ISBN 978-987-4417-90-9

Extended Petri Net Processor for Embedded Systems

Luis Orlando Ventre, Orlando Micolini, and Emiliano Daniele

Depto. Computación Lab. de Arquitectura de Computadoras
Facultad de Ciencias Exactas Físicas y Naturales - U.N.C
Córdoba, Argentina
{luis.ventre,orlando.micolini}@unc.edu.ar,emiliano.daniele@mi.unc.edu.ar
https://fcefyn.unc.edu.ar/

Abstract. The evolution of technology and electronic devices, the widespread use of IoT, and the compliance with specific regulatory requirements of the industry have made the process of designing embedded systems more complex and challenging. These systems are generally parallel, concurrent, reactive, and/or event-driven. In these systems, the data and events are heterogeneous and non-deterministic as they interact with the external environment. Extended Petri nets constitute an elective platform and system-independent modeling language, which makes it appropriate for modeling embedded systems. To take full advantage of the modeling efforts, it is desirable to use the built models to obtain part of the system implementation. This paper presents the design and implementation of an Extended Petri Processor and its modular architecture. This processor makes use of the extended state equation of Petri Nets, executing the model of the mentioned systems, intending to mitigate the time needed for development, and to reduce programming errors.

Keywords: Petri Processor, Petri Nets, Synchronization, IoT, Code Generation, FPGA.

1 Introduction

Nowadays, critical, reactive (RS) and event-driven embedded systems (EDA) [1] are in high demand, especially by Industry 4.0 [2]. The design of these systems must meet strict non-functional requirements since they are parallel, concurrent systems and interact with variables and events from the system itself and from the outside world, where the data and events are heterogeneous and non-deterministic [3]. This article presents the design and implementation of an extended Petri processor and its modular architecture, for the development of these types of systems. This processor aims to reduce development time and mitigate programming errors.

The design phases of an embedded system include the development of the model based on a set of requirements [4]. This model is the basis for other stages, including the application building stage [5].

ISBN 978-987-4417-90-9

Non-autonomous and extended PN is a general-purpose modeling language that supports the modeling of reactive, concurrent, and parallel systems regardless of the platform. In the design of RS and EDA, the transformation of the model into software implies a translation work that leads to interpretation and implementation errors. In order to mitigate these errors, this processor capable of running the model regardless of the platform has been developed.

According to our research documents, there is no history of automatic code generation or model execution performed by a processor. The innovation of this proposal includes the design and implementation of a modular heterogeneous architecture processor, which executes the extended state equation of PN [12], and which has the expression capacity of a Turing machine. This processor keeps all the properties verified in the model, since the model is essentially not interpreted and/or transcribed in code, but rather executed.

As a precedent to this work, and as part of this same research project, the development of a modular Petri processor (PP) that executes ordinary PN can be found in [6]. The main difference is the expression capacity, which is not that of a Turing machine since it cannot execute models with different types of arms or guards. Also, in [7], a study was carried out on more than 70 references that make use of PN for the solution of SR and EDA.

The following section sets out the objectives of this work; while in section 3 a brief theoretical framework is presented. Then, in section 4, the architecture of the Extended Petri Processor (PPX) is described as the proposed solution. In section 5, the results, the tests carried out and finally the conclusions are exposed.

2 Objectives

The system's logic model contains the necessary information for implementing the software logic. The coding stage involves successive iterative efforts to refine, interpret, and transcript the model into code, detecting and correcting errors in the process. This entails an overload of effort and time in the development stages.

The main objective of this work is the design and implementation of a processor that executes the logic model as code, thus unifying the modeling and coding stages. As a secondary objective, a modular design and interconnection with a traditional processor is carried out.

3 Methodology and Tools

There are several modeling tools, among which are: UML diagrams [8] and Petri Nets (PN) [9, 10]. UML diagrams provide the necessary characteristics, partially, but they essentially lack mechanisms for formal verification that strictly guarantee compliance with critical requirements, which are fundamental aspects to be implemented in the SR and EDA.

