*The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland*

DIGITAL HARDWARE IMPLEMENTATION OF PETRI NET BASED SPECIFICATIONS: DIRECT TRANSLATION FROM SAFE AUTOMATION PETRI NETS TO CIRCUIT ELEMENTS

Murat UZAM, Mutlu AVCI and M. Kürşat YALÇIN

Niğde Üniversitesi, Mühendislik-Mimarlık Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü,
51100 NİĞDE, TURKEY. Tel: ++ 90 388 225 01 15, Fax: ++ 90 388 225 01 12,
e-mail: *murat_uzam@hotmail.com, <mutluavci, kursat_yalcin>@yahoo.com*

Abstract: *In this paper, a new method is proposed for the digital hardware implementation of Petri net based specifications. The purpose of this paper is to introduce a new discrete event control system paradigm, where the control system is modelled with extended Petri nets and implemented as an asynchronous controller using circuit elements. The results provided in this paper on the digital hardware implementation of Petri nets may be view as a better version of a previously introduced method [1], in terms of the implementation of transitions.*

Key Words. *Discrete event systems, Petri nets, hardware implementation, asynchronous controllers.*

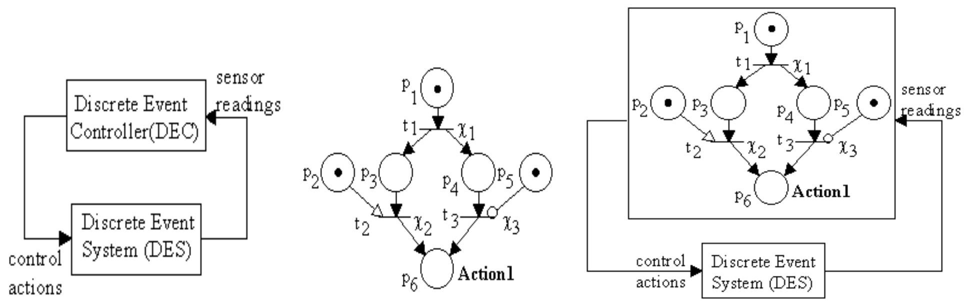
1. INTRODUCTION

Discrete event systems (DES), examples of which include communication networks, manufacturing systems, robots, etc., exhibit properties such as non-determinism, conflict, and concurrency. The study (i.e. design, analysis, synthesis, etc.) of DES has been carried out mainly by using two modelling techniques: finite state machines (FSM) and Petri nets. FSM based studies suffer from so called state explosion problem. FSMs provide sequential models. When using FSMs graphical visiualisation of the modelled system can not be realised easily [2]. Petri nets have been used as an alternative formalism for the study of DESs due to their easily understood graphical representation in addition to their well-formed mathematical formalism. In this paper, we are concerned with the control of DES and we use Petri nets. The control of DESs is done firstly by modelling the controller as a Petri net model and then by implementing it in software or hardware. The implementation is carried out by simulating the Petri net model in terms of software or hardware structures. The software implementation has been done using either high level languages or low level languages. Synchronous or asynchronous controllers of Petri net based specifications have been obtained as hardware implementation. For a detailed information on Petri nets and digital hardware design, the reader is referred to [3]. In an asynchronous circuit, there is no global clock, i.e. they are self timed. Asynchronous circuits can be viewed as hardwired versions of parallel and distributed programs. The program statements are physical components, i.e. logic gates, memory elements, etc. Asynchronous circuits are better than the synchronous counterparts in terms of performance, robustness, low power, low electromagnetic emission, modularity and re-use, and testability [3]. In this paper, we deal with asynchronous circuit implementation of Petri net based specifications. This type of implementation is carried out based on the idea of “physical simulation” and achieved by associating each place in the Petri net with a memory latch. Examples of this style can be found in [1, 4, 5, 6].

