Procesamiento de imágenes en FPGA con visualización en una pantalla VGA

Felipe Santiago Espinosa

Instituto de Electrónica y Mecatrónica

Universidad Tecnológica de la Mixteca, Carretera a Acatlima km 2.5, Huajuapan de León, Oaxaca, México

fsantiago@mixteco.utm.mx

Felipe Trujillo-Romero

División de Estudios de Posgrado

Universidad Tecnológica de la Mixteca, Carretera a Acatlima km 2.5, Huajuapan de León, Oaxaca, México

ftrujillo@mixteco.utm.mx

Resumen

En este trabajo se presenta la implementación de algoritmos básicos de procesamiento digital de imágenes sobre una plataforma FPGA. Los algoritmos que se implementaron fueron: 1) el negativo, y la obtención de gradientes en 2) la dirección *x*, 3) la dirección *y* y 4) la dirección *xy*. Para ello se diseñó un sistema modular con el lenguaje VHDL, usando la herramienta Active-HDL de Aldec y sintetizando mediante el entorno ISE de Xilinx. La implementación del sistema se hizo en la tarjeta Nexys-2, la cual tiene un FPGA Spartan 3E-500 de Xilinx. La imagen a procesar se recibe por el puerto serie de la tarjeta a una velocidad de 115200 baudios y tiene una resolución de 640 x 480 pixeles con 8 bits para el color. En la memoria de la tarjeta se almacena la imagen original y las imágenes resultantes del procesamiento, mediante tres interruptores se elige la imagen a mostrar en una pantalla VGA. Con el desarrollo del presente sistema se va organizando un repositorio de módulos funcionales que pueden reutilizarse para la implementación de algoritmos más complejos.

Palabras Claves: Algoritmos embebidos, FPGA, Nexys 2, Procesamiento digital de imágenes.

1. Introducción

El procesamiento digital de imágenes es un área de interés para investigadores y académicos debido al inmenso campo de acción que posee, éste va desde la mejora de una imagen hasta la realización de procesamiento de alto nivel, como lo es el reconocimiento de escenas. Cabe mencionar que la mayor parte de este tipo de procesamiento se realiza en software en una computadora. Dos ejemplos de ello son: 1) el ToolBox de Procesamiento Digital de Imágenes (PDI) desarrollado por Mathworks e incluida en Matlab [1] y 2) la librería OpenCV [2]. Ambos de gran uso y difusión en la comunidad académica, como se puede ver por la gran cantidad de libros que se pueden encontrar de ambos. Por ejemplo, los trabajos de González y Wood [3, 4], quienes tienen un par de libros en los cuales se aborda el PDI, en uno se analizan los algoritmos del procesamiento de imágenes y en el otro se implementan en Matlab los algoritmos estudiados. También se pueden comentar un par de referencias relacionadas con OpenCV, como el libro de Parker [5] en el cual expone algoritmos de procesamiento de imágenes usando OpenCV para su implementación. Otro trabajo con OpenCV es el realizado por Bradski y Kaehler [6] quienes en su libro “Learning OpenCV” presentan una introducción a la visión por computador mediante la librería OpenCV.

Sin embargo también la comunidad ha utilizado la implementación de estos algoritmos para realizar cosas más complejas tales como el reconocimiento de objetos [7, 8, 9], rostros [10] o situaciones de la vida real [11]. Siendo estos algunos ejemplos de lo que se puede encontrar en la literatura científica del PDI y de Visión por computadora.

Si bien es cierto que la cantidad de artículos es amplia, la mayoría usa el procesamiento vía software, empleando una computadora como soporte. Esto porque en cierta medida es más sencillo implementar este tipo de técnicas en software, en donde el desarrollador sólo se concentra en la implementación propiamente dicha, dejando a un lado todo lo referente al manejo interno de datos. Como el uso de la memoria para el almacenamiento temporal de las imágenes durante su procesamiento.

No obstante, hay trabajos en donde la implementación se realiza en hardware, específicamente se utiliza un FPGA para el procesamiento de imágenes. En [12], Cho et al, implementaron un sistema de reconocimiento de rostros en un FPGA utilizando la transformada de Haar; siendo esta transformada un algoritmo clásico para realizar el reconocimiento de rostros. Bouris et al, [13] desarrollaron un sistema de detección de características para el reconocimiento de objetos usando el descriptor SURF (Speeded Up

Robust Features) sobre un FPGA. Finalmente, está el trabajo de Meng et al, [14] quienes implementaron un sistema de reconocimiento de objetos mediante la síntesis de una red bayesiana un FPGA. En estos trabajos de PDI, usualmente las imágenes son enviadas desde la computadora al FPGA para su procesamiento y posteriormente regresadas para exhibir el resultado en el monitor de la computadora.

En este trabajo se plantea la implementación de algoritmos de PDI básicos que pueden servir como componentes para realizar procesamiento más complejo. Además de mostrar el resultado de tales procesamientos en una pantalla VGA que está controlada por el FPGA como se verá en las secciones siguientes. Las cuales están estructuradas como sigue; en la sección 2 se describen las especificaciones del sistema, la sección 3 presenta su organización, describiendo los cinco bloques en que se divide para su diseño y los módulos que integran cada bloque. En la sección 4 están los resultados alcanzados y finalmente, en la sección 5 se presentan las conclusiones a las que se llegó después de la puesta en marcha del sistema.

2. Especificación del Sistema

Se requiere de un sistema modular enfocado a un FPGA para el procesamiento de imágenes; aunque en este trabajo sólo se implementan 4 algoritmos, la idea principal es organizar una plataforma que permita ser adecuada para la evaluación de diferentes algoritmos de PDI. El sistema está enfocado a la tarjeta de desarrollo Nexys-2, la cual es manufacturada por la empresa Digilent [15], por lo que fue necesario considerar sus características para establecer las especificaciones del sistema, las cuales son:

- La imagen se recibe desde el puerto serie de la tarjeta, la comunicación es a 115, 200 bit por segundo, para reducir el tiempo de transferencia.
- El tamaño de la imagen es de 640 x 480 pixeles y se utiliza un byte por cada pixel, se emplea la organización del puerto VGA de la tarjeta, dedicando tres bits para el color rojo, tres para el verde y dos para el azul. Por lo tanto, una imagen requiere de 307 200 bytes para su almacenamiento.
- Después de que el sistema reciba una imagen, debe mostrarla en una pantalla VGA, la imagen ya tiene el formato RGB siguiente: [BBGGGRRR].

