Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2013) 21: 394 – 419
© TÜBİTAK
doi:10.3906/elk-1110-29

An improved hybrid approach for the PLC-based implementation of reduced RW supervisors

Murat UZAM[1,*], **Gökhan GELEN**[2]
[1]Department of Electrical and Electronics Engineering, Faculty of Engineering and Architecture, Melikşah University, 38280 Talas, Kayseri, Turkey
[2]Department of Mechatronics Engineering, Faculty of Engineering and Natural Science, Gaziosmanpaşa University, Taşlıçiftlik Campus, 60150, Tokat, Turkey

Received: 14.10.2011 • **Accepted:** 16.01.2012 • **Published Online:** 22.03.2013 • **Printed:** 22.04.2013

Abstract: A successful application of a hybrid (mixed Petri net/automaton) approach for the real-time supervisory control of an experimental manufacturing system was reported recently. The hybrid approach includes a Ramadge–Wonham (RW) supervisor in the form of an automaton. The reduced RW supervisor offers fewer states for the programmable logic controller (PLC) implementation of the hybrid controller with less memory requirements, but due to a problem called the avalanche effect, the PLC implementation of the hybrid controller with the reduced RW supervisor was not possible. This paper proposes a method to both detect and eliminate the avalanche effect problem for a RW supervisor in the form of an automaton, which enables the PLC implementation of the hybrid controller with the reduced RW supervisor. In addition, this paper improves the recently proposed hybrid approach by including the reduced RW supervisor and the avalanche effect detection and elimination method. The applicability of the improved hybrid approach is demonstrated by the PLC-based real-time control of an experimental manufacturing system.

Key words: Automata, Petri nets, supervisory control theory, hybrid supervisor, PLC, real-time implementation, ladder logic diagram

1. Introduction

The control theory concepts for continuous systems were extended to the discrete-event environment with the introduction of the supervisory control theory (SCT) [1–3]. In the SCT, conditions to be avoided are specified as a forbidden state problem, an example of which is the simultaneous utilization of some resource by 2 or more users. In addition, in the SCT, generally the controlled system is required to be nonblocking (namely that the specified target states, often just the initial state, be maintained as reachable) and to be maximally permissive, i.e. to permit the occurrence of all events not leading to a violation of the foregoing requirements. Alternatively, Petri net (PN)-based approaches to supervisory control design have also been considered. This is because the state-space representation of PNs as a vector addition system can result in a compact system description. Therefore, the net structure is kept small even though the number of possible markings may become large. PN models can easily be used for the systematic construction of supervisory controllers and also for the analysis of various qualitative properties and quantitative performance evaluations [4]. However, in general, optimal supervisors need not exist within the class of PNs [5] and nonblockingness (e.g., reversibility) is difficult to achieve by standard PN methods.

[*]Correspondence: murat_uzam@meliksah.edu.tr

There are mainly 2 types of PN supervisors proposed in the literature: the compiled supervisor and the mapping supervisor. In the former, the control policy is represented as a net structure, while in the latter, the control actions (event enabling/disabling) are computed by an on-line controller as a feedback function of the marking of the system. When fully compiling the supervisor action into a net structure, the following advantages are obtained [4]: the computation of the control action is faster, and the same PN execution algorithms may be used for both the original system and the supervisor. In addition, a closed-loop model of the system under control can be built with standard net composition constructions. It is evident that compiled supervisors are preferable to mapping supervisors. However, optimal PN-based controllers need not always exist, even though the counterpart SCT problem may be solvable in the framework of automata [4]. In [6], a set of heuristic methods was proposed for the PN-based supervisory control of discrete event systems (DESs) together with the ladder logic diagram (LLD)-based PLC implementations of these proposed methods. The results obtained in [6] were reported in following works [7–10]. The work reported in [7] showed how to convert PNs into LLD code and an example system was considered to show the applicability of the proposed method for the control of industrial systems. In [8], a more systematic method was reported compared to that in [7] in the design on PN-based control systems. This improvement included the introduction of automation PNs (APNs), a class of interpreted PNs, which extends the ordinary PNs by assigning sensory information to transitions and by assigning a control action at some places in the PN. These extensions enable the control engineers to design PN-based controllers, but the proposed method was still heuristic rather than formal. In [10], an alternative PN-based method was proposed to address the problem of obtaining a systematic PN controller. In the computation of the closed-loop control system, an uncontrolled PN model of the system was controlled by means of enabling arcs. The computation of these enabling arcs includes the reachability graph of the uncontrolled PN model and the control specifications. However, the problem with the method proposed in [10] is that it is necessary to duplicate the controllable transitions of the original PN model and it suffers from the state explosion problem. Optimal PN-based controllers need not always exist, even though the counterpart SCT problem may be solvable in the framework of automata. In [11], a hybrid approach to the supervisory control of DESs for forbidden state problems was proposed, which couples SCT (automaton) supervisors to uncontrolled PN models through inhibitor arcs. This new type of compiled supervisor is effective even for problems whose optimal PN-based controllers do not exist. The method improves on previous work reported in [9]; in particular: 1) PN reduction techniques are used to initially reduce the computational size; 2) the SCT is used to compute the supervisor (as an automaton, which will then always exist provided the original problem is indeed solvable; and 3) the state size of the computed supervisor is reduced (often significantly) by control congruence.

Assuming that an uncontrolled bounded PN model of a (plant) DES and a set of forbidden state specifications are given, the approach proposed in [11] computes a maximally permissive and nonblocking closed-loop hybrid model. This method is entirely straightforward logically, graphically, and technologically, and it becomes a useful alternative to mapping supervisors, especially when an optimal PN-type controller for the given problem does not exist. The method proposed in [11] is general in the sense that it is not limited to any special subclass of bounded PNs, such as marked graphs or state machines. Therefore, the method should be widely applicable and of practical interest. A successful application of a hybrid (mixed PN/automaton) approach for the real-time supervisory control of an experimental manufacturing system was reported in [12]. The hybrid approach includes a Ramadge–Wonham (RW) supervisor in the form of an automaton. The reduced RW supervisor offers fewer states for the programmable logic controller (PLC) implementation of the hybrid controller, with less memory requirements. However, due to a problem called the avalanche effect, the PLC implementation of the hybrid controller with the reduced RW supervisor was not possible, as explained in [12].

The PLC-based implementation of supervisors (in automaton form) has been studied by many researchers [13–23]. In [13–16], the application of the SCT to flexible manufacturing cells and PLC-based implementations was studied. In [17,21], local modular supervisors and their PLC-based implementations were considered. The implementation of the supervisors was proposed in LLD code [17] and in sequential function charts [21]. The main feature of these works is that the control system was used as an interface between the real input–output signals and the theoretical supervisors [17,19,21]. In [18,22], the conversion of finite state machines (automaton) to LLD was proposed. The LLD implementation of finite state machines was studied in [20]. In a recent work [23], a new approach was proposed for the LLD implementation of supervisors. The method of [23] deals mainly with the assignment of actions (output signals) to the related states of supervisors.

The problems and possible solutions for the PLC implementation of supervisors in automaton form were discussed in [24] The avalanche effect problem makes the program skip over an arbitrary number of states during the same PLC scan cycle [4]. Arranging PLC code rungs in the reverse order was proposed as a simple solution for the avalanche effect problem in [24]. However, this not a general solution and may fail in some cases [25]. In order to solve the avalanche effect problem, a general method based on expressing automata in the logical domain was proposed in [25]. The disadvantage of the method proposed in [25] is that it requires 2 memory elements, i.e. Boolean variables, for implementing each state of a supervisor. This is a big problem especially when implementing a supervisor with a huge number of states in a PLC with very limited memory resources. In this paper, a new method is proposed for the detection and elimination of the avalanche effect problem. There are 2 contributions of this paper. In the first, an algorithm is proposed for the detection of the events that cause the avalanche effect in a given supervisor (automaton). In the second, a very simple and effective avalanche effect elimination method (AEEM) is proposed.

The purpose of this paper is to improve the hybrid approach of [12] using the reduced RW supervisor and the avalanche effect detection and elimination method. The applicability of the improved hybrid approach is demonstrated by the 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 the following: the SCT; APNs (a class of interpreted PNs); token passing logic (TPL) methodology, a technique to convert interpreted PNs into LLDs for implementation on PLCs; the avalanche effect problem; and the detection and elimination of the avalanche effect problem The improved version of the hybrid methodology of [12] is proposed in Section 3. The applicability of the improved hybrid approach is demonstrated by the PLC-based real-time control of an experimental manufacturing system in Section 4. The hybrid controller proposed in [12] and the reduced hybrid controller proposed in this paper are compared in Section 5. Finally, conclusions are given and some future research directions are provided in Section 6.

2. Preliminaries

2.1. Supervisory control theory

The SCT was introduced to extend control theory concepts for continuous systems to the discrete-event environment [1–3]. In the SCT, a forbidden state problem [3] specifies conditions that must be avoided, typically the simultaneous utilization of some resource by 2 or more users. In addition, the SCT generally requires the controlled system to be nonblocking and to be maximally permissive, i.e. to permit the occurrence of all events not leading to a violation of the foregoing requirements. DESs evolve on spontaneously occurring events. Let $\sum$ be a finite set of events. The set of all of the finite concatenations of events in $\sum$ is denoted by $\sum^*$. 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 an empty string. A subset L $\subseteq \sum^*$ is called a language over $\sum$. For a string s $\in \sum^*$, $\bar{s}$ denotes the prefixes of s and is defined as $\bar{s} = \{s_p \in \sum^* \mid \exists t \in \sum^* \ (s_p t = s)\}$. The extension of this definition to the language prefix closure of a language L is denoted by $\overline{L}$. A language L satisfying the condition L $= \overline{L}$ is said to be prefix-closed [26,27].

An automaton, denoted by G, is a 6-tuple G $=$ (Q, $\sum$, f, Γ, q_0, Q_m), where Q is the set of states, $\sum$ is the finite event set, and f: Q $\times \sum \rightarrow$ Q is the partial transition function. Γ: q $\rightarrow 2^{\Sigma}$ is the active event function. Γ(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 2 states is shown in Figure 1. This automaton has 2 states labeled with q0 and q1. q0 with a double arrow is the initial (and marked) state, while q1 with an exiting arrow is the marked state of the automaton. A directed arrow represents the transition functions of the automaton. The labels of the transitions (e1, e2) correspond to events.

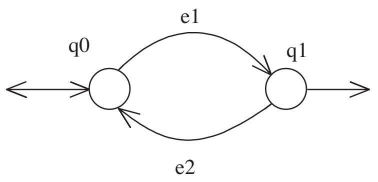


Figure 1. A simple automaton model.

The language generated by G is denoted by L(G) and is defined as L(G) $= \{s \in \sum^*: f(q_0,s) \text{ is defined}\}$. The language marked by G is denoted by L_m(G) and is defined as L_m(G) $= \{s \in \sum^*: f(q_0,s) \in Q_m\}$. The DES modeled as automaton G is said to be nonblocking if L(G) $= \overline{L_m(G)}$. DESs named as A and B can be composed with synchronous product (parallel composition) operation. The synchronous product of 2 automata is denoted by A||B and it represents the synchronous behavior of 2 automata. In the resulting automaton, common events occur synchronously, while the other events occur asynchronously [26,27].

