Como citar este artículo: De Castro, Osberth and Murrugarra, Cecilia, "GASIM: The Gate Array Graphical Simulator for FPGA Architecture Learning" in IEEE Revista Iberoamericana de Tecnologias del Aprendizaje, pp. 95-105, Aug. 2019. doi: 10.1109/RITA.2019.2942254

GASIM: El Simulador Gráfico de Arreglos de Compuertas para el Aprendizaje de Arquitectura de FPGAs

Osberth De Castro, *Member, IEEE,* and Cecilia Murrugarra, *Member, IEEE*

***Title*—GASIM: The Gate Array Graphical Simulator for FPGA Architecture Learning**

***Abstract*—Field Programmable Gate Arrays, also called FPGA, are the main contemporary tool for Digital systems learning and development. Its benefits are more than evident, but also are its internal complexity. The widespread use of FPGA devices has created a necessity for understanding the inner logic using diagrams or theoretical explanations. This paper presents an innovative approach for FPGA basic architecture learning called GASIM. A conceptually complete model for a simple but general 2D graphical FPGA has been designed and implemented in such a way that is easy to understand, aesthetically appealing from the student's point of view, and also easy to use by any teacher. The aim of this kind of design is to fill the conceptual gap we have found while teaching FPGA based courses. The final tool is a 2D modular graphical library for a standalone logic simulator that not only allows teachers and students to program an FPGA Model and simulate every aspect of its internal logic complexities but also to construct their own limited, but complete FPGA structure using drag and drop graphical modules and programmable interconnections. Our study provides a technical description of the tool, and our first results in the classroom.**

***Index Terms*—FPGA, Simulators, Teaching, Learning, Digital Systems, Devices.**

I. Introducción

GLOBALMENTE, y durante más de dos décadas, la enseñanza en los cursos de Diseño de Circuitos y Sistemas Digitales, en programas universitarios de ingeniería, se ha implementado utilizando dispositivos PLD (Dispositivos Lógicos Programables) y FPGA (Arreglos de Compuertas Programables en Campo). Se han informado algunos buenos resultados sobre el uso de estos dispositivos como la herramienta principal para aprender metodologías y entornos, como en [1]–[4] y [5]. Nosotros también hemos utilizado estos dispositivos en nuestras clases durante mucho tiempo. Desde el primer uso de metodologías basadas en PLD y FPGA, las ventajas prácticas fueron claras. Estos dispositivos permitirían a maestros y estudiantes diseñar y crear, dentro de un chip preconfigurado y confiable, circuitos digitales complejos en muy poco tiempo, por lo que la implementación es más rápida, más limpia y las actividades de aprendizaje de diseño en clase pueden ser más profundas y más significativas que antes.

Al igual que en cualquier área de enseñanza de la ingeniería electrónica, utilizar herramientas más complejas tiene un precio. En nuestra experiencia de aula, este precio se presenta en forma de una brecha conceptual significativa entre la lógica y el circuito que resulta en una comprensión deficiente de las consecuencias de la lógica digital del diseño e implementación de circuitos dentro del chip FPGA. Como resultado, crear circuitos digitales en un FPGA en los primeros cursos de circuitos digitales se convierte en una práctica de codificación abstracta en lenguajes de descripción de hardware, por ejemplo Verilog y VHDL, en la que la mayoría de los estudiantes piensan sus implementaciones como una ejecución secuencial de declaraciones, como en cualquier programa de computadora, pero no como los circuitos reales que realmente están creando dentro del dispositivo FPGA.

Nuestra hipótesis es que la brecha conceptual creada por el uso de dispositivos FPGA en cursos básicos de pregrado puede cerrarse parcialmente mediante el uso de simulaciones gráficas utilizando modelos de FPGA genéricos, simples pero bien construidos, de la misma manera que los cursos de organización de computadoras usan simuladores de procesadores para ayudar a comprender su funcionamiento.

A. La Complejidad Interna del FPGA

Los FPGA son circuitos electrónicos integrados muy complejos que permiten implementaciones rápidas de sistemas digitales en un solo dispositivo. Hoy en día son la herramienta preferida para la creación rápida de prototipos y pruebas, debido a las ventajas de tiempo de diseño a comercialización que ofrece su uso. Para hacer posibles esas ventajas, un chip de FPGA es un circuito increíblemente grande y complejo [6], que consiste en una arquitectura específica de compuertas lógicas y circuitos de enrutamiento programables. Para

O. De Castro trabaja para este proyecto con los programas de Ingeniería Multimedia e Ingeniería de Sistemas de la Universidad de San Buenaventura, en Bogotá, Colombia, y con el Departamento de Electrónica y Circuitos de la Universidad Simón Bolívar, de Caracas, Venezuela.
E-mail: odecastro@usbbog.edu.co

C. Murrugarra trabaja con el programa de Ingeniería Electrónica de la Universidad El Bosque, en Bogotá, Colombia y con el Departamento of Electrónica y Circuitos de la Universidad Simón Bolívar, en Caracas, Venezuela.
E-mail: cmurrugarra@unbosque.edu.co

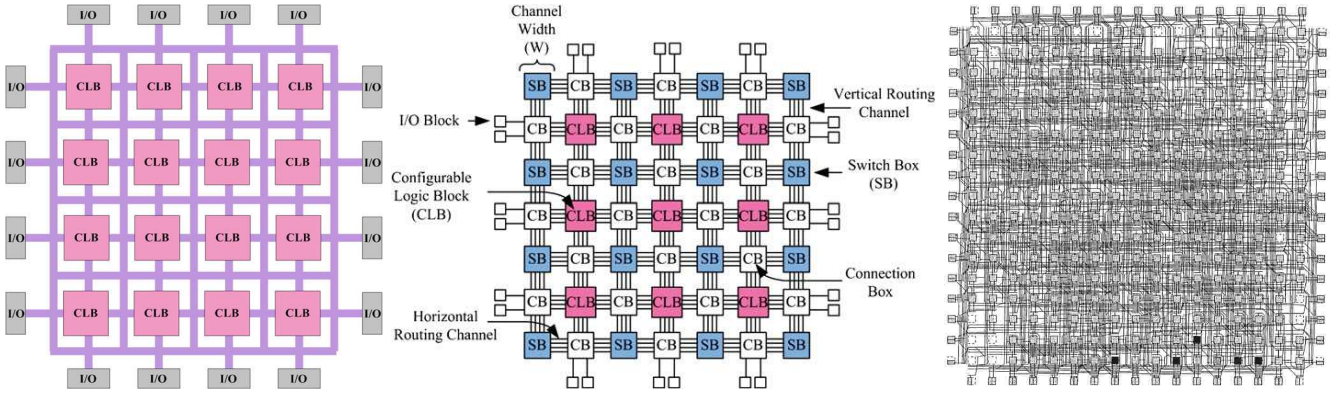

Fig. 1. Estructura de bloques de un FPGA simple, tal como se muestra en [6] y [7]. A la derecha, un ejemplo de enrutamiento de conexiones para un FPGA, como se muestra en [8], obtenido mediante simulación.

implementar un circuito digital, en lugar de diseñar desde cero los circuitos, un desarrollador programa circuitos de enrutamiento FPGA y tablas de verdad en la forma de *Lookup Tables* para hacer lógica y conexiones entre sus compuertas internas disponibles. Para programar esas rutas y lógica, el desarrollador utiliza un lenguaje de especificación de hardware (HDL), como Verilog [9] o VHDL [10], que se compila generando una especificación de conexión de enrutamiento de bajo nivel con algún grado de optimización, de acuerdo con un dispositivo objetivo específico y herramientas desarrollo seleccionadas.

Se puede ver una idea detallada pero general de un FPGA en [6]–[8], y [11]. En esos libros y artículos, podemos encontrar diagramas como la figura 1, donde un FPGA se muestra como una malla de **CLB** (Bloques Lógicos Configurables), pero no se muestran bloques de interconexión. Se puede ver una vista un poco más completa pero aún simple en el centro, que muestran los mismos **CLB**, pero también varias **CB** (Cajas de conexión) y **SB** (Cajas de Interconexión). Estas dos representaciones simples y muy comunes en los libros de texto de diseño digital y las aulas de teoría.

Si queremos ver una representación de enrutamiento más detallada para un FPGA simple, en [8] podemos ver en la figura 1 un enrutamiento resultante obtenido mediante la simulación de código Verilog. Hay muchas otras representaciones disponibles para tratar de comprender el mecanismo de trabajo de un FPGA. Por ejemplo, herramientas de desarrollo como Intel Altera Quartus[1] o ISE Design Suite by Xilinx[2] utiliza representaciones de hardware para ayudar a los desarrolladores a ver las configuraciones y el enrutamiento de los módulos. Tengamos en cuenta que todavía no hemos alcanzado la arquitectura de nivel de compuerta lógica, por lo que en este momento todavía no conocemos los circuitos lógicos internos reales de este dispositivo y cómo funcionan. Se necesita algo más.

