Análisis de velocidad y complejidad de diferentes arquitecturas

Kevin Giribuela

Índice

1. Introducción

En el presente informe se analizan las arquitecturas más comunes que se pueden encontrar en los sumadores, multiplicadores y filtros. En un primer lugar se realiza una breve descripción teórica de cada arquitectura, resaltando cuales son sus ventajas y desventajas respecto a otro tipo de estructura para finalmente realizar una comparación de rendimiento, complejidad, y velocidad.

2. Sumadores

Los sumadores son utilizados en la suma, la resta, la multiplicación, y la división. La velocidad de cualquier sistema de comunicaciones o de procesamiento de señal digital depende principalmente de estos bloques. El sumador *Ripple Carry Adder* es el más lento de la familia de sumadores. A pesar de que la propagación de acarreo (*carry*) ralentiza al sumador, su simplicidad otorga un mínimo número de compuertas.

Para poder solucionar el problema de la propagación del carry es que se diseñan los sumadores rápidos. Un sumador rápido muy popular es el *Carry Look-ahead Adder*, donde el carry de entrada a cada sumador se genera simultáneamente por una lógica de predicción.

Otro dispositivo rápido es el *Carry Select Adder* (CSA). Este sumador particiona la suma en K grupos. Para ganar velocidad, se replica la lógica y en cada grupo se raliza la suma asumiendo ambos escenarios de carry posibles (0 o 1). El valor correcto de la suma y el carry de salida de cada grupo es seleccionado por la salida del carry del grupo anterior.

A continuación se realiza una breve descripción de los bloques que son utilizados en el presente informe para su posterior análisis.

2.1. Breve descripción teórica

2.1.1. Ripple Carry Adder (RCA)

Al RCA se lo percibe como el sumador más lento. Esta percepción resulta errónea si a esta arquitectura se la mapea en una FPGA con una lógica de cadena de acarreo embebida. En la Figura 1 se muestra un RCA de 6-bits.

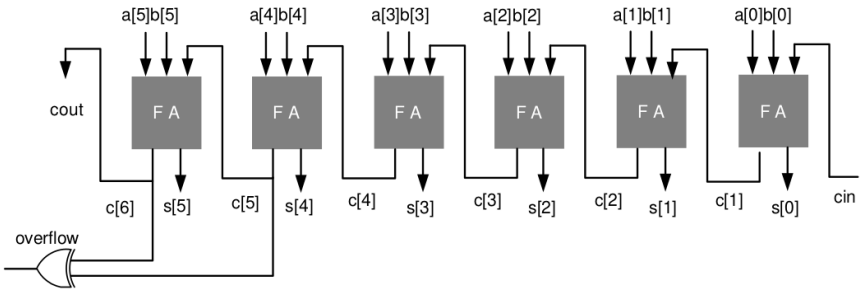

Figura 1: Ripple Carry Adder de 6-bits

Un RCA consume el área mínima y presenta una estructura regular. Un sumador del tipo RCA que sume dos números de N-bits requiere N *Full Adders* (FA). A partir de la figura anterior es posible obtener una expresión del camino crítico:

$$T_{RCA} = (N-1)T_{FA} + T_m \tag{1}$$

donde T_{RCA} es el retardo de un RCA de N bits, mientras que T_{FA} y T_m son los retardos de un FA y la lógica de generación de carry de los FA, respectivamente. Es evidente de (1) que el desempeño de un RCA se encuentra restringido por la propagación del carry. Con el fin de solucionar este inconveniente, la mayoría de las FPGAs son construidas con lógica de propagación de acarreo. De esta forma, se ayuda al diseñador para que implemente RCA rápidos en una FPGA.

2.1.2. Binary Carry Look-ahead Adder

Para describir este nuevo sumador primero es necesario conocer el comportamiento de un *Carry Look-ahead Adder*. *Grosso modo*, en un CLA los acarreos que ingresan a todas las posiciones de bits del sumador son generados simultáneamente. Esto es, el cálculo del carry se realiza en paralelo con el cálculo de la suma. Esto resulta en una suma constante independiente de la longitud del sumador.

A medida que la palabra a sumar crezca, la organización del hardware comienza a ser cada vez más compleja.

Es posible obtener una expresión para el i-ésimo carry considerando la lógica de un FA [1]. En (2) se observa tal expresión

$$c_i = g_{i-1} + \sum_{j=0}^{i-2} \left(\prod_{k=j+1}^{i-1} p_j \right) g_j + \prod_{k=0}^{i-1} p_j c_0 \tag{2}$$

En la Figura 2 se observa la lógica necesaria para generar la salida de c_4 en una suma.