In this paper, a new method is proposed for the digital hardware implementation of Petri net based specifications. The purpose of this paper is to introduce a new discrete event control system paradigm, where the control system is modelled with extended Petri nets and implemented as an asynchronous controller using circuit elements. The results provided in this paper on the digital hardware implementation of Petri nets may be view as a better version of a previously introduced method [1], in terms of the implementation of transitions. The remainder of this paper is organised as follows: The next section defines safe automation Petri nets (SAPN). In the 3rd section, the digital hardware implementation of SAPN is explained. Finally, conclusions are given.

2. SAFE AUTOMATION PETRI NETS

Automation Petri nets (APN) have recently been introduced as a new formalism for the design of Discrete Event Control Systems [2]. Since ordinary Petri nets do not deal with sensors and actuators, the Petri net concepts are extended, by including actions and sensor readings as formal structures within the APN. These extensions involve extending the Petri nets to accommodate sensor signals at transitions and to assign level actions to places (and similarly to assign impulse actions to transitions). In this section, we define Safe (1-bounded) Automation Petri nets (SAPN) to be used for direct translation from Petri nets to circuit elements. A typical discrete event control system (DECS) is shown in Figure 1.(a). It consists of a discrete event system (DES), to be controlled and a discrete event controller (DEC). Sensor readings are regarded as inputs from the DES to the DEC, and control actions are considered as outputs from the DEC to the DES. The main function of the DEC is to supervise the desired DES operation and to avoid forbidden operations. To do this, the DEC processes the sensor readings and then it forces the DES to conform to the desired specifications through control actions. Petri nets can be used to design such DECs. However, ordinary Petri nets do not deal with actuators or sensors. Because of this, it is necessary to define a Petri net-based controller (Automation Petri net - APN), which can embrace both actuators and sensors within an extended Petri net framework. An SAPN is shown in Figure 1.(b). In the SAPN, 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 Petri net pre-conditions to fire transitions. In the SAPN, two types of actuations can be considered, namely impulse actions and level actions. Level actions are associated with places, while impulse actions are associated with transitions. With these additional features, it is possible to design Discrete Event Control Systems. Figure 1.(c) shows how an SAPN can be used as a DEC in a DECS.



(a) (b) (c)

Fig. 1. (a). A typical discrete event control system (DECS). (b). Safe Automation Petri Net (SAPN). (c). APN used as a controller in a DECS.

Formally, a Safe Automation Petri Net can be defined as follows:

$$\mathbf{SAPN = (P, T, Pre, Post, In, En, \chi, Q, M_0)}$$

Where,

- $P = \{p_1, p_2, ..., p_n\}$ is a finite, nonempty set of places,
- $T = \{t_1, t_2, ..., t_m\}$ is a finite, nonempty set of transitions, $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$,
- Pre: $(P \times T) \rightarrow \{0,1\}$ is an input function that defines ordinary arcs from places to transitions,
- Post: $(T \times P) \rightarrow \{0,1\}$ is an output function that defines ordinary arcs from transitions to places,
- In: $(P \times T) \rightarrow \{0,1\}$ is an inhibitor input function that defines inhibitor arcs from places to transitions,
- En: $(P \times T) \rightarrow \{0,1\}$ is an enabling input function that defines enabling arcs from places to transitions,
- $\chi = \{\chi_1, \chi_2, ..., \chi_m\}$ is a finite, nonempty set of firing conditions associated with transitions,
- $Q = \{q_1, q_2, ..., q_n\}$ is a finite set of level actions that might be assigned to places or impulse actions that might be assigned to transitions,
- $M_0 : P \rightarrow \{0,1\}$ is the initial marking.

