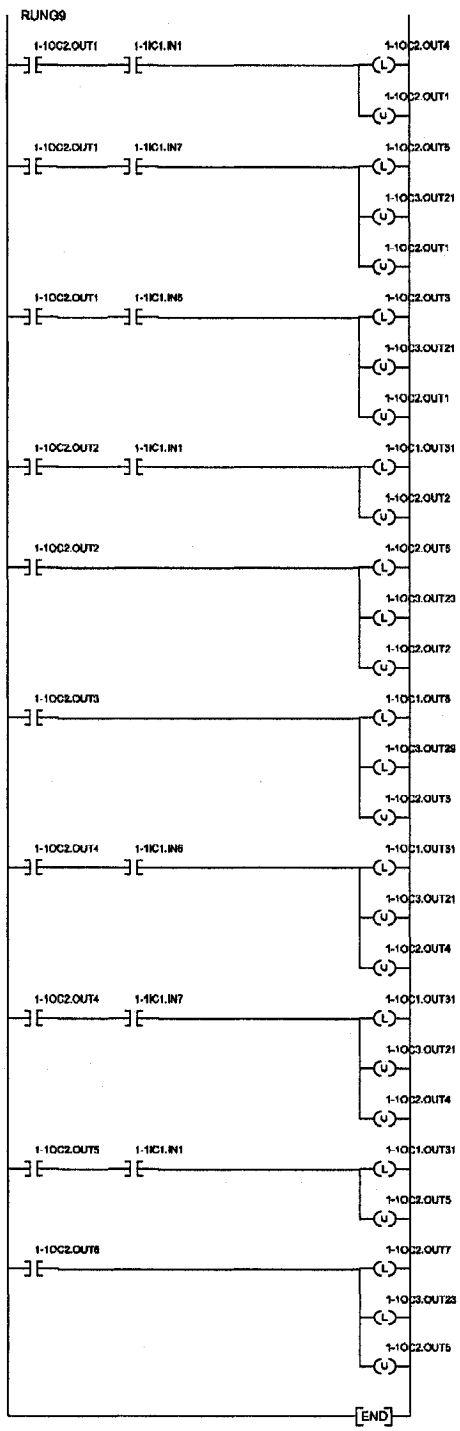

RUNG4
1-1OC1.OUT11
TM2.DN
1-1OC1.OUT14
1-1OC3.OUT3
1-1OC1.OUT11
1-1OC1.OUT11
1-1IC1.IN1
1-1OC1.OUT15
1-1OC1.OUT11
1-1OC1.OUT11
1-1IC1.IN5
1-1OC1.OUT13
1-1OC1.OUT11
1-1OC3.OUT9
1-1OC1.OUT12
1-1IC1.IN1
1-1OC1.OUT10
1-1OC1.OUT12
1-1OC1.OUT12
TM2.DN
1-1OC1.OUT5
1-1OC3.OUT3
1-1OC1.OUT12
1-1OC1.OUT12
1-1OC1.OUT16
1-1OC3.OUT3
1-1OC1.OUT12
1-1OC1.OUT13
1-1IC1.IN1
1-1OC1.OUT10
1-1OC1.OUT13
1-1OC1.OUT13
TM2.DN
1-1OC1.OUT5
1-1OC3.OUT3
1-1OC1.OUT13
1-1OC1.OUT18
1-1OC1.OUT17
1-1OC3.OUT7
1-1OC1.OUT13
1-1OC1.OUT14
1-1IC1.IN4
1-1OC1.OUT5
1-1OC3.OUT9
1-1OC1.OUT14
END

