Instituto Tecnológico y de Estudio Superiores de Monterrey

Campus Ciudad de México

TECNOLÓGICO DE MONTERREY®

Proyectos de Ingeniería II

Sistema de Redimensionamiento de Video Digital

Reporte Final

Alumnos:

Alejandro López Robles [redacted]

Emilio Galán Medina [redacted]

Gabriel López Resendiz [redacted]

Profesor:

Ing. Vicente Quintanilla Bonifaz

Asesores:

M. en C. Alfredo Mantilla Caeiros

Dr. Emmanuel Moya Anica

Fecha de entrega: 18 de abril de 2006

INDICE DE CONTENIDO

I. INTRODUCCIÓN

1. Área de desarrollo

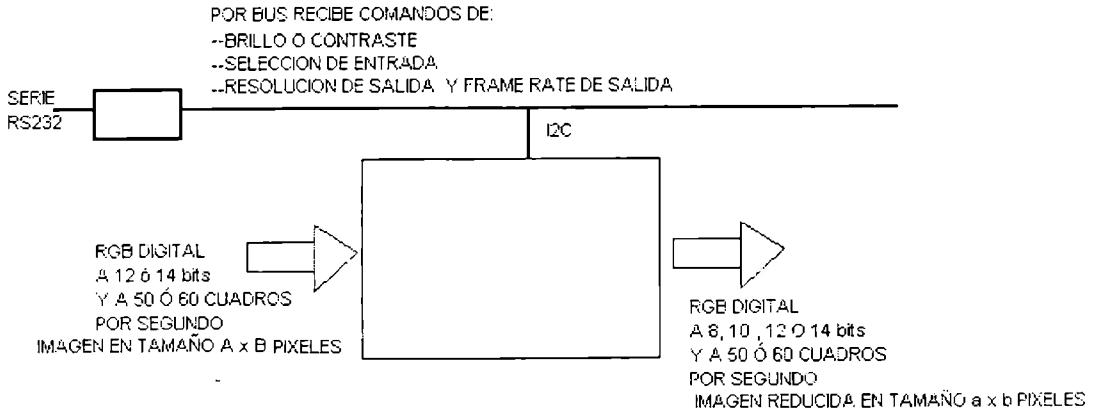
Nuestro proyecto forma parte de una tecnología emergente que es el video digital sobre pantallas a base de diodos emisores de luz (LEDs). Esta tecnología requiere de técnicas de procesamiento digital, que permitan tener la mejor relación entre tiempo de procesamiento y calidad en la resolución del video.

Las áreas de desarrollo que abarcará nuestro proyecto son:
- Procesamiento Digital de Señales
- Electrónica y Sistemas Digitales

2. Descripción del proyecto

El proyecto consiste en el diseño e implementación de un sistema de redimensionamiento de la imagen de video digital. El sistema recibe video digital en formato "RGB" (Rojo, Verde, Azul) de 12 o14 bits de un tamaño cualquiera. Por medio de un bus I^2C recibe valores de ancho, alto, resolución, brillo y contraste de la imagen de salida así como valores de ancho, alto, resolución, de la imagen de entrada. Una vez recibidos los valores de conversión, el sistema realiza un redimensionamiento de la imagen de entrada, produciendo una imagen de salida con los parámetros especificados.

2.1 Diagrama de Bloques del Sistema



3. Justificación

3.1 Tecnológica

La mayoría de las pantallas luminosas que podemos ver actualmente en nuestro país, en lugares como estadios, avenidas, universidades y centros comerciales, normalmente son pantallas de LEDs de unos cuantos colores, que solo despliegan caracteres alfanuméricos, imágenes sencillas y animaciones básicas. Nuestro proyecto representa una evolución en la tecnología de las pantallas luminosas, pasando de las tradicionales pantallas ya mencionados, a pantallas con formato de color RGB digital, capaces de desplegar imágenes y animaciones de alta calidad, así como video en tiempo real.

3.2 Económica

Actualmente, el mercado de sistemas de video digital sobre pantallas a base de LEDs en México es muy limitado. Esto se debe principalmente a que en México todavía no se desarrollan sistemas con esta tecnología y a que las grandes compañías internacionales que ofrecen esta tecnología, imponen costos muy altos a sus sistemas, lo que hace que dichos sistemas se vuelvan poco rentables para los usuarios.

Sin embargo, la demanda de sistemas de video digital sobre pantallas de LEDs, tanto a nivel internacional como en nuestro país, es cada vez más grande. Por esta razón, la empresa Kokai Electrónica S.A. de C.V., ha decidido incursionar en el mercado del video digital sobre pantallas de LEDs, proponiendo el desarrollo de sistemas menos costosos que los que actualmente se encuentran disponibles y con buena calidad, para poder ser competitivos en el mercado nacional e incluso internacional.

3.3 Social

Debido a la calidad y a los costos relativamente bajos que se tendrá con la tecnología de video sobre pantallas de LEDs, estos sistemas pueden tener muchas aplicaciones que van más allá del ámbito puramente comercial. Esto es, se pueden utilizar para fines que puedan contribuir en el aspecto social. Un ejemplo muy claro es en el tránsito de las ciudades. Se pueden colocar tableros con video en las principales avenidas, donde se desplieguen secuencias de video que muestren el estado del tránsito en la zona donde se encuentra la pantalla, así como en las rutas alternativas para circular.

4. Objetivos y metas

4.1 Objetivos

El objetivo principal de nuestro proyecto es diseñar y construir un sistema de redimensionamiento de imagen que brinde la mejor relación de tiempo de procesamiento contra calidad en la resolución de la imagen de video, para acoplarlo en el sistema video digital sobre pantallas de LEDs propuesto por Kokai Electrónica.

4.2 Metas

- Primer semestre:

Las metas específicas para el trabajo del primer semestre fueron las siguientes:

- Entender la teoría matemática y los algoritmos de Interpolación Bilineal y Vecino más Cercano, utilizados en el proceso de redimensionamiento de video digital.
- Desarrollar un algoritmo de redimensionamiento de imágenes a partir de eliminación de píxeles.
- Diseñar y construir una interfaz RS232 – I^2C para el envío de parámetros.
- Conocer de manera general el funcionamiento de los procesadores digitales de señales (DSP) de la familia C6000 de Texas Instruments.
- Realizar pruebas de los algoritmos de redimensionamiento de imágenes en Matlab y Simulink.

- Segundo semestre:

Debido a una serie de contratiempos y dificultades a las que nos enfrentamos a lo largo del segundo semestre de trabajo y que serán explicadas a detalle más adelante en el trabajo, las metas específicas para este período se vieron modificadas considerablemente, quedando como se muestra a continuación:

- Implementar y realizar la simulación de los algoritmos de redimensionamiento de imágenes en el DSP.
- Desarrollar e implementar un sistema en el FPGA que realice la adquisición de video en formato RGB digital de 8 bits por canal y lo redimensione a la mitad de su tamaño original.

5. Estado del arte

Actualmente no existe una empresa mexicana que fabrique pantallas de LEDs con la capacidad para desplegar video digital en tiempo real. Por este motivo, en el mercado existen muy pocas opciones, todas compañías extranjeras, como Daktronics o HighTech que fabrican este tipo de pantallas.

Debido a la falta de desarrollo, la tecnología del video digital sobre pantallas de LEDs en México es relativamente nueva, sin embargo, debido a la creciente demanda que actualmente tienen las pantallas de LEDs full color para despliegue de video, Kokai Electrónica es la primera empresa mexicana que incursiona en el desarrollo de dicha tecnología.

En este sentido nuestro proyecto tendrá una importante contribución al estado del arte, ya que representa el primer acercamiento hacia el desarrollo de sistemas de video digital sobre pantallas de LEDs fulll color, por parte de una empresa mexicana.

6. Infraestructura y Recursos Necesarios

La infraestructura y recursos que utilizamos en el proyecto se dividieron de acuerdo a las diferentes etapas de desarrollo del mismo:

La primera etapa consistió en realizar una investigación sobre los algoritmos de redimensionamiento, las características del bus I^2C y el uso y funcionamiento del DSP C-6000. Para poder llevar a cabo esta investigación requerimos de un extenso material bibliográfico que consistió en libros, revistas, tutoriales y artículos. Nuestras principales fuentes para obtener este material fueron la biblioteca del campus, la biblioteca digital del Sistema Tecnológico de Monterrey y el sitio en Internet de Texas Instruments.