The SAPN consists of two types of nodes called *places*, represented by circles (O), and *transitions*, represented by bars (—). There are three types of arcs used in the SAPN, namely, *ordinary arcs*, represented by a directed arrow ($\longrightarrow$), *inhibitor arcs*, represented by an arc, whose end is a circle (—○), and finally *enabling arcs*, represented by a directed arrow, whose end is empty (—▷). Directed ordinary arcs connect places to transitions and vice versa, while enabling and inhibitor arcs connect only places to transitions. The number of tokens in places represent the current state of the system and firing of a transition represents the movement of the system from one state to another state. Each transition has a set of input and output places, which represent the pre-condition and post-condition of the transition. The level actions (Q), may be assigned to places, and the impulse actions may be assigned to transitions. Level actions may be enabled when there is a token at a place, while impulse actions may be enabled at the instant when a transition is fired. More than one action may be assigned to a place or a transition. Firing conditions in the SAPN are recognised by external events (signals) such as sensor readings, switch positions, etc. Six firing conditions, may be associated with a transition *t*: χ, $\overline{\chi}$, $\uparrow\chi$, $\uparrow\overline{\chi}$, $\chi\downarrow$ and, $\overline{\chi\downarrow}$. The firing condition χ is a Boolean variable that can be 0, in which case related transition *t* is not allowed to fire, or it can be 1, in which case related transition *t* is allowed to fire if it is enabled, i.e. all input places have one token each. The firing condition $\overline{\chi}$ is the complement of the firing condition χ and is a Boolean variable that can be 1, in which case related transition *t* is not allowed to fire, or it can be 0, in which case related transition *t* is allowed to fire if it is enabled. The rising-edge-firing-condition $\uparrow\chi$ is a Boolean variable that can be 0, in which case related transition *t* is not allowed to fire, or it can be 1, in which case related transition *t* is allowed to fire if it is enabled. The complement-rising-edge-firing condition $\overline{\uparrow\chi}$ is a Boolean variable that can be 1, in which case related transition *t* is not allowed to fire, or it can be 0, in which case related transition *t* is allowed to fire if it is enabled. The falling-edge-firing-condition $\chi\downarrow$ is a Boolean variable that can be 1, in which case related transition *t* is not allowed to fire, or it can be 0, in which case related transition *t* is allowed to fire if it is enabled. Finally, The complement-falling-edge-firing-condition $\overline{\chi\downarrow}$ is a Boolean variable that can be 0, in which case related transition *t* is not allowed to fire, or it can be 1, in which case related transition *t* is allowed to fire if it is enabled. The marking of the SAPN is represented by the number of tokens in places. Tokens are represented by black dots (•). Movement of tokens between places describes the evolution of the SAPN and is accomplished by the firing of the enabled transitions. The following rules are used to govern the flow of tokens:

Enabling Rules:

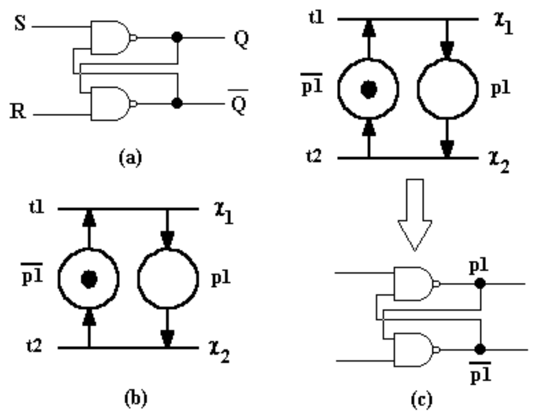
1. If the input place p_1 of a transition t_1 is connected to t_1 with an ordinary arc Pre(p_1,t_1), then t_1 is said to be enabled when p_1 contains a token, i.e., M(p_1) = 1.
2. If the input place p_1 of a transition t_1 is connected to t_1 with an enabling arc En(p_1,t_1), then t_1 is said to be enabled when p_1 contains a token, i.e., M(p_1) = 1.
3. If the input place p_1 of a transition t_1 is connected to t_1 with an inhibitor arc In(p_1,t_1), then t_1 is said to be enabled when p_1 contains no token, i.e., M(p_1) = 0.

Firing Rules: In the SAPN, an enabled transition *t* can or can not fire depending on the external firing condition χ of *t*. These firing conditions can be either of the above mentioned firing conditions, i.e. χ, $\overline{\chi}$, $\uparrow\chi$, $\overline{\uparrow\chi}$, $\chi\downarrow$ or $\overline{\chi\downarrow}$, namely positive level, zero level, rising edge, complement rising edge, falling edge or complement falling edge of a (signal) sensor reading or a switch position. Broadly speaking, a firing condition of a transition *t* may include more than one sensor reading with 'AND', 'OR' and 'NOT' logical operators. When dealing with more than one sensor readings as firing conditions, the logical operators of 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 with a related firing condition χ, it removes one token from each input place p_i and deposits, at the same time, one token to each output place p_o. It should be noted that, the firing of an enabled transition *t* does not change the marking of the input places that are connected to *t* only by an enabling or an inhibitor arc.