B. La abstracción y el Circuito

Es claro que puede ser un reto enorme aprender efectivamente los mecanismos lógicos internos de un FPGA durante un curso inicial de Circuitos Digitales, de modo que la práctica usual es usar un chip de FPGA como herramienta de implementación y nada más, lo cual es válido. Por otro lado, es muy común que, para entender el nivel más bajo de abstracción de arquitecturas del computador y diseño digital, se utilicen suposiciones teóricas como herramienta principal. Estas suposiciones son con frecuencia incompletas o imprecisas, lo cual llega a errores conceptuales y de cálculo en las actividades de aula. Algunos de los mejores libros de texto de Diseño Digital, como Brown and Vranesic [12] y Wakerly [13] describen la estructura interna de los FPGA de esta manera. No hay forma práctica en estos casos de tocar internamente y entender el funcionamiento del chip cuando la parte práctica de los libros (el código en VHDL y Verilog) es implementado. Al final, el uso de la abstracción teórica es correcto, pero no suficiente. Incluso en ambientes simulados, una lógica interactiva es la mejor opción.

C. Cerrar la brecha usando simulación Gráfica

Los simuladores son excelentes herramientas para la comprensión de las estructuras subyascentes y el comportamiento de sistemas y circuitos complejos. En los temas de Organización del Computador existen opciones para este estudio. Por ejemplo, [14], [15] y [16] como simuladores del procesador MIPS [17]. Algunos esfuerzos han creado incluso arquitecturas pedagógicas especiales de amplio uso, como el MIC1 y su simulador [18], o el procesador USB08 [19]. De estas opciones, sólo el último es un circuito electrónico simulado. Este trabajo toma la idea de usar componentes digitales básicos que construyen arquitecturas complejas para crear nuestro propio simulador de arquitectura de FPGA. Este tipo de simulador es, por lo tanto, construido en base a operadores lógicos básicos (AND, OR Not) en un ambiente gráfico seleccionado.

1) El Reto de un FPGA Gráfico: La idea detrás de la construcción de un simulador gráfico interno al FPGA es la construcción de una librería gráfica de componentes internos para un modelo de FPGA específico, y un conjunto de requerimientos de estética, usabilidad y corrección lógica. Tal simulador debería permitir a los estudiantes construir su propio FPGA, usando componentes complejos de la librería, y además poder programarlo. A diferencia del procesador

[1]https://www.intel.com/content/www/us/en/products/programmable.html
[2]https://www.xilinx.com/products/design-tools/ise-design-suite.html

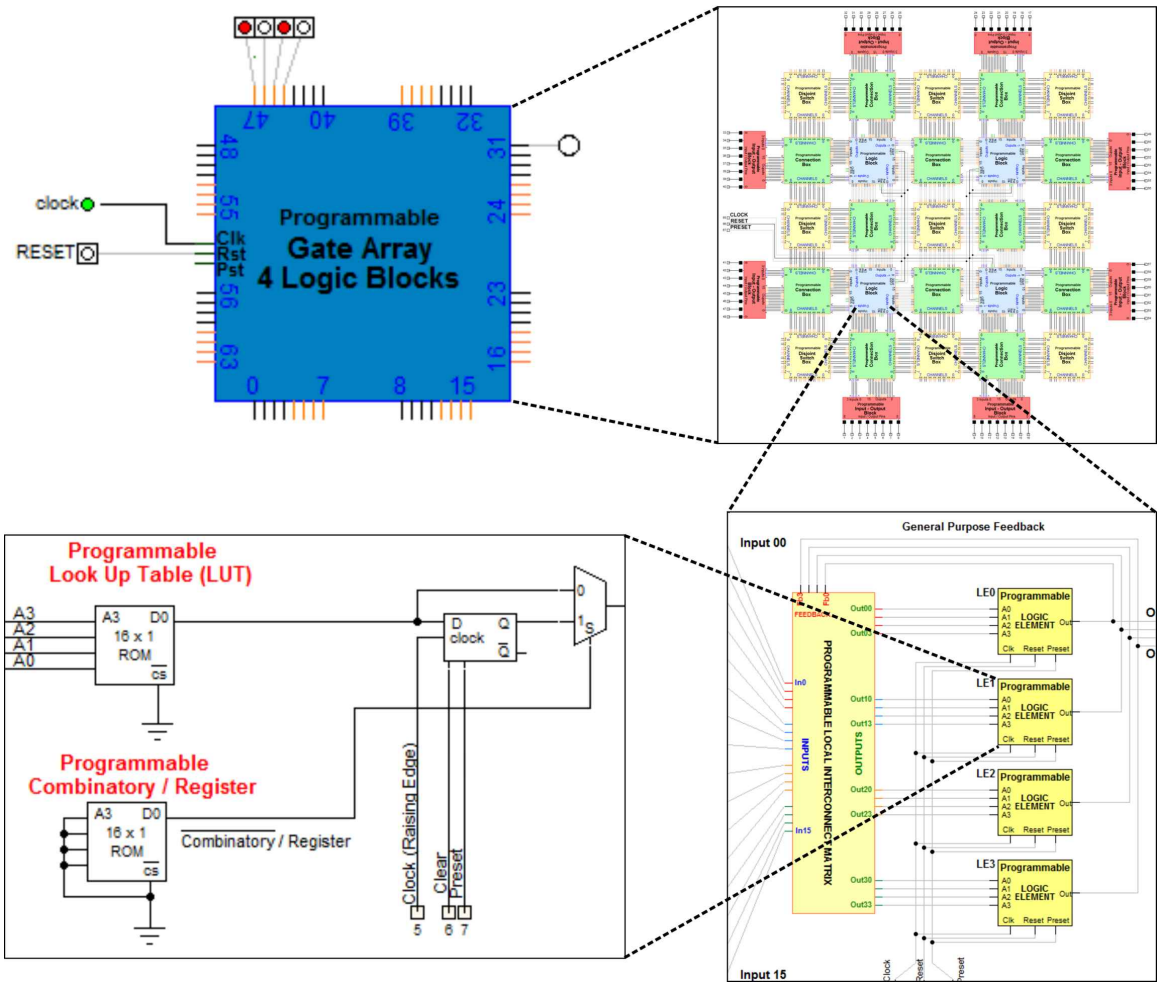

Fig. 2. Jerarquía de GASIM. Se pueden apreciar cuatro niveles en la jerarquía. de izquierda a derecha: El FPGA en su encapsulado (nivel 4), que puede ser usado para programación de una aplicación simple, el nivel de estructuras principales (nivel 3) que sirve para ensamblar un FPGA, el nivel de bloques básicos (nivel 2) que contiene los bloques básicos de bajo nivel, y el nivel de circuitos digitales (nivel 1). Todas las imágenes son directas del simulador.

de un computador, donde la programación es una operación externa, la programación del FPGA consistiría en reconfigurar (programar) todos los componentes para la implementación del enrutamiento y las operaciones lógicas requeridas.

Como se indicó anteriormente, no existen simuladores de FPGA que cumplan esta tarea específica, de modo que es un reto construir una primera versión que cumpla con los requerimientos. El primer reto es estético. Un FPGA es un circuito muy complejo, incluso en modelos simples, de modo que una representación gráfica entendible debe ser cuidadosamente diseñada. La mayoría de los simuladores de circuitos son en dos dimensiones, y todas las representaciones de FPGA en la literatura también 1, de modo que el simulador ideal sería un circuito 2D que luzca como las representaciones de los libros, pero que permita la interacción del usuario para la reconfiguración. El segundo reto es funcional. El simulador debe ser capaz de ejecutar las configuraciones y funciones lógicas sin error, incluida la propagación de señales de reloj y la programación de líneas bidireccionales de Entrada/Salida.

En las siguientes secciones, este trabajo muestra un diseño, implementación y pruebas de un modelo de simulación de FPGA que cumple con todas las consideraciones, llamado GASIM (Gate Array Simulator). Todas las figuras del artículo provienen de vistas reales del simulador. Además se presentan las herramientas seleccionadas y la metodología empleada, se explica el uso del simulador y se discuten los primeros resultados de su uso en el aula de los cursos de circuitos digitales. Es muy importante aclarar que GASIM no es una representación física fiel a un FPGA, sino una abstracción lógica interactiva cercana, con el propósito de conceptualizar un modelo simple pero general de FPGA.

II. El Simulador de Arreglos de Compuertas GASIM

GASIM es un conjunto de módulos reconocibles que permiten el ensamblaje y programación de un FPGA de modelo simplificado, o la manipulación de uno pre-diseñado para aprendizaje y pruebas. Cada uno de los módulos ha sido implementado con circuitos de operadores lógicos básicos en forma modular, de modo que se puede analizar todas las estructuras internas, hasta el nivel de los operadores más básicos.

