

A general technique for the PLC-Based implementation of RW supervisors with time delay functions

Murat Uzam

Received: 13 October 2011 / Accepted: 28 November 2011 / Published online: 11 December 2011
© Springer-Verlag London Limited 2011

Abstract The Supervisory Control Theory (SCT) introduced by Ramadge–Wonham (RW) is a general theory to design supervisors (controllers) for discrete event systems. Although over the two decades SCT has received wide attention in academia, industrial applications are very few, due to the fact that there is a discrepancy between the abstract RW supervisor and its physical implementation. In this paper, an easy to use, general and practical technique is proposed for the PLC-based implementation of RW supervisors with time delay functions. The applicability of the proposed method is demonstrated by means of a PLC-based real-time control of an experimental manufacturing system.

Keywords Supervisory control theory · Discrete event systems · RW supervisor · Automata · PLC · Real-time control · Ladder logic diagram

1 Introduction

The Supervisory Control Theory (SCT) introduced by Ramadge–Wonham (RW) is a general theory to design supervisors (controllers) for discrete event systems (DES) [1, 2]. Although over the two decades SCT has received wide attention in academia, industrial applications are very few [3]. This is mainly due to the fact that there is a discrepancy between the abstract RW supervisor and its physical implementation. Nowadays, programmable logic

controllers (PLC) with a popular programming language, namely ladder logic diagram (LLD; or ladder logic code), are widely used as an implementation tool to carry out control tasks in all modern industrial systems. The problem of physical implementation of a RW supervisor with PLC is specifically noticeable [3] as the step towards the physical implementation is not straightforward. To address the problem a number of efforts have been carried out. In [4–8], the application of SCT to flexible manufacturing cells and PLC-based implementations are studied. In [9, 10], local modular supervisors and their PLC-based implementations are considered. LLD [9] and sequential function charts [10] implementations of RW supervisors are proposed. The main feature of these works is that the control system is used as an interface between the real input–output signals and theoretical RW supervisors [9–11]. In [12, 13], the conversion of finite state machines to LLD is proposed. The LLD implementation of finite state machines represented as state diagram is studied in [14]. The problems and possible solutions for PLC implementation for SCT are discussed in [3]. One of such problems is called avalanche effect, which makes the program skip over an arbitrary number of states during the same PLC scan cycle. An approach to eliminate avalanche effect problem is proposed in [15]. These results are mainly concerned with the implementation of a given untimed finite state machine as LLD on a PLC.

On the other hand, today's manufacturing systems can be classified as DES with timing requirements. Timing may be associated with the duration of an operation with an expected time before some event occurrence such as a failure [16]. As an automaton framework, timed automata (TA) were introduced in [17], and have been extensively studied [18–21]. These models are extension of finite automata with clocks, guards, resets, and enable one to specify real-time systems. A transition in the TA would be labeled with an event and its associated time intervals. This time

M. Uzam (✉)
Melikşah Üniversitesi,
Mühendislik Mimarlık Fakültesi,
Elektrik-Elektronik Mühendisliği Bölümü,
Talas 38280 Kayseri, Turkey
e-mail: murat_uzam@melikshah.edu.tr

interval would be with respect to a specific clock and reset. The transitions between states are performed by evaluation of clocks and guards. A fundamental framework about control of timed discrete event systems was introduced in [22]. Supervisory control methods have been developed based on this framework for various situations [23, 24]. Timed automata models can be used for analysis, verification, or controller synthesis. Although it is possible to analyze and control these timed discrete event systems formally, general methods applied in the industry are heuristic rather than formal. For controlling small size systems, a simple time delay function may yield practical solution. If the systems to be controlled become more complex, then intuitive methods will become insufficient. On the other hand, in this case, the timed SCT based solution would not be practical due to computational complexity and state explosion problem. To date, there is no general and practical technique for the PLC-based implementation of RW supervisors with time delay functions. Our recent research project has focused on this problem. In this respect, three contributions are provided. A new approach for LLD implementation of RW supervisors is proposed in [25]. This method mainly deals with the assignment of actions (output signals) to the states of the supervisor in a systematic way. A simplified version of timed automata, called timed transition automata (TTA), is proposed to be used for the control of discrete event systems [26]. In TTA models, time delay values are associated with respective events. The proposed TTA model is very similar to T-Timed Petri nets described in [16, Chapter 3.4]. In TTA, a transition is labeled with an event and it is associated with a time delay such as (e, τ) . The automaton changes its state from one state to another, after occurrence and presence of event e and τ time has elapsed. TTA is based on the assumption that e be present during the τ time. Unfortunately, due to this assumption, TTA is not general. As a general method, in [27] the concept of *postponed event* is proposed for introducing the time delay functions within untimed automata. Postponed event pe is defined as the shifted version of an event e on the time axis by a deterministic time delay value τ . Postponed event is a general, easy to use, and practical method for PLC implementation of supervisory controllers with time delay functions. Supervisory controllers are obtained as in untimed systems by using the SCT framework [25] with the assumption that postponed events are taken into account as if they are normal (untimed) events. Then they are refined at the implementation stage by the assignment of deterministic time delay values to respective events. The implementation of postponed events is realized by using on delay timers, off-delay timers or pulse timers of PLCs [27]. By their definition, RW supervisors allow events to happen but they do not force them. On the other hand PLC-based controllers force all the enabled output events at any scan time. This is

one of the most problematic conceptual barriers between RW supervisors and their PLC-based controllers. To overcome this problem, automaton models of actuators are defined in a simple structure within this paper.

The main objective of this paper is to propose an easy to use, general, and practical technique for the PLC-based implementation of RW supervisors with time delay functions. To accomplish this task, the action assignment to the states of a RW supervisor method of [25] and the concept of postponed event proposed in [27] are both integrated within the design steps of SCT. TCT tool [28] is used to carry out the necessary computations to obtain RW type supervisors. The method proposed in this paper can be applied to both *high-level control*, where the role of the supervisor is to coordinate control of, in the manufacturing sense, machines, robots, fixtures, conveyors etc., and to *low-level control*, where the role of the supervisor is to arrange low-level interaction between the control devices, such as motors, actuators, etc. In [25] and [26], the applicability of the proposed method to high-level control was shown by means of a simple manufacturing system including two machines and one buffer. The applicability of the proposed method to low-level control is shown by means of a PLC-based real-time control of an experimental manufacturing system.

The remainder of this paper is organized as follows: Section 2 provides preliminary information about SCT; the assignment of actions to the related states of RW supervisors; the conversion of an untimed RW supervisor into LLD code; the concept of postponed event; and the implementation of postponed event. The proposed approach is presented in Section 3. The applicability of the proposed approach is demonstrated by means of the PLC-based real-time control of an experimental manufacturing system in Section 4. Finally, conclusions are given in Section 5.

2 Preliminaries

2.1 Supervisory control theory

Supervisory control theory (SCT) was introduced to extend control theory concepts for continuous systems to the discrete event environment [1, 2]. In SCT, a *forbidden state problem* [2] specifies conditions that must be avoided, typically the simultaneous utilization of some resource by two or more users; in addition SCT generally requires the controlled system to be nonblocking (namely that specified target states, often just the initial state, be maintained reachable), and to be *maximally permissive*, i.e., to permit the occurrence of all events not leading to violation of the foregoing requirements. Discrete event systems evolve on spontaneously occurring events. Let Σ be finite set of events. The set of all finite concatenations of events in Σ

is denoted by Σ^* . An element of this set is called a string. The number of events gives the length of the string. The string with no element is denoted by ϵ and is called empty string. A subset $L \subseteq \Sigma^*$ is called a language over Σ . For a string $s \in \Sigma^*$, \bar{s} denotes the prefixes of s and is defined as $\bar{s} = \{s_p \in \Sigma^* | \exists t \in \Sigma^* (s_p t = s)\}$. Extension of this definition to language prefix closure of a language L is denoted by \bar{L} . A language L satisfying the condition $L = \bar{L}$ is said to be prefix closed [29].

An automaton, denoted by G , is a sextuple $G=(Q, \Sigma, f, \Gamma, q_0, Q_m)$ where, Q is the set of states, Σ is the finite event set, $f: Q \times \Sigma \rightarrow Q$ is the partial transition function. $\Gamma: q \rightarrow 2^\Sigma$ is the active event function. $\Gamma(q)$ is the set defined for every state of G and represents the feasible events of q . q_0 is the initial state and $Q_m \subseteq Q$ is the set of marked states representing the completion of a given task or operation. A simple automaton model with two states is shown in Fig. 1. This automaton has two states labeled with q_0 and q_1 . q_0 with a double arrow is the initial (and marked) state, while q_1 with an exiting arrow is the marked state of the automaton. A directed arrow represents the transition functions of the automaton. Labels of transitions (e_1, e_2) correspond to events.

The language generated by G is denoted by $L(G)$ and is defined as $L(G) = \{s \in \Sigma^* : f(q_0, s) \text{ is defined}\}$. The language marked by G is denoted by $L_m(G)$ and is defined as $L_m(G) = \{s \in \Sigma^* : f(q_0, s) \in Q_m\}$. The DES modeled as automaton G is said to be nonblocking if $L(G) = \bar{L_m(G)}$. DESs named as A and B can be composed with synchronous product (Parallel Composition) operation. Synchronous product of two automata is denoted by $A \parallel B$ and it represents the synchronous behavior of two automata. In the resulting automaton, common events occur synchronously, while the other events occur asynchronously [29].

SCT makes use of formal languages to model the uncontrolled behavior of discrete event systems (plant) and specifications for the controlled behavior. The objective is to restrict the behavior of the system to a desired behavior, which is represented by the specifications. This is done by disabling some events to prevent the occurrence of some undesired strings in the system. The disabling action is accomplished by another simultaneously executing automaton called the supervisor. The system cannot be forced by supervisor to generate new events. In SCT, events are divided into two disjoint sets, the controllable events and

uncontrollable events. These sets are denoted by Σ_c and Σ_{uc} respectively. The supervisor has no effect on uncontrollable events, which means that the supervisor cannot disable uncontrollable events. The existence of a supervisor is guaranteed if the desired language satisfies controllability conditions. This condition is defined as $\bar{K} \Sigma_{uc} \cap M \subseteq \bar{K}$, where \bar{K} is the language that will be generated under the control of supervisor and M is the language generated by the uncontrolled system.