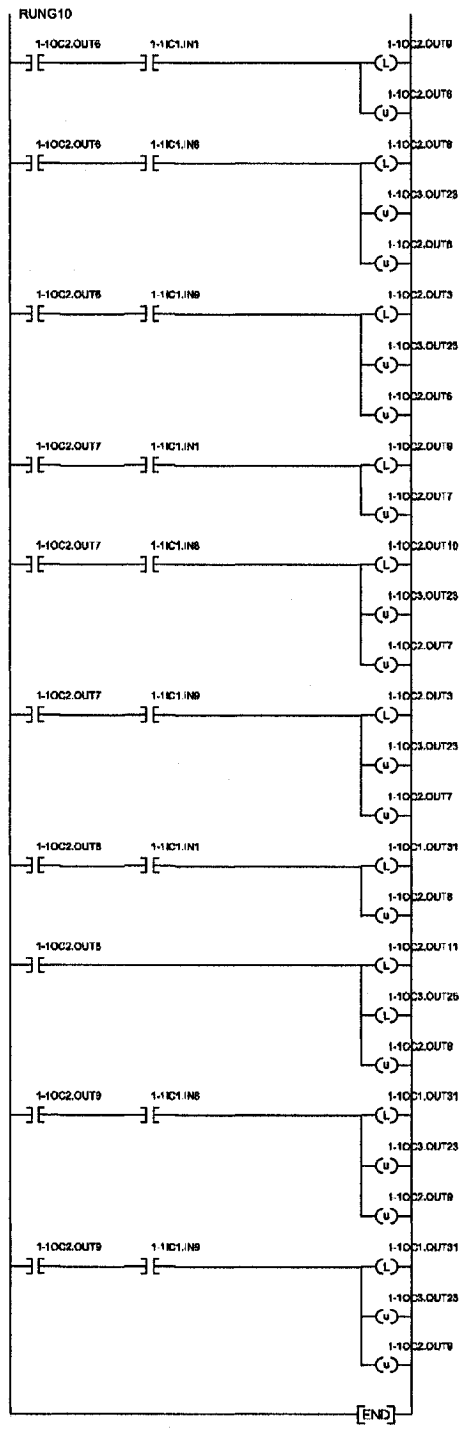

RUNG5
1-1OC1.OUT14
1-1IC1.IN5
1-1OC1.OUT5
1-1OC3.OUT9
1-1OC1.OUT14
1-1OC1.OUT15
1-1IC1.IN4
1-1OC1.OUT10
1-1OC3.OUT9
1-1OC1.OUT15
1-1OC1.OUT15
1-1IC1.IN5
1-1OC1.OUT10
1-1OC3.OUT9
1-1OC1.OUT15
1-1OC1.OUT16
1-1IC1.IN1
1-1OC1.OUT15
1-1OC1.OUT16
1-1OC1.OUT18
1-1IC1.IN2
1-1OC1.OUT20
1-1OC1.OUT18
1-1OC1.OUT18
1-1IC1.IN3
1-1OC1.OUT19
1-1OC1.OUT18
1-1OC1.OUT17
1-1IC1.IN1
1-1OC1.OUT10
1-1OC1.OUT17
1-1OC1.OUT17
TM2.DN
1-1OC1.OUT5
1-1OC3.OUT9
1-1OC1.OUT17
1-1OC1.OUT17
1-1OC1.OUT21
1-1OC3.OUT9
1-1OC1.OUT17
1-1OC1.OUT18
1-1OC1.OUT0
1-1OC3.OUT5
1-1OC1.OUT18
END