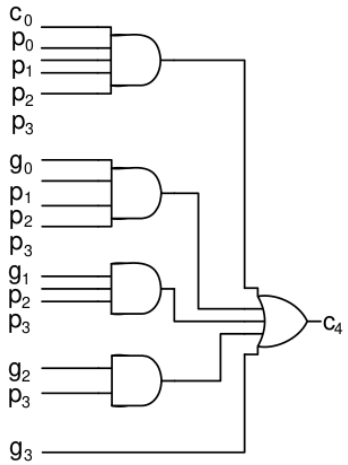

Figura 2: Lógica de un CLA para calcular el carry

Si bien de la figura anterior es claro que el retardo involucrado en el cálculo del carry es de 2 compuertas lógicas, esto en la práctica no es así ya que en las FPGAs se suele contar con compuertas de 2 entradas y no 4. Mas adelante veremos cómo impacta este hecho en la implementación física de los sumadores.

Ya con una idea del funcionamiento de un CLA, se introduce el BCLA. Este tipo de sumador trabaja en un grupo de 2 bits adyacentes, y luego combina sucesivamente dos

grupos desde el LSB al MSB para formular un nuevo grupo y su correspondiente carry. La lógica para la generacion del carry de un sumador de N bits es

$$g_i = a_i b_i \tag{3a}$$

$$p_i = a_i \oplus b_i \tag{3b}$$

$$(G_i, P_i) = (g_i, p_i) \cdot (g_{i-1}, p_{i-1}) \cdot \ldots (g_1, p_1) \cdot (g_0, p_0) \tag{3c}$$

El problema puede ser resuelto de manera recursiva como

$$(G_0, P_0) = (g_0, p_0)$$
for i = 1 hasta N-1
$$(G_i, P_i) = (g_i, p_i) \cdot (G_{i-1}, P_{i-1})$$
$$c_i = G_i + P_i c_0$$
end

Existen diversas optimizaciones reportadas en la literatura para mejorar la realización serie del algoritmo, en la Figura 3 se muestra la implementación del sumador Brent-Kung de 8 bits.

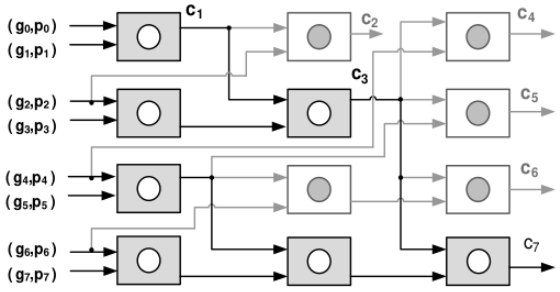

Figura 3: Sumador Brent-Kung de 8 bits

2.1.3. Carry Select Adder

El CSA no es tan rápido como lo es el CLA y requiere considerablemente más hardware si se lo mapea en un ASIC, pero tiene un diseño favorable si se lo implementa en FPGA con lógica de cadena de acarreo rápida (al igual que sucede con los RCA). El CSA particiona un sumador de N bits en K grupos, donde

$$k = 0, 1, 2, \ldots, K-1 \tag{4a}$$

$$n_0 + n_1 + \cdots + n_{K-1} = N \tag{4b}$$

$$n_0 \leq n_1 \leq \cdots \leq n_{K-1} \tag{4c}$$

donde n_K representa el número de bits en el grupo k. La idea es colocar dos sumadores de n_k bits en cada etapa k. Un conjunto de sumadores computa la suma asumiendo que el carry es 1, y otro hace lo mismo pero asumiendo que el carry es 0. La suma y el carry correcto luego son seleccionados utilizando un multiplexor de 2-1 basado en el carry del grupo anterior. En la Figura 4 se muestra un sumador CSA de 16 bits.

