IP Core Para Redes de Petri con Tiempo

Ornaldo Micolini[1], Julián Nonino[1] y Carlos Renzo Pisetta[1]

[1]Facultad de Ciencias Exactas Físicas y Naturales
Universidad Nacional de Córdoba, Argentina
omicolini@compuar.com, {noninojulian, renzopisetta}@gmail.com

Abstract. En este trabajo, se presenta un procesador de Redes de Petri con Tiempo, el que es la evolución del Procesador de Petri Temporizado. Este procesador es programado directamente con las matrices y vectores del formalismo de Petri, lo que permite aprovechar el poder de las redes de Petri para modelar sistemas de tiempo real y verificar formalmente sus propiedades, evitando errores de programación al implementar el programa a ejecutar.
Este desarrollo ha sido realizado como un IP-cores y es usado en un sistema Multi-core. De esta manera, es posible realizar la implementación del sistema utilizando este IP-core, lo que asegura las propiedades del modelo realizado con la red de Petri con Tiempo, que verifican los requerimientos del modelo que representa al sistema real, sean cumplido.

Key words: Multi-core, Red de Petri, Procesador

1 Introducción

Los sistemas informáticos son complejos tanto en su estructura como en su comportamiento, más aun cuando tienen un gran número de estados y numerosas combinaciones de datos y eventos de entrada.

Abordar soluciones de sistemas complejos y crítico, para dar solución a sistemas en tiempo real, tiene problemas como: la complejidad inherente de la especificación, la coordinación de tareas concurrentes, la falta de algoritmos portables, entornos estandarizados, software y herramientas de desarrollo.

Y teniendo en cuenta, las tendencias inequívocas en el diseño de hardware, que indican que un solo procesador no puede ser capaz de mantener el ritmo de incrementos de rendimiento. Por lo que la evolución de los procesadores, que es consecuencia de la mayor integración y la composición de distintos tipos de funcionalidades integradas en un único procesador. Más aun, hoy la disponibilidad de transistores ha hecho factible construir en una sola pastilla varios núcleos de procesador que ha resultado en el desarrollo de la tecnología Multi-core [1].

La obtención de rendimiento decreciente del paralelismo a nivel de instrucción (ILP) y el costo del incremento en la frecuencia debido principalmente a las limitaciones de potencia (se sugiere que un 1% de aumento de velocidad de reloj resultados en un aumento de potencia del 3% [2]) ha motivado el uso de los Multi-core.

Por lo cual los procesadores Multi-core son una propuesta para obtener aumento de rendimiento. Lo que se traduce principalmente en menores tiempos de ejecución, consumo ruido, densidad de energía, latencia y más ancho de banda en las comunicaciones inter-core. Si también consideramos a los Multi-core heterogéneos que tienen como ventaja emplear cores especializados, diseñados para tareas específicas. Es decir, optimizado según la necesidad. Estos tienen la capacidad de usar los recursos de hardware disponibles donde el software específicamente lo requiere. [3]

Con el fin de aumentar el desempeño, estos sistemas hacen uso colaborativos de multi-hilos y/o multi-tarea, lo que permite aprovechar los múlti-núcleos. Pero se requiere de más trabajo en el diseño de las aplicaciones, ya que emergen con fuerza la problemática de los sistemas concurrentes.

Por lo que con estos procesadores, la programación paralela es indispensable para la mejora del desempeño del software en todos los segmentos de desarrollo y con más razón en el segmento de sistemas de tiempo real.

Para dar solución a los sistemas reactivos, paralelos y de tiempo real, en relación con los siguientes aspectos:

- Problemas de concurrencia que emergen en la programación paralela, por no ser componible, es decir, no se puede obtener un programa paralelo de la composición directa de dos programas secuenciales.
- Que el hardware de soporte a la implementación de sistemas concurrentes, permitiendo mejorar los algoritmos paralelos.
- Asegurar los requerimientos temporales en los sistemas de tiempo real, es decir, los intervalos mínimos y máximos para la ocurrencia de un evento. Para lo cual el hardware facilite la programación de estas restricciones en forma directa.
- Tareas de codificación, que se requieren para la implementación de un modelo, conducen a errores e incrementan el esfuerzo, por lo que es muy valorable que no exista ninguna tarea entre el modelo y el software a ejecutar.