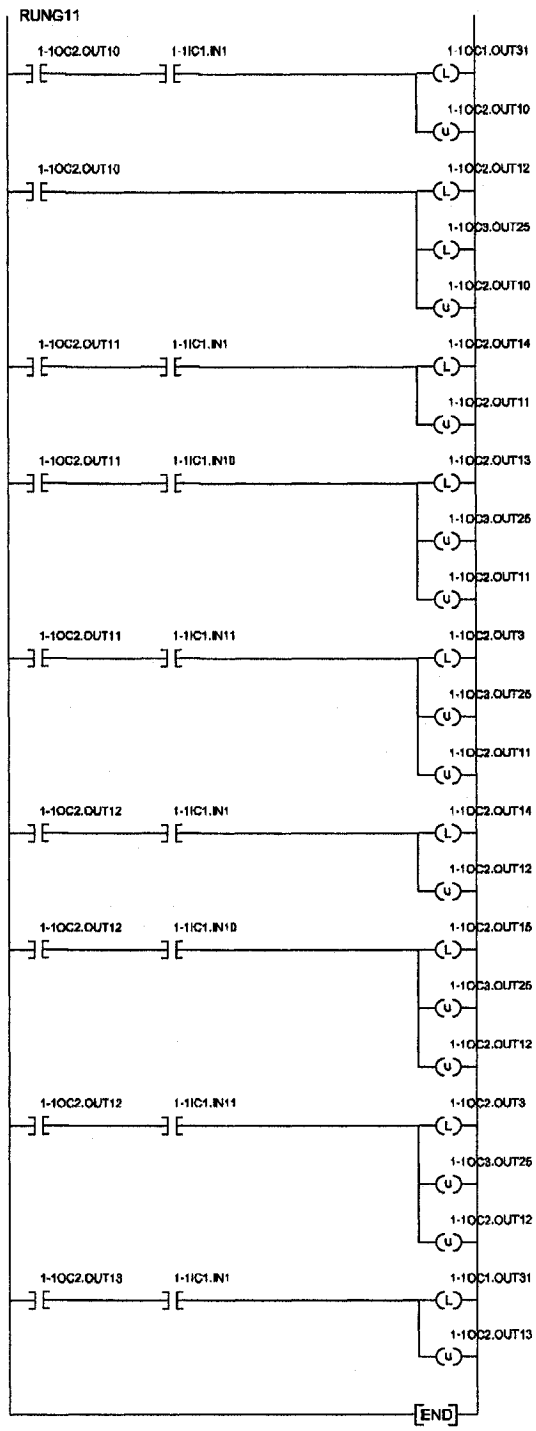

RUNG6
1-1OC1.OUT19
1-1IC1.IN1
1-1OC1.OUT18
1-1OC1.OUT19
1-1OC1.OUT19
1-1OC1.OUT22
1-1OC3.OUT11
1-1OC1.OUT19
1-1OC1.OUT20
1-1IC1.IN1
1-1OC1.OUT18
1-1OC1.OUT20
1-1OC1.OUT20
1-1OC1.OUT23
1-1OC3.OUT13
1-1OC1.OUT20
1-1OC1.OUT21
1-1IC1.IN1
1-1OC1.OUT15
1-1OC1.OUT21
1-1OC1.OUT21
TM2.DN
1-1OC1.OUT14
1-1OC3.OUT3
1-1OC1.OUT21
1-1OC1.OUT21
1-1IC1.IN5
1-1OC1.OUT24
1-1OC3.OUT9
1-1OC1.OUT21
1-1OC1.OUT21
1-1IC1.IN4
1-1OC1.OUT12
1-1OC3.OUT9
1-1OC1.OUT21
1-1OC1.OUT22
1-1IC1.IN1
1-1OC1.OUT26
1-1OC1.OUT22
END