La segunda etapa del proyecto consistió en el desarrollo de la interfaz RS232-I^2C. En esta etapa, los recursos e infraestructura utilizada fueron una computadora personal con puerto serie RS232, un microprocesador PIC16F877 de Microchip y las herramientas de software para programación en C/C++ Microsoft Visual Studio, Borland C++ Builder y Microchip MPLAB.

En la tercera etapa de desarrollo, se trabajó en la implementación y simulación de los algoritmos de redimensionamiento de imágenes en el DSP, por lo que en esta etapa los recursos utilizados fueron los siguientes: Computadora personal, software de simulación Matlab/Simulink de MathtWorks y el kit de desarrollo TMS320C6416T DSK de Spectrum Digital, el cual incluye todas las herramientas necesarias para programar y hacer pruebas con el DSP C614 de Texas Instruments.

La etapa final de nuestro trabajo consistió en desarrollar e implementar un sistema de adquisición y redimensionamiento de video digital en formato RGB de 24 bits. Este sistema fue realizado utilizando el kit de desarrollo XC3S200, para la familia de FPGAs Spartan-3 de Xilinx, el kit de desarrollo 161B-RX-DVI de Silicon Image y el software para programación en VHDL Xilinx ISE.

II. MARCO TEÓRICO

1. Bus I^2C

1.1 Introducción

Originalmente el bus I^2C fue creado para enlazar pocos elementos a una misma tarjeta. La capacitancia máxima permitida fue puesta a 400 pF para permitir una subida y una bajada de óptima de la señal de reloj y la integridad de la señal de datos con una velocidad máxima de transmisión de 100 Kb/s. En 1998 los estándares del bus I^2C incrementaron la velocidad a 3.4 Mb/s. Actualmente todos los dispositivos I^2C son diseñados para comunicarse en un sistema de dos cables. Por esta razón el bus I^2C es actualmente usado en sistemas en donde se requiere que varios dispositivos estén conectados a una misma tarjeta. En la actualidad existen nuevas extensiones de bus y de dispositivos para incrementar el límite de los 400 pF a una expansión de 20 dispositivos lo que permite un mayor manejo de dispositivos.

El bus I^2C es el mejor bus para mantener velocidades bajas y aplicaciones de control donde los dispositivos puedan ser conectados y desconectados del sistema.

1.2 Terminología del bus I^2C

- Transmisor – Es el dispositivo que manda datos al bus. Un transmisor puede ser un dispositivo que pone datos en el bus (un transmisor amo) o puede tener respuestas de otros dispositivos (un transmisor esclavo)
- Recibidor – Es el elemento que recibe datos del bus.
- Amo – Es el componente que inicializa la transferencia, genera la señal de reloj y termina la transferencia. Un amo puede ser un transmisor o un recibidor.
- Multiamo – La habilidad para que más de un amo co-exista en el bus al mismo tiempo sin tener colisiones.
- Arbitración – Es el rango precedente que autoriza a un solo amo en un tiempo determinado tomar el control del bus.
- Sincronización - Es el rango precedente que sincroniza al reloj proveer dos o mas amos.
- SDA – Línea de Señal Datos (En serie Data)
- SCL – Línea de Señal de Reloj (En serie Clock)

1.3 Características

- Es un bus en serie para velocidades bajas y medias
- Resistente a impulsos (glitches) y ruidos
- Soportado por un rango largo y de diversos dispositivos en serie
- Es conocido como un protocolo robusto
- La comunicación puede soportar distancias grandes
- Sólo dos líneas de bus son requeridas: una línea en serie de datos (SDA) y una línea en serie de reloj (SCL).
- Cada dispositivo conectado al bus es direccionable por medio de software por una única dirección y una relación amo-esclavo que existe en todo momento. Los amos pueden operar como amos-transmisores o amos-recibidores.
- Orientado a un puerto de 8 bits con transferencias bidireccionales de 100Kb/s en modo estándar, 400Kb/s en modo rápido y de 3.4Mbps en modo de alta velocidad.

- El número máximo de electos conectados al bus está limitado a la carga de 400pF.

1.4 Procedimiento de comunicación

Cada dispositivo conectado al bus debe:

- Esperar hasta que no haya actividad en el bus I^2C. Cuando las señales de SDA y SCL estén en nivel alto indica que el bus esta libre.
- Pone un mensaje en el bus que dice "es mío" – He EMPEZADO a usar el bus y todos los demás dispositivos ESCUCHAN al bus para que se les pueda asignar una dirección.
- Provee la señal de reloj a la línea (SCL). Esta será usada como referencia de tiempo para los demás dispositivos para que cada bit de DATOS en el cable de datos (SDA) sea correcto y usado. El dato en el bus de datos (SDA) debe ser valido en el tiempo en que el cable reloj (SCL) conmuta un voltaje de alto a bajo.
- Pone en forma en serie la dirección binaria del dispositivo que quiere comunicarse con el bus I^2C.
- Pone un mensaje (un bit) que indica que si MANDA o RECIBE datos de otro dispositivo.
- Manda una señal (un bit) que ha reconocido su dirección y que esta listo para tener comunicación.
- Después de que el otro dispositivo ha detectado esta señal, la transmisión empieza a efectuarse.
- Cuando la transferencia de datos ha sido terminada el primer dispositivo debe acabar la sesión con un mensaje de ALTO, que es un bit de información que se transmite por los cables SDA y SCL del bus.

Las reglas del bus dicen que cuando datos o direcciones están siendo mandadas, el cable de DATOS solo está permitido a ser cambiado en voltaje (1 o 0) cuando el voltaje en la línea de reloj es BAJO.

Cualquier dispositivo con la capacidad de iniciar mensaje es llamado "amo". El puede conocer exactamente los otros chips que están conectados y cuyo caso sólo asigna una dirección al dispositivo con el cual desea tener comunicación.

1.5 Teoría de operación

Es importante mencionar que la medición del tiempo de subida de señal del reloj se estipula en un 30% y 70% de V_{DD} (fig. 1.1), esto debido a que los "buffers" distorsionan los bordes de subida de la señal en el bus, por lo que este 30% y 70% no se considera como parte de la subida de tiempo.

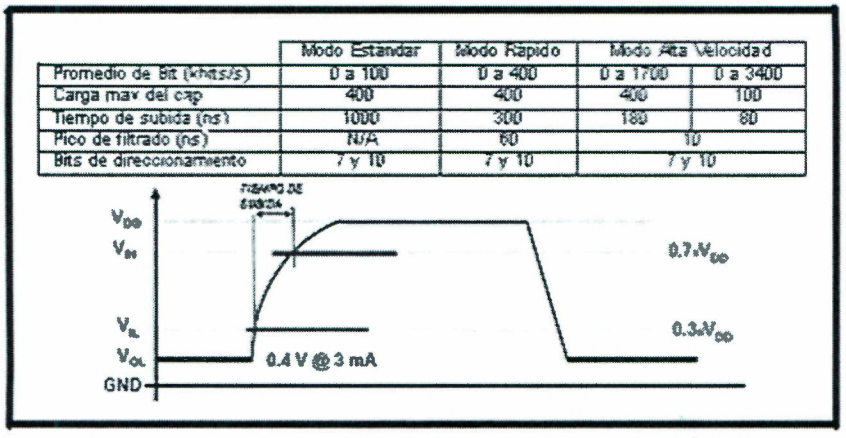


Fig. 1.1 Subida de la señal de la señal de reloj

1.6 Condiciones de inicio y alto (START & STOP conditions)

Condición START: Una transmisión en alto o un bajo en la línea de SDA mientras SCL es alto.
Condición STOP: Una transmisión de bajo a alto en la línea de SDA mientras SCL es alto.

El amo siempre genera las condiciones de START y STOP (fig. 1.2). El bus es considerado como ocupado después de la condición de START. El bus es considerado libre después de la señal de STOP. El bus permanece ocupado si se repite una señal START (Sr) en vez de una señal de STOP. En este caso la señal START (S) y la señal repita START (Sr) son condiciones iguales. El símbolo S será usado como un término genérico para representar las señales START y la señal START repetida.

