Estimación de Movimiento en MPEG mediante Técnicas de Aritmética On-Line sobre FPGA

Joaquín OLIVARES[1], Francisco J. HORMIGO[2], José I. BENAVIDES[1], Julio VILLALBA[2] y Emilio L. ZAPATA[1]

[1]Dpto. Electrotecnia y Electrónica
Universidad de Córdoba, España
y
[2]Dpto. Arquitectura de Computadores
Universidad de Málaga, España

RESUMEN

Hoy en día, una de las tareas críticas en compresión de video para los estándares H.26x, MPEG-1, -2 y -4 es la estimación de movimiento (ME). La mayoría de los algoritmos para el cálculo de ME se basan en el cálculo del mínimo de las sumatorias de las diferencias absolutas (MAD) [1] entre un bloque de referencia y los posibles bloques candidatos. En este artículo proponemos un diseño en FPGA que permite obtener un cálculo rápido del MAD. Gracias a la aplicación de técnicas de aritmética on-line (OLA) obtenemos dos ventajas: es posible calcular el MAD de macrobloques completos de 16x16 píxeles en una sencilla FPGA; y en segundo lugar, permite acelerar el cálculo computacional gracias a que el cálculo de la sumatoria de las diferencias absolutas (SAD) puede ser truncado cuando el bloque candidato presenta un mayor SAD que el mejor calculado hasta ese momento. El diseño se ha sintetizado utilizando Xilinx ISE 5.2i en dispositivos pertenecientes a las familias VIRTEX-II y SPARTAN-II. Para implementar un MAD para un macrobloque de 16x16 píxeles se requieren 1945 look-up tables (LUTs) obteniendo una frecuencia de trabajo de 425 MHz. Además se presenta una comparación con otros trabajos relacionados.

Palabras clave: Estimación de Movimiento, FPGA, Sumatoria de las Diferencias Absolutas, Aritmética On–Line, MPEG.

1. INTRODUCCIÓN

El cálculo de la ME ocupa un importante lugar en los sistemas de codificación y procesamiento de video, esto se debe a que los vectores de movimiento aportan una información crítica para la reducción de redundancias temporales. La ME se utiliza ampliamente en los estándares de compresión de video H.26x, MPEG-1, -2 y -4; y se define como la búsqueda del mejor vector de movimiento, mientras que el vector de movimiento representa el desplazamiento de un bloque o macrobloque en la imagen actual sobre la imagen anterior.
Existen varios algoritmos para el cálculo de la ME: *hierarchical block-matching, n-stepsearch, full-search block-matching,*etc.; el diseño que proponemos para el cálculo del MAD es aplicable a cualquiera de ellos.

El alto coste computacional de algoritmos de *block-matching* (BMA) puede ser un problema significativo en aplicaciones para codificación de video en tiempo real. Para reducir la complejidad computacional se han propuesto varios algoritmos que realizan la búsqueda en un subconjunto del bloque candidato [7]. A su vez, se han desarrollado diferentes arquitecturas para acelerar el cálculo aritmético de conjuntos con un elevado número de datos que presentan cierta asociatividad [1].

La solución a este problema puede ser combinar un procesador programable con una *field-programmable gate-arrays* (FPGA); ésta se ocupará del procesamiento de tareas críticas caracterizadas por ser paralelizables. El diseño propuesto pretende acelerar el cálculo del MAD mediante su implementación en FPGA (*MAD processor*) mientras un procesador suministra los bloques referencia y candidato a la FPGA (*dispatcher*), ver fig. 1. En este artículo desarrollamos el diseño de la arquitectura en FPGA para calcular el MAD. El diseño propuesto se puede integrar con cualquier BMA.

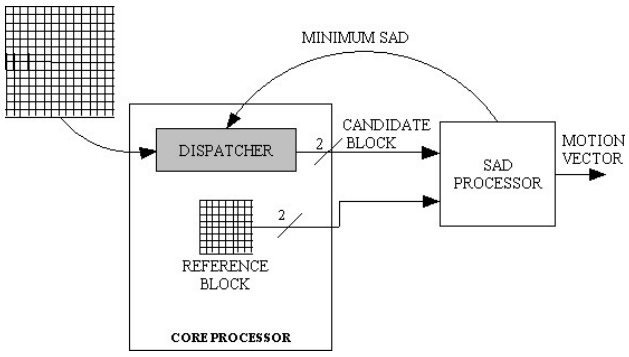

Fig.1 Sistema para la estimación del movimiento.

