IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 17, 2020, accepted March 17, 2020, date of publication March 25, 2020, date of current version April 13, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2983276

FPGA Implementation of Fuzzy Interpreted Petri Net

ZBIGNIEW HAJDUK[1] **AND JOLANTA WOJTOWICZ**[2]
[1]Faculty of Electrical and Computer Engineering, Rzeszów University of Technology, 35-959 Rzeszow, Poland
[2]Computer Science Department, Polytechnical Institute, State Higher Vocational School Stanislaus Pigonia in Krosno, 38-400 Krosno, Poland

Corresponding author: Zbigniew Hajduk (zhajduk@kia.prz.edu.pl)

ABSTRACT The paper introduces new implementation methods of the fuzzy interpreted Petri net (FIPN) on FPGAs. The realization of FIPN is based on fast array multipliers and multipliers utilizing DSP blocks available on FPGA systems. The paper contains descriptions of particular network components' architectures and results of simulations of these components. In the paper a real control system is designed, which is used to show properties of FIPN. A few slightly different implementations of the example control system as well as their comparison in terms of FPGA resources requirement and calculations speed are also featured. The conducted experiments revealed that the proposed FPGA implementation is many times faster than software realizations of the same control system exercising typical microprocessors and microcontrollers.

INDEX TERMS Fuzzy interpreted Petri net, intelligent control systems modeling and analysis, FPGA implementation.

I. INTRODUCTION

High demands are placed on modern solutions for control systems modeling of practical industrial processes regarding their abilities to deal with complex problems while optimizing the performance of these approaches in terms of the use of hardware resources. The application of FPGA chips in these solutions allows the design and implementation of dedicated architecture, which can be adapted to the current needs of the modeled industrial process. FPGA systems are used with success in real-life models, which require implementation of neural networks [1], [2], fuzzy logic algorithms [3], [4] or Petri nets [5], [6].

Nowadays, two main directions of Petri nets development can be distinguished. The first of them is concerned with high-level Petri nets, which embrace higher-level concepts such as the use of complex structured data as tokens and employ algebraic expressions to annotate net elements. Various colored Petri nets with different extensions belong to this category [7]. The second direction focuses on low-level Petri nets, which main applications are centered around programming of industrial controllers or reconfigurable hardware. This group consists mainly of various types of interpreted Petri nets [8].

The associate editor coordinating the review of this manuscript and approving it for publication was Giovanni Pau.

An essential point in development of complex control systems is evaluation of their performance. For this task stochastic Petri nets can be applied. In [9] a Generalized Stochastic Petri net is used to evaluate performance of distributed systems. Generalized Stochastic Petri Net carries out performance evaluation through means of simulation and numerical methods. In [10] a software solution called GreatSPN is described, which implements GSPN algorithm. Mobius Framework is a programming solution, which implements stochastic Petri nets and allows the use of multiple modeling formalisms [11]. SIMTHES [12], is another tool supporting modular approach and multi-formalism modeling. SIMTHES provides means to apply product-form solution theory to multi-formalism compositional modeling techniques.

An interesting and efficient tool, combining Petri nets and fuzzy logic and allowing for effective analysis of modeled systems in terms of their performance and structure, is Fuzzy Interpreted Petri Net (FIPN) introduced in [13]. For this net, transitions have conditions assigned to them and their fulfillment is reliant on variables of the simulated system. A truth degree function defines degrees of fulfillment for conditions associated with transitions on the basis of signals' values. In many real engineering scenarios possible signals' values are within a specific interval. These signals can be regarded on the basis of fuzzy logic as events having degrees of truth in the interval [0, 1]. This interpretation is convenient with regards to signals originating from sensors, actuators and

other control system components. Thus FIPN makes it possible to use both analogue and binary signals for the control of processes. It also allows for modeling of the resources' quantitative changes to be modeled and convenient interpretation of fuzzy tokens' positions. The graphic representation of FIPN more accurately shows the net's dynamics in comparison with discrete PN representation thanks to the possibility of using analogue sensors. The aforementioned features are unique to the FIPN and are not encountered in other existing types of fuzzy Petri nets. Selected FIPN properties which can be analyzed exercising the FIPN's coverability graph have also been portrayed in [14].

It is of note that despite the fact that many different varieties of Petri nets were developed not many of them were implemented in hardware. A popular platform for Petri nets implementation constitutes programmable logic controllers (PLCs) [15]–[24]. Taking advantage of FPGAs, a significantly higher operation speed of Petri nets can be achieved, which is pertaining to the feasibility of a parallel execution of many operations. Other advantages of FPGA implementations of Petri nets in comparison with software solutions may include lower power consumption and the reliability. Some examples of FPGA implementations of binary and fuzzy Petri nets can be found in [5] and [25].

An idea of hardware implementation of the Fuzzy Interpreted Petri Net has also been outlined in [13]. However, the FIPN has actually not been implemented in FPGAs. Instead, programmable controllers have been exercised as the implementation platform [26], [27]. The proposed idea of hardware implementation of the FIPN, coming from [13], relies on the utilization of fuzzy RS flip-flops and fuzzy gates based on sum and limited multiplication operations. However, these components are costly to implement in FPGAs, have relatively high resources requirement and the overall calculations speed of the FIPN is rather low.

In this paper a new implementation method of the FIPN, suitable for FPGAs, is introduced. Contrary to [13], the method does not require implementation costly components, such as fuzzy flip-flops or fuzzy gates, and is highly oriented on the reduction of FPGAs resources requirement and maximization of calculations speed. The paper also presents a few developed variations of the implementation method and shows how slightly different approaches to the design influence on the FPGA resources requirement, maximum clock frequency, number of clock cycles and overall calculations time of the implemented digital circuit. Taking advantage of the developed practical example of a control system, a speed comparison between FPGA and software implementations of the FIPN is also presented in the paper.

II. FORMAL DESCRIPTION OF FUZZY INTERPRETED PETRI NETS

For the FIPN the transition $t \in T$ is enabled under the marking $M : P \rightarrow [0, 1]$ from the moment in which [13]:

$$\forall p \in^{\bullet} t, M(p) \geq \frac{W(p,t)}{K(p)} \tag{1}$$

and:

$$\forall p \in^{\bullet} t, M(p) \leq \frac{K(p) - W(t,p)}{K(p)} \tag{2}$$

to the moment in which:

$$\exists p' \in^{\bullet} t, M(p) = 0 \tag{3}$$

or:

$$\exists p' \in^{\bullet} t, M(p) = 1 \tag{4}$$

where $K(p)$ is the capacity of the p place, $W(p,t)$ and $W(t,p)$ are the arcs' weights connecting the p place with the t transition and vice versa, ${}^{\bullet}t = p \in P | (p,t) \in R$ is the set of input places for the t transition, $t^{\bullet} = p \in P | (t,p) \in R$ is the set of output places of the transition, p is a type I place (p') or type II place (p'').