3. DIGITAL HARDWARE IMPLEMENTATION OF SAFE AUTOMATION PETRI NETS : DIRECT TRANSLATION FROM SAPN TO CIRCUIT ELEMENTS

The direct translation method from SAPN to circuit elements, proposed in this paper, is based on the idea of physical simulation of every Petri net marking reachable from the initial marking in terms of the state of the circuit. In order to achieve the direct translation, the following three steps are followed: i) each place in the SAPN is associated with a memory element, i.e. an SR-flip-flop (SR-latch), ii) each transition in the SAPN is implemented with a logical gate (NAND gate), iii) the initial marking is set-up by using an RC (Resistor and Capacitor) element. Let us now consider each of these three steps:

i) The first step in achieving the direct translation is to use a memory element to represent the presence or absence of a token in a place. If there is a token in a place then the output of the memory element is set to 1. In contrast, if there is no token in a place then the output of the memory element is reset to 0. To implement this operation we use an SR-flip-flop, as shown in Fig. 2.(a). An SR-flip-flop is constructed from two NAND gates connected back to back. The cross-coupled connections from the output of one gate to the input of the other gate constitutes a feedback path. Therefore, the circuit is classified as asynchronous. Each flip-flop has two outputs; Q and $\overline{Q}$, and two inputs; set (S) and reset (R). The truth table of the SR-flip-flop is given in Fig. 2.(d). The application of a momentary 0 to the S input causes output Q to go to 1 and $\overline{Q}$ to go to 0. The outputs of the circuit do not change when the S input returns to 1. A momentary 0 applied to the R input causes an output of $Q = 0$ and $\overline{Q} = 1$. The state of the flip-flop is always taken from the value of its normal output Q. When $Q = 1$, we say that the flip-flop stores a 1 and is in the *set* state. When $Q = 0$, we say that the flip-flop stores a 0 and is in the *reset* state. The SR-flip-flop manifests an undesirable condition if both inputs go to 0 simultaneously. When both inputs are 0, outputs Q and $\overline{Q}$ will go to 1, a condition which is normally meaningless in flip-flop operation. In Fig. 2.(b) an SAPN is shown, in which there are two places; p1 and its complement $\overline{p1}$, and two transitions; t1 and t2, with firing conditions χ_1 and χ_2 respectively. This is an explicit representation of the safe place p1. The implementation of places using the SR-flip-flop is shown in Fig. 2.(c). Output Q of the SR-flip-flop is used to represent place p1 and output $\overline{Q}$ is used to represent place $\overline{p1}$. When there is a token in p1, the SR-flip-flop is set, i.e. $Q = 1$ and $\overline{Q} = 0$. When there is a token in $\overline{p1}$, the SR-flip-flop is reset, i.e. $Q = 0$ and $\overline{Q} = 1$. It is assumed that the model will not permit both outputs becoming $Q = 1$ and $\overline{Q} = 1$. That is to say that the designer must take some action to assure that S = R = 0 will never occur.