La suma de las diferencias absolutas (SAD) para un macrobloque completo en MPEG de 16x16 píxeles de 8 bits supone 512 operandos de 8 bits. El cálculo en paralelo del SAD para un macrobloque implica un elevado número de operandos, esto ha hecho inviable hasta ahora la implementación del SAD completamente paralelizado sobre FPGAs. En [1] se propone el cálculo de sólo una fila de un macrobloque (16x1) y su implementación en FPGA. Así mismo proponen replicar el diseño (lo cual es inviable en una FPGA sencilla) o bien utilizar un pipeline para conseguir así el cálculo para el macrobloque (16x16) completo. Se propone la aplicación de técnicas OLA para reducir drásticamente los requerimientos hardware e implementar el cálculo completo del SAD sobre una sencilla FPGA.

Por otra parte, OLA se caracteriza por operar calculando el dígito más significativo (MSD) en primer lugar. Esto permite

truncar los cálculos; ya que cuando el SAD del bloque candidato que se está calculando es mayor que el SAD del bloque de referencia, no es necesario calcular el SAD del bloque candidato completamente.

La organización de este artículo es la siguiente: en la sección 2 se realiza una breve descripción de las técnicas OLA; en la sección 3 se detalla cómo calcular el MAD utilizando OLA; en la sección 4 se muestra la implementación en FPGA del diseño propuesto; en la sección 5 se incluye una comparación con otros trabajos relacionados; y por último, en la sección 6 se presentan los resultados más importantes del trabajo.

2. ARITMÉTICA ON-LINE

Las técnicas OLA se han utilizado para resolver problemas de procesamiento de señal como filtrado digital, la transformada de Fourier, cálculo sobre matrices y otros [2 – 5]. Recientemente se han presentado trabajos que muestran los atractivos de la utilización de técnicas OLA en diseños sobre FPGA [8].

El principio sobre el que trabajan las técnicas OLA es operar y obtener resultados dígito a dígito [3] de forma que comienzan a operar por el dígito más significativo. Para generar el primer dígito del resultado son necesarios δ + 1 dígitos de los operandos. Por tanto, después de δ dígitos introducidos de los operandos se obtiene el primer dígito del resultado, posteriormente se obtendrá un nuevo dígito del resultado en cada ciclo de reloj. Por esta razón, δ es conocido como el retardo on-line.

Para generar MSD del resultado sin haber computado todos los dígitos de los operandos, se utiliza un sistema de codificación redundante. Esto implica que pueden producirse varias representaciones para un dígito, lejos de ser un inconveniente esta característica nos va a ser útil pues nos permite introducir una compensación en el cálculo del dígito si fuese necesario.

Se pueden utilizar diferentes sistemas de representación. En este trabajo se ha utilizado la representación *signed-digit* (SD). La representación SD radix-2 ofrece el siguiente conjunto de dígitos {-1, 0, 1}. Para representar cada dígito son necesarios dos bits, como se muestra en la tabla 1. El primer bit tiene un peso negativo mientras el segundo bit tiene peso positivo. Este sistema numérico elimina el acarreo de propagación tradicional en las operaciones de adición quedando reducido a los dos dígitos previos de cada operando.

Valor del dígito	Representación
+ 1	01
0	00
0	11
- 1	10

Tabla 1. Codificación de los dígitos para una representación RADIX-2 SD

En resumen, las ventajas de utilizar OLA son: se reduce el número de dígitos necesarios para el cálculo y por tanto el número de líneas de entrada al sistema, la característica MSD permite realizar cálculos posteriores sobre los resultados que se están obteniendo sin ser necesario el cálculo completo de los mismos, y se elimina la propagación del acarreo gracias a la representación SD.