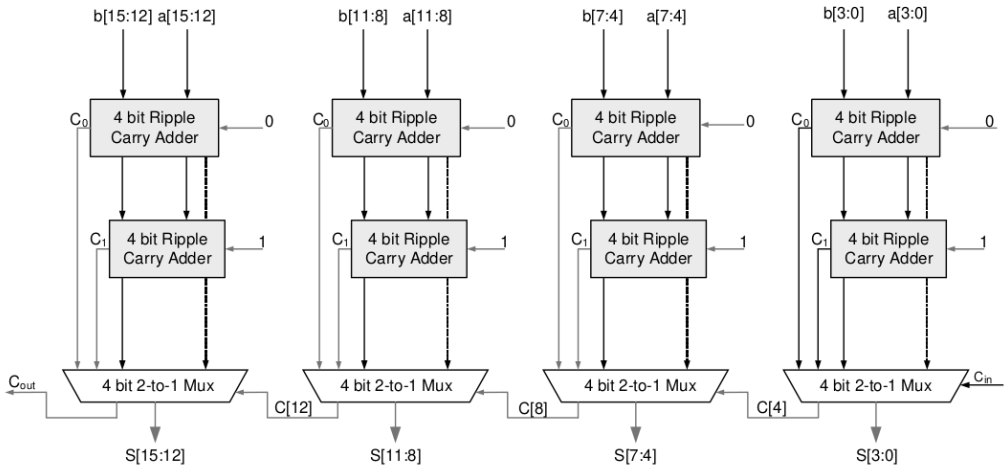

Figura 4: CSA de 16 bits

2.2. Simulaciones

A continuación se presentan los resultados obtenidos de sintetizar los 3 tipos de sumadores descriptos (RCA, BCLA, HCSA) a partir de los cuales se realiza un análisis de área, complejidad y velocidad.

2.2.1. Área

El primer punto que se observa es la cantidad de celdas relativa que utiliza cada uno de estos sumadores y el nivel de complejidad de hardware. Para ello se realizaron *pie charts* con la cantidad de elementos utilizados. En la Figura 5 se observan dichos gráficos.

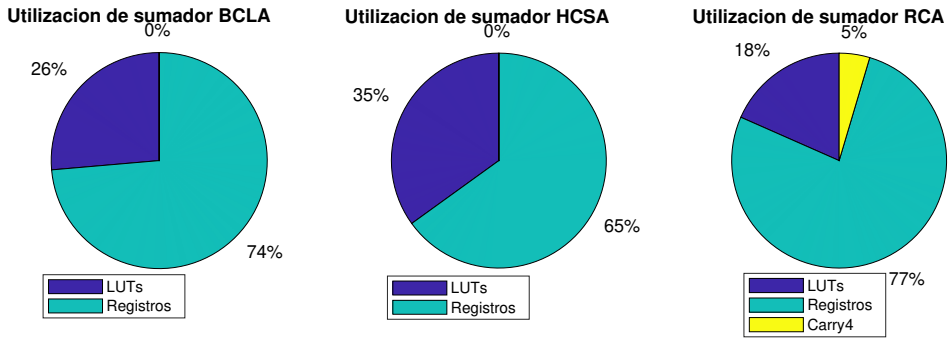

Figura 5: Consumo de celdas relativo

Lo primero que se observa de la figura anterior es que el RCA es el único sumador que utiliza carry. Esto se debe a que en la etapa de diseño simplemente se escribió `A+B`, dándole lugar a la herramienta para que realice las optimizaciones que considere necesarias, entre ellas, utilizar lógica de acarreo. Esto está en concordancia según lo discutido en 2.1.1, donde se hizo notar el hecho de que las FPGAs cuentan con cadena de acarreo dedicadas para este tipo de operaciones facilitando al diseñador el uso de los RCAs.

Por otra parte, el BCLA y HCSA, no cuentan con el carry. Esto es porque cuando se lo diseñó desde cero, se le restringió de algún modo a la herramienta operar sobre la lógica para obtener un mejor rendimiento.

2.2.2. Complejidad vs Velocidad

Si bien el gráfico anterior refleja el porcentaje relativo de celdas que componen a cada sumador, no otorga una visión absoluta de la cantidad de componentes. En la Figura 6 se muestra un gráfico de barras donde se muestran la cantidad de LUTs, regitros y carrys utilizados por cada uno de los sumadores. Al mismo tiempo, sobre el eje y derecho, se grafica la frecuencia máxima de trabajo que admite cada uno de ellos.

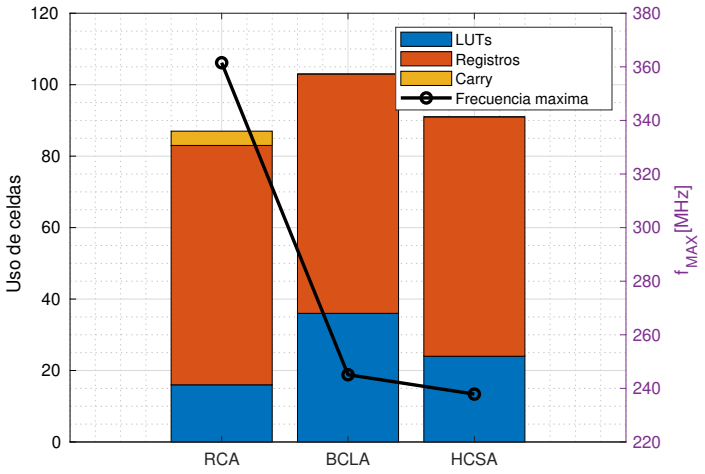

Figura 6: Complejidad vs Velocidad

Ahora es posible observar que la cantidad de LUTs que utilizan el BCLA y el HCSA es mayor que las utilizadas por el RCA. Este resultado se debe a que la lógica encargada de la comparación las hacen las LUTs, mientras que para el caso del RCA esta tarea la realizan los carry.