When the transition is enabled and the degree of the transition's condition fulfillment $\Theta(t) = \vartheta \in [0, 1]$ increases by the $\Delta\vartheta$ value, the FIPN's new marking can be determined by the following formula:

$$M'(p) = \begin{cases} M(p) - \Delta\vartheta \dfrac{W(p,t)}{K(p)} \Leftrightarrow p \in^{\bullet} t \\ M(p) + \Delta\vartheta \dfrac{W(t,p)}{K(p)} \Leftrightarrow p \in t^{\bullet} \\ M(p) \Leftrightarrow p \notin^{\bullet} t \cup t^{\bullet} \end{cases} \tag{5}$$

III. FPGA IMPLEMENTATION OF FIPN

The proposed implementation method of the FIPN relies on the assignment of both type I and type II places. It also requires implementation of a transition, dedicated digital modules, which connected together in a specific way comply with the dynamic equation and the firing conditions of the FIPN. The method directly applies equations (1),(2),(3), (4), (5) and does not require the usage of any fuzzy flip-flops, e.g., [25], which architectures are rather complicated and their FPGA implementation consumes significant amount of logic resources. The important goal of the design was to ensure a high speed of the FIPN calculations and low logic resources requirement.

According to the definition, all variables' values belong to the continuous interval [0, 1]. The proposed digital realization of the FIPN is based on the fixed point arithmetic, which implementation is simpler, requires reasonably less logic resources and ensures higher throughput than the floating point one. Therefore, the continuous interval [0, 1] must be mapped to the discrete set $0, 1, \ldots, 2^Q - 1$, where Q stands for the required number of bits.

In the following subsections we shall describe architectures of place and transition modules and a manner in which the modules should be connected to realize a general fragment of FIPN.

A. TYPE I PLACE MODULE

For FIPN type I places, the arcs' weights must be equal to 1, $W(p,t) = W(t,p) = 1$ and the places' capacities have to

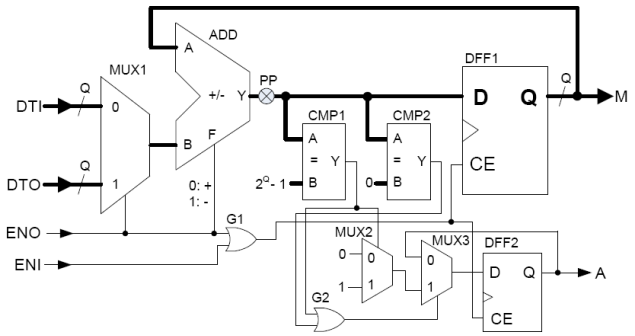

FIGURE 1. The architecture of the type I place module.

be equal to 1, $K(p) = 1$, as well. Thus, the dynamic equation (5), can be simplified to the following form:

$$M'(p) = \begin{cases} M(p) - \Delta\vartheta \Leftrightarrow p \in^{\bullet} t \\ M(p) + \Delta\vartheta \Leftrightarrow p \in t^{\bullet} \\ M(p) \Leftrightarrow p \notin^{\bullet} t \cup t^{\bullet} \end{cases} \tag{6}$$

The digital module, which represents behavior of FIPN type I place, must obey equation 5. The detailed architecture of the type I place module is portrayed in Fig. 1. The bold lines in the figure represent multi-bit buses. The module consists of the two-input, K-bit multiplexer (MUX1), K-bit adder/subtractor (ADD), set of Q flip-flops type D (DFF1), two comparators (CMP1, CMP2), two binary multiplexers (MUX2, MUX3), D flip-flop (DFF2) and two OR gates (G1, G2). The ADD block performs the addition ($F = 0$) or subtraction ($F = 1$) operation, depending on the value on its F input. Both comparators determine if the value on their A input is equal to the value on the B input (if so, the Y output is set to the high state). Taking into consideration the constant values on the second comparators' input, the CMP1 and CMP2 blocks inform whether the result of the addition/subtraction operation reaches the two extreme values: $2^Q - 1$ (which constitutes the discrete counterpart of the 1.0 constant) and 0, respectively. The module has two Q-bit inputs (DTI, DTO) corresponding to the increase of the degree of the input and output transition's condition fulfillment ($\Delta\vartheta$), Q-bit output (M) storing the output marking of the place, two module's activation inputs (ENI, ENO) and activation output (A). The PP point in the schematic from Fig. 1 indicates the location where the additional pipe-line registers can be inserted. The set of D flip-flops similar to the DFF1 but without the clock enable (CE) input can be used as the pipe-line registers. The pipe-line registers allow higher clock frequency to be attained (however, at the expense of the additional clock cycle which simultaneously increases the overall calculations time).

The functioning of the type I place module can be described by the following formulas:

$$M[n+i] = \begin{cases} M[n] + DTI \Longleftrightarrow ENI = 1 \\ M[n] - DTO \Longleftrightarrow ENO = 1 \\ M[n], \quad otherwise \end{cases} \tag{7}$$

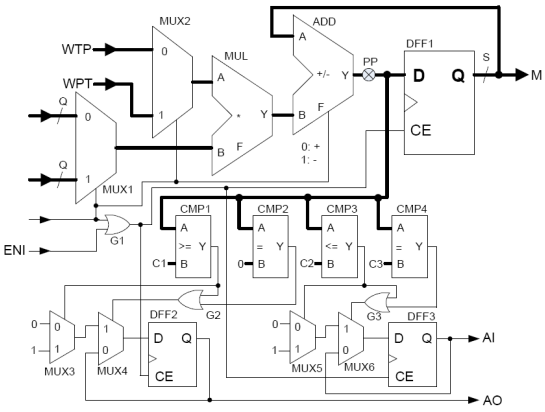

FIGURE 2. The architecture of the type II place module.

$$A[n+i] = \begin{cases} 1 \Longleftrightarrow M[n] = 2^Q - 1 \\ 0 \Longleftrightarrow M[n] = 0 \\ A[n], \quad otherwise \end{cases} \tag{8}$$

where n denotes the actual clock cycle, $i = 1$ or $i = 2$ for the module's version without or with the pipe-line registers, respectively. As it can be seen later, the activation output A facilitates the test of the transition's firing condition for the realization of a FIPN. The A output goes high from the moment in which all bits of the M output are high (the marking of the place achieves the maximum value), to the moment in which all bits of the M output are low (the marking of the place reaches the minimum value).

B. TYPE II PLACE MODULE

Since the FIPN's place type II may have the capacity value greater than one, as well as the arcs' weights may also have different values than one ($W(p,t) \geq 1, W(t,p) \geq 1, K(p) \geq 1$), the architecture of the module implementing the place type II is slightly more complicated (Fig. 2). In comparison with the module type II it also contains additional two-input, Q-bit multiplexer (MUX2), multiplier (MUL), and more complex part of the circuit responsible for the determination of the two activation outputs (AI and AO) values.

The important part of the circuit from Fig. 2 is the Q-bit multiplier block. Two options for the implementation of the MUL block, namely the usage of the FPGA embedded multipliers as well as the application of the synchronous version of the fast array multiplier, briefly described in [28], have been considered. The fast array multiplier requires a considerable amount of FPGA resources and needs a few pipe-line stages (a few clock cycles to complete calculations) in order to achieve a reasonable clock frequency. But it does not use the FPGA embedded multipliers, which may be an advantage in some cases (e.g., when the embedded multipliers are not available). On account of the multiplication operation, the number of bits of the module's output (S) is higher than the number of bits of the inputs (Q). The S value must be properly chosen by the designer, taking into account the values of the arcs' weights (WPT, WTP).