3. CÁLCULO DEL MAD MEDIANTE TÉCNICAS OLA

El propósito de nuestro diseño sobre FPGA es encontrar cual de los bloques candidatos (suministrados por el módulo gestor de datos, *dispatcher* en la fig.1) ofrece mayor similitud con el bloque de referencia. La métrica más común para determinar el bloque que ofrece una mayor similitud es la sumatoria de las diferencias absolutas (SAD), y posteriormente determinar cual de estas sumatorias es la menor, esto es, el mínimo de las sumatorias de las diferencias absolutas (MAD).

En cada iteración el SAD correspondiente al bloque candidato se calcula utilizando simultáneamente todos sus píxeles. El valor generado, denominado SAD actual (SADc), es comparado con el SAD de referencia (SADr), SADr representa al mínimo SAD calculado hasta el momento. Si SADc es menor que SADr, es almacenado como nuevo SADr para posteriores iteraciones.

Las operaciones para el cálculo del SAD y la comparación se han desarrollado utilizando técnicas OLA, esto nos permite comenzar la comparación cuando el primer dígito del SAD es generado y permite por tanto parar el cálculo del SAD si los primeros dígitos calculados nos indican que el SADc que estamos calculando va a ser mayor que el SADr. Esta característica del diseño propuesto, va a implicar un ahorro significativo del número de cálculos requeridos.

Cálculo Del SAD Mediante Técnicas OLA
El SAD es la sumatoria de las diferencias absolutas píxel a píxel entre los bloques candidato y referencia.

$$\mathrm{SAD} = \sum_{i=1,\, j=1}^{N,N} \left| c_{i,j} - r_{i,j} \right| \qquad (1)$$

Donde $r_{i,j}$ representa a los píxeles del bloque de referencia y $c_{i,j}$ representa a los píxeles del bloque candidato.

El cálculo del SAD se puede dividir en tres pasos:

- Cálculo de las diferencias: $d_{i,j} = c_{i,j} - r_{i,j}$
- Determinación del valor absoluto de cada diferencia: $| d_{i,j} |$
- Adición de todas las diferencias.

Conversión a representación SD y cálculo de las diferencias: En representación SD radix-2, cada dígito se compone de dos bits, el primero tiene peso negativo y el segundo peso positivo. Así, un número SD puede ser interpretado como la diferencia de dos números sin signo.

Esta propiedad se utiliza para realizar simultáneamente la conversión del valor de cada píxel a representación SD y calcular la diferencia entre los píxeles de los bloques candidato y referencia sin coste computacional. Basta tomar cada uno de los bits candidato y de referencia y se forma con ellos el dígito SD correspondiente a la diferencia de los mismos. En la figura 2

se muestra un ejemplo para la diferencia de dos números binarios de 4 bits, y la obtención de la diferencia de los mismos ya en representación SD.

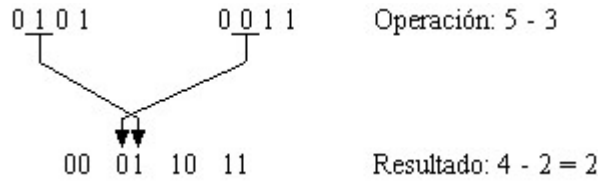

Figura 2. Ejemplo de diferencia y conversión a SD radix-2 sin coste computacional.

Obtención del valor absoluto: Para esta operación en MSD es necesario comprobar que el primer dígito distinto de cero es positivo (01) o negativo (10); en caso de ser negativo es necesario cambiar el signo del valor. En SD radix-2, ese cambio de signo se puede realizar sencillamente intercambiando los bits de cada dígito. Así, sí el primer dígito distinto de cero es negativo se intercambiarán los bits de los dígitos restantes.

Sumatoria de las diferencias absolutas: El cálculo de la diferencia absoluta entre cada píxel se realiza en paralelo para todos los píxeles de los bloques referencia y candidato. Conforme son calculados los dígitos del proceso anterior se procede a la sumatoria de todos ellos en un árbol de sumadores para OLA. Cada sumador (como se observa en la figura 3) es un sumador on-line estándar para representación SD.

El número de pasos en la adición para un árbol completo de sumadores es $log_2(N^2)$, donde N^2 representa el número de píxeles de un bloque o superbloque MPEG. Este sumador completo presenta un retardo para la adición de 2. Esto es, el MSD del resultado es obtenido dos ciclos de reloj después de que el MSD de cada entrada llegue al sumador. En nuestro caso, el bit de acarreo es utilizado como MSD del resultado, y este dígito se obtiene un ciclo antes.

