IP Core for Timed Petri Nets

Orlando Micolini
Laboratorio de Arquitectura de Computadoras
FCEFyN-UNC
Córdoba, Argentina
omicolini@compuar.com

Julián Nonino
noninojulian@gmail.com

Carlos R. Pisetta
renzopisetta@gmail.com

Abstract—**In this article, we present a Timed Petri Nets Processor which can be directly programmed using vectors and matrixes of Petri Nets formalism. This processor can leverage the power of Petri Nets for modeling real-time systems and formally verify their properties, which prevent programming errors.**

The Petri Nets Processor was developed as an IP-core to be inserted in a Multi-Core system. Therefore, we can model the system requirements with Petri Nets, formally verifying all its properties and using the IP-core to implement the system is possible to ensure that all properties will be met.

Keywords—Multi-Core, Petri Net, Processor.

I. Introduction

The evolution of the processors is consequence of the greater integration and composition of different types of functionalities integrated into a single processor. The availability of transistors has made possible to integrate several processor cores on a single chip, which has resulted in the development of Multi-Core technology [1].

Diminishing returns of Instruction Level Parallelism (ILP) and the cost of the increase of frequency, mainly due to power limitations (suggests that a 1% increase in clock speed results in a power increase of 3%) [2], leads to the use of Multi-Core processors to improve performance. This increase deficiency results in lower run times, lower consumption, lower energy density, lower latency and higher bandwidth inter-core communications.

Therefore Multi-Core processors are a proposal to obtain higher performance. This arises as a better performance of each of those parameters. Furthermore, the heterogeneous Multi-Core systems have the advantage of employing specialized cores, each of them designed for specific tasks. That is, optimized for a particular need. These processors have the ability to use the available hardware resources when they are specifically required by the software [3].

In order to increase performance, these systems make use of multi-threading and/or multi-tasking allowing take advantage of the Multi-Cores. However, it takes more effort to design applications because they must provide solution to the problems of concurrent systems.

That is the reason why with these processors, the parallel programming is essential for improving the performance in all segments of software development and even more so in the segment of real-time systems.

Petri model is suitable to implement, validate and verify a parallel system with concurrence, At the Computer Architectures Laboratory of the FCEFyN-UNC a Petri processor has been developed to directly execute ordinary Petri Nets. In this article, we present a new Petri Nets Processor capable to execute Timed Petri Nets and to be programmed directly with the vectors and matrixes that define the system and its state.

There are different ways to implement Petri Nets of software and hardware. Its remarkable that, through our research, it hasn’t been found works similar to ours. Here we enumerate the most distinguished ones and the main difference between them and our work.

- Gary A. Bundell only implements the shoot algorithm of a transition [4].
- Hideki Murakoshi, Miki Sugiyama, and Guojun Ding describe a matrix of Petri controller which it is not programmable [5].
- Sergio C. Brofferio has implemented a controller without using state equation [6].
- Ramón Piedrafita Moreno and José Luis Villarroel Salcedo. They programmed a high level software controller [7].
- Xianwen Fang, Zhicai XU y Zhixiang Yin don’t use the state equation as a program. They only program with a high level language [8].
- Another researchers:
    - Murakoshi, Hideki [9].
    - Wegrzyn, Marek; Wolanski, Pawel; Adamski [10].
    - Aybar, Aydın and Iftar, Altuğ [11].
    - Murakoshi, Hideki [12].

II. Objectives

A. Main Objective

The main objective of this work is to design and implement a Petri Nets Processor capable to execute the Timed Petri Nets semantics and to be programmed directly from the model’s state equations.

B. Secondary Objectives

The secondary objectives are:

- Briefly describe Timed Petri Nets in order to implement a processor capable of execute them.
- Keep executing ordinary Petri Nets with time parameters on two processor clock cycles.
- Implement the Timed Petri Nets Processor as an IP-core.

III. Petri Nets Considering Time

In the formalism of Ordinary Petri Nets, the time is not considered and this results in indeterminism regarding time. It is not specified when a sensibilized transition will be fired or even if it will be fired. Neither can be said which transition from a group of transitions in conflict will be fired.