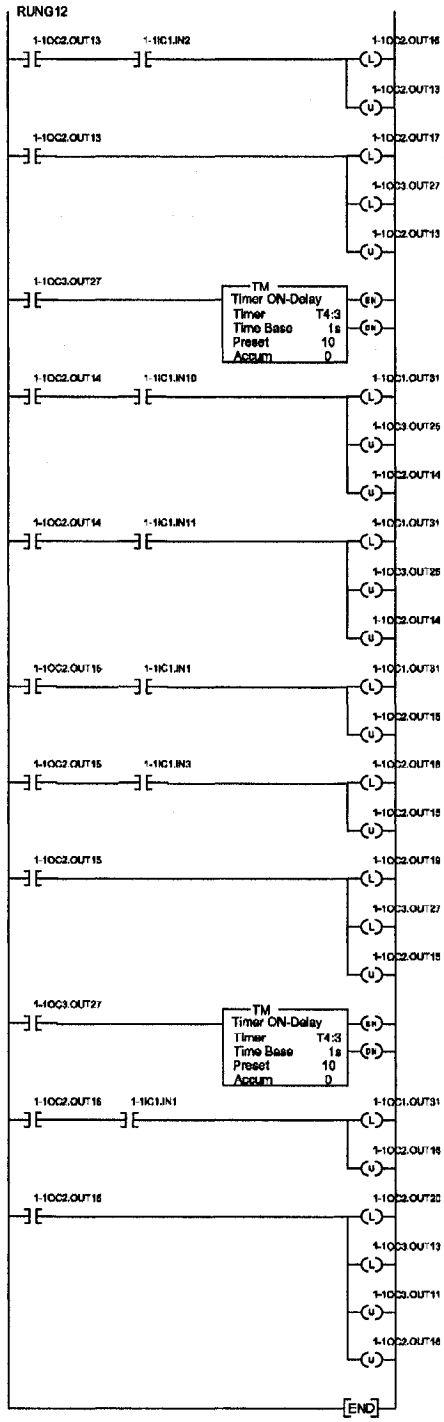

RUNG7
1-1OC1.OUT22
1-1OC1.OUT25
1-1OC3.OUT19
1-1OC1.OUT22
1-1OC1.OUT23
1-1IC1.IN1
1-1OC1.OUT26
1-1OC1.OUT23
1-1OC1.OUT23
1-1OC1.OUT27
1-1OC3.OUT19
1-1OC1.OUT23
1-1OC1.OUT24
TM2.DN
1-1OC1.OUT5
1-1OC3.OUT3
1-1OC1.OUT24
1-1OC1.OUT25
1-1IC1.IN1
1-1OC1.OUT29
1-1OC1.OUT25
1-1OC1.OUT25
1-1OC1.OUT28
1-1OC3.OUT15
1-1OC1.OUT25
1-1OC1.OUT26
1-1OC1.OUT18
1-1OC3.OUT11
1-1OC3.OUT13
1-1OC1.OUT26
1-1OC1.OUT27
1-1IC1.IN1
1-1OC1.OUT29
1-1OC1.OUT27
1-1OC1.OUT27
1-1OC1.OUT30
1-1OC3.OUT17
1-1OC1.OUT27
END