The SCT makes use of formal languages to model the uncontrolled behavior of DESs (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 the supervisor to generate new events. In the SCT, events are divided into 2 disjoint sets, controllable events and uncontrollable events. These sets are denoted by $\sum_c$ and $\sum_{uc}$, 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 the controllability conditions. This condition is defined as $\overline{K}\sum_{uc} \cap$ M $\subseteq \overline{K}$, where $\overline{K}$ is the language that will be generated under the control of the supervisor and M is the language generated by the uncontrolled system.

2.2. Automation PNs

PNs are widely used as a formal method for design, analysis, and control of DESs. They were named after Carl A. Petri, a contemporary German mathematician and computer scientist who introduced a net-like mathematical tool for the study of communication with automata [28]. Ordinary PNs do not deal with actuators or sensors. Because of this, it is necessary to define a PN-based controller (APN) that can embrace both actuators and sensors within an extended PN framework [8]. An APN model is shown in Figure 2. In the APN, sensor readings can be used as firing conditions at transitions. The presence or absence of sensor readings can be used in conjunction with the extended PN preconditions to fire transitions. In the APN, 2 types of actuation can be considered, namely impulse actions and level actions. Actions are associated with places. Formally, an APN can be defined as follows:

$$APN = (P, T, Pre, Post, In, En, X, Q, M_0), \tag{1}$$

where $P = \{p_1,\ p_2,\ ...,\ p_n\}$ is a finite, nonempty set of places; $T = \{t_l,\ t_2,\ ...,\ t_m\}$ is a finite, nonempty set of transitions; and $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$, Pre: $(\mathrm{P} \times \mathrm{T}) \rightarrow \mathrm{N}$ is an input function that defines directed ordinary arcs from places to transitions, where N is set of nonnegative integers.

Post: $(\mathrm{T} \times \mathrm{P}) \rightarrow \mathrm{N}$ is an output function that defines directed ordinary arcs from transitions to places.

In: $(\mathrm{P} \times \mathrm{T}) \rightarrow \mathrm{N}$ is an inhibitor input function that defines inhibitor arcs from places to transitions.

En: $(\mathrm{P} \times \mathrm{T}) \rightarrow \mathrm{N}$ is an enabling input function that defines enabling arcs from places to transitions.

$\chi = \{\chi_1,\ \chi_2,\ \ldots,\ \chi_m\}$ is a finite, nonempty set of firing conditions associated with the transitions.

$\mathrm{Q} = \{\mathrm{q}_1, \mathrm{q}_2, \ldots, \mathrm{q}_n\}$ is a finite set of actions that might be assigned to the places.

$\mathrm{M}_0 = \mathrm{P} \rightarrow \mathrm{N}$ is the initial marking.

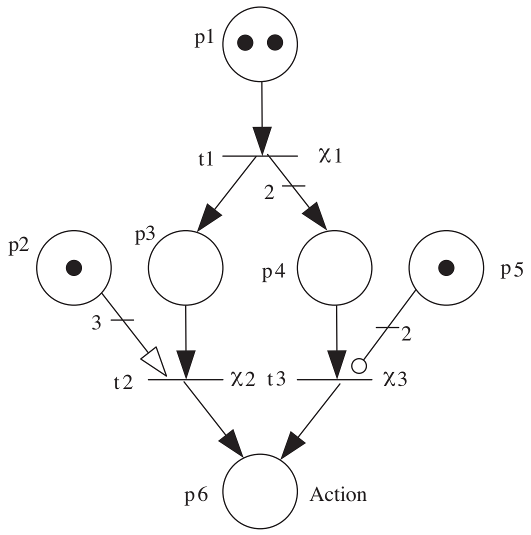


Figure 2. An APN model.

The APN consists of 2 types of nodes called places, represented by circles (◯), and transitions, represented by bars (—). There are 3 types of arcs used in the APN, namely ordinary arcs, represented by a directed arrow ($\longrightarrow$); inhibitor arcs, represented by an arrow, whose end is a circle ($\multimap$); and enabling arcs, represented by a directed arrow, whose end is empty ($\longrightarrow\triangleright$). Weighted and directed ordinary arcs connect places to transitions and vice versa, whereas weighted enabling arcs and inhibitor arcs connect only places to transitions.

The number of tokens in places represents the current state of the system and the transitions represent events. Each transition has a set of input and output places, which represent the precondition and postcondition of the transition. The actions (Q) assigned to the places can be either impulse actions or level actions. Impulse actions are enabled at the instant when a token is deposited into the place, and level actions are enabled when there is a token(s) at the place. More than one action may be assigned to a place. Firing conditions in the APN are recognized as external events, such as sensor readings. A firing condition, χ, associated with a transition, t, is a Boolean variable that can be '0', in which case the related transition t is not allowed to fire, or it can be '1', in which case the related transition t is allowed to fire if it is enabled. The marking of the APN is represented by the number of tokens in each place. Tokens are represented by black dots ($\bullet$). The movement of tokens between places describes the evolution of the APN and is accomplished by the firing of the enabled transitions. The following rules are used to govern the flow of tokens.

Enabling rules: In the APN, there are 3 rules that define whether a transition is enabled to fire. These preconditions must be satisfied for a transition to fire.

1. If an input place p of a transition t is connected to t with a weighted ordinary arc Pre(p,t), then t is said to be enabled when p contains at least the number of tokens equal to the weight of the directed ordinary arc, i.e. $M(p) \geq Pre(p,t)$.
2. If an input place p of a transition t is connected to t with a weighted enabling arc En(p,t), then t is said to be enabled when p contains at least the number of tokens equal to the weight of the enabling arc, i.e. $M(p) \geq En(p,t)$.
3. If an input place p of a transition t is connected to t with a weighted inhibitor arc In(p,t), then t is said to be enabled when p contains less tokens than the weight of the inhibitor arc, i.e. $M(p) < In(p,t)$.

Firing rules: In the APN, an enabled transition t can or cannot fire depending on the external firing condition χ of t. These firing conditions can be leading edge ($\uparrow$), falling edge ($\downarrow$), positive level, or zero level of a sensor reading. Broadly speaking, a firing condition χ may include more than one sensor reading with "AND", "OR", and "NOT" logical operators. When dealing with more than one sensor reading as firing conditions, the logical operators of the firing conditions must be taken into account accordingly. In the special case where $\chi =$ 1, transition t is always allowed to fire when it is enabled. When an enabled transition t fires, it removes from each input place p the number of tokens equal to the weight of the directed ordinary arc connecting p to t. It deposits, at the same time, in each output place p the number of tokens equal to the weight of the directed arc connecting t to p. It should be noted that the firing of an enabled transition t does not change the marking of the input places, which are connected to the transition t only by enabling or inhibitor arcs.

It is also possible to consider timed APNs, as in normal PNs. Ordinary PNs do not include a concept of time. With this class of nets, it is possible only to describe the logical structure of the modeled system, but not its time evolution. Due to the need for the temporal requirements of DESs, the concept of time has been introduced into PNs in a variety of ways. In this paper, the timed-transition PN is considered as described in [29, p. 94]. A timed-transition APN (TTAPN) is a tuple defined as follows:

$$TTAPN = (APN,\tau), \tag{2}$$

where the APN is an automation PN and τ is a function from the set of transitions to the set of positive or zero rational numbers. $\tau(t_i) = \mathrm{T}_i =$ timing associated with transition t_i. In this case, a token can have 2 states: it can be reserved for the firing of a timed-transition t_i or it can be unreserved. If a timed transition is enabled, then it is ready to fire. When the firing condition for the transition occurs and holds true for a specified amount of time (T_i), the token of input place to this transition is said to be reserved for T_i. When time T_i has elapsed, the transition is effectively fired: the reserved token is removed from the input place and an unreserved token is put into the output place(s). This is illustrated in Figure 3, where the transition t3 is a timed-transition with the time delay T3. At the beginning, there is a token in place p1, as shown in Figure 3a. When transition t2 is fired the token is removed from p1 and a token is deposited in place p2, thereby resulting in the enabling of timed-transition t3, as shown in Figure 3b. Next, the firing condition χ_3 for transition t3 may occur at any moment after this. When the firing condition χ_3 occurs and holds true for the time delay T3, the token required for this firing is reserved, as shown in Figure 3c. When time delay T3 has elapsed, the transition is effectively fired. The token reserved for firing is then removed from place $p2$ and an unreserved token is deposited in place $p3$, as shown in Figure 3d.

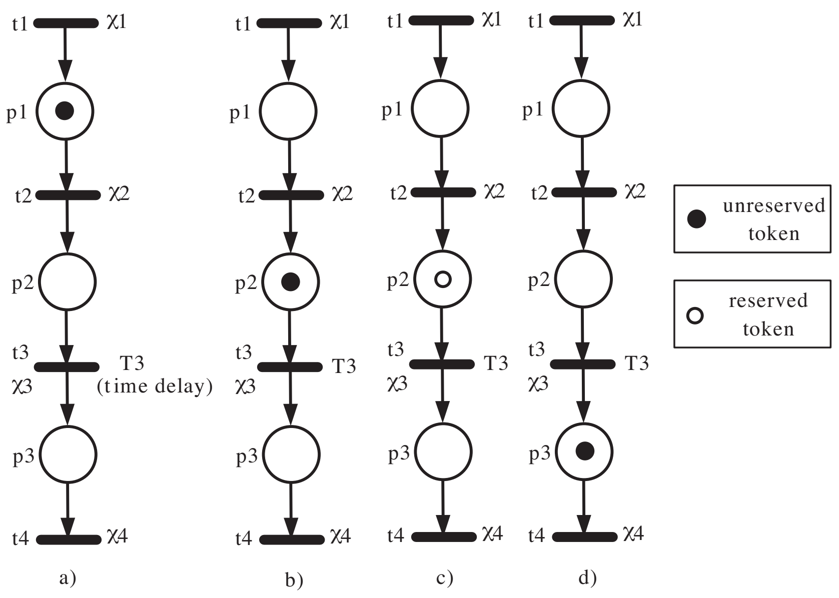


Figure 3. Timed transition in an APN.

