Execution of Timed Petri Nets with IP cores

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

Julian Nonino
Laboratorio de Arquitectura de Computadoras
FCEFyN-UNC
Córdoba, Argentina
Email: noninojulian@gmail.com

Carlos Renzo Pisetta
Laboratorio de Arquitectura de Computadoras
FCEFyN-UNC
Córdoba, Argentina
Email: renzopisetta@gmail.com

***Abstract*—In this article, we present a Time Petri Nets Processor which is the evolution of Timed Petri Nets Processor. This processor can be directly programmed using Petri Nets formalism's vectors and matrixes. This processor can leverage the power of Petri Nets for modeling real-time systems and formally verifying 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 by using the IP-core to implement the system is possible to ensure that all properties will be met.

***Index Terms*—Multi-Core, Petri Net, Processor.**

I. Introduction

The computer systems are complex as much its structure as its behavior, even more when they have a great number of states and many combinations of data and input events.

Develop solutions of complex and critical systems for give a solution a real-time systems have problems such as: the inherent complexity of the specification, the coordination of concurrent tasks, the lack of portable algorithms, standardized environments, software and development tools.

And taking into account unambiguous trends in the hardware design, which indicate that one processor may not be able to keep pace with increase performance. Therefore the evolution of the processors is consequence of the greater integration and composition of different types of functionalities integrated into a single processor. Even more today, 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 multicore 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 mainly involves lower execution time, energy consumption, energy density, latency and more bandwidth inter-core communications. 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.

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 Time Petri Nets and to be programmed directly with the vectors and matrixes that define the system and its state.

II. Objetives

A. Main Objective

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

B. Secondary Objectives

The secondary objectives are:

- Briefly describe Time 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 Time 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 [4].

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:

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

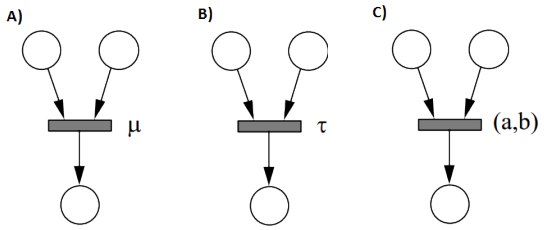

Figure 1. Different ways to intruduce time in Petri Nets

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

1) Generalized Stochastic Petri Nets (GSPNs) [2] 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 $\bullet t$ 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) Time 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].

[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-

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 $\bullet t$ 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 $t\bullet$ with the transition t.

IV. Time Petri Nets

In this nets, each transition has an associated interval $[a, b]$ which represents the possible duration of the activity modeled by the transition.

A. Mathematical Definition

A *Marked Time Petri Net* [4], is mathematically defined as a 8-tuple as follows:

$$PN = \{P, T, I^-, I^+, H, C, m_0, IS\}$$

Where,

- **P** is a non-empty finite set of places.
- **T** is a non-empty finite set of transitions;

$$P \cap T = \oslash$$

- I^+, I^- are the possitive and negative incidence matrices.

$$P \times T \to \mathbb{Z}$$

- **H** is the inhibitors arcs matrix.

$$P \times T \to \{0, 1\}$$

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

$$C \to \mathbb{N}$$

- $\boldsymbol{m_0}$ is the net initial marking.

$$P \to \mathbb{N}$$

- **IS** is the set of static intervals associated with each transition.

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

The ***IS*** function associates to each transition a pair of values representing the minimun and maximum time limits between which the transition may be triggered. In a way that $IS(t) = [min, max]$. Where, t is a transition included in the T set.

As the IS function represents a time interval must meet the following conditions.

- $0 \leq min < \infty$
- $0 \leq max \leq \infty$
- $min \leq max$ if $max \neq \infty$
- $min < max$ if $max = \infty$

The value min is called **Earliest Firing Time (EFT)**. And, the value max is known as **Latest Firing Time (LFT)**.

There are two remarkable interval types:

- *Intervalo puntual*

$$[\alpha, \alpha]$$

  In this case, the sensibilization time is fixed. After the time indicated by α the transition has to be fired. An inmediate firing is represented by $\alpha = 0$.
- *Interval without temporal restriction*

$$[0, \infty]$$

  The transition has no temporal restrictions to be fired, it will be fired sometime after been enabled.

B. Time Petri Nets States

In these Petri Nets, the net state is defined by the marking vector (m) and an *interval* vector that indicates the time stamp of each trasition. Therefore the net state is:

$$S = (m_0, timer)$$

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 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 $t_j(w(p_i, t_j))$. Mathematically:

$$p_i \in \bullet t_i, m(p_j) \geq w(p_i, t_j)$$

If the transition time is zero, $timer_t$ must be enabled to auto-increment with time. The sensitized transitions can be fired in the interval [a, b], and his shot causes a new marking, its mean a change of state.