2 Objetivo

2.1 Objetivo Principal

El objetivo principal de este trabajo es diseñar e implementar un procesador de Redes de Petri con Tiempo, que ejecute la semántica temporal y se programe en forma directa a partir de las ecuaciones de estado del modelo.

2.2 Objetivos Secundarios

Los objetivos secundarios de este trabajo son:

- Describir brevemente las Redes de Petri con Tiempo con el fin de realizar su implementación por hardware.
- Mantener la ejecución de las Redes de Petri ordinarias con parámetros temporales en dos ciclos de reloj.
- Implementar el procesador de Redes de Petri en un IP-core.

3 Redes de Petri con Tiempo

En estas redes, cada transición con tiempo tiene asociado un intervalo de tiempo [a, b] que establece el intervalo de tiempo dentro del cual puede ser disparada la transición, con el fin de homogenizar las definición matemática definimos transiciones inmediatas con límite inferior cero. [4]

3.1 Definición Matemática

Una Red de Petri con Tiempo (TPN) [5] y marcada, se define matemáticamente como una 8-tupla de la siguiente manera:

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

Donde $\{P, T, I^+, I^-, H, C, m_0\}$ es una red de Petri plaza transición marcada con brazos inhibidores y plazas acotadas, y IS es la función estática de intervalos $[a, b]$ asociados a cada transición.

Dónde:

P: es un conjunto finito y no vacío de plazas.

T: es un conjunto finito y no vacío de transiciones, P y T son conjuntos disjuntos

$\boldsymbol{I^+}$, $\boldsymbol{I^-}$: son las matrices de incidencia positiva y negativa. La matriz ***I*** es las diferencias entre $\boldsymbol{I^+}$, $\boldsymbol{I^-}$.

$$PxT \rightarrow Z$$

H: es la matriz de brazos inhibidores.

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

C: es el vector de cota de plaza

$$C \rightarrow N$$

IS: es la función estática de intervalos asociados a cada transición.

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

La función IS asocia a cada transición un par de valores que representan los límites temporales máximo´ y mínimo entre los cuales la transición podrá ser disparada. De manera tal que

$$IS(t) = [min, max] \; \forall \; t \in T$$

Como la función IS representa un intervalo temporal, para cada transición ***t*** sensibilizada se introduces el valor $timer_t$, que se auto incrementa con el tiempo, si la transición esta sensibilizada y se cumplir: $\min \leq timer_t \leq max$ el disparo sea posible.

Estas cotas deben cumplir las siguientes condiciones:

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

Al valor min lo lamamos Earliest Firing Time EFT (Instante de disparo más temprano). Y, al valor max se le llama Latest Firing Time LFT (Instante de disparo más tardío).

Existen dos tipos de intervalos destacables:

- Intervalo puntual $[a, a]$. En este caso, el tiempo de disparo es fijo, después de sensibilización se espera un tiempo a.
  Un disparo inmediato es representado por α = 0 y se comporta como en las Redes de Petri plaza transición.
- Intervalo sin restricción temporal, $[a, \infty]$. Se disparara en algún momento después de sensibilizarse y un tiempo a.

3.2 Estados en una Red de Petri Temporizada

En las Redes de Petri con tiempo, el estado de la red es definido por el vector de marcado m_i y por el vector de valores de intervalos de transición $timer$ de la red, que lleva la cuenta de tiempo de cada transición sensibilizada. Por lo tanto el estado es:

$$E = (mi, timer)$$

3.3 Transición Sensibilizada y Disparo de una Transición

Cundo nos referimos a una transición hay que distinguir las siguientes cuestiones: transición habilitada o sensibilizada, transición no habilitada y disparo de una transición.

En una Red de Petri marcada, con una marca $\boldsymbol{m_k}$, se dice que una transición t_j se encuentra habilitada o sensibilizada si y solo si (sii) todos los lugares del conjunto de plazas $\bullet\, \boldsymbol{t_j}$ de entrada a la transición tienen al menos la cantidad de marcas igual al peso de los arcos ($\boldsymbol{w(p_i, t_j)}$) de entrada a la transición $\boldsymbol{t_j}$, esto es:

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

Si el $timer$ de la transición es cero, se debe habilitar $timer_t$ para que se auto incremente con el tiempo.