En esta primera versión, a diferencia de un FPGA real, cada módulo de GASIM es programable internamente. Esta

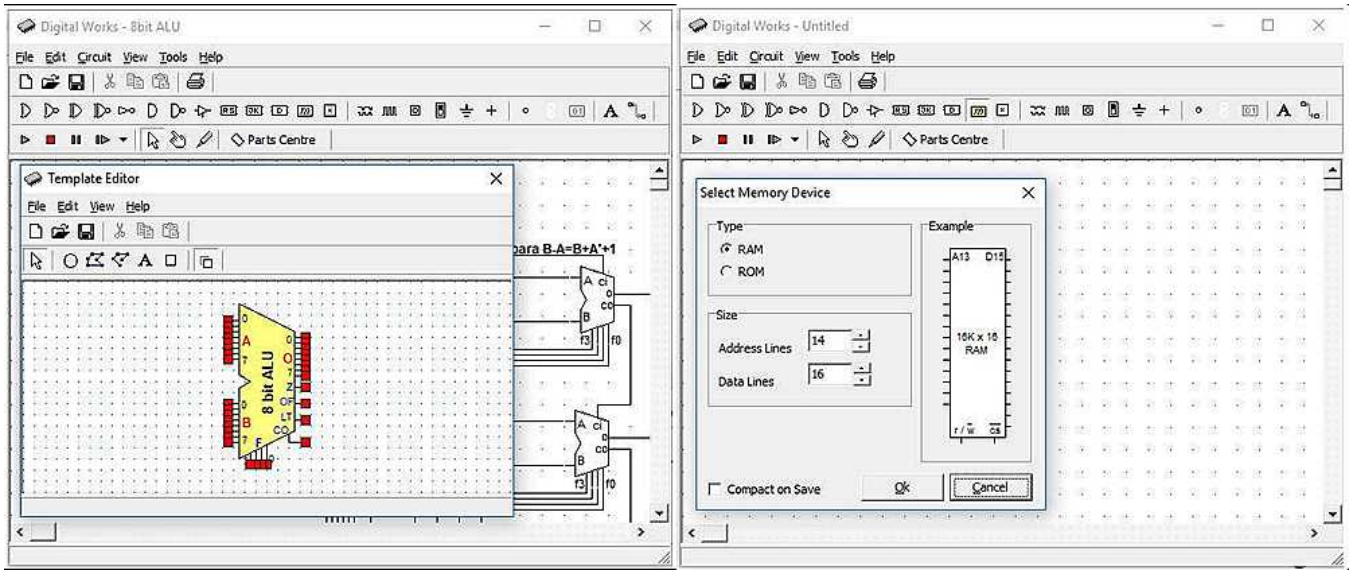

Fig. 3. Características del software Digital Works [20]. Editor de plantillas (izquierda) and dispositivos programables (derecha).

solución tiene ventajas y desventajas. Por ejemplo, es fácil programar y probar independientemente un módulo de cualquier nivel, y permite gran modularidad, pero la programación de un ensamblaje de módulos, o del FPGA completo es más tediosa y propensa a errores. Este compromiso entre modularidad y programabilidad es razonable a nivel pedagógico, porque hace innecesario el uso de memoria externa y otra circuitería de soporte en cada paso, permitiendo pruebas y verificación rápida. Por ejemplo, los estudiantes podrían desear probar un módulo específico, o un pequeño ensamblaje, antes de construir un sistema más grande, y permitir estas pruebas en tiempo real simplifica las actividades del proceso de aprendizaje en el aula.

Es importante aclarar que GASIM es sólo una herramienta de aprendizaje conceptual para aprendizaje, y no una herramienta de desarrollo. La dificultad de programación permite al estudiante comprender la complejidad y limitaciones de un proceso de síntesis para un FPGA, y tomarlo en cuenta más adelante, en su trabajo como desarrollador. GASIM no está diseñado para uso más allá del aula de clase, y su uso debe ser seguido por el aprendizaje en el uso de chips reales y herramientas de desarrollo apropiadas.

A. Arquitectura del Simulador GASIM

La arquitectura general de GASIM es una abstracción jerárquica de cuatro niveles, como muestra la figura 2. Cada nivel puede ser probado, programado, o utilizado para construir un nivel superior.

- **Nivel 4, Aplicación y pruebas:** En el nivel más alto, se provee un FPGA ensamblado para pruebas. Este permite ver todas las capas, un encapsulado externo, y programar una aplicación pequeña.
- **Nivel 3, Estructuras Principales:** Este nivel contiene los componentes estructurales de alto nivel que la mayoría de los libros muestran en diagramas. Los bloques de enrutamiento principal, los bloques lógicos y periféricos. No se muestra operación interna. Estos bloques pueden ser utilizados para construir un FPGA en 2D.
- **Nivel 2, Estructuras de Bajo Nivel:** Este nivel contiene los bloques locales específicos que hacen todo funcionar, tales como los Elementos Lógicos, Bloques de Interconexión local, celdas de entrada/salida y otras.
- **Nivel 1, Circuitos Digitales:** en este nivel encontramos los circuitos de operadores digitales básicos que constituyen la implementación de todos los niveles superiores. Aquí el usuario puede modificar las estructuras de bajo nivel o crear nuevos usando compuertas, flip-flops y otros dispositivos de la librería.

Para lograr que esta jerarquía funcione apropiadamente, se deben cumplir varias condiciones. En primer lugar, es importante que cada diseño sea suficientemente simple y manejable, pero manteniendo la complejidad necesaria de estructuras lógicas funcionales, y trabajando en un ambiente esquemático 2D que resulte atractivo. Para esta combinación, se debe enfrentar dos retos: uno estético y uno funcional, que nos lleva a la selección de herramientas específicas de desarrollo, modelos factibles de FPGA y estilo de diseño.

1) El Reto Estético: Dado que GASIM es una herramienta gráfica, son críticos los siguientes requerimientos estéticos:

1) **Simulación Gráfica:** GASIM debe mostrar una simulación gráfica interactiva. Cambios en las señales lógicas deben ser claramente visibles.
2) **Modularidad Gráfica:** Es vital la capacidad de crear estructuras lógicas en forma modular sin perder interactividad. Esto significa la creación de bloques gráficos dentro de bloques con facilidad de navegación gráfica entre ellos.
3) **Diseño Libre Externo de Bloques:** también es importante la capacidad de crear la forma externa de todas las estructuras, incluyendo la forma, color y posiciones de pines con la menor restricción posible.

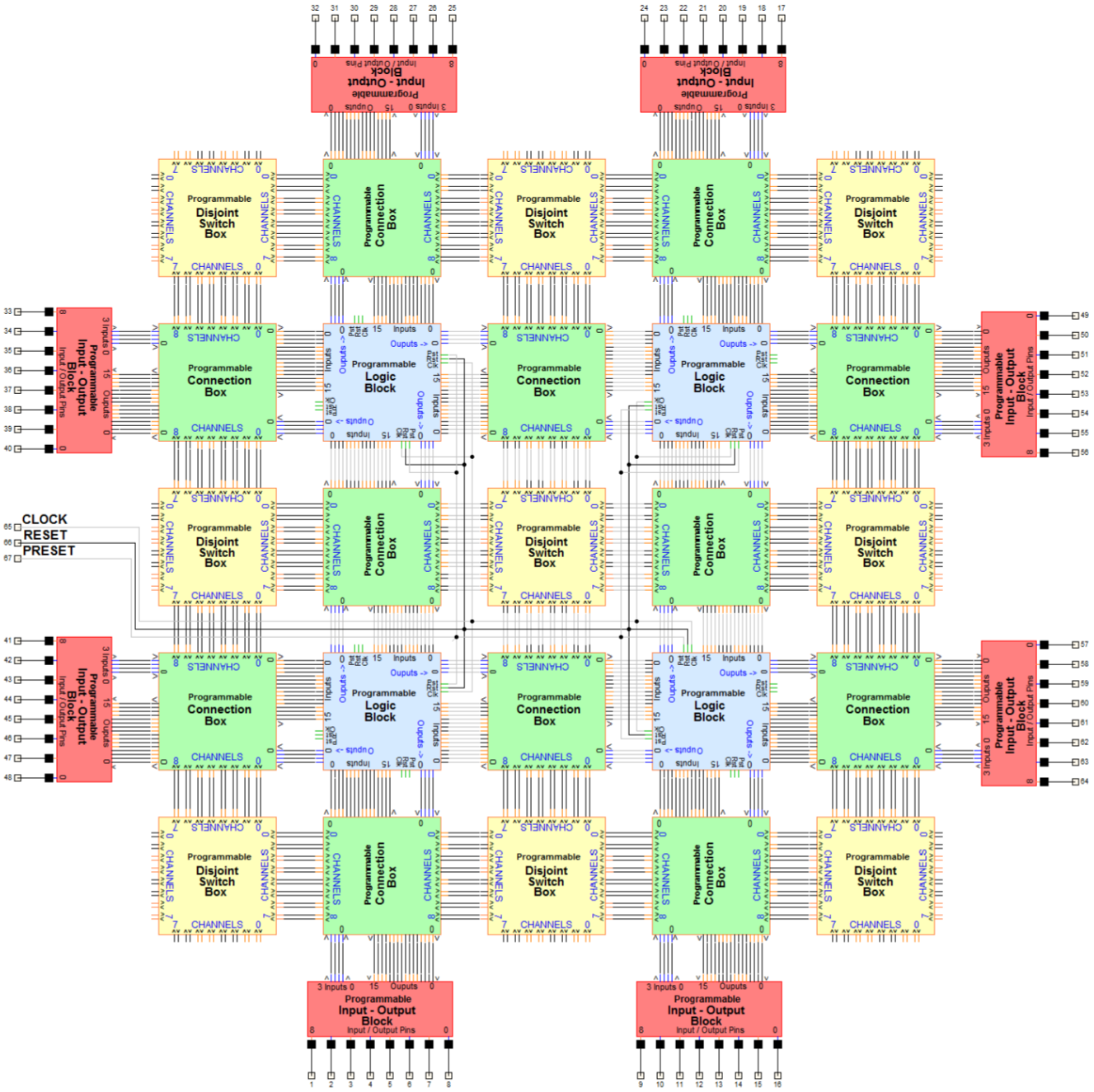

Fig. 4. FPGA completo usando GASIM, mostrando la arquitectura de malla. Los bloques amarillos son Bloques de Enrutamiento (Switch Boxes SB), los verdes son Bloques de Interconexión (Connection Boxes CB), los azules son Bloques Lógicos (Logic Blocks LB) y los rojos son Bloques de Entrada/Salida (Input-Output Blocks IOB).