- Con los botones de la tarjeta se marca el inicio del procesamiento, se emplea un botón para cada algoritmo.
- Es necesario emplear la memoria SDRAM de 16 MBytes incluida en la tarjeta, dado que se deben almacenar los cuatro resultados del procesamiento, para que se pueda conmutar entre los ellos en forma inmediata, mostrándolos en la salida VGA.
- Con los interruptores se selecciona la imagen a mostrar en la pantalla. Se emplean 3 porque el usuario puede decidir ver a la imagen original o a alguno de los cuatro resultados del procesamiento.
- Con tres LED's de la tarjeta se indica el estado actual del sistema, básicamente puede estar en uno de tres estados: Recibiendo una imagen, ejecutando un algoritmo de procesamiento o mostrando la imagen seleccionada, se emplea un LED para cada estado. Se destina un cuarto LED para indicar si hubo un error durante la recepción serial de la imagen.

En la Fig. 1 se muestra a la tarjeta Nexys 2 y se indica el uso de periféricos para el presente sistema.

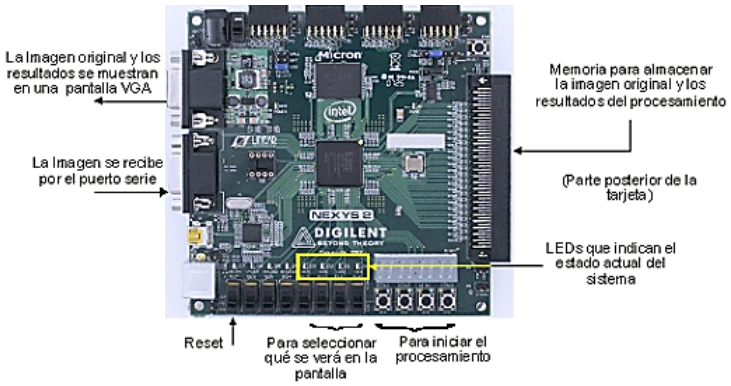

Fig. 1. Recursos empleados en la tarjeta Nexys 2 para el desarrollo del sistema.

3. Organización del Sistema

El sistema se desarrolló empleando la metodología Top-Down, que consiste en la división de un sistema complejo en módulos de menor jerarquía, los cuales a su vez se conformarán por otros módulos, hasta llegar a niveles tan simples, cuya implementación no requerirá de mucho esfuerzo [6].

El nivel de mayor jerarquía corresponde al FPGA, en la Fig. 2 se muestran sus entradas y salidas, en donde se distinguen las señales de los tres recursos empleados: El puerto serial para recepción, el puerto VGA para la proyección y la memoria para el almacenamiento.

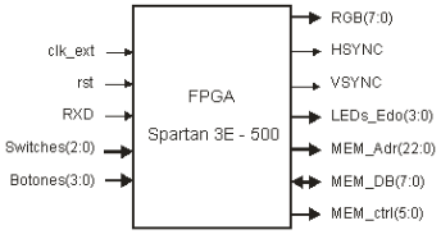

Fig. 2. Nivel de mayor jerarquía: Entradas/Salidas del sistema.

La organización para el siguiente nivel jerárquico comprende cinco bloques: Receptor Serial, Manejo de la pantalla, Procesamiento Digital de Imágenes, Manejo de la Memoria y Control. En la Fig. 3 se muestra la relación entre los diferentes bloques, debe notarse que la memoria es escrita cuando se recibe una imagen, es leída y escrita durante para el procesamiento y es leída durante la proyección; por lo tanto, por medio del control se determina quien tiene acceso a la memoria y con el apoyo de multiplexores se define el flujo en el camino de datos.

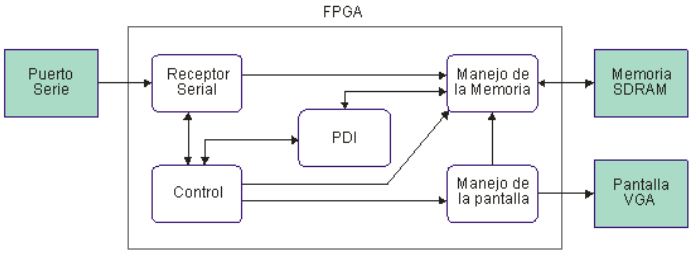

Fig. 3. Nivel de mayor jerarquía: Entradas/Salidas del sistema.

3.1. Receptor Serial

Para la recepción de la imagen se emplea el formato estándar RS-232, transmitiendo a una velocidad de 115 200 baudios, con un bit de inicio, datos de 8 bits y un bit de paro. La base para la recepción es una máquina de estados finitos (FSM, *Finite State Machine*) que espera al bit de inicio, garantiza su validez, para luego muestrear cada bit y desplazar para integrar el dato, finalmente espera un bit de paro válido, si éste no tiene el valor de 1, la FSM indica que ocurrió un error.

La tarjeta Nexys 2 opera a 50 MHz (la duración del ciclo de reloj es de 20 nS), dado que la recepción serial se hace a 115 200 bits/seg, cada bit recibido tarda 8.68 µS, que corresponden a 437.02 ciclos de reloj por bit, de manera que es conveniente el empleo de un divisor de frecuencia para que la FSM no espere tantos ciclos de reloj, dividiendo por un factor de 16, cada bit recibido va a requerir de 27 ciclos de reloj. En la Fig. 4 se muestran los módulos que integran al receptor serial, la frecuencia de 3.125 MHz también se emplea como base para el resto del sistema. El divisor de frecuencia también genera una frecuencia de 25 MHz que servirá para el manejo de la pantalla.