Las transiciones sensibilizadas pueden ser disparadas en el intervalo [a, b], y su disparo provoca un nuevo marcado es decir un cambio de estado. La ecuación para calcular el cambio de estado o la nueva marca alcanzada por el disparo de $\boldsymbol{t_j}$ es $\partial\,(m_k, t_j)$, y se define por la siguiente expresión:

$$\partial\,(m_k, timer_t) = \begin{cases} m_{k+1}(p_i) = m_k(p_i) - w_{ij} & ,\forall\, p_i \in t_j \bullet \\ m_{k+1}(p_i) = m_k(p_i) + w_{ji} & ,\forall\, p_i \in t_j \bullet \\ \min \leq timer_t \leq max & \\ m_{k+1}(p_i) = m_k(p_i) & ,en\ el\ resto\ de\ los\ casos; \end{cases}$$

$$min = a, max = \mathrm{b}$$

Donde el $timer_{t_j}$ se incrementa en cada ciclo de reloj mientras la transición se encuentra sensibilizada.

4 Arquitectura y Funcionamiento del Procesador de Petri con Tiempo

El procesador ejecuta la ecuación de cambio de estado resolviendo solo un disparo de una transición a la vez, esto permite resolver todos los casos de disparos, los simples (disparo único) y los disparos múltiples, realizándolos como una secuencia de disparos simples, esto simplifica el hardware.

Las disparos son transmitidos por los hilos que se ejecutan en los cores a través del bus del sistema, según las solicitudes emergentes del sistema que se está ejecutando. Esto disparos son recibidos por el Procesador de Petri con Tiempo y almacenado en la cola de disparos de entrada. Existe una cola FIFO por cada transición, la salida de este conjunto de colas es una palabra con tamaño igual a la cantidad de transiciones, la cual tiene unos en las posiciones correspondientes a las transiciones con disparos solicitados, el orden del bit en la palabra es igual al número de la transición que solicita el disparo. Los bits, que se corresponden con las transiciones que no tiene disparo solicitado, son cero, es decir no hay solicitud de disparo.

La cola de salida tiene una estructura similar, pero comunica los disparos resueltos a los hilos.

En la Fig. 1 se muestran los distintos módulos que componen el procesador, resaltando las principales diferencias con versiones anteriores. [6]

El módulo de I/O Datos gestiona el acceso de los cores a las matrices y vectores que programan el sistema.

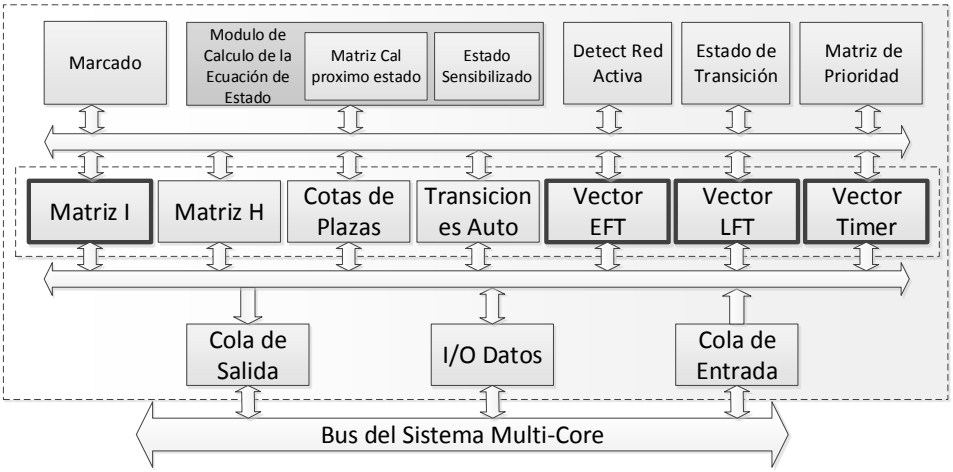

Fig. 1. Procesador de Petri con Tiempo.

El programa del sistema son las matrices y vectores descriptas en la ecuación de estado, esto permite programar el procesador en forma directa a partir de la Red de Petri con Tiempo.

Aquí se han agregado la matriz de Brazos Inhibidores y el vector de Cota de Plaza que no figuran en la ecuación de estado presentada en este trabajo, pero son los mismos que en el Procesador de Petri presentado en otros trabajos [7].