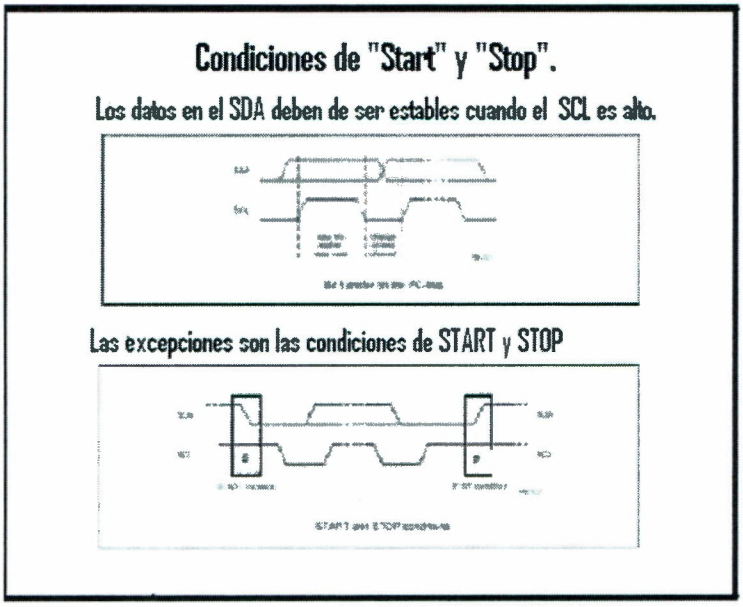


Fig. 1.2 Condiciones de STOP Y START

1.7 Configuración de Hardware

La figura 1.3 muestra la configuración del hardware para el bus I^2C. Los cables del bus (SCL y SDA) tienen la misma configuración, estos son puestos a un nivel lógico "alto" a través de resistencias conectadas a una fuente positiva, usualmente es de 3.3V o 5.5.

Todos los dispositivos conectados tienen un dispositivo de colector abierto (drenaje abierto para CMOS) que puede transmitir datos poniendo el bus en bajo y una alta impedancia a los amplificadores que monitorean el voltaje del bus para recibir datos. A menos que los dispositivos se estén comunicando poniendo al transistor en bajo para poner en bajo al bus, ambas líneas permanecen en "alto". Para incitar comunicación un dispositivo pone a la línea de SDA en bajo, después tiene la responsabilidad de manejar la línea de SCL con los pulsos de reloj hasta que sea acabada la sesión, por lo que es llamado amo.

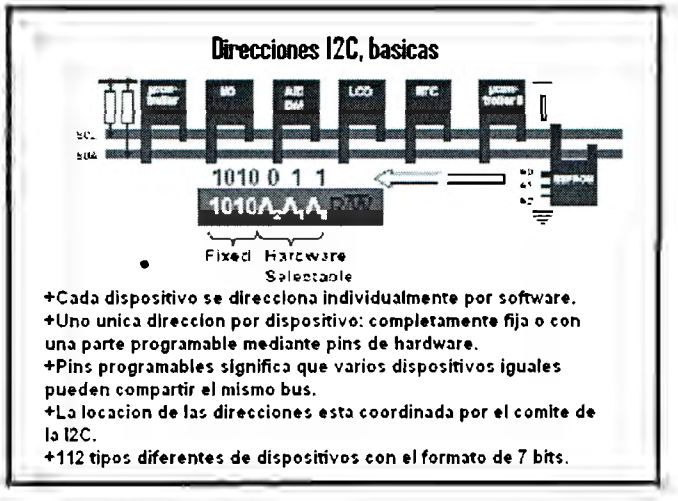


Fig. 1. 3 Configuración del Hardware

1.8 Direccionamiento

Cualquier dispositivo en el bus I^2C puede ser agregado y puede tener comunicación con todos los dispositivos. Cada dispositivo tiene una dirección única de 7 o 10 bits. Para dispositivos de 7bits, típicamente los primeros cuatro bits son fijos y los otros tres bits son puestos para los pines de la dirección del hardware (A0, A1 y A2) que permiten al usuario modificar las direcciones para un total de ocho dispositivos en el bus I^2C, estos pines son puestos a Vcc, a veces a través de una resistencia o simplemente puestos a tierra (GND).

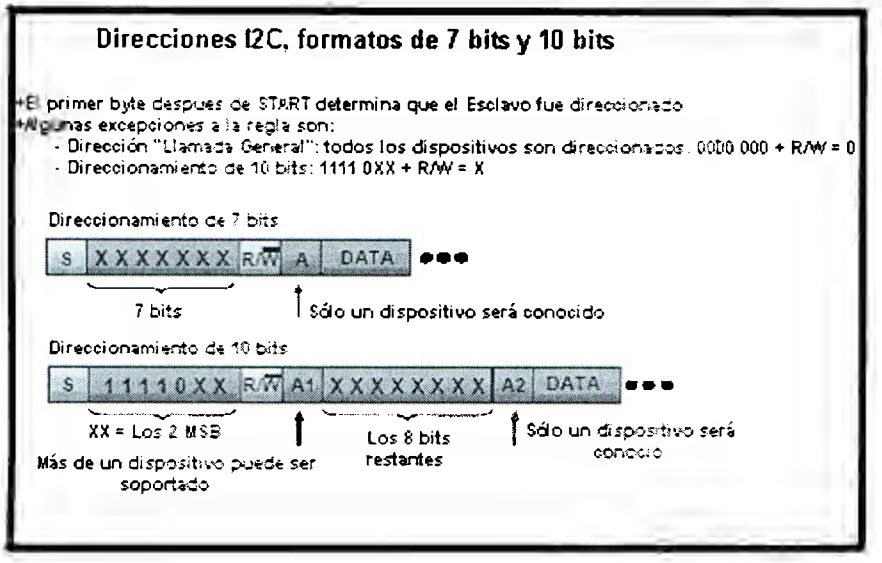


Fig. 1.4 Direccionamiento de 7 y 10 bits

El último bit del octeto inicial indica si el amo mandará (write) o recibirá (read) datos del esclavo. Cada secuencia de transmisión debe empezar con la condición de Stara y terminar con la condición de stop.

En el octavo pulso, la línea SDA es puesta en "alto" si los datos serán leídos desde otro dispositivo, o "bajo" si los datos serán mandado (write). Durante el noveno pulso el amo libera a la línea de SDA, si otro dispositivo es conectado al bus y ha decodificado y reconocido su dirección, pondrá a la línea SDA en bajo, por lo tanto el chip que responde a la petición es llamado esclavo.

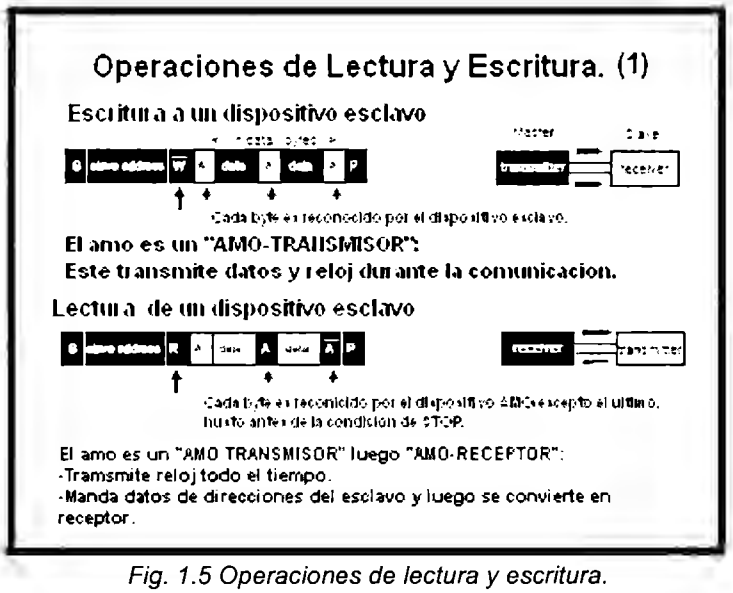


Fig. 1.5 Operaciones de lectura y escritura.