Otro resultado importante es la velocidad máxima de trabajo de cada sumador. Claramente el RCA permite trabajar a una frecuencia considerablemente mayor que el BCLA y HCSA. Este comportamiento se lo podemos asignar nuevamente al hecho de la utilización de lógica de acarreo en las operaciones de suma, estando en concordancia una vez más con lo discutido en 2.1.1.

3. Multiplicadores

La mayoría de los algoritmos de procesamiento digital de señales usan multiplicadores. Debido a esto, los vendedores de FPGAs están embebiendo muchos multiplicadores dedicados en ellas. Es muy importante comprender las técnicas que se practican para optimizar la implementación de los multiplicadores. A pesar de que un multiplicador secuencial puede ser diseñado para que tome múltiples ciclos de reloj para computar un producto, la arquitectura de los multiplicadores que computan el producto en un ciclo de reloj son de gran interés para los diseñadores en sistemas de alta tasa.

Un bloque fundamental en la construcción de las arquitecturas de multiplicadores en paralelo es el CSA (*Carry Save Adder*, por sus siglas en inglés). La utilización de los CSAs es una de las técnicas más eficientes y ampliamente utilizadas para acelerar sistemas de procesamiento digital de señales. En la Figura 7 se muestra su estructura básica. Debido a que su entrada admite 3 operandos y a su salida sólo hay 2, también es conocido con el nombre de compresor de 3:2.

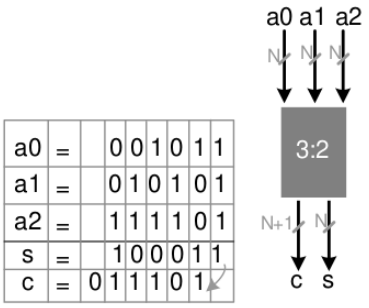

Figura 7: CSA

3.0.1. Multiplicador de Booth recodificado

Una forma de optimizar la operación de multiplicación es reduciendo el numero de productos parciales (PP). El algoritmo de Booth se encarga de esta tarea.

Idea: Al multiplicar dos numeros signados de N bits a y b, la técnica genera un producto parcial cada dos bits de a pares tomados. La técnica, mientras se desplaza desde el LSB al MSB, toma bits de pares, cuyos valores posibles son 00_2, 01_2, 10_2, y 11_2. Al multiplicar por 00_2, 01_2, y 10_2 el resultado puede ser 0, a y $2a = (a \ll 1)$ respectivamente, donde cada PP es apropiadamente calculado. La cuarta posibilidad de $11_2 = 3$ es $2 + 1$, y un simple desplazamiento podría no generar el PP requerido; mas bien esta multiplicación resultará en dos PPs y ellos son a y $2a$. Esto implica que en el peor caso, el multiplicador termina obteniendo N PPs.

El problema de 11_2 se resuelve utilizando el algoritmo de Booth recodificado. El algoritmo sigue funcionando con grupos de 2 bits cada uno pero recodificando cada grupo para usar uno de los cinco valores equivalentes: 0, 1, 2, -1, -2. Estos valores son almacenados en una tabla de look up. En la Figura 8 se muestra un multiplicador de 8 bits utilizando este algoritmo.

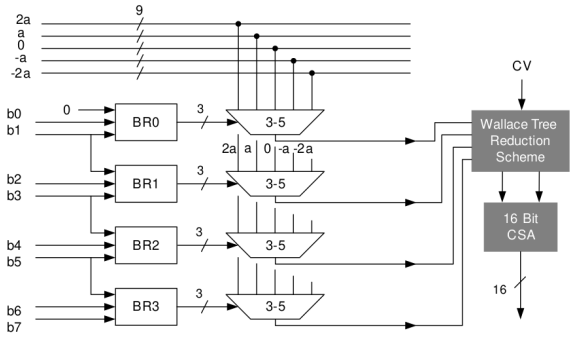

Figura 8: Multiplicador de 8 bits utilizando algoritmo de Booth recodificado

3.1. Simulaciones

Al igual que se hizo con los sumadores, a continuación se realiza un análisis del comportamiento de estos dos multiplicadores desde el punto de vista de área consumida, complejidad, y velocidad.

