RIELAC, Vol.XXXIII 2/2012 p. 57-71 Mayo - Agosto ISSN: 1815-5928

RIELAC
Revista de Ingeniería Electrónica, Automática y Comunicaciones

Impacto de la memoria cache en la aceleración de la ejecución de algoritmo de detección de rostros en sistemas empotrados

E. del Toro[1], A.Cabrera[2], S. Sánchez[3], A.Cabrera[4]

[1]Instituto Superior Politécnico "José Antonio Echeverría", Facultad de Ingeniería Eléctrica, ernesto@electrica.cujae.edu.cu

[2]Instituto Superior Politécnico "José Antonio Echeverría", Facultad de Ingeniería Eléctrica, alex@electrica.cujae.edu.cu

[3]Centro Nacional de Microelectrónica, Instituto de Microelectrónica de Sevilla, santiago@imse-cnm.csic.es

[4]Complejo de Investigaciones Tecnológicas Integradas, acabrera@udio.cujae.edu.cu

RESUMEN / *ABSTRACT*

En este trabajo se analiza el impacto de la memoria cache sobre la aceleración de la ejecución del algoritmo de detección de rostros de Viola-Jones en un sistema de procesamiento basado en el procesador Microblaze empotrado en un FPGA. Se expone el algoritmo, se describe una implementación software del mismo y se analizan sus funciones más relevantes y las características de localidad de las instrucciones y los datos. Se analiza el impacto de las memorias cache de instrucciones y de datos, tanto de sus capacidades (entre 2 y 16 kB) como de tamaño de línea (de 4 y 8 palabras). Los resultados obtenidos utilizando una placa de desarrollo Spartan3A Starter Kit basada en un FPGA Spartan3A XC3S700A, con el procesador Microblaze a 62,5 MHz y 64 MB de memoria externa DDR2 a 125 MHz, muestran un mayor impacto de la cache de instrucciones que la de datos, con valores óptimos de 8kB para la cache de instrucciones y entre 4 y 16kB para la cache de datos. Con estas memorias se alcanza una aceleración de 17 veces con relación a la ejecución del algoritmo en memoria externa. El tamaño de la línea de cache tiene poca influencia sobre la aceleración del algoritmo.

Palabras claves: Aceleración de algoritmo, detección de rostros, FPGA, memoria cache, Microblaze, Viola-Jones

The impact of cache memory over the speed up of Viola-Jones face detection algorithm on a FPGA embedded processing system based in Microblaze processor is analyzed in this paper. The Viola-Jones face detection algorithm is exposed and its software implementation is described, analyzing its main functions and data locality. The impact of size (among 2 and 16 kB) and line-length (between 4 and 8 words) of code and data cache memories are analyzed. Using a Spartan3A Starter Kit board, based on XC3S700A Spartan3A FPGA, with Microblaze processor running at 62,5 MHz and 64MB of DDR2 external memory running at 125 MHz, code cache shows a higher impact than data cache, with optimal values of 8kB for code cache and among 4 to 16kB for data cache. A speed-up of 17 times over external memory execution of the algorithm is achieved with these cache memories sizes. Cache line-length has little influence over the speed up of the algorithm

Ernesto del Toro Hernández, Alejandro J. Cabrera Sarmiento, Santiago Sánchez Solano,
Alejandro Cabrera Aldaya
RIELAC, Vol.XXXIII 2/2012 p. 57-71 Mayo - Agosto ISSN: 1815-5928

.

Key words: Algorithm speed up, cache memory, face detection, FPGA, Microblaze, Viola-Jones

Cache memory impact on face detection algorithm speed up in embedded systems

INTRODUCCION

La detección y reconocimiento de un rostro es una tarea que los seres humanos realizan de forma cotidiana, casi sin esfuerzo, en su vida diaria. Pero traducir a nivel algorítmico un proceso de diferenciación y reconocimiento facial es un asunto extremadamente difícil en el cual existe un amplio campo de estudio tanto en universidades y centros de investigación (MIT [1], Manchester [2], CMU [3], Cambridge, INRIA) como en grandes empresas: IBM [4], SONY [5], Intel [6], Siemens [7] y Google por solo nombrar unas pocas. También existen diversos ejemplos de sistemas biométricos comerciales desarrollados por varias empresas [8, 9]. El proceso de reconocimiento de rostros se ha hecho más relevante en la medida que la tecnología ha ido avanzando y su implementación puede encontrarse en cámaras digitales, Internet y dispositivos móviles[10], así como en aplicaciones de tecnología dedicada a la seguridad.

El primer paso en el proceso de reconocimiento de rostros es la detección de los mismos. Existen varias propuestas de algoritmos para la detección de rostros basados en diferentes estrategias. Schneiderman y Kanade[11] proponen un método estadístico para detectar rostros utilizando histogramas para representar una gran variedad de atributos visuales. Otro método de detección de rostros puede ser visto en [12], donde H. Rowley, S. Baluja y T. Kanade entrenan dos sistemas de clasificación basados en redes neuronales: el primero realiza una estimación de la pose de un rostro en una imagen y el segundo se encarga de la detección de rostros en sí. Existen otras propuestas donde se realiza la detección de rostros teniendo en cuenta las texturas y colores de la piel [13,14].

Uno de los algoritmos de detección de rostros más ampliamente utilizado es el propuesto por P. Viola y M. Jones [15]. Este algoritmo, basado en etapas de clasificación de complejidad creciente, es considerado como uno de los mejores en cuanto a tiempo de ejecución y efectividad de detección [16-19].

Muchos de los sistemas de detección e identificación de rostros son implementados mediante un programa que se ejecuta sobre plataformas de procesamiento convencionales (computadoras personales, servidores, etc.) en donde la velocidad de sus procesadores y la capacidad de sus recursos disponibles permiten obtener elevadas velocidades de procesamiento. Sin embargo, para aquellas aplicaciones en donde existan limitaciones de tamaño, costo y consumo de potencia, es necesario recurrir a la utilización de sistemas empotrados como plataforma para la implementación de los algoritmos de detección e identificación de rostros, con recursos y velocidades de ejecución considerablemente inferiores a las de los sistemas de procesamiento convencionales.

La utilización de un FPGA (Field Programmable Gate Array) como soporte hardware para la implementación de un sistema empotrado de detección de rostros permite desarrollar diferentes estrategias de implementación: totalmente software, en donde todo el algoritmo es ejecutado por un sistema de procesamiento empotrado; totalmente hardware; en donde todo el procesamiento se implementa mediante arquitecturas de procesamiento específicas; e híbrida hardware-software, en donde partes del algoritmo (usualmente las de mayor consumo de tiempo) se implementan en hardware y las restantes son ejecutadas por un procesador empotrado [20-27]. Cada una de estas variantes de implementación tiene sus ventajas y desventajas. Las primeras, en contraposición a las segundas, son las que mayor flexibilidad ofrecen al permitir la utilización de diferentes tamaños de imágenes y parámetros de detección, aunque son las de menor velocidad de respuesta. Las realizaciones híbridas son las que permiten establecer un mejor compromiso entre flexibilidad y velocidad de la solución.