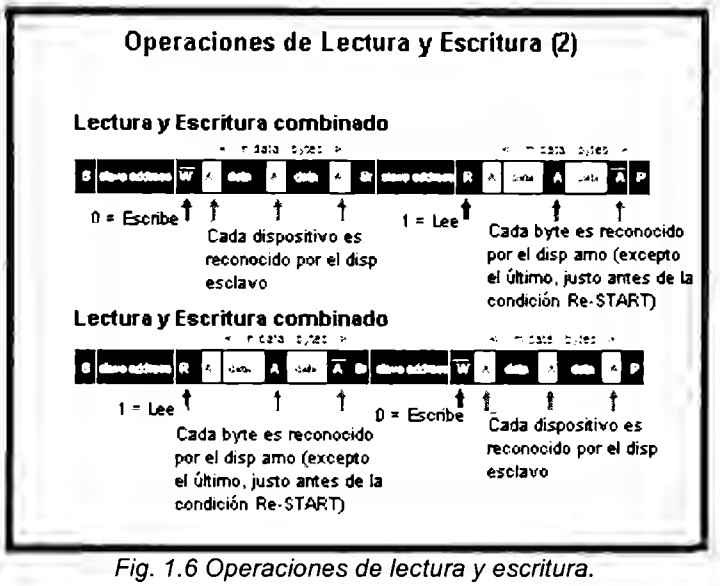


Fig. 1.6 Operaciones de lectura y escritura.

1.9 Terminología del bus de transferencia

- F (FREE) – Indica que el bus está libre; la línea de datos SDA y la de reloj SCL son puestas en estado alto.
- S (START) o S_R (START repetida) – Indica que la transferencia comienza con la condición de start (no el bit de start). El nivel de SDA cambia de estado alto a bajo, mientras que SCL permanece en alto. Cuando esto ocurre el bus está ocupado.
- C (CHANGE) – Mientras la línea de reloj SCL está en bajo, el bit de datos para ser transmitido puede ser puesto en la línea de SDA por un transmisor. Durante este tiempo, SDA puede cambiar su estado, con lo que SCL permanece en bajo
- D (DATA) – Un bit bajo o alto de información en la línea SDA es válida durante el nivel alto del SCL.
- P (STOP) – La transferencia de datos es terminada por la condición de stop (no un bit de stop). Esto ocurre cuando el nivel de SDA pasa de un estado bajo a un estado alto, mientras que SCL permanece en alto. Cuando la transferencia de datos ha sido terminada, en bus esta libre nuevamente

En la figura 1.7 se muestra la fase de reconocimiento y como los esclavos pueden ajustar la señal de reloj.

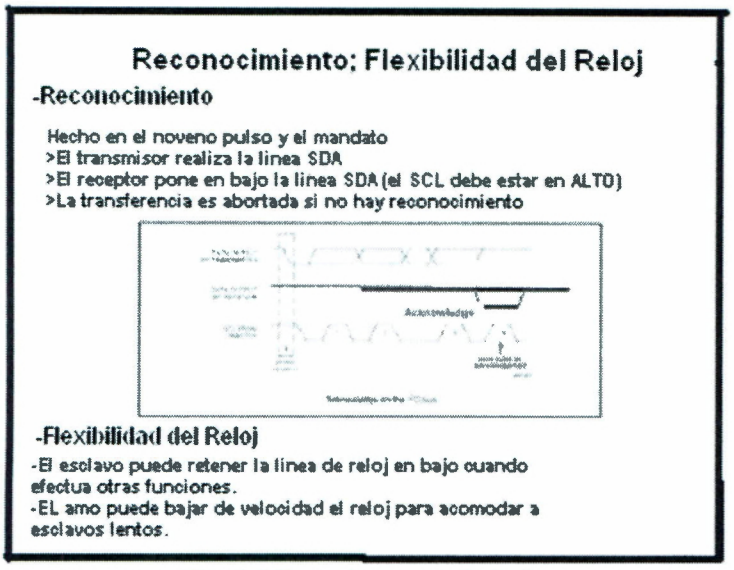


Fig. 1.7 Fase de reconocimiento.

Si hay dos amos en el mismo bus, hay una arbitrarización si ambos amos tratan de tomar el control del bus al mismo tiempo. Cuando dos dispositivos tratan de iniciar la comunicación al mismo tiempo, pueden tratar de generar pocos ciclos del reloj y el dato que se escoge, pero eventualmente una salida a "bajo" cuando el otro trata por una en estado "alto". La señal en "bajo" gana, así que el dispositivo que no obtuvo la comunicación esperara hasta que el bus se encuentre nuevamente en estado libre.
Una vez que el amo (e.g., microcontrolador) tiene el control, ningún otro amo puede tomar el control hasta que el primer amo mande una condición de stop y pone el bus en un estado en estado de espera.

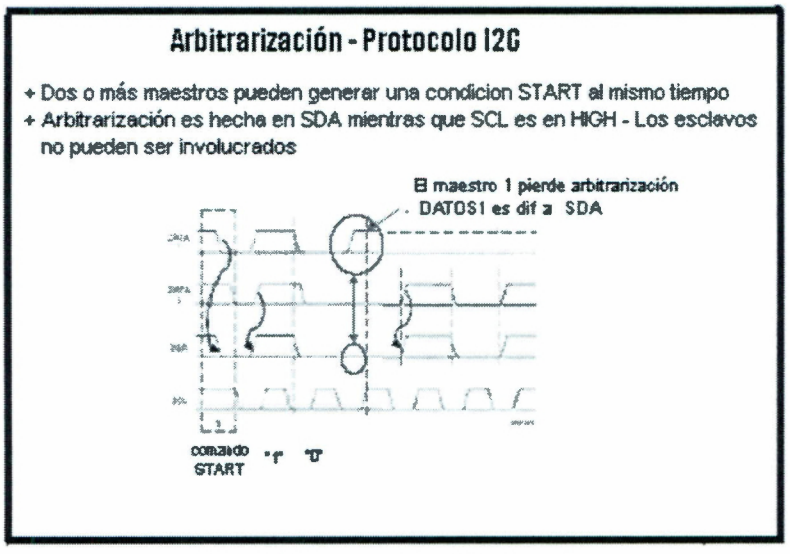


Fig. 1.8 Proceso de arbitrarización

2. Elementos de una imagen

Cada pequeña área de luz o sombra recibe el nombre de un píxel, que es la abreviación en inglés de "picture element". Todos estos elementos juntos tienen toda la información visual de una escena. Si estos elementos son transmitidos y reproducidos con los mismos niveles de luz o sombra como la señal original y en una posición correcta, entonces la imagen es reproducida.

2.1 Análisis horizontal y vertical

Una imagen de televisión es escaneada en una serie de líneas horizontales, una debajo de otra. Este barrido permite para una señal de video, incluir todos los elementos de una imagen. Para obtener una señal de video para todas las variaciones de luz y sombra, todos los detalles de la imagen son escaneados en un determinado orden de tiempo.

El barrido hace que la reproducción de una imagen en televisión, sea diferente que en una foto. En una fotografía la imagen se reproduce toda a un mismo instante, en televisión, se hace línea por línea y trama por trama.

Los elementos de una imagen son escaneados sucesivamente, de izquierda a derecha y de arriba hacia abajo. Este método se llama "barrido lineal horizontal".

La secuencia de barrido se hace de la siguiente manera:

El rayo de electrones barre una línea horizontal, cubriendo todos los elementos de la imagen en ese tiempo.

Al final de cada línea, el rayo regresa muy rápidamente al lado izquierdo para iniciar el barrido de la siguiente línea horizontal. El tiempo de regreso se llama retrazo o "flyback". No se registra información en el retraso pero, este debe ser muy rápido porque si la imagen esta transmitiendo, se puede perder información.

Cuando el rayo haya regresado al lado izquierdo, la posición vertical baja de nivel para que el rayo no haga el barrido de la misma línea. Cuando el rayo se encuentra hasta el último nivel de líneas, este regresa hasta su posición inicial, hasta arriba y en el lado izquierdo para iniciar el barrido de otra imagen.

2.2 Líneas

El número de barrido o exploración de líneas para una imagen debe ser grande, esto para incluir tantos elementos sean posibles de la imagen. Pero, otros factores limitan la opción. El estándar son 525 barrido de líneas para una imagen completa o trama, esto, para la transmisión de televisión de 6 Mhz.

2.3 Tramas

El rayo se mueve lentamente para bajo mientras se hace el barrido horizontal. Este movimiento de barrido vertical es necesario para que las líneas no sean escaneadas una sobre la otra. El barrido horizontal produce las líneas de izquierda a derecha, mientras que el barrido vertical recorre las líneas para llenar la trama de arriba hacia abajo.