3.1.1. Área

En la Figura 9 se muestra la cantidad de celdas relativas utilizadas por cada arquitectura.

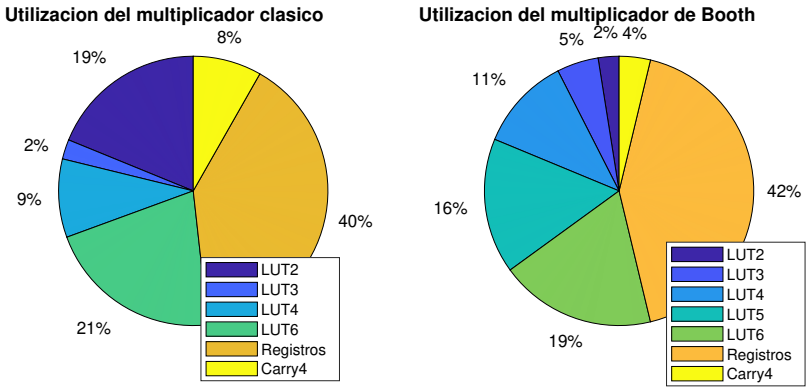

Figura 9: Consumo de celdas relativo

3.1.2. Complejidad vs velocidad

En la Figura 10 se muestra la cantidad absoluta de celdas utilizadas y su velocidad correspondiente para cada multiplicador.

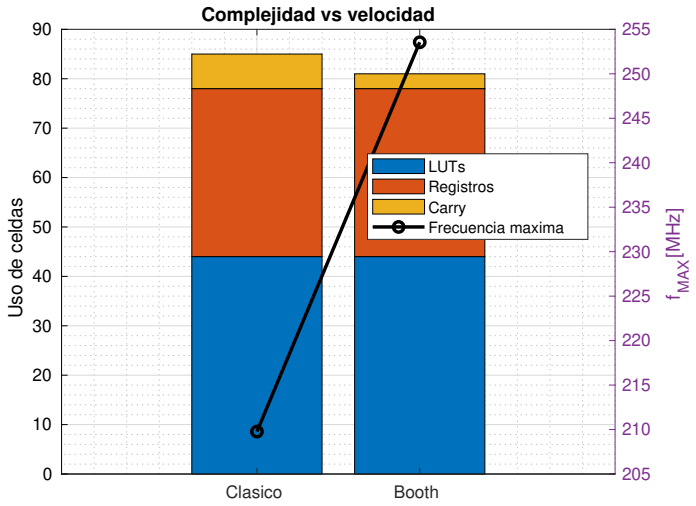

Figura 10: Complejidad vs velocidad de los multiplicadores

4. Filtros FIR

En esta sección se realiza el estudio de las diferentes formas de implementar filtros FIR utilizando las técnicas de suma y multiplicación de las secciones anteriores. Además, se hace uso de técnicas como la de vector de corrección y representación de dígito de signo canónico para mejorar el rendimiento de los multiplicadores. Al mismo tiempo, se analiza

el desempeño de los filtros FIR implementados utilizando aritmética distribuida y forma directa traspuesta.

4.1. FIR - Forma directa I

La forma directa I es la manera trivial de implementar un filtro FIR. Su implementación consiste en replicar la ecuación en diferencias que lo describe por medio de registro de almacenamientos, multiplicadores y sumadores.

$$y[n] = \sum_{k=0}^{L-1} h[k]x[n-k] \tag{5}$$

En (5) se encuentra la ecuación que lo describe y en la Figura 11 la implementación del mismo considerando $L = 3$.

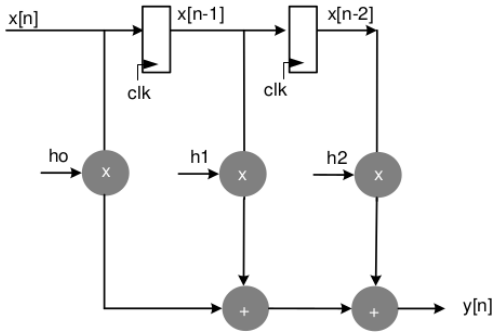

Figura 11: Filtro FIR

En este tipo de implementación, si bien la complejidad no es elevada, la velocidad a la que puede funcionar el filtro queda restringida por el camino crítico conformado por el valor de $x[0]$, la multiplicación por $h[0]$ y el árbol de sumas. De esta forma, el camino crítico aumenta a medida que la cantidad de coeficientes aumente. Una técnica que permite cortar el camino crítico es introducir registros.