2.3. TPL methodology

The control community all around the world has been very active for the past 2 decades producing PN-based discrete event control design techniques, especially for manufacturing systems. Generally, PLCs are the preferred implementation tools for such controllers in today's modern factories. In addition, LLDs are the most common and preferred programming languages used in today's PLCs. Therefore, it was a necessity to be able to convert the PN-based discrete event controllers into LLDs. To address this need, the TPL concept was introduced [6–9,30]. For a detailed survey about LLD- and PN-based discrete event control design methods, the reader is referred to [31]. It can be clearly seen from [31] that the TPL concept is very well established and very well received by the control community for converting PNs into LLDs. The prime feature of the TPL concept is that it facilitates the direct conversion of PN-based logic controllers into LLDs. This is achieved by adopting the PN concept of tokens as the main mechanism for controlling the flow of the control logic. Hence, each place within the PN corresponds to a counter (or a flag) within the LLD program. Each action at a PN place corresponds to an action within the LLD program, and each transition within the PN involving a movement of tokens corresponds to a simulated movement of tokens between LLD rungs. This simulated movement of tokens is achieved by deploying separate counters at each place within the LLD program and incrementing and decrementing these counters to simulate token flow. Thus, each place within the PN has an associated counter (or a flag), and the current count value of the counter represents the number of tokens that would be at the corresponding place within the PN. Furthermore, in consonance with the PN approach, if the count value of the counter associated with the place is nonzero, then any actions associated with that place are activated. Finally, to complete the PN synergy, if the counter associated with a place is nonzero and a PN-like transition associated with that place becomes active, then the counter at the place is decremented by 1 and the subsequent place linked by the output transition is incremented by 1. Moreover, the PLC on delay timers can be readily used to implement timed-transition PNs. In essence, the PN places are represented by separate counters. The count value of the counter represents the PN tokens. The counting down and counting up of the counters simulates the flow of the PN tokens. For safe (1-bounded) PN places, flags (memory bits) can be deployed, in which case the setting and resetting of the related flags will be used to simulate the flow of the tokens. In theory, the methodology can cope with any number of tokens and provide a visual description of the LLD program, which has all of the advantages of a full PN analysis. Furthermore, colored PNs and hierarchical PNs can also be converted into LLDs using this technique, simply by adding more counters (or flags) to each place. For a detailed treatment of the TPL concept, the reader is referred to [6].

2.4. Solving the avalanche effect problem

2.4.1. Avalanche effect problem

In the PLC implementation of a supervisor (automaton), states are represented by Boolean variables and events can be defined as the rising or falling edges of the signals. For such a definition of events, –|P |– (positive transition sensing contact) and –|N |– (negative transition sensing contact) can be used, respectively. Some symbols of the IEC 61131-3 LD (ladder diagram) PLC programming language [32] used in this paper are depicted in Table 1. The conversion of a supervisor into LLD code for PLC implementation is a straightforward process. From the literature, one can find some well-established methods. The following briefly explains the method utilized in this paper [23]: a Boolean variable (a memory bit) is assigned to each state of the supervisor. Initially, the Boolean variable representing the active (initial) state is SET and all of the other Boolean variables representing the rest of the RW supervisor states are RESET. Next, each transition between the states of the supervisor is implemented as a separate PLC ladder rung using the SET and RESET commands. When associating the events with the rising and falling edges of signals, care must be taken not to skip over an arbitrary number of states during the same scan cycle. This is called the avalanche effect and is a consequence of the sequential evaluation of the Boolean expressions [24]. For example, consider the automaton shown in Figure 4. The automaton moves from state q0 to q1 on the occurrence of ‘a’. It is then supposed to move from q1 to q2 on the new occurrence of ‘a’. If a ‘b’ occurs before another ‘a’, then it moves from q1 to q3. Thus, this automaton represents 2 strings, ‘aa’ and ‘ab’ [24]. The LLD implementation of this automaton, shown in Figure 5a, does not mimic this behavior. Rather, the avalanche effect problem causes a direct transition from q0 to q2 on the same rising edge of signal ‘a’. One simple cure for this problem is to arrange the rungs in the reverse order [24]. In this particular example, if rungs 2 and 3 of the LLD code shown in Figure 5a are arranged in the reverse order, as shown in Figure 5b, then the problem is solved. However, this methodology is highly dependent on the problem at hand and may fail in some cases. As an example, a simple automaton is given in Figure 6 [25]. Obviously, event ‘a’ causes the avalanche effect problem: when state q0 is active, an occurrence of event ‘a’ causes a transition to q1 and then back to q0 in the same PLC scan cycle. Therefore, the above-mentioned simple method does not solve the avalanche effect problem for this automaton [25].

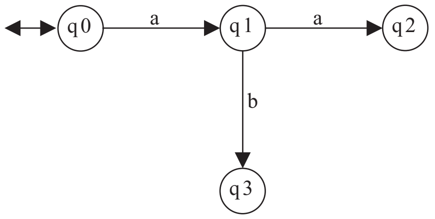


Figure 4. An example automaton represents 2 strings, ‘aa’ and ‘ab’ [25].

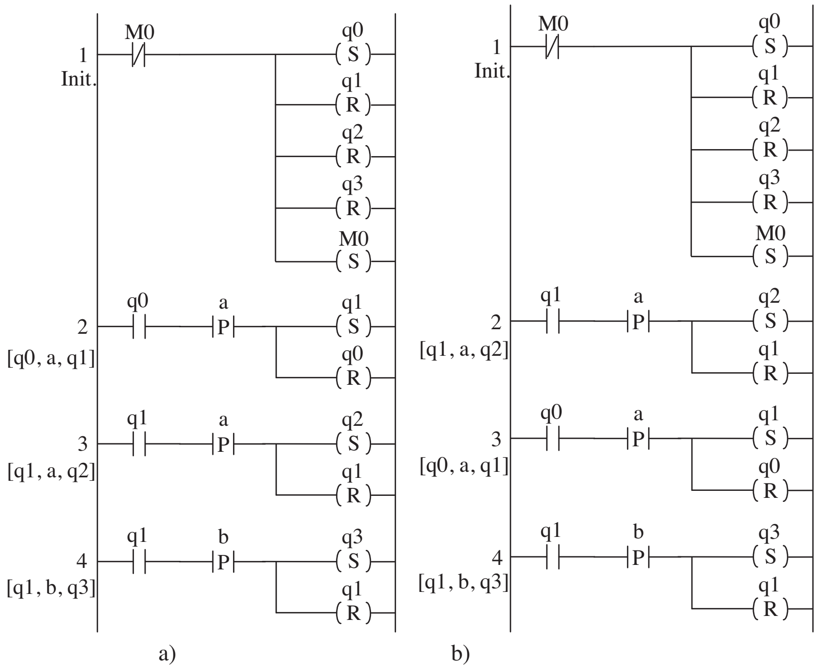


Figure 5. LLD implementations of the automaton shown in Figure 4: a) the LLD implementation represents only the occurrence of a single ‘a’; b) the LLD implementation represents the occurrence of both strings ‘aa’ and ‘ab’.

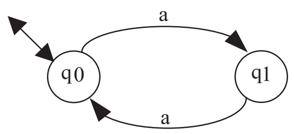


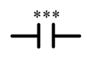
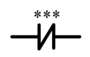
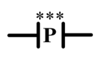
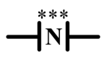
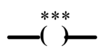
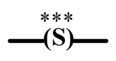
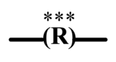
Figure 6. An example automaton with the avalanche effect problem [25].

2.4.2. Detection of avalanche effect problem in an automaton

There are a number of software tools to compute RW supervisors. In this paper, it is assumed that a RW supervisor is given in an automaton form generated using TCT software [33]. In TCT, an automaton is described in an “.ADS” file. The integer numbers are used for state labels (0, . . . , 2000000) and event labels (0, . . . , 999), where ‘0’ is used for the initial state. In TCT, the transitions of an automaton are entered as in

"*Exit_(Source)_State Transition_Label Entrance_(Target)_State*" format. For example, 2 0 1 means that there is a transition labeled with event '0' from state 2 to state 1. As a special example, 1 0 1 means that there is a self-loop labeled with event '0' on the state labeled with '1'. All of the transitions are written as a list at the end of the file. This transition list can be considered as a matrix. The number of lines of this matrix is equal to the number of all of the transitions. The row number of this matrix is equal to 3. The 1st row consists of labels of source states, the 2nd contains labels of events, and the 3rd consists of labels of target states. An example automaton is shown in Figure 7, whose TCT representation "TEST.ADS" is provided in Figure 8, based on the following event and state codings:

Table 1. Some symbols of the IEC 61131-3 LD language.

Symbol	Description
I	Input location
Q	Output location
M	Memory location
 ***	Normally open contact 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.
 ***	Normally closed contact 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.
 ***	Positive transition-sensing contact 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 will be OFF at all other times.
 ***	Negative transition-sensing contact 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 will be OFF at all other times.
 ***	Coil The state of the left link is copied to the associated Boolean variable (indicated by ***) and to the right link.
 ***	SET (latch) coil The associated Boolean variable (indicated by ***) is set to the ON state when the left link is in the ON state, and it remains set until reset by a RESET coil.
 ***	RESET (unlatch) coil The associated Boolean variable (indicated by ***) is reset to the OFF state when the left link is in the ON state, and it remains reset until set by a SET coil.

Event coding for the example automaton:

Automaton:	a	b	c	d	e	f
TCT:	1	2	3	4	5	6

State coding for the example automaton:

Automaton:	q0	q1	q2	q3	q4
TCT:	0	1	2	3	4

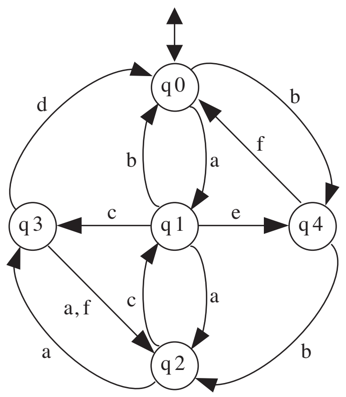
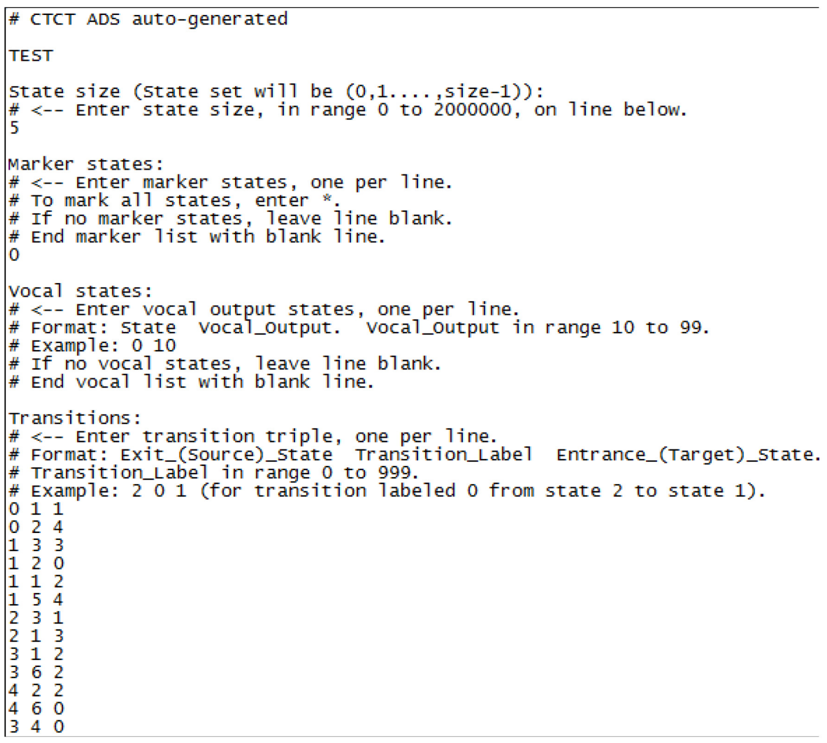


Figure 7. An example automaton with events ‘a’, ‘b’, and ‘c’ causing the avalanche effect.