2.4 Imagen

Como en el cine, en la televisión tienen que existir suficientes cuadros por segundo para poder realizar el efecto del movimiento. En ves de usar la tasa de 24 cuadros por segundo usada en el cine, la repetición de cuadros por segundo en la televisión es de 30.

En televisión cada trama es dividida en 2 partes, así de esta manera 60 vistas de la escena es lo que el ojo humano captara.

Este efecto se obtiene ínter lazando el barrido horizontal en 2 grupos, uno con los números pares y otro con los nones. Cada grupo con pares o nones es llamado un "campo".

Entonces, la repetición de 60 tramas por segundo se da porque se están escaneando durante un periodo de trama de 1/30 seg.

2.5 Frecuencias de barrido vertical y horizontal

La tasa de campo de 60 tramas por segundo es la frecuencia de barrido vertical. Esta es la tasa en la que el rayo de electrones completa su ciclo de movimiento vertical de arriba hacia abajo y de regreso arriba otra vez.

El número de líneas de barrido en un campo es la mitad del total de las 525 líneas para una trama completa, porque cada campo contiene todas las líneas. Esto hace que sean 262.5 líneas horizontales para cada campo vertical.

Como el tiempo para un campo es 1/60 seg. y el campo contiene 262.5 líneas, el numero de líneas por segundo son 262.5x60=15,750.

O considerando 525 líneas para un par sucesivo de campos, podemos multiplicar la tasa de tramas 30 por 525 que da lo mismo--15,750 líneas escaneadas en un segundo.

Esta frecuencia de 15,750 es la frecuencia a la cual el rayo de electrones completa su ciclo del movimiento horizontal de izquierda a derecha y de nuevo a la izquierda.

Nótese que al mismo tiempo en que el rayo se mueve horizontalmente también se mueve verticalmente de arriba hacia abajo. Esto crea una ligera inclinación a las líneas.

Este barrido se efectúa en los dos campos y estos dos campos conforman una trama o cuadro. Un cuadro es una imagen completa de 525 líneas, este proceso se le conoce como barrido ínter lazado horizontal.

El interlineado necesita un número impar de líneas y un número par de campos.

2.6 Línea horizontal de tiempo

El tiempo para cada línea horizontal es 1/15750 segundos, en términos de microsegundos esto es H=1000000/15750=63.5 m icrosegundos.

2.7 Sincronización horizontal y vertical

El tiempo empleado en el análisis corresponde a la distancia en la imagen. Para guardar el barrido entre transmisor y receptor, señales especiales de sincronización se necesitan y son transmitidas junto con la información de la imagen, Estas señales de tiempo son pulsos rectangulares que son usados para controlar el barrido.

Estos bits se señalización ocurren en un tiempo de supresión (blanking time), que es un tiempo donde no hay información de la imagen. Un pulso de sincronización horizontal al final de cada línea determina el comienzo de un "regreso" horizontal. Es importante que la sincronización sea al comienzo del "regreso" o al final del barrido. Sincronización vertical al final de cada campo determina el regreso vertical. En este punto el rayo de electrones de barrido esta en la parte mas baja de la imagen.

2.8 Supresión vertical y horizontal

Como parte de la señal de video, el voltaje de supresión esta en un nivel de negro. El voltaje de video en un nivel de negro corta la corriente del rayo de electrones. El propósito de los pulsos de supresión es hacer invisibles los retrasos requeridos en el barrido.

El tiempo requerido para supresión horizontal es de aproximadamente 16% de cada línea horizontal. Para el vertical es de aproximadamente 8% de la línea vertical. Los regresos ocurren durante el tiempo de supresión debido a la sincronización y el barrido.

2.9 La señal de video compuesto

Específicamente las 2 señales transmitidas de color son luminancia y crominancia:

2.9.1 Señal de luminancia

Esta señal contiene solo variaciones de brillo en la información de la imagen, incluyendo los detalles finos. La señal de luminancia es usada para reproducir la imagen en blanco y negro o monocromático. Se le llama también la señal Y.

2.9.2 Señal de crominancia

Esta señal contiene la información del color.

2.10 Características de la imagen

2.10.1 Brillo

Es la intensidad promedio de la iluminación. Determina el nivel del fondo en una imagen reproducida. Elementos individuales de la imagen pueden variar por arriba y por debajo de este promedio. El brillo en la pantalla depende en la cantidad de alto voltaje. En los televisores el control del brillo varía la polarización dc en el tubo de la imagen. La pantalla fluorescente de la imagen tan solo un pequeño punto en un tiempo.

2.10.2 Contraste

El contraste es la diferencia en intensidad entre las partes negras y blancas en la imagen reproducida. El rango de contraste debe ser tan grande para producir una imagen fuerte, con un blanco brillante y negro oscuro.

3. Modelos de Color

El propósito de un modelo de color (también llamado *espacio de color*, o *sistema de color*) es facilitar la especificación de colores en algún modo aceptado de manera general. En esencia, un modelo de color es una especificación de un sistema de coordenadas y un subespacio dentro de dicho sistema, en el cual cada color es representado por un solo punto.

La mayoría de los modelos de color utilizados hoy en día están orientados ya sea hacia hardware (como monitores a color o impresoras), o bien, hacia aplicaciones cuyo objetivo es la manipulación de color (como la creación de gráficos de color para animaciones). En términos de procesamiento digital de imágenes, los modelos más utilizados en la práctica son el modelo RGB (rojo, verde, azul) para monitores a color y una gran variedad de cámaras fotográficas y de video; el modelo CMYK (cian, magenta, amarillo, negro) para impresoras a color y el modelo HSI (tonalidad, saturación, intensidad), que corresponde en gran medida a la forma en la que los humanos describimos e interpretamos los colores.

3.1 El modelo RGB

En el modelo RGB, cada color aparece en sus componentes espectrales primarias de rojo, verde y azul. Este modelo se basa en un sistema de coordenadas cartesianas. El subespacio de interés es el cubo mostrado en la fig. 6.7, en el cual los valores RGB se encuentran en tres ejes; los valores CMY se encuentran en las otras tres esquinas; el valor para el negro se encuentra en el origen y el blanco se encuentra en la esquina más alejada del origen. En este modelo, la escala de grises (puntos para valores RGB iguales) se extiende desde el negro hasta el blanco por la línea que une ambos puntos. Los diferentes colores en este modelo son puntos sobre, o dentro del cubo y son definidos por vectores que se extienden desde el origen. Por conveniencia, se supone que todos los valores de color han sido normalizados de manera que el cubo mostrado en la figura sea el cubo unitario. Esto es, se supone que todos los valores de R, G y B se encuentran en el rango [0,1].

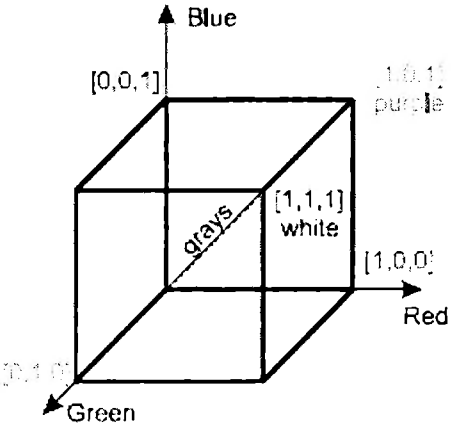


Figura 3.1 Cubo RGB

Las imágenes representadas en el modelo RGB consisten en tres imágenes componentes, una para cada color primario. Cuando son alimentadas a un monitor RGB, estas tres imágenes se combinan en la pantalla de fósforo para producir una imagen de color compuesto. El número de bits utilizado para representar cada píxel en el espacio RGB se conoce como *profundidad de píxel*. Consideremos una imagen RGB en la cual cada una de las imágenes, rojo, verde y azul, es una imagen de 8 bits. Bajo estas condiciones, se dice que cada píxel de color RGB (esto es, una tripleta de valores RGB) tiene una profundidad de 24 bits (tres planos de imagen multiplicados por el número de bits por plano). El término imagen *full-color* se usa comúnmente para denotar imágenes RGB de 24 bits. El número total de colores en una imagen RGB de 24 bits es $(2^8)^3 = 16{,}777{,}216$.