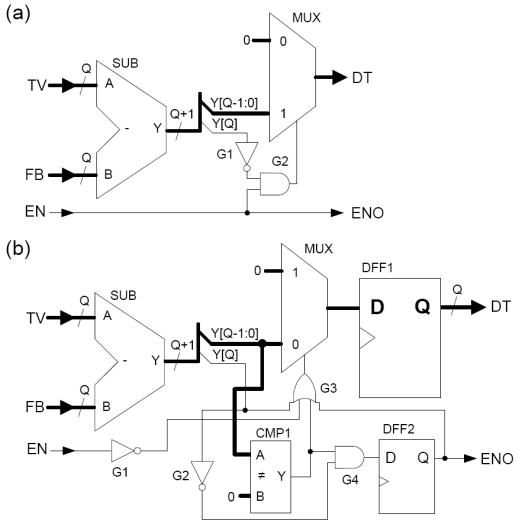

FIGURE 3. The architecture of the transition module, basic version (a), version for place modules with pipe-line registers (b).

The functioning of the type II place module can be described as follows:

$$M[n+i] = \begin{cases} M[n] + DTI \cdot WTP \Longleftrightarrow ENI = 1 \\ M[n] - DTO \cdot WTP \Longleftrightarrow ENO = 1 \\ M[n], \quad otherwise \end{cases} \tag{9}$$

$$A[n+i] = \begin{cases} 1 \Longleftrightarrow M[n] \geq C_1 \\ 0 \Longleftrightarrow M[n] = 0 \\ A[n], \quad otherwise \end{cases} \tag{10}$$

$$AO[n+i] = \begin{cases} 1 \Longleftrightarrow M[n] \leq C_2 \\ 0 \Longleftrightarrow M[n] = C_3 \\ AO[n], \quad otherwise \end{cases} \tag{11}$$

where $i = 1$ or $i = 2$ are for the module's version without or with the pipe-line registers respectively, C_1, C_2 and C_3 are constants so that $C_1 = WPT * (2^Q - 1)$, $C_2 = (K - WTP) * (2^Q - 1)$, $C_3 = K * (2^Q - 1)$, $WTP = W(t, p)$, $WPT = W(p, t)$ are arcs' weights, and $K = K(p)$ denotes the capacity of the place. It is of note that the constants C_1, C_2 and C_3 from equations (E), (F) directly correspond to the values delivered to the second inputs of the comparators CPM1, CMP3 and CMP4 from Fig. 2. It is also important to note that the M module's output value does not represent the direct marking of a FIPN's type II place, but the marking multiplied by the value of the capacity of the place. This considerably simplifies the hardware implementation of the place (no division operation is needed).

C. TRANSITION MODULE

The transition module is responsible for the calculation of the increase of the degree of a transition's condition fulfillment. Its architecture is presented in Fig. 3.

The basic version of the transition module (Fig. 3a) is targeted for the place modules, which do not contain the pipe-line registers. This version of the module is a purely combinational circuit consisting of the Q-bit subtractor block (SUB), multiplexer (MUX) with two Q-bit inputs, and two logic gates. The notation Y[Q] from Fig. 2 stands for the bit selection with the Q number of the Y bus/variable, whereas the Y[Q−1:0] denotes a part selection of the Y bus comprised of the bits from 0 to Q−1. The output of the SUB block counts Q+1 bits. If the subtraction operation yields a negative value then the most significant bit of the SUB block output (Y[Q]) is set to 1, and on the account of the MUX block the value existing on the DT output of the transition module is 0. The same idea also applies to the second version of the transition module (Fig. 3b). The main difference between the two version of the module is that the output of the second version is registered by the set of D flip-flops (DFF1) and the specific output value on the output is held only by a single clock cycle. Additionally, the ENO output generates single pulse (lasting one clock cycle) exactly one clock cycle later after the moment in which the output DT is set. For the basic version of the transition module the output ENO simply follows the input EN. It is of note that the circuit from Fig. 3b can only return to its initial state when the values on the inputs TV and FB are the same.

The functioning of the transition module from Fig. 3b can be described by the following equations:

$$DT[n+i] = \begin{cases} TV - FB \Longleftrightarrow TV \geq FB \;\&\; EN = 1 \;\&\; i = 1 \\ 0 \quad , otherwise \end{cases} \tag{12}$$

$$ENO[n+i] = \begin{cases} 1 \Longleftrightarrow DT[n] \neq 0 FB \;\&\; EN = 1 \;\&\; i = 1 \\ 0, \quad otherwise \end{cases} \tag{13}$$

where n denotes the actual clock cycle and $(i = 0, 1, 2, \ldots)$.

Since the basic version of the transition module is a combinational circuit, the formulas simplify to the following form:

$$DT = \begin{cases} TV - FB \Longleftrightarrow TV \geq FB \;\&\; EN = 1 \\ 0, \quad otherwise \end{cases} \tag{14}$$

$$ENO = EN \tag{15}$$

D. IMPLEMENTATION OF GENERAL FIPN FRAGMENT

Fig. 4 shows the general fragment of the FIPN and its implementation exercising the previously described places and transition modules. The FIPN's fragment consists of two input places p'_1, p''_1 (type I and type II) and two output places p'_2, p''_2 (type I and type II as well) for the particular transition t_3. Each place and each transition has its own hardware counterpart as portrayed in Fig. 4b.

Taking into account equations (8), (10), (11) and the conditions given by the definition of the FINP it can be directly inferred that the transition is enabled if the A activation output of all place modules type I representing the input places is high, and the A activation output of all type I place modules representing the output places is low, and the AI activation output of all type II place modules representing the input places as well the AO output of all type II place modules representing the output places are high. Thus, the output of the G1 AND gate from Fig. 4b determines the enabling

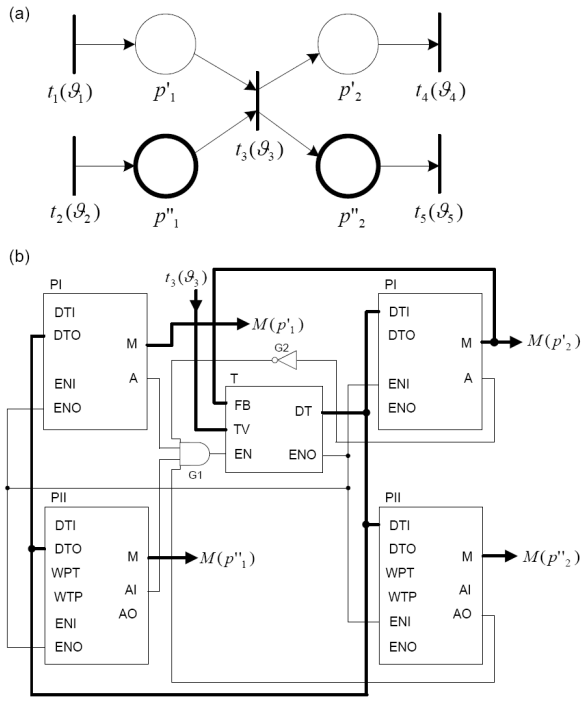

FIGURE 4. General FIPN fragment (a) and its implementation (b).

conditions of the transition. It is of note that the FB input of the transition module T is connected to the marking output of the type I place module, representing the selected output place of the transition. Taking into consideration the connections from Fig. 4b as well as equations (7) and (14) and assuming that $TV \geq FB$ for the transition module, we can determine the marking of the p'_2 place:

$$\begin{aligned} M'_2[n+1] &= M'_2[n] + TV - FB \\ &= M'_2[n] + TV - M'_2[n] = TV \end{aligned} \tag{16}$$

Therefore, the difference value between the TV and FB inputs of the transition module (which also constitutes the DT output value of the transition module) is equal to the positive increase of the degree of the transition's condition fulfillment:

$$TV - FB = \vartheta_3 - M'_2[n] = \Delta\vartheta_3 \tag{17}$$

Considering this fact and taking into account equations (7) and (9) it is easy to notice that the circuit from Fig. 4b complies with the dynamic equations of the FIPN.