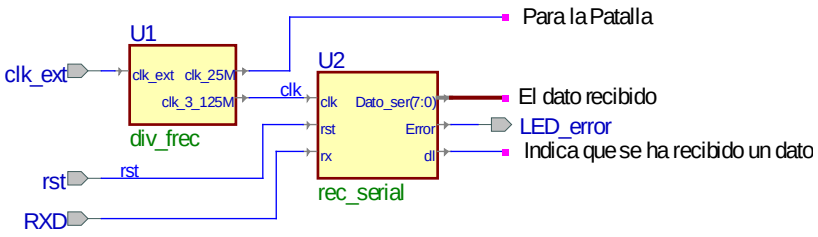

Fig. 4. Módulos que integran al bloque Receptor Serial.

La señal *dl* (dato listo) le indica al control la presencia de un dato por el puerto serie, el control debe encausar los datos y coordinar su escritura en la memoria.

3.2. Manejo de la pantalla

El bloque para el manejo de la pantalla VGA comprende sólo un módulo, éste genera las señales de sincronía horizontal y vertical para la pantalla. El módulo opera a 25 MHz y la pantalla se maneja con una resolución de 640 x 480 pixeles, en la Fig. 5 se muestra el diagrama de tiempos para el recorrido de una línea, el barrido se hace de izquierda a derecha y las líneas se recorren a una frecuencia de 31.5 kHz (31.7 mS por línea).

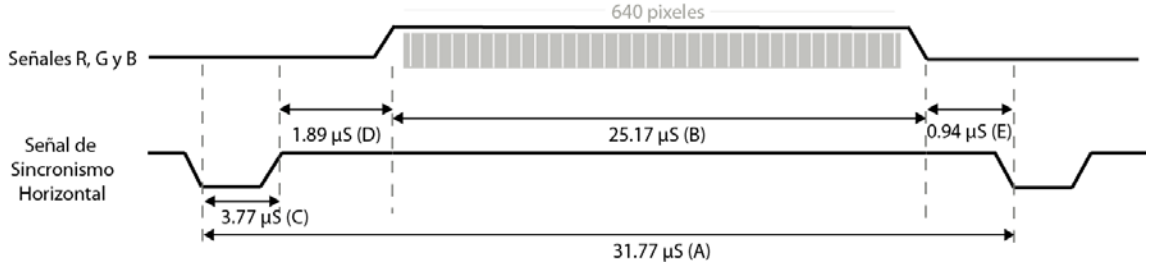

Fig. 5. Señales para la sincronización Horizontal de la pantalla.

Dado que son 480 líneas, su recorrido (de arriba abajo) requiere de un tiempo de 15.4 mS, aunado a la zona de blanqueo y los pulsos de sincronía vertical, la pantalla se refresca cada 16.6 mS, es decir, se tiene una frecuencia de 60 pantallas por segundo. En la Fig. 6 se muestra el diagrama de tiempos para el recorrido de una pantalla.

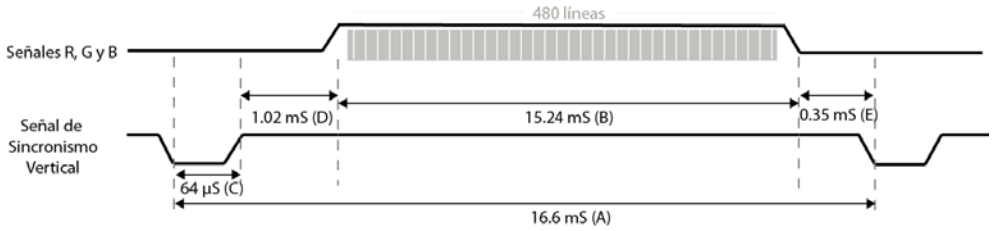

Fig. 6. Señales para la sincronización Vertical de la pantalla.

Con los tiempos mostrados en las Fig. 5 y 6, y considerando la frecuencia de operación del módulo (25 MHz), se calcula la cantidad de pulsos de reloj a la que corresponde cada uno de estos intervalos, en la tabla 1 se muestra este cálculo, los símbolos A, B, C y D pueden verse en ambos diagramas de tiempo (Fig. 5 y 6).

Tabla 1.Tiempos de operación y ciclos de reloj de las señales de sincronismo.

	Sincronismo Horizontal		Sincronismo Vertical		
Símbolo	**Tiempo (µS)**	**Ciclos Reloj**	**Tiempo (mS)**	**Ciclos Reloj**	**Líneas**
A	32	800	16.7	416,800	521
B	25.6	640	15.36	384,000	480
C	3.84	96	64	1,600	2
D	0.64	16	320	8,000	10
E	1.92	48	928	23,200	29

Por lo tanto, el módulo para el manejo de la pantalla internamente contiene un par de contadores (horizontal y vertical), estos contadores se comparan con los valores de la tabla 1 para generar los pulsos de sincronía y la posición del pixel en la pantalla. Además, puesto que la información a mostrar en pantalla se obtiene de la memoria, dentro del mismo módulo VGA se calcula la dirección de acceso, simplificando el diseño. En la Fig. 7 se puede ver al módulo dedicado al manejo de la pantalla, con la entrada *show* se determina si al puerto RGB se mandará el valor del pixel, recibido en el bus *PIXEL_RGB*, o si se mandará el valor 0xFF, que corresponde al valor blanco. Esto porque el sistema puede estar recibiendo una

imagen o haciendo procesamiento, y con ello, aunque el módulo VGA genere una dirección no tendrá acceso a la memoria, dado que ésta estará siendo usada por otros bloques del sistema.

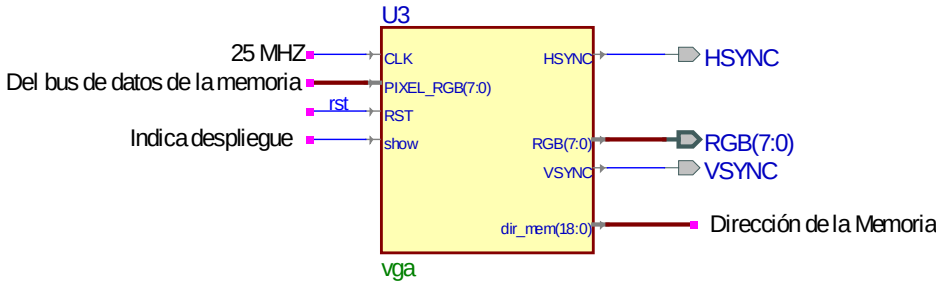

Fig. 7. Módulo que integra al bloque Manejo de la Pantalla.