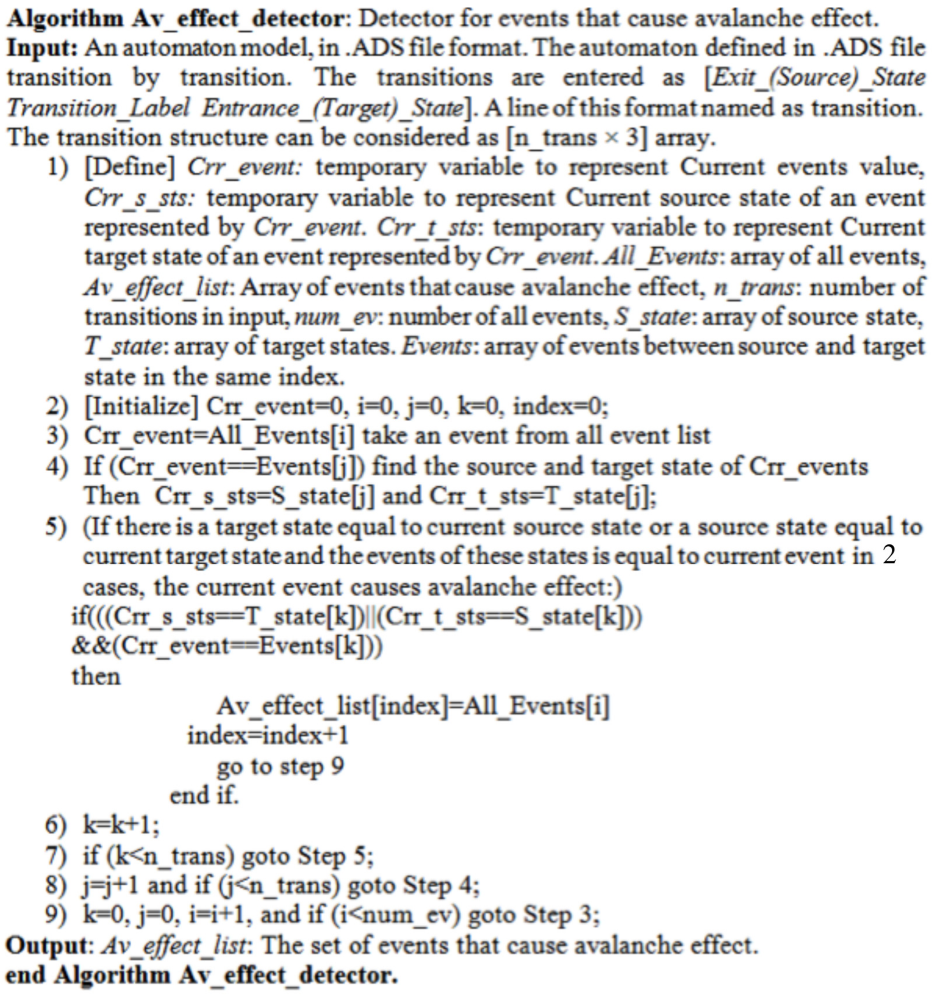


```
# CTCT ADS auto-generated

TEST

State size (State set will be (0,1....,size-1)):
# <-- Enter state size, in range 0 to 2000000, on line below.
5

Marker states:
# <-- Enter marker states, one per line.
# To mark all states, enter *.
# If no marker states, leave line blank.
# End marker list with blank line.
0

Vocal states:
# <-- Enter vocal output states, one per line.
# Format: State  Vocal_Output.  Vocal_Output in range 10 to 99.
# Example: 0 10
# If no vocal states, leave line blank.
# End vocal list with blank line.

Transitions:
# <-- Enter transition triple, one per line.
# Format: Exit_(Source)_State  Transition_Label  Entrance_(Target)_State.
# Transition_Label in range 0 to 999.
# Example: 2 0 1 (for transition labeled 0 from state 2 to state 1).
0 1 1
0 2 4
1 3 3
1 2 0
1 1 2
1 5 4
2 3 1
2 1 3
3 1 2
3 6 2
4 2 2
4 6 0
3 4 0
```

Figure 8. The format of an example .ADS file: “TEST.ADS”.

The search algorithm shown in Figure 9 is proposed in [34] to detect events that cause the avalanche effect.

The main feature of this algorithm is to find at least 2 successive transitions whose events are the same. The above algorithm was implemented in C code. Next, the C code was compiled and named as “Av_effect_detector.exe”. In the code, the .ADS file is read and the transition list is converted into a matrix. The self-looped transitions, whose source and target states are the same, are ignored because they do not cause a state change, i.e. the avalanche effect problem. If there are events causing the avalanche effect, then these events are reported on the screen, and at the same time, they are written in a log file called “logfile.txt”. When the “Av_effect_detector.exe” program was run and the above-mentioned automaton called TEST.ADS was given as the input to check whether there is an avalanche effect problem in this automaton, the screenshot of this program was provided by the “Av_effect_detector.exe”, as seen in Figure 10. This means that there is an avalanche effect problem in the automaton, shown in Figure 7, represented by TEST.ADS, and events labeled with “1, 2, and 3”, i.e. events ‘a’, ‘b’, and ‘c’, cause the avalanche effect.

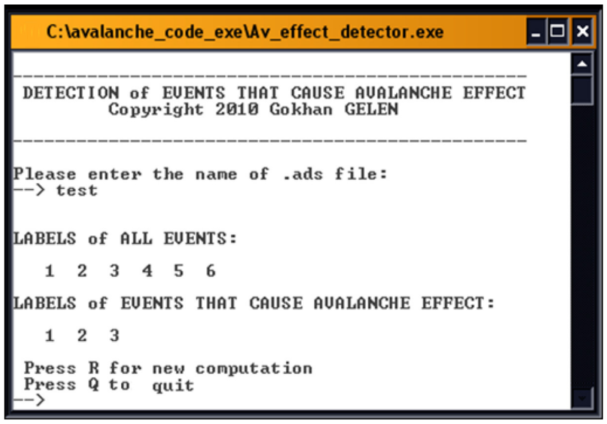


```
Algorithm Av_effect_detector: Detector for events that cause avalanche effect.
Input: An automaton model, in .ADS file format. The automaton defined in .ADS file
transition by transition. The transitions are entered as [Exit_(Source)_State
Transition_Label Entrance_(Target)_State]. A line of this format named as transition.
The transition structure can be considered as [n_trans × 3] array.
    1) [Define] Crr_event: temporary variable to represent Current events value,
       Crr_s_sts: temporary variable to represent Current source state of an event
       represented by Crr_event. Crr_t_sts: temporary variable to represent Current
       target state of an event represented by Crr_event. All_Events: array of all events,
       Av_effect_list: Array of events that cause avalanche effect, n_trans: number of
       transitions in input, num_ev: number of all events, S_state: array of source state,
       T_state: array of target states. Events: array of events between source and target
       state in the same index.
    2) [Initialize] Crr_event=0, i=0, j=0, k=0, index=0;
    3) Crr_event=All_Events[i] take an event from all event list
    4) If (Crr_event==Events[j]) find the source and target state of Crr_events
       Then  Crr_s_sts=S_state[j] and Crr_t_sts=T_state[j];
    5) (If there is a target state equal to current source state or a source state equal to
       current target state and the events of these states is equal to current event in 2
       cases, the current event causes avalanche effect:)
      if(((Crr_s_sts==T_state[k])||(Crr_t_sts==S_state[k]))
      &&(Crr_event==Events[k]))
      then
                    Av_effect_list[index]=All_Events[i]
                  index=index+1
                    go to step 9
                end if.
    6) k=k+1;
    7) if (k<n_trans) goto Step 5;
    8) j=j+1 and if (j<n_trans) goto Step 4;
    9) k=0, j=0, i=i+1, and if (i<num_ev) goto Step 3;
Output: Av_effect_list: The set of events that cause avalanche effect.
end Algorithm Av_effect_detector.
```

Figure 9. The search algorithm used to detect events causing the avalanche effect.

2.4.3. An efficient method for the elimination of the avalanche effect problem

The avalanche effect problem occurs when a program skips over an arbitrary number of states during the same PLC scan cycle. The occurrence of an event can be realized in the LLD code by means of the rising or falling edge of a signal. The rising edge of a signal is detected by comparing the signal between 2 consecutive scan cycles; if the signal was low in the previous scan cycle and is now high, then a rising edge of the signal has been detected [24]. In [34], an efficient method was proposed for the elimination of the avalanche effect problem



```
C:\avalanche_code_exe\Av_effect_detector.exe
------------------------------------------------
 DETECTION of EVENTS THAT CAUSE AVALANCHE EFFECT
         Copyright 2010 Gokhan GELEN

------------------------------------------------

Please enter the name of .ads file:
--> test

LABELS of ALL EVENTS:

   1  2  3  4  5  6

LABELS of EVENTS THAT CAUSE AVALANCHE EFFECT:

   1  2  3

 Press R for new computation
 Press Q to  quit
-->
```

Figure 10. The screenshot of the “Av_effect_detector.exe” program for the input “TEST.ADS”.

when implementing an automaton as LLD. The method is applied only to events that cause the avalanche effect problem. Other events are treated as usual. The AEEM involves the following 3 steps:

1. After the initialization of the LLD code, SET a Boolean variable *Mi* with the occurrence of each event causing the avalanche effect problem ($i = 1, 2, \dots, n$; n: the number of events causing the avalanche effect problem).

2. RESET the Boolean variable *Mi* on each LLD rung implementing a transition, which includes the event causing the avalanche effect problem ($i = 1, 2, \dots, n$).

