Optimización de Soft Core parametrizable con procesamiento de Redes de Petri

Julián Nonino, *Member, IEEE,* Carlos Renzo Pisetta, *Member, IEEE,* and Orlando Micolini, *Member, IEEE*

***Resumen*—Este desarrollo, mediante el empleo de Redes de Petri, planteó una solución a problemas de sincronización en arquitecturas multicore. El implemento de un procesador reprogramable en tiempo de ejecución que utilice estos formalismos en hardware con uso de FPGA, permitió diseñar modificaciones en la arquitectura y expandir funcionalidades del IP Core, las cuales han sido descriptas en este trabajo.**

***Index Terms*—Petri Nets, MultiCore, FPGA, IP Core.**

I. Introducción

El presente trabajo inició con el estudio realizado sobre el Procesador de Petri (PP) descripto en trabajos previos [**?**].

Luego de evaluar sus capacidades y funcionamiento se diseñaron distintas alternativas para mejorar sus prestaciones. En las siguientes secciones, se abordaron los distintos objetivos planteados y descripción de las soluciones implementadas.

I-A. Objetivos

- Implementar el Procesador de Petri parametrizable usando Verilogo como HDL.
- Determinar cantidad máxima de tokens por plaza (Plazas Acotadas).
- Capacidad para contemplar transiciones automáticas.
- Cambio de colas de entrada y salida de disparos.
- Generar soporte para interrupciones enmascarables.
- Analisis de crecimiento del tamaño del procesador.

II. Marco Teorico

II-A. Redes de Petri

Una Red de Petri o Petri Net es un modelo gráfico, formal y abstracto útil para describir y analizar el flujo de información. Conforma una herramienta matemática aplicable, especialmente, a los sistemas paralelos que requieran simulación y modelado de la concurrencia en los recursos compartidos. Existen trabajos previos donde se ha demostrado que es posible implementar sus formalismos en hardware [**?**] [**?**]

Las Redes de Petri constan de cuatro componentes fundamentales.Fig**??**

- Plazas: Permiten representar el estado del sistema. Podrían definirse como variables de estado que toman valores enteros (cantidad de tokens). Se representan con un círculo.
- Transiciones: Representan el conjunto de sucesos que hacen que el estado del sistema cambie. Son representadas con un rectángulo.
- Arcos: Los arcos indican las conexiones entre lugares y transiciones. Nunca unen dos lugares o dos transiciones en forma sucesiva. Pueden entrar o salir varios arcos de una misma transición o de un mismo lugar. Los arcos tienen asociado un "peso" que indica la cantidad de tokens que se consumen o generan al atravesarlo. El disparo de una transición retira tantos tokens de cada uno de sus lugares de entrada como lo indican los pesos de los arcos conectores y añade los tokens a sus lugares de salida como lo indican los pesos de los arcos de salida.
- Tokens: Los tokens representan el valor específico de la condición o estado. Son graficados como un punto negro o un número natural o cero dentro de una plaza.

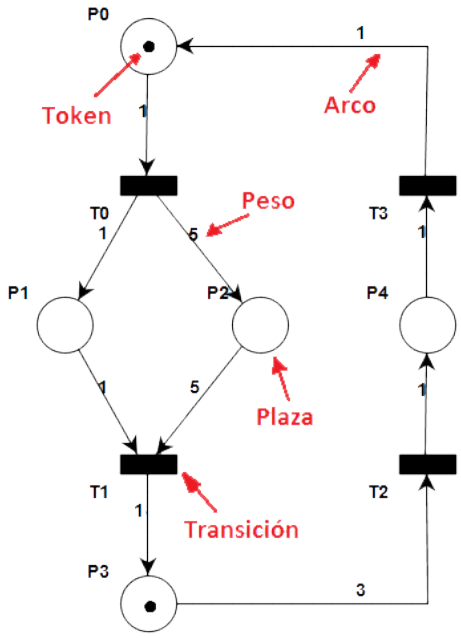


Figura 1. Red de Petri

II-A1. Estructura de una Red de Petri: Una Red de Petri Marcada queda definida como una 5-tupla de la siguiente manera:

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

Donde:

- $\boldsymbol{P = \{p_1, p_2, p_3 \dots p_m\}}$, conjunto de m lugares con m finito y distinto de cero.
- $\boldsymbol{T = \{t_1, t_2, t_3 \dots t_n\}}$, conjunto de n transiciones con n finito y distinto de cero.
- $\boldsymbol{I}$, matriz de incidencia. Esta matriz es de dimensiones $m \times n$, y representa los pesos de los arcos, siendo sus valores positivos cuando el arco va desde una transición hacia una plaza; y negativos, los inversos. Así mismo,

para representar la estructura, esta matriz de incidencia debe separarse en dos:

- $\boldsymbol{I-}$, matriz de incidencia negativa. Esta matriz es de dimensiones $m \times n$ representa los pesos de los arcos que ingresan desde los lugares de P a las transiciones de T.
- $\boldsymbol{I+}$, matriz de incidencia positiva. Esta matriz es de dimensiones $m \times n$ representa los pesos de los arcos que salen desde las transiciones de T hacia los lugares de P.

- $\boldsymbol{m_0}$ es el marcado inicial de la red, un vector de asignación de tokens a las plazas de la red, de esta forma se define la configuración inicial de los tokens de la red. Por ejemplo, puede definirse el marcado de una plaza como $m(i)$, lo que indica la cantidad de tokens ubicados en la plaza "i".

II-A2. Ejecución de una Red de Petri: El comportamiento dinámico de una Red de Petri esta definido por la sensibilización y el disparo de sus transiciones.

- Transición sensibilizada: Se dice que una transición t_j está sensibilizada si y solo si todas las plazas de entrada a la transición tienen una cantidad de tokens igual o mayor al peso indicado por el arco que la une con la transición. Formalmente:

- Disparo de una transición: El disparo de una transición es lo que provoca que el estado (marcado) de una Red de Petri cambie.

II-A3. Ecuación de estado de una Red de Petri: La ecuación de estado determina el estado de la Red de Petri a cada instante, queda definida a partir de la matriz de incidencia y un vector de disparo que indica la transición o transiciones que deben ser disparadas.

$$m_{k+1} = m_k + I \times \delta_k$$

Donde m_{k+1} es el marcado futuro.

II-A4. Plazas Acotadas: Un lugar p_i se dice que es acotado si existe un número $k \in \mathbb{N}$ tal que, para todo el conjunto de marcados alcanzables desde m_0, el número de marcas en p_i no es más grande que k (p_i se dice que es k-acotado). De esta manera una transición ademas de lo visto anteriormente, debe cumplir que ninguna plaza de salida supere su cota maxima k luego de darse el disparo. De ser superado con un determinado disparo, este no se puede llevar a cabo.

III. DESARROLLO

Basados en los diseños previos se desarrolló un IP Core con las mismas funcionalidades que su predecesor realizado en VHDL utilizando Verilog. Esto fue motivado debido a las sintaxis semejantes a C y su capacidad de expresividad. Este diseño permite superar algunos limitantes del procesador de Petri original en cuanto a la parametrizacion. De esta manera se puede sintetizar procesadores de petri a medida, variando el tamaño máximo de las estructuras de datos internas.

III-A. Semantica

La semantica que posibilitó esta implementacion relaciona el marcado con el estado del sistema, mientras que los disparos de una transición se asocian con situaciones que desencadenan un cambio de estado en el sistema, como son: la necesidad de un recurso, sincronización. Por lo tanto, se hacen estas estas peticiones, desde el software, al procesador de petri y éste responde con un disparo ejecutado a la salida cuando esta accion es posible.

III-B. Algoritmo de Ejecución

Basados en la teoria descrita, se creó un algoritmo de ejecución de disparos en una Red de Petri, que se describe a continuación. Es sintetizable en hardware y requiere únicamente 2 ciclos de reloj para ejecutar todos los pasos, y a la vez permita un diseño parametrizable en cuanto al tamaño y cantidad de elementos que soporte.