3.3. Procesamiento Digital de Imágenes

En el sistema se aplican cuatro algoritmos de procesamiento, en la tabla 2 se describe cómo se genera cada pixel en la imagen de salida (S), a partir de la imagen de entrada (E). En los gradientes se debe realizar una comparación entre los pixeles antes de realizar la diferencia, esto equivalente a recorrer la imagen en ambas direcciones. Cada algoritmo deja la imagen resultante en un espacio de memoria diferente, con esto se simplifica el desarrollo del algoritmo 4, puesto que el gradiente en ambas direcciones equivale a la OR lógica entre los gradientes de las direcciones individuales.

Tabla 2. Algoritmos de procesamiento implementados.

Algoritmo	Descripción
Negativo	$S_1[i][j] = NOT (E[i][j])$
Gradiente en dirección *x*	Aplica la máscara [-1 0 1] o [1 0 -1] $S_2[i][j] = Mayor \{ E[i][j + 1] - E[i][j - 1],\ E[i][j - 1] - E[i][j + 1] \}$
Gradiente en dirección *y*	Aplica la máscara $[-1\ \ 0\ \ 1]^T$ o $[1\ \ 0\ \ -1]^T$ $S_3[i][j] = Mayor \{ E[i + 1][j] - E[i - 1][j],\ E[i - 1][j] - E[i + 1][j] \}$
Gradiente en dirección *xy*	Aplica la máscara horizontal y vertical $S_4[i][j] = S_2[i][j]$ OR $S_3[i][j]$

Se desarrolló un módulo para cada algoritmo, la estructura general es la misma, un doble ciclo repetitivo en el que se va construyendo la imagen S a partir de la imagen E, en alto nivel se tendría una secuencia de código similar a:

```
for( i = 0; i < 480; i++)
        for( j = 0; j < 640; j++)
                S[i][j] = E[i][j – 1] - E[i][j + 1];
```

Sin embargo, los algoritmos se implementarán en hardware y deben tomarse en cuenta dos aspectos:

1. Los ciclos *for* no pueden usarse porque generan bloques anidados de hardware con los que fácilmente se saturan los recursos en el FPGA, en su defecto, debe aprovecharse la naturaleza repetitiva de los procesos en VHDL.
2. Una sentencia como *S[i][j] = E[i][j – 1] - E[i][j + 1]* implica dos lecturas en direcciones diferentes de la memoria, una operación aritmética y una escritura, los accesos a memoria requieren dos ciclos de reloj, dado que además de la dirección se deben acondicionar las señales de control, para una sentencia como la anterior se requerirán de 7 ciclos de reloj y se deberán sincronizar los accesos a la memoria.

Es por ello que en el desarrollo de cada algoritmo se utilizó una FSM, con contadores internos se controla el recorrido de la imagen y en cada estado se define la operación a realizar. En la Fig. 8 se muestra la FSM mediante la que se obtiene el negativo de la imagen, con la señal *ini* se marca el inicio del procesamiento y con la señal *fin* la FSM señaliza la culminación del algoritmo.

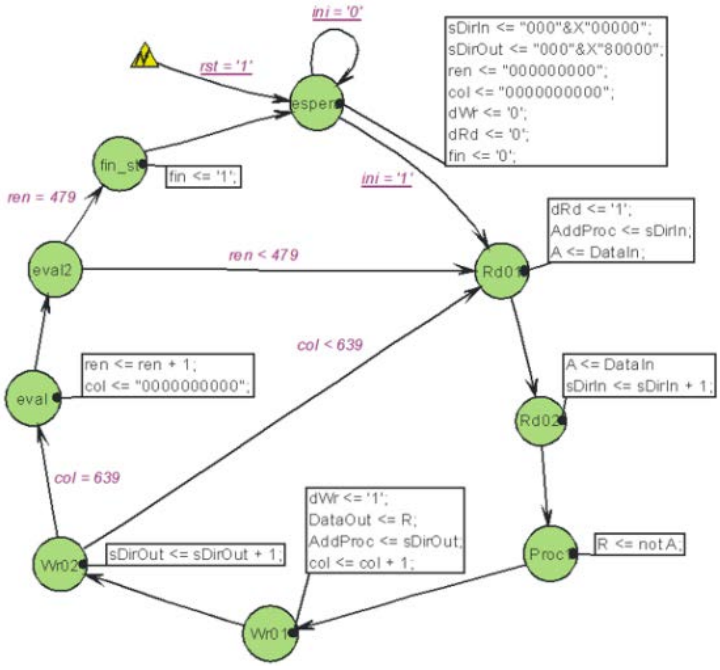

Fig. 8. Procesamiento para la obtención del negativo de una imagen.

En la Fig. 9 se muestra la interfaz del módulo con el que se obtiene el negativo de la imagen, los otros módulos de procesamiento tienen las mismas señales externas aunque internamente sus estados difieren y los resultados son ubicados en localidades diferentes. Para los otros algoritmos se requiere de dos lecturas antes del procesamiento y después se realiza la escritura. En el algoritmo 2 se debe omitir el primer y último renglón, mientras que el algoritmo 3 se debe omitir la primera y última columna, en ambos casos, estás posiciones en la imagen de salida son llenadas con ceros.

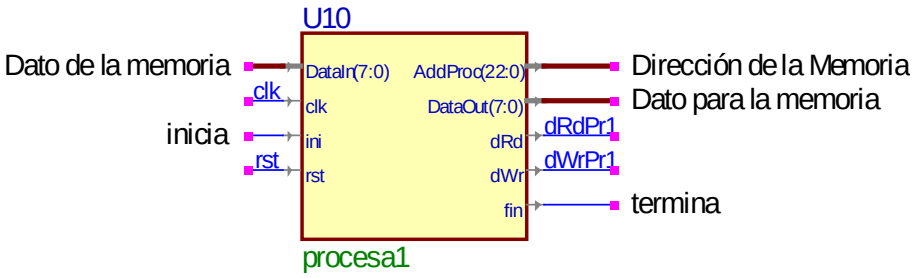

Fig. 9. Módulo para la obtención del negativo de una imagen.