3. RESET the Boolean variable *Mi* at the end of the LLD code ($i = 1, 2, \dots, n$).

Using this method, an event (the rising or falling edge of a signal) is assigned to a memory bit *Mi* each time it occurs. When *Mi* is set, it can only be used once by 1 of the 2 or more transitions causing the avalanche effect problem, and then it is RESET by the very transition using *Mi*. If *Mi* is not used by one of the transitions, then it is automatically RESET at the end of the LLD code. This is a very simple and effective way of solving the avalanche effect problem. As an example to show how this method is applied, consider the automaton shown in Figure 4. In this automaton event ‘a’ is the cause of the avalanche effect. Therefore, the AEEM is applied only to this event. The LLD code for this automaton obtained using the AEEM proposed is depicted in Figure 11. In this code, the first rung initializes the automaton by setting q0 and resetting other states: q1, q2, and q3. The occurrence of ‘a’ M1 is set in rung 2. In rungs 3, 4, and 5, transitions [q0,a,q1], [q1,a,q2], and [q1,b,q3] are implemented, respectively. As stated before, in this automaton, transitions labeled with ‘a’ are the cause of the avalanche effect. Hence, M1 is reset at the end of rungs 3 and 4. If M1 is still set, then the last rung clears the memory bit M1 for another detection of event ‘a’. As another example, consider the automaton shown in Figure 6. In this automaton, event ‘a’ is the cause of the avalanche effect. The LLD code for this automaton obtained using the AEEM proposed is depicted in Figure 12. In this code, the first rung initializes the automaton by setting q0 and by resetting q1. With the occurrence of ‘a’, M1 is set in rung 2. In rungs 3 and 4, transitions [q0,a,q1] and [q1,a,q0] are implemented, respectively. As 2 transitions labeled with ‘a’ are the cause of the avalanche effect, M1 is reset at the end of rungs 3 and 4. Note that in this particular example, there is no need to reset M1 at the end of the LLD code, because M1 will be reset either in rung 3 or in rung 4.

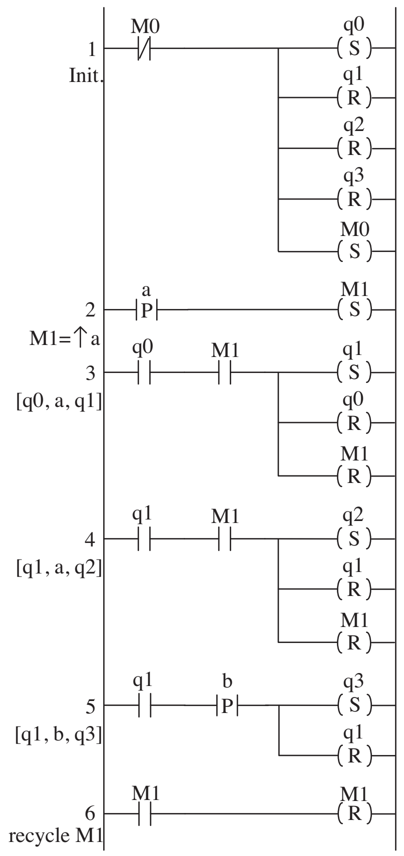


Figure 11. LLD code for the automaton shown in Figure 4 obtained using the AEEM.

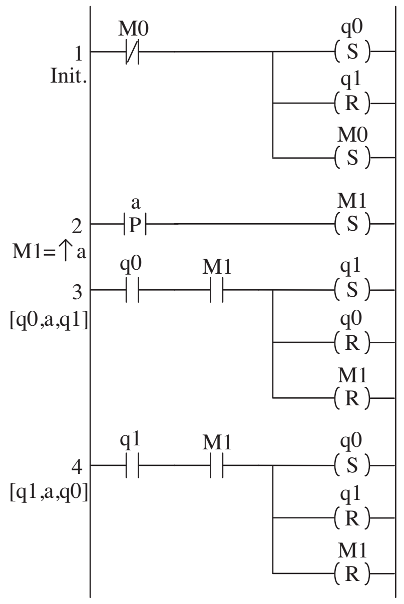


Figure 12. LLD code for the automaton shown in Figure 6 obtained using the AEEM.

2.4.4. Illustrative example

In this section, as an illustrative example, the automaton model consisting of 5 states and 13 transitions shown in Figure 7 is considered. There are 6 events, namely a, b, c, d, e, and f. As explained before, events labeled with a, b, and c are the cause of the avalanche effect. Therefore, in the PLC LLD implementation of this automaton, the AEEM is applied to only these 3 events. The PLC LLD code implementation of this automaton model can

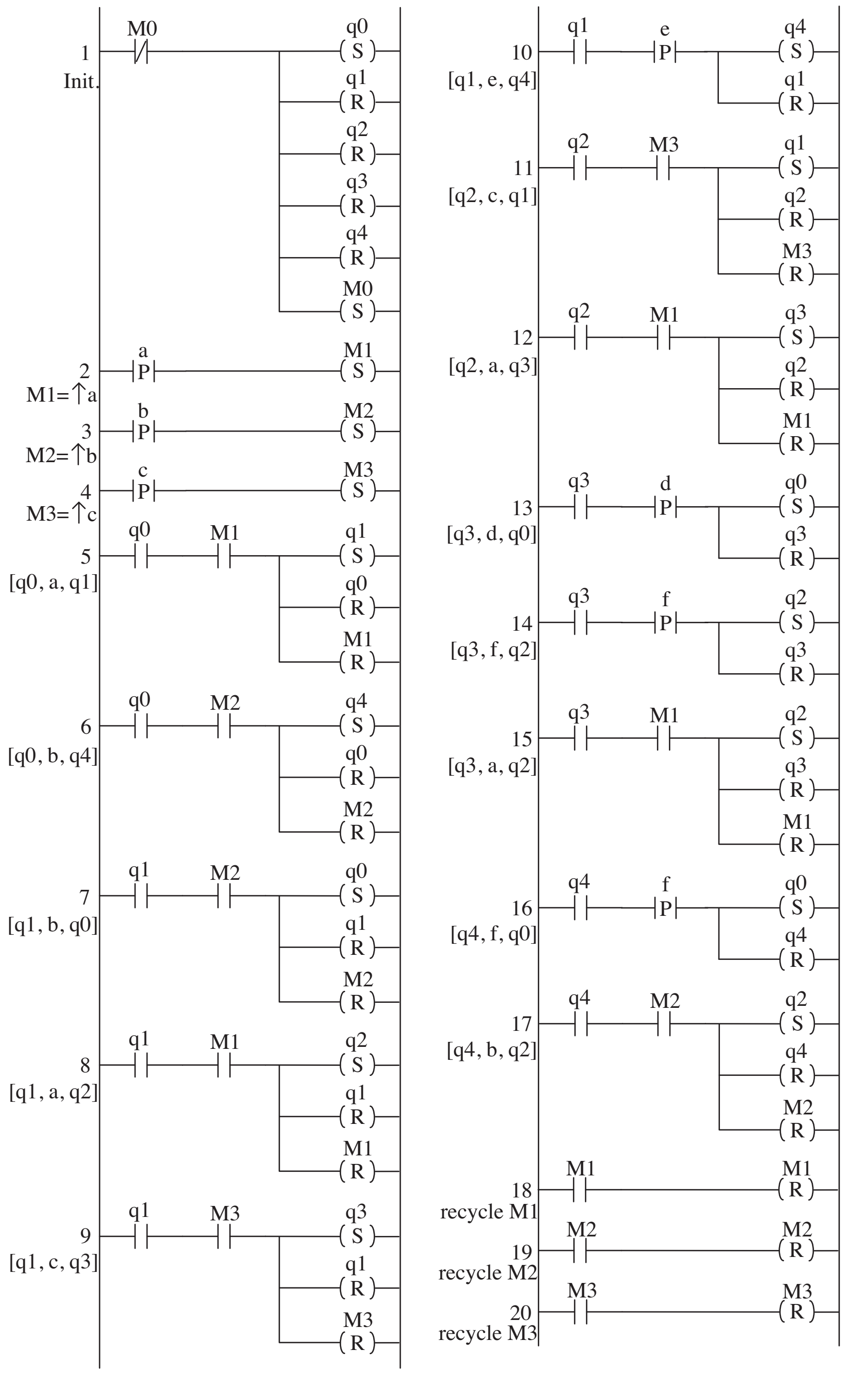

1 Init. M0 q0 (S) q1 (R) q2 (R) q3 (R) q4 (R) M0 (S)
2 M1=↑a a P M1 (S)
3 M2=↑b b P M2 (S)
4 M3=↑c c P M3 (S)
5 [q0, a, q1] q0 M1 q1 (S) q0 (R) M1 (R)
6 [q0, b, q4] q0 M2 q4 (S) q0 (R) M2 (R)
7 [q1, b, q0] q1 M2 q0 (S) q1 (R) M2 (R)
8 [q1, a, q2] q1 M1 q2 (S) q1 (R) M1 (R)
9 [q1, c, q3] q1 M3 q3 (S) q1 (R) M3 (R)
10 [q1, e, q4] q1 e P q4 (S) q1 (R)
11 [q2, c, q1] q2 M3 q1 (S) q2 (R) M3 (R)
12 [q2, a, q3] q2 M1 q3 (S) q2 (R) M1 (R)
13 [q3, d, q0] q3 d P q0 (S) q3 (R)
14 [q3, f, q2] q3 f P q2 (S) q3 (R)
15 [q3, a, q2] q3 M1 q2 (S) q3 (R) M1 (R)
16 [q4, f, q0] q4 f P q0 (S) q4 (R)
17 [q4, b, q2] q4 M2 q2 (S) q4 (R) M2 (R)
18 recycle M1 M1 M1 (R)
19 recycle M2 M2 M2 (R)
20 recycle M3 M3 M3 (R)

Figure 13. LLD implementation of the automaton shown in Figure 7.

be seen from Figure 13. The first rung initializes the automaton by setting q0 and resetting other states q1, q2, q3, and q4. With the occurrence of ‘a’ (respectively, ‘b’, ‘c’), M1 (respectively, M2, M3) is set in rung 2 (respectively, rung 3, rung 4). In rungs 5, 6, …, 17, all of the 13 transitions are implemented. As transitions labeled with ‘a’ (respectively, ‘b’, ‘c’) are the cause of the avalanche effect, M1 (respectively, M2, M3) is reset at the end of rungs 5, 8, 12, and 15 (respectively, rungs 6, 7, and 17; rungs 9 and 11). If M1 (respectively, M2, M3) is still set, then rung 18 (respectively, 19, 20) clears the memory bit M1 (respectively, M2, M3) for another detection of event ‘a’ (respectively, ‘b’, ‘c’). All of the LLD codes shown in this paper were implemented using a Siemens S7-300 PLC and they worked properly, as expected.

3. Reduced supervisor-based hybrid approach to supervisory control of DESs

A hybrid approach to the supervisory control of DESs, coupling RW supervisors to PNs was proposed in [11]. The applicability of this hybrid method to the PLC-based real-time control of DESs was then reported in [12]. The hybrid approach to the supervisory control of DESs proposed in [12] is recalled and improved here. In this paper, the reduced (simplified) RW supervisor is utilized instead of the ordinary one as in [12]. The supervisory control of a DES based on the hybrid approach with a reduced supervisor is updated and illustrated, as shown in Figure 14. The architecture consists of 4 parts: a) the DES to be controlled; b) the controller (supervisor); c) sensor readings, regarded as outputs from the DES and as inputs to the controller; and d) control actions, regarded as outputs from the controller and as inputs to the DES. The supervisor must guarantee that no forbidden state will be reached, that the specified target states remain reachable (nonblocking), and that the controlled behavior is maximally permissive, i.e. the supervisor does not unnecessarily constrain the system operation and is in this sense ‘optimal’.