Since the extended, non-autonomous, PN [11] allows to model concurrency, local and global state, and parallelism, it is possible to verify them formally.

ISBN 978-987-4417-90-9

They are executable [7] and scalable when expressed with the extended state equation [12]; they have been considered to be the most convenient formalism and have been selected as a modeling tool and a processor execution language.

The development tools and implementation of the PPX are explained and described in subsection 4.5.

3.1 Petri Nets

A marked PN, denoted as PN, is a quadruple [5] defined by:

$$PN = (P, T, I, M_0) \tag{1}$$

Where:

- $P = p_1, p_2, \cdots, p_n$ is a finite, non-empty set of places.
- $T = t_1, t_2, \cdots, t_m$ is a finite, non-empty set of transitions.
- I is the incidence matrix that relates places with transitions and vice versa.
- M_0 is the initial markup of the PN.

3.2 Synchronized or Non-Autonomous PN

This type of PN introduces events into the model and it is an extension of autonomous PN [11] [7]. Non-autonomous PNs model systems in which firings are synchronized with external discrete events. Events are associated with transitions, and the firing occurs when two conditions are met: the transition is enabled and the event associated with the transition had taken place.

External events correspond to changes in the state of the system's environment (including time) while internal events are changes in the state of the system itself. Synchronized PNs can then be defined as a triplet:

$$PN_{sync} = (PN, E, sync) \tag{2}$$

Where:

- PN is a marked PN,
- E is a set of external events and
- $sync$ is the function that relates the transitions T with $E \cup \{e\}$, where $\{e\}$ is the null event, that is, those transitions that are automatically fired.

Perennial, Non-Perennial and Null Events
There are different types of events. A detailed description can be found in [7].

Extended Equation of State
In order to mathematically represent the existence of the new inhibitor, reader and reset arms, a matrix is required for each type of arc. These matrices are similar to the matrix I. When the arcs have a weight equal to one, the terms of the matrix are binary. A transition can be enabled if it meets the following conditions: if it has an inhibitor arm that does not have a token in the associated place; if it has a reading arm, and the place associated has one or more tokens; if

ISBN 978-987-4417-90-9

it has a guard and the guard value is equal to true; if it has an event associated, and one or more events were queued and if it has a label with a time interval, and the counter is in the valid time range.

So the extended equation of state is now defined as:

$$M_{(j+1)} = M_j + I * (\sigma \ and \ Ex)\#A \tag{3}$$

In this expression Ex is the extended enabled vector, represented by:

$$Ex = E \ and \ B \ and \ L \ and \ G \ and \ Z \tag{4}$$

Where E, B, L, G y Z are the enabled vectors of the different arcs. The details of the calculation of (3) and (4) are found in [12].

4 Architecture of the Solution

The hypothesis of this work is based on the fact that a model made with a non-autonomous PN is a set of instructions, equations and restrictions or rules to generate the I/O behavior of a system. That is, the model is described as state transitions and mechanisms to accept input trajectories and generate output trajectories depending on their state.

The defining, in terms of system specifications, has the advantage of a solid mathematical basis and unequivocally defined semantics. To specify a behavior, the model needs an agent. This is basically a computer system capable of executing the model. The same model, expressed in a formalism, can be executed by different agents, thus enabling portability and interoperability at a high level of abstraction.

In this project, the PPX is the agent in charge of executing the model by making use of (3), so that it can generate the desired behavior. Extended PNs allow to model systems of events or stimuli, states, logic, policy and actions, which means it can be decoupled and they manage the control and execution of the whole system.

4.1 Architecture of the PPX

The main blocks of the PPX implement (3) which are: matrix-program, calculation-state, queues, and policies.

4.2 PPX modules

The Fig.1 represents a synthesized version of the processor.