IV. MODIFIED FIPN IMPLEMENTATION METHOD

The modification of the FIPN implementation method involves some changes in the architectures of type I place modules and transition modules. It allows the number of required FPGA resources to be significantly reduced.

As equation (13) indicates, if the degree of the transition's condition fulfillment is higher than the marking value of an output place of the transition, then the new marking of the place is equal to the value of the degree of the transition's condition fulfillment. Analogously, considering equations (5)

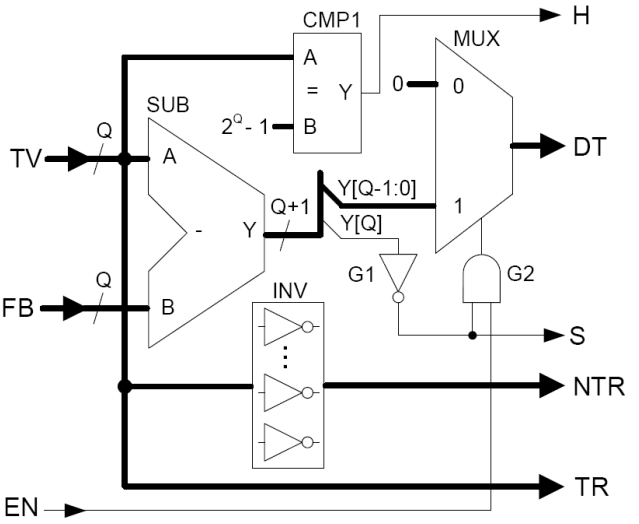

FIGURE 5. Modified transition module.

and (17) we can determine the marking of the p'_1 input place of the transition from Fig. 4a:

$$M'_1[n+1] = M'_1[n] - \Delta\vartheta_3 = M'_1[n] - TV + M'_2[n] \tag{18}$$

However, observing equation (5) for the type I places, it is easy to note that $M(p'_2) = 1 - M(p'_1)$ for the discrete version $M^1_2[n] = (2^Q - 1) - M'_1[n]$, where Q denotes the number of implementation bits and n is the actual clock cycle. Therefore, equation (18) reduces to:

$$\begin{aligned} M'_1[n+1] &= M'_1[n] - TV + (2^Q - 1) - M'_1[n] \\ &= (2^Q - 1) - TV \end{aligned} \tag{19}$$

Equations (16) and (19) are valid for all of the output and input places of a particular transition, respectively. It is also of note that the digital realization of equation (19) is equivalent to the inversion of all bits of the TV bus. Thus, the function of the modified modules representing type I places may only be the storage of the particular value (the degree of the transition's condition fulfillment or its inversion) and the other necessary operations can be performed by the modified transition modules.

The architecture of the modified transition module is shown in Fig. 5. The DT output performs the same function as for the module from Fig. 3a. Apart from the aforementioned output the modified transition module also makes available two other Q-bit outputs, namely the TR output simply following the input TV (it can be buffered which is not shown in Fig. 5) and its inversion NTR, and two binary outputs. The H output is set high when all bits of the TV input are in the high state. The high state on the S output indicates, in turn, that $TV \geq FB$.

The modified architecture of the place module is portrayed in Fig. 6. Apart from the set of Q flip-flops (DFF1), which stores the TR or NTR input values, depending on the values delivered to the ENO, SO, ENI and SI inputs, it integrates the DFF2 flip-flops controlling the A activation output. The A

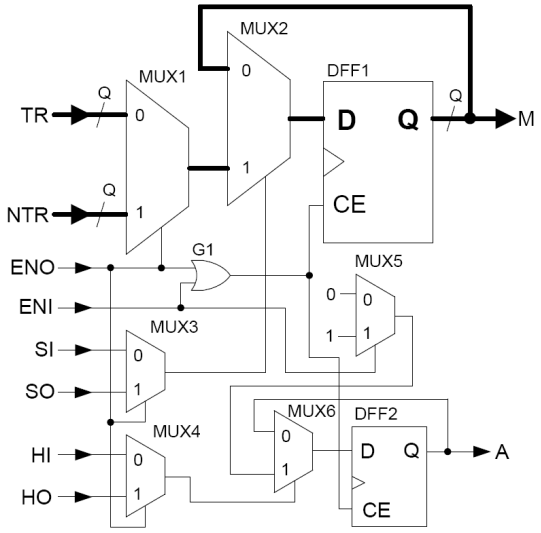

FIGURE 6. Architecture of the modified place module.

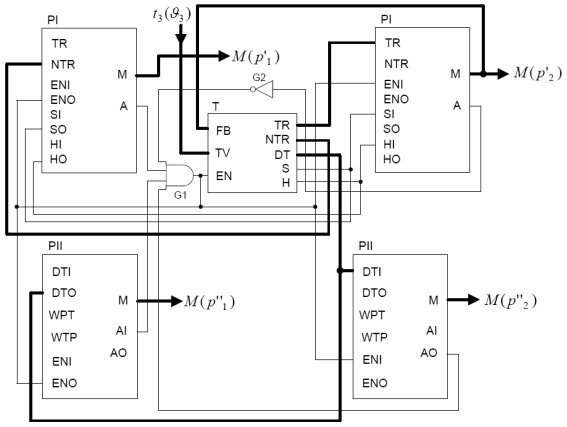

FIGURE 7. Implementation of the general FIPN's fragment with the usage of modified modules.

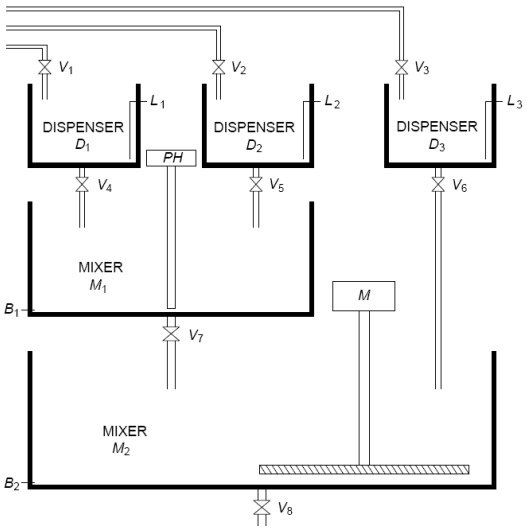

FIGURE 8. Scheme of the control system.

output is set depending on the HI, HO inputs which should be connected to the particular outputs of the modified transition module. The specific connections between the modified type I place modules (PI) and modified transition module (T), which implements the general fragment of the FIPN from Fig. 4a, is shown in Fig. 7. The type II place modules (PII) in Fig. 7 have the architecture shown in Fig. 2 (since the type II places can have different capacity and the arcs can have different weights as well, it means that the architecture of type II places cannot be modified in the similar manner as the architecture of type I places). Carrying out similar considerations as in the previous paragraph, it can be proven that the connections from Fig. 7 comply with the transition enabling conditions as well as dynamic equations of the FIPN.