There are three different interpretations about how the time should be consider. All of them have its focus on reducing the indeterminism regarding time in Petri Nets [13]:

- Stochastic Petri Nets: Introduces a stochastic estimation on the instant of firing of a transition.
- Timed Petri Nets: Introduces a time condition, which specifies the duration of the transition.
- Time Petri Nets: Introduces temporary dimensions between which the transition should be fired.

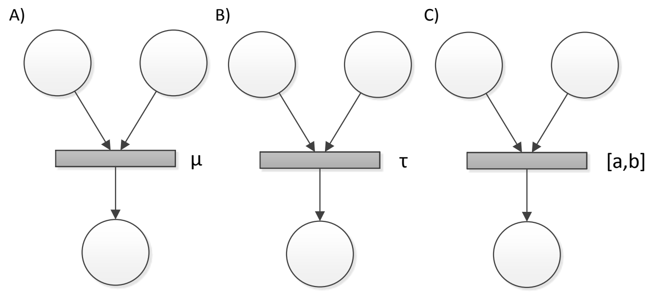

Figure 1 Different ways to introduce time in Petri Nets

The temporal parameters associated with transitions can be interpreted in these three different ways[1]:

1. Generalized Stochastic Petri Nets (GSPNs) [14]have two different types of transitions: immediate transitions and timed transitions. When a transition t is sensitized, its firing could be: a) with a duration equal to zero if the transition t is immediate. b) after a lapse of a random time. This random time is expressed by an exponential distribution. The A Net from the figure 1 graphically represents a stochastic timed transition where its probability to be fired is represented by μ.
2. Timed Petri Nets have two different types of transitions: immediate transitions and timed transitions. When a transition t is sensitized, its firing could be: a) with a duration equal to zero if the transition t is immediate. b) with immediate removal of tokens from set •t but placing the tokens in the $t\bullet$ only after time τ has elapsed. Meanwhile, the transition cannot be sensitized. The B Net from the figure 1 graphically represents a timed transition with a delay equal to τ.
3. Timed Petri Nets have two different types of transitions: immediate transitions and time transitions. When a transition t is sensitized, its firing could be: a) with a duration equal to zero if the transition t is immediate. b) if it is a time transition, at the time it is sensitized, a timer starts. The transition can only be fired when the timer value is between the limits of the interval [a, b]. Otherwise, the transition cannot be fired. Once the firing was performed, the timer is restarted. The C Net from the figure 1 graphically represents a time transition with an associated interval equal to [a, b].

Should be noted that all the firings are performed in two steps: a) the removal of the tokens from the set $\bullet\, t$. This is an atomic action and the amount of tokens removed from each place is equal to the weight of the arcs joining each place in •t with the transition t. b) The atomic action of placing in each place of set $t\,\bullet$ the amount of tokens indicated by the weight of the arcs joining each place of t• with the transition t.

IV. Timed Petri Nets

In this nets, each timed transition has an associated parameter τ which represents the duration of the transition. In order to standardize the mathematical definition we will call immediate those transitions where τ is zero.

A. Mathematical Definition

A Marked Timed Petri Net [13], is mathematically defined as a8-tuple as follows:

$$\{P,T,I^+,I^-,H,C,\ m_0,\Gamma\}$$

Where the terms $\{P, T, I^+, I^-, H, C, m_0\}$ represent a marked Petri Net with inhibitors arcs and bounded places. Γ is a vector composed of the values of duration τ associated to each transition.

The meaning of each term of the tuple is:

P: is a non-empty finite set of places

T: is a non-empty finite set of transitions, $P \cap T = \emptyset$.