Matrix-Program Modules: responsible for defining the processor program with the matrices and vectors of the state equation. They are represented in Fig.1,identified with $*$, and they are: the matrices I, H, R, Rst, A and Time comparison window, and the vector of automatic firings.

ISBN 978-987-4417-90-9

Calculation-Status Modules: responsible for calculating and maintaining the status of the PN. Its components are marked in Fig.1 with #, which are: the state vector, the new state vector, L, B, V, E, G, S, the timers array, the vector of possible firings, the Adder and the Calculation-reset module.

Queues module: responsible for storing the input and output events and communicating the results of the PN execution with the traditional processor, Microblaze (MCS)[13]. These components are marked in Fig.1 with @ (the firing and exit request queues).

Policies Module: responsible for selecting the transition with the highest priority from the vector of possible firings. Its components have been identified on the Fig.1 with &, they are the Firing Policy Matrix and the highest priority Firing Vector.

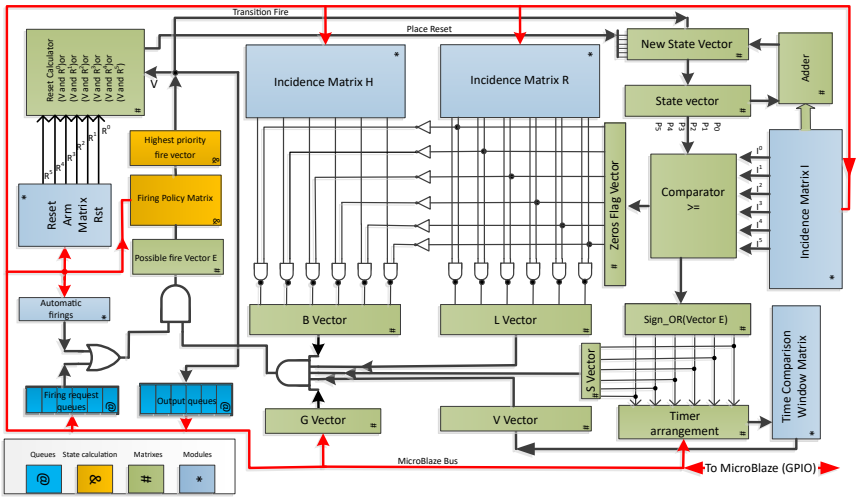

Fig. 1. PPX architecture.

4.3 Modules Description

The parts and functions of each component correspond to (3). They are consistent with the proposed modular structure and the main ones are:

Incidence Matrix I **-** array of integers. Its dimension is $|T| \times |P|$.

Inhibitor Arms Matrix H **-** binary matrix. Its dimension is $|T| \times |P|$.

Reader Arms Matrix R **-** binary matrix. Its dimension is $|T| \times |P|$.

Arm Matrix Reset Rst **-** binary matrix. Its dimension is $|T| \times |P|$.

Time Comparison Window Matrix - stores the alpha and beta values which correspond to the lower and upper time limits [5].

State Vector - vector of positive integers. Its dimension is $|P|$.

ISBN 978-987-4417-90-9

Vector L **-** binary vector. Its dimension is $|T|$. Inhibits the transition if the place is not marked.

Vector B **-** binary vector. Its dimension is $|T|$. Inhibit the transition if the place is marked.

Vector V **-** binary vector. Its dimension is $|T|$. Enable sensitive transition if your timer is in the window range (between alpha-beta).

Guardian Vector G **-** binary vector with the guard values. Its dimension is $|T|$.

Sensitized Vector S **-** binary vector, where each position corresponds to each column of the matrix of possible next states. Its dimension is $|T|$.

Timers arrangement - vector of integers. Its dimension is $|T|$. They are counters that are activated when the associated transition is enabled and are reset when it is disabled or fired.

Firing Request Queues - its interface exposes an input vector to the PPX, where each position of the vector corresponds to a transition.

Exit Queues - its interface exposes an output vector from the PPX, where each position of the vector corresponds to a transition.

