Procesador de Petri Modular para Sistemas Embebidos

Dr. Orlando Micolini, Ing. Emiliano N. Daniele, Ing. Luis O. Ventre, Geol. Marcelo Cebollada y Verdague, Ing. Maximiliano Eschoyez

Laboratorio de Arquitectura de Computadoras (LAC) FCEFyN
Universidad Nacional de Córdoba
{orlando.micolini, emiliano.daniele, luis.ventre, marcelo.cebollada.y.verdague, maximiliano.eschoyez}@unc.edu.ar

Abstract. Los Sistemas embebidos, concurrentes y reactivos ejecutan algoritmos con restricciones según los requerimientos de la implementación. Para implementar estos sistemas mediante el uso de hardware y software es posible usar un Procesador de Petri. Desacoplando la lógica y la política de las acciones del sistema se mejora la mantenibilidad y se facilita la validación. Para lograr esto se integra el procesador de Petri con otros procesadores tradicionales, conformando un sistema heterogéneo, lo que agrega la capacidad de verificar el sistema con los formalismos matemáticos del modelo empleado en las redes de Petri. En este artículo se expone una arquitectura modular del Procesador de Petri y la incorporación de colas programables para mejorar la mantenibilidad, el reúso de los módulos y extender su semántica.

Keywords: Procesador de Petri, Redes de Petri, FPGA, IP Core, Procesadores Heterogéneos

1 INTRODUCCIÓN

Los procesadores heterogéneos agregan capacidades específicas no presentes en los procesadores homogéneos. Los sistemas embebidos, concurrentes y reactivos tienen que cumplir requerimientos no funcionales específicos[1]. Estos sistemas interactúan con el medio, del cual reciben eventos de entrada y generan reacciones de salida. El entorno impone las reacciones a cumplir. Durante la interacción del sistema con su entorno físico, hay situaciones en que éste debe reaccionar lo suficientemente rápido para satisfacer restricciones temporales. Esta respuesta depende tanto del algoritmo como de las características de la plataforma de ejecución; por lo que son determinantes los requerimientos del entorno y las prestaciones de la plataforma a utilizar, para estimar los tiempos de respuesta que el sistema tendrá [2].

El sistema propuesto en este trabajo es un sistema heterogéneo, con un “Procesador de Petri” (PP) y un procesador de propósito general (PG). El PP recibe los eventos y los procesa para calcular el próximo estado del sistema.; mientras que el PG calcula y ejecuta las acciones, desacoplando la lógica de las acciones [3].

La arquitectura propuesta implementa en forma directa e innovadora una relación entre el hardware y la lógica del sistema implementado por un monitor de

concurrencia, por lo que el programa prescinde de secciones críticas y sincronización, dado que es el PP quien realiza estas tareas. Existe una relación unívoca entre el programa ejecutado por el PP y el modelo del sistema realizado con la Red de Petri (RdP), lo que garantiza las propiedades que se verifican en el modelo.

Los requerimientos de los sistemas embebidos, reactivos y concurrentes son: precisión, fiabilidad y flexibilidad estructural. Para alcanzar estos requerimientos el PP se programa directamente con el modelo y esto resuelve la ejecución en relación a los eventos y estado del sistema[4], el PP, procesa y ordena eventos según las restricciones del sistema.

Las aplicaciones de este procesador no sólo se limitan al procesamiento de sistemas reactivos, sino que se puede emplear para sistemas paralelos. Se pueden encontrar numerosos casos resueltos con RdP en [5, 6].

La arquitectura del PP desarrollado en trabajos anteriores es de tipo monolítica [3, 4], en el presente trabajo se propone y desarrolla un PP modular que mantiene las ventajas de los anteriores y se agregan nuevas características. En el proceso se tuvo en cuenta ciertas convenciones de nombre, tanto en la implementación interna como en las interfaces, para así compatibilizar los componentes y estandarizarlos.