4.2. FIR - Forma directa traspuesta

El problema del camino crítico para el filtro en forma directa I puede resolverse a través de un cambio en la estructura del filtro. En la Figura 12 se muestra un filtro FIR implementado en forma traspuesta.

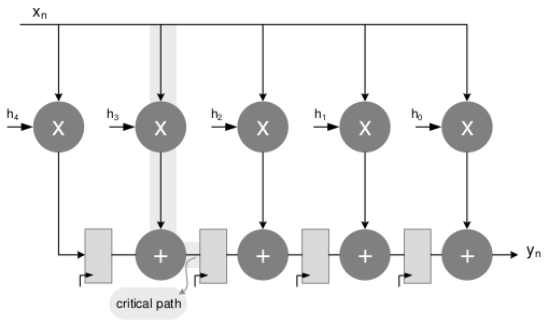

Figura 12: Filtro FIR en forma traspuesta

En la figura anterior se ve que el camino crítico ahora se redujo a un sumador y un multiplicador solamente, sin embargo, el tamaño de los registros aumenta respecto a la forma directa I.

4.3. FIR - Aritmética distribuida

La aritmética distribuida es una de las formas de implementar productos punto donde uno de los vectores está conformado por elementos constantes. El hecho de conocer el valor de los elementos del vector puede explotarse por medio del almacenamiento de dichos valores en memoria y a través de una secuencia específica de valores de entradas realizar la multiplicación correspondiente. En la Figura 13 se muestra cómo sería la implementación de un filtro FIR utilizando aritmética distribuida.

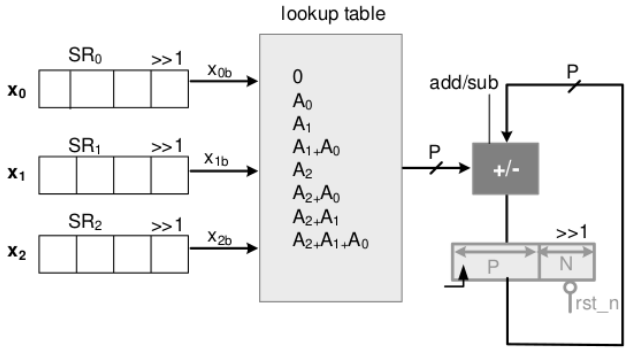

Figura 13: Filtro FIR implementado por medio de aritmética distribuida

4.4. FIR - CV

La implementación de filtro FIR utilizando el concepto de vector de corrección (CV por sus siglas en inglés) permite al diseñador reducir el área que ocupe el módulo. Esto se realiza calculando previamente el CV si se conoce el tamaño de la palabra de entrada y salida. De esta forma, es posible realizar los productos parciales sin necesidad de realizar una extensión de signo y sumar simplemente al final de toda la operación una constante.

Esta técnica permite olvidarse de la complejidad extra que introduce el proceso de extensión de singo.

4.5. FIR - CSD

La técnica de representación de dígito de signo canónica es muy útil al momento de reducir la cantidad de productos parciales en la operación de filtrado. Por ejemplo, convertir el número $16'b0011\_1110\_1111\_0111$ en su representación CSD se traduce en

$$0011111011110111$$
$$001111101111100\bar{1}$$
$$00111111000\bar{1}00\bar{1}$$
$$0100000\bar{1}0000\bar{1}00\bar{1}$$

De esta forma, se redujo el numero productos parciales de 12 a 4. Es por eso que, cuando los coeficientes del filtro son constantes, utilizar esta técnica gana en velocidad.

4.6. Simulaciones

Una vez más, al igual que en los casos anteriores, se sintetizan las diferentes estructuras de los filtros FIR y se realiza un análisis de área, complejidad, y velocidad.

4.6.1. Área

Comenzando con el área relativa utilizada, y la cantidad de celdas que usa cada estructura es que se obtienen los resultados de la Figura 14.