2.2 The assignment of actions (Output Signals) to the related states of RW supervisors

PLC-based implementation of RW type supervisors require some actions (output signals) to be assigned to some states of the supervisor. To automate this process a general methodology is proposed in [25]. This section briefly recalls this method. The following describes the implementation philosophy of an action assigned to a supervisor state: if an action(s) assigned state is active then the assigned action (output signal) is used to activate the actuator(s) within the system. The implementation is very simple but it was not clear how to assign an action to a state and to keep record of the action assigned state when carrying out the necessary computations of SCT. This problem is solved in [25] as follows: firstly actions to be assigned and the respective states are defined at the beginning of the design steps. Then, in order to represent an action assignment, an uncontrollable event is added as a selfloop to a corresponding state (note that selfloop in an automaton is a transition whose source and target state is the same). This is really important because when using the TCT or other computation means for SCT it is necessary to track the information about the action assigned states all along. Therefore, a selfloop is added to a state to represent the assignment of an action. At the end of the design steps the corresponding selfloops are replaced with their related actions on supervisor states. The role of selfloops for representing actions in the computation of RW supervisors may be considered to be similar to the role of a catalyst in a chemical reaction. A catalyst is something that makes a chemical reaction happen faster and in the end of the chemical reaction it remains as it is. Similarly, the use of selfloops for representing actions in the computation of RW supervisors helps one to track the information about the action assigned states and in the end these selfloops are replaced with the corresponding actions.

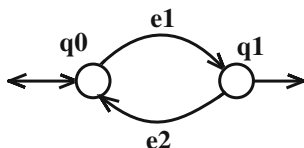


Fig. 1 A simple automaton model

2.3 Conversion of a RW supervisor into LLD code

The conversion of an untimed RW supervisor assigned with actions into LLD for PLC implementation is a straight forward process. From the literature one can find some

well-established methods. The following briefly explains the method utilized in this paper: a Boolean variable (a memory bit) is assigned to each state of the RW supervisor. Initially, the Boolean variable representing the active (initial) state is SET and all other Boolean variables representing the rest of the RW supervisor states are RESET. Then, each transition between states of the RW supervisor is implemented as a separate PLC ladder rung by using SET and RESET commands. The *actions* are implemented at the end of the LLD program. If an action(s) assigned state is active then the assigned action (output signal) is used to activate the actuator(s) within the system. In PLC implementation, an *event* can be defined as a rising or a falling edge of a Boolean variable. For such definition $\neg|P|$ and $\neg|M|$ contacts can be used respectively. Then, the straightforward PLC implementation of a transition say $[q_0, e_1, q_1]$ with an event e_1 shown in Fig. 2a is obtained in LLD language as shown in Fig. 2c. This code simply implements the following statement: “if state q_0 is active and the event e_1 occurs then set state q_1 and reset state q_0 ”.

2.4 The concept of postponed event

Based on the automaton definition given above the postponed event concept is proposed in [27] for introducing the time delay functions within untimed automata. The concept of postponed event is defined as follows:

$$pe = e + \tau$$

Where e is an event, pe is the postponed event related to event e , τ is a deterministic time delay value and “+” is the shift (postponement) operator applied to event e on the time axis. In this framework, the instantaneous occurrence of event e starts off the deterministic time delay function. When the elapsed time reaches the time specified by τ , the postponed event pe occurs instantaneously. In other words the postponed event pe is the shifted version of the event e on the time axis by τ . For the sake of simplicity, it is assumed that once the time delay function starts off for an event e , no more occurrence of e will take place until the elapsed time reaches the time specified by τ . With this easy to use structure, timed discrete event systems can be defined based on the untimed versions. It can easily be seen that if

$\tau=0$ then postponed event pe is the same as event e . It is not difficult to see that pe is a deterministic time-dependent version of e .

The difference between an event and a postponed event can be seen by considering the timing diagrams of both. Fig. 2a depicts a simple automaton with two states (q_0 and q_1) and a transition $[q_0, e_1, q_1]$ with an event e_1 . Fig. 2b is the timing diagram of this automaton. When q_0 is active and the instantaneous occurrence of event e takes place, the automaton transits from q_0 to q_1 . Fig. 3a shows an automaton with two states (q_1 and q_2) and a transition $[q_1, pe_2, q_2]$ with a postponed event pe_2 . When q_1 is active and the instantaneous occurrence of postponed event pe_2 takes place, the automaton transits from q_1 to q_2 . As can be seen from the timing diagram of this automaton in Fig. 3b the occurrence of pe_2 (t_2) depends upon the occurrence of e_2 (t_1) and the deterministic time delay value τ_2 . It is interesting to observe that the occurrence of pe_2 at time t_4 does not cause any state change because at time t_4 the active state is q_2 . Another important observation is that if e_2 is removed from the timing diagram of Fig. 3b the timing diagram then becomes very similar to that of given in Fig. 2b. In this case pe_2 of Fig. 3b plays the same role of e_1 in Fig. 2a. Therefore, when one would like to design a DES with timing requirements in the automata framework it is possible to design the system as if it is an untimed DES. This is especially useful when designing DES controllers with timing requirements. In fact this idea has already been applied by using untimed RW framework as shown in this paper. A DES control problem with time delay requirements is defined by using the postponed event concept and it is solved in untimed RW framework. Then, the postponed events are refined and the related DES controller is implemented successfully.

2.5 Implementation of the postponed event

The implementation of a postponed event with a PLC can be performed by using the standard timer functions, namely the on delay timer, the off-delay timer (TOF), the pulse timer (TP), defined by the IEC 61131–3 standard [30]. These three implementations are proposed in [27]. In this paper the implementation of the postponed event with the TOF is

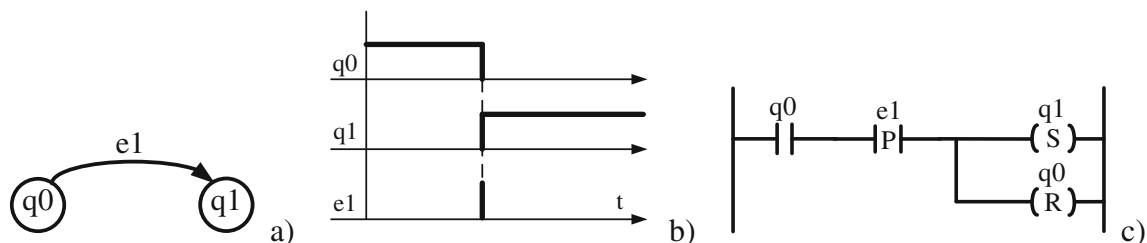
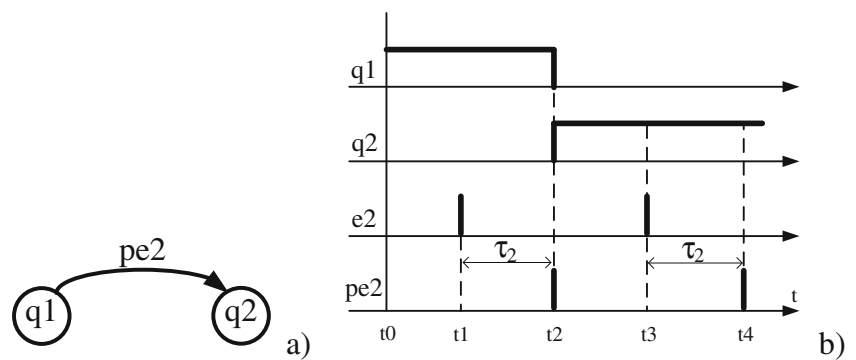


Fig. 2 a An event e_1 in the transition $[q_0, e_1, q_1]$, b the timing diagram, c straight forward LLD implementation of transition $[q_0, e_1, q_1]$.

Fig. 3 a postponed event pe2 in the transition [q1,pe2,q2], b its timing diagram



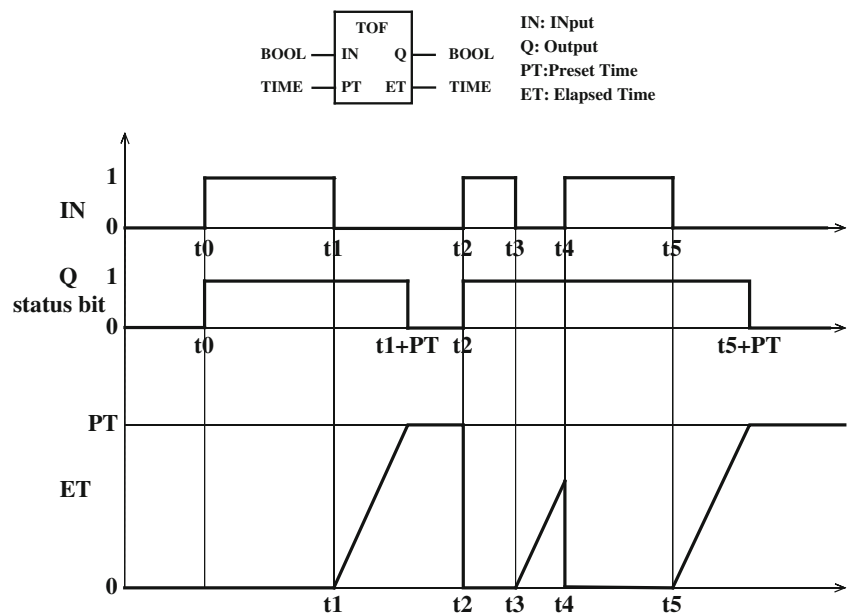
recalled from [27]. Some symbols of IEC 61131–3 LD language used in this paper are provided in Table 1. Before establishing the implementation of the postponed event, the off-delay timer defined by the IEC 61131-3 standard is considered in the following: *Off Delay Timer*: the off-delay timer can be used to delay setting an output false (OFF→0),

for a fixed (deterministic) period of time after an input signal goes false (OFF→0), i.e., the output is held ON for a given period longer than the input. The generic symbol and the timing diagram of the TOF, defined by the IEC 61131-3 standard, are both shown in Fig. 4. As the input signal IN goes true (ON→1), the output Q (status bit) follows and