1. Espera de disparo en Cola de entrada de disparos.
2. Llegado el disparo, se calcula un vector binario de longitud cantidad de transiciones con un único 1 en el lugar correspondiente al número de disparo, en función al número de transición que contenga. Este vector se utiliza para incrementar los contadores de disparos. Son considerados disparos en espera.
3. Se calcula todos los posibles resultados para todos los disparos de la red y se confecciona una matriz resultado donde cada columna C_i representa el nuevo marcado, si la transición T_i se disparara.
4. Se crea una matriz de signos auxiliar con los signos correspondientes a cada elemento de la matriz resultado.
5. Se crea una matriz de inhibición auxiliar en función del marcado actual y la Matriz de Inhibición, lo que determina las plazas con arcos inhibidores que tienen tokens.
6. Se crea una matriz de cotas en función de la matriz resultados y las cotas de las plazas, lo cual determina cuál fue superada para cada plaza por cada posible resultado.
7. Se crea un vector en el cual cada elemento, corresponde a una columna de la matriz de signos y determina si tal transición es o no posible en función si algún elemento de su resultado es negativo.
8. Se crea un vector en el cual cada elemento, corresponde a una columna de la matriz de inhibición y determina las transiciones que no pueden ser disparada debido a arcos inhibidores.
9. Se crea un vector en el cual cada elemento, corresponde a una columna de la matriz de cotas el cual determina qué transiciones no pueden ser disparadas puesto que, de hacerlo, superarían las cotas de las plazas.
10. En función a los vectores creados en los puntos 7, 8 y 9, se crea un vector que determina los disparos posibles.

11. Para determinar las transiciones a disparar, se unen los disparos pendientes y las transiciones automáticas. Luego, en función de los disparos posibles y que la cola de disparos de salida no esté llena, se actualiza el marcado actual a partir del resultado obtenido en el punto 3, considerando la mayor prioridad al índice de mayor valor.
12. Se incrementa contador de cola de salida correspondiente a transición ejecutada.

III-C. Matriz de Prioridades

La forma en la que esta implementado el IPCore posee una prioridad estática donde un disparo T_i tiene un prioridad mayor que un disparo T_j, siempre que $j < i$. Para cambiar esto se utiliza la matriz de prioridades, la cual permite tener distintos ordenes de prioridad en función del marcado actual del sistema. De esta manera se puede definir para los casos necesario una prioridad especifica, y una por defecto para cualquier otro marcado. Actualmente, debido la matriz y mecanismos para darle soporte, ocupan un tamaño considerable por lo cual se siguen investigando nuevos diseños para implementarla.

III-D. Plazas Acotadas

Las plazas acotadas no solo representan una propiedad del sistema que restringe el valor máximo de tokens en una plaza. Si bien este valor de por si ya esta limitado por la implementación en hardware con un número finito de bits es posible limitarlas aún más sin necesidad de alterar el hardware. La implementación en hardware verifica que los posibles marcados futuros no supere las cotas establecidas y dado el caso impedir que una transición sea disparada.

III-E. Colas de Disparos

Las colas de disparos son dos: una, para los disparos de entrada que esperan por ser ejecutados en cuanto la red lo permita; y la otra, para los disparos de salida que ya han sido ejecutados y esperan a que el proceso que solicitó su ejecución los extraiga.

La solucion más natural para este tipo de problemas es una cola FIFO la cual mantiene el orden de los disparos, con el uso de punteros para determinar el estado de la pila y donde almacenar el disparo antes de poder ser procesado. Realizar este tipo de colas en hardware es dificultosa y se encuentra muy limitada en la cantidad de disparos en espera que puede almacenar y el tiempo que demanda la operaciones. Por esto se desarrolló una implementacion que se ajuste más a esta aplicacion, estas nuevas colas denominadas contadores de disparo fueron la solucion implementada.

