Algoritmo de Auto-generación de Bloques Aritméticos Sumadores y Multiplicadores para su Implementación en PLD's

Edgar A. VARGAS, Alberto M. OCHOA
Departamento de Electrónica, Universidad de Colima. C.P. 28400. Coquimatlán (Colima), México.
E-mail: aochoa@ucol.mx

RESUMEN. En este trabajo se presenta el desarrollo de Sumadores y Multiplicadores aritméticos utilizando un algoritmo de auto-generación de bloques. El algoritmo consiste en un pseudocódigo capaz de reconfigurar la estructura de los bloques aritméticos, generando una mayor o menor cantidad de las unidades básicas (FA y HA), únicamente modificando un parámetro especifico que determina el tamaño de los sumadores y multiplicadores de *n* bits. En el trabajo se muestra el diseño e implementación en VHDL de Sumadores (Ripple Carry, Carry by Pass y Carry Select) y Multiplicadores (Array, Carry Save, Árbol de Wallace) de tamaño 2^n bits donde $n = \{2,3,4,5,6\}$. Los resultados obtenidos de la síntesis y simulación de todos los sumadores y multiplicadores permiten realizar un análisis respecto al espacio en hardware y el retardo total de cada módulo. Los resultados permitieron identificar las dos mejores arquitecturas (Sumador Carry by Pass y Multiplicador Carry Save) que ofrecen el mejor rendimiento en PLD's.

Palabras Clave: Algoritmo de autogeneración, VHDL, PLD, Bloques Aritméticos, Multiplicadores, Sumadores.

1. INTRODUCCIÓN

Dentro de las operaciones aritméticas fundamentales se encuentran la suma, resta, multiplicación y división. De las cuales, la operación aritmética básica es la suma, de ella se deriva la resta además de ser parte del procedimiento de multiplicación y división. A partir de estas cuatro operaciones se pueden realizar procesos matemáticos complejos, los cuales son esenciales en el desarrollo de las distintas áreas de conocimiento tecnológico.

En el área de Diseño Digital Electrónico, el desarrollo de circuitos integrados de aplicación especifica (ASIC's), permite producir sistemas digitales capaces de realizar cálculos de alta complejidad. Pero al igual que en el proceso matemático, los sistemas digitales son creados a partir de bloques fundamentales que realizan alguna de las cuatro operaciones básicas. Hay una gran variedad de trabajos referentes al diseño e implementación de distintas arquitecturas de sumadores y multiplicadores, que proponen métodos para la reducción de retardo en la respuesta, utilización de espacio al trasladarlos a hardware, complejidad en el diseño, etc [1][2][3].Todas estas propuestas presentan resultados satisfactorios, mostrando métodos eficaces en la optimización de las características mencionadas. Sin embargo, la mayoría de los casos muestran el desarrollo de bloques aritméticos definidos para una cantidad específica de bits [4][5][6], lo cual resulta inconveniente en sistemas que requieren procesamiento de datos con número variable de bits.

En este trabajo se presenta el desarrollo de un algoritmo para la auto-generación de los bloques aritméticos utilizando bloques simples o fundamentales, en este caso, sumadores bit a bit: Half Adders (HA) y Full Adders (FA). El objetivo del algoritmo mejorar la capacidad de diseñar e implementar sumadores y multiplicadores mediante la auto-generación de bloques fundamentales. El algoritmo propuesto utiliza un pseudocódigo capaz de reestructurar el tamaño de los sumadores y multiplicadores con la modificación de un parámetro, el cual genera una mayor o menor cantidad de los bloques que integran cada arquitectura aritmética según la necesidad del sistema. El modelado algorítmico de sumadores y multiplicadores aritméticos genera arquitecturas de distintos tamaños, iniciando con bloques aritméticos de dos bits, aumentando el tamaño de los bloques de forma exponencial (2^n para $n = \{2,3,4,5,6\}$), finalizando con 64 bits debido a la capacidad de la FPGA (Spartan 3E de 500,000 compuertas). La utilización de este método para la auto-generación de bloques, podrá simplificar el desarrollo de otros módulos de procesamiento de aritmética computacional, tales como: multiplicadores-acumuladores (MAC), filtros digitales (FIR e IIR), módulos CORDIC, etc.

Las arquitecturas más utilizadas de sumadores son: Ripple Carry, Carry by Pass, Carry Select y Carry Lookahead. Por parte de los multiplicadores son: Array, Carry Save, Radix-4, Árbol de Wallace y Serie. El resultado de la implementación de las distintas arquitecturas analizadas permite identificar los dos mejores bloques aritméticos: el Sumador ``Carry by Pass" y el Multiplicador ``Carry Save", ambas son las mejores estructuras en cada una de sus ramas. La determinación de estas características llevo a cabo a través de un estudio comparativo que involucra área y retardo de la propagación de las señales. Las arquitecturas desarrolladas en base al algoritmo de auto-generación pueden complementar el uso de multiplicadores embebidos en PLD's que tienen un limitado o nulo número de ellos en los dispositivos actuales.