Firing Policy Matrix - binary matrix of dimension $|T| \times |T|$. Its values indicate the relative priority between transitions.

Highest Priority Firing Vector - binary vector that represents the transition to be fired.

4.4 Processor Algorithm

Single-server semantics have been adopted in this work, so only one transition is fired at a time. Two cycles are required for each firing to determine and report the new status.

Cicle 1 - Calculations - In this cycle, the necessary calculations are performed to determine which transition to fire.

The tagging vector is compared with each column of the incidence matrix, denoted in Fig.1 as I^i, to get the sign bit of each element of the array of possible next states. The results are binary columns where the i-th column contains the signs of the values of the next marking vector, in case the transition i is executed. This matrix contains only the signs of the possible next states; given a column, if any of the values is negative, it means that the next state will not be reachable by the PN (negative values in a tagging vector indicate that the transition is not enabled). To obtain this value, a logical disjunction is performed between all the elements of each column. The vector S is built using these values.

For each element of the state vector, a zero bit is determined. With these bits, the zeros flag vector is constructed. This indicates whether or not the place is marked and it is indicated in Fig.1 as Zeros Flag. The vectors B y L are calculated with the product of the Zeros Flag vector and the matrices H and R, respectively.

The Guards Vector G is updated from the MCS processor, since it represents all the conditions that are external to the PPX processor.

ISBN 978-987-4417-90-9

The vector V indicates whether a transition is enabled (based on the values of the vector S), and if it is in the time window programmed in the Array of Timers.

The transitions that are possible to be fired are obtained from the logical conjunction between the vectors B, L, G, V and S. The logical disjunction between the Firing Request Queue vector and the Automatic Firings indicates the transitions that are requested to fire. Next, the logical conjunction between these last two vectors is carried out, which indicates the possible firings, noted as E in Fig.1. This vector contains a value of 1 in the positions of the transitions that are possible to fire. Since single-server semantics have been adopted, it is necessary to determine the highest priority transition to fire. To achieve this, the Firing Policies Matrix is used. This returns the highest priority Firing Vector, which contains a value equal to 1 in the transition to fire. In this cycle it is also calculated if any place should be zeroed, so the internal product is performed between the columns R^i of the matrix of reset arms (Rst) and the highest priority Firing Vector. This calculation is carried out by the module named Reset_calculator. If the value is one, the place reset bus sets the place value to zero.

Cycle 2 - Update - This is the cycle that computes the firing of the transition that was selected in the previous cycle. Here, the firing takes effect and the value of the marking vector is updated. To achieve this, the selected transition works as a column selector.

The adder, which is at the top of the Fig.1, performs the sum of the marking vector with the matrix column I selected by the firing vector. The result of that sum is stored as the new marking vector. In this cycle, if applicable, the space indicated on the reset bus is reset. Additionally, the queues are updated; increasing the output queue counter and decrementing the input queue counter.

4.5 PPX, FPGA, and Microblaze MCS

The PPX solves the logic of the system so it operates with an associated processor [7][6] which executes the actions required. To achieve this, the PPX was interconnected with the MCS processor as shown in Fig.2. It was implemented in a Spartan 6 FPGA from Xilinx (Atlys) [14], in which an IP-core MCS was installed since it is included in the ISE tool [15] and it has a low impact on the resources required for its implementation. A communications module (UART) was also installed in order to carry out the testing and a clock management module (DCM). This configuration has been selected to establish comparisons with the work carried out in [6].

4.6 Queues

The input and output queues of the PPX are configurable, each transition has an input queue and an associated output queue. A detailed description is found in [6].

ISBN 978-987-4417-90-9

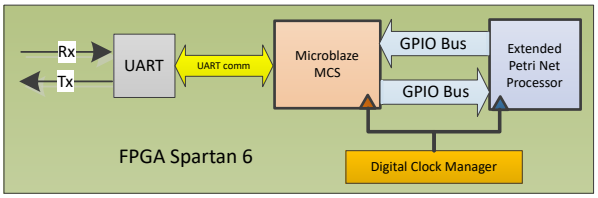

Fig. 2. Interconnection between processors.