Estas colas se realizan con un contador por cada transición, el cual indica cuántos disparos pendientes o ya ejecutados existen para dicha transición. Además, existe un vector binario que indica si la cola de una determina transición esta vacía o tiene algún valor (Figura **??**). Esta implementación obedece a la necesidad de insertar o extraer los disparos en un ciclo de reloj y además, poder contemplar todas las solicitudes de disparos de las distintas transiciones en paralelo. Cada una de las posiciones del vector de estado de la cola indican si esta vacío o no. Esto se hace de la siguiente manera (Figura **??**):

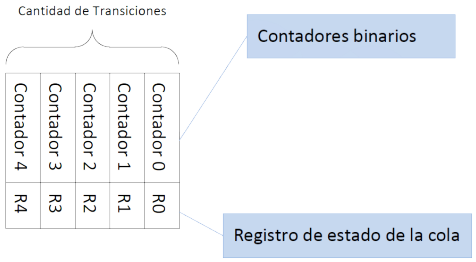


Figura 2. Esquema de las colas de disparos

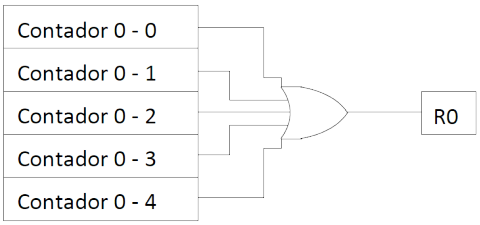


Figura 3. Registro que indica si las colas de disparos están vacías o no

Un *uno* en este vector indica que la cola contiene al menos un elemento. Además, se implementa otro registro de estado que indica si la cola esta llena o no. Este registro se implementó de la siguiente manera. Un *uno* en este vector indica que la cola esta llena y no puede recibir más elementos (Figura **??**).

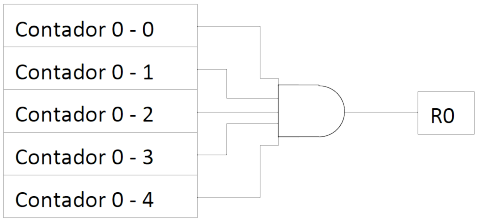


Figura 4. Registro que indica si las colas de disparos están llenas o no

Al realizar las colas de esta manera, se puede almacenar más disparo por transición en un espacio menor y lograr que un disparo necesite 2 ciclos de reloj: desde que llega al procesador de petri hasta que sale, aparece en la cola de salida.

III-F. Transiciones automáticas

Este tipo de transiciones son ejecutadas sin la necesidad de ser solicitadas. Se instancian como un vector binario donde cada elemento representa una transición . Un *uno* en algún elemento indica que la trancicion de igual índice es automática, es decir, al momento que se este disparo es posible se disparara. Su ejecución se realiza con igual prioridad que los disparos pedidos, de manera tal que implique únicamente unirlos en un solo dato antes de resolver los disparos (Figura **??**).

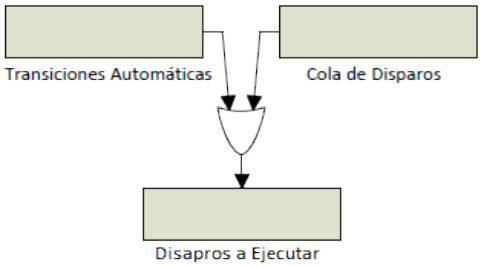


Figura 5. Registro que indica si las colas de disparos están llenas o no

III-G. Tranciciones sin notificacion

Durante el modelado surjen algunos disparos los cuales son necesarios para la evolucion del sistema pero innecesarios para que el programa espere su resolucion. Por esto se desarrolló un módulo que controla los disparos ya ejecutados, antes de ser puestos en cola de salida.

III-H. Manejo de Interrupciones

Una interrupción es un mecanismo por el cual un evento -en éste caso, un disparo ejecutado- puede alterar la ejecución de un programa. De esta manera, un proceso puede continuar su ejecución hasta que el Procesador de Redes de Petri le notifique que el disparo por el cual esperaba, ha sido ejecutado.

La arquitectura del procesador de Redes de Petri, al incorporar el manejo de interrupciones, se ve afectada por el agregado de tres elementos.

- Un vector ***Máscara de Interrupción***, cuyo objetivo es determinar cuáles son las transiciones que al dispararse, generarán interrupciones.
- Un componente generador de interrupciones que contiene la lógica encargada de determinar en qué momento generar la señal la interrupción y de especificar como será dicha señal.
- Un puerto físico por el cual enviar la señal de interrupción.