Comparación on-line para dígitos SD: Tras obtener el primer dígito del SAD correspondiente al bloque actual, se comienza a comparar el SADc con el SADr. Gracias a la característica MSD se puede aplicar una comparación eficiente del algoritmo. Dado que en SD existen varias representaciones para un mismo valor, la comparación entre valores SD es más compleja que en otras convencionales.

En [6] se propone un algoritmo de comparación OLA así como su implementación hardware. En dicho trabajo se propone convertir los números SD a signo magnitud y aplicar posteriormente un algoritmo de comparación estándar que presenta un retardo on-line de dos.

Proponemos una comparación sin retardo on-line. Para ello nos basamos en el análisis del signo en la operación de diferencia en SD.

Dados dos números A y B definidos por *n* dígitos SD:

$$a_i, b_i \in \{\bar{1},0,1\} \tag{2}$$

Operando A-B:

$$A - B = \sum_{i=0}^{n-1} (a_i - b_i) \cdot 2^i \tag{3}$$

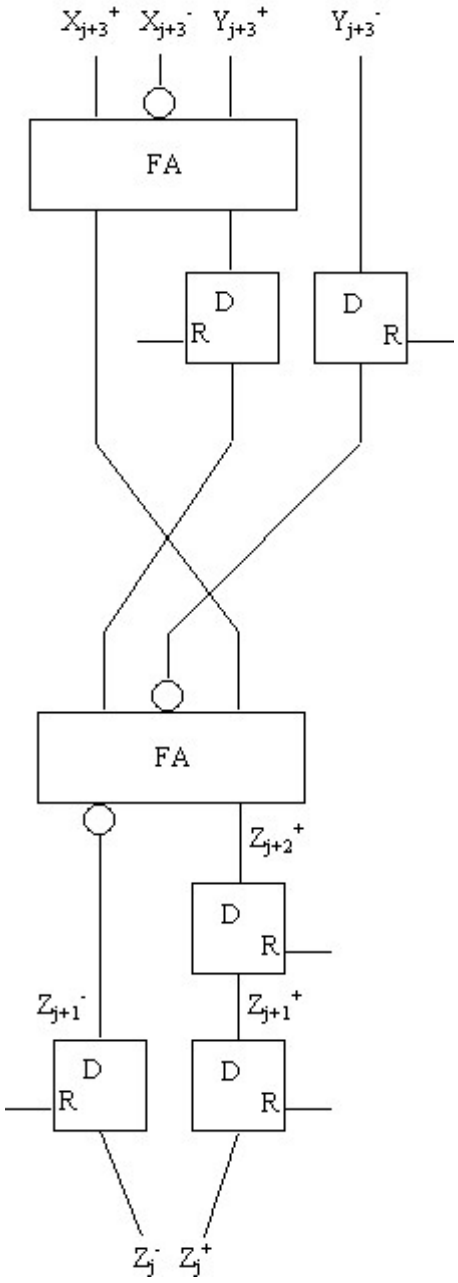

Figura 3. Sumador SD radix-2 mediante técnicas OLA.

Notaremos R como resultado de la diferencia:

$$R = A - B = \sum_{i=0}^{n-1} r_i \cdot 2^i, \quad r_i \in \left\{\bar{2},\bar{1},0,1,2\right\} \tag{4}$$

En representación SD puede ocurrir que el signo de R venga determinado por el signo del dígito *k* sí la suma parcial acumulada cumple:

$$\left|R^k\right| = \left|\sum_{i=k}^{n-1} r_i \cdot 2^i\right| \geq 2^{k+1} \tag{5}$$

Usando la definición previa de R^k:

$$R = A - B = R^k + \sum_{i=0}^{k-1} r_i \cdot 2^i \tag{6}$$

Dado que:

$$\left|\sum_{i=0}^{k-1} r_i \cdot 2^i\right| \leq \sum_{i=0}^{k-1} 2 \cdot 2^i < 2^{k+1} \quad (7)$$