4) **Diseño con Apariencia Bibliográfica:** Dados los requerimientos anteriores, el diseño de todas las estructuras debe ser tener la apariencia familiar de la literatura relacionada con FPGA utilizada en los cursos.
5) **Diseño Limpio de Circuitos:** Dada la complejidad y tamaño de los circuitos, la implementación debe ser la representación en 2D más clara posible.

Es un hecho que los tres primeros requerimientos llevan a una selección adecuada de la herramienta de desarrollo. Sin esa selección, el cuarto requerimiento es imposible de obtener. Para esta tarea, se encontró la herramienta adecuada. Digital Works [20] fue seleccionada como herramienta principal de desarrollo, por encima de otras también conocidas, como Logisim [21]. Digital Works es un ambiente de simulación de circuitos lógicos simple, gratuito[3] que provee todas las características necesarias, a pesar de que su librería de operadores es modesta. La figura 3 muestra, por ejemplo, el editor de plantillas para bloques del software, que nos permitió darle las características estéticas a todos los bloques que se diseñaron. Este editor permite crear formas, dar color, tamaño,

[3]https://www.mecanique.co.uk/shop

añadir texto, posición de pines sin restricciones y mucho más. Además se muestra la característica de fácil programación de dispositivos de memoria. La característica de interacción y modularidad no se muestra en la figura, p ero también está disponible en la herramienta. Digital Works permite la creación de bloques dentro de bloques, y la navegación entre ellos usando mouse.

2) El Reto del Enrutamiento: Aparte de los requerimientos estéticos, es clave la selección de un modelo de enrutamiento estándar que ayude a la representación gráfica d e u n FPGA en GASIM, que permita simplicidad pero completitud. Un FPGA es, en esencia, una máquina de enrutamiento con lógica programable. Para esto, se ha seleccionado un modelo de FPGA de tipo Isla 2D, descrito en [6], y tal como se muestra en la figura 1 . E ste m odelo, a mpliamente u sado, c onsiste en una malla de bloques cuadrados interconectados por canales horizontales y verticales, donde los bloques de enrutamiento para el uso de los canales ocupa entre el 80% y el 90% del circuito. En la figura 4 , s e m uestra e l m odelo d e m alla para un FPGA visible en GASIM. Cada bloque de la malla es de forma cuadrada y el color representa la función. En la figura se pueden identificar c uatro t ipos d e e structuras: C ajas de Enrutamiento (Switch Boxes SB) en color amarillo, Cajas de Interconexión (Connection Boxes CB) en color verde, Bloques Lógicos (Logic Blocks LB) en azul, y Bloques de Entrada-Salida (Input-Output Blocks IOB). Aunque un FPGA completo de la figura n o i ncluye c onexiones l argas, G ASIM e s una herramienta modular, de modo que cualquier configuración, incluyendo una con menos Switch Boxes para bajar el retraso de las señales puede ser también construida, tal como es deseado en muchos casos reales. Además, en un FPGA real, los canales pueden saltar otros bloques, de modo que menos enrutamiento es necesario. En esta versión, se ha evitado construir un FPGA inicial con cableado escondido, de modo que todas las conexiones son evidentes. El modelo de Malla también ayuda a construir modelos de FPGA más grandes y más pequeños.

3) Canales: Los canales (*Channels*) son los huesos de la máquina de enrutamiento. Cualquier señal viajando entre dos puntos de nuestro FPGA lo hace mediante canales. En la figura 4, los canales son los cables horizontales y verticales que pasan por los Connection Boxes y los Switch Boxes. En este modelo, hay 16 canales unidireccionales (8 en cada dirección). Se utilizaron canales unidireccionales debido a una restricción del modelo de 3 estados del simulador Digital Works, que no puede ser evitado. En secciones posteriores esta restricción no afecta el diseño de los Input-Output Blocks.

4) Switch Boxes: El componente más importante de un FPGA con modelo de malla es la Caja de Enrutamiento, o Switch Box (**SB**). Estos bloques son los responsables de la interconexión programable entre los canales verticales y horizontales de la malla, En general pueden manejar canales bidireccionales. En GASIM manejan los 16 canales unidireccionales agrupados por pares. En la figura 4 se muestran como las cajas amarillas con canales conectados a los 4 costados. De esta forma, el **SB** puede conectar dos canales desde cualquier dirección.

5) Connection Boxes: Mostrados en color verde en la figura 4, las cajas de interconexión, o Connection Boxes (**CB**), permite el acceso de los bloques lógicos y los bloques de Entrada-Salida a los canales del FPGA. Para lograr esto, esta caja programable interconecta cualquier salida o entrada de un bloque lógico con cualquier canal que luego irá a un **SB**.

6) Logic Blocks: Soportados por la malla de **SB** y **CB** se encuentran los bloques lógicos o Logic Blocks (**LB**) mostrados en azul en la figura 4. Los **LB** contienen la lógica del usuario para las aplicaciones. Desde el punto de vista del desarrollador, este es el bloque programable que implementa funciones lógicas y máquinas de estados finitos.

7) Input-Output Blocks: Los bloques rojos y delgados de la figura 4 son los Bloques de Entrada-Salida, o Input-Out Blocks (**IOB**). Estos bloques, también programables conectan Entradas y salidas bidireccionales del FPGA a canales y a bloques lógicos de la malla. En este caso, el desarrollador puede programar cada pin externo del (**IOB**) como entrada o salida y enrutar la línea usando un (**CB**).

8) Señales Generales: La estructura de malla además soporta el uso de señales de reloj (Clock o CLK), Reset y Preset, necesarias por los bloques lógicos para aplicaciones síncronas y asíncronas. En el FPGA de la figura 4 estas señales son dirigidas a todos los bloques lógicos, de modo que la programación asociada ocurre en estos (**LB**). Los bloques (**IOB**), (**IOB**) y (**IOB**) no son afectados, ni manejan este tipo de señales en esta versión. Todos los bloques son alimentados internamente, dada la naturaleza lógica del simulador, de modo que no se requiere mostrar redes extra para esto.

B. Implementación

1) Diseño General: Como regla general, todos los circuitos deben ser fáciles de conectar y de entender a nivel visual. En la figura 4, esta característica es evidente, garantizada por el siguiente conjunto de reglas de diseño:

1) La implementación de toda caja o bloque de la malla debe seguir un diseño estándar de tamaño, espaciado de pines y puertos, permitiendo simetría y conexiones rectas perpendiculares. Esa regla aplica a todos los bloques, menos a los bloque de Entrada-Salida, que no forman parte interna de la malla.
2) Las cajas internas de la malla deben tener conexiones disponibles en todos los cuatro lados del componente. Dependiendo de la naturaleza de la conexión, algunos puertos resultan redundantes (la misma señal en los cuatro lados).
3) La dirección de los puertos debe ser explícita. Cada bloque indica la dirección de la señal para cada conjunto de señales de entrada y salida.
4) Los circuitos de programación son internos a los bloques. Cada bloque es programable independientemente e internamente. Esto permite máxima modularidad y rapidez de pruebas de un sistema o subsistema del FPGA.

Las figuras 4 y 5 muestran las reglas de implementación aplicadas a los bloques de alto nivel. Todos los componentes tienen conexiones a los cuatro costados, y puertos terminales

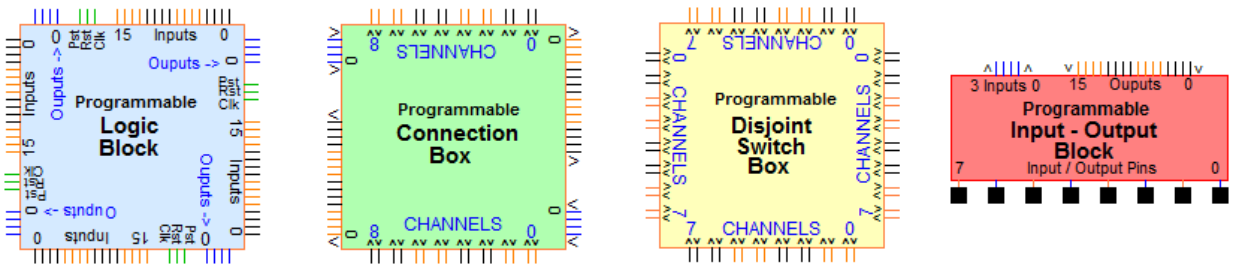

Fig. 5. Componentes de alto nivel de GASIM. Muestra la simetría, redundancia de puertos, direcciones y esquema de agrupamiento de señales.

organizados de tal forma que una malla puede ser construida utilizando solo conexiones rectas.