Table 1 Some symbols of IEC 61131-3 LD language

Symbol	Description
I	<i>Input location</i>
Q	<i>Output location</i>
M	<i>Memory location</i>
	<i>Normally open contact</i>
	The state of the left link is copied to the right link if the state of the associated Boolean variable (indicated by ***) is ON. Otherwise, the state of the right link is OFF.
	<i>Normally closed contact</i>
	The state of the left link is copied to the right link if the state of the associated Boolean variable (indicated by ***) is OFF. Otherwise, the state of the right link is OFF.
	<i>Positive transition-sensing contact</i>
	The state of the right link is ON from one evaluation of this element to the next when a transition of the associated Boolean variable (indicated by ***) from OFF to ON is sensed at the same time that the state of the left link is ON. The state of the right link shall be OFF at all other times.
	<i>Negative transition-sensing contact</i>
	The state of the right link is ON from one evaluation of this element to the next when a transition of the associated Boolean variable (indicated by ***) from ON to OFF is sensed at the same time that the state of the left link is ON. The state of the right link shall be OFF at all other times.
	<i>Coil</i>
	The state of the left link is copied to the associated Boolean variable (indicated by ***) and to the right link.
	<i>SET (latch) coil</i>
	The associated Boolean variable (indicated by ***) is set to the ON state when the left link is in the ON state, and remains set until reset by a RESET coil.
	<i>RESET (unlatch) coil</i>
	The associated Boolean variable (indicated by ***) is reset to the OFF state when the left link is in the ON state, and remains reset until set by a SET coil.

Fig. 4 The symbol and the timing diagram of the off-delay timer (TOF) defined by the IEC 61131-3 standard



remains true (ON—1), until the input signal IN is false (OFF—0) for the period specified in preset time input PT. As the input signal IN goes false (OFF—0), the elapsed time ET starts to increase. It continues to increase until it reaches the preset time input PT, at which point the output Q (status bit) is set false (OFF—0) and the elapsed time is held. If the input signal IN is false (OFF—0) only for a period shorter than the input PT, the output Q remains true (ON—1).

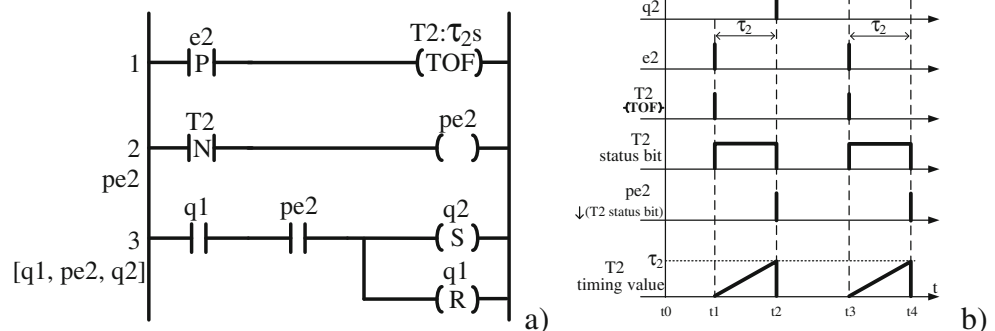
Now, the implementation of a postponed event by using a TOF is considered. The example implementation explained here is for the automaton shown in Fig. 3a. The LLD implementation of the automaton shown in Fig. 3a by using a TOF is provided in Fig. 5 together with its timing diagram. Due to the characteristics of TOF shown in Fig. 4, with the occurrence of $e2$, the coil of TOF is energized (rung 1) instantaneously and at the same time the status bit T2 is set (rung 1—time $t1$). As $e2$ goes OFF, the timing value (elapsed time) starts to increase (between $t1$ and $t2$). It continues to increase until it reaches $\tau2$, at which point the

status bit T2 is set false. $pe2$ is equal to the falling edge of T2 (rung 2—time $t2$). If state $q1$ is active and $pe2$ is ON then state $q2$ is set and $q1$ is reset (rung 3—time $t2$). This implementation requires three LLD rungs and one additional $|\bar{N}|$ -contact.

3 The proposed approach for the PLC-based implementation of RW Supervisors with time delay functions

The supervisory control of a discrete event system (DES) is illustrated in Fig. 6. The architecture consists of four parts: (a) the *discrete event system (Plant)* to be controlled, (b) the *control system (RW supervisor)*, as the controller (c) *sensor readings*, regarded as outputs from the DES and as inputs to the RW supervisor, and (d) *control actions*, regarded as output signals from the RW supervisor and as inputs to the *Plant*. The RW supervisor must guarantee that no forbidden

Fig. 5 a LLD implementation of $pe2$ by using an off-delay timer (TOF), **b** its timing diagram



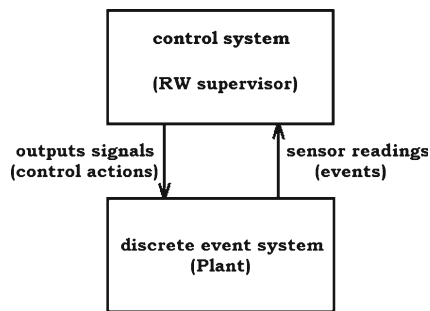


Fig. 6 Supervisory control of a DES

state will be reached, that specified target states remain reachable (nonblocking) and that controlled behavior is maximally permissive, i.e., the supervisor does not unnecessarily constrain system operation and is in this sense “optimal”.

The plant and the supervisor are assumed to run concurrently, as follows. The occurrence of an event in the plant is transmitted to the supervisor as a plant output through sensory feedback, resulting in a supervisor state change. In other words, when an event occurs in the plant, the supervisor changes its state synchronously, in accordance with the active state and the event in question. The supervisor functions as a state-feedback controller, whose control actions are output signals from the control system and inputs to the plant, closing the feedback loop. The controlled behavior of the plant will be the subset of uncontrolled behavior (i.e., sublanguage of uncontrolled event strings) that survives under supervision.

As the supervisory control theory is used to obtain the supervisor, the proposed approach offers the following advantages: the supervisor and control policy obtained are correct by construction, i.e., controlled behavior of the system is *nonblocking* and does not contradict with the forbidden state specifications. All events that do not contradict the forbidden state specifications are allowed to happen, i.e., the controlled behavior of the system is *maximally permissive* within the specifications.

There are a number of software tools to compute RW supervisors. In this paper, TCT [28] is utilized to carry out necessary computations to obtain RW supervisors. The proposed general methodology includes the action assignment procedure, handling of timed events as if they are normal events within the untimed SCT framework using the concept of postponed events, and finally the implementation of the resulting RW supervisor by using a PLC. As a result, the following provides the proposed design and implementation methodology for the supervisory control of DES:

1. Model each system (plant) component as an automaton (G_1, G_2, \dots, G_N) and define these

automata models within TCT by using the Create (.) procedure:

$G_1 = \text{Create}(G_1)$

$G_2 = \text{Create}(G_2)$

⋮

$G_N = \text{Create}(G_N)$

Plant components may include system parts such as machines, robots, conveyors, motors, solenoids, etc. A proper automaton model must be constructed with its states and transitions. Some general forms of automata models for different system parts can be seen from [29]. For the implementation level it is also important to define the events and actions. Events are assigned to transitions to define the conditions under which the state change occurs from one state to another. Actions are assigned to states and they define when the actuators will be switched on. Although there are different systems with different models it is important to explain how to obtain actuator models. Actuators include motors, solenoids, etc. This is important because with the model type introduced here, the PLC implementation of RW supervisors act as if they are forcing controllers. The idea used to obtain actuator models is as follows: normally they have two states, namely “on” and “off”. In general, the initial state is off and the other state is on. In our design method going from the initial state to the other state is automatic unless blocked by the supervisor. This is a controllable event and the condition for this event assigned to this transition is “1”. Based on the specifications, the computed RW supervisor will define under which conditions to stop this controllable event. This mechanism obeys the RW supervisor’s role in its original definition. In this step, timed events are also identified and then they are considered as untimed events.

2. Define actions to be assigned and the corresponding automaton states.

Before the computation of RW supervisor actions are assigned to the related states of the actuator models. It is obvious that the resulting RW supervisor will be a Moore-type finite state machine because the actions (outputs) are defined as assigned on the states. In other words, outputs are the function of states only.

3. Add an uncontrollable event as a selfloop to a corresponding state in order to represent an action assignment.

The reason for this assignment is provided in Section 2.2.

4. Obtain the plant automaton model (PLANT) with parallel composition (also called synchronous composition) of plant components ($PLANT = G1||G2||\dots||GN$) by using Sync(.) procedure:

```
PLANT = Sync(G1, G2)
PLANT = Sync(PLANT, G3)
.
.
PLANT = Sync(PLANT, GN)
```

This step is necessary to obtain all possible interactions within the system components. The plant obtained will be the state space of the modeled system.

5. Model each specification component as an automaton (SPEC1, SPEC2, ..., SPECM) by using Create(.) procedure:

```
SPEC1 = Create(SPEC1)
SPEC2 = Create(SPEC2)
.
.
SPECM = Create(SPECM)
```

In this step, it is necessary to declare the desired system operation by means of automata models. Some specification automata models can be seen from [29].