El resto del artículo está compuesto de la siguiente manera. En la sección 2, se da una breve descripción de las distintas arquitecturas de sumadores y multiplicadores desarrolladas en este trabajo, sus características, principio de funcionamiento y diferencias entre ellos. Posteriormente en la sección 3 se presenta el desarrollo en VHDL, resultados de simulaciones y esquemas de espacio a utilizar al trasladar los bloques aritméticos a una FPGA. Finalmente con los resultados obtenidos de la síntesis y simulación de cada arquitectura aritmética, se muestra en la sección 4 el análisis comparativo de los factores de retardo en la respuesta y espacio a ocupar en hardware por parte de los multiplicadores y sumadores respectivamente.

2. TIPOS DE SUMADORES Y MULTIPLICADORES

La operación aritmética básica es la suma de dos dígitos binarios. Esta operación consiste en cuatro sumas elementales posibles: $0 + 0 = 0,\ 0 + 1 = 1,\ 1 + 0 = 1$ y $1 + 1 = 10$. Las

primeras tres operaciones producen un resultado cuya longitud es en un dígito, pero en el caso de que ambos sumandos sean iguales a 1, la longitud del resultado consiste en dos dígitos [7]. A partir de la suma de dos dígitos binarios es posible desarrollar sumadores aritméticos para una mayor cantidad de bits, además de bloques aritméticos más elaborados, tales como los multiplicadores. Al igual que en el desarrollo manual de las operaciones aritméticas, la implementación electrónica debe iniciarse con la realización de módulos simples, a partir de los cuales se forman sumadores y multiplicadores para la resolución de operaciones complejas. En esta sección se analizan los Half y Full Adders que dan paso a distintas arquitecturas de sumadores y multiplicadores aritméticos basados en estos dos módulos fundamentales.

Sumador Medio (HA)

El HA es un circuito que realiza la suma de dos bits, su estructura es mostrada en la Figura 1a. Este circuito consta de dos entradas X y Y y dos salidas S y C (salida y bit de acarreo respectivamente). Las funciones lógicas para la salida S es $S = x'y + xy'$ y el bit de acarreo C es $C = xy$.

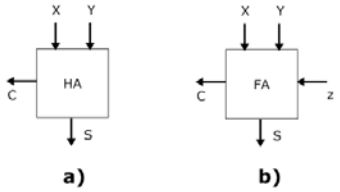

Figura 1.- Esquemas de Sumadores a) HA y b) FA.

Sumador Completo (FA)

Un FA es un circuito combinacional que forma la suma aritmética de tres bits de entrada. Este consiste en tres entradas y dos salidas como lo muestra la Figura 1b. Dos de las variables de entrada denotadas por X y Y representan los dos bits significativos que se suman. La tercera entrada z representa el bit de arrastre de la posición previa menos significativa. Se necesitan dos salidas porque la suma aritmética de tres dígitos binarios varía en valor de 0 a 3 y los números binarios 2 o 3 necesitan dos dígitos. Las funciones lógicas en el caso del FA para la salida S es $S = x'y'z + x'yz' + xy'z' + xyz$ y para el bit de acarreo C es $C = xy + xz + yz$. A partir de estos sumadores básicos se pueden construir bloques más elaborados como sumadores y multiplicadores aritméticos que permiten efectuar operaciones con mayor cantidad de bits ó operaciones más complejas.

Sumadores Aritméticos

Con los bloques HA y FA se pueden implementar sumadores y multiplicadores para números de n bits, construidos a partir de la conexión de dos o más de estas unidades básicas de procesamiento. En base a su arquitectura, existen distintos tipos de sumadores para n bits, tales como: Ripple Carry, Carry by Pass y Carry Select. A continuación se da una breve descripción de cada uno de estos sumadores, desde su definición, características, arquitectura, etc.

Sumador Ripple Carry: Consiste en n FA conectados en cascada, con la salida de acarreo (C_i) de un FA conectado al acarreo de entrada (Z_{i+1}) del siguiente FA, como se muestra en la Figura 2.

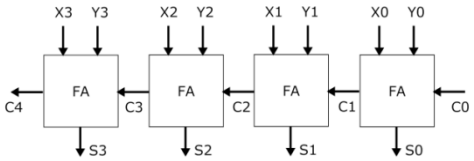

Figura 2.- Esquema de un Sumador Ripple Carry.