Para explicar la estructura de cada bloque, se utilizará el término ”programable” será usado. Por ejemplo los ”selectores programables” son ampliamente utilizados. En nuestra implementación, un dispositivo programable es aquel que dispone de un pequeño módulo de memoria RAM asociado que contiene la configuración programable del dispositivo.

2) Bloques Lógicos y Elementos Lógicos: Desde el punto de vista de utilidad de un FPGA, un bloque lógico **LB** es el núcleo. Cada función lógica implementada en el FPGA, cada Máquina de Estados Finitos es programada dentro de uno o más **LB** interconectados. Debido a esto, cada **LB** debe disponer de 3 tipos de puertos: Entradas lógicas, salidas lógicas, y señales de soporte (Clock, Reset y Preset). En el caso de nuestra **LB**, distribuidas como:

- 16 Entradas Lógicas
- 4 Salidas Lógicas
- 1 Entrada de Clock
- 1 Entrada de Reset
- 1 Entrada de Preset

La figura 5 muestra estos puertos. Todos los puertos están replicados en los cuatro lados, de modo que es la misma señal. Por ejemplo, la **Input 0** es la misma en los cuatro lados, de modo que puede ser alimentada por cualquiera. Internamente, las 4 **Input 0** van a una compuerta **OR**. Esta no es una representación precisa del mecanismo físico utilizado en un FPGA real, donde este tipo de conexiones utilizan circuitos pull-down. GASIM usa compuertas OR lógicas, pues Digital Works no dispone de estos circuitos, que en futuras versiones podrían ser implementados como aproximaciones gráficas. En la figura 6 podemos ver las compuertas OR generando la entrada unificada. Esta característica nos permite simplificar la redundancia, pero debe ser tomada en cuenta al momento de programar el bloque, para evitar resultados inesperados.

La misma lógica es aplicada a las señales de entrada Reset, Preset y CLock. l desarrollador debe asegurarse de que sólo una de las cuatro entradas para cada señal esté conectada externamente. En el caso de las salidas, las correspondientes a los cuatro lados de cada señal están conectadas directamente entre sí. Son, en efecto, la misma señal, sin ninguna lógica extra o protección. Si la señal es necesaria en mas de un lado del bloque, puede ser utilizada. La figura 6 muestra la arquitectura de un Bloque Lógico. Es posible observar que es fácil de seguir en 2D su organización, que consiste en:

- Cuatro Elementos Lógicos (**LE**), cada uno conteniendo una Tabla de búsqueda lógica (Lookup Table) programable de cuatro bits, un flip-flop tipo D para implementación de Máquinas de Estados Finitos, y un selector programable para operación síncrona/asíncrona. La Tabla de búsqueda es, en esencia, una memoria ROM de 16 direcciones y 1 bit. La estructura de un **LE** se muestra en la figura 2.
- Una Matriz de Interconexión Local programable o Local Interconnect Matrix (**LIM**) para 20 entradas, de las cuales 4 son para realimentaciones del **LE**, a 16 salidas. Esta **LIM** es un arreglo programable de selectores de 20 a 1, conectados de tal forma que cualquier entrada puede ser enrutada a cualquier salida.

Esta organización puede ser usada para implementaciones de Máquinas de Estados Finitos de hasta 16 estados y hasta 4 funciones lógicas de 4 bits, o cualquier combinación intermedia.

3) Cajas de Conexión: Las Cajas de Conexión o Connection Boxes (**CB**) son puntos de acceso para los **LB** y los **IOB** a los canales de interconexión de la malla del FPGA. Realiza un mapeo de entradas y salidas a canales. La figura 5 muestra que las cajas de conexión tiene dos tipos de puertos:

1) Puertos de Entrada/Salida. Estos puertos se ajustan a la posición de los pines de los **LB** y los **IOB**.
2) Puertos de canal. Distribuidos en pares (uno de entrada y uno de salida) en la dirección de los canales de FPGA, perpendiculares a los puertos lógicos.

La figura 7 muestra detalles parciales de la estructura interna de una **CB**. Su organización consiste de un arreglo de cajas de conexión miniatura de 1 bit, uno por cada par de canales, basados en selectores programables para conectar salidas a cualquiera de las 16 líneas de canales. Esto incluye la posibilidad de canalizar una salida lógica a una entrada lógica de otro bloque. Por ejemplo, una salida de un **LB** a una entrada de un **IOB** adyacente, sin pasar por canales. Además permite la opción de retornar una señal al mismo bloque de origen. para lograr acceder a estas mini-**CB** o un puerto de salida lógica, el **CB** dispone de selectores programables de 8 a 1. En total, el **CB** contiene 8 mini-**CB** y 16 selectores programables.

En su forma actual, un **CB** no maneja señales de Clock, Reset ni Preset. Esta separación puede verse en la figura 4.

4) Cajas de Interconexión: GASIM implementa cajas de interconexión, o Switch Boxes (**SB**) de tipo desarticulado (disjoint Switch Boxes), para la interconexión de canales

verticales y horizontales en la malla del FPGA. El modelo desarticulado es simple y tiene limitaciones. Este modelo solo es capaz de conectar un par de canales específicos. E n este caso, un canal 0 horizontal solo puede ser conectado con un canal 0 vertical u horizontal que sale por las otras 3 caras del bloque. Sin embargo, dada la modularidad de GASIM, el usuario puede decidir si usa la caja de interconexión, y usar métodos alternos, como líneas largas, como muchos FPGA utilizan.

La figura 8 muestra la organización de una **SB**, que consiste de un selector programable de 4 a 1 para cada canal. En total un **SB** contiene 32 de estos selectores. El diseño del circuito es un arreglo cuadrado de selectores de modo que el usuario solo tiene que seguir el cableado interno para saber qué selector programar para la interconexión necesaria.

5) Bloque de Entrada-Salida: El bloque de Entrada-Salida o Input-output Block (**IOB**) tiene una organización mostrada en la figura 9. Dadas las restricciones de las **SB** y los **LB**, se implementaron **IOB** pequeños de 8 bits con todos los pines bidireccionales, y con la capacidad de conectarse por completo a una **CB**. La **IOB** consiste de los siguientes componentes:

- Ocho *Celdas* de entrada-salida, una por cada pin. Cada celda está implementada con un circuito programable de 3 estados de un bit, que sólo sirve como mecanismo bidireccional. Es esta implementación particular, la programación de las celdas no es individual, sino a través de una memoria de 8 bits, un bit por cada celda. Cada celda no es programable independientemente.
- Dos Matrices de Interconexión Local **LIM**. Una para interconectar 16 salidas a cualquiera de las 8 celdas de entrada-salida, y otra para interconectar 8 celdas a 4 entradas lógicas. Ninguna de estas matrices utiliza las entradas de realimentación, aunque su implementación lo permite, ya que están basadas en las **LIM** de los **LB**.

III. El uso de GASIM

El primer caso de uso de GASIM se realiza programando el FPGA completo que provee la librería, cuya organización interna se muestra en la figura 4. Programar este ensamblaje es, de inicio, una actividad muy interesante para los estudiantes. En el uso de un FPGA real, un desarrollador especifica una representación abstracta de la lógica en lenguajes como VHDL o VERILOG, y luego ejecuta un proceso de dos pasos, soportado por Software especializado: Síntesis e Implementación. la Síntesis consiste en obtener el esquema de circuito lógico a ser construido, independiente del dispositivo objetivo. La implementación consiste en el proceso de tomar ese circuito y generar una configuración posible para un dispositivo objetivo, por ejemplo, un chip FPGA de Xilinx o de Altera. La implementación en sí misma requiere varios pasos intermedios. Por ejemplo, para un proceso de implementación de Xilinx, tal como se explica en [22]:

1) **Traducción**: Reescribir el diseño Sintetizado, usando operadores específicos del dispositivo.
2) **Mapeo**: Mapear la lógica traducida en una implementada con los bloques físicos del FPGA objetivo.

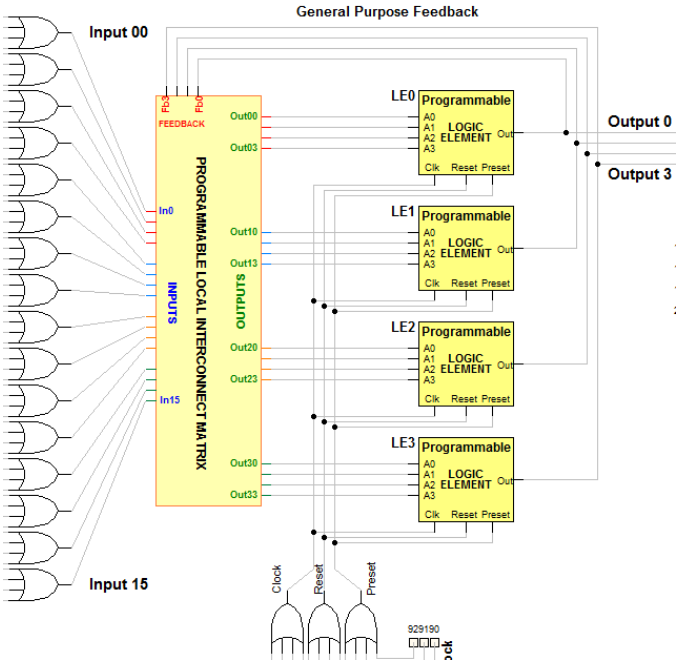

Fig. 6. Organización de un Bloque Lógico, mostrando la Matriz de Interconexión Local y 4 Elementos Lógicos.

