ARQUITECTURA HÍBRIDA FPGA/DSP PARA EL SEGUIMIENTO DE OBJETOS EN TIEMPO REAL

Jaime Alberto Parra Plaza, Eduardo J. Ramírez, Javier A. Moreno y Carlos A. Bolaños

Departamento de Ciencias e Ingeniería de la Computación
Pontificia Universidad Javeriana Cali, COLOMBIA

{jparra, ejramirez, jamoreno, candres}@puj.edu.co

RESUMEN

Se describe una arquitectura híbrida para hacer el seguimiento de múltiples objetos en tiempo real. El sistema se basa en la interacción armoniosa entre un computador personal, un procesador digital de señales (DSP) y un dispositivo de lógica reconfigurable (FPGA). Para lograr el desempeño en tiempo real se establece un alto paralelismo en donde cada etapa alimenta y es alimentada por las otras etapas. Así, el computador se encarga de realizar todas las tareas de preprocesado para eliminar información irrelevante y/o que hagan uso intensivo de operaciones en punto flotante; el DSP maneja la coordinación entre los diversos elementos e implementa el procesamiento de alto nivel (análisis de imagen) mediante redes neuronales; y la FPGA se encarga del procesamiento de bajo nivel (segmentación de objetos y etiquetado) mediante operadores de punto y de continuidad. Adicionalmente, se dispone de una aplicación de usuario para ingresar comandos y datos y visualizar los resultados en tiempo real.

1. INTRODUCCIÓN

La necesidad de sistemas que procesen vídeo para patrones determinados es muy alta, especialmente en aplicaciones de seguridad o de control/monitoreo de procesos de producción automatizados [1].

A pesar de que la literatura reporta varios de estos sistemas [2][3][4], la mayoría se basa en una aplicación sobre un computador acompañada de una tarjeta procesadora de vídeo. Esta combinación genera un producto de un precio elevado y que requiere una máquina de gran potencia de cómputo.

El sistema presentado aquí transfiere el poder de cómputo a hardware especializado, un sistema híbrido FPGA/DSP, con lo cual se logra un producto que procesa a muy altas velocidades, es de bajo costo y puede ser acoplado a un computador de prestaciones modestas, como se ha establecido ya a través de aplicaciones similares [5]. Las limitaciones tecnológicas actuales impiden desarrollar todo el sistema exclusivamente en FPGA o en DSP. El artículo se centra en explicar la sección implementada en la FPGA. El lector interesado en los aspectos del DSP o de la aplicación puede consultar [6].

2. ESQUEMA GENERAL DEL SISTEMA

La figura 1 presenta mediante un diagrama de bloques el sistema implementado. El flujo de información empieza en el computador, en donde una aplicación realiza un preprocesado inicial sobre el vídeo a analizar. El vídeo puede provenir de una cámara conectada al computador o de un archivo almacenado en el mismo. La aplicación se encarga de convertirlo en una secuencia de imágenes tipo BMP y, opcionalmente, si el usuario lo desea, de realizar algunas labores de mejora como eliminación de bordes, ajuste de contraste o reducción de ruido.

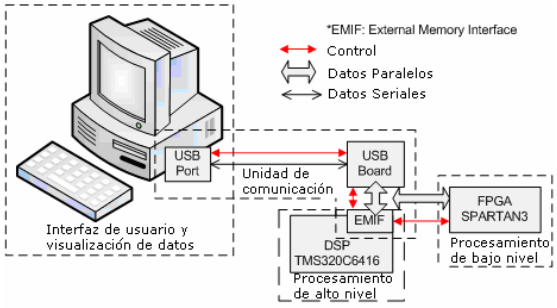

Figura 1: Diagrama general del sistema

La secuencia de imágenes es enviada vía conexión USB [7] hacia el DSP, el cual se encarga de administrar la forma en que se envían las imágenes hacia la FPGA (por ejemplo fraccionándolas si su tamaño es superior al tamaño de la memoria de la FPGA), garantiza la consistencia de la información transmitida y realiza el procesamiento de alto nivel con base en los resultados que entrega la FPGA. Dicho procesamiento implica el análisis de escena, es decir, el seguimiento de los objetos que aparecen en el vídeo sin importar sus velocidades relativas o la oclusión parcial que puedan presentar entre sí en un momento dado.