Un sumador de n bits requiere n FA como lo muestra el esquema de la Figura 2. Si no existe la necesidad de tener un acarreo de entrada, el FA de la posición inicial puede ser reemplazado por un HA, lo que permite reducir el tamaño en hardware del sumador.

Sumador Carry by Pass: Es una implementación que mejora el retardo de un sumador Ripple Carry. El sumador Carry by Pass se basa en el principio de buscar un atajo en el camino de propagación del acarreo, para lo cual, es necesario conocer la forma en que se genera (G) o propaga (P) el acarreo. En la Tabla I, se indica lo que sucede cuando se suman dos bits.

Tabla I.- Generación y Propagación del Acarreo.

A_i	B_i	C_i	C_{i+1}
0	0	0/1	0 (no se propaga **D**elete)
0	1	0/1	0/1 (se propaga **P**ropagate)
1	0	0/1	0/1 (se propaga **P**ropagate)
1	1	0/1	1 (se genera **G**enerate)

En la Tabla I se muestran los casos en que el bit de acarreo de entrada (C_{in}) se borra, se propaga o se genera hacia el acarreo de salida (C_{out}), dependiendo de los bits de entrada. Con esta tabla de verdad se pueden deducir las ecuaciones para la propagación ($P = A \oplus B$) y para la generación ($G = A \cdot B$) del bit de acarreo. El esquema del sumador Carry by Pass se muestra en la Figura 3. Si $P_0P_1P_2P_3 = 1$, entonces $C_{out} = C_0$, de lo contrario, C_{out} es independiente de C_0.

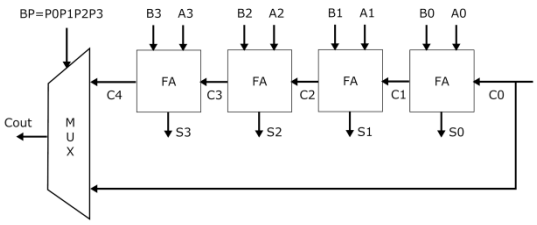

Figura 3.- Esquema de un sumador Carry by Pass.

Sumador Carry Select: Consiste en dos sumadores Ripple Carry de n bits y un multiplexor. En un Carry Select Adder (Véase Figura 4), la suma de dos números de n bits es realizada con dos sumadores (Ripple Carry) a la vez, esto con el fin de realizar doble suma, la primera suma con el acarreo C_{in} de 0 y la otra suma con acarreo de entrada igual a 1. Una vez que el acarreo correcto (C_{in}) es conocido, se selecciona el resultado correcto de suma (S) y acarreo (C_{out}) con el multiplexor.

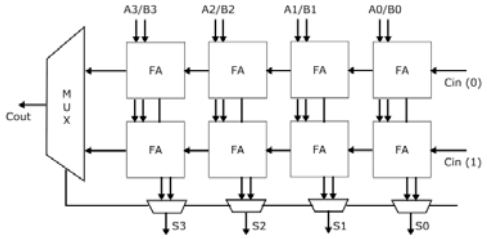

Figura 4.- Esquema de un Sumador Carry Select.

Multiplicadores Aritméticos

Los multiplicadores son matrices de sumadores (FA y HA), donde se consideran dos números binarios sin signo X y Y de N bits respectivamente, el producto de los dos genera un número Z de $2 \times N$ bits. Como en el procedimiento manual de multiplicación, para la implementación digital es necesario obtener los productos parciales. Para ello, se utilizan compuertas AND como se muestra en la Figura 5. Los productos parciales (PP) se obtienen de aplicar la operación lógica AND al multiplicando X y a un bit del multiplicador Y (y_j). Cada fila de la matriz es una copia del multiplicando o una fila de ceros. La generación de n productos parciales requiere $2 \times N$ compuertas AND de 2 bits.

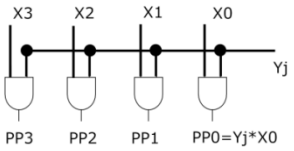

Figura 5.- Arreglo de AND's para la obtención de los PP's.

Posteriormente se utilizan HA y FA para la suma de dichos productos parciales. Como se observa en la Figura 6, el producto se obtiene multiplicando (función AND) bit a bit los operandos X, Y y sumando los resultados parciales.

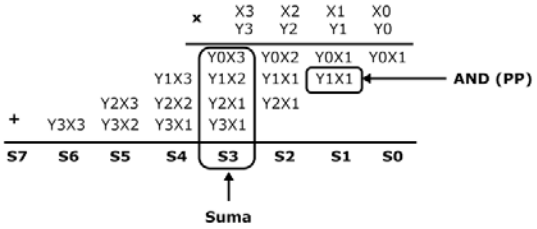

Figura 6.- Procedimiento de Multiplicación.