4.7 Priorities and Conflicts between Transitions

The PPX does not detect conflict states [11] between transitions, this is why it treats all enabled transitions as if they were in conflict (single server). This semantics, in conjunction with the priority policy module, also solves the problem of conflicts and makes it deterministic. The Firing Policies Matrix module determines the transition to fire, this module is configurable at runtime.

5 Results

With the PPX-MCS heterogeneous architecture, different application cases were executed to evaluate its performance. The successful executions of the cases were raised in [16] [17]. The comparison of resources has been carried out taking into account the results obtained in [6]. For the purposes of this comparison, the configuration, FPGA and development tools selected were the same.

FPGA Resource Consumption - The processor was installed with different configurations of vector and matrix elements. Each configuration is expressed with a triplet of integers, which are: $P \times T \times Pa$, where P is the number of places, T the number of transitions, and Pa the length of the word that represents the weight of the arcs and the amount of tokens that a place supports. Multiple kernel syntheses were performed using different word-lengths (4-bit and 8-bit data). In Fig.3 a) the amount of resources that were used from the FPGA is displayed for synthesized configurations. In Fig.3 a) the exponential increase in the consumption of resources is observed, as the number of elements of the matrices and vectors increase. In Fig.3 b) and Fig.4 b) the consumption of resources of the PPX is compared to the PP, where the average increase of LUTs is 10% while the increase in register consumption is 18%. Overall, the impact of the MCS processor in resources is 12.72% for LUTs, and 21.09% for registers. These resources are the same for all the synthesized configurations of the PPX.

Frequency Analysis - The maximum theoretical frequencies for the different processor instances are shown in Fig.4 a). Since optimizations have been made to the modules and interconnects, the PPX has achieved a substantial improvement in frequency over the PP. It is observed that for a configuration of 8x8x8, the maximum frequency is 257MHz while for 8x8x4 it is 271MHz; for a 16x16x8 configuration the frequency is 190 MHz and for 16x16x4 it is 216 MHz. It should be noted that the decrease in frequency, with respect to the length

ISBN 978-987-4417-90-9

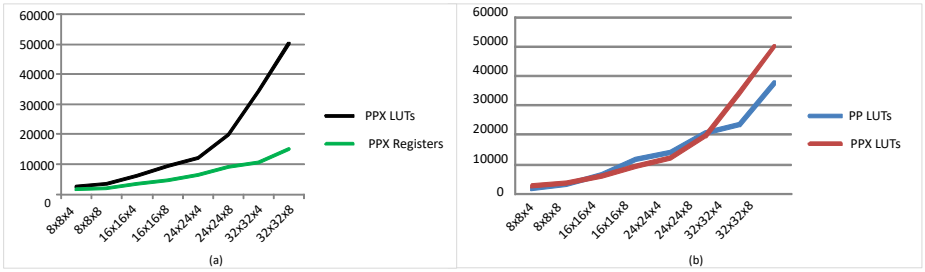

Fig. 3. a) Consumption of LUTs and PPX Registers. b) Comparison of the use of LUTs between PPX and PP.

of the word, is 18% in average. On the other hand, if the length of the word is maintained and the number of places and transitions is increased, that is, the size of the PN, the difference in frequency is significantly larger, on average 30%.