El disponer de un sistema empotrado basado en FPGA, los cuales pueden incorporar una gran cantidad y variedad de recursos internos (bloques de memoria RAM, multiplicadores, procesadores, etc.), permite desarrollar sistemas de procesamiento altamente configurables y evaluar así el impacto de diferentes estructuras sobre la velocidad de ejecución de un algoritmo. En las implementaciones software de los algoritmos de detección e identificación de rostros se requiere la utilización de grandes capacidades de memoria para el almacenamiento de los datos y código. Aunque los FPGA disponen internamente de recursos de memoria, son insuficientes para almacenar todo el programa y sus datos (a no ser que se utilicen costosos FPGA considerados de gama alta [28]) por lo que es necesaria la utilización de memoria externa de gran capacidad de almacenamiento, usualmente memoria dinámica. La utilización de memoria dinámica externa al FPGA implica que el acceso a la misma es mucho más lento que el acceso a la memoria interna del FPGA, por lo que la utilización de los recursos de memoria interna como memoria cache permite incrementar significativamente la velocidad de ejecución de los algoritmos en un sistema empotrado.

Aunque existen trabajos que mencionan la utilización de memoria cache en diferentes implementaciones sobre FPGA del algoritmo de detección de rostros de Viola-Jones, no se ha encontrado alguno que analice el impacto de los diferentes parámetros de la misma sobre la aceleración de la ejecución de este algoritmo [29-33].

En este artículo se expone el impacto de algunos de los principales parámetros de la memoria cache de un sistema de procesamiento empotrado en FPGA sobre la aceleración de la ejecución del algoritmo de detección de rostros de Viola-Jones con relación a su implementación en memoria dinámica externa al FPGA. Primeramente se describe el algoritmo de detección de rostros de Viola-Jones y se expone una implementación software del mismo. A continuación se describe el sistema empotrado sobre el que se realizó la parte experimental de la investigación mediante la implementación de diferentes configuraciones de memoria cache de instrucciones y de datos. Luego se analizan las principales funciones del algoritmo así como las características de los datos con los que trabaja. En el siguiente apartado se exponen los parámetros del sistema de memoria cache del procesador Microblaze que pueden incidir en el incremento de la velocidad de la ejecución de un programa así como los resultados de la aceleración del algoritmo de detección de rostros de Viola-Jones en función de las capacidades de ambos tipos de memoria cache, así como de la longitud de la línea de las mismas. Finalmente se exponen las conclusiones del trabajo.

ALGORITMO DE DETECCIÓN DE ROSTROS VIOLA-JONES

La detección de rostros en una imagen puede ser una tarea de alto costo computacional en función del algoritmo que se utilice. Una forma de implementar un algoritmo de detección de rostros consiste en ir recorriendo la imagen mediante una ventana de un determinado tamaño, la cual puede contener un rostro (candidato). Un aspecto a considerar es la forma de evaluar si la ventana contiene o no un rostro. Después de recorrer la imagen, la ventana puede ser escalada y repetir el proceso con el objeto de detectar posibles rostros de mayor tamaño que no fueron detectados con tamaños de ventana inferiores.

El algoritmo de detección de rostros propuesto por P. Viola y M. Jones, basado en la utilización de ventanas deslizantes, es sin duda el más utilizado, no sólo en tareas de detección de rostros sino también de detección de diversos objetos en general. Los aspectos fundamentales en que se basa este algoritmo son los siguientes [15]:

1. Utilización de rasgos de clasificación
2. Utilización de una imagen integral
3. Organización en cascada de los clasificadores

Rasgos de clasificación. Los rasgos de clasificación son las formas geométricas utilizadas por el algoritmo para detectar zonas de una imagen (o ventana) que pueda contener partes de un rostro. Los rasgos de clasificación utilizados por el algoritmo Viola-Jones son estructuras simples compuestas por dos, tres o cuatro rectángulos grises y blancos, como los mostrados en la Figura 1. Nótese que los rasgos pueden tener 6, 8 ó 9 puntos significativos, correspondientes a las esquinas de cada rectángulo.

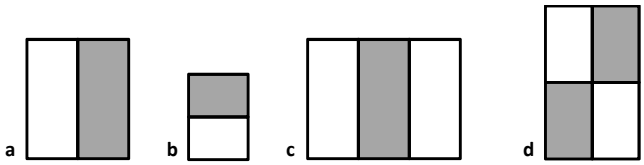

Figura 1. Ejemplos de rasgos de 2 (a y b), 3 (c) y 4 (d) rectángulos utilizados para realizar la detección de rostros

Estas estructuras simples pueden ser asociadas a partes comunes de un rostro, como las correspondientes a los ojos, la nariz, la frente, el pelo, etc. La Figura 2 ilustra la correspondencia de dos posibles rasgos de clasificación con partes de un rostro.

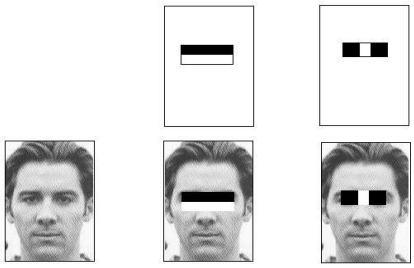

Figura 2. Ilustración de dos rasgos de clasificación y su correspondencia con partes de un rostro

Como el número de rasgos que se pueden corresponder con partes de un rostro en una imagen de un determinado tamaño es considerablemente elevado, Viola y Jones utilizaron un algoritmo de entrenamiento (conocido como AdaBoost [34]) sobre un gran conjunto de imágenes con y sin rostros, para seleccionar aquellos rasgos que mejor distinguen las zonas de una imagen que contienen un rostro [15].

Los rasgos no sólo se caracterizan por su forma, sino también por su tamaño y posición dentro de una ventana así como por la posible contribución de los mismos a la detección de un rostro. Para ello es necesario calcular el *valor del rasgo*, parámetro que se obtiene de la diferencia entre las intensidades de los puntos de las zonas blancas y grises de un rasgo. Si el valor de un rasgo sobrepasa un determinado umbral (*umbral de clasificación*), se considera que contribuye con un determinado valor *alpha* a la detección de un rostro. Todo este conjunto de parámetros (y otros que se expondrán más adelante) forman parte de lo que se conoce como un *clasificador* [15].

Nótese que el cálculo del valor del rasgo puede ser un proceso computacionalmente costoso en tiempo si fuese preciso recorrer todos los puntos del rasgo para su evaluación. La segunda contribución del algoritmo de Viola-Jones reduce este problema.

Imagen integral. Una contribución fundamental del algoritmo de Viola-Jones consiste en acelerar el cálculo del valor del rasgo al trabajar no con la imagen original sino con una imagen integral. Esta no es más que una transformación de la imagen original en donde cada punto de la misma toma el valor de la suma de todos los puntos que están ubicados por encima y a su izquierda. Así, se puede definir una imagen integral ***ii*** según la ecuación 1:

$$ii(x,y) = \sum_{x' \leq x,\, y' \leq y} i(x',y') \tag{1}$$

En donde ***ii(x, y)*** es el punto de la imagen integral en las coordenadas (x,y) e ***i(x, y)*** es el punto de la imagen original en las mismas coordenadas. La Figura 3a ilustra una imagen original, mientras que la 3b ilustra la imagen integral correspondiente.

La imagen integral se puede calcular en un solo recorrido de la imagen original realizando las operaciones descritas en las ecuaciones 2 y 3:

$$s(x,y) = s(x,y-1) + i(x,y) \tag{2}$$

$$ii(x,y) = ii(x-1,y) + s(x,y) \tag{3}$$

En donde $s(x,y)$ es la suma de toda una fila considerando que $s(x,-1) = 0$ e $ii(-1, y) = 0$.