Se cumple:

$$R^k \geq 2^{k+1} \quad \Rightarrow \quad R > 0 \quad (8)$$

$$R^k \leq -2^{k+1} \quad \Rightarrow \quad R < 0 \quad (9)$$

Si la condición expuesta en la ec. (4) no se cumple para $k > 0$, el signo de R no se puede garantizar hasta conocer el último dígito ($k = 0$). Vamos a definir la suma parcial acumulada normalizada como:

$$\overline{R^k} = R^k / 2^k \quad (10)$$

La condición mostrada en ec. (4) será equivalente a:

$$\left|\overline{R^k}\right| = \left|\sum_{i=k}^{n-1} r_i \cdot 2^{i-k}\right| \geq 2 \quad (11)$$

El valor de $\overline{R^k}$ puede ser calculado utilizando recurrencia 'on-line', (k varía entre N-1 y 0)

$$\overline{R^k} = 2 \cdot \overline{R^{k+1}} + (a_k - b_k) \quad (12)$$

El valor de $\overline{R^k}$ sólo depende de los valores de los dígitos previo y actual, por tanto los algoritmos tipo 'on-line' para comparar números bajo representación SD deben contemplar esta característica.

Sí la condición $\overline{R^k} \geq 2$ se cumple para cada iteración, comenzando en $k = n-1$ (MSD), o se cumple que $k = 0$, el signo de $\overline{R^k}$ será el mismo para las próximas iteraciones.

En [24] se evalúan diferentes diseños hardware para el comparador. La implementación más rápida se consigue implementando la máquina de estados cuyo diagrama de estados se presenta en la figura 4.

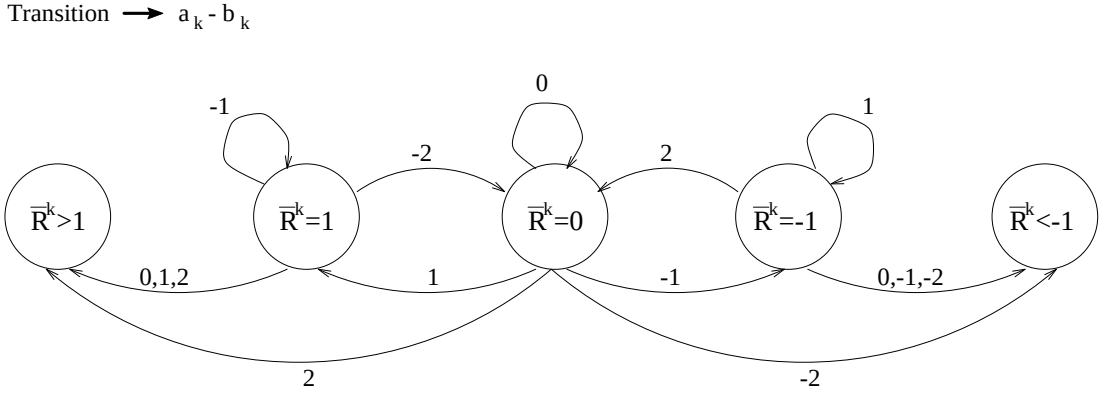

Figura 4. Diseño para la máquina de estados del comparador propuesto con técnicas de aritmética on-line.

4. IMPLEMENTACIÓN EN FPGA DEL PROCESADOR MAD

En la figura 5 mostramos la arquitectura correspondiente al procesador MAD. El valor absoluto de la diferencia es calculado para cada par de píxeles ($|c_{i,j} - r_{i,j}|$) y la sumatoria de todos ellos es calculada en el sumador N^2-OPERAND OLA ADDER. El resultado se almacena dígito a dígito en el registro SADc, y simultáneamente se realiza la comparación con el dígito correspondiente al SADr en el comparador COMP.