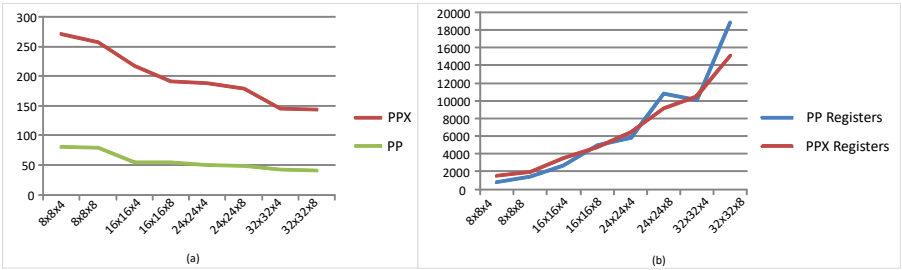

Fig. 4. a) Comparison of theoretical maximum frequencies between PPX and PP. b) Comparison of Records between the PPX and the PP.

6 Conclusion

In this project, a processor (PPX) was designed and implemented for executing the extended state equation, with a modular architecture. This processor extends the semantic capacity of the PP developed in [6]. The inclusions of different types of arcs, temporal semantics, and guards in the PPX give the PPX the expression capacity of a Turing machine without significantly increasing the necessary resources. Queue scheduling and support for different types of events have been maintained, as well as the communications module. The results show that the PPX is suitable, for the FPGA selected, for embedded systems that require up to 32 composite logical conditions, 32 logical variables, and 32 events that must be evaluated simultaneously. From the data obtained in the frequency

ISBN 978-987-4417-90-9

analysis, the substantial improvement in the maximum theoretical frequency with respect to the PP stands out. The serial communication module has facilitated the configuration, debugging, and programming tests from a console. The modular implementation of the PPX implies a breakthrough for maintenance, scalability and future autoconfiguration.

Referencies

1. N. Halbwachs, *Synchronous programming of reactive systems.* Springer Science & Business Media, 2013, vol. 215.
2. K. Schwab, *The fourth industrial revolution.* Currency, 2017.
3. A. Munir, A. Gordon-Ross, and S. Ranka, *Modeling and optimization of parallel and distributed embedded systems.* John Wiley & Sons, 2015.
4. B. P. Zeigler, A. Muzy, and E. Kofman, *Theory of Modeling and Simulation: Discrete Event & Iterative System Computational Foundations.* Academic press, 2018.
5. M. Diaz, *Petri nets: fundamental models, verification and applications.* John Wiley & Sons, 2013.
6. O. Micolini, E. N. Daniele, and L. O. Ventre, "Modular petri net processor for embedded systems," in *Argentine Congress of Computer Science.* Springer, 2017, pp. 199–208.
7. O. Micolini, "Arquitectura asimétrica multicore con procesador de petri," Ph.D. dissertation, Facultad de Informática, UNLP 2015.
8. B. Selic and S. Gérard, *Modeling and analysis of real-time and embedded systems with UML and MARTE: Developing cyber-physical systems.* Elsevier, 2013.
9. M. Zhou and N. Wu, *System modeling and control with resource-oriented Petri nets.* Crc Press, 2018.
10. S. Siewert, *Real-time embedded components and systems.* Cengage Learning, 2016.
11. R. David and H. Alla, *Discrete, continuous, and hybrid Petri nets.* Springer, 2005, vol. 1.
12. O. Micolini, L. O. Ventre, and M. I. Schild, "Generalized state equation for non-autonomous petri nets with different types of arcs," 2016.
13. P. D. Group and ATLAS, "Xilinx, microblaze processor reference guide."
14. C. Digilent. Atlys spartan-6 fpga trainer board. [Online] Available: `https://store.digilentinc.com/`
15. C. XILINX. Ise webpack design software. [Online]. Available: `https://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.html`
16. O. Micolini, L. O. Ventre, and M. Ludemann, "Methodology for design and development of embedded and reactive systems based on petri nets," in *2018 IEEE Biennial Congress of Argentina (ARGENCON).* IEEE, 2018, pp. 1–7.
17. O. Micolini, L. O. Ventre, M. Ludemann, J. I. R. Viano, and C. C. Bien, "Case study of reactive and embedded system design modeled with petri nets," in *2018 IEEE International Conference on Automation(ICA-ACCA).* IEEE, 2018, pp. 1–7.