A partir de la imagen integral se puede calcular rápidamente la suma de todos los puntos contenidos en un rectángulo cualquiera de la imagen utilizando sólo los cuatro valores asociados a sus esquinas, tal como se ilustra en la Figura 3c. Esta característica permite que el cálculo de la suma de los puntos contenidos en un rectángulo de tamaño arbitrario pueda ser realizado en un tiempo constante utilizando sólo cuatro operaciones, por lo que el cálculo del valor de un rasgo se reduce considerablemente.

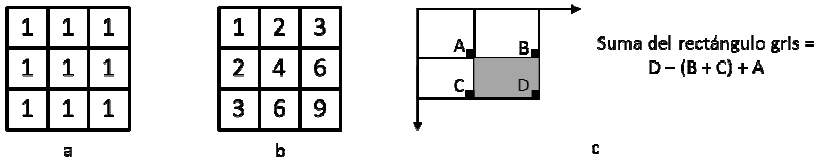

Figura 3. Cálculo de la imagen integral. (a) Imagen original (b) Imagen integral (c) Valor del rectángulo utilizando la imagen integral

A partir de la imagen integral es muy simple la determinación del valor de un rasgo. Conociendo los valores de la imagen integral en los puntos significativos de un rasgo y las características del rasgo, la determinación del valor de un rasgo se reduce a simples operaciones de multiplicación (por coeficientes constantes) y suma. La Figura 4 ilustra este proceso, en donde se muestra un rasgo formado por dos rectángulos y se detalla la determinación de su valor. Nótese que los

coeficientes por los cuales es necesario multiplicar los valores de la imagen integral de los seis puntos significativos de este rasgo son constantes. Al conjunto de estas constantes (6, 8 ó 9 valores según el rasgo) se denomina *vector de pesos del rasgo* (W) y es otro de los parámetros que forman parte de un clasificador.

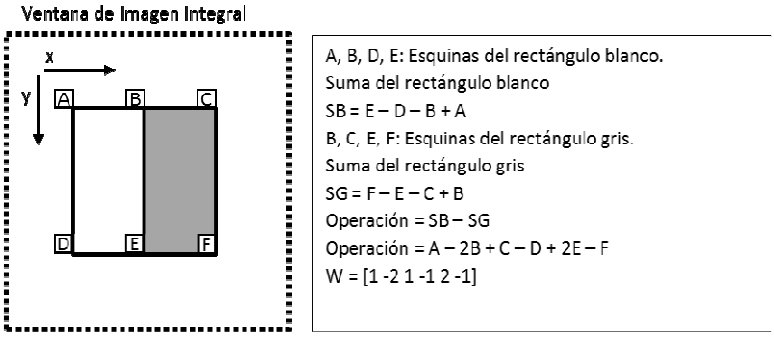

Figura 4. Ilustración del cálculo del valor de un rasgo y del vector W de pesos del rasgo

Organización en cascada de los clasificadores.

La tercera característica del algoritmo Viola-Jones es la utilización de una combinación en cascada de grupos de clasificadores cada vez más complejos (con mayor número de rasgos). Las primeras etapas de la cascada poseen pocos clasificadores pero permiten descartar rápidamente aquellas ventanas que no contienen rostros, concentrando el esfuerzo computacional en las etapas siguientes en aquellas con mayor probabilidad de contener un rostro.

La Figura 5 ilustra una posible distribución de clasificadores en una cascada de cuatro etapas. La cantidad de etapas y de clasificadores en cada etapa dependen de la base de datos de clasificadores utilizada para la el proceso de detección de rostros [3, 35].

En la primera etapa, para cada una de las ventanas de la imagen se calculan sus tres clasificadores y se suman los aportes *alpha* de los mismos. Si este valor es inferior al *umbral de la etapa*, se descarta que la ventana contenga un rostro y se procesa la siguiente. Si es superior, la ventana se procesa con los nueve clasificadores de la segunda etapa, y así sucesivamente hasta que la ventana sea descartada en alguna de las etapas o, si las supera todas, sea identificada como un posible rostro. Los valores del umbral de cada etapa se obtienen también del proceso de entrenamiento [15].

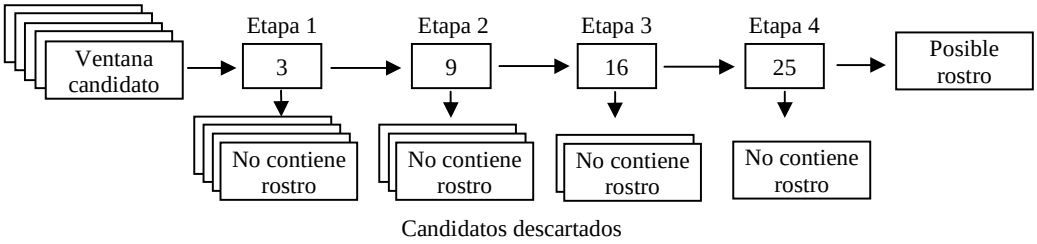

Figura 5. Ilustración de una cascada de clasificadores

Una vez expuestos los aspectos fundamentales en que se basa el algoritmo de detección de rostros de Viola-Jones se puede resumir el mismo. Se comienza por procesar una ventana evaluando los clasificadores de las etapas. Si la ventana no es descartada en alguna de las etapas, se marca como un posible rostro. El proceso continúa desplazando la ventana (en sentido horizontal o vertical) según un *factor de escaneo* determinado. Una vez que se haya recorrido toda la imagen con un tamaño de ventana determinado, se incrementa el tamaño de la ventana (según un *factor de escalado* determinado) y se repite el procesamiento de las etapas y de barrido de la imagen. Los valores de los factores de escaneo y de escalado son obtenidos también durante el proceso de entrenamiento [15].

IMPLEMENTACIÓN SOFTWARE DEL ALGORTIMO VIOLA-JONES

El algoritmo completo se puede expresar en pseudocódigo como se muestra en la Figura 6. La función **imagen_integral** realiza el cálculo de la imagen integral. Luego se entra en un doble ciclo for anidado que se encarga de realizar el barrido por filas (***for (i=0; i < Image.Width-size; i+=P)***) y por columnas (***for (j=0; j <Image.Height-size; j+=P)***) según el factor de escaneo ***P***.

La función **pre-process** realiza un descarte rápido de aquellos candidatos que son muy homogéneos o que tienen regiones muy disímiles. Mediante el cálculo de la desviación estándar los candidatos que no entran en un intervalo de valores

determinados en el proceso de entrenamiento de los clasificadores no se pasan al detector [15]. Si el resultado del pre-procesamiento es positivo (***pre_process = true***) se pasa a la evaluación del candidato en cada una de las etapas. Este proceso se describe en la llamada a la función **ejecutar_detección**. El resultado en la ejecución de esta función se utilizará para decidir si el candidato se considera como rostro y se añade a la lista de los rostros detectados en la etapa.

```
Entrada:
Image : Imagen en escala de grises, en una estructura con propiedades de tamaño (size) = ancho * alto (Width*Heigth).
Ventana_detección : candidatos que se generan como porciones de la imagen integral con sus mismos parámetros.
S : Factor de escalado// incremento en el tamaño de las ventanas
P : Factor de escaneo // pasos en el recorrido de la ventana
Función:
ii = imagen_integral (Image)
while (size < Image.size) do
    for (i=0; i < Image.Width- ventana_detección.Width; i+=P)
        for (j=0; j < Image.Height- ventana_detección.Height; j+=P)
          If pre-process (ventana_detección en ii(i,j)) then
          resultado = ejecutar_detección (ventana_detección en ii(i,j))
            If resultado = true then
                Lista_añadir_rostro (ventana_detección, i, j)
            end if
          end if
        end for
    end for
    size = size * S //incrementar el tamaño de los candidatos
end while
```

Figura 6. Pseudocódigo del algoritmo de detección de rostros de Viola-Jones

Después que todos los candidatos de un tamaño determinado hayan sido generados y analizados, se incrementa el tamaño de la ventana deslizante según el factor de escalado establecido mediante la operación ***size = size*S*** y se comienza el barrido de la imagen otra vez hasta que se llegue a un límite prefijado con antelación, en este caso el tamaño de la imagen (***Image.size***).