Dependiendo de la arquitectura en la organización que puedan tener los HA y FA, existen varios tipos de multiplicadores aritméticos. En este trabajo se analizaron las arquitecturas de multiplicadores: Array, Carry Save y Árbol de Wallace.

Multiplicador en Array: La arquitectura de este multiplicador consiste en Sumadores Ripple Carry conectados en cascada, con la variación de que el bloque inicial FA de cada Ripple Carry es sustituido por un HA con el objetivo de reducir el espacio en hardware. La mayor parte del área del multiplicador está dedicada a la suma de los N PP's, que requiere $N-1$ sumadores de n bits. En la Figura 7, se presenta un multiplicador en Array de 4 bits, en donde se observa que se requieren $N-1=3$ sumadores Ripple Carry de $n=4$ bits para realizar la multiplicación.

Multiplicador Carry Save: La arquitectura del multiplicador Carry Save es similar a la de un multiplicador en Array, con la diferencia que la propagación del acarreo entre HA y FA se realiza diagonalmente hacia abajo, en lugar de hacia la izquierda (Véase Figura 8). El objetivo de esta arquitectura es reducir el retardo máximo del sistema mejorando la propagación del acarreo.

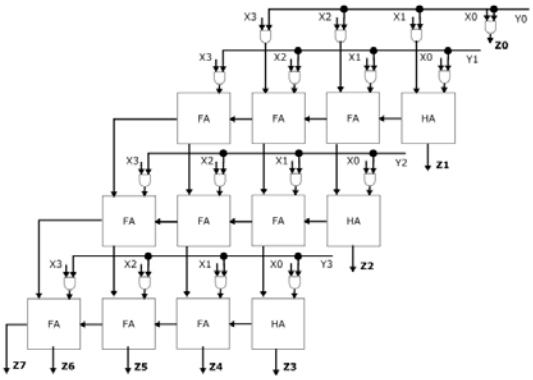

Figura 7.- Esquema de un Multiplicador Array.

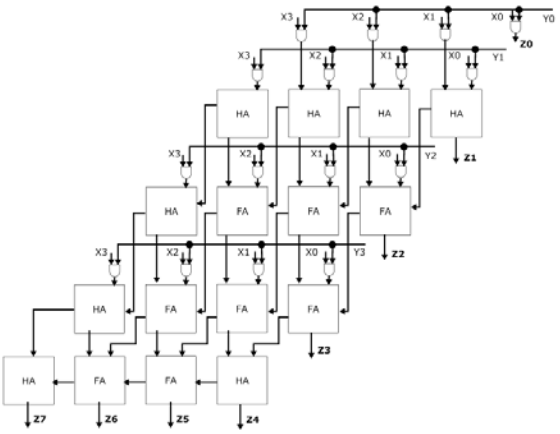

Figura 8.- Esquema de un Multiplicador Carry Save.

Multiplicador Árbol de Wallace: Un árbol de wallace es una forma de organizar sumadores Carry Save para tratar de mejorar el espacio (véase Figura 9). Por lo tanto este Multiplicador organiza los sumadores de tal forma que se mejora la velocidad de procesamiento.

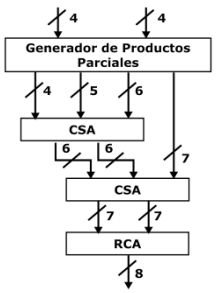

Figura 9.- Esquema de un Multiplicador en Árbol de Wallace.

3. DESCRIPCIÓN DEL ALGORITMO DE AUTO-GENERACIÓN

En esta sección se da una breve descripción del código para el algoritmo de auto-generación de bloques, utilizado para implementar los sumadores y multiplicadores aritméticos para n bits. Como ejemplo se presenta el pseudocódigo en VHDL del Sumador Ripple Carry de la Figura 2, el cual consta de n FA conectados en serie a través de las señales de acarreo (C_{in} y C_{out}) correspondientes a la entrada y salida de cada FA. El algoritmo reconfigura la estructura de los sumadores o multiplicadores mediante la modificación del número de bits (n_bits), generando n bloques fundamentales (FA y HA).