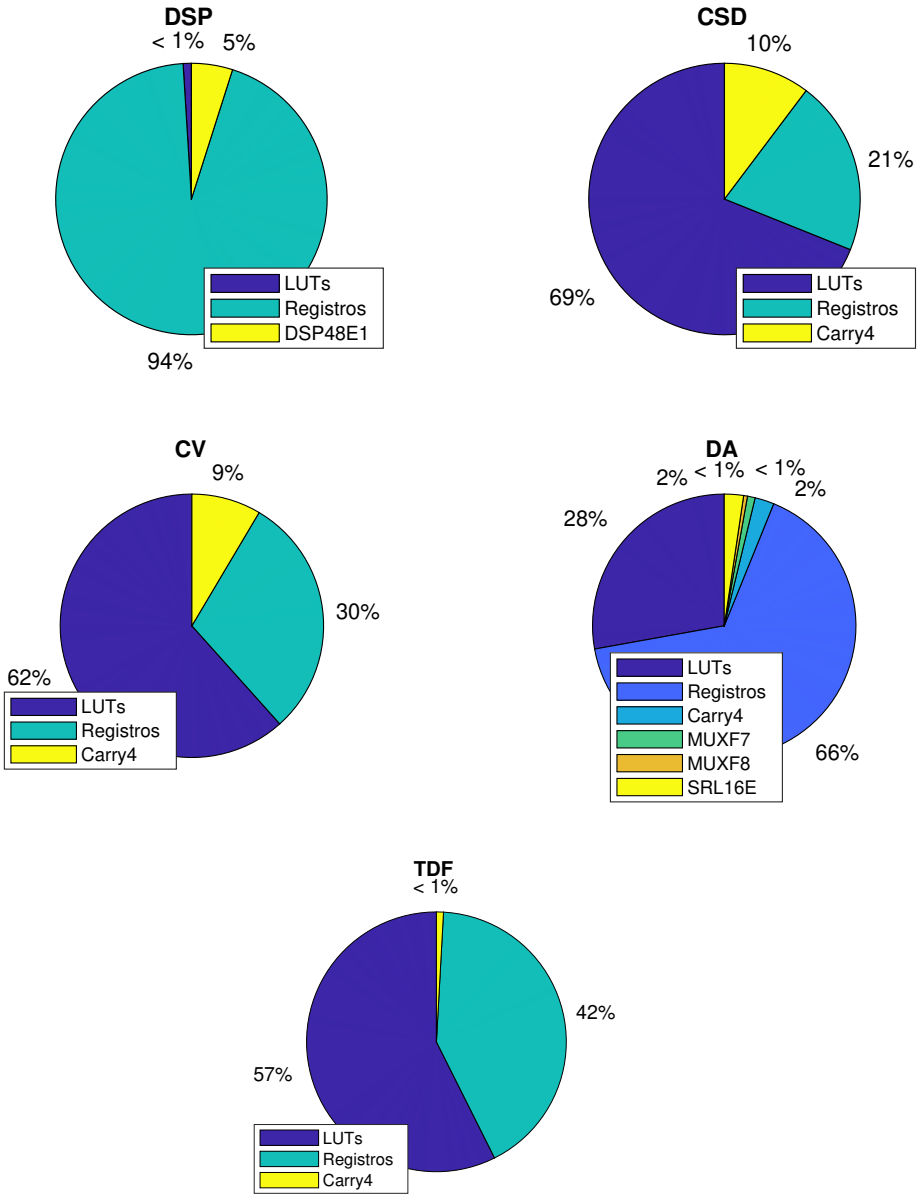

Figura 14: Área relativa consumida por un filtro FIR en sus direfentes implementaciones

En dicha imagen se aprecia que todos los filtros utilizan LUTs para realizar las operaciones aritméticas. Ahora, algo particular sucede para el caso de la implementación en la forma directa I (DSP), en ella se observa que es la única estructura que utiliza un DSP. Esto se debe principalmente a que se le ha dado cierto grado de libertad a la herramienta para que sintetice las operaciones correspondiente, de esta forma, dado que se cuenta con

un bloque dedicado a realizar dichas operaciones, la herramienta encuentra que lo óptimo sería utilizar un DSP para implementar el filtro.

Por otro lado, se observa que la estructura con aritméica distribuida (DA) utiliza registros de desplazamiento y multiplexores. Según lo discutido en 4.3, esto se corresponde con el hecho de que es necesario desplazar las muestras de entradas y utilizar los multiplexores para decidir qué elemento de la tabla debe seleccionarse.

4.6.2. Complejidad vs velocidad

Para finalizar, se analiza la complejidad de cada filtro y la velocidad admisible. En este análisis se hace incluido el desempeño del filtro de aritmética distribuida a fin de completitud simplemente, su resultado no se debe considerar debido a que para realizar una comparación justa con los demás filtros, su salida debe paralelizarse a fin de comparar con la misma tasa de datos de salida con los demás filtros.