La figura **??** muestra un diagrama de componentes del módulo *Generador de Interrupciones*. La siguiente sección explicará funcionamiento de este módulo y además, el funcionamiento del sistema en su conjunto cuando el procesador de Redes de Petri tiene la capacidad de interrumpir.

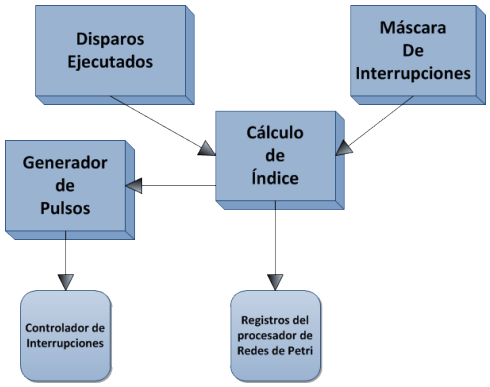


Figura 6. Diagrama de componentes del Generador de Interrupciones

Básicamente, el generador de interrupciones verifica si alguna de las transiciones que están habilitadas para producir interrupciones, se ha ejecutado. Cuando esto sucede, genera una señal en *1* de un ciclo de duración o cambio de nivel, según sea necesario para trabajo por flanco o nivel. Además, determina el índice de dicha transición.

Para limitar el número de transiciones que al ser disparas generan una interrupción, se utiliza el vector de máscara de interrupciones que determina cuáles transiciones generan interrupción y cuáles no. Para que el disparo de una transición genere una interrupción, el bit correspondiente en el vector de máscara de interrupciones debe estar en *1*, de lo contrario se encuentra enmascarada y no generará interrupción alguna. Por lo tanto, para estas transiciones, es tarea del usuario preguntar si ya ha sido ejecutada periódicamente o cuando lo crea necesario.

III-I. Lectura en con decremento

Se creo un registro el cual muestra todos los disparos ejecutados y luego de ser leido decrementa en uno todos las colas de salida que tenian al menos un disparo en ellas. Este registro permite un nuevo mecamismo de uso, en el cual el programa obtiene todos los disparos realizando una unica lectura y luego administrarlos.

IV. Análisis de Rendimiento

Una vez terminado el IP Core para comprobar su correcto funcionamiento y ver si éste realmente tiene una mejora en los tiempos de sincronización, se realizaron mediciones para distinto número de iteraciones, tanto para el Procesador de Petro como para Semáforos. La elección de este segundo método de sincronización se basa en que son el mecanismo más ligero para realziar éstas tareas. A partir de estas mediciones se realizó la gráfica de la Figura **??**

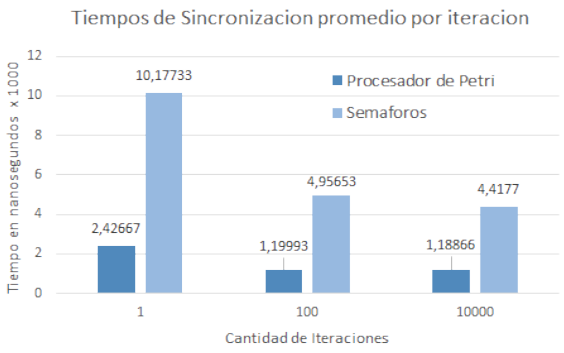


Figura 7. Tiempos de sincronización procesador de Redes de Petri vs. Semáforos

Se puede observar que, para todos los casos, el procesador de Petri es aproximadamente cuatro veces más rapido que los Semáforos.

V. Crecimiento del IP Core

Una vez visto el correcto funcionamiento del IP Core, se evaluó el crecimiento del procesador en función de los

parámetros que posee. Para esto se generaron procesadores de 8x8, 16x16 y 32x32, con capacidad de 7 bits por plaza y se graficaron los valores que se pueden observar en la Figura **??**.