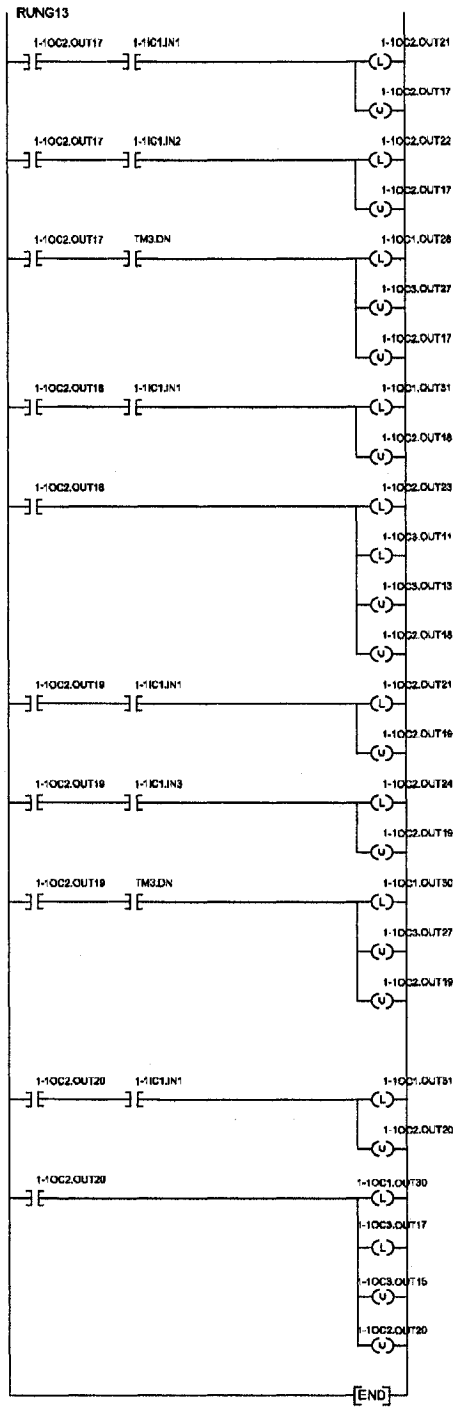

RUNG8
1-1OC1.OUT28
1-1IC1.IN1
1-1OC1.OUT31
1-1OC1.OUT28
1-1OC1.OUT28
1-1OC2.OUT0
1-1OC3.OUT21
1-1OC1.OUT28
1-1OC1.OUT29
1-1OC1.OUT26
1-1OC3.OUT19
1-1OC1.OUT29
1-1OC1.OUT30
1-1IC1.IN1
1-1OC1.OUT31
1-1OC1.OUT30
1-1OC1.OUT30
1-1OC2.OUT1
1-1OC3.OUT21
1-1OC1.OUT30
1-1OC1.OUT31
1-1OC1.OUT29
1-1OC3.OUT15
1-1OC3.OUT17
1-1OC1.OUT31
1-1OC2.OUT0
1-1IC1.IN1
1-1OC2.OUT4
1-1OC2.OUT0
1-1OC2.OUT0
1-1IC1.IN7
1-1OC2.OUT2
1-1OC3.OUT21
1-1OC2.OUT0
1-1OC2.OUT0
1-1IC1.IN6
1-1OC2.OUT3
1-1OC3.OUT21
1-1OC2.OUT0
END