Este parámetro se define dentro de la declaración de entidad del sumador o multiplicador de la siguiente manera:

```
Generic(n_bits: integer:= 4);
```

En el caso del sumador Ripple Carry, utiliza un solo bloque fundamental (FA) y la declaración de entidad de este bloque se describe en el siguiente pseudocódigo, esta declaración de componente corresponde al bloque de la Figura 1b y realiza las funciones descritas en sus ecuaciones correspondientes.

```
COMPONENT fulladder
        PORT(    x, y, Cin : IN std_logic;
                 Cout, Sal : OUT std_logic);
END COMPONENT;
```

Para generar la cantidad de FA definidos por el parámetro n_bits se utiliza un pseudocódigo, en el cual mediante la utilización de un ciclo se producen los bloques necesarios e interconectando la salida de acarreo de un FA a la entrada de acarreo del siguiente FA. Este parte del pseudocódigo descrito a continuación, va colocado dentro de la definición arquitectural del sumador aritmético.

```
RCA: for i in 0 to (n_bits-1) generate
        FA: fulladder PORT MAP(
        x => A(i), y => B(i), Cin => Ci, Cout =>
        vectcarry(i),Sal => sal(i));
end generate;
```

Este algoritmo es igual para las demás arquitecturas de sumadores y multiplicadores, agregando únicamente los componentes que se necesiten para dicha arquitectura y modificando el tamaño de n_bits al valor deseado.

4. DISEÑO E IMPLEMENTACIÓN EN VHDL DE BLOQUES ARITMÉTICOS.

Las arquitecturas de sumadores y multiplicadores aritméticos mostrados en las secciones anteriores, se implementaron mediante lenguaje de descripción de hardware VHDL utilizando el algoritmo para la auto-generación de bloques. Para la síntesis y simulación de los bloques aritméticos se utilizó el software ISE 10.1 de Xilinx. El algoritmo de auto-generación, permite modelar los sumadores y multiplicadores de 2^n bits para $n = \{2,3,4,5,6\}$, con el fin de observar las tendencias de cada bloque aritmético.

Simulación de Sumadores

Para la simulación es necesario introducir valores a las entradas de los sumadores y así obtener el resultado de la operación aritmética. Se colocaron las siguientes tres combinaciones sin importar el número de bits en las entradas de los bloques: unos y ceros alternadamente (10101….10); mitad unos y mitad ceros (111…000) y solo unos (111 … 11) para facilitar las simulaciones. La Figura 10, muestra la simulación de un sumador de ocho bits. El resultado muestra que en la salida se produce el mismo número de bits que el tamaño de los datos de entrada y un bit extra de acarreo C_{out}, en caso de que los valores de entrada sean lo suficientemente grandes para producir un desbordamiento.

Como se ha mencionado a lo largo de este trabajo, se pueden generar fácilmente sumadores para cualquier cantidad de bits, únicamente modificando un parámetro en el pseudocódigo del algoritmo. Esto permite realizar simulaciones de cualquier tamaño de datos de entrada para las tres arquitecturas de sumadores, sin embargo, el tamaño de los datos de entrada de cada bloque aritmético, dependerá de las capacidades en el número de compuertas lógicas con los que cuente la tarjeta a utilizar.

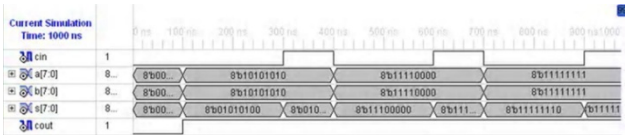

Figura 10.- Simulación Sumador de 8 bits.

Espacio en Hardware de Sumadores

Una vez que se realiza la síntesis y simulación de los circuitos sumadores, se obtiene una vista del diseño de colocación en la tarjeta, dependiendo del número de bits de entrada será la cantidad de bloques que se generan, por lo tanto, el número de bloques fundamentales de una FPGA (slice) utilizados aumenta o disminuye de acuerdo al tamaño del multiplicador.

En la Figura 11, se muestra el Sumador Carry by Pass con 64 bits en el tamaño de las entradas. Al sintetizar el circuito, se genera el espacio utilizado por el Sumador dentro de la FPGA, correspondiente a las partes obscuras del esquema de la tarjeta.

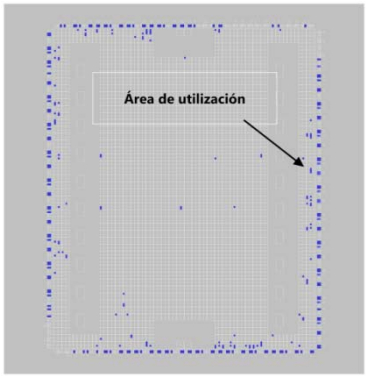

Figura 11.- Espacio en HW de Sumador Carry by Pass de 64 bits.