La función **ejecutar_detección** se puede expresar en pseudocódigo como se muestra en la Figura 7. En esta función se utiliza un ciclo ***for*** especial (***foreach***) para destacar que se evalúan los clasificadores en cada etapa mediante el cálculo de los puntos significativos de los rasgos en la imagen integral. Dentro de cada etapa se realiza el cálculo de cada clasificador en la etapa (***foreach (clasificador) in etapa***) llamando a la función **calcula_clasificador**. Si el resultado de esta operación es mayor que el umbral de clasificación entonces el clasificador aporta un valor *alpha* a la suma de los valores de cada etapa (***suma_del_nivel***). Si el acumulado de los clasificadores de la etapa sobrepasa el umbral de la etapa (***stage.umbral***) entonces se pasaría a procesar los clasificadores de la siguiente etapa y se repetiría el proceso hasta que el candidato sea descartado en alguna etapa o sea marcado como rostro.

```
Entrada:
Ventana_detección
Función:
foreach (etapa) do
    suma_del_nivel = 0
    foreach (clasificador) in etapa do
        clasificador.resultado = calcula_clasificador (clasificador)
        if (clasificador.resultado > clasificador.umbral) then
            suma_del_nivel += clasificador.alpha
        end if
    end foreach
    if (suma_del_nivel < etapa.umbral) then
        return false
    end if
end foreach
return true
```

Figura 7. Pseudocódigo de la función que realiza la detección de rostros

La función **calcula_clasificador** se puede expresar en pseudocódigo como se muestra en la Figura 8. Esta función realiza el cálculo del rasgo de un clasificador que tiene nueve puntos significativos, acumulando el producto de cada punto de la imagen integral ***ii(x,y)*** por su peso ***W(i)*** correspondiente. En el caso que el rasgo contenga sólo seis u ocho puntos significativos, los restantes coeficientes ***W(i)*** correspondientes tendrían el valor cero por lo que no influyen en la operación de cálculo.

```
Entrada:
ii : ventana de detección en formato de imagen integral
clasificador
Función:
fval = 0
For (i = 0; i < 9; i++)
    x = clasificador.x(i)
    y = clasificador.y(i)
    fval = fval + ii(x,y) * clasificador.W(i)
end for
return fval
```

Figura 8. Pseudocódigo de la función que realiza la evaluación de los clasificadores

En la implementación del algoritmo, desarrollado en lenguaje C++, se utilizó la base de datos de los clasificadores obtenidos de la Universidad Carnegie Mellon que son distribuidos junto al sensor de visión CMUcam utilizado en este trabajo de investigación [36]. Esta base de datos forma parte de una distribución simple (244 clasificadores distribuidos en cinco etapas) que está concebida fundamentalmente para ser implementada en sistemas empotrados de bajo costo con requerimientos de poco consumo de potencia y baja velocidad de procesamiento. La misma proporciona toda la información de los datos de los clasificadores: desplazamientos ***x*** y ***y*** de los puntos significativos del rasgo dentro de la ventana; pesos ***W(i)*** del rasgo; umbral del clasificador, etc. También se incluyeron como datos constantes un conjunto de 17 imágenes de tamaño 352 x 288 puntos, extraídas de la base de datos de CMU y que incluyen uno o más rostros [3].

SISTEMA EMPOTRADO UTILIZADO

Para la implementación en un sistema basado en FPGA del algoritmo de detección de rostros de Viola-Jones se diseñó un sistema de procesamiento empotrado basado en el procesador Microblaze de Xilinx [37]. Microblaze es un módulo de propiedad intelectual (IP) totalmente sintetizable (para las FPGA actuales de Xilinx) de un procesador RISC (Reduced Instructions Set Computer) de 32 bits con arquitectura Harvard. El sistema de procesamiento Microblaze dispone de una gran colección de módulos IP de periféricos, controladores de memoria y buses, totalmente configurables y parametrizables, que permiten el desarrollo de un sistema empotrado con los recursos apropiados para una determinada aplicación [37].

La Figura 9 muestra un esquema del sistema de procesamiento que se implementó para esta investigación. El mismo está formado por un procesador Microblaze conectado a un controlador de memoria externa así como a un temporizador, a un transmisor serie asincrónico (UART) y a un controlador de interrupciones a través del bus de expansión PLB.

El controlador de memoria externa se requiere para la conexión de una memoria dinámica externa que sea capaz de almacenar todos los datos y el código del programa. El temporizador será utilizado para la medición del tiempo de ejecución del algoritmo en las diferentes configuraciones de memoria cache del procesador Microblaze desarrolladas en la parte experimental de la investigación, mientras que el UART será utilizado para transmitir hacia una computadora personal las mediciones de tiempo realizadas con el temporizador. El temporizador y el UART son atendidos por interrupción, por lo que sus respectivas solicitudes se envían al controlador de interrupciones y de este al procesador Microblaze.

Las estructuras de memoria cache separadas de instrucciones y de datos forman parte de la estructura del procesador Microblaze y permiten intercambiar información con la memoria externa a través del controlador de memoria (conexión no mostrada en la Figura 9).

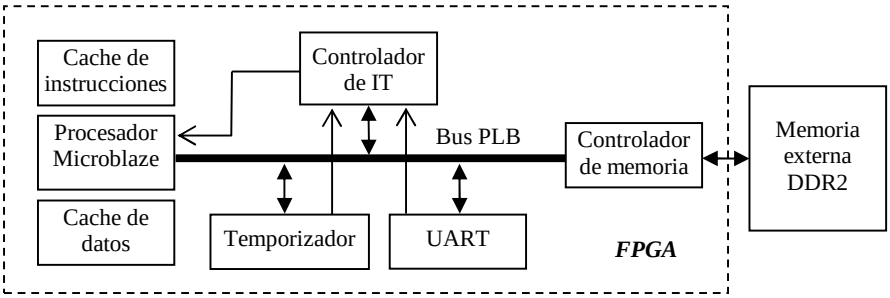

Figura 9. Sistema de procesamiento empotrado basado en el procesador Microblaze

Como soporte para el sistema empotrado anterior se ha utilizado una placa de desarrollo Spartan3A Starter Kit, basada en un FPGA Spartan3A XC3S700A el cual cuenta con 20 bloques de memoria RAM (BRAM) de 18kb configurables, así como 5.888 *slices* (cada *slice* contiene dos tablas de búsqueda de 4 entradas y dos biestables, entre otros recursos lógicos). Esta placa de desarrollo incorpora también una memoria dinámica DDR2 de 64 MB, la cuál puede ser accedida mediante el controlador de memoria empotrado en el FPGA [38].