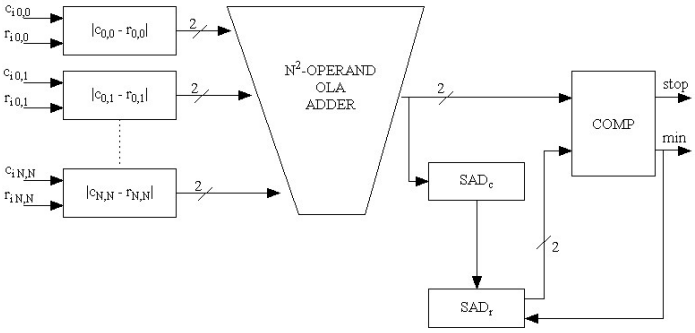

Figura 5. Arquitectura del procesador MAD

El cálculo se paraliza si se detecta SADc > SADr, a continuación se requiere un nuevo bloque candidato para comenzar su procesamiento, evitando así un elevado número de operaciones que son inevitables con representaciones y técnicas convencionales. Por otra parte, si SADc < SADr, SADc se almacena como SADr una vez que el último dígito del mismo es procesado. Si ocurre que SADc = SADr la comparación se realiza completamente, pero SADc no se almacena.

La temporización del cálculo para un procesador MAD 4x4 se presenta en la fig 6. Se consideran los tiempos correspondientes a los diferentes módulos constituyentes del sistema: cálculo de diferencias absolutas ($|c_{i,j} - r_{i,j}|$), cada uno de los cuatro pasos del árbol de sumadores, ($\Sigma(i)$) y el comparador (COMP).

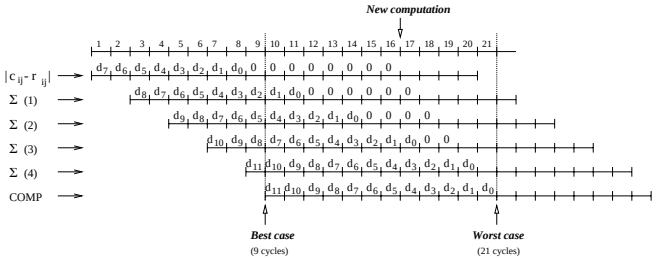

Figura 6. Temporización del MAD para un bloque 4x4.

El peor caso ocurre cuando se encuentra un nuevo SAD mínimo, entonces son necesarios 21 ciclos para el cálculo completo y almacenamiento de SADc en SADr. Aún así se puede comenzar un nuevo cálculo tras 16 ciclos de reloj, como muestra la fig. 6. Por tanto 16 ciclos es el periodo máximo para el cálculo entre SAD consecutivos. Este periodo se reduce si el SADc es rechazado por cumplir la condición SADc > SADr. El mejor caso conlleva 9 ciclos de reloj. Por tanto el número de ciclos de reloj para el cálculo, comparación y almacenamiento de un SAD estará entre 9 y 16 para un bloque 4x4. Para un bloque 8x8 este rango se establece entre 13 y 20 ciclos y entre 17 y 24 ciclos para un bloque de 16x16.

El diseño ha sido implementado en FPGA para las familias de Xilinx: SPARTAN-II y VIRTEX-II para tres tamaños diferentes de bloque: 4x4, 8x8 y 16x16; los dos últimos son los tamaños utilizados en el estándar MPEG. Para compilación, simulación e implementación se han utilizado Xilinx ISE 5.2i y ModelSim 5.6a.

Los resultados principales se muestran en la tabla 2. El ratio área/número de píxeles es relativamente bajo, debido a que las técnicas OLA permiten serializar los cálculos. La frecuencia máxima es independiente del tamaño del bloque y del número de operandos, sólo afecta al número de operaciones en paralelo y de pasos en el árbol de sumadores. Si bien los resultados están ligados a la tecnología utilizada parecen muy prometedores.

	SPARTAN II	VIRTEX II
Block size	Area (4 inputs-LUTs)	
4x4	246	241
8x8	603	595
16x16	1982	1945
Frec. Max	231.24 (MHz)	424.99 (MHz)

Tabla 2. Área y frecuencia de reloj correspondientes a las diferentes implementaciones sobre FPGA.

La tabla 3 muestra la distribución del área y de los retardos para las diferentes partes del diseño para un procesador MAD de 16x16 píxeles por bloque. El porcentaje obtenido referencia al número total de LUTs. La frecuencia máxima del sistema viene determinada por el retardo del sumador. El mayor porcentaje del área ocupada se dedica al árbol de sumadores y al cálculo de las diferencias absolutas, se debe a que son operaciones paralelizables masivamente.