Cabe mencionar que se utilizaron los sumadores acotados hasta 64 bits de entrada, debido a que el número de bloques de entrada/salida (IOB's) utilizados por un sumador de tamaño mayor, requiere más cantidad de IOB's que los disponibles en una tarjeta Spartan 3E de 500 000 compuertas. Además en esta Figura se puede observar que el espacio total utilizado por el sumador es muy bajo y son pocos los slices ocupados por él.

Simulación de Multiplicadores

Para la simulación de multiplicadores se introdujeron igualmente valores a las entradas para observar la respuesta de una operación de multiplicación. Se colocaron las siguientes tres combinaciones sin importar el número de bits en las entradas de los bloques: unos y ceros alternadamente (10101….10); mitad unos y mitad ceros (111…000) y solo unos (111..11), esto con la finalidad de facilitar las simulaciones. La Figura 12, muestra la simulación de un multiplicador en Array de 4 bits. En ésta se presenta el resultado de tres multiplicaciones hechas con diferentes valores de entrada, cada resultado produce 8 bits debido a que, como se citó anteriormente, se genera un producto Z de $2 \times N = 2 \times 4 = 8$ bits. Las mismas simulaciones se realizaron para los multiplicadores Carry Save y Árbol de Wallace, cada uno con los distintos bits de entrada y sus combinaciones.

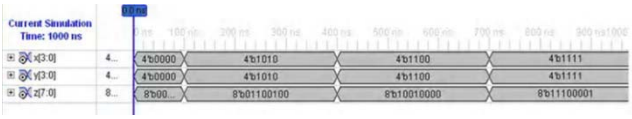

Figura 12.- Simulación Multiplicador Array de 4 bits.

Espacio en Hardware de Multiplicadores

Una vez que se realiza la síntesis y simulación de los circuitos multiplicadores, se obtiene una vista del diseño de colocación en la tarjeta, dependiendo del número de bits de entrada será la cantidad de bloques que se generan, por lo tanto, el número de slices utilizados aumenta o disminuye de acuerdo al tamaño del multiplicador. En la Figura 13 se muestra el multiplicador en Array de 32 bits en el tamaño de las entradas. Al sintetizar el circuito, se generan solamente 17 slices, por lo tanto, el área ocupada por el multiplicador Array de 4 bits es menor al 1\% del total del dispositivo.

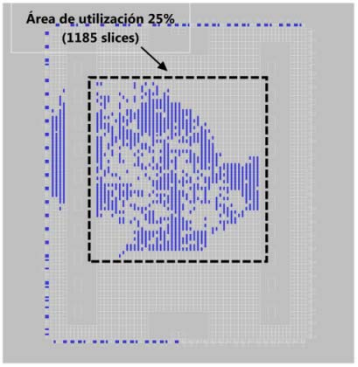

Figura 13.- Espacio en HW de Multiplicador Array de 32 bits.

Se utilizaron los multiplicadores acotados hasta 32 bits de entrada, debido a que el número de slices generados por un bloque de 64 bits es mayor a la capacidad del dispositivo, el cual contiene 4656 slices.

5. ANÁLISIS DE TIEMPO Y ESPACIO EN HARDWARE DE SUMADORES Y MULTIPLICADORES ARITMÉTICOS.

De acuerdo a la síntesis y simulación de cada bloque aritmético en VHDL, se puede hacer una comparación entre los 2 tipos de arquitecturas de bloques simulados. El entorno ISE de Xilinx, contienen herramientas con la que es posibles obtener un reporte de síntesis, en el cual se muestran entre otros factores, el retardo total del circuito y el espacio utilizado (número de slices). Para este trabajo, el reporte de síntesis de los multiplicadores y sumadores se realizó con un dispositivo FPGA SPARTAN 3E de 500,000 compuertas.

En las Tablas II y III se muestran los valores obtenidos del reporte de síntesis de las distintas arquitecturas de sumadores implementados. Los valores de las tablas muestran la tendencia del multiplicador al variar la cantidad de bits de los datos de entrada.

Tabla II.- Espacio en Hardware de Sumadores.

n_bits (2^n)	Espacio RCA	Espacio CbyPA	Espacio CSlctA
n=2	4 slices	4 slices	7 slices
n=3	9 slices	9 slices	14 slices
n=4	18 slices	18 slices	31 slices
n=5	37 slices	37 slices	64 slices
n=6	74 slices	74 slices	130 slices

Tabla III.- Retardo de Sumadores.

n_bits (2^n)	Retardo RCA	Retardo CbyPA	Retardo CSlctA
n=2	8.959 ns	8.920 ns	9.063 ns
n=3	13.203 ns	13.164 ns	12.104 ns
n=4	21.690 ns	21.651 ns	14.343 ns
n=5	38.665 ns	38.626 ns	22.510 ns
n=6	72.616 ns	72.577 ns	38.845 ns

En base a los datos de estas Tablas, obtenidos del reporte de síntesis de VHDL, es posible comparar las arquitecturas de cada sumador y el espacio ocupado en hardware. En la Figura 14, se muestra el incremento en el número de slices utilizados por cada sumador, dependiendo del tamaño en los datos de entrada.