1.1 OBJETIVOS

Objetivo general.
Se propone separar en módulos cada función del PP para optimizar la mantenibilidad del sistema y la programación de las colas.

Lo que se busca en primera instancia es generar componentes de hardware reutilizables y estandarizados, que permitan reemplazar, quitar y agregar nuevos componentes de forma simple; esto permite implementar el procesador, en sistemas embebidos pequeños e instanciar los módulos necesarios.

La programación de las colas permite que el PP ejecute distintas RdP no autónomas, lo que proporciona mayor expresividad semántica.

2 Redes de Petri

2.1 Redes de Petri Ordinarias (PN)

Una RdP ordinaria (PN) [7] es una quíntupla definida por PN = (P, T, I-, I+, M_0) donde:

- $P = \{p_1, p_2, \dots, p_n\}$ es un conjunto finito, no vacío, de plazas.
- $T = \{t_1, t_2, \dots, t_m\}$ es un conjunto finito, no vacío, de transiciones.
- Dado que las plazas y las transiciones forman un conjunto bipartito, se cumple: $P \cap T = \{\emptyset\}$, $P \cup T \neq \{\emptyset\}$
- I^-, I^+ son las relaciones de incidencia de salida y entrada de las plazas, que indican las relaciones entre las Plazas y las Transiciones (I^-) y las relaciones entre las transiciones y las plazas (I^+). La Matriz de Incidencia *es* $\boldsymbol{I} = \boldsymbol{I^+} - \boldsymbol{I^-}$

- $M_0 = [m_0(p_1),\ m_0(p_2) \ldots, m_0(p_n)]$ es el marcado inicial de la red, y representa la cantidad de tokens iniciales que contiene cada plaza.

Ahora una RdP queda definida por una 4-tupla: PN = (P, T, I, M_0).

2.2 RdP sincronizadas o no autónomas.

Este tipo de RdP introducen en el modelado los eventos, y son una extensión de las RdP [3, 8]. Las RdP no autónomas modelan sistemas cuyos disparos son sincronizados con eventos discretos externos. Los eventos se asocian con las transiciones, en la Fig. 1 la transición esta sincronizada con el evento E^3, y el disparo se produce cuando se cumplen las siguientes dos condiciones:

- La transición se encuentra sensibilizada
- Se produce el evento asociado a la transición

Los eventos externos se corresponden con cambios de estado del mundo exterior al sistema (incluyendo el tiempo) mientras que los eventos internos son cambios de estado del sistema en sí. Las RdP sincronizadas pueden definirse entonces como una 3-tupla: PNsync = (PN, E, sync) donde:
PN es una RdP marcada, *E* es un conjunto de eventos externos y *sync* es la función que relaciona las transiciones *T* con $E \cup \{e\}$, donde $\{e\}$ es el evento nulo. Los disparos son atómicos e inmediatos.

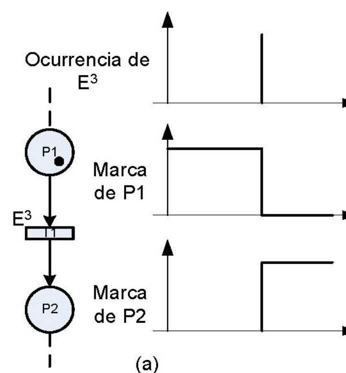

Fig. 1.Transición única asociada a un evento

2.3 Eventos perennes y no perennes.

Existen distintos tipos de eventos que pueden asociarse a una RdP no autónoma, estos son: eventos perennes, no perennes y automáticos o nulos [3].

Los **perennes** son aquellos que cuando ocurren, mantienen su solicitud de disparo hasta que las condiciones de sincronización se cumplan y la transición asociada se dispare. Estos eventos se encolan hasta que sus transiciones puedan dispararse.

Los eventos **no perennes** disparan la transición, si a su arribo esta se encuentra sensibilizada; en caso de que la transición no se encuentre sensibilizada, el evento se pierde. El periodo de tolerancia es de dos ciclos de reloj, por lo que, si luego de los dos ciclos la transición no se sensibiliza, el evento se descarta.