La responsabilidad del Módulo de Cálculo de la Ecuación de estado es la siguiente:

1. Calcular el nuevo estado que resultaría por disparar solamente una transiciones una vez, por lo que resultan tantos vectores de estados calculados como transiciones, y se almacenan. Esto se realiza en paralelo sumando al estado actual a cada columna de I y almacenando todos los vectores resultantes, los que serán evaluados para determinar si son los posibles nuevos estado. Esta operación es realizada siempre que cambia el estado del procesador, vector Marcado.
2. Determinar que transición esta sensibilizada. Se toma todos los vectores calculados en 1 y se verifica que se cumpla que ninguna plaza tenga marcado negativo y tampoco supere la cota de plaza, estas son las transiciones sensibilizadas.
3. Se arranca o para los Timer_t . Si en una transición sensibilizada $\text{Timer}_t = 0$ se arranca Timer_t y si $\text{Timer}_t \neq 0$ no se hace nada.
4. Disparo de una transición. Las transiciones que cumplen con:

$$Vector\ EFT \ \leq \ \text{Vector Timer}_t \ \leq Vector\ LFT$$

   Las transiciones que cumplen con esta condición y han recibido por la cola de entrada un disparo o el disparo están programado como automático, conforman un conjunto de disparos posibles

   De este conjunto se selecciona el de mayor prioridad y se ejecuta la transición.

   Según la transición ejecutada se actualiza el vector de estado, y se pone Timer_t a cero.

5. Se ejecuta como un ciclo continuo los pazos 1, 2, 3 y 4.

El sistema posee una unidad que detecta cuando ninguna transición esta sensibilizada y Vector Timer supera el tiempo máximo; esta condición genera una interrupción que comunica que el sistema ha finalizado o esta interbloqueado, esta característica es de suma utilidad para verificar el diseño e implementación del sistema.

La Tabla 1 muestra las diferencias significativas, desde el punto de vista de la ejecución de las distintas semánticas, estas son:

Tabla 1. Comparación entre Semánticas Temporales.

		Con Tiempo	Temporizada
1	Interrumpible	Si	No
2	Representa las dos semánticas	Si	No
3	Matrices usadas	I	I+, I-
4	Permite contener subredes	No	Si

De este cuadro se desprenden las siguientes observaciones:

1. Siendo que las TPN son interrumpibles y las Redes de Petri Temporizadas (TdPN) no lo son, para el caso de múltiples disparos y transiciones en conflicto, un TPN lo resuelve según el intervalo de tiempo; en cambio una TdPN lo hace explícitamente en la matriz de prioridad. Esto hace más complejo el modelado con TdPN e indispensable incluir en el procesador una matriz de prioridades.

Dado que la mecánica de ejecución de las TdPN requiere de un estado más para no ser interrumpibles los tokens son retirados inmediatamente de la plaza y no pueden ser solicitados por otra transición.

2. Dada una red con TPN, una transición, que por semántica es interrumpible, puede transformarse en una no interrumpible modificando la red. Esto se logra encerrando con una transición inmediata la transición temporiza. Lo que tiene como impacto un incremente de una plaza y una transición adicional por cada transición no interrumpible.
3. Para realizar el cálculo de un nuevo estado las TPN lo hacen con una matriz de enteros con signo mientras que las TdPN lo realizan con dos matrices de enteros sin signo; por lo cual debemos analizar dos casos:
    a. Si los pesos de los arcos son uno:
        i. Las TPN requieren de una matriz con 2 bit por elemento.
        ii. Las TdPN requieren de dos matrices binarias.
    b. Si los pesos de los arcos son uno o mayor a uno:
        i. Las TPN requieren de una matriz de enteros con signo.
        ii. Las TdPN requieren dos matrices de enteros sin signo.

    En el primer caso los recursos utilizados son similares. Por lo que la selección de uno u otro procesador depende de la semántica a utilizar. Mientras que, en el segundo caso los recursos utilizados por las TdPN son mayores. La ventaja de una con respecto a la otra en cuestión de recursos está determinada por incremento de la matriz de incidencia dada por la conversión de las transiciones Time a su equivalente no interrumpibles.
4. El procesador que implementa la semántica TdPN utiliza dos estados para realizar el cálculo de los toquen que entran de una transición y los que salen de esta. Esta diferenciación de estados nos permite insertar una nueva red de Petri entre los dos estados de una transición, lo que posibilita que el procesador puede ser extendido a redes de Petri jerárquicas; ya sea haciendo uso de la semántica TdPN o de las redes de Petri ordinarias. Esto en la actualidad es motivo de una nueva investigación.