V. DESCRIPTION OF THE IMPLEMENTED CONTROL SYSTEM

To demonstrate the properties of a fuzzy interpreted Petri net a liquid mixing control system is considered. The operation of the system allowing the mixing of three liquid components may proceed in a variety of ways. Assuming sensor and actuator labeling in accordance with Fig. 8, an exemplary algorithm for mixing system operation can be realized as follows. Three tanks, denoted as (D_i, $i = 1, 2, 3$) and located in the upper part of the diagram act as dispensers, whose task is to measure appropriate portions of liquid components. Analog sensors are used for determining the filling levels of tanks. The measurements from these sensors are denoted as L_i and normalized to the interval of values [0, 1]. The dispensers are filled independently of each other by opening V_i valves as soon as they are emptied ($L_i = 0$). When the dispensers D_1 and D_2 are full and the mixer M_1 is empty, the filling process starts using two portions of each of the chemical components. The V_4 and V_5 dosing pumps are always switched on simultaneously, starting the process of transferring further portions of ingredients. When two portions of each of the two chemical components are already in the M_1 mixer, the T_{MIN} time element is started, counting down the minimum T_1 time required for the cessation of chemical reactions initiated by combining of these components. In the next step the reading from the PH-meter is carried out and on its basis the number of solvent portions to be fed from the D_3 dispenser is determined.

The dilution process of the chemical obtained in the mixer M_1 is carried out in the M_2 mixer. Initially, one portion of the solvent is poured into this mixer, and then simultaneously the chemical from M_1 mixer is poured into the M_2 mixer together with a certain number of additional portions of solvent, at the same moment switching on the M stirrer. When the mixer M_1 is empty, as indicated by the binary level sensor B_1, the dosing pump V_7 is switched off and the refueling of the mixer starts at the same time. Through the simultaneous realization of these actions the device's performance is increased. When the entire chemical substance from the M_1 mixer and the appropriate number of solvent portions are transferred to the M_2 mixer, then the T_M time element is activated, responsible

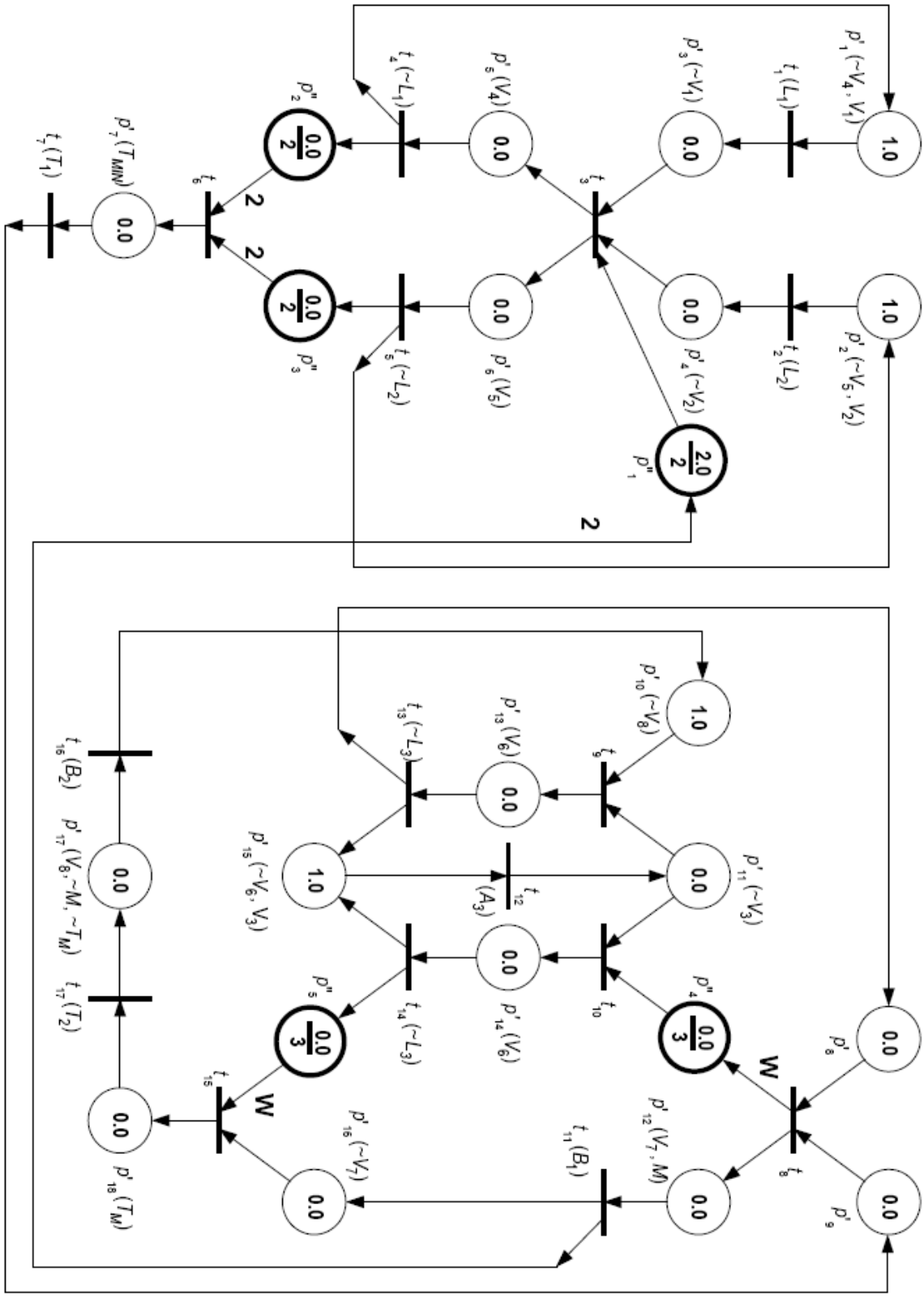

FIGURE 9. Petri net describing the control algorithm of the mixer system from Fig. 8.

for measuring the time for which these chemicals are to be additionally mixed. After this period of time the solution is ready, therefore the stirrer M is turned off and the mixer M_2 is emptied by switching on the pump V_8. At the end of

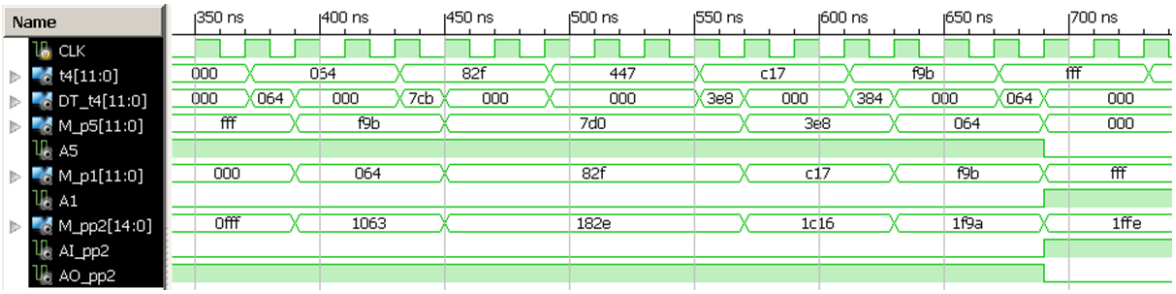

FIGURE 10. Simulation waveforms for the basic version of the HIFPN from Fig. 9 implementation.