Los eventos **nulos o automáticos** {e} están asociados a las transiciones automáticas. Estos son descriptos en el apartado siguiente.

2.4 Transiciones automáticas

Existe un tipo de evento no externo, que ocurre siempre, y que se denomina $\{e\}$.

En una RdP sincronizada, una o más transiciones pueden estar asociadas al evento $\{e\}$, es decir, pueden ser disparadas automáticamente cuando se sensibilizan. Una RdP simple con una transición T_i, que se encuentra asociada a $\{e\}$. Cuando T_i se sensibiliza, se dispara de inmediato, puesto que el evento $\{e\}$ "está ocurriendo siempre".

2.5 Conflicto entre transiciones

Los conflictos entre transiciones en una RdP sincronizada se dan cuando dos o más transiciones se encuentran sensibilizadas, sus eventos asociados ocurren simultáneamente y el disparo de una de ellas desensibiliza la otra transición. Formalmente, definimos los conflictos como:

$$|\{t_s\}| > 1 \wedge |\{E^s\}| > 1 \text{ y } t_i, t_j \in \{t_s\}$$

Dónde:

- $\{t_s\}$ es un conjunto de transiciones sensibilizadas
- $\{E^s\}$ es un conjunto de eventos asociados a las transiciones $\{t_s\}$
- $t_i\ y\ t_j$ son transiciones en conflicto, comparten al menos una entrada con la misma plaza, si se dispara una de ellas la otra transición puede quedar desensibilizada.

Estos conflictos son resueltos con una política de prioridades [6].

2.6 Interpretación de los eventos y la matriz de incidencia

De la semántica de disparo de una RdP no autónoma, se interpreta la matriz de incidencia como la evaluación conjuntiva de las columnas (transiciones) y las filas (plazas). Es decir, para una matriz de incidencia de dimensión $m \times n$ se evalúan n combinaciones de m variables lógicas y si consideramos que cada transición tiene un evento asociado, la expresión es la siguiente:

$$s_i = \left(\bigwedge_{h=0}^{m-1} (M(p_h) \geq i_{hi})\ and\ E^i \right),$$

Donde i_{hi} son los elementos de la matriz I y $M(p_h)$ es la marca en la plaza h. El resultado s_i son las componentes del vector de sensibilizado.

Esta ecuación es ejecutada por el PP y es la base de la programación directa de este, puesto que la matriz y los eventos son el programa. Esta arquitectura ejecuta una RdP extendida no autónoma, por lo cual su capacidad semántica es la de una máquina de Turing [9, 10].

3 Arquitectura del PP

El PP desarrollado en este trabajo presenta cambios en la arquitectura implementada en [11]. La diferencia radica en la programación de las colas, la división en módulos independientes de hardware y la optimización específica de cada módulo.

Los bloques principales del PP son: el núcleo, las colas, el módulo de prioridades y la interface con dispositivos externos.

3.1 Núcleo

La responsabilidad del núcleo es mantener el estado de la RdP que el procesador ejecuta. La Fig. 2 representa una versión sintetizada de la arquitectura del procesador donde los componentes que conforman el núcleo están marcados con un (*).

El núcleo está compuesto por los siguientes componentes: Matriz de Incidencia, Vector de Marcado y Vector con Peticiones de Disparo.