La arquitectura implementada sobre la FPGA trabaja sobre imágenes individuales, realizando un procesamiento de bajo nivel, el cual consiste en determinar cuáles objetos hay en dicha imagen y de

enviar esta información de manera parametrizada al DSP para que el trabajo de éste se pueda hacer muy rápidamente.

3. SEGMENTACIÓN USANDO SNAKE

La segmentación de una imagen implica la separación y discriminación de los objetos disjuntos que la conforman, valga decir, la determinación de los elementos que un ser humano reconocería como *independientes* al observarla.

Existen muchos métodos para realizar segmentación [8], los cuales se basan en seguir el contorno de las figuras, es decir, hallar superficies cerradas e independientes. Igualmente, existe toda una serie de problemas típicos a los cuales se enfrentan dichos métodos, principalmente dos: el primero es cómo hacer frente a los contornos que tienen discontinuidades, pero que efectivamente corresponden a una superficie cerrada, el segundo es cómo determinar si una superficie que posee múltiples vértices (puntos donde se unen tres o más curvas) es en realidad un solo objeto o son varios objetos que se encuentran parcialmente ocluidos unos por otros.

Para hacer frente a estas dificultades y lograr una segmentación robusta se combinaron diversas técnicas para superar las debilidades de cada una. Se usó una variación del denominado algoritmo *snake* [9], el cual se basa en la idea de contornos activos, es decir, en parametrizar dinámicamente el contorno asociado con cada píxel. Para ello, se asigna una energía funcional a cada posible contorno, y se selecciona el contorno con la mínima energía. La energía funcional utilizada es la suma de varios términos, en donde cada uno corresponde a una fuerza actuando sobre el contorno (ecuación 1).

$$E = \int \left(\alpha(s) E_{cont} + \beta(s) E_{curv} + \gamma E_{img} \right) ds$$

Ecuación 1: Energía funcional.

Se considera un contorno C(s), parametrizado por su longitud de arco s. La energía funcional E deseable está constituida por la suma de tres valores, cuya influencia relativa se controla por los parámetros α, β, γ los cuales pueden variarse a lo largo de C(s). Los términos estiman la continuidad y la suavidad del contorno deformable, respectivamente. El término considera la atracción del borde, dirigiendo el contorno hacia el borde de la imagen más próximo. La figura 2 muestra el resultado de aplicar *snake*.

Figura 2: Contorno obtenido con *Snake*

3.1. Localización de esquinas

La capacidad del algoritmo snake para seguir suavemente los contornos se convierte en debilidad cuando la figura posee esquinas con ángulos agudos en donde la curvatura cambia abruptamente, como se ilustra en la figura 2.

Para hacer frente a esa dificultad, se le ayuda al algoritmo mediante un procesamiento adicional que determina justamente esos puntos de cambio, que corresponden a las esquinas de un objeto, método denominado de *detección de curvaturas* [8].

Un píxel puede ser reconocido como esquina si y sólo si se encuentra sobre una línea de contorno y su curvatura es un máximo local. Para determinar una esquina se necesita conocer la medida de la curvatura global de la línea de contorno de cada píxel, la cual está dada por el valor local de gradiente a través de la línea de contorno, gradiente que se obtiene por algún método rápido de detección de bordes, como Sobel [8]. Así mismo, se hace necesario que el contorno posea una anchura de un píxel, es decir, que esté *esqueletizada* [8].

En la proximidad del contorno, el vector gradiente del píxel es siempre perpendicular a la misma. En contornos suaves los vectores son aproximadamente paralelos, pero cerca de las esquinas la dirección varía (figura 3).

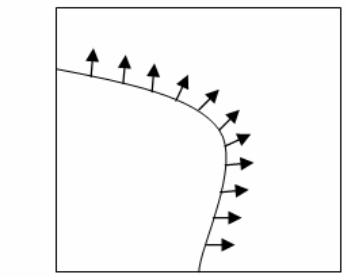

Figura 3: Vectores gradiente en las esquinas

Para cada píxel se calculan dos parámetros, las curvaturas Cm y Cp, dadas por las ecuaciones 2 y 3:

$$Cm = \frac{\sum_{i=1}^{n} (V_{centro} x V_i)}{n}$$

Ecuación 2: Parámetro de curvatura Cm.

$$Cp = (dx_1 - dx_2)^2 + (dy_1 - dy_2)^2$$

Ecuación 3: Parámetro de curvatura Cp.