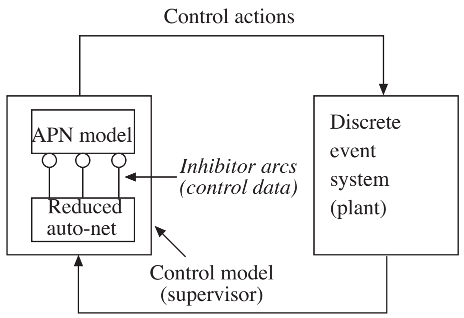


Figure 14. Supervisory control of a DES based on the hybrid approach with reduced auto-net.

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. The supervisor functions as a state-feedback controller, whose enabling/disabling control actions are output as a plant input, closing the feedback loop. The controlled behavior of the plant will be the subset of the uncontrolled behavior (i.e. sublanguage of uncontrolled event strings) that survives under supervision. The controlled model of the plant consists of the uncontrolled plant PN model itself and the controlling reduced auto-net, coupled together by means of inhibitor arcs, which represent the control data (or logic). The reduced auto-net (“reduced automaton net”) is a PN-like graphical representation of the reduced RW automaton supervisor, in which there is only 1 token and all places have a capacity of 1. In the reduced auto-net, as a graphical object, each state and transition arc of the reduced RW supervisor is shown, respectively, as a PN-like-place and a PN-like-transition. Initially, a single token is deposited in the place representing the initial state. The reduced auto-net represents the controlled behavior of the plant, its transitions represent either controllable or uncontrollable events, and its role is to track the plant behavior and to implement control (event disabling) action. For the latter, inhibitor arcs are used. When an event occurs in the plant, the reduced auto-net changes its state synchronously, in accordance with the location of its single token and the event in question. Transitions in the plant and the reduced auto-net are synchronized on the basis of shared labels. Because a transition with a given label may occur in more than one location in the reduced auto-net, the latter is not generally a PN and quite possibly cannot be represented as such. It is this feature that endows the hybrid representation with generality and flexibility.

The improved design and implementation steps can be summarized as follows.

1. Assume that an uncontrolled APN model (UAPNM) of a DES, with an initial marking, and a set of forbidden state specifications are given.
2. Check that the UAPNM is bounded and determine explicit bounds on the place markings.
3. Reduce the UAPNM, if possible, by PN reduction rules.
4. Convert the (reduced) UAPNM and the specifications into equivalent buffer models.
5. Apply the SCT to obtain a RW supervisor, say SUPER, and its control data, say SUPDAT.
6. Apply the SCT to obtain a reduced RW supervisor, say RSUPER, and its reduced control data, say RSUPDAT.
7. Convert RSUPER into a reduced auto-net representation, say RAUTONET.
8. Obtain the closed-loop (or controlled) reduced hybrid model, say CRHM, by coupling RAUTONET to UAPNM using inhibitor arcs according to RSUPDAT.
9. Finally, implement the CRHM on a PLC as a LLD.

The boundedness computation in step 2 can be carried out with available PN tools or (in simple cases) by inspection. The reduction techniques in step 3 are well known and preserve properties such as boundedness, liveness, and reversibility. For step 4, if place p is bounded in accordance with $\mathrm{M}(p) \leq \mathrm{b}$, say, then p is modeled by a buffer with capacity b (i.e. b + 1 states). For example, if an APN place has a token capacity of 3, then it can be represented by 4 states. Step 5 can be performed by standard SCT software; the package in [33] is used for computing SUPER and SUPDAT by procedures Supcon(.) and Condat(.), respectively. Step 6 is performed by the package in [33] for computing RSUPER and RSUPDAT by procedures Supreduce(.) and Condat(.), respectively. The detailed syntax of these steps is provided in the TCT-generated log files for the considered example. In the last step, it is necessary to check whether there is an avalanche effect problem in the reduced RW supervisor (automaton). For this, the "Av_effect_detector.exe" program is used. For the events causing the avalanche effect, the AEEM is utilized to solve the problem.

4. Real-time control of an experimental manufacturing system by using a reduced supervisor

In this section, an experimental manufacturing system is considered to show the applicability of the proposed hybrid method with a reduced RW supervisor to low level real-time supervisory control of DESs. The experimental manufacturing system together with the control specifications is taken from [12]. Therefore, the reader is referred to [12] for the detailed explanations. The first 5 steps are the same as those in [12]. The design steps follow, starting from Step 6 as shown below.

4.1. Step 6

The SCT is applied to obtain a reduced RW supervisor (RSUP) and its control data (RDAT). To accomplish this task, the PLANT and the SUPER automata models, and the control data (SUPDAT), are used in the Supreduce(.) and Condat(.) procedures of TCT software as follows.

RSUP = Supreduce(PLANT,SUPER,SUPDAT) (7,21; slb = 7)

The computed reduced RW supervisor is shown in Figure 15. The reduced control data are obtained as follows.

RDAT = Condat(PLANT,RSUP) Controllable.

RDAT

The control data are displayed as a list of supervisor states where disabling occurs, together with the events that must be disabled there.

Control data:

```
0: 61 71   1: 71      2: 61   3: 61
4: 61 71   5: 61 71   6: 71
```

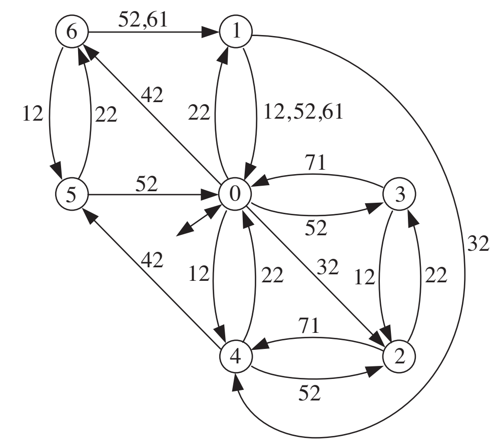


Figure 15. The reduced supervisor RSUP.

4.2. Step 7

The depiction RAUTONET of RSUPER is shown in Figure 16.

4.3. Step 8

The final closed-loop (controlled) reduced hybrid model (CRHM), which is nonblocking and maximally permissive, is obtained by coupling RAUTONET to UAPNM by inhibitor arcs according to RDAT. This is simply done by connecting the inhibitor arcs *In(p11, t6)*, *In(p11, t7)*, *In(p12, t7)*, *In(p13, t6)*, *In(p14, t6)*, *In(p15, t6)*, *In(p15, t7)*, *In(p16, t6)*, *In(p16, t7)*, and *In(p17, t7)* from places P = {p11, p12, ..., p17} to the controllable transitions t6 and t7, as shown in Table 2. The closed-loop (controlled) reduced compiled hybrid model CRHM is depicted in Figure 17.

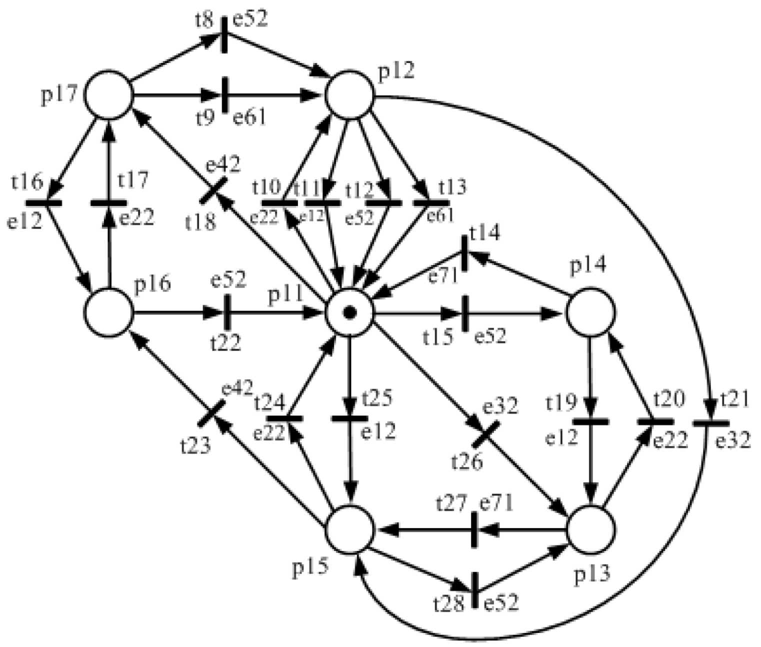
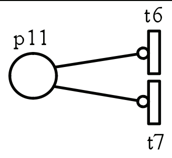
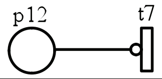
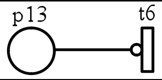
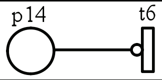
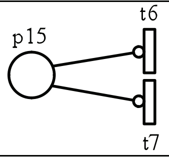
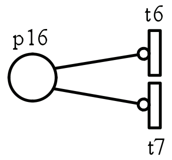
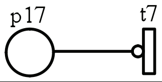


Figure 16. The reduced auto-net (RAUTONET) obtained from RSUP.

Table 2. Implementation of the reduced control data RDAT (control policy) for RSUP by inhibitor arcs.

Control data	Inhibitor arc implementation
0: 61 71	
1: 71	
2: 61	
3: 61	
4: 61 71	
5: 61 71	
6: 71	