3) **Colocar y enrutar (Place and Route)**: Seleccionar los bloques finales que se utilizarán y calcular las rutas de interconexión.
4) **Generación del archivo Bit**: Crear el archivo binario que contiene la configuración de los bloques involucrados en la implementación de acuerdo con el hardware y las interfases de programación del dispositivo objetivo, que permitan transferir la configuración al chip.

En GASIM, la programación es local para cada componente da alto nivel del FPGA. Para realizar esta programación, los estudiantes deben descubrir, o ejecutar manualmente los cuatro pasos de implementación del proceso. Deben **traducir** el diseño a uno realizable con los componentes de GASIM, tales como Tablas lógicas y flip-flops de los **LB**. Luego, deben **mapear** o ensamblar el diseño usando los bloques disponibles de GASIM. Luego, deben seleccionar el camino de los datos a través de las líneas, canales y bloques de interconexión **SB** y **CB**. Evidentemente los estudiantes no generan un archivo binario de programación, pero sí programan manualmente los bits de cada elemento involucrado, uno por uno.

Por ejemplo, programar el FPGA de GASIM para construir una función lógica de 4 bits, conectada a 4 switches como entradas y un LED como salida es una primera actividad suficientemente completa para los estudiantes. Tal ejemplo es mostrado en la figura 2, donde los switches y un led están conectados en la primera vista externa del chip. En este caso, el estudiante debe completar el siguiente procedimiento:

1) Diseñar la función lógica deseada, como una especificación de tabla de búsqueda.
2) Abrir un proyecto en blanco en Digital Works.
3) Tomar el dispositivo FPGA de la librería de GASIM.
4) Conectar periféricos al chip como sea necesario. En este

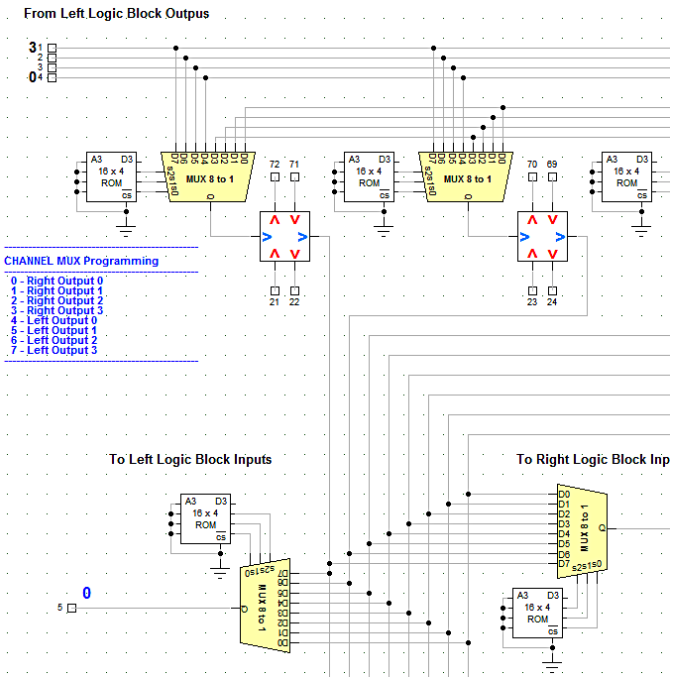

Fig. 7. Organización de una caja de Conexión, que muestra las mini-**CB** y los selectores de canal. En esta figura los canales son verticales y las conexiones lógicas de entrada-salida son horizontales.

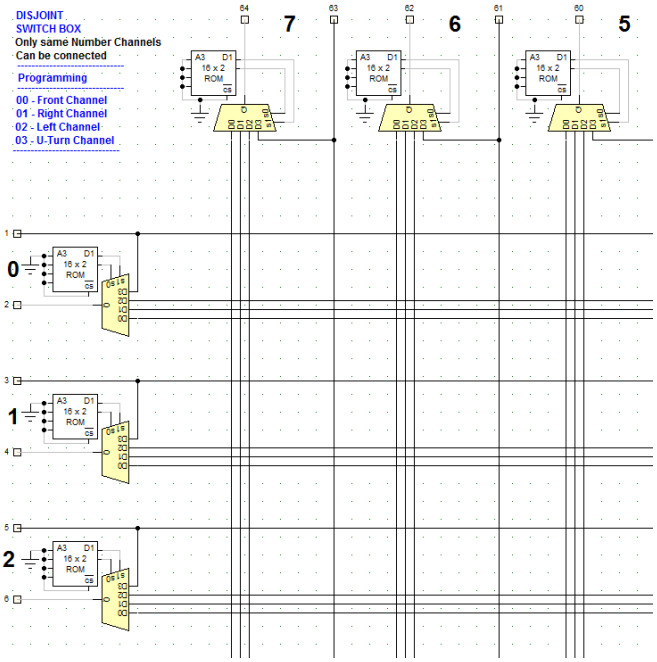

Fig. 8. Organización de una Caja de Interconexión Desarticulada **SB** en GASIM, mostrando el arreglo de selectores programables de 4 a 1.

ejemplo los periféricos son 4 botones nativos y un led nativo de Digital Works.

5) Diseñar el enrutamiento para el FPGA:
   a) Hacer doble click en el FPGA para ir dentro del chip.
   b) Decidir que bloques lógicos implementarán la función lógica. En este caso, una **LB** es suficiente.
   c) Determinar el enrutamiento, dadas las conexiones externas realizadas en el paso 3, y seleccionar los componentes necesarios para su programación.
6) Programación:
   a) Hacer doble click en la **LB** seleccionada para ir dentro del bloque y programar la tabla de búsqueda de la función lógica. Al terminar, salir del bloque.
   b) Hacer doble click en las **CB** y **SB** necesarias para el enrutamiento de las señales, y realizar la programación necesaria de elementos.
   c) Programar los bloques de entrada-salida **IOB** de acuerdo con la conexión externa de periféricos.
   d) Programar las señales generales como sea necesario: Clock, Reset and Preset. En este caso, no se requiere su uso pues es un circuito asíncrono.
7) Pruebas: los estudiantes pueden ejecutar la simulación de GASIM y observar al comportamiento del FPGA navegando por los bloques y elementos analizando como se muestra en la figura 2. Cada cambio en cada señal se puede ver con cambios de color que provee Digital Works. La interacción se realiza mediante los controles nativos de Digital works. Por ejemplo, una simulación puede ser tan simple como:
   a) Asegurar que el FPGA tiene todas las entradas.
   b) Ajustar la señal de Clock de Digital Works de acuerdo con la visibilidad necesaria.
   c) Navegar hasta el nivel mas superior y presionar el botón ”Play”.

A. *Construir un FPGA usando GASIM*

Usando GASIM, estudiantes y profesores pueden fácilmente construir dispositivos basados en arquitectura de malla utilizando los bloques y cajas de la librería. En el proceso de diseñar la apariencia 2D de GASIM, se desarrolló un estándar visual que puede ser seguido para construir un diseño propio. Junto con un conjunto de elementos programables, GASIM incluye plantillas visuales que pueden ser asignadas a circuitos diferentes para mantener compatibilidad. Por ejemplo, un estudiante puede construir un tipo diferente de bloque lógico, pero usar la plantilla visual del bloque lógico original para asegurar que las conexiones estándar se mantienen para la malla de GASIM. Esto ayuda al profesor a crear variaciones, debatir diseños y enriquecer la librería. Por ejemplo:

- Ensamblar un modelo específico de enrutamiento para FPGA utilizando los componentes de alto nivel que la librería provee. Por ejemplo, un FPGA con una estructura de malla no cuadrada.
- Diseñar o implementar un modelo específico de **LB**.
- Diseñar o implementar un modelo específico de **IOB**.
- Diseñar o implementar un modelo específico de **CB**.
- Diseñar o implementar un modelo específico de **SB**.

Es importante finalmente añadir que GASIM no debe reemplazar el proceso de aprendizaje en el uso y desarrollo con FPGA reales, y está diseñado solo como una herramienta de conceptualización y para ganar una comprensión profunda del diseño interno del chip.