3.4. Manejo de la Memoria

La memoria disponible en la tarjeta Nexys 2 es una M45W8MW16 Cellular RAM pseudo-estática DRAM de Micron, su capacidad es de 8M x 16bits y puede ser operada en diferentes modos [17]. En este sistema se utilizó el modo asíncrono por ser el más simple, la memoria funciona como estática y únicamente debe evitarse la colisión en el flujo de información, puesto que el bus de datos es bidireccional. Para ello, en este bloque se incluye un módulo que mantiene a la memoria en uno de tres estados excluyentes: espera, lectura o escritura. También es necesario un módulo simple que ponga la salida en 3er estado cuando se hace una lectura de la memoria.

El acceso a la memoria debe organizarse, debido a que son seis módulos los que tienen acceso: El receptor serial escribe la imagen recibida, el módulo VGA lee una imagen para su proyección y los cuatro módulos de procesamiento leen la imagen entrante para después escribir el resultado generado. Por lo tanto, mediante compuertas OR y el uso de multiplexores se coordina el uso de las señales de control, los buses de datos y direcciones en la memoria. En la Fig. 9 se muestran todos los módulos que integran al bloque para el manejo de la memoria.

La memoria SDRAM comparte los buses de datos, de direcciones y algunas señales de control con la memoria FLASH, es por ello que en la Fig. 10 se puede ver la inhabilitación constante de la FLASH. Las otras señales constantes son para utilizar únicamente los 8 bits menos significativos de la SDRAM, operándola como una memoria de 8 M x 1 byte.

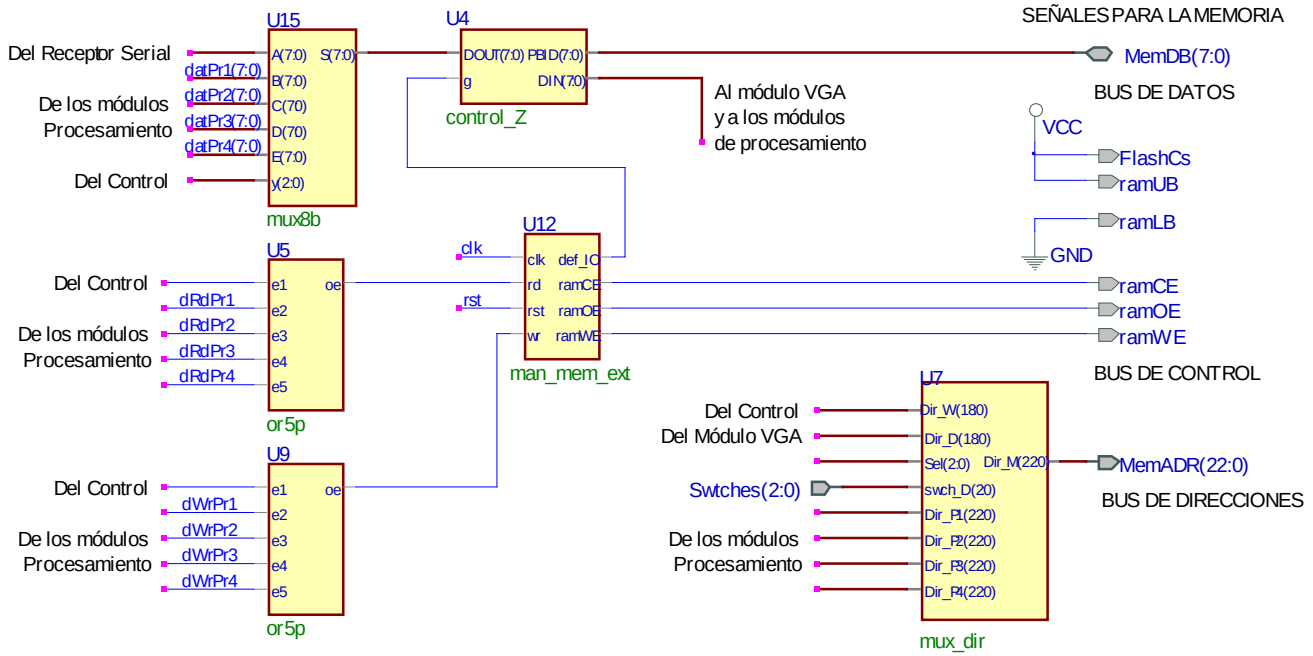

Fig. 10. Módulos para el Manejo de la Memoria.

En la tabla 3 se muestra el mapa de la memoria, cada imagen emplea 307200 bytes (640x480). Para simplificar la organización y evitar el uso de decodificadores complejos se dispone de 19 bits de direcciones, quedando un espacio libre entre cada imagen.

Tabla 3. Mapa de la Memoria Externa.

Intervalo de direcciones	Uso de la Memoria
0x000000 - 0x04AFFF	Imagen Original, recibida de manera serial
0x04B000 - 0x07FFFF	Memoria Libre
0x080000 - 0x0CAFFF	Resultado del primer algoritmo de procesamiento
0x0CB000 - 0x07FFFF	Memoria Libre
0x100000 - 0x14AFFF	Resultado del segundo algoritmo de procesamiento
0x14B000 - 0x17FFFF	Memoria Libre
0x180000 - 0x1CAFFF	Resultado del tercer algoritmo de procesamiento
0x1CB000 - 0x1FFFFF	Memoria Libre
0x200000 - 0x24AFFF	Resultado del cuarto algoritmo de procesamiento
0x24B000 - 0x3FFFFF	Memoria Libre

3.5. El Control del Sistema

El sistema puede estar en uno de tres estados: mostrando una imagen de la memoria, recibiendo una imagen vía serial o ejecutando un algoritmo de procesamiento. Por ello, para el control del sistema se emplea la FSM que se muestra en la Fig. 11, en donde los estados en color azul son estados jerárquicos, la herramienta Active-HDL facilita el uso de este tipo de estados para dar claridad a los diseños, un estado jerárquico internamente incluye dos o más estados simples. En el estado *espera* se generan las señales de control mediante las que se hacen lecturas en la memoria, la dirección proviene del módulo VGA y el dato es encauzado para ser considerado como un pixel en la pantalla. El sistema se mantiene en este estado mientras no se reciba un dato o se presione alguno de los botones.