Las dos semánticas son investigadas, puesto que las TPN requieren de menos recursos para resolver problemas no interrumpibles (que son los más habituales). Mientras que las TdPN presentan potencial de mejora al permitir construir redes de Petri jerárquicas. [8].

5 Análisis de Rendimiento

La implementación de sistema ha sido realizada en una plataforma Atlys™ Spartan-6, los cores utilizados son los MicroBlaze ver8.40 [9] que ejecuta un Sistema Operativo XilKernel ver5.01a. Interconectado con el Procesador de Petri Temporizado por un bus AXI [10].

Para comprobar correcto funcionamiento del IP Core y analizar los tiempos de sincronización, se realizaron mediciones para distinto número de iteraciones y numero

de hilos tratando de acceder a una variable compartida en exclusión mutua. Luego se compararon el Procesador de Petri con una implementación utilizando semáforos, ambos resolviendo un mismo problema. La elección de este segundo método de sincronización se basa en que son el mecanismo más ligero para realizar éstas tareas.

A partir de estas mediciones se calculó el Speedup, los resultados se muestran en la Fig. 2, se puede observar que, para todos los casos, el procesador de Petri es en promedio es entre un 15% y un 30% más rápido que el uso de semáforos para resolver el problema de sincronizar múltiples hilos que desean escribir sobre una variable compartida e incluso, se alcanzan picos de hasta un 70%.

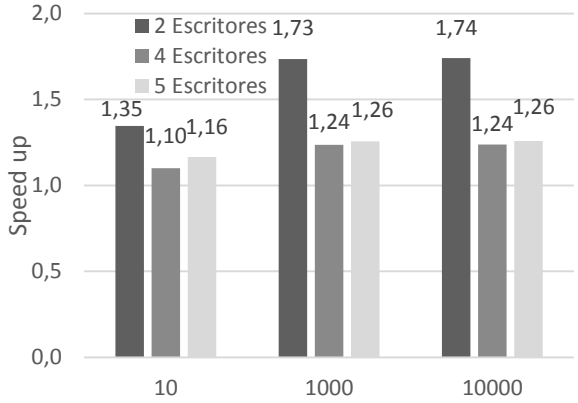

Fig. 2. Tiempos de sincronización por iteración

Estas mediciones se realizaron con tiempos $EFT\ y\ LFT$ cero, de manera que el rendimiento es el mismo obtenido en el procesador de Redes de Petri sin la semántica temporal. Esto es válido ya que el tiempo de una transición es parte del modelo, es decir, es el mismo para el procesador de Petri como para la implementación con semáforos y el propósito es medir únicamente los tiempos de sincronización.

Además, como se observa en la Fig. 3, el procesador necesita únicamente un semi-ciclo de reloj, desde que el contador alcanza el valor EFT hasta que el disparo se coloca en la cola de salida. La demora introducida es despreciable en relación con el tiempo que tiene un δt de un ciclo de reloj.

Teniendo en cuenta lo despreciable de la latencia y tomando el tiempo como parte del modelo es posible analizar el rendimiento sin tener en cuenta los vectores EFT y LFT.

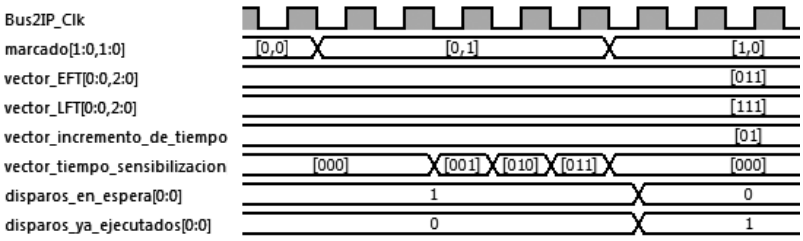

Fig. 3. Ejecución en hardware.

6 Crecimiento del IP Core

Se analizó el crecimiento del procesador en función de los parámetros que posee. Para esto se generaron procesadores de 8x8, 16x16, 32x32, 48x48 y 64x64 (Plazas por Transición) con capacidad de 7 bits por plaza y elementos de tiempo de 48 bits y se graficaron los resultados, los que se pueden observar en la Fig. 4.

Se observa que el crecimiento del IP Core no es algo para despreciar, puesto que la cantidad de elementos empleados crese rápidamente con el producto de las Plazas por las Transiciones.