Muchos de los sistemas que utilizan el sistema RGB hoy en día están limitados a 256 colores, o bien, existen aplicaciones en las cuales simplemente no tiene sentido utilizar más colores. Dada la variedad de sistemas que actualmente se utilizan, es importante considerar un subconjunto de colores que se usan frecuentemente y que deban ser reproducidos con cierta exactitud, independientemente de las capacidades de hardware del observador. A este subconjunto de colores se le conoce como colores RGB seguros (*safe RGB colors*).

4. Procesamiento Digital de Imágenes

4.1 Procesamiento básico de imágenes

El procesamiento de datos en sistemas de visión puede enfocarse desde dos perspectivas:

1) Alteración píxel a píxel de los datos en una escala global (individuales)
2) Operaciones basadas en múltiples puntos (vecindad)

La generación de un nuevo píxel en una nueva imagen será una función, ya sea del valor de cada píxel en su localización individual, o bien de los valores de los píxeles en la vecindad de un píxel dado, como se indica en la figura:

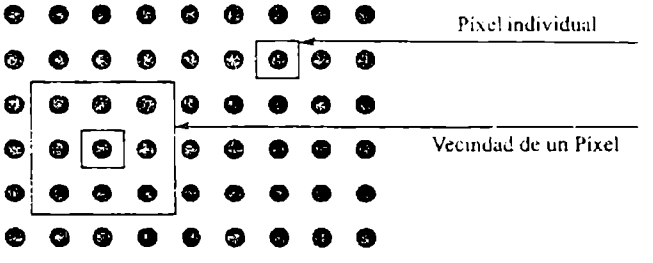


Figura 4.1. Funciones de punto y vecindad

Para efectos de sencillez y de una mejor comprensión de las operaciones que se realizan en el procesamiento de imágenes digitales, todas las operaciones que se presentan a continuación, tanto individuales como de vecindad, se refieren a imágenes en escala de grises o bien a imágenes binarias (blanco y negro puro). Para el caso de imágenes digitales en formato RGB, el procesamiento es esencialmente el mismo, ya que se procesa cada uno de los tres canales (R, G y B) por separado. De esta forma, las operaciones que se presentan a continuación se pueden efectuar con imágenes en formato RGB, aplicando dichas operaciones a cada canal por separado, de la misma manera que si se tratara de la escala de grises.

4.2 Operaciones de Vecindad

Las operaciones de vecindad utilizan el mismo procedimiento excepto que el nuevo valor del píxel en la imagen de salida depende de una combinación de los valores de los píxeles en la vecindad del píxel de la imagen original.

Básicamente consiste en transformar el valor de un píxel p en la posición (x, y) teniendo en cuenta los valores de los píxeles vecinos. Por ejemplo, si consideramos una vecindad $E_8(p)$, realizamos una suma ponderada con los valores de los 8 vecinos y el resultado de dicha suma es el valor del nuevo píxel q de la imagen de salida en la misma posición (x, y). Lo único que resta es definir los valores de ponderación, lo cual se hace generalmente definiendo una máscara con valores constantes. Dicha máscara es realmente un filtro, por lo que dependiendo de la naturaleza del mismo, así será el resultado final. Por ejemplo, si definimos la máscara siguiente:

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

el valor del píxel $q(x, y)$ vendría dado por la siguiente suma ponderada, con los factores de ponderación definidos por la máscara

$$\begin{aligned} q(x,y) = {} & 1 \cdot p(x-1,y-1) + 2 \cdot p(x,y-1) + 1 \cdot p(x+1,y-1) \\ & +0 \cdot p(x-1,y) + 0 \cdot p(x,y) + 0 \cdot p(x+1,y) \\ & -1 \cdot p(x-1,y+1) - 2 \cdot p(x,y+1) - 1 \cdot p(x+1,y+1) \end{aligned}$$

Realizando esta transformación sobre todos los píxeles de la imagen original con la misma máscara obtendremos una nueva imagen de salida cuya dimensión es inferior a la original, ya que esta operación no se puede realizar sobre los píxeles extremos de la imagen original al no tener todos los vecinos.

Mediante este mismo procedimiento de vecindad podemos conseguir un mayor contraste de la imagen original con el uso de la máscara de ponderación siguiente:

$$\begin{pmatrix} -0.1667 & -0.6667 & -0.1667 \\ -0.6667 & 4.3333 & -0.6667 \\ -0.1667 & -0.6667 & -0.1667 \end{pmatrix}$$

4.2.1 Transformaciones geométricas

A menudo, para el análisis de imágenes, queremos investigar más específicamente un área dentro de una imagen, llamada Región de Interés (RDI). Para hacer esto necesitamos operaciones que modifiquen las coordenadas espaciales de la imagen, las cuales se denominan operaciones geométricas. El objetivo fundamental de una operación geométrica es la transformación de los valores de una imagen tal y como podría observarse desde otro punto de vista. Así las operaciones de modificar o reducir una imagen, no es sino aproximar o alejar el punto de vista, rotarla equivale a girar el punto de observación, trasladarla es hacer lo propio con dicho punto.

El paso previo a toda operación geométrica comienza por observar la distribución espacial de los píxeles en la imagen original y en la transformada. Generalmente, en la

imagen original asumirán un estructura matricial (tal y como están ordenados en la memoria) de la forma que se observa en la figura 4.3.

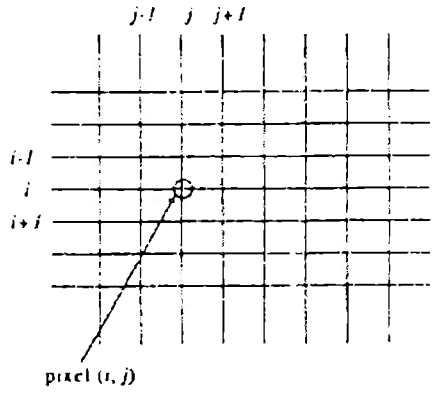


Figura 4.2 Disposición en forma de rejilla de los píxeles de una imagen

Puesto que la imagen digital es discreta, no existen valores de intensidad entre los valores discretos de las coordenadas "*x*" y "*y*", que coinciden con las intersecciones de los valores discretos horizontales y verticales. Al transformar esta rejilla según un desplazamiento, un giro o un acercamiento, los nuevos píxeles ya no tienen por qué quedar situados sobre las intersecciones y caerán, en general, sobre puntos intermedios de ellos. Al tener que proyectar estos píxeles sobre los de la imagen final, que deben asumir una estructura similar a las de la imagen original, es necesario pasar de las coordenadas pseudodiscretas a las discretas definitivas, y en concreto, es preciso calcular los valores de los píxeles finales en función de los transformados.

En la figura 4.4 se muestra una transformación cualquiera de rotación. La rejilla continúa muestra la disposición convencional de píxeles de la imagen original, mientras que la rejilla discontinua ilustra cómo queda la rejilla original después de girarla un ángulo dado alrededor de un punto determinado.

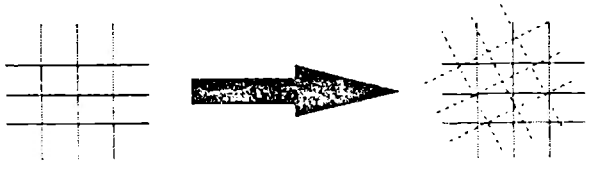


Figura 4.3 Ante cualquier transformación geométrica, los píxeles de la rejilla transformada (líneas discontinuas) no tienen por qué coincidir con los de la rejilla destino.

Por consiguiente, es preciso:

- Determinar las coordenadas de cada píxel (*i,j*) en la rejilla transformada (en líneas discontinuas). En general, los píxeles (*i´* y *j´*) obtenidos tras la transformación, no serán valores enteros y por tanto, no coincidirán con píxeles de la rejilla destino.
- Calcular los valores de los píxeles (*x,y*) en la rejilla destino a partir de los valores conocidos de píxeles (*i´* y *j´*) entre ellos.

El primer paso depende de la transformación depende de la transformación a realizar, mientras que el segundo se corresponde con una operación de interpolación.

4.2.2 Interpolación

La interpolación puede considerarse como el cálculo del valor de intensidad de un píxel, en una posición cualquiera, como una función de los píxeles que lo rodean (y que en nuestro caso ocuparán las posiciones enteras de las rejillas destino). Supongamos que deseamos calcular el valor del píxel de coordenadas (x,y) dado en la figura 4.5, en función de los valores de los píxeles de la rejilla.