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 $\delta(m_k, t_j)$, and it is defined as (1):

$$\delta(m_k, t_j) = \begin{cases} m_{k+1}(p_i) = m_k(p_i) - W_{ij} & \forall p_i \in \bullet t_j \\ m_{k+1}(p_i) = m_k(p_i) + W_{ij} & \forall p_i \in t_j \bullet \\ min \leq timer_{(t_j)} \leq max & \\ m_{k+1}(p_i) = m_k(p_i) & \text{in the rest} \\ & \text{of the cases} \end{cases} \quad (1)$$

$$min = a, max = b;$$

$timer_{(t_j)}$ is incremented in every clock cycle after the transition have been sensitized.

D. Interpretation of the firing of transitions in the system

The figure 2 represents a reactive systems that responds to events which come from the environment, it interacs with the environment. Those events are directed to the Time Petri Nets processor.

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

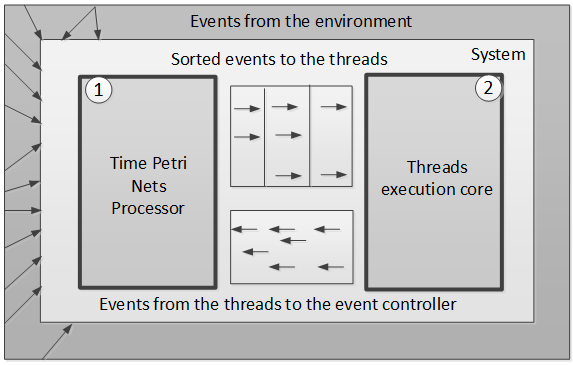

Figure 2. Reactive systems

Module 1 from the Figure 2 receive unsorted events from the environment and from the system itself. After the sorting of the events the Time Petri Nets processor transmits the result to the threads execution cores (module 2) of the system and the propers actions are taken [5].

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 Time Petri Process:

1) The transition must comply with sensitization conditions named in 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 Time Petri Nets processor will execute first the firing with higher priority.

Figure 3 show us how the Time Petri Nets processor is conected in a multicore system.

In case that the firing of the transition can not be resolve, it is queued in the input queue, as shown in Figure 3, until the conditions of the system allow 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.

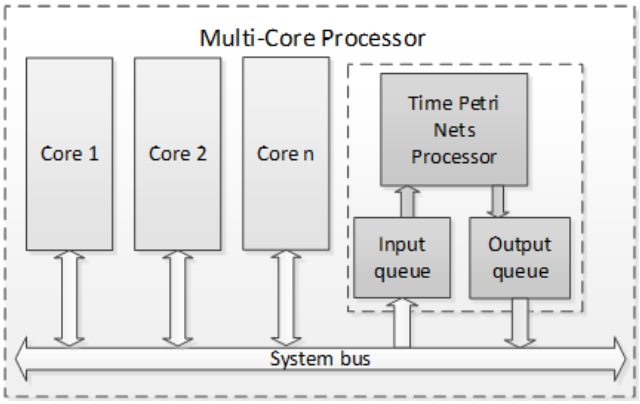

Figure 3. Multicore system with Time Petri Nets processor

V. Time 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 Time Petri processor and stored in the input queue. This queue is FIFO according to each transition, the output of this queue is a binary word which size equals 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 Time Petri Net.

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 the work [6].

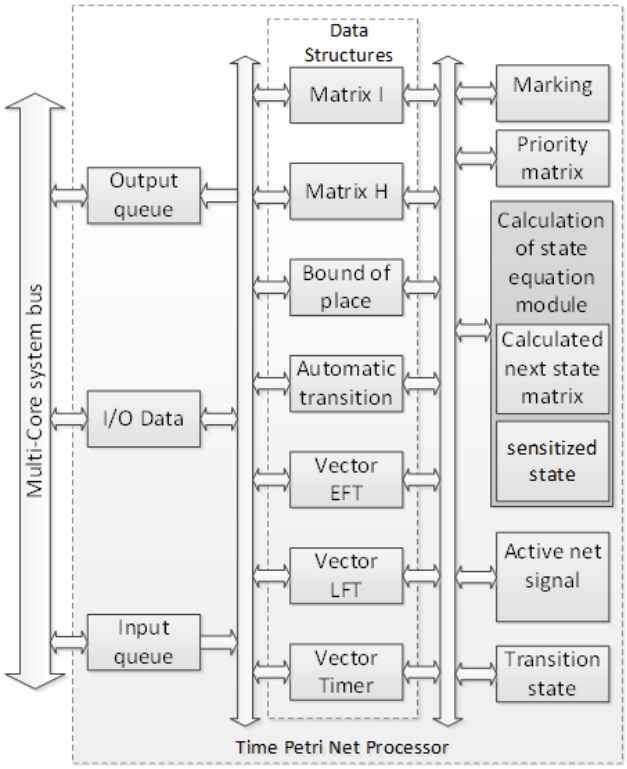

Figure 4. Time Petri Nets Processor

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 time 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 of tokens it [2]can hold.
3) It starts or stops the timer. If a transition t has been sensitized and $Timer_t = 0$ starts $Timer_t$, if $Timer_t \neq 0$ does nothing.
4) Firing of one transition. Transitions that meet:

$$VectorEFT \leq VectorTimer_t \leq VectorLFT$$

The transitions that meet that condition and have received one firing from input queue or the firing pro-

[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.

grammed as automatic (which form a set of available firings).
From that set, the firing with highest priority is selected and the transition is executed.
According the executed transition, the state vector is actualized and the $Timer_t$ is resetted to zero.

5) 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 none $Timer_t$ is higher than LFT_t. 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 [7], cores used are the MicroBlaze v8.40 [8] running an XilKernel v5.01a [9] operating system. Interconnected with Time Petri Processor by AXI bus [10].

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.

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%.

These measurements were performed with times τ zero, in this way the performance is the same that i s obtained in the Petri Net Processor without temporal semantics. This is valid because the transition's time is part of the model, so, it is the same for the processor as for semaphore implementation.

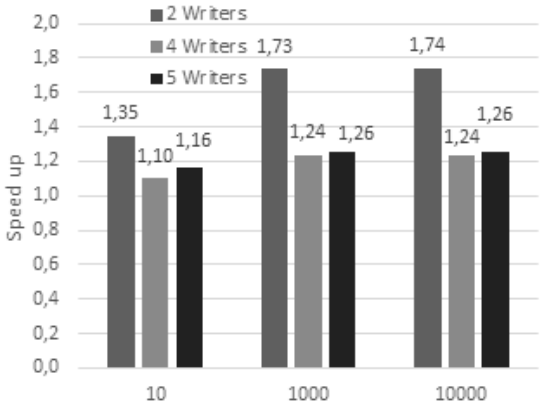

Figure 5. Synchronization Speedup per iteration

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

Taking into account how insignificant the latency is and taking the time as part of the model it is possible to analyze the performance without taking into account the vector Γ.

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

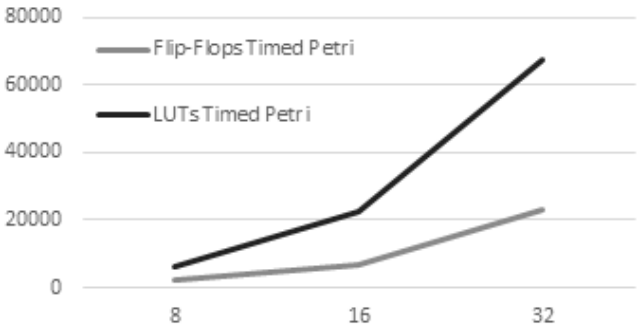

Figure 7. IP Core Growth

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, a Time Petri Processor is developed, which decouples the concurrency from sequential processing, it has the following particularities:

- 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 state equation's matrixes and vectors.
- Multiple are admitted shots simultaneously in the same transition.
- Allows programming priorities 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. This 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 and D. A. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 2007.

[2] M. Domeika, *Software Development for Embedded Multi-core Systems: A Practical Guide Using Embedded Intel Architecture*. Pearson Educacion, 2006.

[3] S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors : scheduling and synchronization*, ser. Signal processing and communications. Boca Raton: CRC Press, 2009. [Online]. Available: http://opac.inria.fr/record=b1128488

[4] F. J. García Izquierdo, "Modelado e implementación de sistemas de tiempo real mediante redes de petri con tiempo," Ph.D. dissertation, Universidad de Zaragoza, 1999.

[5] J. L. Peterson, "Petri nets," *ACM Comput. Surv.*, vol. 9, no. 3, pp. 223–252, Sep. 1977. [Online]. Available: http://doi.acm.org/10.1145/356698.356702

[6] M. M. G. N. A. A. M. A. Micolini, Orlando; Pereyra, "Sistema multicore heterogéneo, sincronizado por un procesador de petri sobre fpga." 2012.

[7] *Atlys Board*.

[8] *MicroBlaze Processor Reference Guide. Embedded Development Kit 14.2 (UG081 v14.2)*.

[9] *OS and Libraries Document Collection (UG643 v14.2)*.

[10] *LogiCORE IP AXI Interconnect v 1.06.a (DS768)*.