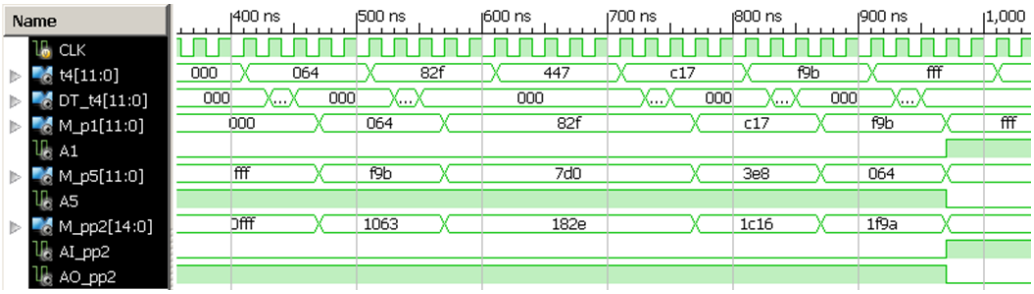

FIGURE 11. Simulation waveforms for the pipelined implementation version.

this operation, the V_8 pump is turned off and one portion of solvent from dispenser D_3 is poured into the M_2 mixer so that another emptying process of the M_1 mixer can start as soon as possible. The algorithm describing the operation of the mixer is shown in Fig. 9 in the form of a fuzzy interpreted Petri net.

The p' type network places had control signals assigned, which were marked as follows:

V_n - opening of the valve or activation of the dosing pump V_n, $n = 1, 2, 3, 4, 5, 6, 7, 8$ ($\sim V_n$ - closing of the valve or switching off the dosing pump)

T_{MIN} - starting the T_{MIN} timer counting down the T_1 time for which the solution remains in the M_1 mixer, ($\sim T_{MIN}$ - turning off the T_{MIN} timer)

T_M - starting the T_M timer counting down the T_2 time of mixing the solution in the M_2 mixer ($\sim T_M$ - turning off the T_M timer)

M - starting of the mixer, ($\sim M$) - turning off the mixer

PH test and determination of W value - weight calculation $W(t_8, p''_4) = W(p''_5, t_{15})$ on the basis of PH-meter indication. In practical realization these indications were replaced with signals from the $SM374$ module.

Transitions t have been assigned signals from sensors and time elements, assuming the following designations:

L_i - filling of the dispenser D_i, $i = 1, 2, 3$ - analog signal ($\sim L_n = 1 - L_n$)

B_k - emptying of the mixer M_k, $k = 1, 2$ - binary signal

T_1 - expiration of the minimum period of time for the solution to remain in the mixer M_1

T_2 - expiration of the time for mixing of the solution in the M_2 mixer

VI. SIMULATIONS AND IMPLEMENTATION RESULTS

The FIPN from Fig. 9 has been described in Verilog Hardware Description Language using methods presented in Section III for the number of bits amounting to 12 ($Q = 12$). Figs. 10,11,12 show the simulation waveforms of the FIPN involving the p'_1 p'_5 and p'_2 places, and realized with three slightly different methods, namely the basic version introduced in Section III (Fig. 10), the basic version with the pipeline registers (Fig. 11) and the version with the fast array multiplier (Fig. 12). The simulation waveforms for the modified implementation method, described in Section IV, are exactly the same as for the basic version. Thus, they have not been presented here.

Since the simulation results from Figs. 10,11,12 only vary in the number of clock cycles needed to obtain the current FIPN's marking (1, 3, and 7 clock cycles are required for the waveforms from Fig. 10,11 and 12, respectively), only the waveform from Fig. 10 will be discussed in detail. The simulation from Fig. 10 starts at the simulation time of $350ns$ where the $t4$ transition value is equal to 0, no fuzzy marker is present in the $P1$ type I place ($M\_p1 = 000h$), the full fuzzy marker is present in the $P5$ type I place ($M\_p5 = FFFh$), and 1 of 2 fuzzy markers exists in the $P2$ type II place ($M\_pp2 = 0FFFh$). The activation outputs for the $P1$ and $P5$ type I places, as well as the activation outputs AI and AO for the $P2$ type II place are reset and set, respectively ($A1 = 0, A5 = 1$ as well as $AI\_pp2 = 0, AO\_pp2 = 1$). The state of the activation outputs indicates that the transition $t4$ is enabled. At the simulation time of $370ns$ the degree of the $t4$ transition's condition fulfillment changes from $000h$

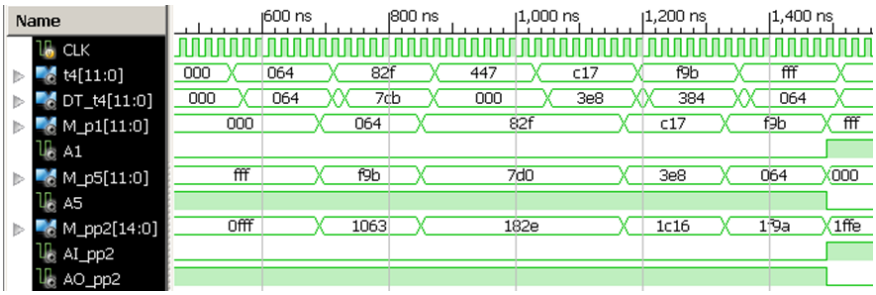

FIGURE 12. Simulation waveforms for the version with the fast array multiplier.

TABLE 1. FPGA resources utilization, maximum clock frequency, number of clock cycles and FIPN's update time for the Xilinx Zynq FPGA.

Version	LUTs	Slices	FFs	DSPs	F_{MAX} [MHz]	Number of clock cycles	FIPN's update time [ns]
A	1225 (2.3%)	458 (3.4%)	331 (0.3%)	5 (2.3%)	91.7	1	10.9
B	1094 (2.1%)	520 (3.9%)	835 (0.8%)	5 (2.3%)	149.2	3	20.1
C	688 (1.3%)	316 (2.4%)	303 (0.3%)	5 (2.3%)	97.1	1	10.3
D	1282 (2.4%)	504 (3.8%)	1062 (1.0%)	0	322.5	7	21.7

to $064h$, which immediately causes the change of the DT_t4 output of the $t4$ transition module ($DT\_t4 = 064h$). The DT_t4 output represents the increase of the degree of the $t4$ transition's condition fulfillment. For the waveforms from Fig. 10, the change of the DT_t4 output is immediate since this output is a combinational function of the transition value and the M_p1 value (for the simulations from Fig.11 and Fig. 12, the value of the DT_t4 output is not immediate - it is delayed by a single clock cycle). During the nearest rising edge of the clock signal, after the change of the $t4$ value, the outputs of all place modules are updated. The marking of output places for the $t4$ transition is increased by the value existing on the DT output, whereas the marking of the input place for the $t4$ transition is decreased by the same value. Thus, $M\_p1 = 064h$, $M\_p5 = f9Bh$ and $M\_pp2 = 1063h$. It is of note that the DT_t4 output value changes back to the 0 value immediately after the update of the marking of the place modules for the $t4$ transition.