***I*⁺, *I*⁻** : are the positive and negative incidence matrixes

$$PxT\ \rightarrow\ Z$$

H: is the inhibitors arcs matrix.

$$PxT\ \rightarrow \{0,1\}$$

C: is a vector containing the values that represent the maximum amount of tokens that each place of the net can hold.

$$C \rightarrow N$$

Γ: is the set of static intervals associated with each transition

$$T \rightarrow \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \infty)$$

For each transition$t$ the associated value $timer_t$ is:

[1]$\bullet\, t$ is the set of places that are inputs to a transition, mathematically defined as: $\bullet\, t\ =\ \{p\ \in\ P{:}\,(p,t)\ \in\ F\}$
$t\,\bullet$ is the set of positions that are outputs of a transition, mathematically defined as: $t\,\bullet = \{p\ \in\ P{:}\,(t,p)\ \in\ F\}$
F is the set of arcs, input and output to the transitions

$$\Gamma\,(t) = \tau_t\,, where\; t\; \in\; T\;\; and\;\; \Gamma \rightarrow\; \mathbb{Q}^{+}$$

$timer_t$ represents the time elapsed since the firing of the transition *start and its value is zero at any other moment.* τ_t is the duration of the transition. For that reasons, the following conditions must be met:

$$0\; \leq\; timer_t\; \leq\; \tau_t.$$

$$0\; \leq\; \tau_t <\; \infty$$

m0: is the net initial marking and must fulfill that: *Γ=0.*

$$P \rightarrow N$$

B. States of a Timed Petri Net

In these Petri Nets, the net state is defined by the marking vector (m_i) and a timer vector that indicates the timestamp of each transition. Therefore the net state is:

$$S= (m_{i,}timer)$$

C. Sensitization of the Transitions and Firing Rules

When we refer to transitions, we have to establish the difference between an enabled or sensitized transition, a not enabled or sensitized transition and the firing of a transition.

In a Marked Petri Net whose current mark is $\boldsymbol{m_k}$, we say that transition t_j is enabled or sensitized if and only if $timer_t =$ 0 and the amount of tokens in all places p_i belonging to these $\bullet\, t_j$ is at least equal to the weight of the arc that connects them with the transition $\boldsymbol{t_j}$ $\left(w(p_i, t_j)\right)$. Mathematically:

$$p_i\; \in\;\; \bullet\, t_i, m(p_j) \geq w(p_i, t_j) \wedge timer_{t_j} = 0$$

In summary, every place connected to the transition t_j has at least the number of tokens indicated by the weight of the arc and there is no firing in progress for that transition.

Sensitized transitions can be fired and every time the firing of a transition is completed it generates a new marking for the Petri Net. This means that the net changes its state.

The equation to calculate the new state or new marking reached by the firing of t_j is $\partial\;(m_k, t_j)$, and it is defined as:

$$\partial\;(m_k, t_j) = \begin{cases} m_{k+1}(p_i) =\; m_k(p_i) - w_{ij}\,, & \forall\; p_i \in \bullet\, t_j \\ \quad timer_{t_j}\; start\;; & \\ m_{k+1}(p_i) =\; m_k(p_i) + w_{ji}\,, & \forall\; p_i \in\; t_j \bullet \wedge \\ \quad timer_{t_j} = 0; & timer = \tau_j \\ m_{k+1}(p_i) =\; m_k(p_i); & in\; the\; rest \\ & of\; the\; cases; \end{cases}$$

$timer_{t_j}$ is incremented in every clock cycle after the firing of the transition has started.

D. Interpretation of the firing of transitions in the system

The Figure 2 represents a reactive system that responds to events, which come from the environment, in other words the system interacts with the environment. Those events are directed to the Time Petri Nets Processor.

The responsibility of the processor is to arrange events according to the system constraints. These constraints are modeled by the Timed Petri Net which is used to program the processor. On the other hand, Multi-Core system threads also generate events (to request resources, to synchronize) that are directed to the processor to be sorted according to system constraints.

Module 1 from the Figure 2 receives unsorted events from the environment and from the system itself. After sorting them, the Timed Petri Nets Processor transmits the result to the threads execution cores (module 2) of the system and the proper actions are taken [15].

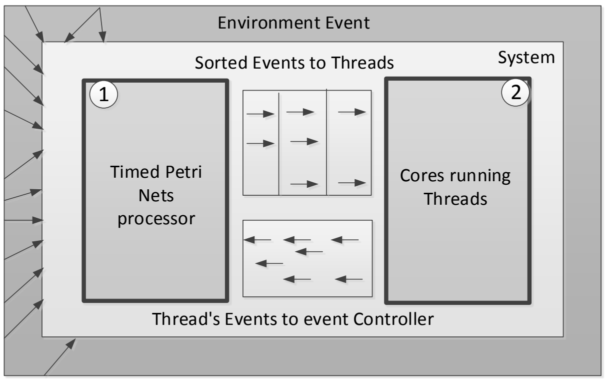

Figure 2 Reactive systems

In our system the fulfillment of program conditions is associated to sensitized transitions, the resolution of a shot represents the fulfillment of those restrictions and if we associate the request for verification of the conditions to a shot request, the resolution of a shot communicates that conditions have been met.