El sistema fue implementado utilizando una frecuencia de reloj de 62,5 MHz para el procesador Microblaze y de 125 MHz para la memoria externa DDR2. Los detalles de la implementación escapan a los objetivos de este artículo por lo que no serán expuestos.

El código completo del programa compilado para el procesador empotrado Microblaze, incluyendo las funciones del preprocesamiento de la imagen, las funciones relacionadas con la medición de los tiempos de ejecución del algoritmo (mediante el temporizador) y las de transmisión de la información mediante el UART hacia la computadora personal, ocupa algo más de 15kB, mientras que los datos ocupan algo más de 35MB, debido fundamentalmente a los valores digitalizados de las 17 imágenes procesadas (algo más de 34MB) así como a los datos de los clasificadores y valores de la imagen integral. Dadas las limitaciones de los recursos de memoria interna del sistema empotrado utilizado, es necesario ubicar los datos en memoria externa. Aunque el tamaño del código de esta implementación no implica la necesidad de que sea ubicado en memoria externa, debe tenerse en cuenta que este algoritmo es sólo una parte de un proceso más complejo (como el de identificación de rostros) por lo se ubica también en memoria externa.

ANÁLISIS DE LAS FUNCIONES Y DATOS DEL ALGORITMO

Independientemente del sistema de memoria cache que se utilice, la propia estructura de un programa (organización de su código y de sus datos) incide en la efectividad de un sistema de memoria cache. Mientras más cumpla el programa con los principios de localidad espacial (acceso a localizaciones de memoria cercanas a las recientemente accedidas) y temporal (acceso en breve tiempo a la misma localización de memoria) mayor será la tasa de aciertos (relación entre el número de veces que la información se encuentra en cache con relación al número total de accesos a memoria) de un sistema de memoria cache y mayor su contribución al incremento de velocidad en la ejecución de un programa.

Es por ello que para realizar el análisis del impacto de la memoria cache sobre el incremento de la velocidad de ejecución del algoritmo de detección de rostros de Viola-Jones con relación a su ejecución desde memoria externa, es conveniente analizar las características de las funciones y datos del mismo.

Con objeto de evaluar la distribución del tiempo de ejecución entre las distintas partes del algoritmo se utilizó la herramienta de análisis de rendimiento (profiling) que incorpora el software de desarrollo de Microblaze [37]. Se trata de una herramienta intrusiva que hace que el compilador añada al código del programa una función que interrumpe la ejecución del mismo con una frecuencia determinada y guarda en un espacio de memoria reservado el contenido del contador de programa. Para realizar el análisis la herramienta utiliza el temporizador que esté implementado en el sistema. Por este motivo se reelaboró el programa para dejar libre al temporizador, se incluyeron las bibliotecas para el análisis del rendimiento, se recompiló el programa y se realizó un ciclo de ejecución del algoritmo una sola vez por cada una de las 17 imágenes.

Una vez finalizada la ejecución se realizó un análisis de los datos obtenidos y se determinó qué función corresponde con cada valor del contador de programa. A partir de este análisis se determinó cuánto tiempo estuvo el procesador ejecutando cada una de las funciones, cuántas llamadas a la función se hicieron y otros datos estadísticos. Los resultados se pueden visualizar en un gran número de formatos, como el diagrama que se muestra en la Figura 10 en donde aparece el porcentaje de tiempo que está ejecutándose cada función en relación con el tiempo total de ejecución del programa.

La zona número uno corresponde a la función que realiza el cálculo del rasgo de un clasificador (**calcula_clasificador**). Como se puede apreciar esta ocupa casi la mitad del tiempo total de ejecución del programa. La zona número dos corresponde a la función que realiza el cálculo de la imagen integral y la número tres al resto del algoritmo. La zona número cuatro está asociada a otras funciones no directamente relacionadas con la detección de rostros (comunicación con el puerto serie, visualización de los resultados, etc.).

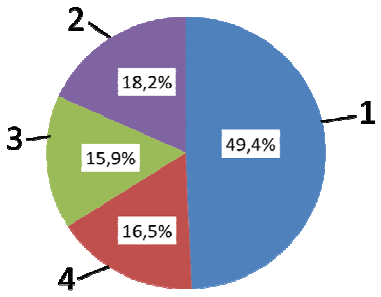

Figura 10. Distribución en tiempo de las funciones del algoritmo de detección de rostros

El gran impacto en tiempo de la función del cálculo del rasgo de un clasificador se debe a la gran cantidad de veces que es ejecutada. Sin embargo, nótese que es una función muy simple (Figura 8). Dado que ejecuta una operación iterativa, la ubicación de esta función en memoria cache de instrucciones permite explotar las características de localidad espacial y temporal del código de la misma y obtener una elevada tasa de aciertos en el acceso a la memoria cache de instrucciones.

En esta función es necesario realizar una gran cantidad de accesos a datos en memoria en cada iteración. Se realizan accesos a memoria para obtener los valores de los desplazamiento ***x*** y ***y*** de cada uno de los nueve puntos significativos de un rasgo. Con estos valores se calcula la dirección de memoria donde se encuentra el valor de la imagen integral de cada uno de estos nueve puntos y se accede a los mismos. También se accede a memoria para leer los valores de cada uno de los pesos ***W(i)*** del rasgo. En total se realizan 36 accesos a memoria sólo para el cálculo del valor de un rasgo.

Si bien los datos correspondientes a los desplazamientos ***x*** y ***y***, así como los de los pesos ***W(i)*** del rasgo se organizan en estructuras que permiten sacar provecho de la localidad espacial (datos en direcciones cercanas a las recientemente accedidas) y pueden estar ubicados en una misma línea de la cache de datos, esto no ocurre así con el acceso al valor de la imagen integral en los puntos significativos del rasgo. Dado que estos puntos no están ubicados en localizaciones consecutivas y se distancian más a medida que se incrementa el tamaño de la ventana, es de esperar que la mayor parte de los fallos (*cache miss*) en el acceso a la cache de datos en la ejecución de esta función se deban a los acceso a los valores de la imagen integral.

La función que calcula la imagen integral se ejecuta una sola vez para cada imagen, debiendo leer de memoria externa en direcciones consecutivas donde se encuentran almacenados los valores de la imagen original y escribir también en memoria externa los valores de la imagen integral. Dado que los valores de la imagen integral deben permanecer en memoria externa, no se sacaría provecho de utilizar una política de escritura obligada (*write back*) por lo que se emplea la de escritura inmediata (*write through*), de estructura hardware más simple, al implementar la memoria cache de datos.

IMPACTO DE LA MEMORIA CACHE

Son varios los parámetros de diseño de un sistema de memoria cache que pueden influir en el incremento de la tasa de acierto de la misma y, por consiguiente, en el incremento de velocidad en la ejecución de un programa. Entre estos parámetros se encuentran la propia estructura de la cache (cache de instrucciones y datos separadas o unificadas), la capacidad de la memoria cache, su esquema de organización, el tamaño de la línea de cache y la política de escritura utilizada.

En el sistema de memoria cache para el procesador Microblaze algunos de estos parámetros están predeterminados y no pueden ser configurados. Se utilizan estructuras de cache de instrucciones y de datos separadas para optimizar la operación de sus etapas de pipeline. Ambas estructuras de memoria cache utilizan un esquema de organización de mapeo directo, el menos eficiente de los existentes para una cache aunque el más simple de implementar.