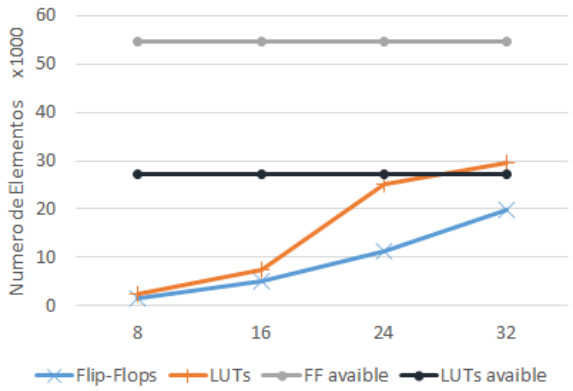


Figura 8. Diferentes implementaciones del procesador de Redes de Petri

Es posible ver que el crecimiento del IP Core no es algo para despreciar, puesto que se incrementa rápidamente a medida que aumentamos el tamaño de elementos soportados. Se observa también que si bien la pendiente aumenta, éstos incrementos son cada vez menores, tendiendo a linealizar para los IP Cores más grandes.

Tambien podemos observar los elementos disponibles que tenemos en una determinada FPGA, para este caso una Spartan6 de Xilinx. Vemos que desarrollar una implementacion de 32x32 es inalcansable, en este caso es necesario pasar a una con mas recursos.

VI. Conclusiones

En éste trabajo se presentó un nuevo procesador de Petri y se realizó un IP Core parametrizable que verifica y resuelve una ejecución en dos ciclos desde que entra en la cola de entrada, hasta que es depositado en la cola de salida.

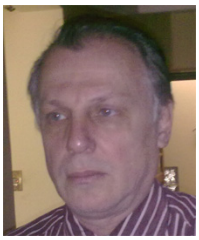
La nueva capacidad de interrumpir permite un nuevo método de espera no activo, sin eliminar el método por consulta que utilizaba el diseño anterior. Las transiciones automaticas permite, la evolución del sistema sin la necesidad de pedir un disparo desde el exterior, y las cotas permite acotar cada plaza a un número determinado de tokens simplemente cargando los datos desde el software, sin la necesidad de modificar el HDL.

Acknowledgment

The authors would like to thank...

Julián Nonino (M'2011) nació en Rio Cuarto, Córdoba, Argentina el día 16 de octubre de 1988. En el año 2012 obtuvo el título de grado Ingeniero en Computación de la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de Córdoba. Actualmente se desempeña como Ingeniero de Software en Motorola Mobility of Argentina S.A. En el ambito universitario, precisamente en la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de Córdoba, colabora en las cátedras Ingeniería de Software y Gestión de la Calidad de Software. También, en el Laboratorio de Arquitectura de Computadoras de la misma universidad, realiza actividades relacionadas con el diseño e implementación de software y hardware optimizado para sistemas de computación en paralelo basados en Redes de Petri.

Carlos Renzo Pisetta (M'2011) nació en Córdoba, Argentina el día 07 de junio de 1989. En el año 2012 obtuvo el título de grado Ingeniero en Computación de la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de Córdoba. Actualmente se desempeña como investigador en el Laboratorio de Arquitectura de Computadoras en la UNC(FCEFyN) abocado al diseño e implementación de software y hardware optimizado para sistemas de computación en paralelo basados en Redes de Petri.



Orlando Micolini Grado en Ingeniero Electricista Electrónico, año 1981, de la UNC (FCEFyN) Argentina. Postgrado Especialista en Telecomunicaciones, año 2002, de la UNC (FCEFyN) Argentina. Magíster en Ciencias de la Ingeniería, año 2002, de la UNC (FCEFyN) Argentina. Director de SCS S.R.L, desde 1991 a 2008. Director de la Carrera de Ingeniería en Computación en la UNC (FCEFyN) Argentina (2004-actualidad). Titular de la asignatura Arquitectura de Computadoras en la UNC (FCEFyN) Argentina (1996-actualidad). Actualmente está trabajando, realizando el doctorado de Ciencias Informática en la Universidad Nacional de la Plata, en sistemas multi-core heterogéneos con Redes de Petri.