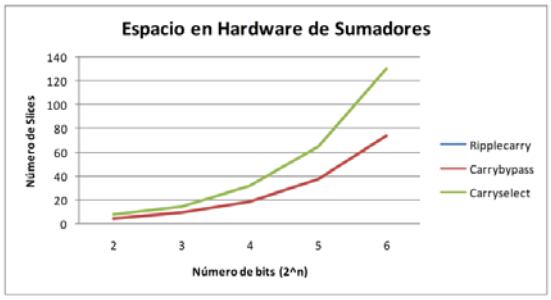

Figura 14.- Grafica Comparativa de Espacio en Hardware entre Sumadores.

De la Tabla II y la Figura 14 se observa que los sumadores Ripple Carry y Carry by Pass generan el mismo número de bloques, independientemente del tamaño, mientras que el sumador Carry Select aumenta considerablemente el número de bloques utilizados. Al analizar la Tabla III y la Figura 15, se puede observar que el sumador "Carry Select" es el que mejor desempeño presenta, produciendo un retardo mucho menor que los otros dos sumadores.

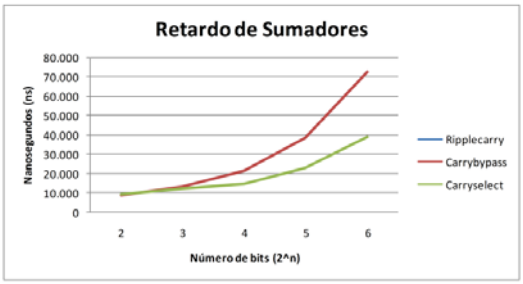

Figura 15.- Gráfica Comparativa de Retardo entre Sumadores.

El mismo análisis se realizó con los multiplicadores para obtener el espacio en hardware (Tabla IV) y el retardo máximo (Tabla V) de las arquitecturas.

Tabla IV.- Espacio en Hardware de Multiplicadores.

n_bits (2^n)	Espacio ArrayM	Espacio CSM	Espacio WalM
n=2	17 slices	17 slices	17 slices
n=3	72 slices	75 slices	75 slices
n=4	293 slices	300 slices	309 slices
n=5	1185 slices	1212 slices	1212 slices
n=6	4727 slices	4786 slices	4786 slices

Tabla V.- Retardo de Multiplicadores.

n_bits (2^n)	Retardo ArrayM	Retardo CSM	Retardo WalM
n=2	13.805 ns	11.738 ns	12.584 ns
n=3	27.447 ns	20.950 ns	21.529 ns
n=4	54.489 ns	39.057 ns	39.563 ns
n=5	117.713 ns	75.743 ns	75.999 ns
n=6	235.495 ns	148.601 ns	148.852 ns

A partir del análisis de estas Tablas y de la información presentadas en las Figuras 16 y 17, se puede determinar que la mejor arquitectura de multiplicación es la del multiplicador “Carry Save”.

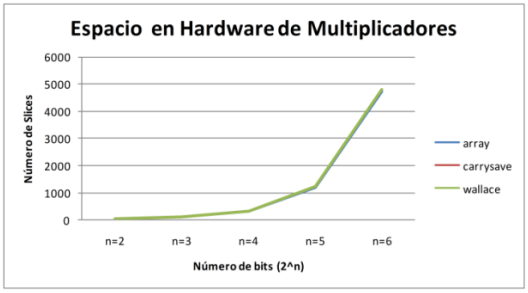

Figura 16.- Gráfica Comparativa de Espacio en Hardware entre Multiplicadores.

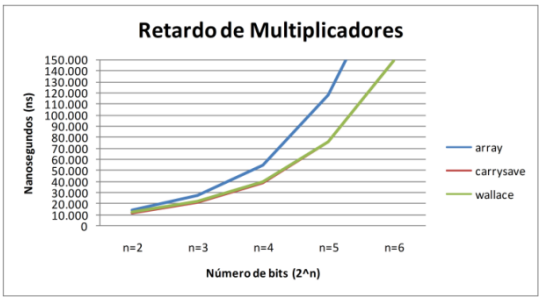

Figura 17.- Gráfica Comparativa de Retardo entre Multiplicadores.