La capacidad de las memorias cache de datos y de código puede ser configurada desde 64bytes a 64kB. Por supuesto, los valores máximos de las mismas dependerán de los recursos de memoria (sobre todo de BRAM) del FPGA utilizado. Los tamaños de las líneas es otro parámetro que puede ser configurado en ambas memorias, con valores de 4 u 8 palabras (16 ó 32 bytes). Por último, la política de escritura utilizada para la cache de datos es también configurable, pudiendo ser de escritura obligada (*write back*) o inmediata (*write through*). Considerando lo apuntado con relación a los datos del programa, en esta investigación sólo se ha considerado analizar diferentes capacidades de ambas memorias y diferentes tamaños de línea.

Para evaluar el impacto de ambas estructuras de memoria sobre la aceleración del algoritmo de detección de rostros de Viola-Jones se realizó una primera implementación del sistema de procesamiento sin memoria cache por lo que todos los accesos a las instrucciones y a los datos se realizan en memoria externa. Los resultados de la implementación hardware del sistema empotrado y del tiempo de ejecución del programa sobre las 17 imágenes de la base de datos de CMU se muestran en la segunda columna de la Tabla 1. La cantidad de *slices* utilizados en esta implementación representa un 45,7% de los del FPGA. Los 6 bloques de memoria RAM (30% del total) son utilizados por el controlador de memoria externa y por el propio procesador Microblaze.

Nótese que la demora promedio en la ejecución del programa sobre una imagen es de más de 26 segundos, valor que se utilizó como referencia para determinar la aceleración de la ejecución del programa con diferentes configuraciones de memoria cache.

Tabla 1. Resultados experimentales con I-Cache de 2 kB y D-Cache entre 2 y 16 kB

Cache de instrucciones	0	2	2	2	2
Data Cache de datos	0	2	4	8	16
BRAMs	6	8	9	11	15
Slices	2.691	2.908	2.767	2.747	2.718
Pulsos del temporizador	1.634.536.215	179.274.217	176.005.196	174.894.711	173.752.431
Demora promedio (seg)	26,152	2,868	2,816	2,798	2,78
Aceleración	1	9,11	9,28	9,34	9,40

A partir del sistema anterior se realizaron un total de 32 implementaciones del sistema de procesamiento utilizando diferentes configuraciones de memoria cache con capacidades entre 2 y 16kB para cada una y tamaños de línea de 4 y 8 palabras. No fueron consideradas capacidades inferiores de cache por no ser relevantes para la investigación. La capacidad máxima de ambas memorias cache está limitada por la cantidad de bloques de memoria disponibles en el FPGA (20).

Las restantes columnas de la Tabla 1 muestran un ejemplo en donde aparecen los resultados de implementaciones con capacidades de 2kB de cache de instrucciones y cache de datos entre 2 y 16 kB, utilizando 4 palabras por línea. Nótese el incremento en la cantidad de bloques de RAM utilizados debido a ambas memorias cache. La cantidad de *slices* varía ligeramente entre las diferentes implementaciones debido a las variaciones en los controladores de las memorias cache. La aceleración alcanzada en estos casos es de algo más de nueve veces con relación a la ejecución del programa sin memoria cache. La Figura 11 muestra gráficamente los resultados de demora de ejecución y aceleración en función del tamaño de la cache de datos.

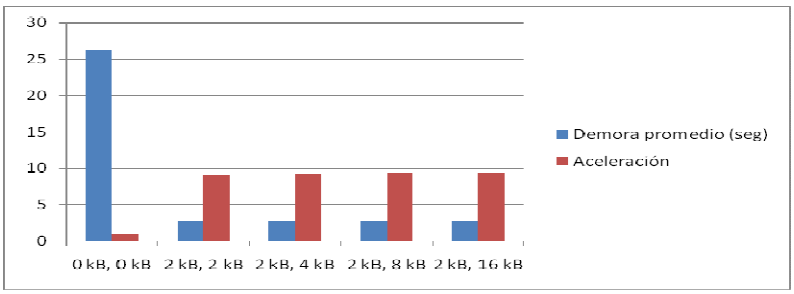

Figura 11. Ilustración de la demora de ejecución y de la aceleración con I-Cache de 2 kB y D-Cache entre 2 y 16 kB

La Tabla 2 muestra los resultados de aceleración para capacidades entre 2 y 16 kB de ambas memorias cache, con 4 palabras por línea, mientras la Figura 12 los ilustra gráficamente. Analizando las mismas se pueden obtener conclusiones acerca del impacto de cada una de las memorias cache.

Nótese en la Tabla 2 como con capacidades de cache de instrucciones entre 2 y 4 kB se obtienen resultados se aceleración similares (algo más de 9 veces) para los diferentes tamaños de memoria cache de datos. En este rango de valores de la memoria cache de instrucciones, duplicar la capacidad de la misma no incrementa mucho la tasa de aciertos y por consiguiente la aceleración de la ejecución del programa.

Sin embargo, al duplicar la cache de instrucciones de 4kB a 8kB se observa un incremento de la aceleración del programa entre 15,42 y 16,99 veces, lo cual equivale a incrementos de la aceleración entre un 68 y un 80%. Esto se debe a que con 8kB de cache de instrucciones, la mayor parte de las funciones iterativas del programa pueden radicar en cache, incrementando así la tasa de aciertos. Nótese también que incrementar la cache de instrucciones por encima de 8kB no repercute en un incremento de la aceleración del programa, por lo que el valor recomendado para el tamaño de esta cache es de 8kB.

Tabla 2. Resultados de aceleración para 16 configuraciones diferentes de cache con 4 palabras por línea

I-Cache / D-Cache	2	4	8	16
2	9,11	9,28	9,34	9,40
4	9,14	9,32	9,38	9,44
8	15,42	16,15	16,42	16,99
16	15,42	16,15	16,42	17,02

Con relación a la cache de datos, se puede apreciar que, para un tamaño de cache determinado, la duplicación de su capacidad no repercute significativamente en la aceleración del algoritmo. Obsérvese la fila 3 de la Tabla 2, en donde con 8kB de cache de instrucciones, la aceleración varía sólo alrededor de un 10% (entre 15,42 y 16,99 veces) para capacidades de la cache de datos entre 2 y 16kB.

Este menor impacto de la cache de datos era de esperar, dado que el acceso a los valores de la imagen integral en los puntos significativos de los rasgos están en direcciones más dispersas, lo cual reduce la tasa de aciertos al disminuir la localidad espacial de estos datos y ser de poca utilidad la localidad temporal de los mismos.