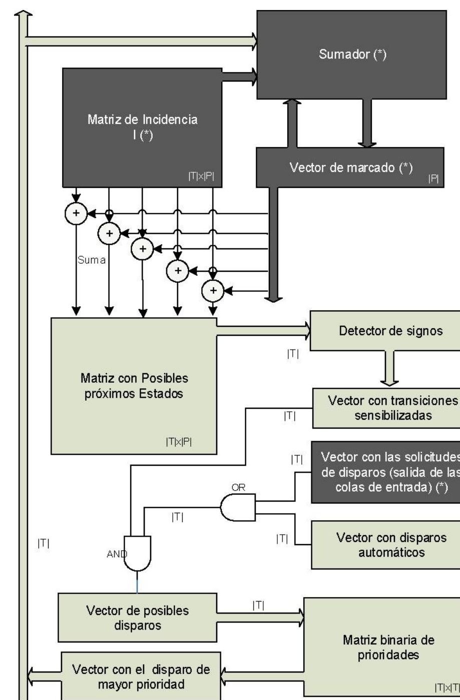

Fig. 2. Arquitectura del PP

Donde las partes y responsabilidad de cada uno de estos componentes son:

- **Matriz de Incidencia I**: almacena la matriz I de la RdP. Su dimensión es de |T|x|P|. Los valores que almacena son enteros.
- **Vector de marcado**: almacena el vector M de la RdP, es decir, el estado de la red. Los valores que guarda son enteros positivos.
- **Colas de entrada**: su interface expone un vector al PP, donde cada posición del vector se corresponde con una transición. El valor de una componente del vector es 1 si hay uno o más eventos en la cola, y es 0 si no hay eventos.
- **Vector de disparos automáticos**: cada posición del vector se corresponde con una transición. Si una transición se configura como automática, el valor correspondiente será siempre igual a 1.
- **Matriz con posibles próximos estados**: corresponde a la suma de cada columna de la matriz I con el vector de marcado. Cada i-ésima columna de esta matriz tiene el estado que se alcanzaría si se dispara la i-ésima transición.
- **Detector de signos**: es un vector en donde cada posición se corresponde con cada columna de la matriz de posibles próximos estados. Si algún valor de columna es

negativo, entonces la posición se marca con un 1, indicando que el disparo de dicha columna corresponde a un marcado inalcanzable (valores negativos significa que se trataría de disparar una transición no sensibilizada).

- **Vector con transiciones sensibilizadas**: es la salida negada del vector de signos.
- **Vector de posibles disparos**: almacena los disparos posibles de la RdP, que se calcula a partir del vector de transiciones sensibilizadas y del vector de transiciones con solicitud de disparo (colas de entrada + transiciones automáticas).

3.2 Funcionamiento del procesador.

Siguiendo las características de procesadores anteriores [3], esta nueva versión del PP también ejecuta la RdP en dos ciclos. Se ha adoptado la semántica de servidor único.

Ciclo 1 – Cálculos. En este ciclo se calcula la transición a disparar. Para esto, el vector de marcado se suma a cada columna de la matriz de incidencia, con lo que se obtiene los signos de los posibles próximos estados para cada posible disparo. Esto se totaliza en una matriz, donde la columna *i* contiene los signos de los próximos marcados.

La salida de la matriz de signos de Posibles Próximos Estados se utiliza como entrada del módulo detector de signos. Este módulo realiza un operación *or* con los valores almacenados en cada columna, si alguno de los valores es negativo, significa que ese próximo estado no será un estado alcanzable por la RdP (la transición no está sensibilizada).

El vector de transiciones sensibilizadas es un vector que contiene un 1 en las posiciones de aquellas columnas de la Matriz de Posibles Próximos Estados que no tenían valores negativos.

Para calcular las transiciones que pueden ser disparadas, necesitamos la información de dos vectores adicionales: el vector de solicitudes de disparo (la salida de las colas de entrada) y el vector de transiciones automáticas. El primero indica cuáles fueron las transiciones que se solicitaron por medio de instrucciones explícitas al procesador (evento de entrada). En cambio, el vector de transiciones automáticas indica aquellas transiciones que no requieren de solicitudes de disparo explícitas {e}.

Por último, para determinar la transición a disparar, el vector con posibles disparos se utiliza como entrada al módulo de cálculo de prioridades. La salida de este módulo es la transición con mayor prioridad, con un único 1 en la posición de la transición seleccionada. Dicha transición se disparará en el próximo ciclo.