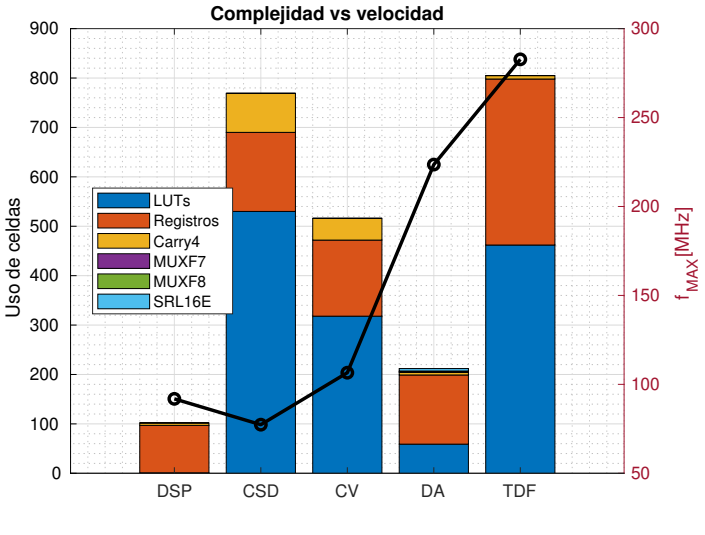

Figura 15: Análisis de complejidad vs velocidad

En la Figura 15 se observa cómo predomina la forma directa I frente a las demás implementaciones en cuanto al consumo de área. Esto es debido principalmente a la sencillez de sus estructura. Sin embargo, es evidente que si lo que se busca es velocidad, quizás no sea la mejor opción.

El caso contrario es la implementación del filtro utilizando la forma traspuesta. Tal como se discute en 4.2, el problema del camino crítico que presentaba la forma directa I se soluciona invirtiendo la posición de los registros de almacenamiento y el orden de las multiplicaciones, sin embargo, debido a que los registros ahora deben ser más grandes para almacenar el valores de entrada, el área efectiva es mayor.

Un caso no tan óptimo se observa para la implementación de los filtros utilizando las ideas de CSD y CV. En ambos casos es posible observar un incremento de área considerablemente mayor respecto al filtro en forma directa I. Si bien hay una ganancia en frecuencia para el caso de CV, no es comparable con el área utilizada. Incluso, el diseño con CSD es más lento y más complejo que para el caso de la forma directa I.

5. Conclusiones

A modo de cierre, se resumen las características de los distintos módulos y sus implementaciones por medio de filtros FIR.

- **Sumadores:** Se ha visto que la manera óptima de realizar una suma en una FPGA es mediante el uso de sumadores RCA. Se ha comprobado que su desempeño es superior frente a las implementaciones de BCLA y HCSA debido a que para el caso de FPGAs se cuenta con cadenas de acarreo embebidas, de manera que la herramienta puede hacer un uso eficiente de los recursos disponibles para aumentar la velocidad, más aún, la complejidad que demanda este tipo de sumadores no es compleja comparada con sus competidores.

- **Multiplicadores:** En cuanto a los multiplicadores, se ha comprobado que la teoría y la práctica se cumplen. Es decir, se pudo comprobar que el algoritmo de Booth recodificado presenta un desempeño superior al esquema clásico utilizando prácticamente la misma cantidad de celdas. Sin embargo, es importante mantener en vista que al momento de diseñar, no es necesario realizar multiplicaciones forzando la implementación de dicho algoritmo, mas bien la herramienta es la encargada de determinar cuándo y cómo realizar estas optimizaciones.

- **Filtros FIR:** Por último, se observó que si lo que se quiere priorizar es el área consumida por el diseño, a la hora de realizar un filtro FIR es aconsejable seguir el esquema tradicional de implementación. Esto tiene como contra parte el hecho de que la velocidad de trabajo se verá limitada por el gran camino crítico que presenta la forma directa I.

  Ahora, si lo que se busca es un diseño veloz, donde el área no sea un factor crítico, la implementación de un filtro FIR mediante la estructura de la forma traspuesta es la mejor opción. De los resultados obtenidos por simulación se observó que la velocidad máxima obtenida fue casi 3 veces la del filtro implementado en forma directa I. Sin embargo, el área incrementó de manera importante siendo 8 veces la necesaria para implementar un filtro FIR en forma directa I.

  Por último, los filtros implementados mediante las estructuras CSD y CV no presentaron una ventaja considerable respecto a las estructuras mencionadas previamente, incluso la implementación del filtro con CSD consume casi 8 veces el área de la forma directa I y la frecuencia de trabajo es menor.

Referencias

[1] S. A. Khan, *Digital design o signal processing systems*, 1st ed. John Wiley, 2011, ch. 5.