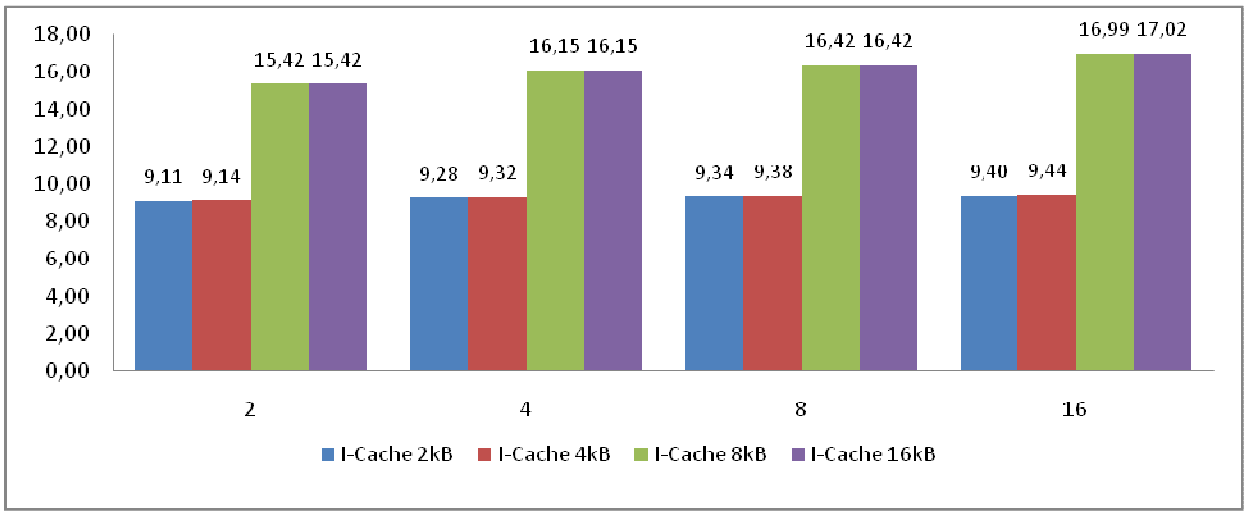

Figura 12. Resultados de aceleración para 16 configuraciones diferentes de cache con 4 palabras por línea

También se realizaron otras 16 implementaciones similares con un tamaño de 8 palabras por línea. La Tabla 3 muestra los resultados de aceleración para capacidades entre 2 y 16 kB de ambas memorias cache. Nuevamente se observa un mayor impacto de la cache de instrucciones que la de datos sobre la aceleración del programa, así como que un incremento de la cache de instrucciones por encima de 8 kB no incrementa la aceleración de la ejecución del algoritmo.

Comparando las tablas 2 y 3 se puede apreciar que para capacidades de ambas memorias entre 2 y 4kB, la aceleración con 8 palabras por línea es ligeramente superior a la alcanzada con 4 palabras por línea. Esto se debe a que, al ser mayor el tamaño de la línea, los valores de los puntos de la imagen integral necesarios para calcular el valor de algunos rasgos pueden estar contenidos en una misma línea e incrementar la tasa de aciertos.

Tabla 3. Resultados de aceleración para 16 configuraciones diferentes de cache con 8 palabras por línea

I-Cache / D-Cache	2	4	8	16
2	10,08	10,33	10,42	10,76
4	10,10	10,36	10,45	10,80
8	15,13	15,83	16,09	17,08
16	15,13	15,83	16,09	17,10

Sin embargo, para capacidades de cache entre 8 y 16kB, la aceleración disminuye ligeramente (excepto para 16kB de cache de datos). Una posible interpretación de este resultado puede deberse a que no se compensa el incremento de la tasa de aciertos con la mayor demora en transferir una línea de mayor tamaño entre memoria principal y la cache. En cualquiera de los casos, se observa que el tamaño de la línea de cache tiene poco impacto sobre la aceleración del algoritmo de detección de rostros de Viola-Jones.

CONCLUSIONES

Analizando las funciones del programa desarrollado para implementar el algoritmo de detección de rostros de Viola-Jones se evidencia que la función que calcula el valor de los rasgos de los clasificadores es la de mayor impacto en la demora de ejecución del algoritmo. Esto se debe a la gran cantidad de veces que se ejecuta esta función.

La simplicidad de esta función y su carácter iterativo hacen que se pueda incrementar la velocidad de ejecución del algoritmo mediante su almacenamiento en diferentes líneas de la cache de instrucciones. Esta función ejecuta en cada iteración 36 accesos a memoria. Aunque algunos de los datos pueden estar en direcciones consecutivas y sacar provecho de la localidad espacial de la cache, los datos de la imagen integral de los puntos significativos de un rasgo están en direcciones dispersas y no cumplen con este principio, por lo que este hecho influye negativamente en el incremento de la tasa de aciertos de la cache de datos.

Se ha realizado un conjunto de 33 implementaciones con diferentes estructuras de capacidad y tamaño de línea de ambas memorias cache utilizando una placa de desarrollo Spartan3A Starter Kit basada en un FPGA Spartan3A XC3S700A, con el procesador Microblaze a 62,5 MHz y 64 MB de memoria externa DDR2 a 125 MHz.

Con una primera implementación sin memoria cache se obtiene una demora de procesamiento de una imagen de 26,152 segundos, la cual se utiliza como referencia para las posteriores mediciones. Se realizaron 16 implementaciones con capacidades de ambas cache entre 2 y 16kB y cuatro palabras por línea. Los resultados obtenidos muestran un mayor impacto de la cache de instrucciones que la de datos, con valores óptimos de 8kB para la cache de instrucciones y entre 4 y 16kB para la cache de datos. Con estas memorias se alcanza una aceleración de alrededor de 17 veces con relación a la ejecución del algoritmo en memoria externa.

Se realizaron otras 16 implementaciones con memorias cache entre 2 y 16 kB pero ahora con ocho palabras por línea. Los resultados obtenidos muestran que el tamaño de la línea de cache tiene poca influencia sobre la aceleración de este algoritmo.

REFERENCIAS

1. MIT. 2011. Computer Science and Artificial Intelligence Laboratory. Disponible en: http://www.csail.mit.edu/, http://groups.csail.mit.edu/vision/welcome/.
2. THE UNIVERSITY OF MANCHESTER. 2011. Imaging Sciences Research, Face recognition and visual processing. Disponible en: http://www.medicine.manchester.ac.uk/imaging/.
3. CARNEGIE MELLON UNIVERSITY. 2011. The Robotics Institute, Vision and Autonomous Systems Center (VASC),. Pittsburgh. Disponible en: http://www.ri.cmu.edu/research_center_detail.html?center_id=4&menu_id=262.
4. IBM CORPORATION. IBM Smart Surveillance System. Disponible en: https://researcher.ibm.com/researcher/view_project.php?id=2039.
5. SONY. Sony Face Recognition Technology. Disponible en: http://www.sony.net/SonyInfo/technology/technology/theme/sface_01.html.
6. INTEL. 2011. Intel® Integrated Performance Primitives. Disponible en: http://software.intel.com/en-us/articles/using-intel-ipp-with-opencv/?wapkw=%28opencv%29.
7. SIEMENS. Siemens Medical Informatics Patents Page. Disponible en: http://www.usa.siemens.com/en/about_us/research/research_dev/integrated_data_systems_patents.htm.
8. L1 IDENTITY SOLUTIONS. Disponible en: http://www.l1id.com/pages/116-face.
9. BIOID. Biometric authentication as a Service (BaaS). Disponible en: http://www.bioid.com/products-and-services.html.
10. SHAN, TING , BIGDELI, ABBAS , LOVELL, BRIAN & CHEN, SHAOKANG: "Robust Face Recognition Technique for a Real-Time Embedded Face Recognition System". en: VERMA, B. & BLUMENSTEIN, M. (eds.) Pattern Recognition Technologies and Applications: Recent Advances. Hershey, PA, U.S.A. : 2008.
11. SCHNEIDERMAN, HENRY & KANADE, TAKEO "A statistical method for 3d object detection applied to faces and cars". en: IEEE Conference on Computer Vision and Pattern Recognition, 2000 Hilton Head Island, SC. 746 - 751. 2000.
12. ROWLEY, HENRY, BALUJA, SHUMEET & KANADE, TAKEO "Neural Network-Based Face Detection" en IEEE Transactions on Pattern Analysis and Machine Intelligence, 20, 23 - 38. 1998.
13. RAVI KUMAR, C.N. & BINDU, A.: "An Efficient Skin Illumination Compensation Model for Efficient Face Detection". en: IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference on, 6-10 Nov 2006 Paris. 3444 - 3449. 2006.
14. KRISHNA, S. & PANCHANATHAN, S.: "Combining Skin-Color Detector and Evidence Aggregated Random Field Models towards Validating Face Detection Results". en: Computer Vision, Graphics & Image Processing, 2008. ICVGIP '08. Sixth Indian Conference on, 16-19 Dec. 2008 2008. 466-473. 2008.
15. VIOLA, PAUL & JONES, MICHAEL "Rapid object detection using a boosted cascade of simple features". en: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR, 2001. 511-518. 2001.
16. YANG, MING-HSUAN, KRIEGMAN, DAVID.J. & AHUJA, NARENDRA: "Detecting faces in images: a survey" en IEEE Transactions on Pattern Analysis and Machine Intelligence, 24, 34 - 58. 2002.
17. MATHEW, BINU, DAVIS, AL & EVANS, ROBERT: "A Characterization of Visual Feature Recognition". en: IEEE 6th annual workshop on workload characterization, 2003. 3-11. 2003.
18. BIGDELI, ABBAS, SIM, COLIN, BIGLARI-ABHARI, MORTEZA & LOVELL, BRIAN C.: "Face Detection on Embedded Systems ". en: Proceedings of the 3rd International Conference on Embedded Software and Systems: Lecture Notes in Computer Science, 14 - 16 May, 2007 2007 Daegu, Korea. Berlin, Heidelberg: Springer-Verlag, 295 - 308 2007.
19. ZHANG, CHA & ZHANG, ZHENGYOU: A Survey of Recent Advances in Face Detection. Microsoft Research 2010.