Ciclo 2 – Actualización. En este ciclo se realiza el disparo y se actualiza el valor del vector de marcado. Para lograr esto, la transición seleccionada funciona como un selector de columna. El sumador, que se encuentra en la parte superior de la Fig. 2 lleva a cabo la suma del vector de marcado, con la columna de la matriz I seleccionada por el vector de disparo. El resultado de esa suma se almacena como el nuevo vector de marcado.

Además, según corresponda, se actualizan las colas; incrementando el contador de la cola de salida y decrementando el contador de la cola de entrada.

3.3 Colas.

Las colas de entrada y salida del PP fueron rediseñadas, para ser configurables.

Existe una instancia de cola de entrada y de salida por cada transición que posea el PP. Cada instancia incluye un contador saturado en ambos extremos, que cuenta valores enteros positivos. Cada contador posee un detector de máximo que se utiliza para la señal de overflow, y un detector de cero para indicar que la cola está vacía.

Cola de entrada.

La cola de entrada almacena las peticiones de disparo para las transiciones de la RdP, en la Fig. 3 (a) se muestran las señales necesarias para implementar este módulo. Las colas poseen tres modos de funcionamiento.

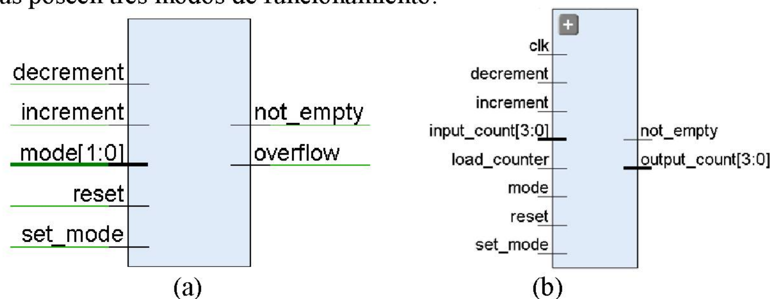

Fig. 3. (a) Cola de entrada, (b) Cola de salida

Modo 1 – Modo normal (evento perenne, modo por defecto): El contador binario se incrementa con cada solicitud de disparo (instrucción explícita que llega al procesador), y se decrementa por cada disparo realizado en el ciclo 2 del funcionamiento del PP. Al contabilizar eventos perennes, el contador se mantiene con su cuenta hasta que se incremente o decremente a través de las señales de entrada de la cola. En este caso, los eventos encolados se mantienen en ese estado hasta que se dispare su transición asociada. Este modo es el que se utiliza en los PP previos

Modo 2 – Transición automática: Este modo de funcionamiento se caracteriza por mantener la señal **not_empty** en alto, ver Fig. 3 (a). Esto se puede interpretar como una transición que siempre se encuentra con solicitud de disparo, o lo que es lo mismo, como una transición automática.

Modo 3 – Evento no perenne: Los eventos que se acumulan en la cola son no perennes. Esto significa que, pasado un determinado tiempo (dos ciclos del PP), el contador asociado vuelve su cuenta a cero. Esto indica que el evento encolado si no ha disparado la transición no deberá dispararla.

Cola de salida.

Como se observa en la Fig. 3 (b), las colas de salida almacenan la cuenta de los disparos realizados. Si se solicita disparar una transición, la cola de salida asociada incrementa en uno su contador y cuando la cola es leída se decrementa en uno. Las colas de salida pueden configurarse, en dos modos de funcionamiento.

Modo 1 – Modo informa: en este modo, el contador interno se incrementa cuando se dispara su transición asociada, y se decrementa cuando se lee para saber si la

transición se disparó. Si se activan las interrupciones del procesador, se genera una interrupción cuando el contador es distinto de cero.
Modo 2 – Modo no informa: en este modo, el contador interno no se utiliza y la señal **not_empty** es cero.

Prioridades y conflictos entre transiciones.