	Retardo (ns)		Area	
Módulos	SP-II	VX-II	LUTs	%
D. Absolutas	3.675	1.839	1024	52.7 %
A. Sumadores	4.3246	2.353	768	39.5 %
Comparador	3.783	1.907	7	0.3 %
Control y Con.	-	-	146	7.5%

Tabla 3. Distribución del retardo y la ocupación en LUTs para el procesador MAD 16 x 16.

5. COMPARACIÓN

En esta sección se compara el diseño propuesto con otros trabajos [6] y [1] publicados recientemente.

La utilización de técnicas OLA para calcular el MAD son propuestas en [6] para una implementación ASIC. Los autores proponen un esquema general similar al nuestro, esto es, primero una conversión a SD, el cálculo de las diferencias, su conversión a valores absolutos y por último, la suma. Para la obtención de las diferencias utilizan un sumador SD, nosotros eliminamos la necesidad de dicho hardware aprovechando la conversión a SD. Esto conlleva un ahorro significativo en área y tiempo, debido a que estás operaciones hay que realizarlas para todos y cada uno de los píxeles.

Por otra parte, para la comparación del SAD proponen un algoritmo basado en técnicas OLA. En primer lugar los dos números SD son convertidos a representación signo magnitud, a continuación se utiliza una comparación para signo magnitud estándar. Dicha arquitectura introduce un retardo de dos y requiere un hardware relativamente complejo. La principal ventaja de nuestro sistema es que no introduce retardo para la comparación, de modo que se aumenta la velocidad de cálculo. Por otra parte nuestro diseño se basa en un diseño más sencillo y que requiere un menor costo hardware.

El autor no proporciona suficientes datos sobre su implementación ASIC para poder realizar una comparación cuantitativa en términos de área y retardo. En [6] se cifra el ciclo de reloj en el correspondiente a un sumador SD más un 2-to-1 MUX más una puerta two-input-AND más una puerta three-input-OR. Dado que el tiempo de ciclo en nuestro diseño se corresponde tan sólo con el de un sumador SD podemos estimar que una implementación ASIC de nuestro diseño mejoraría significativamente los resultados obtenidos en [6].

En [1] se presenta el cálculo del SAD para 16 píxeles (SAD16), esto es equivalente a una fila de un macrobloque MPEG, y su implementación en FPGA. El diseño está basado en sumadores carry-save adders (CSA) los cuales permiten el cálculo en paralelo de todos los dígitos del dato. Han presentado un diseño sintetizado utilizando FPGA Express para Sinopsis sobre un dispositivo FLEX20KE de Altera, obteniendo un área de 1699 LUTs y una frecuencia máxima de 197 MHz con 19 ciclos de latencia (96ns). Para realizar una comparación adecuada presentamos los resultados de nuestra implementación utilizando la familia VIRTEX-II, que tiene unas características relativamente similares al dispositivo utilizado en [1]. El peor caso para nuestro diseño equivalente (4x4 o 16 píxeles) ocurre cuando se encuentra un nuevo SAD mínimo, en ese caso se requieren 21 ciclos para realizar el proceso por completo (ver

sección 5), para el primer SAD evaluado y de 16 ciclos para los siguientes; esto implica 49ns para el primer caso y 38ns para los restantes a una frecuencia de 425 MHz considerando siempre el peor caso. Además, nuestro diseño requiere tan sólo 241 LUTs, esto es, siete veces menos área que en [1].

Los autores ofrecen algunas notas de cómo extender el diseño presentado en [1] para calcular un bloque 16x16 de dos formas diferentes. La primera es implementar 16 veces el módulo SAD16 e implementar un árbol de sumadores final. Para ello estiman que son necesarios 27 ciclos de reloj. Sin embargo, el número de LUTs necesarias para implementarlo estaría en torno a las 30000, esto hace inviable su síntesis en dispositivos sencillos, sólo se podría sintetizar en dispositivos de un elevado número de puertas lógicas; nuestro diseño para 16x16 requiere tan sólo 1945 LUTs, por lo que es sintetizable en una sencilla FPGA.