En donde *dx* y *dy* son los gradientes dados por el operador Sobel, *Vcentro* es el valor para el píxel y *Vi* es el producto cruz de los gradientes i-ésimos.

Después del cálculo de la curvatura local, se tiene un valor para cada píxel de la línea de contorno. Haciendo un barrido por sus vecindades interior (pixeles en contacto directo con él) y exterior (pixeles separados una unidad de él) se sigue que un punto es una esquina si se cumplen las siguientes restricciones: Cp es el máximo de la vecindad interior y es mayor que el máximo de la vecindad exterior y Cm es mayor que el máximo valor de los puntos que no cumplen la condición sobre Cp. La figura 4 indica el resultado de aplicar este proceso.

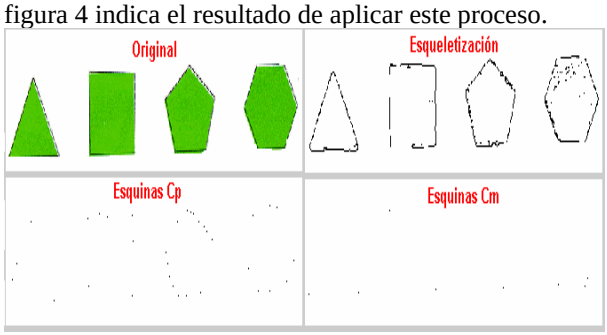

Figura 4: Obtención de esquinas mediante curvaturas.

4. ETIQUETADO: MOMENTOS INVARIANTES

Una vez determinados los objetos que constituyen la imagen se hace necesario *identificar* cada objeto de manera única, proceso denominado *etiquetado*. El etiquetado es importante en procesamiento de video porque permite seguir un objeto en el tiempo, sin importar si morfológicamente es igual a otro.

Muchos de los métodos usuales para realizar etiquetado se basan ya sea en el crecimiento de regiones (partir de un punto al interior de la figura e ir avanzado en el etiquetado de cada píxel hasta llegar al contorno) o en el seguimiento del contorno (tomar un punto sobre el contorno y etiquetar cada punto del mismo hasta llegar al píxel de origen). Todos estos métodos tienen la desventaja de que dependen de la selección apropiada de algún punto de inicio. Para eliminar esta condición, se optó por utilizar un método más robusto llamado *momentos invariantes* [10].

La geometría de una región plana se basa en su tamaño, posición, orientación y forma. Todas estas medidas están relacionadas con una familia de parámetros llamados momentos. Estos momentos se pueden normalizar con respecto a su área para así obtener momentos normalizados independientes del tamaño, posición y ángulo del objeto. A manera de ejemplo, la ecuación 4 muestra la forma de calcular los momentos no normalizados de orden p+q:

$$M_{p,q} = \sum_{x=1}^{N} \sum_{y=1}^{M} x^p y^p I(x,y)$$

Ecuación 4: Momento p+q para una imagen NxM.

5. ARQUITECTURA SINTETIZADA EN LA FPGA

La figura 5 esquematiza parte de la arquitectura de procesamiento contenida en la FPGA, la cual se implementó en una FPGA Xilinx Spartan3 XC3S5000, un dispositivo que posee 5 Mcompuertas, más de 2Mbits de RAM de usuario y está optimizado para el trabajo con FIFOs. El procesador posee dos FIFOS que almacenan imágenes de NxM pìxeles, en donde N y M pueden configurarse para valores entre 32 y 256. El código VHDL está parametrizado de tal forma que, dependiendo del valor de la imagen que se escoja, el sistema decide el número de multiplicadores y de procesadores compartidos, de tal manera que la arquitectura ocupa la totalidad del FPGA objetivo una vez sintetizado el diseño.

En las FIFOS se almacenan las imágenes presente y anterior respectivamente. Para evitar procesamiento innecesario, primero se hace una resta de las imágenes acompañada de una erosión [11], si el resultado posee un nivel de energía bajo, indica que esa sección de la imagen no posee elementos o que éstos no se han movido, así que no se hace ningún procesamiento adicional.

Las FIFOS pueden ser accedidas simultáneamente por los diversos bloques que se encuentran en la unidad de procesamiento y que realizan de manera concurrente los procesos de detección de bordes (Sobel), adelgazamiento de bordes (esqueletización), segmentación (contorno activo) y etiquetado (momentos invariantes).