6. Adjoin to each state of SPEC i ($i=1, 2, \dots, M$) all events that are possible in PLANT but not constrained by the SPEC i ($i=1, 2, \dots, M$) by using Selfloop(.) procedure:

```
SPEC1 = Selfloop(SPEC1, all events that are possible in
PLANT but not constrained by the SPEC1)
SPEC2 = Selfloop(SPEC2, all events that are possible in
PLANT but not constrained by the SPEC2)
.
.
SPECM = Selfloop(SPECM, all events that are possible in
PLANT but not constrained by the SPECM)
```

The monolithic RW supervisor is obtained by using a PLANT automaton model and a SPEC automaton model. To obtain a single composed specification automaton model (SPEC) from all specification models, firstly all events that are possible in PLANT but not constrained by a specification must be adjoined to that specification automaton model.

7. Obtain a composed specification model (SPEC) from all specification components (SPEC1, SPEC2, ..., SPECM) by using Meet(.) procedure:

```
SPEC = Meet(SPEC1, SPEC2)
SPEC = Meet(SPEC, SPEC3)
.
.
SPEC = Meet(SPEC, SPECM)
```

8. Obtain the RW supervisor, say SUPER, by using Supcon(.) procedure:

```
SUPER = Supcon(PLANT, SPEC)
```

The RW supervisor SUPER is of a monolithic type. The supervisory control theory, and its TCT implementation, provides a maximally permissive and nonblocking controller at this step. In other words, the obtained RW supervisor SUPER is correct by construction.

9. Replace the corresponding selfloops with their related actions on supervisor states.

At this design step the corresponding selfloops are replaced with their related actions on supervisor states. This is just for converting the resulting RW supervisor SUPER into a Moore-type automaton. Because the RW supervisor SUPER contains only states and transitions. In order to implement the SUPER on a PLC using LLD the outputs will be defined after this process.

10. Declare timed events as postponed events.

Similar to the previous step now timed events are refined before the PLC implementation. Up to this design step they are all considered as normal events but before the PLC implementation of the resulting supervisor they are treated as postponed events. The supervisor after this process is called “the RW supervisor with postponed events and actions assigned to places”

11. Convert the RW supervisor with postponed events and actions assigned to places into LLD code for PLC implementation. The design phase is only the first step towards the control of DESs. After designing a controller (supervisor), it is necessary to have an automatic means for the generation of control code from the controller. The supervisory control can be enforced by implementing the supervisor on an industrial computer. Since programmable logic controllers (PLCs) with a graphical symbolic programming language, called LLDs, are the most popular implementation tools in today’s automated modern factories, in this paper the techniques described in the previous

sections are used for converting the RW supervisor (with postponed events and actions assigned to places) into an LLD code for implementation on a PLC.

The summary of the above explained design steps is that the abstract RW supervisor model is refined by the intermediate design steps included within the original supervisory control theory in order to obtain implementable model before the PLC-based implementation.

4 Real-time supervisory control of an experimental manufacturing system

In this section, an experimental manufacturing system is considered to show the applicability of the proposed method to low-level real-time supervisory control of discrete event systems. Firstly, the experimental manufacturing system is explained together with the control specifications as a DES control problem. Then, this problem is solved by the proposed method explained in the previous section.

4.1 Problem description

The experimental manufacturing system, shown in Fig. 7, represents a component sorting and assembly processes that can be controlled by virtually any controller such as micro-controller or PLC. The upper conveyor and the lower conveyor are driven by the upper conveyor motor (Actuator 1) and the lower conveyor motor (Actuator 2), respectively. A random selection of metallic pegs and plastic rings are placed on the upper conveyor. The rings and pegs need to be identified and separated. This is done by two sensors, a proximity sensor (Sensor 1) and an infra-red reflective sensor (Sensor 2). By using these two sensors, a distinction can be made between the peg and the ring. By means of the sort solenoid (Actuator 3), plastic rings can be ejected down the assembly chute, which can have up to five plastic rings. Metallic pegs, meanwhile, continue on the upper conveyor and are deflected down the feeder chute. The feeder chute automatically feeds pegs onto the lower conveyor. An infra-red emitter/detector (Sensor 3) is used to determine whether or not the assembly area is empty. If it is, the assembly solenoid (Actuator 4) is used to dispense a ring from the assembly chute into the assembly area. The assembly area is positioned just above the lower conveyor and, when a metallic peg passes, the peg engages with the hole in the ring and the two components are assembled. The lower conveyor is used to carry completed components into the collection tray. In this work, a Siemens S7-300 (CPU319)

PLC is used to control the process, and software called “Simatic Manager” is used to program the PLC. PLC inputs and outputs are given in Tables 2 and 3, respectively.

For the sake of simplicity, it is assumed that the assembly chute can have only one ring at a time. It is also assumed that when the system is switched on, both the upper conveyor motor and the lower conveyor motor are switched on automatically. The control specifications are as follows:

1. Operate the sort solenoid only when there is a ring at the sort area and there is space in the assembly chute.
2. Operate the assembly solenoid only when there is a ring in the assembly chute and there is space at the assembly area.

4.2 Design and implementation steps for solving the control problem

The problem stated in the previous section is a DES control problem and is solved by the proposed method as described in this section. Now the design and implementation steps of the proposed method are followed.

4.2.1 Step 1

For the stated problem, the automata models are obtained as shown in Fig. 8 for the sort area (SA), the assembly chute (AC), the assembly area (AA), the sort solenoid (SS), and the assembly solenoid (AS). Event labels used in these automata models and their descriptions together with the related signals are provided in Table 4. Events 32, 42, and 52 are timed events with time delays 1, 0.7, and 1.5 s, respectively. Now, a brief explanation for each of these models is given as follows:

SA States s_0 and s_1 represent the *absence* and the *presence* of a plastic ring at the sort area respectively. Initially it is assumed that there is no plastic ring at the sort area (s_0 is active). When the presence of a ring is detected at the sort area ($\overline{I0.1} \wedge I0.0 \uparrow$), the event 22 occurs and the automaton goes from s_0 to s_1 . When there is a plastic ring at the sort area (s_1 is active) there are two possibilities for the plastic ring. In the first possibility, if the sort solenoid is activated ($Q0.0=1$) then the plastic ring is being pushed into the assembly chute by the sort solenoid. It takes 1 s for the plastic ring to move from the assembly area into the assembly chute. After this time has elapsed the event 32 occurs ($Q0.0 \wedge \overline{I0.1} \wedge I0.0 \downarrow, 1s$) and the automaton goes from s_1 to s_0 . Secondly, if the sort solenoid is not activated ($Q0.0=0$) and the absence of the plastic ring is detected ($\overline{I0.1} \wedge I0.0 \downarrow$) then the event 12 occurs and the automaton

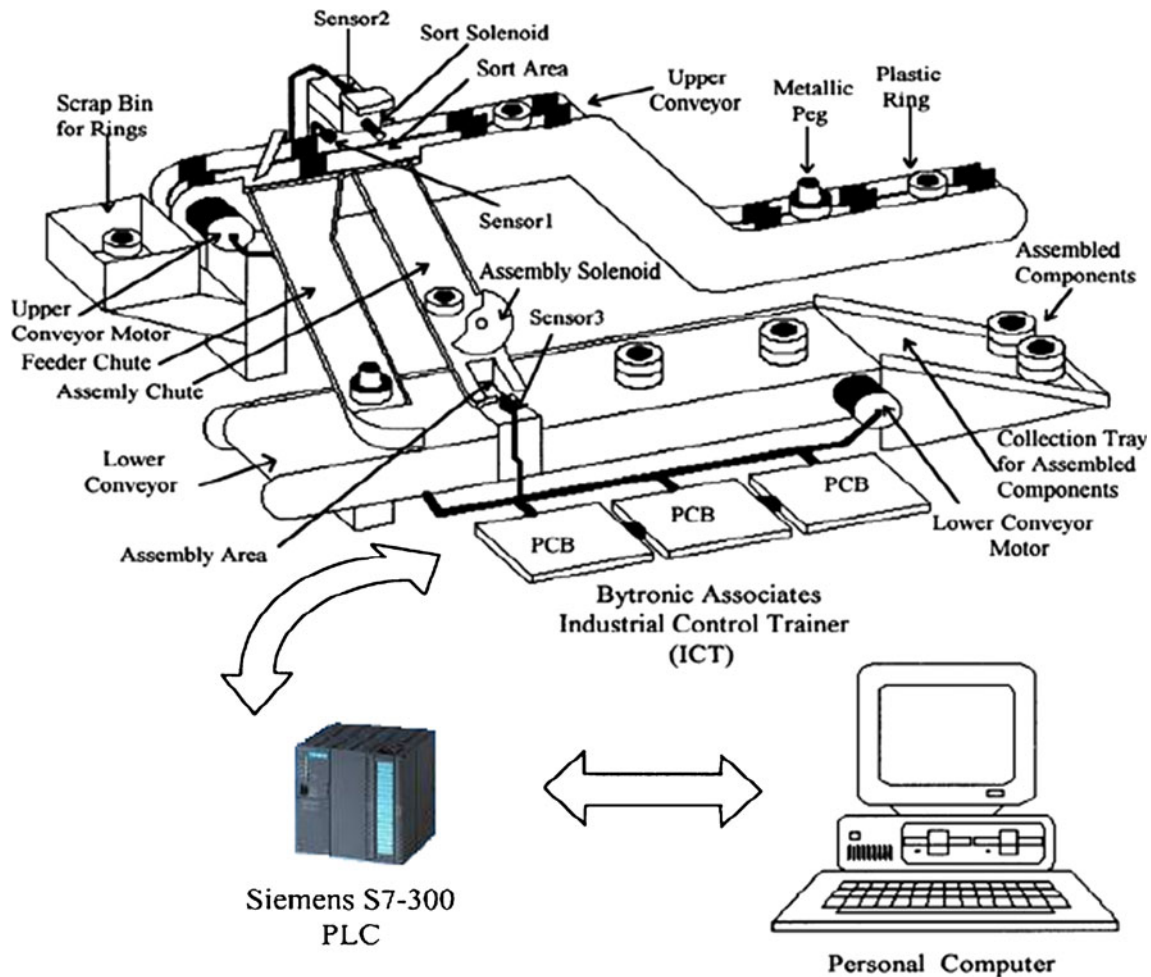


Fig. 7 The experimental manufacturing system

goes from $s1$ to $s0$. This means that the plastic ring has just left the sort area towards the scrap bin for rings.