La segunda posibilidad presentada en [1] se base en reutilizar la unidad SAD16 para el cálculo de cada una de las filas del bloque, y acumular en un búfer la suma de los resultados. Para ello son necesarios 42 ciclos de reloj y añadir al diseño un acumulador de 16 bits. Además el comportamiento intrínseco de pipeline de las unidades SAD16 es eliminado. Para un área similar, nuestro diseño calcula, compara y almacena (si es necesario) un nuevo SAD cada 24 ciclos de reloj para el primer SAD y 16 para los restantes, contemplando siempre el peor caso.

Queremos enfatizar que nuestra comparación contempla nuestro peor caso (24 ciclos para un SAD 16x16). Si bien, el mejor caso, que ocurre cuando el SAD candidato es rechazado tras el cálculo y la comparación de su MSD, requiere tan sólo 17 ciclos de reloj para el SAD 16x16 (ver sección 4). Por otra parte, nuestros resultados incluyen la comparación, en [1] necesitarían además varios ciclos de reloj y necesitaría esperar al cálculo completo del SAD debido a la propagación del acarreo.

6. CONCLUSIÓN

Presentamos una implementación para FPGA del núcleo de la estimación de movimiento basado en el cálculo del MAD. El núcleo propuesto se puede integrar con cualquier algoritmo que implemente cualquiera de las diferentes estrategias de búsqueda utilizadas en la actualidad. Los cálculos se han realizando aplicando técnicas OLA. Se han adaptado eficientemente operaciones implicadas en el cálculo del MAD a las técnicas OLA y se ha propuesto un nuevo diseño de un comparador para representación SD que no presenta retardo. El diseño propuesto es sintetizable en cualquier FPGA del mercado, incluso en las más sencillas. Además, se acelera el cálculo gracias a la posibilidad de paralizar el cálculo de un SAD candidato cuando se demuestra que este es mayor que el SAD de referencia, esto permite adelantar el cálculo del siguiente bloque candidato. Es más, la implementación en FPGA del diseño hace posible reconfigurar el dispositivo para permitir bloques de 8x8 y 16x16 píxeles de acuerdo con los requisitos del estándar MPEG-4.

Se detallan los tiempos de ejecución y el área de ocupación del dispositivo para tamaños de 4x4, 8x8 y 16x16 píxeles por bloque. También se compara con otros trabajos relacionados, lo cual muestra los beneficios de la arquitectura que presentamos.

7. REFERENCIAS

[1] S. Wong, S. Vassiliadis, S. Cotofana "A Sum of Absolute Diferences Implementation in FPGA Hardware", **28th Euromicro Conference**, pp.183–188, Dortmund, Germany, 2002.

[2] M. Ercegovac and T. Lang, "On-Line Arithmetic for DSP Applications", **32nd Midwest Symposium on Circuits and Systems**, pp. 365–368, August, 1989.

[3] Lau, D.; Schneider, A. Ercegovac, M.D.; Villasenor, J., "FPGA-based structures for online FFT and DCT" **Proc.7th IEEE Symposium Field-Programmable Custom Computing Machines**, pp. 310–311, 1999.

[4] Brackert, R.H., Jr.; Willson, A.N., Jr.; Ercegovac, M.D."High-speed recursive digital filter using on-line arithmetic", **IEEE Int. Symposium on Circuits and Systems**, pp. 1552–1555, May 1989.

[5] S. Rajagopal and J. R. Cavallaro. "On-line arithmetic for detection in digital communication receivers". In **15th IEEE Symposium on Computer Arithmetic: ARITH-15**, pp. 257–265, Colorado, 2001.

[6] C. Su and C. Jen, "Motion Estimation Using On-Line Arithmetic", **IEEE Int. Symposium on Circuits and Systems (ISCAS-2000)**, pp. 683–686, May 28-31, 2000.

[7] Y. Chan, W. Siu "An efficient Search strategy for block Motion Estimation Using Image Features", **IEEE Trans. on Image Processing**, vol.10, pp.1223–1238, 2001.

[8] McIlhenny, R.; Ercegovac, M.D.;"On the design of an on-line FFT network for FPGA's", **33rd Asilomar Conference on Signals, Systems, and Computers**, vol. 2, pp.1484–1488, 1999.