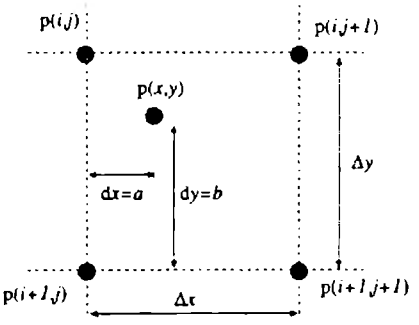


Figura 4.4 Esquema gráfico de la interpolación bilineal

Una forma de calcular este valor es suponer que cada píxel toma el mismo valor que el más cercano de entre los píxeles que los rodean. Para decidir cuál es el más cercano se puede utilizar la distancia Euclídea. Otra posibilidad es asociarle la intensidad media asociada a los dos píxeles más cercanos, uno en x y el otro en y.

4.2.3 Interpolación bilineal

Otra forma de interpolar con mejores resultados pero con mayor coste computacional es la interpolación bilineal. La cual asigna al píxel en cuestión un valor medio ponderado de las intensidades de los píxeles que lo rodean. Los factores de ponderación vienen dados por la distancia entre el píxel y los del entorno. Los factores de ponderación se calculan de la manera siguiente.

$$a_1 = \left(1 - \frac{dx}{\Delta x}\right)\left(1 - \frac{dy}{\Delta y}\right)$$

$$a_2 = \frac{dx}{\Delta x}\left(1 - \frac{dy}{\Delta y}\right)$$

$$a_3 = \left(1 - \frac{dx}{\Delta x}\right)\frac{dy}{\Delta y}$$

$$a_4 = \frac{dx}{\Delta x}\,\frac{dy}{\Delta y}$$

Donde $0 \le dx \le 1, 0 \le dy \le 1$, $\Delta x = 1$ y $\Delta y = 1$, por lo que obtendríamos,

$$a_1 = (1 - dx)(1 - dy)$$

$$a_2 = dx(1 - dy)$$

$$a_3 = (1 - dx)dy$$

$$a_4 = dxdy$$

Finalmente, el valor del píxel interpolado, en función de los cuatro de su entorno, queda:

$$p(x,y) = a_1 p(i,j) + a_2 p(i,j+1) + a_3 p(i+1,j) + a_4 p(i+1,j+1)$$

5. Procesadores Digitales de Señales (DSPs)

5.1 Conceptos Generales

DSP es el acrónimo de *Digital Signal Processor*, que significa Procesador Digital de Señal, un nombre bastante descriptivo, pues su función no es otra sino recibir una señal como entrada, hacer unas operaciones sobre esa señal y sacar a su salida una nueva señal.

Un DSP es un sistema basado en un procesador o microprocesador que posee un juego de instrucciones, un hardware y un software optimizados para aplicaciones que requieran operaciones numéricas a muy alta velocidad. Debido a esto es especialmente útil para el procesado y representación de señales analógicas en tiempo real: en un sistema que trabaje de esta forma (tiempo real) se reciben muestras, normalmente provenientes de un conversor analógico/digital (ADC), el sistema debe hacer todas las operaciones con las muestras recibidas antes de que llegue el siguiente.

Se ha dicho que puede trabajar con señales analógicas, pero es un sistema digital, por lo tanto necesitará unos conversores analógicos/digitales a sus entradas y salidas. Como todo sistema basado en procesador programable necesitará una memoria donde almacenar los datos con los que trabajará y el programa que ejecutará. Si se combina que un DSP puede trabajar con varios datos en paralelo y un diseño e instrucciones específicas para el procesado digital, se puede dar una idea de su enorme potencia para este tipo de aplicaciones. Estas características constituyen la principal diferencia de un DSP y otros tipos de procesadores.

5.2 Procesamiento Digital vs Analógico

Los procesadores digitales de señales (DSPs) han llegado a ser el fundamento de la llamada Revolución Digital. En la actualidad los procesadores digitales de señales se encuentran en los celulares, en los reproductores de audio y video, cámaras digitales, infraestructura de telefonía, en los sistemas de control para motores, por mencionar algunas aplicaciones. Cada una de estas aplicaciones trabaja con una señal o con una cadena de datos.

Décadas atrás, los desarrolladores descubrieron que procesar esas señales en formato digital significaba tener varias ventajas sobre las señales analógicas:

- Las señales digitales pueden ser mandadas a grandes distancias sin perder información como lo hacen las señales analógicas.
- Para filtrar y limpiar señales análogas se requieren diversos componentes que pueden elevar los costos, mientras que para filtrar una señal digital se implementa un código flexible.
- Copias perfectas de señales digitales como contenido de video y audio son posibles, haciendo que cada copia sea tan buena como la original, comparado con copias análogas que degradan cada copia generada.
- Mejorar la calidad de la señal o la densidad del canal es una cuestión de incrementar el rendimiento del DSP.

- Debido a que los DSP's son programables, permiten a los usuarios constantemente mejorar la calidad sin tener que rediseñar un hardware.

5.3 Medición del Rendimiento

En los primeros días de los procesadores, el rendimiento era medido por el número de instrucciones que un procesador podía ejecutar en un segundo. Al basarse los procesadores en la misma tecnología de antes, esto sirvió para poder comparar el rendimiento de diferentes procesadores.

Los procesadores de hoy en día son muy diferentes a sus antecesores. Los primeros procesadores tomaban varios ciclos para ejecutar una operación simple como sumar o almacenar algún dato en memoria. El rendimiento era medido en miles de instrucciones por segundo.

Los procesadores modernos alcanzan un alto rendimiento a través de tecnologías como la de procesamiento en paralelo, circuitos internos especializados para determinadas funciones y periféricos integrados, por lo que el rendimiento de estos procesadores se mide en millones o billones de operaciones por segundo.

5.4 Ventajas de los DSPs

- Los DSPs son optimizados para aplicaciones de procesamientos de señales comparados con los procesadores los procesadores de propósito general (GPP).
- Los DSPs ofrecen varias características en sus arquitecturas que actualmente reducen el número de instrucciones necesarias para eficientar el proceso de una señal. En otras palabras, el rendimiento de un procesador en mucho más que contar instrucciones ejecutadas en un segundo
- La flexibilidad en programar los DSPs permiten a los usuarios implementar en software algoritmos complejos. Un DSP no sólo puede soportar código de video como MPEG-2 y manejar fácilmente diferentes resoluciones con una actualización de software, sino que puede implementar códigos emergentes y estándares como sean necesarios sin tener que rediseñar el hardware
- Las ventajas de digitalizar son claras, se tiene un alto rendimiento, es por eso que el procesamiento digital de señales han penetrado campos como las multimedia y aplicaciones de procesamiento de voz. Los DSP's por ejemplo han incrementado la eficiencia de motores y sistemas de control de potencia simplificando el diseño, extendiendo la funcionalidad y reduciendo costos.

6. Field Programmable Gate Arrays (FPGAs)

6.1 Fundamentos Teóricos

Un FPGA (Field Programmable Gate Array - Arreglo de Compuertas Programable en Campo) se compone de elementos con recursos no comprometidos que pueden ser seleccionados, configurados e interconectados por usuario. Recordemos que en un PLD las interconexiones entre los elementos ya están hechas, solamente podemos habilitar o deshabilitar la interconexión; en el caso de un FPGA no hay nada interconectado.

Estos dispositivos se componen de cierto número de Módulos Lógicos, que determinan la capacidad del dispositivo. Los módulos son independientes entre sí y pueden interconectarse para formar un modulo más complejo. Dependiendo del fabricante, estos módulos pueden ser Bloques Configurables, como en los FPGA's de

Xilinx, o bien, Elementos de Función Fija formados por arreglos de compuertas, como en el caso de los dispositivos de Actel.

Los módulos en un FPGA, se interconectan por medio de Canales Configurables (figura 8.1). Al proceso de interconexión, se le conoce como Enrutamiento y consiste en determinar la mejor estrategia de interconectar los módulos, ya sea en forma manual o mediante alguna herramienta de diseño electrónico (EDA).