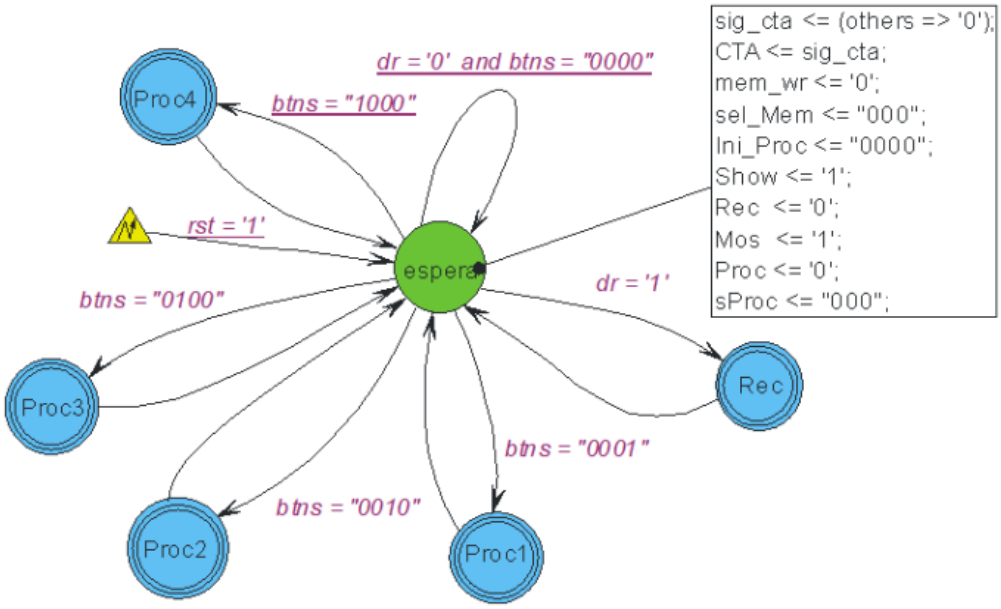

Fig. 11. Control del Sistema.

Con la llegada del primer dato el sistema pasa al estado *Rec*, este estado es expandido en la Fig. 12, el sistema recibe uno a uno los pixeles que conforman la imagen, si una imagen no tiene el tamaño esperado, el sistema se mantendrá en este estado.

Los estados jerárquicos dedicados al procesamiento básicamente tienen dos estados simples, en el primero se marca el inicio de la ejecución del algoritmo y en el segundo espera la conclusión del mismo. En la Fig. 13 se muestra al módulo del control, se agrega una OR de 4 entradas para sólo esperar por una señal como el indicador de fin de procesamiento.

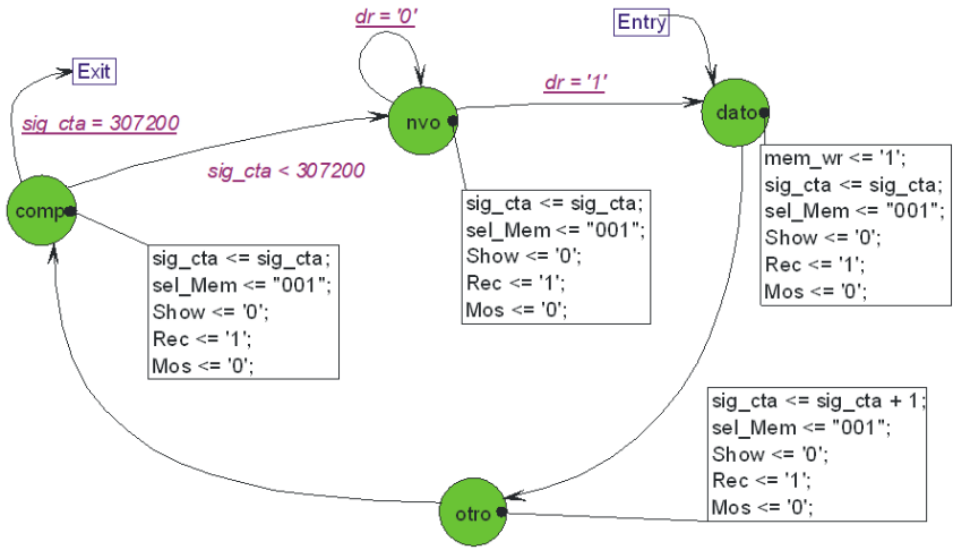

Fig. 12. Recepción de la Imagen.

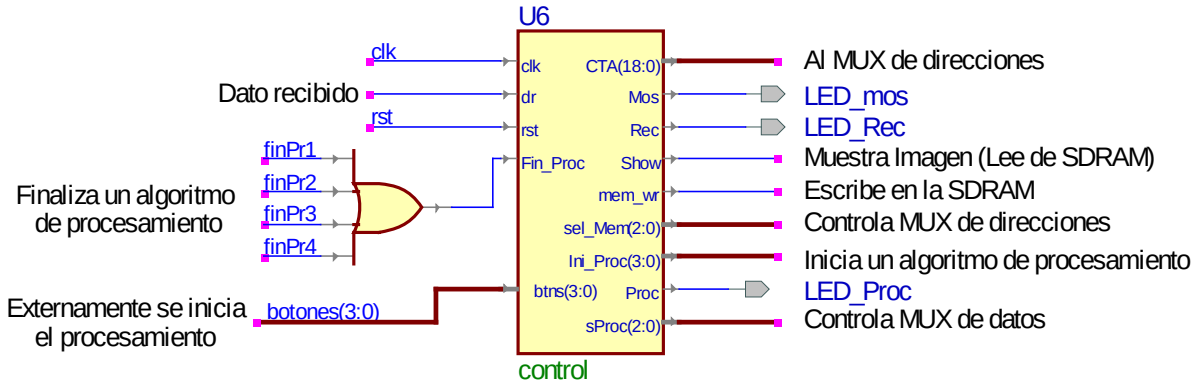

Fig. 13. Módulos para el Control del Sistema.

La Fig. 14 muestra al sistema completo, se le agregaron unos filtros para evitar el ruido de los botones y detectar flancos limpios.