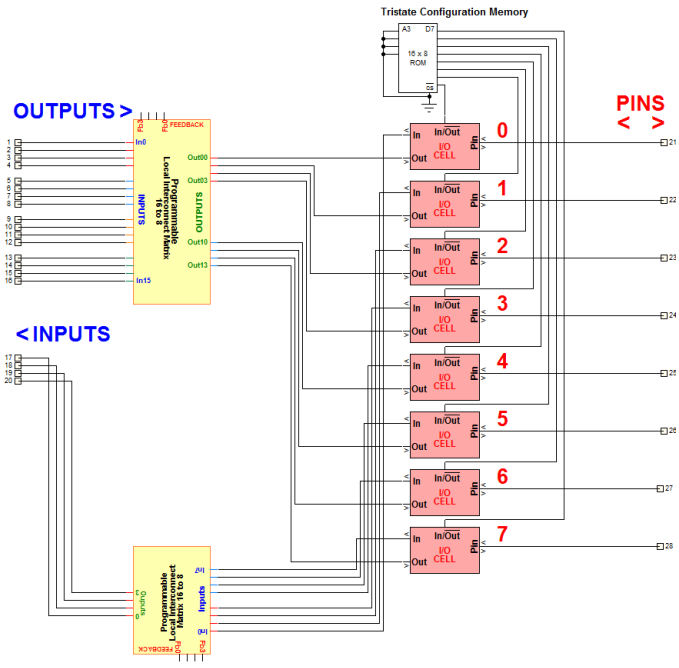

Fig. 9. Organización de un bloque de Entrada-Salida, mostrando las dos matrices de interconexión local, y una celda de entrada-salida programable para un pin externo.

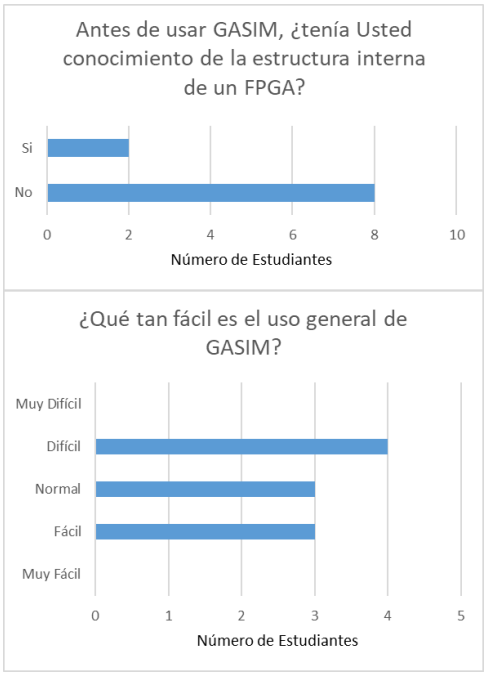

Fig. 10. Encuesta de percepción para una muestra de 10 estudiantes. Experiencia previa en el uso de FPGA y usabilidad general de GASIM.

IV. Resultados

GASIM puede ser encontrado en su repositorio github[4]. Ha sido utilizado en cursos de Circuitos Digitales y Diseño Digital de pre-grado en ingeniería electrónica con un total de 47 estudiantes, en talleres y actividades de 3 horas de duración con excelentes resultados. Nuestros estudiantes han trabajado programando el FPGA ensamblado por defecto en GASIM, y también han programado y probado independiente los bloques de alto nivel para tareas específicas.

En cada sesión, la reacción inicial era de sorpresa ante la gran complejidad de los circuitos digitales que componen el FPGA, pero luego de la tradicional clase teórica de nomenclatura y funcionamiento de los componentes, seguido de un ejemplo corto realizado por el instructor, los estudiantes tomaron la iniciativa con gran entusiasmo y finalizaron la actividad antes de las 3 horas. La recepción general fue positiva y significativa, pues (en sus propias palabras) ”Está construido con los mismos elementos que conocíamos de clases anteriores”. Los estudiantes además encontraron muy intuitiva la navegación top-down-top que Digital Works les permitía en GASIM. Luego de haber sido utilizado en dos cursos trimestrales, en los que no se tomó la previsión de utilizar instrumentos de medición de percepción, se establecieron sesiones de trabajo con encuestas en un tercer curso de 10 estudiantes para evaluar la percepción, usando las preguntas mostradas en las figuras 10 a 13. El grupo de muestra consistió en estudiantes con experiencia previa en FPGA. La figura 10 muestra que el 80% del grupo no tenía conocimiento sobre la estructura interna de un FPGA, aunque todos los han usado. Cuando se preguntó sobre el uso general de GASIM, el 40% percibió dificultad de uso, 30% contestó facilidad regular, y el otro 30% respondió que el uso había sido fácil. Este resultado era esperado, dada la complejidad intrínseca del tema en estudio. Sin embargo, cuando se preguntó acerca de la utilidad del simulador para comprender el funcionamiento del FPGA (figura 11), 70% percibió a GASIM como ”útil” o ”muy útil”, indicando la importancia para ellos del uso de un tipo de herramienta como la creada. La representación lógica del FPGA fue también bien recibida, con 100% de respuestas entre ”regular” y ”Excelente”. Una pregunta específica sobre la calidad estética del simulador también fue aplicada, resultando en su mayoría una buena percepción. Acerca de la tarea específica de programación de los componentes del FPGA, 70% percibió la experiencia como positiva (figura 12)

Finalmente, el grupo fue encuestado sobre su propia percepción sobre la calidad del aprendizaje obtenido usando GASIM (figura 13). ¿Cuánto piensan que aprendieron usando GASIM? 60% respondieron ”mucho”, 30% percibió que aprendió una cantidad regular de conocimientos, y solo un 10% percibió no haber aprendido suficiente con la herramienta. La última pregunta pidió sugerencias sobre el lugar que, actividades utilizando GASIM, deberían tener en el proceso de aprendizaje de los FPGA. ¿Antes, durante o Después del aprendizaje en el uso de un FPGA? Solo 20% respondió ”después” del uso de un FPGA. El resto de los estudiantes prefirieron que se aplicara GASIM antes o como introducción al estudio del uso de un chip FPGA.

Adicionalmente, fue obtenida realimentación informal de estudiantes más avanzados soportados por tecnología FPGA. Durante el desarrollo posterior de aplicaciones basadas en FPGA, los estudiantes utilizaron el conocimiento obtenido con GASIM para entender los problemas de síntesis y de

[4]https://github.com/odcastro/gasim

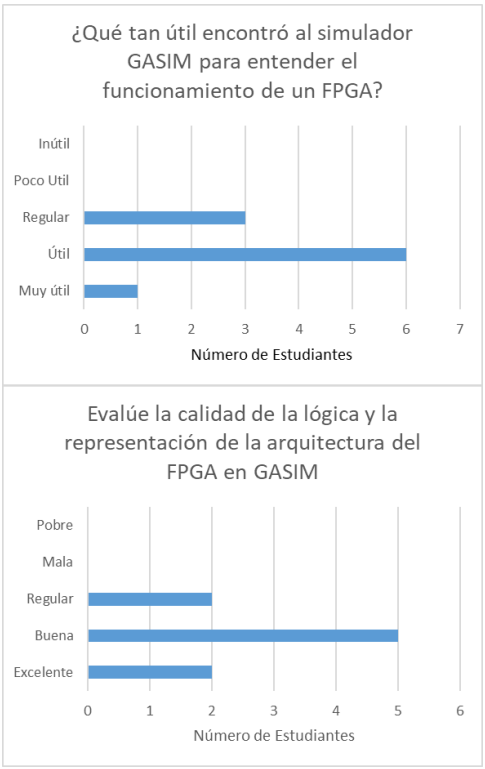

Fig. 11. Encuesta de Percepción para una muestra de 10 estudiantes. Utilidad y Calidad lógica de GASIM.

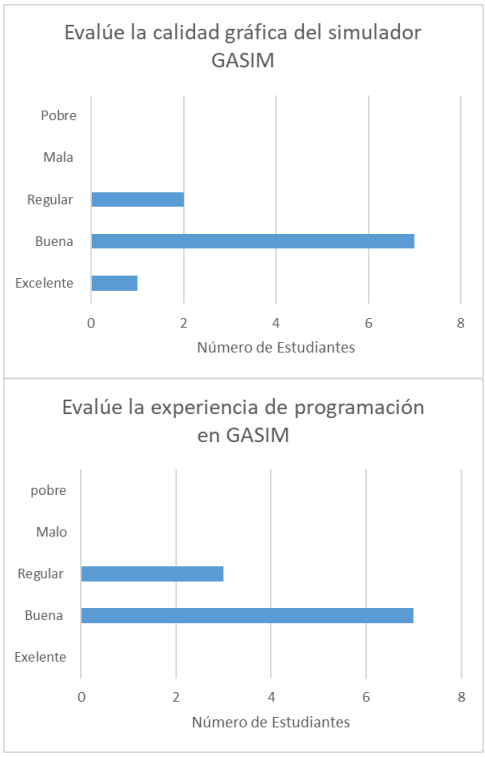

Fig. 12. Encuesta de Percepción para una muestra de 10 estudiantes. Percepción estética y de experiencia de programación en GASIM.

implementación que se les presentaban durante el uso de las herramientas de desarrollo. Por primera vez en nuestros cursos avanzados, nuestros estudiantes tenían gran claridad teórico-práctica en el uso correcto de un chip FPGA, comparados con generaciones anteriores de estudiantes que no entendían gran parte del proceso de desarrollo, por falta de conocimiento del chip.