AC States $s0$ and $s1$ represent the *absence* and the *presence* of a plastic ring in the assembly chute respectively. Initially it is assumed that there is no plastic ring in the assembly chute ($s0$ is active). When the event 32 occurs ($Q0.0 \wedge \overline{I0.1} \wedge I0.0 \downarrow, 1s$), a plastic ring is put into the assembly chute and the automaton goes from $s0$ to $s1$. When there is a plastic ring in the assembly chute ($s1$ is active), if the assembly solenoid is activated ($Q0.1=1$) then the plastic

ring is being moved from the assembly chute into the assembly area by the assembly solenoid. This process takes 0.7 s. After this time has elapsed, the event 42 occurs ($I0.2 \uparrow, 0.7 s$) and the automaton goes from $s1$ to $s0$.

AA States $s0$ and $s1$ represent the *absence* and the *presence* of a plastic ring in the assembly area respectively. Initially it is assumed that there is no plastic ring in the assembly area ($s0$ is active). When the event 42 occurs ($I0.2 \uparrow, 0.7 s$), a plastic ring is put into the assembly area and the automaton goes from $s0$ to $s1$. When there is a plastic ring in the

Table 2 PLC inputs

PLC Input	Sensor	Definition
I0.0	Sensor 1	Detects a ring or a peg at the sort area
I0.1	Sensor 2	Detects a peg at the sort area
I0.2	Sensor 3	Detects a ring in the assembly area

Table 3 PLC outputs

PLC Outputs	Actuator numbers	Definition
Q0.0	Actuator 1	Sort solenoid
Q0.1	Actuator 2	Assembly solenoid
Q0.3	Actuator 3	Upper conveyor motor
Q0.4	Actuator 4	Lower conveyor motor

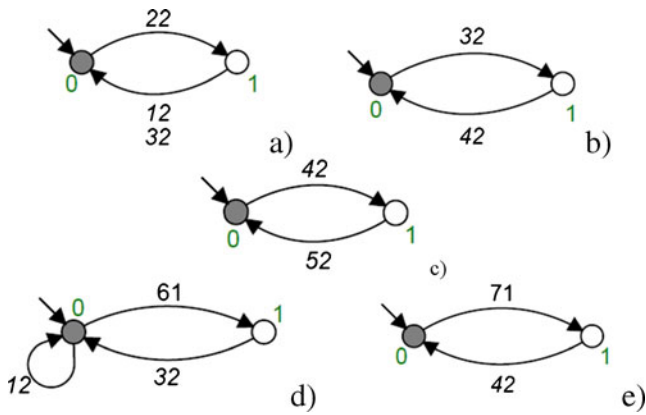


Fig. 8 Automata models for: **a** sort area (SA) **b** assembly chute (AC) **c** assembly area (AA) **d** sort solenoid (SS) **e** assembly solenoid (AS)

assembly chute ($s1$ is active), if the absence of the plastic ring is detected ($I0.2\downarrow$) then the plastic ring is being leaving the assembly area. This process takes 1.5 s. After this time has elapsed the event 52 occurs ($I0.2\downarrow$, 1.5 s) and the automaton goes from $s1$ to $s0$.

SS States $s0$ and $s1$ represent *off* and *on* states of the sort solenoid, respectively. Initially, it is assumed that the sort solenoid is off ($s0$ is active). When the decision is made to activate the sort solenoid (event 61 occurs), the automaton goes from $s0$ to $s1$. When the sort solenoid is *on* ($s1$ is active and $Q0.0=1$), if the absence of a plastic ring is detected at the sort area ($\overline{I0.1} \wedge I0.0 \downarrow$) then it takes 1 s for a plastic ring to move from the assembly area into the assembly chute. After this time has elapsed, the plastic ring is put into the assembly chute (the event 32 occurs) and thus the automaton goes from $s1$ to $s0$. At the sort area, in order to distinguish a plastic ring going into the scrap bin from the one being pushed into the assembly chute the event 12 is adjoined to $s0$ as a selfloop.

AS States $s0$ and $s1$ represent *off* and *on* states of the assembly solenoid, respectively. Initially, it is assumed that the assembly solenoid is off ($s0$ is active). When the decision is made to activate the assembly solenoid (event 71 occurs), the automaton goes from $s0$ to $s1$. When the assembly solenoid is *on* ($s1$ is active and $Q0.1=1$), if the presence of a plastic ring is detected ($I0.2\uparrow$) at the assembly area then it takes 1 s for a plastic ring to move from the assembly chute into the assembly area. After this time has elapsed, the plastic ring is put into the assembly area (the event 42 occurs) and thus the automaton goes from $s1$ to $s0$.

The first three automata models are defined within TCT by using the Create(.) procedure as follows:

```
SA = Create(SA, [mark0], [tran[0, 22, 1], [1, 12, 0], [1, 32, 0]]) (2, 3)
AA = Create(AA, [mark0], [tran[0, 42, 1], [1, 52, 0]]) (2, 2)
AC = Create(AC, [mark0], [tran[0, 32, 1], [1, 42, 0]]) (2, 2)
```

4.2.2 Step 2

The SS and the AS automata models are identified as automata models to be assigned with actions, because they are related to the operation of the related solenoids. When the SS automaton model shown in Fig. 8d is considered, it can be seen that it is appropriate to assign an action to the state 1, because when the SS automaton model arrives to this state with the occurrence of the event 61, the assigned action will force the sort solenoid to work. Therefore, the action SSa is assigned to the state 1 of the SS automaton model as can be seen from Fig. 9a. In the same manner, to represent the assembly solenoid operation, the action ASa is assigned to the state 1 of the AS automaton model as can be seen from Fig. 9b

Table 4 Event labels and their description together with related signals

Event labels	Description	Related signals
12	A plastic ring has just left the sort area towards the scrap bin for rings	$\overline{Q0.0} \wedge \overline{I0.1} \wedge I0.0 \downarrow$
22	The presence of a plastic ring is detected at the sort area	$I0.1 \wedge I0.0 \uparrow$
32	A plastic ring is being put into the assembly chute	$Q0.0 \wedge \overline{I0.1} \wedge I0.0 \downarrow$, 1 s.
42	A plastic ring is being moved from the assembly chute to the assembly area	$I0.2\uparrow$, 0.7 s
52	A plastic ring is being leaving the assembly area	$I0.2\downarrow$, 1.5 s
60	Sort solenoid action	–
61	Activate sort solenoid	1
70	Assembly solenoid action	–
71	Activate assembly solenoid	1

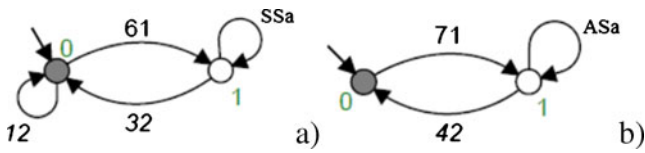


Fig. 9 a The sort solenoid automaton model with assigned action SSa. b The assembly solenoid automaton model with assigned action ASa

4.2.3 Step 3

Actions SSa and ASa are represented by two uncontrollable events, namely 60 and 70, respectively, as shown in Fig. 10 to be included within TCT computations.

Next, the SS and AS automata models with assigned actions are defined within TCT by using the Create(.) procedure as follows:

```
SS = Create(SS, [mark0], [tran[0, 12, 0], [0, 61, 1], [1, 32, 0], [1, 60, 1]]) (2, 4)
AS = Create(AS, [mark0], [tran[0, 71, 1], [1, 42, 0], [1, 70, 1]]) (2, 3)
```

4.2.4 Step 4

The PLANT automaton model is obtained with the parallel composition of all system components (PLANT = SA||AA||AC||SS||AS) by using Sync(.) procedure as follows and it is shown in Fig. 11:

```
PLANT = Sync(SA, AA) (4, 10) Blocked_events = None
PLANT = Sync(PLANT, AC) (8, 16) Blocked_events = None
PLANT = Sync(PLANT, SS) (16, 42) Blocked_events = None
PLANT = Sync(PLANT, AS) (32, 112) Blocked_events = None
```

4.2.5 Step 5

Now the automata models of the two control specifications are considered. The first specification states the following: “Operate the sort solenoid only when there is a ring at the sort area and there is space in the assembly chute”. It may be

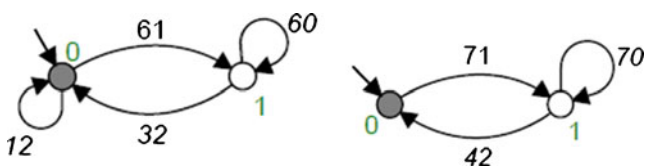


Fig. 10 Representation of actions SSa and ASa by two uncontrollable events “60” and “70”

re-stated as follows: “Operate the sort solenoid when there is a ring at the sort area AND Operate the sort solenoid when there is space in the assembly chute”. The first part of the first specification, namely “Operate the sort solenoid when there is a ring at the sort area” is represented as an automaton model, called SPEC1A, shown in Fig. 12a. Similarly, the second part of the first specification, namely “Operate the sort solenoid when there is space in the assembly chute” is represented as an automaton model, called SPEC1B, shown in Fig. 12b. The following Create (.) TCT procedures are carried out for defining both SPEC1A and SPEC1B:

```
SPEC1A = Create(SPEC1A, [mark0], [tran[0, 22, 1],
[1, 12, 0], [1, 32, 0], [1, 61, 1]]) (2, 4)
SPEC1B = Create(SPEC1B, [mark0], [tran[0, 32, 1],
[0, 61, 0], [1, 42, 0]]) (2, 3)
```