4. Resultados

El sistema se desarrolló con la herramienta Acvite-HDL de Aldec, dado que facilita el diseño con máquinas de estados y con diagramas a bloques. La síntesis e implementación se realizaron con el entorno ISE de Xilinx. En la tabla 4 se muestra el uso de recursos en el FPGA, referentes al número de Slices, Bloques de entrada/salida y Bloques RAM internos.

El envío de una imagen requiere un periodo de 27 seg, por cada pixel se envían 10 bits a una velocidad de 115 200 bits/seg y son 307 200 piexeles/imagen.

Tabla 4. Recursos empleados en el FPGA.

Recurso	Uso	Disponibilidad	Porcentaje de Uso
Número de Slices	525	6,822	7%
Número de Bloques de Entrada/Salida	102	218	46%
Número de Bloques RAM internos	0	116	0%

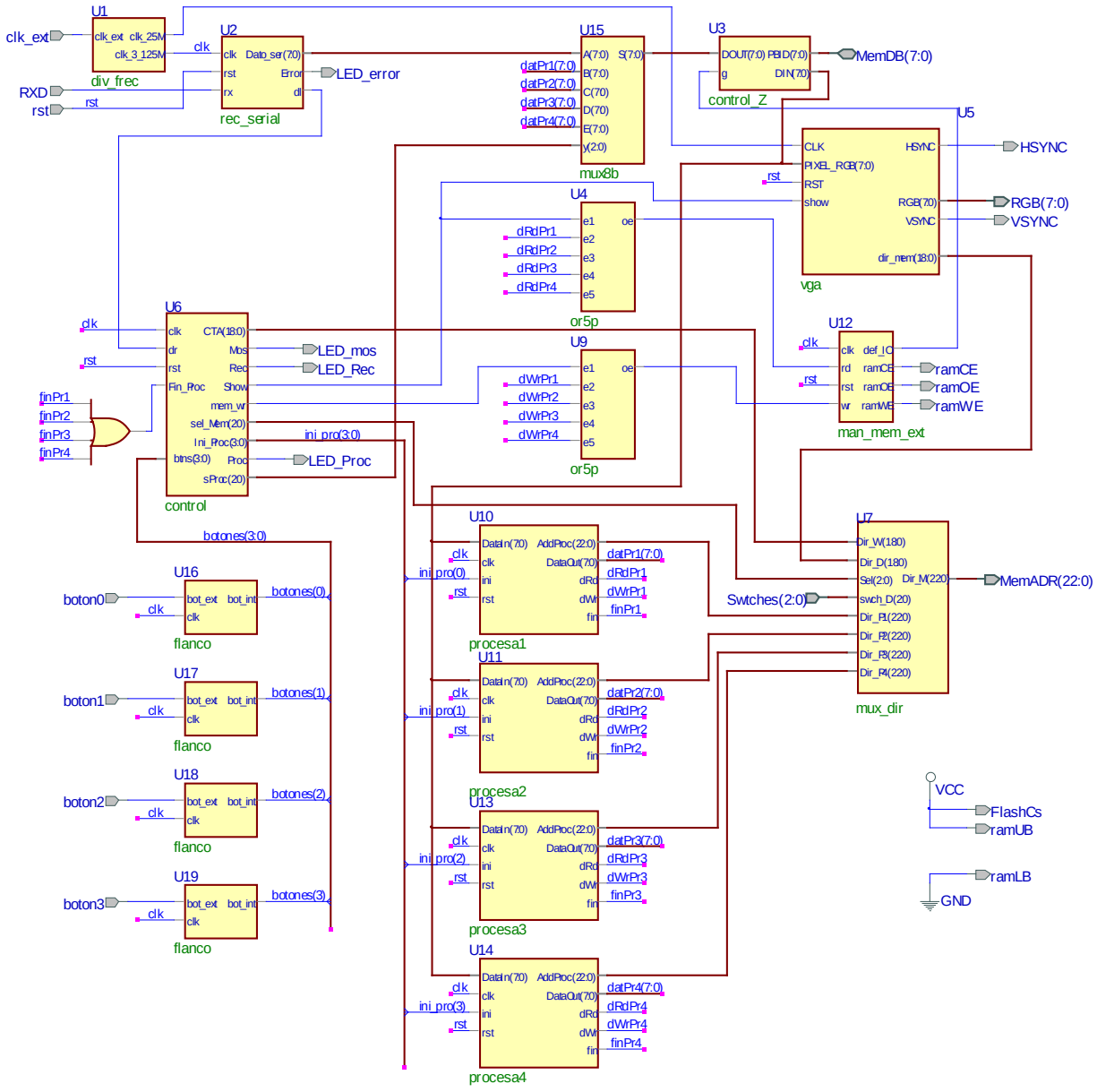

Fig. 14. Organización modular del sistema.

En la Fig. 15 se muestra al sistema implementado, se distingue a la tarjeta Nexys 2 y una pantalla RGB con una de las imágenes a ser procesadas. La imagen debe ser acondicionada

antes de su envío a la tarjeta, con la ayuda de un pequeño programa en Matlab se construye el archivo binario de la imagen (sin cabecera) y éste es descargado a la tarjeta empleando al programa Serial Port Monitor. En la Fig. 16(a) se puede ver a la imagen original y en la Fig. 16(b) al negativo de la imagen.

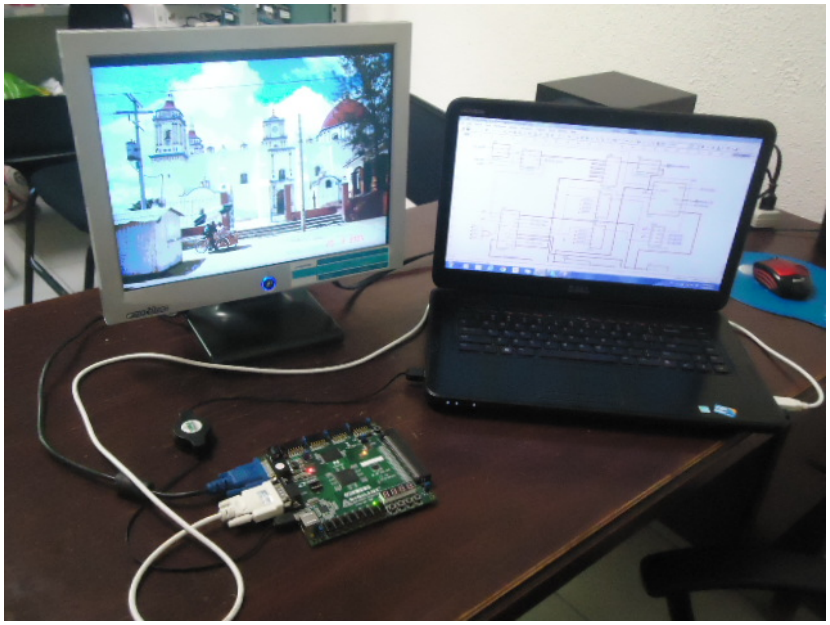

Fig. 15. Plataforma de desarrollo.