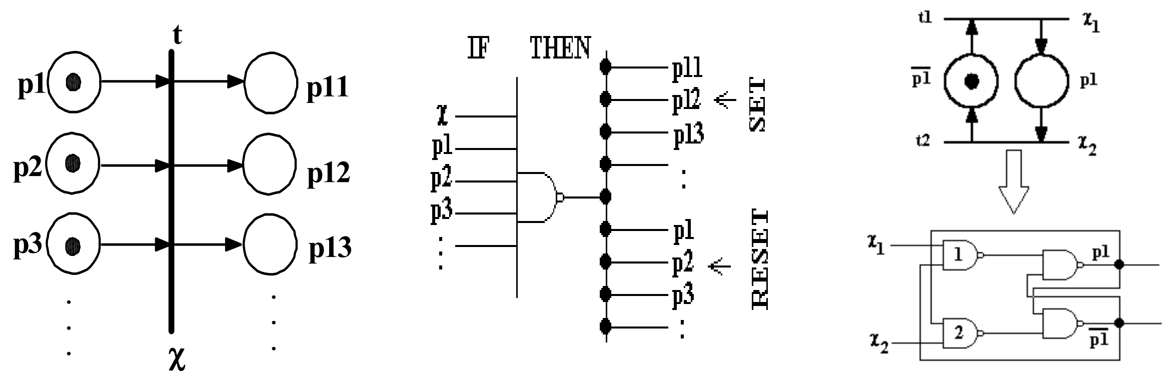
S	R	Q	$\overline{Q}$	Comment
0	1	1	0	Set
1	1	1	0	After S = 0 and R = 1
1	0	0	1	Reset
1	1	0	1	After S = 1 and R = 0
0	0	1	1	Not allowed

(d)

Fig. 2. a). An SR-flip-flop, b) an SAPN, c) the implementation of places. d). The truth table of the SR-flip-flop,

ii) The second step in achieving the direct translation is to use a NAND gate to implement transition in SAPN. The behaviour of a transition in SAPN may be summarised as follows: IF there is a token each in the input places of a transition *t* AND the firing condition χ of *t* occurs, THEN all the tokens are removed from the input places and one token each is deposited to the output places of *t*. To show how this behaviour is implemented, the SAPN shown in Fig. 3.(a) is used. In this case, the transition *t* fires when all input places p1, p2, p3,

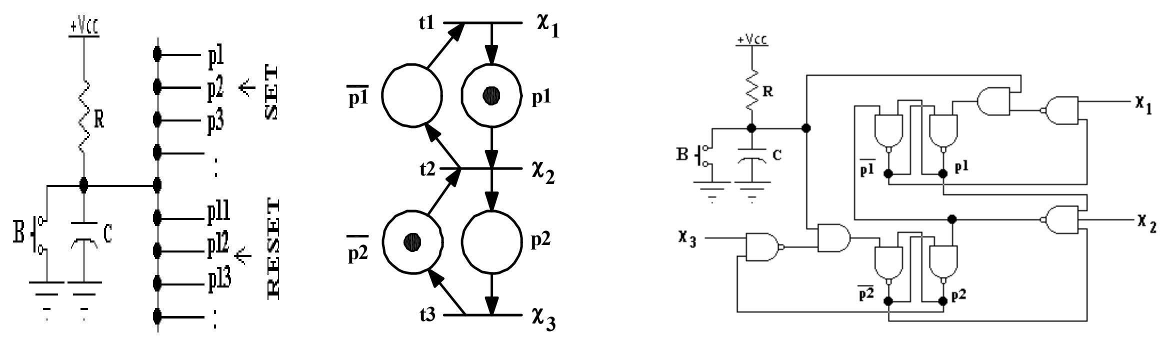
... have one token each and the firing condition χ occurs. When *t* fires it removes all the tokens from the input places p1, p2, p3, ..., and at the same time, it deposits one token each to the output places, p11, p12, p13,... . To implement the transition *t*, the structure shown in Fig. 3.(b) is used. In this case, when all input flip-flops are set and χ occurs *t* is fired by resetting all the input flip-flops and at the same time by setting all the output flip-flops. Please note that the difference between our approach and [1] is that, in [1] the removal of tokens from input places and adding tokens to output places has a duration and there is an intermediate state between the two operations, while in our approach the removal of tokens from input places and adding tokens to output places is instantaneous. Fig. 3.(c) shows the implementation of transitions t1 and t2 of Fig. 2(c). In the SAPN, t1 fires when there is a token in $\overline{p1}$ and χ_1 occurs. When fired, t1 removes the token from $\overline{p1}$ and deposits a token in p1. The NAND gate 1 implements t1 as follows: when output $\overline{p1}$ = 1 and χ_1 occurs, i.e. χ_1 becomes 1, p1 is set to 1 by applying an instantaneous 0 from the output of the NAND gate 1 to the S input of the flip-flop and at the same time the output $\overline{p1}$ is reset, i.e. $\overline{p1}$ = 0. The same applies to t2 in a similar manner.