The “AND” statement of the specification is accomplished by applying the synchronous product procedure Sync(.) of TCT to SPEC1A and SPEC1B and thus yielding the SPEC1 as follows:

```
SPEC1 = Sync(SPEC1A, SPEC1B) (4, 8) Blocked_events = None
```

Now the second specification stating that “Operate the assembly solenoid only when there is a ring in the assembly chute and there is space at the assembly area” is considered. The second specification may be re-stated as follows: “Operate the assembly solenoid when there is a ring at the assembly chute AND Operate the assembly solenoid when there is space at the assembly area”. The first part of the second specification, namely “Operate the assembly solenoid when there is a ring at the assembly chute” is represented as an automaton model, called SPEC2A, shown in Fig. 13a. Similarly, the second part of the second specification, namely “Operate the assembly solenoid when there is space at the assembly area” is represented as an automaton model, called SPEC2B, shown in Fig. 13b. The following TCT procedures are carried out for defining both SPEC2A and SPEC2B:

```
SPEC2A = Create(SPEC2A, [mark0], [tran[0, 32, 1],
[1, 42, 0], [1, 71, 1]]) (2, 3)
SPEC2B = Create(SPEC2B, [mark0], [tran[0, 42, 1],
[0, 71, 0], [1, 52, 0]]) (2, 3)
```

The “AND” statement of the specification is accomplished by applying the synchronous product procedure Sync(.) of TCT to SPEC2A and SPEC2B and thus yielding the SPEC2 as follows:

```
SPEC2 = Sync(SPEC2A, SPEC2B) (4, 6) Blocked events = None
```

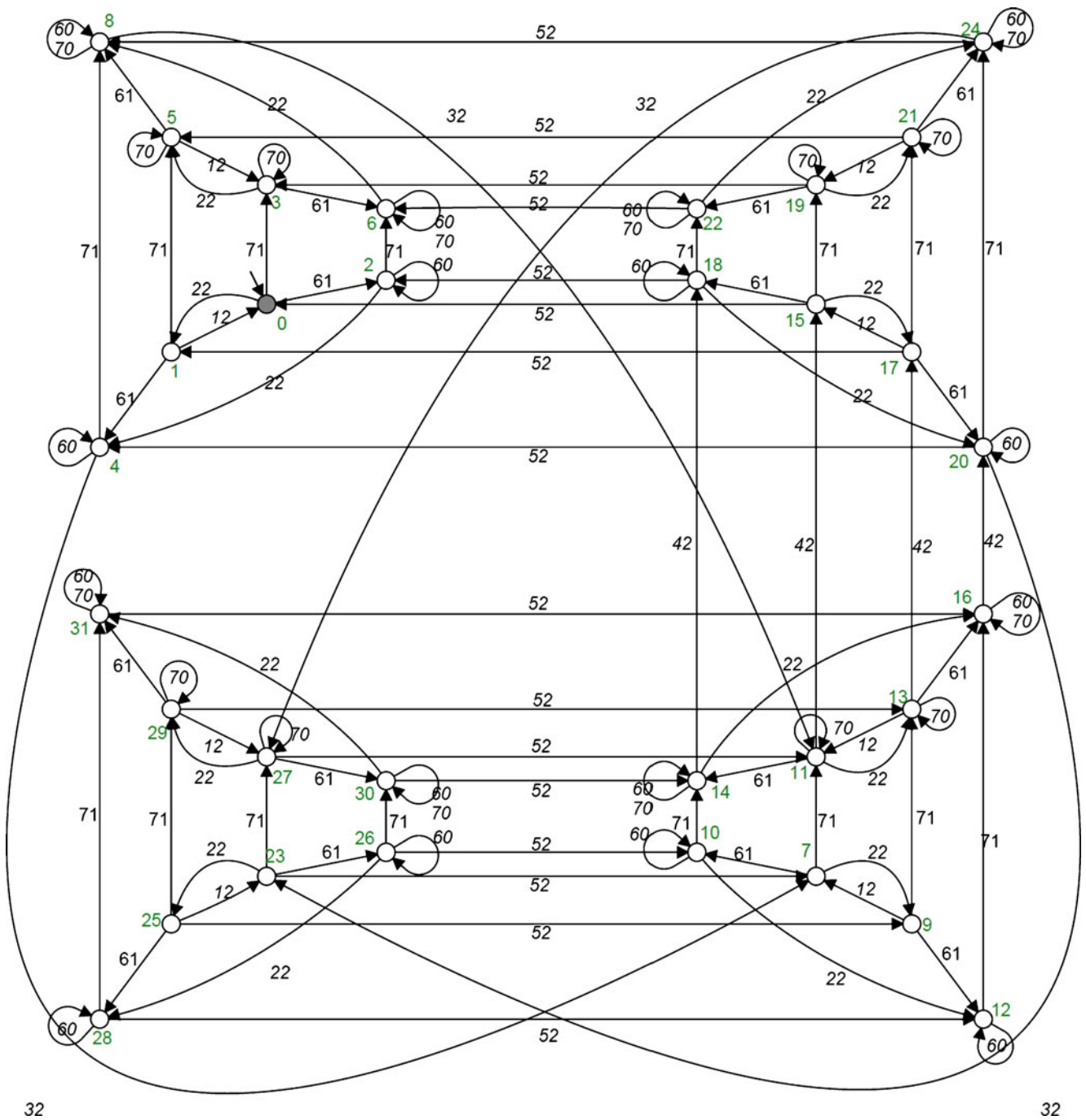



Fig. 11 PLANT automaton model

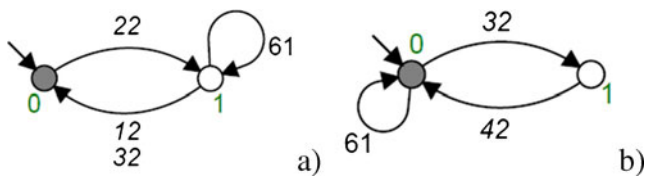


Fig. 12 Automata models for the first specification: a SPEC1A. b SPEC1B

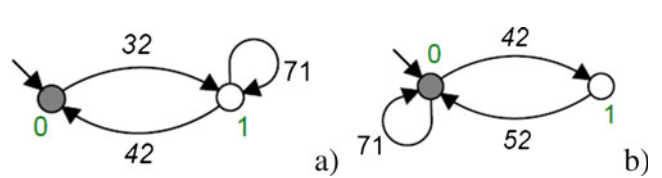


Fig. 13 Automata models for the second specification: a SPEC2A. b SPEC2B

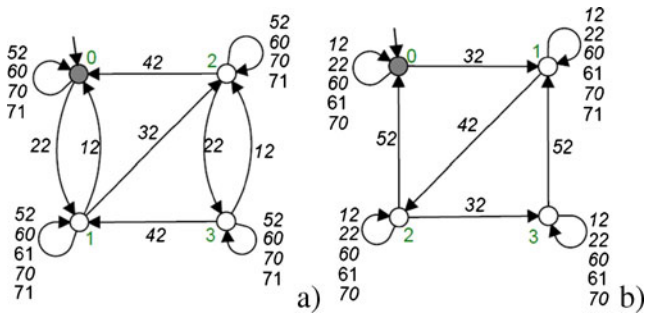


Fig. 14 Automata models for the two specifications: **a** SPEC1. **b** SPEC2

4.2.6 Step 6

In this step, all events that are possible in PLANT but not constrained by the SPEC1 and SPEC2 are adjoined to each state of these specification models. For the first specification model SPEC1, selfloops with event labels “52”, “60”, “70”, and “71” are adjoined to each state of the SPEC1 as follows:

$$\text{SPEC1} = \text{Selfloop}(\text{SPEC1}, [52, 60, 70, 71]) (4, 24)$$

As a result the automaton model SPEC1 is obtained as shown in Fig. 14a, satisfying the first specification.

For the second specification model SPEC2, selfloops with event labels “12”, “22”, “60”, “61” and “70” are adjoined to each state of the SPEC2 as follows:

$$\text{SPEC2} = \text{Selfloop}(\text{SPEC2}, [12, 22, 60, 61, 70]) (4, 26)$$

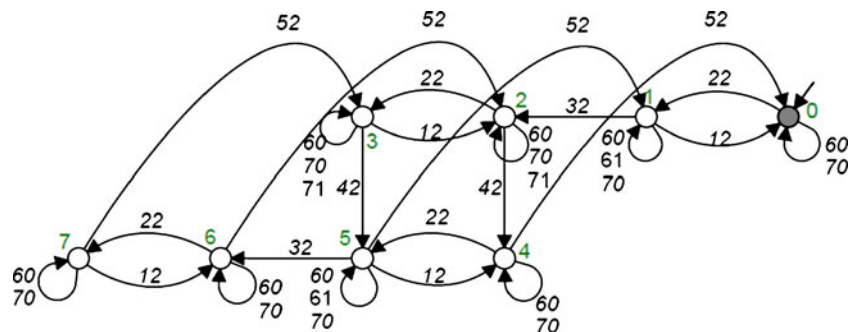
As a result the automaton model SPEC2 is obtained as shown in Fig. 14b, satisfying the second specification.

4.2.7 Step 7

To obtain a single specification, namely SPEC, from the two specifications SPEC1 and SPEC2, the Meet(.) procedure of TCT is applied as follows and the automaton model of SPEC is shown in Fig. 15:

$$\text{SPEC} = \text{Meet}(\text{SPEC1}, \text{SPEC2}) (8, 36)$$

Fig. 15 SPEC automaton model



4.2.8 Step 8

In this step, the Supervisory Control Theory (SCT) is applied to obtain a RW supervisor (SUPER). To accomplish this task, both the PLANT and the specification (SPEC) automaton models are used in the Supcon(.) procedure of TCT software as follows:

$$\text{SUPER} = \text{Supcon}(\text{PLANT}, \text{SPEC}) (12, 27)$$

As a result the SUPER automaton model which represents the complete controlled behavior [29] is depicted in Fig. 16.