An analogous change of the outputs values takes place at the simulation time of $430ns$, when the new value of $82Fh$ emerges on the $t4$ transition (it causes the following changes: $DT\_t4 = 7CBh$ and then $M\_p1 = 82Fh$, $M\_p5 = 7D0h$, $M\_pp2 = 182Eh$). An important event happens at the simulation time of $490ns$, when the $t4$ transition value drops from $82Fh$ to $447h$. This does not cause the change of the DT_t4 output (there is no increase of the degree of the $t4$ transition's condition fulfillment) and subsequent changes of the marking for the $t4$ transition's places. This is an important feature of the FIPN enabling the suppression of disturbances, which may be present on the transitions' inputs. When finally the degree of the $t4$ transition's condition is fully satisfied ($t4 = FFFh$), which takes place at simulation time of $670ns$, no fuzzy marker is left in the $P5$ type I place ($M\_p5 = 000h$), the full fuzzy marker emerges in the $P1$ type I place ($M\_p1 = FFFh$) and two fuzzy markers are present in the $P2$ type II place ($M\_pp2 = 1FFEh$). Then, the activation outputs of the place modules are updated as well ($A1 = 1$, $A5 = 0$, $AI\_pp2 = 1$, $AO\_pp2 = 0$). It is of note that the described behavior of the FIPN's fragment, expressed by the simulation waveforms from Figs. 10, 11, 12 satisfies the FIPN's dynamic equations 5 as well as the transition's enabling conditions 1, 2, 3,4.

Apart from simulations, the FIPN from Fig. 9 has also been implemented using the Xilinx Zynq FPGA (the XC7Z020 from the Zedboard has been exercised). The implementation results, including the number of utilized look-up tables (LUTs), FPGA's slices, flip-flops (FFs) and embedded digital signal processing (DSPs) blocks, as well as the maximum allowable clock frequency (FMAX), number of clock cycles required for the update of a new FIPN's marking and overall FIPN's update time, are presented in Table 1. The overall FIPN's update time is given under the assumption that the FIPN's implementation is clocked with the maximum allowable clock frequency. The resources utilization in percentages, related to the accessible number of resources for the chosen FPGA chip, has been given in the parenthesis. Four implementation versions have been considered in Table 1, namely: the basic version described in Section III (A), basic version with pipe-line registers (B), modified implementation version described in Section IV (C), and version with the fast array multiplier (D). Similar to the simulations, the number of bits has been set to 12 (Q=12). As Table 1 indicates, the least FPGA logic resources (in terms of the number of LUTs and FFs) is needed for the C version, whereas the most logic

TABLE 2. Calculations times of the FIPN from Fig. 9 implemented in software and speed factors.

Platform and processor type	Calculations time $[\mu s]$	Speed factor
PC computer Intel Atom x5-Z8500 @ 1.4GHz (64-bit)	0.122	11.9
Raspberry Pi 2 ARM Cortex-A7 @ 900MHz (32-bit)	1.191	115.6
Xilinx Zynq (Processing System part) ARM Cortex-A9 @ 667MHz (32-bit)	1.043	101.3
Arduino MEGA Atmel AVR ATMEGA2560 @ 16MHz (8-bit)	38.136	3702.5

resources is required by the D version. However, contrary to the all other versions, the D version does not exercise the FPGA embedded multipliers. The D version also allows remarkable higher clock frequency. Yet, it is achieved at the expense of a significantly higher number of clock cycles needed to update the FIPN's marking. Taking into consideration the maximum allowable clock frequency and the required number of clock cycles, the shortest FIPN's update time is provided by the A and C versions. Although the versions with pipe-line registers (the B and D versions) enable higher clock frequencies, they are characterized by almost twice longer FIPN's update time.

For the speed comparison purposes of the proposed FPGA implementation methods, the FIPN from Fig. 9 has also been realized in software (the C language has been used) exercising a few different hardware platforms. The FIPN's calculations times for the PC computer, Raspberry Pi 2 module, processing system part of the Xilinx Zynq (the bare metal system implementation has been used) and Arduino MEGA board, are presented in Table 2. The calculations times for all of the platforms, with the exception of the Intel processor, have been measured by the external universal counter. For the PC platform, the calculations time has been obtained by reading the system time before and after the calculations. The speed factor from Table 2 informs how many times the FPGA implementation of the FIPN (the C implementation version from Table 1 has been taken as the reference) is faster than the subsequent software implementations. The FIPN from Fig. 9 has also been implemented exercising previously developed FPGA-embedded multiprocessor programmable controller [29] and the CPDev software environment (the ST language has been used). Only single processor-core of the controller has been applied. The attained calculations time amounted to 40.92μs. Since it has been shown in [29] that the controller is faster than typical PLCs, the calculations time of the FIPN software implementation using industrial PLCs may be essentially longer.

It is of note that the software calculations time of the FIPN strongly depends on the FIPN's size (the number of places and transitions). The higher the FIPN's size, the longer the FIPN's software calculations time. This is a quite contrary to the proposed FIPN's hardware implementations, where the FIPN's size has no impact on the FIPN's update time.

However it must be pointed out that the update time of the FIPN hardware implementation may be affected by the number of implementation's bits (the Q parameter). The higher number of implementation's bits entails an increase of the update time. Nevertheless, this increase is rather small. The number of implementation's bits has also strong influence on the FPGA resources requirement - more resources are needed for higher number of implementation's bits.

VII. CONCLUSIONS

It has been shown that the FIPN can be easily implemented in FPGAs without the need of the expensive fuzzy hardware components, such as fuzzy RS flip-flops and fuzzy gates. Contrary to [13] the direct implementation methods, with a few different versions, have been proposed. The developed versions slightly differ in the FPGA resources requirement and FIPN's calculations time. It is worth to note that the application of pipe-line registers, which is a common technique allowing the increase of the maximum clock frequency in synchronous systems, does not always lead to the decrease of the overall calculations time (more clock cycles are needed for a pipe-lined implementation which may not be compensated by a higher maximum clock frequency).

As far as the speed comparison between FPGA and software implementations of FIPNs is concerned, it is no surprise that the FPGA implementations are remarkably faster than their software counterparts (from 11 times faster for the 64-bit PC platform up to 3702 times faster for 8-bit Arduino platform). Therefore, FPGA implementations of FIPNs are particularly well suited for, so called, fast control plants, e.g., mechatronic plants, and may also find applications in such areas as space and defense systems.

The FPGA implementation of the FIPN can also be used as the hardware function block for the previously developed multiprocessor programmable controller [29], greatly contributing to the shortening of the execution time of specific control algorithms.

A kind of a disadvantage of the proposed implementation method of the FIPN is the need of manual alteration of HDL code and reimplementation of the whole FPGA project - in case of the change of the FIPN structure. Therefore, future work may include the development of a software tool which would be able to automatically generate the complete HDL code for the particular structure of the FIPN. It would also be interesting to modify the proposed implementation method in the way similar to [2], where the actual structure of the neural network is stored in FPGA embedded Block RAM memory, and the alteration of the net's structure requires only the change of the memory content.