(a) (b) (c)

Fig. 3. a). A transition in SAPN. b).Implementation of the transition, b). Implementation of transitions t1 and t2 of Fig. 2.(c).

<u>*iii) The third and last step*</u> is about setting-up the initial marking by using an RC (Resistor and Capacitor) element. It is necessary for proper functioning to set-up the initial marking before operating the circuit. It is a common practice to use an RC element to establish the power on reset (POR) and at the same time to use a button connected parallel to the capacitor such that at any time desired by pressing the button we are able to set the system back to the initial marking. The time delay τ = R.C defines how long the setting-up time will be for the initial marking. How this process is accomplished, is shown in Fig. 4.(a). When the power is first applied to the circuit, a 0 is applied, for the period of τ time, to the S inputs of flip-flops, which represent places with initial marking 1, i.e. all places p1, p2, p3, ... are set to 1. At the same time, a 0 is also applied to the R inputs of flip-flops, which represent places with initial marking 0, i.e. all places p11, p12, p13, ... are reset to 0. After the power is being applied to the circuit, at any time it is also possible to set-up the circuit back to the initial marking by pressing the button B. An example SAPN is shown in Fig. 4.(b). The hardware implementation of the SAPN shown in Fig. 4.(b) is given in Fig. 4.(c). In this circuit places and transitions are implemented as described before. The initial marking, i.e. $M_0(p1, \overline{p1}, p2, \overline{p2}) = (1,0,0,1)^T$, is implemented by setting the first flip-flop and by resetting the second flip-flop.



(a) (b) (c)

Fig. 4. a). Setting-up the initial marking. b). An example SAPN. c). Setting-up the initial marking of the SAPN shown in Fig. 4.(b).

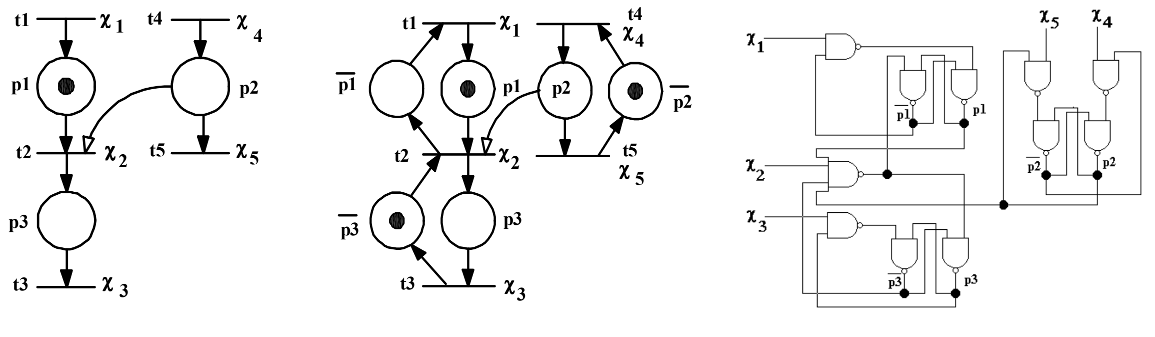
To show how our technique is applied to the hardware implementation of the SAPN, in this paper we consider the following SAPN structures:

- Enabling arc
- Inhibitor arc
- Actions

Please, note that for the sake of simplicity the implementation of the initial marking in the following SAPN structures are not shown.