El PP no puede detectar un estado de conflicto, por esto todas las transiciones sensibilizadas como si estuvieran en conflicto. El vector de posibles disparos del PP representa a todas aquellas transiciones que se encuentran en condiciones.
El PP dispara solo una transición por ciclo de ejecución (servidor único). Esto soluciona el problema de los conflictos y mantiene al sistema determinístico; la decisión, de que transición disparar, la toma el módulo de prioridad. Este es implementado por una matriz binaria que establece una relación para determinar la transición de máxima prioridad. Este módulo es también programable en tiempo de ejecución.

Interfaz con Microblaze MCS.

Dado que el PP trabaja con un procesador asociado [3], para construir el sistema es necesario conectar al PP con un procesador tradicional, que ejecute las acciones requeridas por el sistema. Se elige el softcore microblaze MCS [12], puesto que tiene un bajo impacto en los recursos necesarios para su implementación y es soportado por la herramienta de desarrollo de Xilinx [13].
La arquitectura del software que se ejecuta es muy simple y consta de un programa principal y dos drivers. El primer driver es el encargado de exponer una interfaz de comunicación con el PP (**pp_driver**) y el segundo driver (**external_comm**) es el encargado de inicializar y controlar al módulo UART para enviar instrucciones, mediante conexión serial.

4 Resultados

Con el PP implementado se han ejecutado diferentes casos de aplicación para evaluar su desempeño, podemos destacar que se han corrido con éxito los problemas de líneas de producción planteados por Naiqi Wu y MengChu Zhou en [14], las comparaciones se han realizado con respecto resultados obtenidos en [3], arrojando resultados similares.

4.1 Consumo de recursos de la FPGA

Para determinar la cantidad de recursos utilizados y luego compararlas con las demás implementaciones, se instanció al procesador con diferentes cantidades de elementos de los vectores y matrices. Se realizaron múltiples síntesis del núcleo utilizando datos de 4 y de 8 bits de longitud.
La Fig. 4 muestra, para las configuraciones sintetizadas, la cantidad de recursos que se utilizaron de la FPGA. Los registros corresponden a los flip-flops consumidos,

mientras que las LUTs, están relacionadas directamente con la arquitectura de la FPGA Atlys, donde se implementó el procesador.

En la Fig. 4 se observa el aumento exponencial del consumo de recursos, a medida que se aumenta la cantidad de elementos de las matrices y vectores. Una característica importante es que las configuraciones de 4 bits ocupan aproximadamente la misma cantidad de registros que la configuración de 8 bits inmediatamente anterior (incluso en algunas configuraciones la cantidad de registros disminuye). Por ejemplo, la configuración 32x32x4 ocupa menos recursos que la configuración 24x24x8. Eso significa que se pueden procesar redes más grandes, disminuyendo la cantidad de bits utilizados.

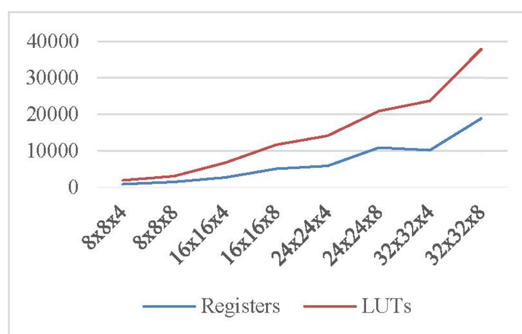

Fig. 4. Gráfico de consumo de recursos

El impacto Microblaze el Microblaze es del 12,72% de las LUTs, y el 21,09% de los registros. Estos recursos son los mismos para todas las arquitecturas del PP.

4.2 Análisis de frecuencias

Para realizar un análisis de las frecuencias, se utilizó el mismo procedimiento de la sección anterior. Esta muestra las frecuencias máximas que se alcanzaron, en cada una de las instancias implementadas del PP.