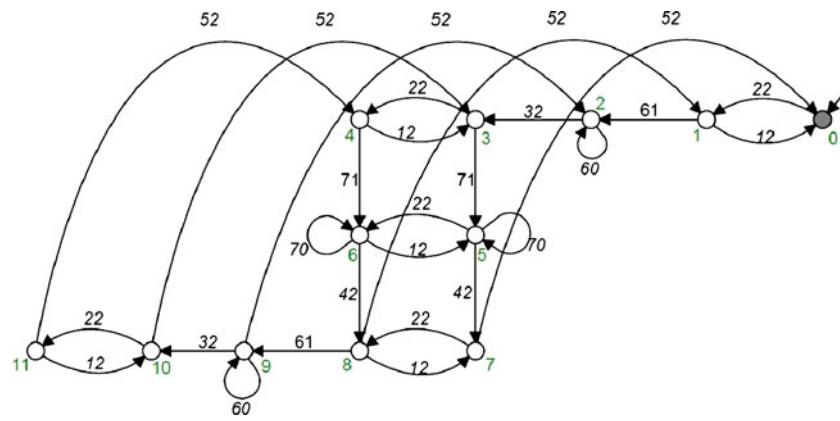
4.2.9 Step 9

As the SUPER automaton model is now obtained it is time to replace the selfloops representing the actions with the respective actions. Since the uncontrollable event 60 represents the sort solenoid action (SSa), selfloops [2,60,2] and [9,60,9] of the SUPER automaton model, depicted in Fig. 16, are replaced with SSa as shown in Fig. 17. Similarly, the uncontrollable event 70 represents the assembly solenoid action (ASa). Therefore, selfloops [5,70,5] and [6,70,6] of the SUPER automaton model, depicted in Fig. 16, are replaced with ASa as shown in Fig. 17.

4.2.10 Step 10

Timed events are now reconsidered and declared as postponed events. All events are provided in Table 4. Events 32, 42 and 52 are timed events with deterministic time delay values 1, 0.7, and 1.5 s, respectively. The postponed event pe32 (resp. pe42, pe52) is related to the event e32 (resp. e42, e52). The event e32 is defined by the following signals: $[Q0.0 \wedge \overline{I0.1} \wedge I0.0 \downarrow, 1s]$ This time delay provides enough time for a plastic ring to move from the sort area into the assembly chute. The event e42 is defined by the following signal: $[I0.2 \uparrow, 0.7 s.]$ This time delay provides enough time for a plastic ring to move from the assembly chute into the assembly area. The event e52 is defined by the following

Fig. 16 SUPER automaton model



signal: [0.2], 1.5 s.] This time delay provides enough time for a plastic ring to clear the assembly area.

4.2.11 Step 11

Finally, the implementation of the SUPER automaton model shown in Fig. 17 is considered. In order to represent states of SUPER, memory bits q_1, q_2, \dots, q_{11} are assigned to the states 1, 2, ..., 11, respectively, of the SUPER automaton model, as shown in Fig. 17. Off-delay timers T_0 with 1-s time delay, T_1 with 0.7-s time delay, and T_2 with 1.5-s time delay are used to implement postponed events pe32, pe42 and pe52 of the SUPER automaton model, respectively. After doing these assignments by using a direct mapping, the LLD code shown in Fig. 17, is obtained. This LLD code is written for a Siemens S7- 300 (CPU 319) PLC. The LLD symbols (as used in this paper) of a Siemens S7-300 PLC are defined in Table 5.

The LLD code is structured in such a way that the network 1 (NW1) initializes the system by means of the initialization memory bit M0.0. In NW1, the initial state q_0 is set and the rest of the states, namely q_1, q_2, \dots, q_{11} are

reset. In addition, PLC outputs Q0.3 and Q0.4 are also set in NW1 and they always stay set as long as the PLC is running, because these outputs are used to activate both upper and lower conveyor motors, respectively. M0.0 is set in the first PLC scan time and this ensures that the above mentioned initialization procedures will be done only once. Networks NW2, NW3, ..., NW11 implement the events $e_{12}, e_{22}, pe_{32}, pe_{42}, pe_{52}, e_{61},$ and e_{71} respectively. Networks NW12, NW13, ..., NW34 implement 23 transitions of the SUPER as shown in Fig. 18. Note that SUPER is computed as having 12 states and 27 transitions but 4 of the transitions of the SUPER are removed because they are used to represent assigned actions on the respected states. Therefore, 23 transitions are implemented. The implementation of the transitions is straight forward. For example NW12 implements the transition [0, 22 ,1] this means that if the state 0 is set and the event 22 occurs then the SUPER will move from state 0 to state 1. The implementation of transition [0,22,1] mimics this behavior as follows: If q_0 is set and at the same time e_{22} occurs then set q_1 and reset q_0 . Other transitions are implemented in the same manner. However there is one important point to be explained in terms of

Fig. 17 SUPER automaton model with assigned actions SSa and ASa

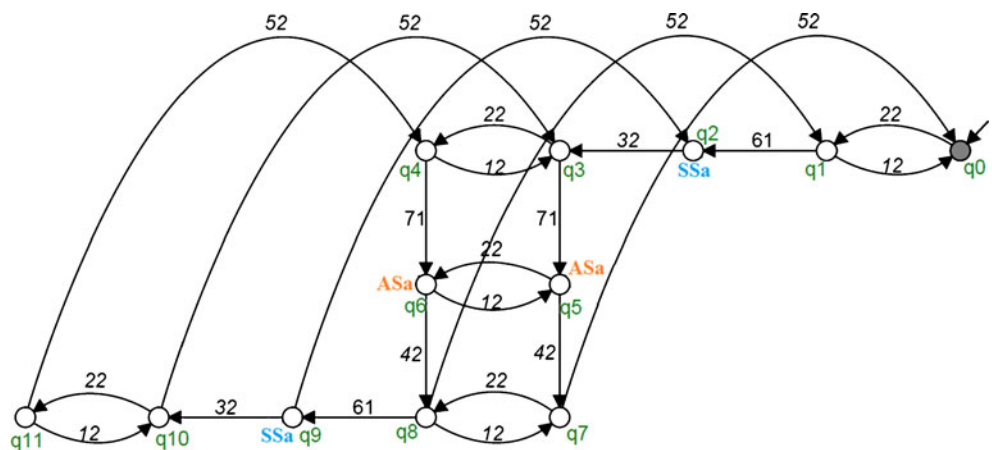


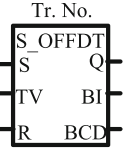
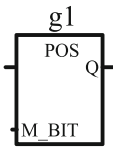
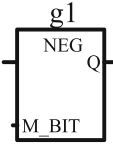


Table 5 Some symbols of Siemens S7-300 PLC

LLD Symbol	Definition
--(S)	Set
--(R)	Reset
--()	Output coil
T	Timer
I	Input
Q	Output
M	Memory bit
	Normally open contact
	Normally closed contact
	Off Delay S5 Timer S: Start input Q: Status of the timer TV: Preset Time Value BI: Remaining Time Value BCD: Remaining Time Value in BCD format R: Reset input Tr. No: Number of Timer
	The rising edge of the input signal g1 is transferred to output Q. Q: Output M_BIT: Memory bit
	The falling edge of the input signal g1 is transferred to output Q. Q: Output M_BIT: Memory bit

the implementation order of some transitions. These transitions are the ones in which events e_{61} and e_{71} are present. It can be seen that these events are always available. Therefore, in the implementation of the following set of transitions, the lowest priority must be given to the transitions including events e_{61} and e_{71} . There are four set of such transitions: $E1 = \{[1,12,0],[1,61,2]\}$, $E2 = \{[7,8,12],[8,52,1],[8,61,9]\}$, $E3 = \{[3,4,22],[3,71,5]\}$, $E4 = \{[3,4,12],[4,71,6]\}$. It can be seen that each set of transitions has a common source state, which means that transitions of each set originate from the same state. In the first set $E1$, the transition $[1,12,0]$ is implemented before the transition $[1,61,2]$. In the second set $E2$, the transitions $[7,8,12]$, and $[8,52,1]$ are implemented before the transition $[8,61,9]$. In the third set $E3$, the transition $[3,4,22]$ is implemented before the transition $[3,71,5]$. Finally, in the set $E4$, the transition $[3,4,12]$ is implemented before the transition $[4,71,6]$. This order of LLD rungs ensures the proper operation of the PLC in controlling the system. Finally, actions SSa

(sort solenoid action) and ASa (assembly solenoid action) are implemented in networks $NW35$ and $NW36$ respectively. By using a PC software called “Simatic Manager”, this LLD code was programmed on a Siemens S7-300 (CPU 319) PLC in the experimental set-up shown in Fig. 7. The LLD code implemented the control specifications as required and it did not unnecessarily constrain the behavior of the experimental manufacturing system.

The LLD code shown in Fig. 18 can be validated by using timing diagrams of the inputs, outputs, and the variables used within the LLD code. In order to accomplish such a task, a scenario can be characterized to inspect PLC inputs, the variables used and output signals as carried out in [31]. However, as the experimental manufacturing system worked very well as specified by the specifications the validation is not considered. For verification purposes, it may well be the case to use a PLC simulation software as in [13].