3.1. Hardware Implementation of the Enabling Arc

The modelling power of Petri nets can be extended by adding the '*one testing*' ability, i.e., the ability to test whether a place has a token. This is achieved by introducing an *enabling arc*. The enabling arc connects an input place to a transition and is represented by a directed arrow, whose end is empty. The presence of an enabling arc connecting an input place to a transition means that the transition is only enabled if the input place has a token. The firing does not change the marking in the enabling arc connected places. In an SAPN, an enabling arc, *En(p_2,t_2)*, is shown in Fig. 5.(a). The transition t2 is fired if both p1 and p2 have one token each and firing condition χ_2 occurs. When t2 is fired, a token is removed from place p1 and a token is deposited into the output place p3, but the marking of enabling arc connected place p2 does not change. The transition t2 is not enabled to fire, if there is no token in place p2. Fig. 5.(b) shows the complement places of places p1, p2 and p3. The hardware implementation of the SAPN shown in Fig. 5.(b) is given in Fig. 5.(c). In this circuit places and transitions are implemented as described before.



(a) (b) (c)

Fig. 5. a). An enabling arc, En(p_2,t_2), in an SAPN. b). The complement places of places p1, p2 and p3. c). Hardware implementation of the SAPN shown in Fig. 5.(b).

3.2. Hardware Implementation of the Inhibitor Arc

The modelling power of Petri nets can be extended by adding the '*zero testing*' ability, i.e., the ability to test whether a place has no token. This is achieved by an *inhibitor arc*. The inhibitor arc connects an input place to a transition and is represented by an arc, whose end is a circle. The presence of an inhibitor arc connecting an input place to a transition means that the transition is enabled if the input place has no token. The firing does not change the marking in the inhibitor arc connected places. In an SAPN, an inhibitor arc, *In*(p_2,t_2), is shown in Fig. 6.(a). The transition t2 is fired if place p1 has a token and p2 has no token and firing condition χ_2 occurs. When t2 is fired, a token is removed from the input place p1 and a token is deposited into the output place p3, but the marking of inhibitor arc connected place p2 does not change. The transition t2 is not enabled to fire, if there is a token in place p2. Fig. 6.(b) shows the complement places of places p1, p2 and p3. Note that the inhibitor arc *In*(p_2,t_2), shown in Fig. 6.(a), can be replaced by the enabling arc *En*(p_2,t_2). The hardware implementation of the SAPN shown in Fig. 6.(b) is given in Fig. 6.(c). In this circuit, places and transitions are implemented as described before.

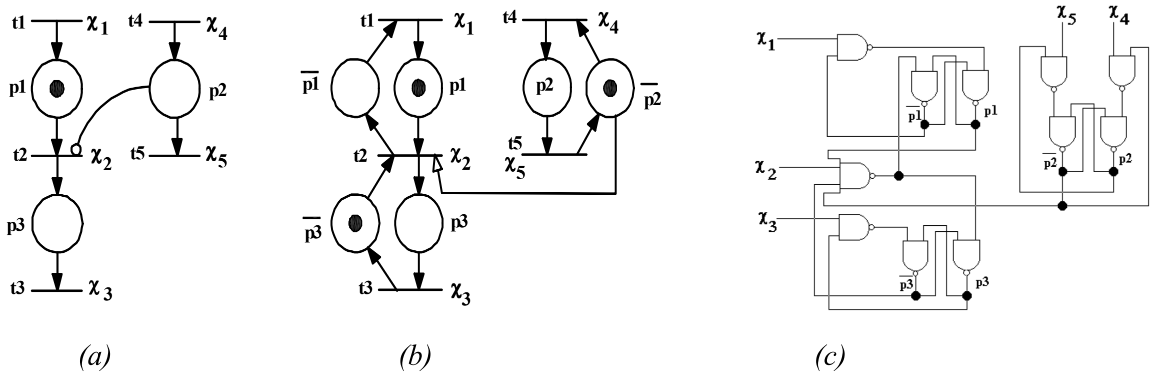


Fig. 6. a). An inhibitor arc, In(p_2,t_2), in an SAPN. b). The complement places of places p1, p2 and p3. c). Hardware implementation of the SAPN shown in Fig. 6.(b).