(a) (b)

Fig. 16. (a) Imagen Original y (b) Negativo de la Imagen.

n la Fig. 17 se presentan los gradientes de la imagen de la Fig. 16(a), en 17(a) se tiene el gradiente horizontal y en 17(b) al gradiente vertical. Ambos gradientes se combinan en la Fig. 18, en donde se pueden apreciar todos los contornos de la imagen original. Las imágenes mantienen 8 bits por pixel durante el procesamiento, con la distribución [BBGGGRRR], por ello en los resultados se aprecian algunos colores.

(a) (b)

Fig. 17. Gradiente de una imagen: (a) en dirección *x* y (b) en dirección *y*.

Fig. 18. Gradiente en las direcciones *x y*.

5. Conclusiones

El sistema funcionó favorablemente y con un rendimiento muy aceptable, dado que el procesamiento de una imagen de 640 x 480 pixeles se realiza en menos de dos segundos. La velocidad puede aumentarse si los módulos se sincronizan con una señal de reloj con una frecuencia mayor.

Se realizó un módulo por cada algoritmo de procesamiento, sin embargo, estos no pueden trabajar en forma concurrente porque requieren del acceso a la misma memoria.

Observando la tabla 4, se puede ver que la cantidad de Slices empleada es mínima (7 % del total), quedando una disponibilidad alta de recursos para algoritmos más complejos. Durante el proceso de desarrollo se manejó una imagen de 50 x 50 pixeles, sin emplear los bloques de memoria interna se utilizaba el 17 % de los Slices disponibles. Con esto se resalta el papel de la memoria externa, dado que se reduce considerablemente el uso de recursos internos.

La gama de trabajos que pueden continuar al presente se pueden encauzar por tres vertientes: Se puede emplear una cámara CCD para la adquisición de una escena real, se pueden aprovechar los recursos disponibles en el FPGA para aplicar técnicas de procesamiento más complejas y empleando los bloques de memoria interna, se puede acondicionar una especie de memoria caché, con la que se favorezca la velocidad de procesamiento o se haga posible el procesamiento concurrente.

6. Referencias

[1] Página web del Toolbox de PDI Matlab: http://www.mathworks.com/products/image/index.html. Último acceso Junio 2014.

[2] Página web de la librería Opencv: http://opencv.org/. Último acceso Junio 2014.

[3] Gonzalez, R. C. and Woods, R. E., *Digital Image Processing*, *3rd ed.*, Prentice Hall, Upper Saddle River, NJ.

[4] Gonzalez, R. C., Woods, R. E., and Eddins, S. L., Digital Image Processing Using MATLAB, 2nd ed., Gatesmark Publishing, Knoxville, TN.

[5] J. R. Parker, Algorithms for Image Processing and Computer Vision, Wiley, 2010.

[6] G. Bradsky and A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, 1st ed., O'Reilly Media, 2008.

[7] Barinova, Olga, Victor Lempitsky, and Pushmeet Kholi. "On detection of multiple object instances using hough transforms." Pattern Analysis and Machine Intelligence, IEEE Transactions on 34.9 (2012): 1773-1784.

[8] Butko, Nicholas J., and Javier R. Movellan. "Optimal scanning for faster object detection." Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009.

[9] Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011, November). ORB: an efficient alternative to SIFT or SURF. In Computer Vision (ICCV), 2011 IEEE International Conference on (pp. 2564-2571). IEEE.

[10] Li, Jianguo, Tao Wang, and Yimin Zhang. "Face detection using surf cascade." Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on. IEEE, 2011.

[11] R. Messing, C. Pal and H. Kautz, "Activity recognition using the velocity histories of tracked keypoints." Computer Vision, 2009 IEEE 12th International Conference on. IEEE, 2009.

[12] Cho, J., Mirzaei, S., Oberg, J. and Kastner, R. (2009, February). Fpga-based face detection system using haar classifiers. In Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays (pp. 103-112). ACM.

[13] Bouris, Dimitris, Antonis Nikitakis, and Ioannis Papaefstathiou. "Fast and efficient FPGA-based feature detection employing the SURF algorithm." Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on. IEEE, 2010.

[14] Meng, H., Appiah, K., Hunter, A., and Dickinson, P. (2011, June). Fpga implementation of naive bayes classifier for visual object recognition. In Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on (pp. 123-128). IEEE.

[15] Digilent Nexys2 Board Reference Manual. Copyright Digilent, Inc. All rights reserved. Doc: 502-107. Revision: June 21, 2008.

[16] Romero Troncoso, R., “Electrónica Digital y Lógica Programable”; Universidad de Guanajuato, 2007, ISBN: 968-864-449-8.

[17] Async/Page/Burst CellularRAMTM 1.5, MT45W8MW16BGX. Product Specification. Micron Technology, Inc. All rights reserved, 2004.

7. Autores

M. C. Felipe Santiago Espinosa es Maestro en Ciencias con especialidad en Electrónica por parte del INAOE, incorporado al IEM de la Universidad Tecnológica de la Mixteca, en donde es Profesor-Investigador desde 1998. Actualmente está cursando el Doctorado en Robótica en la misma institución. En el año de 2012 publicó su libro titulado “Los Microcontroladores AVR de ATMEL”.

Felipe de Jesús Trujillo Romero es Dr. en Sistemas Informáticos por el Instituto Nacional Politécnico de Toulouse, Francia, actualmente es profesor investigador de la División de Estudios de Posgrado de la Universidad Tecnológica de la Mixteca. Sus temas de interés son: la robótica, la visión por computadora y los sistemas reconfigurables.