6. CONCLUSIÓNES

En conclusión, la realización de bloques genéricos permite desarrollar el diseño de circuitos de una forma más fácil, ya que se tiene control sobre todo el sistema a implementar y evita utilizar multiplicadores embebidos, los cuales podrían resultar limitados en procesos donde se involucran muchas operaciones aritméticas y por otro lado, el número fijo de bits que procesan los bloques embebidos puede no ser el adecuado para sistemas específicos.

La síntesis y simulación de diferentes arquitecturas de multiplicadores, arrojó resultados que ayudan en la implementación de sistemas que utilizan este tipo de bloques aritméticos. La tendencia de velocidad y espacio de cada multiplicador respecto al aumento en el número de bits, indica cuales son las arquitecturas de sumadores y multiplicadores con mejor rendimiento siendo el ``Sumador Carry by Pass'' y el ``Multiplicador Carry Save'' los mejores en su correspondiente área, por ser los que utiliza menos espacio y tiempo al realizar la operación. Estos dos factores sumados a su facilidad de diseño comparada con los otros bloques aritméticos facilita el desarrollo de procesos más complicados.

El tamaño de los bloques aritméticos que se pueden implementar está limitado por la capacidad del chip donde se pretenda alojar el diseño. Debido a esto, el tamaño de los bloques desarrollados en este trabajo están acotados a 64 bits empleándose una FPGA con una capacidad de 500K compuertas, sin embargo, la cantidad de bits a procesar puede ser mayor si el PLD así lo permite.

Finalmente, como trabajo a futuro se pretende utilizar los bloques propuestos en este trabajo para el diseño y construcción de la estructura funcional de una red neuronal artificial con el propósito de clasificar objetos. La importancia de este trabajo radica en que los PLD actuales solo cuentan con un número limitado de multiplicadores embebidos y al realizar diseños complejos que requieren de un número mayor de estos elementos, los bloques diseñados son una buena solución para complementar la cantidad de elementos aritméticos requeridos.

7. REFERENCIAS

[1] O. Gustafsson, A. Dempster, and L. Wanhammar, “Multiplier blocks using carry-save adders,” in Circuits and Systems, 2004. ISCAS ’04. Proceedings of the 2004 International Symposium on, vol. 2, may 2004, pp. II – 473–6 Vol.2.

[2] R. S. Waters and E. E. Swartzlander, “A reduced complexity Wallace multiplier reduction,” Computers, IEEE Transactions on, vol. 59, no. 8, pp. 1134 –1137, aug. 2010.

[3] S. Pezaris, “A 40-ns 17-bit by 17-bit array multiplier,” Computers, IEEE Transactions on, vol. C-20, no. 4, pp. 442 – 447, april 1971.

[4] K. Swee and L. H. Hiung, “Performance comparison review of 32-bit multiplier designs,” in Intelligent and Advanced Systems (ICIAS), 2012 4th International Conference on, vol. 2, june 2012, pp. 836 –841.

[5] N. Itoh, Y. Naemura, H. Makino, and Y. Nakase, “A compact 54 times;54-bit multiplier with improved wallace-tree structure,” in VLSI Circuits, 1999. Digest of Technical Papers. 1999 Symposium on, 1999, pp. 15 –16.

[6] S. Shah, A. Al-Khalili, and D. Al-Khalili, “Comparison of 32-bit multipliers for various performance measures,” in Microelectronics, 2000. ICM 2000. Proceedings of the 12th International Conference on, 2000, pp. 75–80.

[7] M. M. Mano, Diseõ Digital. Pearson, 2003.

[8] Al-Ashrafy, M.; Salem, A. & Anis, W. (2011), An efficient implementation of floating point multiplier, *in* 'Electronics, Communications and Photonics Conference (SIECPC), 2011 Saudi International', pp. 1-5.

[9] Al-Khaleel, O.; Tulic, N. & Mhaidat, K. (2012), FPGA implementation of binary coded decimal digit adders and multipliers, *in* 'Mechatronics and its Applications (ISMA), 2012 8th International Symposium on', pp. 1-5.

[10] Aly, M. & Sayed, A. (2012), A study of signed multipliers on FPGAs, *in* 'Electronics Design, Systems and Applications (ICEDSA), 2012 IEEE International Conference on', pp. 33-38.

[11] Malik, A. & Ko, S.-B. (2005), Effective implementation of floating-point adder using pipelined LOP in FPGAs, *in* 'Electrical and Computer Engineering, 2005. Canadian Conference on', pp. 706-709.

[12] Rani, R.; Singh, L. & Sharma, N. (2009), FPGA implementation of fast adders using Quaternary Signed Digit number system, *in* 'Emerging Trends in Electronic and Photonic Devices Systems, 2009. ELECTRO '09. International Conference on', pp. 132-135.