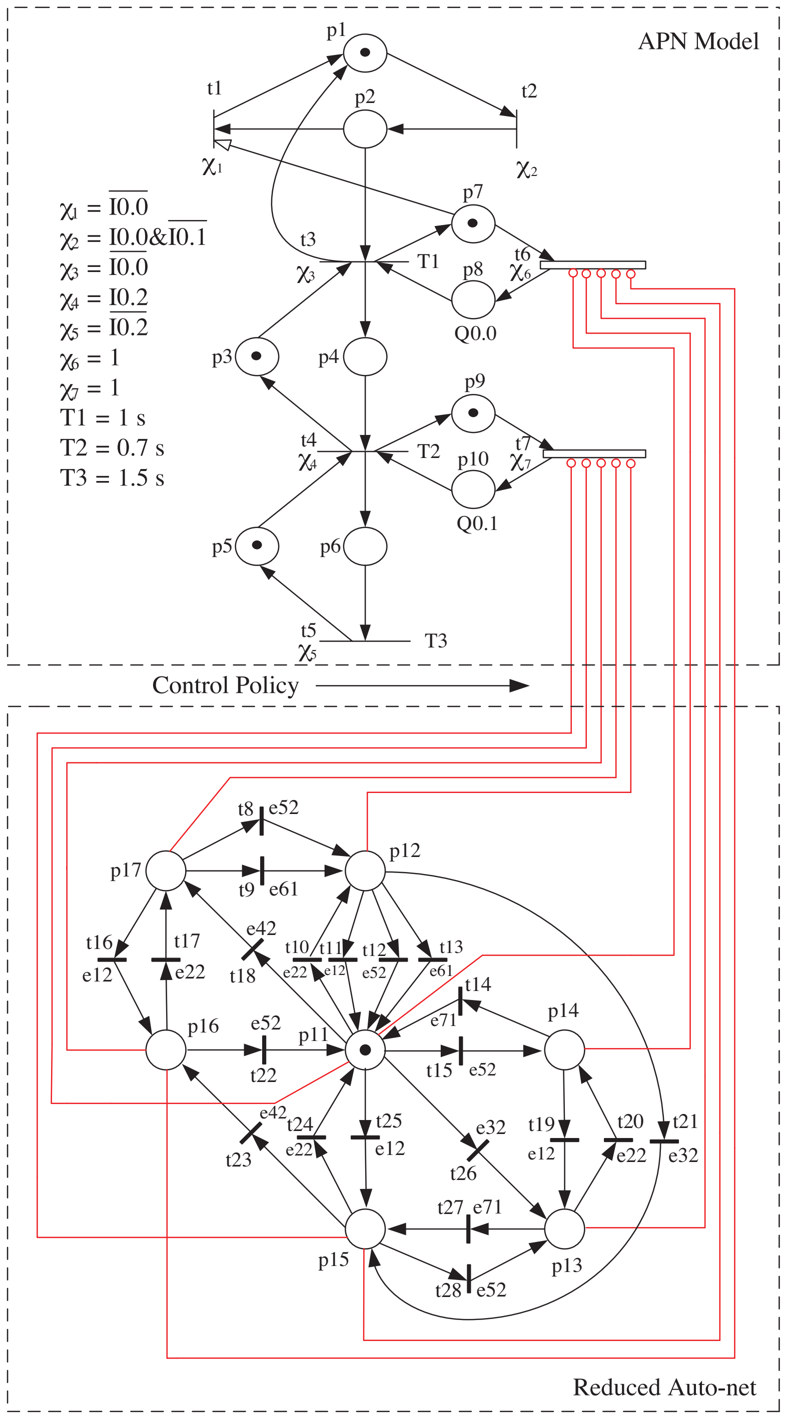
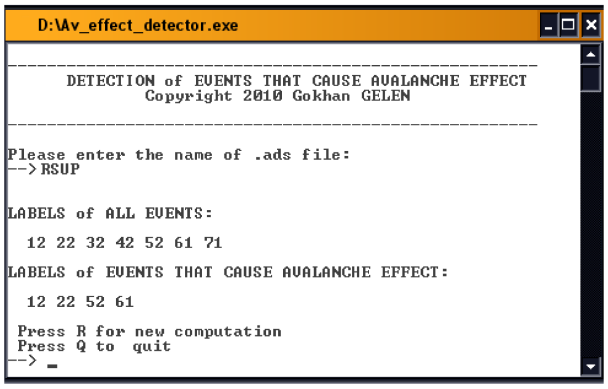


Figure 17. The closed-loop (controlled) reduced compiled hybrid model CRHM.

4.4. Step 9

When using a reduced RW supervisor, it is necessary to check whether there is an avalanche effect problem in the supervisor (automaton) before the implementation step. When the "Av_effect_detector.exe" program was run and the automaton RSUP.ADS, depicted in Figure 15, was given as input, this program provided the screenshot seen in Figure 18. This means that there is an avalanche effect problem in the automaton, shown in Figure 15, represented by RSUP.ADS, and thus events labeled with "12, 22, 52, and 61", i.e. events 'e12', 'e22', 'e52', and 'e61', are the cause of the avalanche effect problem. This means that the AEEM must be applied to these 4 events in the implementation step.



```
--------------------------------------------------------
       DETECTION of EVENTS THAT CAUSE AVALANCHE EFFECT
               Copyright 2010 Gokhan GELEN

--------------------------------------------------------

Please enter the name of .ads file:
--> RSUP

LABELS of ALL EVENTS:

  12 22 32 42 52 61 71

LABELS of EVENTS THAT CAUSE AVALANCHE EFFECT:

  12 22 52 61

 Press R for new computation
 Press Q to  quit
--> _
```

Figure 18. The screenshot of "Av_effect_detector.exe" program for the input "RSUP.ADS".

Now the implementation of the controlled model (CRHM) by means of a Siemens S7 300 PLC is considered. The TPL methodology is used for converting the CRHM into an LLD code. It should be noted that for proper functioning, the order of the LLD code must be arranged as follows: first, the initial marking is written; next, the LLD code for the APN model of plant is written; and finally, the LLD code related to the reduced auto-net is written. This is because after the initial marking is represented as LLD, the APN model mimics the system behavior through the sensory feedback and changes its state accordingly, and at the same time, i.e. on the same PLC scan cycle, the current state of the reduced auto-net is updated due to the shared events. According to the current state of the reduced auto-net and the control policy, if necessary, the behavior of the APN model is restricted by means of inhibitor arcs. Note that while "on delay timers" are associated only with the timed transitions in the APN model, the time evolutions of these timers are followed by the respective events within the auto-net. This improved order of the ladder rungs is slightly different from that proposed in [12].

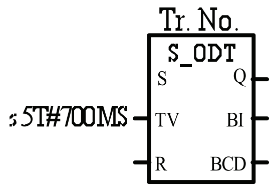
As a result, to convert the CRHM into an LLD code, 17 memory bits, namely p1, p2, ..., p17, are used to represent the places $P = (p1,\ p2,\ \ldots,\ p17)$ of the CRHM. On delay timers T1 with a 0.7 s time delay, T2 with a 1.5 s time delay, and T3 with a 1 s time delay are assigned to the timed-transitions t3, t4, and t5 of the APN model, respectively. After doing these assignments using direct mapping, the LLD code, shown in Figure 19, is obtained. This LLD code was written for a Siemens S7-300 (CPU 319) PLC. The LLD symbols (as used in this paper) of a Siemens S7-300 PLC are provided in Table 3.

The LLD code is structured in such a way that the network 0 (NW0) initializes the system by means of the initialization memory bit called "Init". The APN model is converted into LLD code at the networks from NW1 to NW5 and NW27, NW28, where networks 1, 2, ..., 5 and 27, 28 represent the transitions $T = (t1,\ t2,\ \ldots,\ t5,\ t6,\ t7)$, respectively. The reduced auto-net is converted into LLD at the networks from NW6 to NW26, where the networks 6, 7, ..., 26 represent the transitions T = (t8, t9, ..., t28), respectively. Finally, action places p8 and p10 are represented by networks 29 and 30, respectively. By adopting this concept, further clarity can be added to the system documentation and it is easy to understand and modify the LLD code if necessary. Using 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 Figure 4 [12]. The LLD code representing the controlled model, i.e. the supervisor, implemented the control specifications as required and did not unnecessarily constrain the behavior of the experimental manufacturing system. The physical system performed very well under the proposed control, as described by the specifications.

The LLD code shown in Figure 19 can be validated using the timing diagrams of the inputs, outputs, and the variables used within the LLD code, similar to that given in [12]. In order to accomplish such a task, a scenario can be characterized for the controlled plant to follow the supervisor's disabling commands and also to inspect the PLC inputs, the variables used, and the output signals.

Finally, in this section, the implementation of the events, namely 'e12', 'e22', 'e52', and 'e61', that cause the avalanche effect is considered. To eliminate the avalanche effect problem, the AEEM is used. To eliminate the avalanche effect due to event 'e12', the memory bit with the same name, i.e. e12, is set at NW1 when transition $t1$ of APN model is fired. The memory bit e12 is then reset at the end of networks 9, 14, 17, and 23. To eliminate the avalanche effect due to event 'e22', the memory bit with the same name, i.e. e22, is set at NW2 when transition $t2$ of the APN model is fired. The memory bit e22 is then reset at the end of networks

Table 3. Some LLD symbols for the Siemens S7-300 PLC.

LLD symbol	Definition
S	Set
R	Reset
T	Timer
I	Input
Q	Output
M	Memory bit
—| |—	Normally open contact
—|/|—	Normally closed contact
 Tr. No. S_ODT S Q S5T#700MS TV BI R BCD	On 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 timers

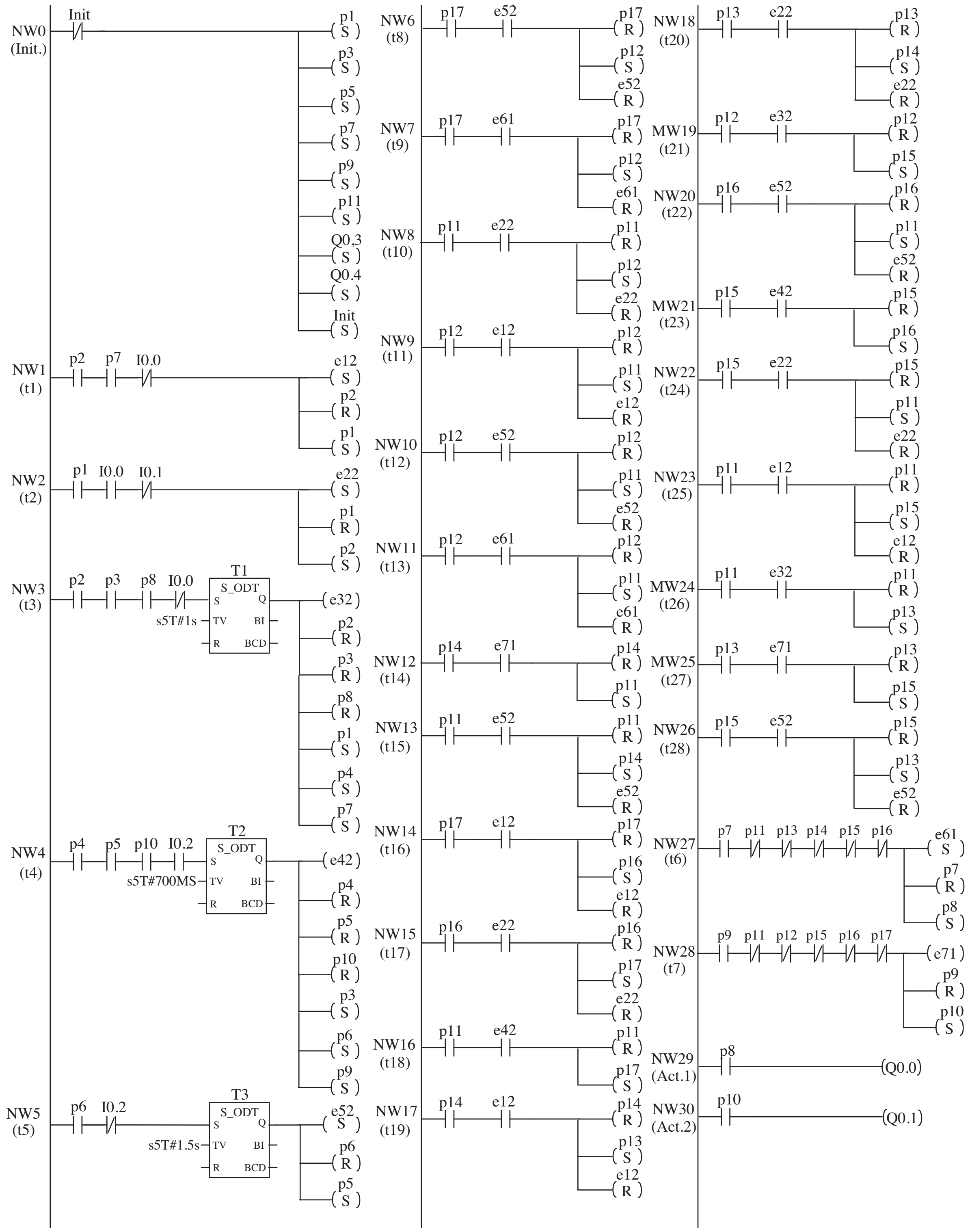


Figure 19. The LLD code implementation of the CRHM.