Definition: conditions for firing a transition from Timed Petri Process:

1. The transition must meet the sensitization conditions of section IV.C
2. The shot must be explicitly communicated by the processes or implicitly recorded in the automatic shots module.
3. Since it is possible that multiple transitions simultaneously satisfy the conditions described in paragraphs 1 and 2, the Timed Petri Nets Processor will execute first the firing with higher priority.

Figure 3 show us how the Timed Petri Nets Processor is connected in a Multi-Core system.

In case that the firing of the transition cannot be resolve, it is queued in the input queue, as shown in Figure 3, until the conditions of the system allows its resolution. The solution of the firing is notified to threads through the system bus, using the output queue. The threads of the system will execute the proper actions as indicated by the firings that have been resolved, since the resolution of the firings depends on the

Time Petri Nets Processor state, which itself represents the state of the system.

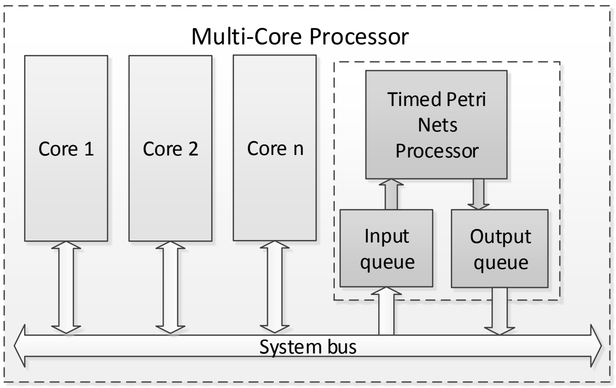

Figure 3 Multi-Core with Timed Petri Nets Processor

The explicit shots require an external event, if this event does not occur the system must contemplate it in the network design. On the other hand the implicit shots automatically generate events due to the fulfillment of the transition conditions. In both cases it is necessary to take into account the priorities, so a shot can be delayed for not being the one with the highest priority. In both cases, the non-fulfillment of the times, deadlock the system, for which the active system signal is tested. This signal is included for debugging and / or to recover the system from a deadlock.

V. Timed Petri Nets Processor Architecture and Operation

The processor executes the state equation solving only one firing of a transition at a time, this way it can solve all cases of firings, the simple ones (single firings) and the multiple firings, performing as a single-firing sequence, as a result, the hardware is simpler.

The resolution of firings is requested by the threads running on the cores through the system bus, as emerging requests that system is running. These firing requests are received by the Timed Petri processor and stored in the input queue. Each transition has a FIFO queue, the output of each queue is a bit the composition of all outputs, which are all shots, form a word the output of this queue size is a word equal to the number of transitions. This word has ones in the positions corresponding to transitions with firings requested. The order of the bit in the word equals the number of transition over which the firing is requested. The bits that correspond to the transitions which have no firing request are zero.

The output queue has a similar structure, but its function is to communicate to the threads those firings that have been resolved.

The data I/O module manages the access of the cores to the matrixes and vectors that program the system. The module manages the access of the cores to the matrixes and vectors that program the system.

The matrixes and vectors described in the equation of state are the system program. This allows us to program the processor directly from the Timed Petri Net.