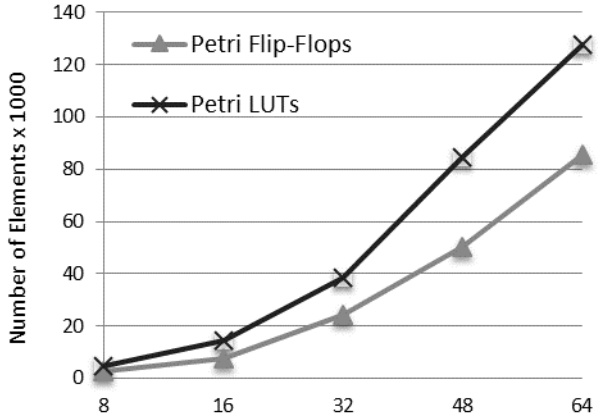

Fig. 4. Crecimiento del IP Core

Por otra parte, ya que es posible sintetizar un procesador para cada semántica es deseable determinar y comparar el consumo de recursos para cada uno. La Fig. 5 muestra la comparación del crecimiento entre las distintas implementaciones.

Se puede observar que ambos procesadores utilizan aproximadamente la misma cantidad de Flip-Flops pero la implementación para redes temporizadas utiliza un 90% mas LUTs para el mismo número de plazas y transiciones.

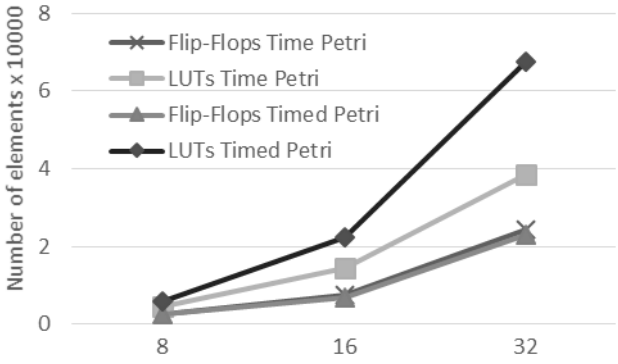

Fig. 5. Recursos usados por distintas semánticas.

7 Conclusión y Aportes

En el presente trabajo, se desarrolla el Procesador de Petri con Tiempo, que permite desacoplar los la concurrencia del procesamiento secuencial. Teniendo en cuanta que el Procesador de Petri con Tiempo permite utilizar Redes de Petri Temporizadas, este procesador puede remplazar a su predecesor y preserva sus particularidades.

El modelo de Petri es adecuado para implementar, validar y verificar un sistema paralelo con concurrencia, este tiene una representación algebraica que este procesador usa como el código ejecutable. Las ventajas de este procesador son la disminución de:

- Esfuerzo de programación, la ecuación de estado es ejecutada directamente en el procesador, y no se requiere programación adicional.
- El gap entre las restricciones temporales y sus programaciones. Puesto que se trata de los vectores temporales propios de la semántica usada por el procesador.

Referencias

1. Hennessy, John L. Computer Architecture A Quantitative Approach.: Denise E. M. Penrose, 2007.
2. Domeika, M. Software Development for Embedded Multi-core Systems. 0 Corporate Drive, Suite 400, Burlington, MA 01803, USA : Linacre House, Jordan Hill, Oxford , UK., 2008.
3. Sundararajan Sriram, S. S. B. EMBEDDED MULTIPROCESSORS, Scheduling and Synchronization. Boca Raton, 2009.
4. Ramachandani, C. Analysis of Asynchronous Concurrent Systems by Timed Petri Nets. Cambribge, Massachussets : Massachussets Institute of Technology, 1974.
5. Izquierdo, García. Modelado e implementación de sistemas de tiempo real mediante redes de petri con tiempo. Zaragoza, 1999.
6. IP Core for Timed Petri Nets. Micolini, Orlando, Nonino, Nulián and Pisetta, Carlos R. Buenos Aires, Argentina: s.n., 2013. CASE 2013,unpublished
7. Procesador de Petri para la Sincronización de Sistemas Multi-Core Homogéneos. Micolini, Orlando, y otros. Buenos Aires, Argentina : s.n., 2012. CASE 2012.
8. Jensen , Kurt y Kristensen, Lars M. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. New York : Springer, 2009 .
9. Xilinx. MicroBlaze (UG708). 2012.
10. AXI Interconnect (DS768). 2012.