8, 15, 18, and 22. To eliminate the avalanche effect due to event 'e52', the memory bit with the same name, i.e. e52, is set at NW5 when the timed transition $t5$ of the APN model is fired. The memory bit e52 is then reset at the end of networks 6, 10, 13, 20, and 26. Finally, to eliminate the avalanche effect due to event 'e61', the memory bit with the same name, i.e. e61, is set at NW27 when the controllable transition $t6$ of the APN model is fired. The memory bit e61 is then reset at the end of networks 7 and 11. In this example, there is no need to reset memory bits e12, e22, e52, and e61 at the end of the LLD code, because they will all be reset in their related networks.

5. Discussion

The reduced RW supervisor with a strictly minimal state size is the control equivalent to the original monolithic RW supervisor. For example, when comparing the RW supervisor shown in Figure 10 of [12] and the reduced RW supervisor shown here in Figure 15, it can be seen that they have 12 and 7 states, respectively. From the practical point of view, it is desirable to implement a supervisor with the least possible number of states, because for every state it is necessary to use a memory bit. In this respect it is better to implement the closed-loop reduced model shown in Figure 18 instead of that shown in Figure 12 of [12]. Table 4 is provided to make a comparison between the hybrid controller proposed in [12] and the reduced hybrid controller proposed in this paper. It can be seen from Table 4 that the controller obtained with the reduced RW supervisor has fewer transitions and places. This results in less memory usage (380 bytes compared to 422 bytes) in the PLC implementation of this controller. However, to implement the controller obtained with the reduced RW supervisor, it is necessary to use some additional program elements, which in return requires the use of additional memory bytes. If the number of events causing the avalanche effect problem is too high, then it may be possible to have a PLC implementation of a controller obtained with the reduced RW supervisor requiring more memory bytes than the ordinary one. A detailed comparative study is necessary in order to assess the PLC memory usage for different control scenarios, in which the RW supervisor and the reduced RW supervisor are relatively bigger than those used in this paper and that in [12].

Table 4. Comparison between the hybrid controller proposed in [12] and the reduced hybrid controller proposed in this paper.

Hybrid controller	Number of transitions	Number of places	Total number of transitions and places	PLC memory usage (bytes)
Plant + the RW supervisor from [12]	30	22	52	422
Plant + the reduced RW supervisor proposed in this paper	28	17	45	380

6. Conclusions and future research

In this paper, an improved hybrid (mixed PN/automaton) approach is proposed. The improvements include the use of a reduced RW supervisor and the detection and elimination of the avalanche effect problem. The reduced RW supervisor is utilized in the hybrid model to reduce the high memory requirements in PLC-based implementations. However, the PLC implementation of the hybrid model with the reduced RW supervisor results in the avalanche effect problem if there are events appearing on successive transitions within the reduced RW supervisor. A recently established method is used to both detect and eliminate the avalanche effect problem successfully. The applicability of the improved hybrid approach has been demonstrated by the PLC-based real-time control of an experimental manufacturing system.

A detailed comparative study is planned to be carried out in order to assess the PLC memory usage for different control scenarios in which the RW supervisor and the reduced RW supervisor are relatively bigger than those used in this paper and that in [12].

The preliminary results obtained show that it seems to be possible to adopt modular or decentralized RW supervisors in the hybrid method with even less PLC memory requirements. Therefore, further studies will be carried out to obtain a hybrid method with modular or decentralized RW supervisors.

Acknowledgment

This work was supported by a research grant from the Scientific and Technological Research Council of Turkey (TÜBİTAK) under project number TÜBİTAK-107E125.

References

[1] P.J. Ramadge, W.M. Wonham, "Supervisory control of a class of discrete event processes", SIAM Journal on Control and Optimization, Vol. 25, pp. 206–230, 1987.

[2] P.J. Ramadge, W.M. Wonham, "The control of discrete event systems", Proceedings of IEEE, Vol. 77, pp. 81–98, 1989.

[3] W.M. Wonham, P.J. Ramadge, "On the supremal controllable sublanguage of a given language", SIAM Journal on Control and Optimization, Vol. 25, pp. 637–659, 1987.

[4] A. Giua, "Petri net techniques for supervisory control of discrete event systems", 1st International Workshop on Manufacturing and Petri Nets, pp. 1–21, 1996.

[5] A. Giua, F. DiCesare, M. Silva, "Generalized mutual exclusion constraints on nets with uncontrollable transitions", IEEE International Conference on Systems, Man, and Cybernetics, pp. 974–979, 1992.

[6] M. Uzam, "Petri-net-based supervisory control of discrete event systems and their ladder logic diagram implementations", PhD, University of Salford, Salford, UK, 1998.

[7] M. Uzam, A.H. Jones, "Design of a discrete event control system for a manufacturing system using token passing ladder logic", IMACS Multiconference, Symposium on Discrete Events and Manufacturing Systems, pp. 513–518, 1996.

[8] M. Uzam, A.H. Jones, "Discrete event control system design using automation Petri nets and their ladder diagram implementation", The International Journal of Advanced Manufacturing Technology, Vol. 14, pp. 716–728, 1998.

[9] M. Uzam, A.H. Jones, I. Yücel, "Using a Petri-net-based approach for the real-time control of an experimental manufacturing system", The International Journal of Advanced Manufacturing Technology, Vol. 16, pp. 498–515, 2000.

[10] M. Uzam, A.H. Jones, "A new Petri-net-based synthesis technique for supervisory control of discrete event systems", Turkish Journal of Electrical Engineering & Computer Sciences, Vol. 10, pp. 85–109, 2002.

[11] M. Uzam, W.M. Wonham, "A hybrid approach to supervisory control of discrete event systems coupling RW supervisors to Petri nets", The International Journal of Advanced Manufacturing Technology, Vol. 28, pp. 747–760, 2006.

[12] M. Uzam, G. Gelen, "The real-time supervisory control of an experimental manufacturing system based on a hybrid method", Control Engineering Practice, Vol. 17, pp. 1174–1189, 2009.

[13] B.A. Brandin, "The real-time supervisory control of an experimental manufacturing cell", IEEE Transactions on Robotics and Automatation, Vol. 12, pp. 1–14, 1996.

[14] S.C. Lauzon, J.K. Mills, B. Benhabib, "An implementation methodology for the supervisory control of flexible manufacturing workcells", Journal of Manufacturing Systems, Vol. 16, pp. 91–101, 1997.

[15] A. Ramirez-Serrano, S.C. Zhu, B. Benhabib, "Moore automata for the supervisory control of robotic manufacturing workcells", Autonomous Robots, Vol. 9, pp. 59–69, 2000.

[16] A. Ramirez-Serrano, S.C. Zhu, S.K.H. Chan, S.S.W. Chan, M. Ficocelli, B. Benhabib, "A hybrid PC/PLC architecture for manufacturing system control-theory and implementation", Journal of Intelligent Manufacturing, Vol. 13, pp. 261–281, 2002.

[17] M.H. de Queiroz, J.E.R. Cury, "Synthesis and implementation of local modular supervisory control for a manufacturing cell", 6th International Workshop on Discrete Event Systems, pp. 377–382, 2002.

[18] J. Liu, H. Darabi, "Ladder logic implementation of Ramadge-Wonham supervisory controller", 6th International Workshop on Discrete Event Systems, pp. 383–389, 2002.

[19] D. Gouyon, J.F. Petin, A. Gouin, "Pragmatic approach for modular synthesis and implementation", International Journal of Production Research, Vol. 42, pp. 2839–2858, 2004.

[20] S. Manesis, K. Akantziotis, "Automated synthesis of ladder automation circuits based on state-diagrams", Advances in Engineering Software, Vol. 36, pp. 225–233, 2005.

[21] A.D. Vieira, J.E.R. Cury, M.H. de Queiroz, "A model for implementation of supervisory control of a discrete event systems", IEEE Conference on Emerging Technologies and Factory Automation, pp. 225–232, 2006.

[22] M. Moniruzzaman, P. Gohari, "Implementing supervisory control maps with PLC", American Control Conference, pp. 3594–3599, 2007.

[23] M. Uzam, G. Gelen, R. Dalcı, "A new approach for the ladder logic implementation of Ramadge-Wonham supervisors", 22nd International Symposium on Information, Communication and Automation Technologies, pp. 113–119, 2009.

[24] M. Fabian, A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems", Proceedings of the 37th IEEE Conference on Decision and Control, Vol. 3, p. 3305–3310, 1998.

[25] İ.T. Hasdemir, S. Kurtulan, L. Gören, "An implementation methodology for supervisory control theory", The International Journal of Advanced Manufacturing Technology, Vol. 36, pp. 373–385, 2008.

[26] C.G. Cassandras, S. Lafortune, Introduction to Discrete Event Systems, Dordrecht, Kluwer Academic Publishers, 1999.

[27] W.M. Wonham, "Supervisory control of discrete event systems", ECE Department, University of Toronto, 1997–2011. Available at http://www.control.utoronto.ca/DES/, updated 2011.07.01.

[28] C.A. Petri, "Kommunikation mit automaten schriften des rheinisch", Westfalischen Inst. fur Intrumentelle Mathematik and der Universitat Bonn. Translation by C.F. Green, Applied Data Research Inc., Suppl. 1 to Tech Report RADC-TR-65-337, New York, 1962.

[29] R. David, H. Alla, Petri Nets, Grafcet: Tools for Modeling Discrete Event Systems, New York, Prentice Hall, 1992.

[30] A.H. Jones, M. Uzam, A.H. Khan, D. Karimzadgan, S.B. Kenway, "A general methodology for converting Petri nets into ladder logic: the TPLL methodology", Rensselaer's 1st International Conference on Computer Integrated Manufacturing and Automation Technology, pp. 357–362, 1996.

[31] S. Peng, M.C. Zhou, "Ladder diagram and Petri net based discrete-event control design methods", IEEE Transactions on Systems, Man, and Cybernetics Part C, Vol. 34, pp. 523–531, 2004.

[32] IEC 61131-3, A Standard of the International Electrotechnical Commission (IEC): Programmable Controllers – Part 3: Programming Languages, 2003.

[33] TCT, A Software Tool Supporting Supervisory Control Theory. Systems Control Group, ECE Department, University of Toronto. Available at www.control.utoronto.ca/DES, accessed in 2011.

[34] M. Uzam, G. Gelen, R. Dalcı, "An effective method for detection and elimination of avalanche effect problem", IEEE International Conference on Mechatronics and Automation, pp. 1988–1993, 2010.