La comunicación entre estos bloques se da por banderas que indican cuándo el trabajo sobre una sección o imagen ha concluido. Igualmente se da una comunicación interactiva con el DSP para optimizar el proceso de seguimiento y para alimentar con nuevas imágenes o secciones de la misma en caso de que la imagen sea de mayor tamaño al tamaño actual de la FIFO [12].

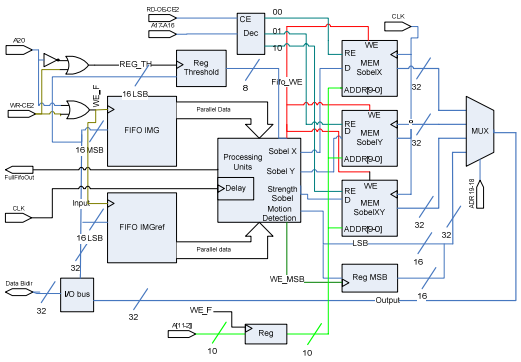

Figura 5: Arquitectura en la FPGA.

Para explotar al máximo el paralelismo [13], se tiene una jerarquía de miniprocesadores, cada uno asociado con un píxel y que se comunican con sus vecinos inmediatos. Cada vecindad reporta a otro miniprocesador lo cual incrementa la velocidad de segmentación y etiquetado.

Existen bloques para procesar todas y cada una de las características descritas en los numerales 3 y 4. Entrar en detalles del hardware específico es innecesario y en algunos casos imposible, puesto que el diesño del código VHDL se hizo para lograr una equivalencia con los códigos elaborados en C y ASM, respetando claro está el paradigma del hardware reconfigurable y optimizando el sistema para su síntesis en el dispositivo escogido.

6. PRUEBAS Y RESULTADOS

Este proyecto hace parte de un macroproyecto general que busca implementar un laboratorio de sistemas reprogramables y reconfigurables [14], por tanto, en este caso se tiene que todos los algoritmos necesarios para el seguimiento de objetos se han implementado en cuatro plataformas: como aplicación desarrollada en Matlab [15], como aplicación desarrollada en VisualC++, código C y *assembly* para DSP y código VHDL para FPGA.

El sistema puede correr en una sola de estas plataformas o de manera híbrida en una combinación de ellas [16], la estructura descrita aquí es una de las opciones híbridas que permite el procesado en tiempo real. Dada la versatilidad del sistema, no es inmediata una comparación con otros sistemas. Igualmente, no hay referencia a sistemas que procesen vídeo en tiempo real a través de una combinación como la mostrada aquí. Los sistemas que procesan vídeo lo hacen ya sea sobre DSP [4] o sobre otras arquitecturas diferentes a FPGA [2][3].

La figura 6 muestra un pantallaza de la aplicación, en donde cada objeto está etiquetado con un cuadrado que le rodea y que se desplaza con él. La figura 7 muestra la robustez del sistema frente al problema de oclusión: si dos objetos se traslapan momentáneamente, el sistema es capaz de seguir reconociéndolos correctamente después de que se separan. La figura 8 ofrece una comparación entre el tiempo de procesamiento requerido por una aplicación en VisualC++ y por el sistema DSP/FPGA para distintos tamaños de imágenes. Se resalta no sólo que el tiempo usado por el sistema es mucho menor, sino que su dependencia con el tamaño de la imagen es lineal y no exponencial y que es lo suficientemente pequeño como para permitir un seguimiento en tiempo real.

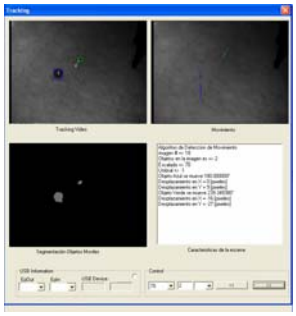

Figura 6: Ejemplo de salida en la Aplicación.

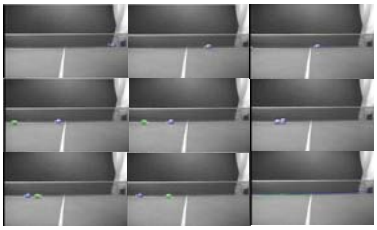

Figura 7: Ejemplo mostrando la solución a la oclusión.