Un dato importante a tener en cuenta es que el aumento de la longitud de datos almacenados en las matrices y vectores, no afecta de manera significativa a la frecuencia máxima alcanzable. Por ejemplo, si tomamos la configuración 8x8x4 y 8x8x8, la diferencia de frecuencia es solo de 2 MHz (80,073 MHz y 78,422 MHz respectivamente). Sin embargo, si aumentamos la cantidad de elementos de las matrices, es decir, el tamaño de la RdP, la frecuencia es significativamente más baja. Por ejemplo, entre las configuraciones 8x8x8 y 16x16x8 existe una diferencia de 23 MHz aproximadamente (78,422 MHz y 55,32 MHz respectivamente). La limitación en la frecuencia está dada principalmente por la FPGA utilizada.

5 Conclusiones

En el presente proyecto, se amplió y se hizo modular la arquitectura del PP implementado en [3], obteniendo un sistema conformado por componentes interconectados en reemplazo del bloque monolítico. El diseño de módulos y la inclusión de las colas programables en el PP no aumentaron los recursos necesarios

La programación de las colas soporta distintos tipos de eventos. Se ha simplificado la mantenibilidad del procesador, por lo que es posible agregar nuevas funcionalidades, como redes jerárquicas y temporales, extendiendo solo la etapa de control.

Los resultados obtenidos de las optimizaciones de hardware muestran que es apto para sistemas embebidos que requieran de hasta 32 condiciones (transiciones), 32 variables lógicas (plazas) y 32 eventos que deban ser evaluados simultáneamente.
La inclusión del módulo de comunicación serial ha facilitado las pruebas del PP, configurarlo y programarlo desde una consola.
La implementación modular del PP implica un avance para el mantenimiento, la escalabilidad y la futura autoconfiguración del PP. Aún más, la inclusión de colas programables ha ampliado la capacidad semántica del PP.

6 Bibliografía

[1] A. Munir, A. Gordon-Ross, and S. Ranka, *Modeling and Optimization of Parallel and Distributed Embedded Systems*, 2016.

[2] A. Gamatié, *Designing embedded systems with the Signal programming language: synchronous, reactive specification*: Springer Science & Business Media, 2009.

[3] O. Micolini, "ARQUITECTURA ASIMÉTRICA MULTI CORE CON PROCESADOR DE PETRI," Doctor, Informatica, UNLaP, La Plata, Argentina, 2015.

[4] E. Bainomugisha, A. L. Carreton, T. Van Cutsem, S. Mostinckx, and W. De Meuter, "A survey on reactive programming," in *ACM Computing Surveys*, 2012.

[5] F. Moutinho and L. Gomes, *Distributed Embedded Controller Development with Petri Nets: Application to Globally-Asynchronous Locally-Synchronous Systems* vol. 150: Springer, 2015.

[6] M. Haustermann. (2017). *Applications of Petri Nets*. Available: https://www.informatik.uni-hamburg.de/TGI/PetriNets/applications/

[7] M. Diaz, *Petri Nets Fundamental Models, Verification and Applications*. NJ USA: John Wiley & Sons, Inc, 2009.

[8] R. David and H. Alla, *Discrete, continuous, and hybrid Petri nets*. Springer Science & Business Media, 2010.

[9] J. E. H. R. M. J. D. Ullman, "Introduction to Automata Theory, Languages, and Computation " *Prentice Hall,* 2006.

[10] L. Popova-Zeugmann, "Time and Petri Nets," *Springer,* 2013.

[11] M. Orlando, P. Martín, G. N. A., and A. M. A., "Procesador de Petri para la Sincronización de Sistemas Multi-Core Homogéneos," in *CASECongreso Argentino de Sistemas Embebidos*, 2012, pp. 3-8.

[12] I. Xilinx, "Microblaze processor reference guide," *reference manual,* vol. 23, 2011.

[13] XILINX. (2017). *ISE WebPACK Design Software*.

[14] M. Z. Naiqi Wu, *System Modeling and Control with Resource-Oriented Petri Nets*. Boca Raton, FL, 2010.