3.3. Hardware Implementation of Actions

In the SAPN, two types of actuations can be considered, namely impulse actions and level actions. Impulse actions are associated with transitions and they are enabled at the instant, when the related transition is being fired. Level actions are associated with places and they are enabled when there is a token in the related place. More than one action may be assigned to a transition or a place. Fig. 7.(a) shows an SAPN in which there is an impulse action assigned to t2, and there is a level action assigned to p3. The hardware implementation of the SAPN shown in Fig. 7.(a) is given in Fig. 7.(b). In this circuit, places and transitions are implemented as described before.

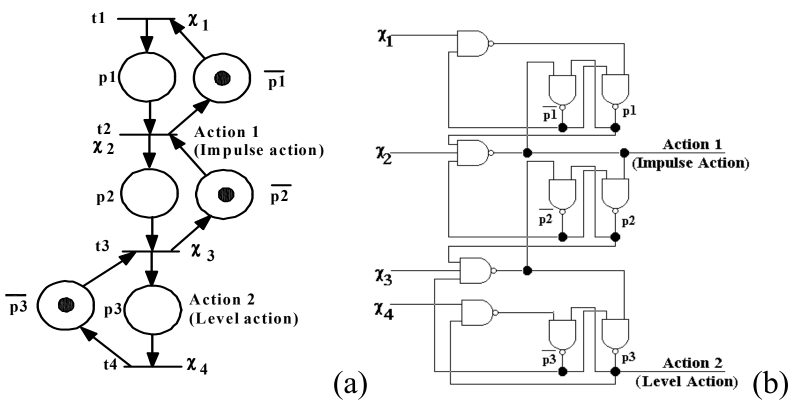


Fig. 7. a). Actions in an SAPN. b). Hardware implementation of the SAPN shown in Fig. 7.(a).

4. CONCLUSIONS

In this paper, a new method has been proposed for the digital hardware implementation of Petri net based specifications. The purpose of this paper has been to introduce a new discrete event control system paradigm, where the control system is modelled with extended Petri nets and implemented as an asynchronous controller using circuit elements. The results provided in this paper on the digital hardware implementation of Petri nets may be view as a better version of a previously introduced method [1], in terms of the implementation of transitions. Some Petri net structures, such as join, merge, fork, conflict, toggle, select, timed-transition have already been implemented by using our methodology, but due to the limited space they are not shown in this paper. Although the implementations considered in this paper are only for safe APNs, it is also possible to apply our method to general APNs and use up/down counters to represent places, instead of flip-flops. The results reported in this paper have already been applied to the control of an experimental manufacturing system. Our forthcoming publications will be about these results.

REFERENCES

1. V. Varshavsky & V. Marakhowsky, "Hardware Support for Discrete Event Coordination", *Proc. of Int. Workshop on Discrete Event Systems (WODES'96)*, Edinburg, UK, pp. 332-340, IEE, August 1996.
2. M. Uzam, *Petri-Net-Based Supervisory Control of Discrete Event Systems and Their Ladder Logic Diagram Implementations*. The University of Salford, UK, PhD. Thesis, 395 pages, 1998.
3. A.V. Yakovlev & A.B. Koelmans, "Petri Nets and Digital Hardware Design", *Lecture Notes in Computer Science, Lectures on Petri Nets II: Applications, Advances in Petri Nets, W. Reisig & G. Rozenberg (Eds.)*, Springer, pp. 154-236, 1998.
4. L.A. Hollaar, "Direct Implementation of Asynchronous Control Circuits", *IEEE Trans. On Computers, C-31(12)*, pp. 1133-1141, 1982.
5. V. Varshavsky, M. Kishinevsky, V. Marakhovsky, V. Peschansky, R. Rosenblum, A. Taubin, & B. Tzirlin, "Self-Timed Control of Concurrent Processes", *Kluwer Academic Publishers, Dordretch, The Netherlands*, 1990, V.I. Varshavsky, Ed.
6. V. Varshavsky & V. Marakhowsky, "Asynchronous Control Device Design by Net Model Behaviour Simulation", Lecture Notes in Computer Science, Vol. 1091: Proc. of the 17th Int. Conf. On Applications and Theory of Petri Nets, Osaka, pp. 497-515. Springer Verlag, 1996.