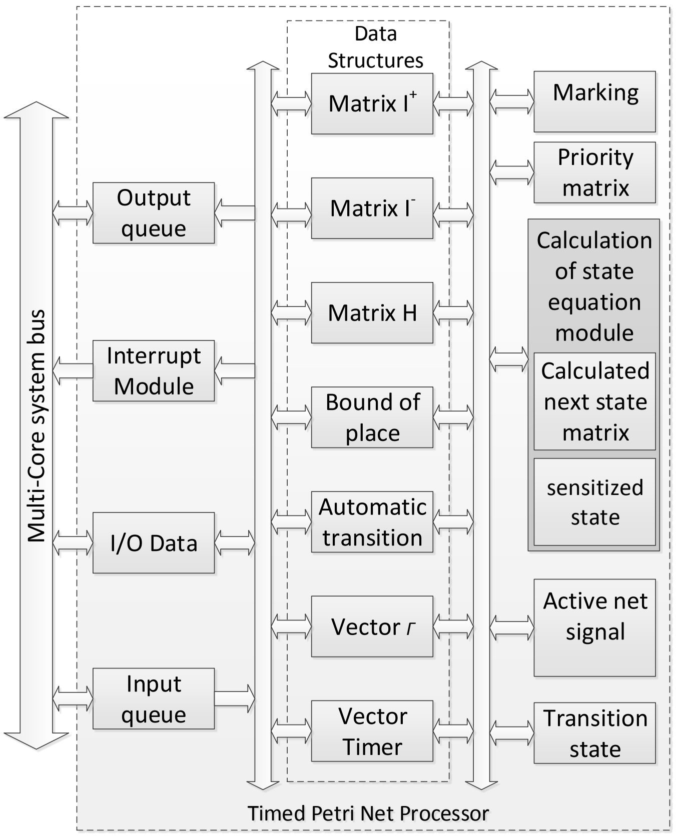

Figure 4 Timed Petri Nets Processor

Here we have added the inhibitor arcs matrix and the vector indicating the maximum number of tokens in the places. This terms are not present in the state equations shown in this work but you can consult work [16].

The module in charge of solving the state equation of the Petri has the following responsibilities:

1. Calculating the new state that would result from each transition firing only once, thereby generating a number of vectors calculated states equal to the number of transitions. Then, these vectors are stored. This is performed by subtracting the current state parallel to each column of I^- and storing all resulting vectors, which will be evaluated to determine if the new state that each transition would produce is valid. This operation is performed whenever you change the timed Petri Nets Processor status (current marking vector).

2. Determine which transition is sensitized. To do this, take all vectors calculated in step 1 and verify that there is no place to have a negative marking and neither exceeding the limit[2] of tokens it can hold.

3. From the group of sensitized transitions determined at step 2 and its firing has been requested we select the one with higher priority. This transition will be used

[2] It is noted that this is a weak bound, since the marks in the squares are incremented in step 4 and the limit is checked in step 2. This simplification facilitates hardware implementation.

to determine the new state of the net. This update of the marking vector will be perform by replacing of the current vector with the one calculated in step 1 corresponding to the selected transition. At that moment, in Timer Module, starts the timer corresponding to the transition fired.

4. Compare each component of the Γ vector with the one in Timer vector and verify that it meets the following condition:

$$\text{Vector } \Gamma_t \leq \text{ Vector Timer}_t$$

5. The fulfillment of this condition means that the transition t has reached the delay time required. Then, the transition of higher priority than meets the above condition is chosen to update the marking vector. This means, add to the current marking vector the column of the matrix I^+ corresponding to the transition t. At the same time the position t of the $Timer$ vector is set to zero ($Timer_t = 0$).

6. Execute the steps 1, 2, 3 and 4 as a continuous cycle.

The system also has a unit that detects when no transition is sensitized and the $Timer$ vector is zero. When this happens, the system generates an interruption notifying that the system has finished its execution or is deadlocked. This feature is very useful to verify the operation of the design and implementation of the system.

VI. Performance Analysis

System implementation has been performed on a Atlys ™ Spartan-6 Digilent platform [17], cores used are the MicroBlaze v8.40 [18] running an XilKernel v5.01a operating system. Interconnected with Timed Petri Processor by AXI bus [19].

To verify the correct operation and analyze the IP Core synchronization times, measurements were made for different numbers of iterations and number of threads trying to access a shared variable in mutual exclusion. Then we compared the Petri Processor with an implementation using semaphores, both solving the same problem. The choice of this second synchronization method is based on that they are the lightest mechanism to perform these tasks.