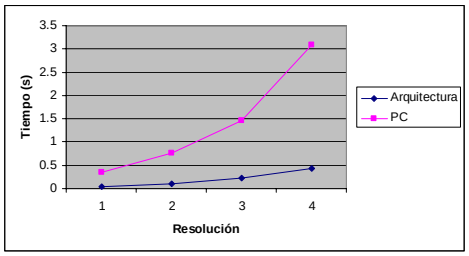

Figura 8: Tiempo para procesar una imagen (1=320x240; 2=640x480; 3=1280x960; 4=2590x1920).

7. CONCLUSIONES

La implementación híbrida FPGA/DSP presentada tiene ventajas costo/beneficio importantes con respecto a una implementación PC / tarjeta de adquisición de imágenes.

Considerando la tecnología actual, no es viable aún implementar todo el sistema en un FPGA o en un DSP sin elevar el costo. La alternativa viable es la híbrida.

Los FPGAs poseen una arquitectura que se presta para la implementación de procesadores de imágenes, dada su disposición matricial. Pero para explotar apropiadamente esta arquitectura, es importante que el sistema diseñado se base también en una matriz de miniprocesadores.

Dada la debilidad actual de los FPGAs para procesar números en punto flotante, una mejor opción para implementar un analizador de escenas es un DSP.

El puerto USB mostró ser válido para la transferencia de grandes cantidades de datos y permite al sistema compatibilidad con dispositivos actuales.

8. REFERENCIAS

[1] A.D. Kulkarni. *Computer vision and fuzzy-neural systems.* Estados Unidos: Prentice Hall, 2001.

[2] G. Linan, y S. Espejo, "ACE4k: An analog I/O 64x64 Visual Microprocessor Chip With 7-bit Analog Accuracy," *Intl Journal Of Circuit Theory and Apps*, 30:89–116, 2002.

[3] D.L. Vilarino y D. Cabello, "Cellular Neural Networks and Active Contours: A Tool for Image Segmentation," *Image and Vision Computing*, 21(2):189–204, 2003.

[4] C. Dumontier, F. Luthon, J. Charras. "Real-time DSP implementation for MRF-based video motion detection". *IEEE Transactions on Image Processing.* Vol 8 No 10. 1999.

[5] J.A. Parra y F. Benítez, "Lab de Visión Artificial Usando DSP en FPGA", *Tesis de Grado*, U. Javeriana Cali, 2004.

[6] J.A. Parra, E.J. Ramírez, J.A. Moreno y C.A. Bolaños, “Arquitectura Paralela FPGA/DSP para el Procesamiento de Video,” *XI Simposio* STSIVA, Bogotá, Colombia, 2006.
[7] TI. “TMS320C6000_ EMIF to USB Interfacing Using Cypress EZ-USB SX2”. *App. Report* SPRAA13A - 2005.
[8] G. Pajares, J.M. De la Cruz. *Visión por computador: Imágenes Digitales y Aplicaciones*. España: Rama, 2001.
[9] M. Kass y A. Witkin. “Snakes: Active Contours Models,” *Intl J. on Computer Vision*, 1:321–331, 1988.
[10] Hu, M.K., “Visual Pattern Recognition by Moment Invariants,” *IT*, Vol 8, No. 2, February, pp. 179-187, 1962.
[11] P. Maragos. “Morphological Signal and Image Processing”, *DSP Handbook*, CRC & IEEE Press), Boca Raton, 1998, pp.~74-1:74-30.
[12] C.S. Tong, P.C. Yuen y Y.Y. Wong. “Dividing snake algorithm for multiple object segmentation,” *Optical engineering*. Vol. 41, No12, pp. 3177-3182, 2002.
[13] T. Braunl. *Parallel Image Processing*. Alemania: Springer-Verlag, 2001.
[14] J.A. Parra y F. Amaya, “Didactic Tool for Teaching Comm Sys and DSP” en *Comp. Methods in Circuits and Systems Apps*, pp. 296-304. WSEAS Press, 2004.
[15] B. Bemis, B. Tannenboun. “Video Processing System Design with Matlab and Simulink,” www.mathworks.com/webinars. 10 agosto 2005.
[16] J.A. Parra y F. Amaya, “Brain-Learning-Model-Based DSP Teaching Environment for Communication Systems” *IEEE ICASSP*, Montreal, Canada, 2004.