REFERENCES

[1] X. Ju, B. Fang, R. Yan, X. Xu, and H. Tang, "An FPGA implementation of deep spiking neural networks for low-power and fast classification," *Neural Comput.*, vol. 32, no. 1, pp. 182–204, Jan. 2020.
[2] Z. Hajduk, "Reconfigurable FPGA implementation of neural networks," *Neurocomputing*, vol. 308, pp. 227–234, Sep. 2018.
[3] G. T. Tchendjou, E. Simeu, and R. Alhakim, "Fuzzy logic based objective image quality assessment with FPGA implementation," *J. Syst. Archit.*, vol. 82, pp. 24–36, Jan. 2018.
[4] C. Chang, T. Jiang, Z. Zhou, and Y. Yuan, "Field programmable gate array implementation of a single-input fuzzy proportional–integral–derivative controller for DC–DC buck converters," *IET Power Electron.*, vol. 9, no. 6, pp. 1259–1266, May 2016.
[5] A. Bukowiec and M. Adamski, "Transition based synthesis with code markers of Petri nets into FPGAs," *IFAC Proc. Volumes*, vol. 46, no. 28, pp. 181–186, 2013.
[6] C.-K. Chen, "A Petri net design of FPGA-based controller for a class of nuclear I&C systems," *Nucl. Eng. Design*, vol. 241, no. 7, pp. 2597–2603, Jul. 2011.
[7] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Berlin, Germany: Springer, 1997, doi: 10.1007/978-3-642-60794-3.
[8] R. David and H. Alla, *Discrete, Continuous, and Hybrid Petri Nets*, 2nd ed. Berlin, Germany: Springer, 2010, doi: 10.1007/978-3-642-10669-9.
[9] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, "Modelling with generalized stochastic Petri nets," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 26, no. 2, p. 2, Aug. 1998, doi: 10.1145/288197.581193.
[10] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo, "Greatspn 1.7: Graphical editor and analyzer for timed and stochastic Petri nets," *Perform. Eval.*, vol. 24, pp. 47–68, 1995, doi: 10.1016/0166-5316(95)00008-L.
[11] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, "The mobius framework and its implementation," *IEEE Trans. Softw. Eng.*, vol. 28, no. 10, pp. 956–969, Oct. 2002, doi: 10.1109/TSE.2002.1041052.
[12] E. Barbierato, G.-L. Dei Rossi, M. Gribaudo, M. Iacono, and A. Marin, "Exploiting product forms solution techniques in multiformalism modeling," *Electron. Notes Theor. Comput. Sci.*, vol. 296, pp. 61–77, Aug. 2013, doi: 10.1016/j.entcs.2013.07.005.
[13] L. Gniewek, "Sequential control algorithm in the form of fuzzy interpreted Petri net," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 43, no. 2, pp. 451–459, Mar. 2013, doi: 10.1109/TSMCA.2012.2202107.
[14] L. Gniewek, "Coverability graph of fuzzy interpreted Petri net," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 44, no. 9, pp. 1272–1277, Sep. 2014, doi: 10.1109/TSMC.2014.2298379.
[15] M. Uzam and A. Jones, "Discrete event control system design using automation Petri nets and their ladder diagram implementation," *Int. J. Adv. Manuf. Technol.*, vol. 14 , pp. 716–728, Jan. 1998, doi: 10.1109/TAC.2012.2200372.
[16] G. Frey, "Automatic implementation of Petri net based control algorithms on PLC," in *Proc. Amer. Control Conf. (ACC)*, Chicago, IL, USA, Jun. 2000, pp. 2819–2823, doi: 10.1109/ACC.2000.878725.
[17] M. Minas and G. Frey, "Visual PLC-programming using signal interpreted Petri nets," in *Proc. Amer. Control Conf.*, Anchorage, AK, USA, May 2002, pp. 5019–5024, doi: 10.1109/ACC.2002.1025461.
[18] S. Klein, G. Frey, and M. Minas, "PLC programming with signal interpreted Petri nets," in *Proc. 24th Int. Conf.*, Eindhoven, The Netherlands, Jun. 2003, pp. 440–449, doi: 10.1007/3-540-44919-1_27.
[19] G. Lee, Z. Han, and J. S. Lee, "Automatic generation of ladder diagram with control Petri net," *J. Intell. Manuf.*, vol. 15, pp. 245–252, Apr. 2004, doi: 10.1023/B%3AJIMS.0000018036.84607.37.
[20] S. S. Peng and M. C. Zhou, "Ladder diagram and Petri-Net-Based discrete-event control design methods," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 34, no. 4, pp. 523–531, Nov. 2004, doi: 10.1109/TSMCC.2004.829286.
[21] D. F. Bender, B. Combemale, X. Crégut, J. Farines, B. Berthomieu, and F. Vernadat, "Ladder metamodeling and PLC program validation through time Petri nets," in *Proc. 4th Eur. Conf. Model Driven Archit.-Found. Appl. (ECMDA-FA)*, Berlin, Germany, Jun. 2008, pp. 121–136, doi: 10.1007/978-3-540-69100-6_9.
[22] S. Korotkin, G. Zaidner, B. Cohen, A. Ellenbogen, M. Arad, and Y. Cohen, "A Petri net formal design methodology for discrete-event control of industrial automated systems," in *Proc. IEEE 26th Conv. Electr. Electron. Eng.*, Nov. 2010, pp. 431–435, doi: 10.1109/EEEI.2010.5662187.
[23] M. Heiner and T. Menzel, "A Petri net semantics for the PIC language instruction list," *Proc. IEE Workshop Discrete Event Syst. (WODES)*, Jun. 1998, pp. 161–165.
[24] L. D. da Silva, L. P. de Assis Barbosa, K. Gorgonio, A. Perkusich, and A. M. N. Lima, "On the automatic generation of timed automata models from function block diagrams for safety instrumented systems," in *Proc. 34th Annu. Conf. IEEE Ind. Electron.*, Nov. 2008, pp. 291–296, doi: 10.1109/IECON.2008.4757968.
[25] J. Kluska and Z. Hajduk, "Digital implementation of fuzzy Petri net based on asynchronous fuzzy RS flip-flop," in *Proc. 7th Int. Conf. Artif. Intell. Soft Comput. (ICAISC)*, Zakopane, Poland, Jun. 2004, pp. 314–319, doi: 10.1007/978-3-540-24844-6_44.
[26] M. Markiewicz, L. Surdej, and L. Gniewek, "Transformation of a fuzzy interpreted Petri net diagram into structured text code," in *Proc. 21st Int. Conf. Methods Models Autom. Robot. (MMAR)*, Miedzyzdroje, Poland, Aug. 2016, pp. 94–99, doi: 10.1109/MMAR.2016.7575114.
[27] M. Markiewicz and L. Gniewek, "A program model of fuzzy interpreted Petri net to control discrete event systems," *Appl. Sci.*, vol. 7, no. 4, p. 422, Apr. 2017, doi: 10.3390/app7040422.
[28] J. Kluska and Z. Hajduk, "Hardware implementation of P1-TS fuzzy rule-based systems on FPGA," in *Proc. 12th Int. Conf. Artif. Intell. Soft Comput. (ICAISC)*, Zakopane, Poland, Jun. 2013, pp. 282–293, doi: 10.1007/978-3-642-38658-9_26.
[29] Z. Hajduk, B. Trybus, and J. Sadolewski, "Architecture of FPGA embedded multiprocessor programmable controller," *IEEE Trans. Ind. Electron.*, vol. 62, no. 5, pp. 2952–2961, May 2015.

ZBIGNIEW HAJDUK received the Ph.D. degree in computer engineering from the University of Zielona Góra, Poland, in 2006, and the D.Sc. degree from the Czestochowa University of Technology, Poland, in 2019. He is currently an Associate Professor with the Department of Computer and Control Engineering, Rzeszów University of Technology. His main area of interest includes digital systems design with FPGAs.

JOLANTA WOJTOWICZ was born in Krosno, Poland, in 1974. She received the B.S. and M.S. degrees in computer science from the University of Rzeszow, in 2000, and the Ph.D. degree in computer science from the AGH University of Science and Technology, Cracow, in 2007. Her research interests include modeling of dynamic systems and applications of machine learning methods in control and simulation of processes.