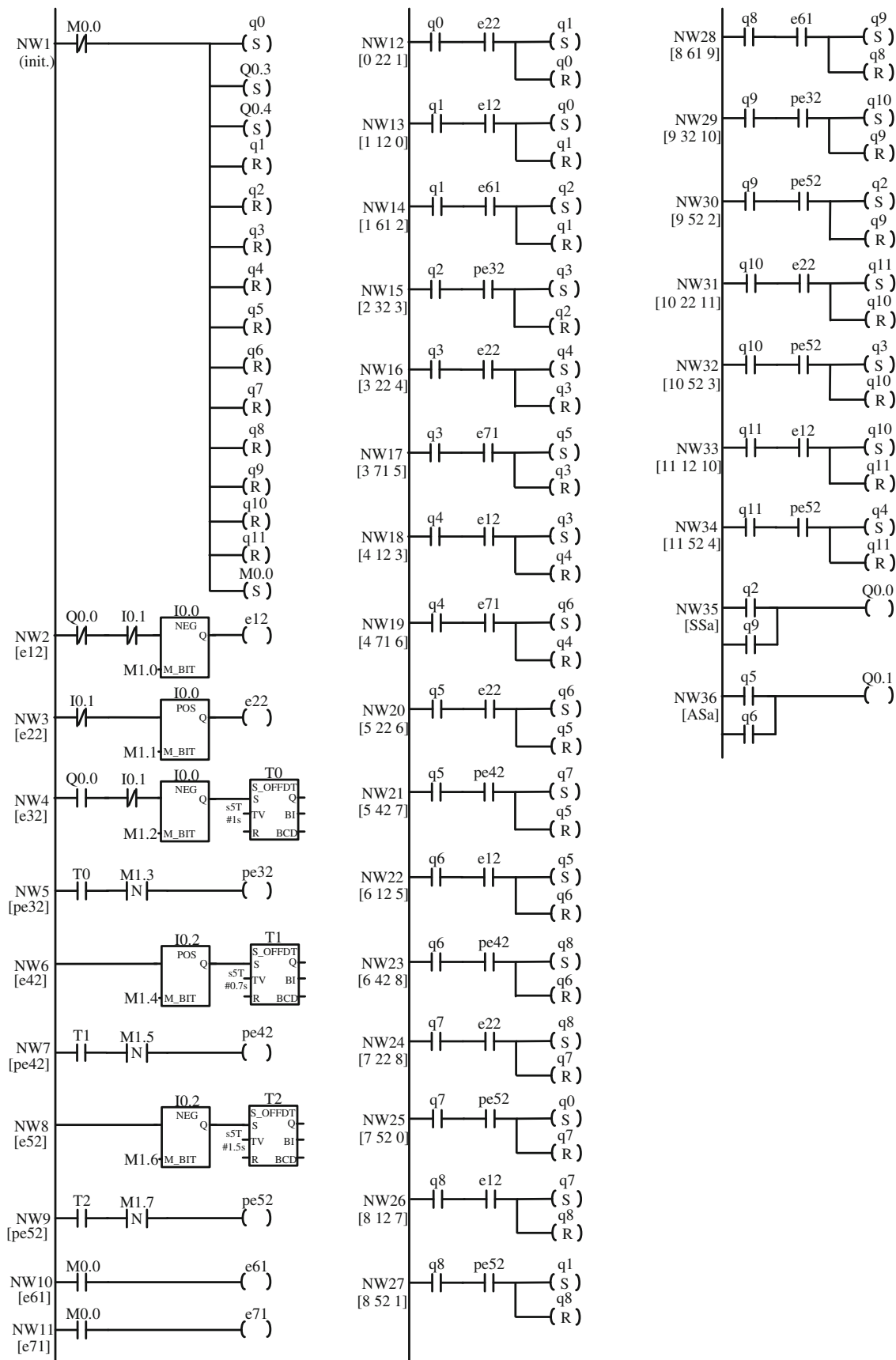


Fig. 18 The LLD code implementation of the SUPER automaton model shown in Fig. 17

5 Conclusions

In this paper, an easy to use, general and practical technique is proposed for the PLC-based implementation of RW supervisors with time delay functions. This paper has particularly shown the applicability of the proposed approach to low-level real-time control where the role of the supervisor is to arrange low-level interaction between the control devices, such as motors, actuators, etc. The RW supervisor obtained in this study is of a monolithic type. This means that when this method is used for complex systems, the state space and therefore the number of the states within the RW supervisor would get very large. This is a problem when implementing each state by means of a memory element in a commercial PLC, because there are a limited number of such elements. To solve this problem, further studies will be carried out to obtain reduced RW supervisors and modular supervisors as opposed to monolithic supervisors.

Acknowledgments This work was supported by the research grant of The Scientific and Technological Research Council of Turkey (Türkiye Bilimsel ve Teknolojik Araştırma Kurumu—TÜBİTAK) under the project number TÜBİTAK-107E125.

References

- Ramadge PJ, Wonham WM (1989) The control of discrete event systems. *Proc IEEE* 77(1):81–98
- Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. *SIAM J Contr Optim* 25(1):206–230
- Fabian M, Hellegren A (1998) PLC-based implementation of supervisory control for discrete event systems. in *Proc. 37th IEEE Conf. Dec. Contr.*, Tampa, Florida, USA, pp. 3305–3310
- Brandin BA (1996) The real-time supervisory control of an experimental manufacturing cell. *IEEE Trans Robot Autom* 12(1):1–14
- Lauzon SC, Ma AKL, Mills JK, Benhabib B (1996) Application of discrete-event-system theory to flexible manufacturing. *IEEE Cont. Systems Magazine*, pp. 41–48
- Lauzon SC, Mills JK, Benhabib B (1997) An implementation methodology for the supervisory control of flexible manufacturing workcells. *J Manuf Syst* 16(2):91–101
- Ramirez-Serrano A, Zhu SC, Benhabib B (2000) Moore automata for the supervisory control of robotic manufacturing workcells. *Auton Robot* 9:59–69
- Ramirez-Serrano A, Zhu SC, Chan SKH, Chan SSW, Ficocelli M, Benhabib B (2002) A hybrid PC/PLC architecture for manufacturing-system control-theory and implementation. *J Intell Manuf* 13:261–281
- De Queiroz MH, Cury JER (2002) Synthesis and implementation of local modular supervisory control for a manufacturing cell. in *Proc. 6th Int. Workshop on Discrete Event Syst.*, Zaragoza, Spain, pp. 377–382
- Vieira AD, Cury JER, De Queiroz MH (2006) A model for implementation of supervisory control of a discrete event systems. in *Proc. IEEE Conf. on Emerging Tech. and Factory Automation*, ETFA'2006, pp. 225–232
- Gouyon D, Petin JF, Gouin A (2004) Pragmatic approach for modular synthesis and implementation. *Int J Prod Res* 42 (14):2839–2858
- Liu J, Darabi H (2002) Ladder logic implementation of Ramadge-Wonham supervisory controller. in *Proc. 6th Int. Workshop on Discrete Event Syst.*, Zaragoza, Spain, pp. 383–389
- Moniruzzaman M, Gohari P (2007) Implementing supervisory control maps with PLC. in *Proc. 2007 American Contr. Conf.*, New York City, USA, pp. 3594–3599
- Manesis S, Akantziotis K (2005) Automated synthesis of ladder automation circuits based on state-diagrams. *Adv Eng Softw* 36:225–233
- Hasdemir İT, Kurtulan S, Gören L (2008) An implementation methodology for supervisory control theory. *Int J Adv Manuf Tech* 36:373–385
- David R, Alla H (2005) *Discrete, continuous, and hybrid Petri nets*. Springer
- Alur R, Dill D (1994) A theory of timed automata. *Theor Comput Sci B* 126:183–235
- Fix L, Alur R, Henzinger TA (1994) A determinizable class of timed automata. In: David L (ed) Dill, editor, sixth International Conference on Computer-Aided Verification CAV, volume 818 of Lecture Notes in Computer Science. Springer, California, pp 1–13
- Larsen KG, Pettersson P, Yi W (1995) Model-checking for real-time systems. in *Proc. of Fundamentals of Computation Theory*, Lecture Notes in Computer Science, vol. 965, pp. 62–88
- Bouyer P, Dufourd C, Fleury E, Petit A (2000) Are timed automata updatable. in *Proc. 12th Int. Conf. Computer Aided Verification (CAV'2000)*, Chicago, IL, USA. Lecture Notes in Computer Science, vol. 1855, pp. 464–479. Springer
- Alur R, Fix L, Henzinger TA (1999) Event-clock automata: a determinizable class of timed automata. *Theor Comput Sci* 211:253–273
- Barandin A, Wonham WM (1994) Supervisory control of timed discrete event systems. *IEEE Trans. on Autom. Control*, vol.39, no.2
- Takai S, Ushio T (2006) A new class of supervisors for timed discrete event systems under partial observation. *Discrete Event Dyn Syst Theor Appl* 16:257–278
- Park S-J, Cho K-H, Lim J-T (2004) Supervisory control of real time discrete event system under bounded time constraints. *IEEE Proc Control Theory Appl* 151:347–352
- Uzam M, Gelen G, Dalcı R (2009) A new approach for the ladder logic implementation of Ramadge–Wonham supervisors. in *Proc. 22nd Int. Symp. on Information, Communication and Automation Technologies*, Sarajevo, Bosnia and Herzegovina, October 29–31, pp. 113–119
- Uzam M, Gelen G, Dalcı R (2009) Timed transition automata and their ladder logic implementation. in *Proc. 22nd International Symposium on Information, Communication and Automation Technologies*, Sarajevo, Bosnia and Herzegovina, October 29–31, pp. 120–127
- Gelen G, Uzam M, Dalcı R (2010) The concept of postponed event in timed discrete event systems and its PLC implementation. in *Proc. 2010 IEEE Int. Conf. on Syst. Man, and Cybern. (SMC 2010)*, İstanbul, Turkey, October 10–13, pp. 2753–2759
- TCT (2011). A software tool supporting supervisory control theory. Systems Control Group, ECE Dept., University of Toronto, Posted at URL: www.control.utoronto.ca/DES
- Wonham WM. Supervisory control of discrete event systems. ECE dept, University of Toronto, 1997–2011, <http://www.control.utoronto.ca/DES/>, Updated 2011.07.01
- IEC 61131-3 (2003) Programmable controllers—part 3: programming languages
- Uzam M, Gelen G (2009) The real-time supervisory control of an experimental manufacturing system based on a hybrid method. *Control Eng Pract* 17(10):1174–1189