V. Trabajo Futuro

Existen tres líneas de trabajo en desarrollo en este momento relacionadas con GASIM:

- Implementación de modelos adicionales de componentes de alto nivel de FPGA, tales como bloques de memoria dedicados, multiplicadores y otros bloques de cálculo, típicos de arquitecturas modernas de FPGA.
- Implementación de modelos alternativos de **SB** con más capacidad de interconexión, para la construcción de otros modelos de enrutamiento.
- Creación de una herramienta de simulación moderna que sustituya a Digital Works, que, siendo una excelente herramienta para la implementación de GASIM, tiene limitaciones como es en el caso de las herramientas de 3 estados y en el manejo de circuitos de gran tamaño. Una nueva herramienta podría ser construida en base a las excelentes características gráficas de Digital Works.
- Desarrollo de experiencias de laboratorio para el diseño avanzado de estructuras y modelos de FPGA para cursos de postgrado.

VI. Conclusión

Este artículo presenta la primera arquitectura simple de simulador interactivo de FPGA 2D, basado en el modelo de tipo de malla, con completitud lógica y estructural llamada GASIM, para ser utilizado en un aula de clases. La simplicidad y fácil interacción con la herramienta es la contribución de este trabajo, como se muestra en la sección de resultados.

Herramientas simples de simulación, cuando se utilizan con suficiente profundidad, cuidado visual y capacidad interactiva, pueden tener impacto en la experiencia del aprendizaje de sistemas complejos. Tal vez se pudiera añadir que, en el futuro, sería útil introducir la organización y arquitectura del FPGA como tema del currículo de ingeniería electrónica, dado el contexto adecuado.

Nuestros primeros resultados nos llevan a concluir que, incluso en circuitos complejos como GASIM, una experiencia de construcción de FPGA puede tener una gran influencia en el desempeño de futuros desarrolladores y diseñadores de aplicaciones basadas en FPGA. La curva de aprendizaje del uso y desarrollo con FPGA será más pronunciada y la brecha conceptual existente se resolverá hasta cierto compromiso. De cualquier forma, GASIM tiene su propio lugar en el proceso de aprendizaje en el aula y nunca reemplazará el uso de herramientas reales y profesionales en la universidad. Herramientas como Quartus, Vivado, ISE Design Suite y otras herramientas de desarrollo profesional deben ser utilizadas para el entrenamiento de los estudiantes. Finalmente, concluimos que una herramienta rápida e interactiva para circuitos complejos como los FPGA permiten el tiempo necesario para el desarrollo del pensamiento de diseño y discusiones teóricas

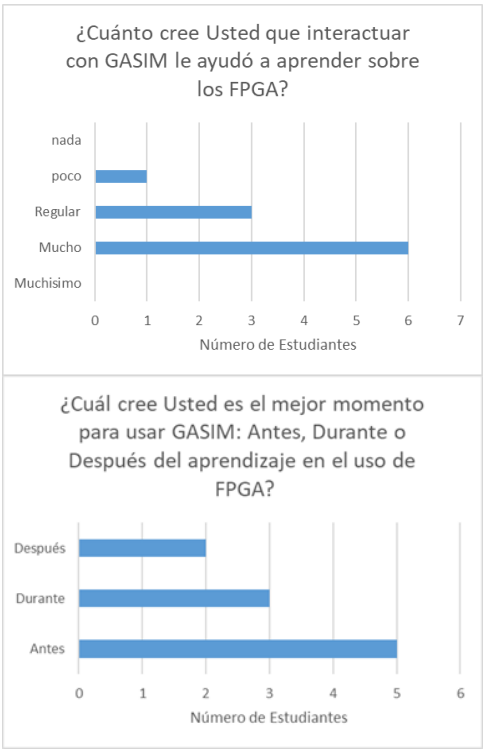

Fig. 13. Encuesta de Percepción para una muestra de 10 estudiantes. Percepción de mejor uso e interacción.

relevantes de forma eficiente, ya que mucho del tiempo que se usa normalmente para enfrentar las pequeñas pero numerosas sutilezas en el uso de las herramientas profesionales, en este caso no existen. La noción de paralelismo y sincronización en el diseño digital usando VHDL y Verilog pueden ser discutidos y resueltos con los estudiantes en un ambiente más temprano, antes de enfrentar la aplicación real.

REFERENCES

[1] J. Gray, "Hands-on Computer Architecture - Teaching Processor and Integrated Systems Design with FPGAs," in *in Workshop on Computer Architecture Education*, 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.2914

[2] J. Olivares, J. M. Palomares, J. M. Soto, and J. C. Gámez, "Teaching microprocessors design using FPGAs," in *2010 IEEE Education Engineering Conference, EDUCON 2010*, 2010, pp. 1189–1193.

[3] M. Holland, J. Harris, and S. Hauck, "Harnessing FPGAs for computer architecture education," in *Proceedings - 2003 IEEE International Conference on Microelectronic Systems Education: Educating Tomorrow's Microsystems Designers, MSE 2003*, 2003, pp. 12–13.

[4] C. Ttofis, T. Theocharides, and M. K. Michael, "FPGA-based laboratory assignments for NoC-based manycore systems," *IEEE Transactions on Education*, vol. 55, no. 2, pp. 180–189, 2012.

[5] P. Bulić, V. Guštin, D. Šonc, and A. Štrancar, "An FPGA-based integrated environment for computer architecture," *Computer Applications in Engineering Education*, vol. 21, no. 1, pp. 26–35, 2013.

[6] U. Farooq, Z. Marrakchi, and H. Mehrez, "FPGA Architectures: An Overview," in *Tree-based Heterogeneous FPGA Architectures*, 2012.

[7] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 101 –1029, 1993.

[8] Vaughn Betz, Jonathan Rose, and Alexander Marquardt, *Architecture and CAD for Deep-Submicron FPGAS*. New York: Springer US, 1999.

[9] IEEE, *IEEE Standard for Verilog*. IEEE publications, 2006.

[10] D. Automation, S. Committee, and I. Computer, *IEEE Standard VHDL Language Reference Manual*. IEEE Publications, 1993.

[11] S. Brown and J. Rose, "Architecture of FPGAs and CPLDs: A tutorial," *IEEE Design and Test of Computers*, 1996.

[12] S. Brown and Z. Vranesic, "Fundamentals of digital logic with VHDL design," p. 892, 2005. [Online]. Available: http://courses.ee.sun.ac.za/Digital_Systems_144/SS144Ch1+2.pdf

[13] J. F. Wakerly and J. F., *Digital design : principles and practices*. Prentice Hall, 2001. [Online]. Available: https://dl.acm.org/citation.cfm?id=517759

[14] M. Scott, "WinMIPS64," 2012. [Online]. Available: http://indigo.ie/~mscott/

[15] M. Brorsson, "MipsIt - A Simulation and Development Environment Using Animation for Computer Architecture Education," *Proceedings of the 2002 workshop on Computer architecture education*, pp. 65–72, 2002.

[16] K. Vollmar and P. Sanderson, "MARS: An Education-oriented MIPS Assembly Language Simulator," in *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '06. New York, NY, USA: ACM, 2006, pp. 239–243. [Online]. Available: http://doi.acm.org/10.1145/1121341.1121415

[17] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design, 4th Ed, D. A. Patterson and J. L. Hennessy.pdf*, 2009, vol. 4th, no. 0.

[18] A. S. Tanenbaum, *Structured Computer Organization (5th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2005.

[19] O. De Castro, C. Murrugarra, and J. Sepúlveda, "USBSIM08: Graphic Simulator for Active and Meaningful Learning of Computer Architectures," in *Memorias del Congreso Internacional de Tecnología, Ingeniería e Innovación (CITII 2017)*. Bogotá D.C.: Editorial Bonaventuriana, 2017.

[20] Mecanique, "Digital Works," 2013. [Online]. Available: https://www.mecanique.co.uk/shop/index.php?route=product/category&path=89

[21] C. Burch and Carl, "Logisim: a graphical system for logic circuit design and simulation," *Journal of Educational Resources in Computing*, vol. 2, no. 1, pp. 5–16, 3 2002. [Online]. Available: http://portal.acm.org/citation.cfm?doid=545197.545199

[22] Xilinx Inc., "Implementation Overview for FPGAs," p. 1, 2009. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/ise_c_implement_fpga_design.htm

Osberth De Castro Es Ingeniero Electrónico de la Universidad Nacional Experimental Politécnica Antonio José de Sucre, en Puerto Ordaz, Venezuela, de 1997, y Magíster en Ciencias de la Computación de la Universidad Simón Bolívar, de Caracas, Venezuela, de 2002. Actualmente trabaja en la Universidad De San Buenaventura, sede Bogotá, en Bogotá, Colombia. Sus intereses de investigación van desde Machine Learning, Sistemas para salud y educación, y Architecturas de computación.

Cecilia Murrugarra Es Ingeniero Electrónico de la Universidad Nacional Experimental Politécnica Antonio José de Sucre, en Puerto Ordaz, Venezuela, de 1997, y Doctor en Ingeniería de la Universidad Simón Bolívar, de Caracas, Venezuela, de 2009. Actualmente trabaja en la Universidad El Bosque, en Bogotá, Colombia. Sus intereses de investigación incluyen la robótica, aplicaciones en la salud y educación, y Arquitecturas de procesamiento y computación.