20. CHANGJIAN, GAO & SHIH-LIEN, LU: "Novel FPGA based Haar classifier face detection algorithm acceleration". en: International Conference on Field Programmable Logic and Applications. FPL 2008. , 8-10 Sept. 2008. 373-378. 2008.
21. HUNG-CHIH, LAI, SAVVIDES, M. & TSUHAN, CHEN: "Proposed FPGA Hardware Architecture for High Frame Rate Face Detection Using Feature Cascade Classifiers". en: First IEEE International Conference on Biometrics: Theory, Applications, and Systems. BTAS 2007., 27-29 Sept. 2007 2007. 1-6. 2007.
22. CHO, JUNGUK, BENSON, BRIDGET, MIRZAEI, SHAHNAM & KASTNER, RYAN: "Parallelized Architecture of Multiple Classifiers for Face Detection". en: IEEE International Conference on Application-specific Systems, Architectures and Processors, 2009. 2009.
23. THEOCHARIDES, T., VIJAYKRISHNAN, N. & IRWIN, M. J.: "A parallel architecture for hardware face detection". en: IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, 2-3 March 2006 2006. 2006.
24. HIROMOTO, M., NAKAHARA, K., SUGANO, H., NAKAMURA, Y. & MIYAMOTO, R.: "A Specialized Processor Suitable for AdaBoost-Based Detection with Haar-like Features". en: IEEE Conference on Computer Vision and Pattern Recognition. CVPR '07, 17-22 June 2007 2007. 1-8. 2007.
25. ZHENG, DING, FENG, ZHAO, TINGHUI, WANG, WEI, SHU & MIN-YOU, WU: "Hecto-Scale Frame Rate Face Detection System for SVGA Source on FPGA Board". en: IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 1-3 May 2011 2011. 37-40. 2011.
26. LUO, R. C. & HSIN-HUNG, LIU: "Design and implementation of efficient hardware solution based sub-window architecture of Haar classifiers for real-time detection of face biometrics". en: International Conference on Mechatronics and Automation (ICMA), 4-7 Aug. 2010 2010. 1563-1568. 2010.
27. NAIR, VINOD, LAPRISE, PIERRE-OLIVIER & CLARK, JAMES J.: "An FPGA-Based People Detection System" en European Association for Signal Processing, EURASIP Journal on Applied Signal Processing, 1047–1061. 2005.
28. XILINX INC.: 7 Series FPGAs Overview, Advance Product Specification. DS180 (v1.9). 2012.
29. MATHEW, B., DAVIS, A., EVANS, R., IEEE International Workshop on Workload Characterization, IEEE Conference Publications, 3-11, 2003.
30. NAIR, V., LAPRISE, P., CLARJ, J., An FPGA-Based People Detection System, EURASIP Journal on Applied Signal Processing vol. 7, 1047–1061, 2005
31. JUNGUK, C., BENSON, B., MIRZAEI, S., KASTNER, R., Parallelized Architecture of Multiple Classifiers for Face Detection, 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, IEEE Conference Publications, 75-82, 2009
32. HEFENBROCK, D., OBERG, J., NHAT, T., KASTNER, R., BADEN, S., Accelerating Viola-Jones Face Detection to FPGA-Level Using GPUs, 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE Conference Publications, 11-18, 2010
33. KYRKOU, C., THEOCARIDES, T., A Flexible Parallel Hardware Architecture for AdaBoost-Based Real-Time Object Detection, IEEE Transactions on Very Large Scale Integration Systems, Vol. 19, Issue: 6, pp 1034 - 1047, 2011
34. FREUND, Y., SCHAPIRE, R., A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting, Journal of Computer and System Sciences, pp. 119-139, 1997.
35. GARAGE, W., Open Source Computer Vision library. Disponible en: http://opencv.willowgarage.com/wiki/Welcome
36. ROWE, A., GOODE, A., GOEL, D. , ROSENBERG, C. & NOURBAKHSH, I. 2011. CMUcam3: Open Source Programmable Embedded Color Vision Platform. Disponible en: http://www.cmucam.org/.
37. XILINX INC., ISE Design Suite User Guide, Version 12.4, 2011.
38. XILINX INC., Spartan-3A FPGA Starter Kit Board User Guide, 2007.

AUTORES

Ernesto del Toro Hernández, Ingeniero en Automática, Máster en Sistemas Digitales. Profesor Asistente, Centro de Investigaciones Microelectrónicas, Instituto Superior Politécnico "José Antonio Echeverría", Cuba, ernesto@electrica.cujae.edu.cu. Actualmente desarrolla su tema de investigación doctoral sobre aceleración de algoritmos de detección de rostros mediante hardware reconfigurable.

Alejandro José Cabrera Sarmiento, Ingeniero Electricista, Doctor en Ciencias Técnicas. Profesor Titular, Departamento de Automática y Computación, Instituto Superior Politécnico "José Antonio Echeverría, Cuba, alex@electrica.cujae.edu.cu.

Santiago Sánchez Solano, Licenciado en Física-Electrónica, Doctor en Física por la Universidad de Sevilla. Investigador Principal del CSIC, Director del Instituto de Microelectrónica de Sevilla, Centro Nacional de Microelectrónica, Consejo Superior de Investigaciones Científicas (IMSE-CNM-CSIC), España, santiago@imse-cnm.csic.es.

Alejandro Cabrera Aldaya, Ingeniero en Automática, Máster en Sistemas Digitales. Complejo de Investigaciones Tecnológicas Integradas (CITI), acabrera@udio.cujae.edu.cu.