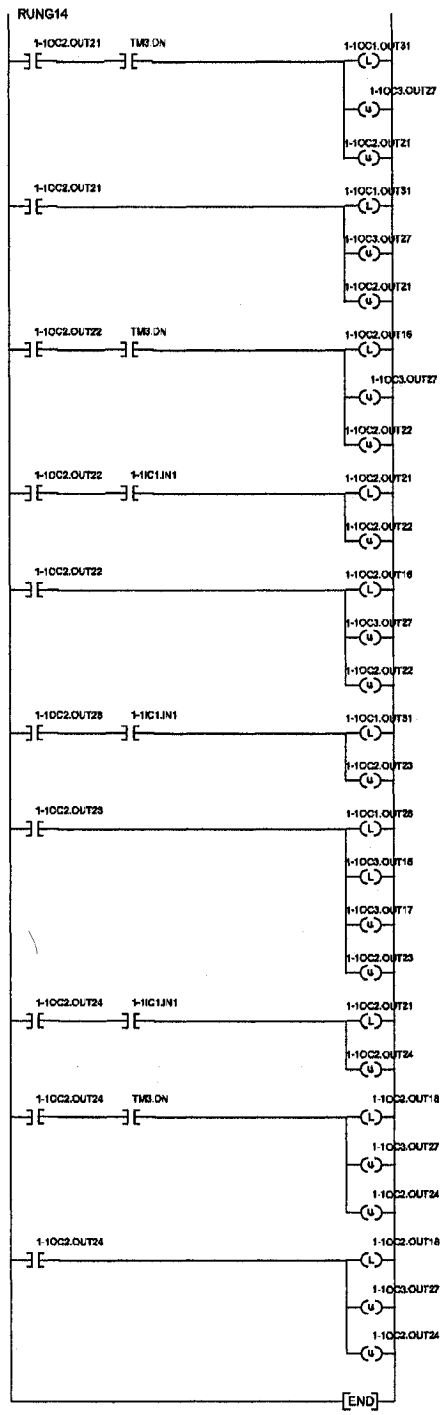

RUNG9
1-1OC2.OUT1
1-1IC1.IN1
1-1OC2.OUT4
1-1OC2.OUT1
1-1OC2.OUT1
1-1IC1.IN7
1-1OC2.OUT5
1-1OC3.OUT21
1-1OC2.OUT1
1-1OC2.OUT1
1-1IC1.IN6
1-1OC2.OUT3
1-1OC3.OUT21
1-1OC2.OUT1
1-1OC2.OUT2
1-1IC1.IN1
1-1OC1.OUT31
1-1OC2.OUT2
1-1OC2.OUT2
1-1OC2.OUT5
1-1OC3.OUT23
1-1OC2.OUT2
1-1OC2.OUT3
1-1OC1.OUT8
1-1OC3.OUT29
1-1OC2.OUT3
1-1OC2.OUT4
1-1IC1.IN6
1-1OC1.OUT31
1-1OC3.OUT21
1-1OC2.OUT4
1-1OC2.OUT4
1-1IC1.IN7
1-1OC1.OUT31
1-1OC3.OUT21
1-1OC2.OUT4
1-1OC2.OUT5
1-1IC1.IN1
1-1OC1.OUT31
1-1OC2.OUT5
1-1OC2.OUT6
1-1OC2.OUT7
1-1OC3.OUT23
1-1OC2.OUT5
END


RUNG10
1-1OC2.OUT6
1-1IC1.IN1
1-1OC2.OUT9
1-1OC2.OUT6
1-1OC2.OUT6
1-1IC1.IN8
1-1OC2.OUT9
1-1OC3.OUT25
1-1OC2.OUT6
1-1OC2.OUT6
1-1IC1.IN9
1-1OC2.OUT9
1-1OC3.OUT25
1-1OC2.OUT6
1-1OC2.OUT7
1-1IC1.IN1
1-1OC2.OUT9
1-1OC2.OUT7
1-1OC2.OUT7
1-1IC1.IN8
1-1OC2.OUT10
1-1OC3.OUT25
1-1OC2.OUT7
1-1OC2.OUT7
1-1IC1.IN9
1-1OC2.OUT9
1-1OC3.OUT25
1-1OC2.OUT7
1-1OC2.OUT8
1-1IC1.IN1
1-1OC1.OUT31
1-1OC2.OUT8
1-1OC2.OUT8
1-1OC2.OUT11
1-1OC3.OUT25
1-1OC2.OUT8
1-1OC2.OUT9
1-1IC1.IN8
1-1OC1.OUT31
1-1OC3.OUT23
1-1OC2.OUT9
1-1OC2.OUT9
1-1IC1.IN9
1-1OC1.OUT31
1-1OC3.OUT23
1-1OC2.OUT9
[END]


RUNG11
1-1OC2.OUT10
1-1IC1.IN1
1-1OC1.OUT31
1-1OC2.OUT10
1-1OC2.OUT10
1-1OC2.OUT12
1-1OC3.OUT25
1-1OC2.OUT10
1-1OC2.OUT11
1-1IC1.IN1
1-1OC2.OUT14
1-1OC2.OUT11
1-1OC2.OUT11
1-1IC1.IN10
1-1OC2.OUT13
1-1OC3.OUT25
1-1OC2.OUT11
1-1OC2.OUT11
1-1IC1.IN11
1-1OC2.OUT3
1-1OC3.OUT25
1-1OC2.OUT11
1-1OC2.OUT12
1-1IC1.IN1
1-1OC2.OUT14
1-1OC2.OUT12
1-1OC2.OUT12
1-1IC1.IN10
1-1OC2.OUT15
1-1OC3.OUT25
1-1OC2.OUT12
1-1OC2.OUT12
1-1IC1.IN11
1-1OC2.OUT3
1-1OC3.OUT25
1-1OC2.OUT12
1-1OC2.OUT13
1-1IC1.IN1
1-1OC1.OUT31
1-1OC2.OUT13
L
U
END