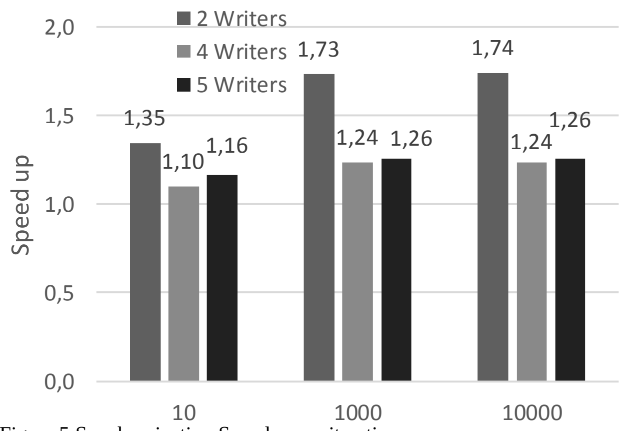

Figure 5 Synchronization Speedup per iteration

From these measurements, Speedup was calculated. The results are shown in Figure 5, where can be observed that for all cases, the processor is on average between 15% and 30% faster than use of semaphores to solve the trouble of synchronizing multiple threads that want to accesses a shared resource and even show peaks up to 70%.

This paper establishes if could be obtain performance improvements in synchronize with respect to semaphores and compare the resources used by the Petri Processor with others of the same kind.

Taking into account that the processing times are composed by:

- Runtime: These correspond to the execution of sequential operations.
- Waiting time: own of the algorithm, to synchronize, to avoid race conditions, etc.
- Times to determine synchronization, race conditions, etc.

Developing a Petri processor seeks to reduce this last timeslot, which rely exclusively on Petri processor's ability to perform this task (such as semaphores depend on OS implementation). With incorporating temporal semantics, the processors can check on execute conditions compliance own of the algorithm. Currently this work is do with soft or hard timers own of the microcontroller or outside. On the other hand the execution times depend on the other processors of the Multi-Core system for which we do not consider it in the analysis. To evaluate the synchronization ability of our processor, the execution times and waiting times must be zero. The result will be compared to the time that semaphores consume to perform the same synchronization task, which is done in the performance analysis section.

As observed in Figure 6, the processor needs only one half-clock cycle since the counter reaches the value τ until a shot is placed in the output queue. The delay introduced is insignificant in relation to the time takes a δt of one clock cycle.

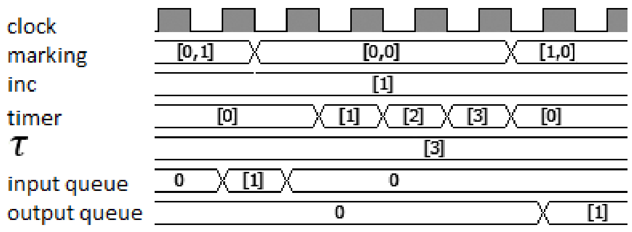

Figure 6 Running on hardware

VII. Ip Core Growth

The processor's growth was analyzed in function of the parameters that it has. For this purpose, processors of 8x8, 16x16 and 32x32 (places x transitions) were generated, with capacity of 7 bits by place and elements of time of 48 bits. The results are plotted in Figure 7.

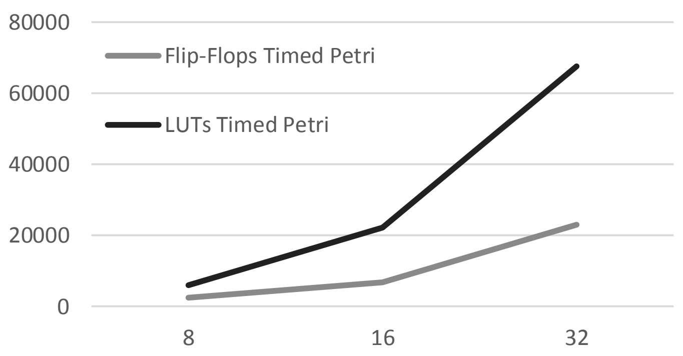

Figure 7 IP Core Growth

Is observed that the growth of IP Core is not something to ignore, since the number of elements used grows quickly with the product of places and transitions.