RUNG12
1-1OC2.OUT13
1-1IC1.IN2
1-1OC2.OUT16
1-1OC2.OUT13
1-1OC2.OUT13
1-1OC2.OUT17
1-1OC3.OUT27
1-1OC2.OUT13
1-1OC3.OUT27
TM
Timer ON-Delay
Timer T4:3
Time Base 1s
Preset 10
Accum 0
1-1OC2.OUT14
1-1IC1.IN10
1-1OC1.OUT31
1-1OC3.OUT25
1-1OC2.OUT14
1-1OC2.OUT14
1-1IC1.IN11
1-1OC1.OUT31
1-1OC3.OUT25
1-1OC2.OUT14
1-1OC2.OUT15
1-1IC1.IN1
1-1OC1.OUT31
1-1OC2.OUT15
1-1OC2.OUT15
1-1IC1.IN3
1-1OC2.OUT18
1-1OC2.OUT15
1-1OC2.OUT15
1-1OC2.OUT19
1-1OC3.OUT27
1-1OC2.OUT15
1-1OC3.OUT27
TM
Timer ON-Delay
Timer T4:3
Time Base 1s
Preset 10
Accum 0
1-1OC2.OUT15
1-1IC1.IN1
1-1OC1.OUT31
1-1OC2.OUT15
1-1OC2.OUT15
1-1OC2.OUT20
1-1OC3.OUT13
1-1OC3.OUT11
1-1OC2.OUT15
END


RUNG13
1-1OC2.OUT17
1-1IC1.IN1
1-1OC2.OUT21
1-1OC2.OUT17
1-1OC2.OUT17
1-1IC1.IN2
1-1OC2.OUT22
1-1OC2.OUT17
1-1OC2.OUT17
TM3.DN
1-1OC1.OUT28
1-1OC3.OUT27
1-1OC2.OUT17
1-1OC2.OUT18
1-1IC1.IN1
1-1OC1.OUT31
1-1OC2.OUT18
1-1OC2.OUT18
1-1OC2.OUT23
1-1OC3.OUT11
1-1OC3.OUT13
1-1OC2.OUT18
1-1OC2.OUT19
1-1IC1.IN1
1-1OC2.OUT21
1-1OC2.OUT19
1-1OC2.OUT19
1-1IC1.IN3
1-1OC2.OUT24
1-1OC2.OUT19
1-1OC2.OUT19
TM3.DN
1-1OC1.OUT30
1-1OC3.OUT27
1-1OC2.OUT19
1-1OC2.OUT20
1-1IC1.IN1
1-1OC1.OUT31
1-1OC2.OUT20
1-1OC2.OUT20
1-1OC1.OUT30
1-1OC3.OUT17
1-1OC3.OUT15
1-1OC2.OUT20
END


RUNG14
1-1OC2.OUT21
TM3.DN
1-1OC1.OUT31
1-1OC3.OUT27
1-1OC2.OUT21
1-1OC2.OUT21
1-1OC1.OUT31
1-1OC3.OUT27
1-1OC2.OUT21
1-1OC2.OUT22
TM3.DN
1-1OC2.OUT16
1-1OC3.OUT27
1-1OC2.OUT22
1-1OC2.OUT22
1-1IC1.IN1
1-1OC2.OUT21
1-1OC2.OUT22
1-1OC2.OUT22
1-1OC2.OUT16
1-1OC3.OUT27
1-1OC2.OUT22
1-1OC2.OUT28
1-1IC1.IN1
1-1OC1.OUT31
1-1OC2.OUT23
1-1OC2.OUT28
1-1OC1.OUT28
1-1OC3.OUT16
1-1OC3.OUT17
1-1OC2.OUT23
1-1OC2.OUT24
1-1IC1.IN1
1-1OC2.OUT21
1-1OC2.OUT24
1-1OC2.OUT24
TM3.DN
1-1OC2.OUT18
1-1OC3.OUT27
1-1OC2.OUT24
1-1OC2.OUT24
1-1OC2.OUT18
1-1OC3.OUT27
1-1OC2.OUT24
[END]