VIII. Conclusion and Contributions

In this paper, is developed a Timed Petri Processor, which decouples the concurrency from sequential processing, it has the following particularities:

- Development time can be reduced, since the system is verified in analysis and design stages
- On tasks, where measurements have been done, the processor allows synchronization of threads, with improvements up to 70%.
- There is a direct relationship between the graph and the processor program, since this is programmed with the matrixes and vectors of the state equation.
- Are admitted multiple shots simultaneously in the same transition.
- Allows priorities programming; since the shots are solved in parallel and are selected according to priorities module.
- Decides if the shot can be executed or not in 2 clock cycles.
- The system programming is easier to do, since the processes are decoupled from the concurrency.
- This processor can be programmed at run time, thus it is possible to decrease the size of the matrix in hardware by using spatial and temporal locality.

The difficulty of this implementation is because of the growth of the resources needed by the increase of places and transitions. It implies that is difficult to implement a system for dimensions greater than 32x32 in ZedBoard, to mitigate this difficulty new designs have been proposed and are being worked on, these are: Petri Net Processor with pipeline architecture and support for Hierarchical Petri Nets.

References

[1] J. L. Hennessy, Computer Architecture A Quantitative Approach, Denise E. M. Penrose, 2007.

[2] M. Domeika, Software Development for Embedded Multi-core Systems, 0 Corporate Drive, Suite 400, Burlington, MA 01803, USA: Linacre House, Jordan Hill, Oxford OX2 8DP, UK., 2008.

[3] S. S. B. Sundararajan Sriram, EMBEDDED MULTIPROCESSORS, Scheduling and Synchronization, Boca Raton, 2009.

[4] G. A. Bundell, «An FPGA Implementation of the Petri net Firing Algorithm,» Department of Electrical and Electronic Engineering Information Systems Engineering Research Group, The University of Western Australia, 1997.

[5] H. Murakoshi y M. a. D. G. Sugiyama, «A High Speed Programmable Controller Based on Petrinet,» Faculty of Engineering Yokohama National University, Japan, 1966.

[6] S. Brofferio, «A Petri Net Control Unit for High-speed Modular Signal Processors,» IEEE Transactions on Communications, 1987.

[7] R. Piedrafita Moreno y J. L. Villarroel Salcedo, «Adaptive Petri Nets Implementation. The Execution Time Controller,» de *Workshop on Discrete Event Systems*, Göteborg, Sweden, 2008.

[8] X. Fang y Z. a. Y. Z. Xu, A Study about the Mapping of Process-Processor based on Petri Nets, Anhui University of Science and Technology, 2006.

[9] H. Murakoshi, «Memory Reduction of Fire Unit for Petri Net Controlled Multiprocessor,» *IEEE*, 1991.

[10] M. Wegrzyn, P. Wolanski y M. a. M. J. L. Adamski, «Coloured Petri Net Model of Application Specific Logic Controller Programs,» de *ISIE*, Portugal, 1997.

[11] A. Aybar y A. Iftar, «Deadlock Avoidance Controller Design for Timed Petri Nets Using Stretching,» *IEEE Systems Journal*, vol. 2, nº 2, pp. 178-189, 2008.

[12] H. Murakoshi, «Hardware Architecture for Hierarchical Control of Large Petri Net,» 1993.

[13] G. Izquierdo, Modelado e implementación de sistemas de tiempo real mediante redes de petri con tiempo, Zaragoza, 1999.

[14] F. Bause y P. Kritzinger, Stochastic Petri Nets: An Introduction to the Theory, Vieweg, 2002.

[15] J. L. Peterson, Petri Nets, ACM Computing Surveys: 223-252, 1997.

[16] O. Micolini, M. Pereyra, N. A. Gallia y M. A. Alasia, «Procesador de Petri para la Sincronización de Sistemas Multi-Core Homogéneos,» de *CASE 2012*, Buenos Aires, Argentina, 2012.

[17] Atlys, «Digilent,» 2012. [En línea]. Available: Digilent.com.

[18] Xilinx, «MicroBlaze (UG708),» 2012.

[19] Xilinx, «AXI Interconnect (DS768),» 2012.