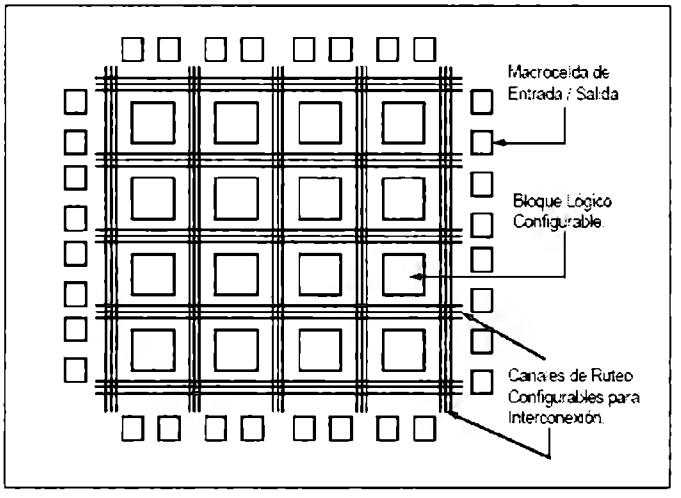


Figura 6.1 Arquitectura demostrativa, no detallada, de un FPGA de Xilinx.

Por la capacidad de un FPGA (el más sencillo contiene 1,200, hasta los más grandes que contienen 4'000,000 de compuertas lógicas), se dice que son dispositivos para diseños LSI (Large Scale Integration) y VLSI (Very Large Scale Integration).

La gran ventaja de utilizar estos dispositivos radica en que todo el desarrollo se lleva a cabo en un solo ambiente de trabajo. El diseñador propone la función lógica a realizar y en base a métodos de descripción define los parámetros de su problema.

Esto se hace por medio de código programable, que puede ser un Lenguaje de Descripción de Hardware, o bien, un diagrama esquemático de conexiones.

Una vez delimitado el problema, se optimiza su representación lógica mediante métodos de minimización (Síntesis Lógica); posteriormente se simula, lógica y eléctricamente (Simulación). Con la ayuda del software actual que es muy sofisticado, este último paso puede llegar a ser muy aproximado al comportamiento real del dispositivo.

Se selecciona, entonces, el dispositivo que mejor se adapte a las condiciones de nuestro problema según criterios de capacidad, velocidad, consumo de energía, costo, etc., y finalmente se programa en campo.

6.2 Ventajas del diseño con FPGAs

- Minimización del número de componentes en un diseño. Con esto se reducen los gastos de inventario, inspección y prueba, así como el número de fallas a nivel circuito impreso, propiciando un ahorro de espacio físico. Así, una medida de la eficiencia de un Dispositivo Programable se expresa mediante el número de dispositivos de función fija (Circuitos Integrados de Catálogo) que pueden remplazarse.

- Reducción en el tiempo de diseño. Debido a su naturaleza programable, reducen el tiempo y los costos de desarrollo, no sólo de nuevos productos sino también de aquellos que requieren modificaciones (reingeniería), ya que son reutilizables tantas veces como sea necesario. Esto último se debe a que los cambios en el diseño son realizados mediante una nueva programación que se prueba inmediatamente si se está utilizando un dispositivo programable en el mismo circuito (IS).

- Uso de una gran variedad de herramientas de Diseño Asistido por Computadora (CAD), disponibles actualmente en el mercado. Estas herramientas promueven y facilitan el diseño sobre este tipo de dispositivos. Así mismo, no se requiere de grandes recursos de cómputo.

6.3 FPGAs de Granularidad Gruesa y Granularidad Fina

La complejidad de los elementos contenidos en los módulos lógicos, es factor determinante para medir el desempeño de un FPGA.

La complejidad de los elementos contenidos en los módulos lógicos, es factor determinante para medir el desempeño de un FPGA. La arquitectura avanzada, llamada así por la densidad de sus componentes y sus estrategias de interconexión entre módulos, tiene dos derivaciones estructurales de acuerdo al tipo de módulos lógicos que la conforman: Granularidad Gruesa y Granularidad Fina.

Los módulos lógicos en una arquitectura de Granularidad Gruesa (Coarse Grained – CG), son módulos grandes generalmente consistentes de una o más Tablas de Búsqueda y dos o más flip – flop's. Si analizamos la configuración física de cada módulo lógico o Grano, podemos advertir que particularmente cada uno de ellos, puede ejecutar una función simple o una función compleja, que adicionada a otra función ejecutada por un módulo diferente conforman un sistema más complejo. La Tabla de Búsqueda (LookUp Table - LUT) actúa como una memoria donde se encuentra almacenada la tabla de verdad que representa la función lógica del circuito, así en una LUT es posible implementar cualquier función deseable. Por lo anterior se dice que un módulo que contiene elementos como éstos, es un Grano Grueso.

Por otra parte, una arquitectura de Granularidad Fina (Fine Grained – FG), está estructurada por una gran cantidad de módulos lógicos pequeños que realizan funciones relativamente simples. Cada Grano o módulo en este tipo de arquitectura está compuesto de un circuito de dos entradas que realiza una función lógica determinada, o en algunos otros casos por un multiplexor 4 a 1. Adicionalmente contienen un solo flip – flop.

Como hemos de suponer, entre ambas granularidades existen diferencias. La CG permite implementaciones menos detalladas debido a que desde un nivel muy básico se tienen módulos complejos. Sin embargo, son dispositivos con una gran densidad de compuertas, ya que el hecho de utilizar LUT deja entrever que se pueden realizar diseños grandes. Los FPGAs con tecnología SRAM, como los de Xilinx o los de Altera, tienen arquitecturas CG, y son ISP (Programables en Sistema).

La arquitectura FG, como la de los FPGAs de Actel, está relacionada con la tecnología de programación Antifuses. La simpleza de la constitución de cada módulo, permite implementaciones más detalladas y sobre todo más veloces. Debido a que para llegar a implementar una función compleja se requiere el uso de varios módulos, se trata de dispositivos de alta densidad de módulos (comparándolos con los anteriores), sien embargo, realmente son FPGAs con una gran cantidad de módulos (por su tamaño),

pero cada módulo tiene un número mínimo de compuertas lógicas a diferencia de los FPGAs CG, que pueden tener menor número de módulos pero cada módulo tiene un número grande de compuertas lógicas. La apreciación puede ser engañosa. Los FPGAs con tecnología Antifuses como los de la arquitectura FG, son Programables Fuera del Sistema (OSP).

Los FPGAs de Xilinx y los de Altera son MTP, programándose mediante tecnología SRAM, por lo que pueden ser Programados en Sistema, a diferencia de los OTP que se programan Fuera de Sistema. Debido a que los SRAM necesitan almacenar su configuración en RAM, son módulos volátiles y requieren en ocasiones una memoria exterior para hacerlo, lo que implica mayor hardware de soporte. Así mismo, se trata de dispositivos más lentos en comparación a los OTP, pero con la gran ventaja de que son completamente Reconfigurables y tienen una mayor capacidad de compuertas lógicas.

6.4 FPGAs de Xilinx

Xilinx está considerado como uno de los fabricantes más fuertes a nivel mundial. Sus FPGAs (también fabrica PLDs y CPLDs) están basados en la tecnología SRAM y son dispositivos MTP, programables en sistema (IS).

Sus principales familias de FPGAs son: XC3000, XC4000, XC Virtex, y XC Spartan. La estructura de estos dispositivos está compuesta por módulos lógicos llamados por Xilinx, CLBs (Configurable Logic Blocks – Bloques Lógicos Configurables), basados en Tablas de Búsqueda (LookUp Tables - LUTs). Cada CLB contiene circuitos que les permiten realizar operaciones aritméticas eficientes (como la de un algoritmo de Suma Paralela).

También los usuarios pueden configurar las tablas de búsqueda como celdas "read/write" (lectura/escritura)de RAM. A la vez, a partir de la serie XC 4000, se incluye un generador interno de señal de reloj con 5 diferentes frecuencias.

Además de los CLBs, los FPGA de este fabricante incluyen otros bloques complejos que configuran la entrada de los pines físicos que conectan el interior del dispositivo con el exterior, a estos bloques se les llama IOBs (Input/Output Blocks – Bloques de Entrada/Salida). Cada IOB contiene una lógica compleja que permite que un pin pueda actuar como entrada, salida o un tercer estado.

La figura 8.2 muestra a detalle, la arquitectura del XC4003E de Xilinx. Nótese la complejidad de un CLB, así como la disposición de los IOBs alrededor del dispositivo, sirviendo como interfases entre el FPGA y el mundo exterior

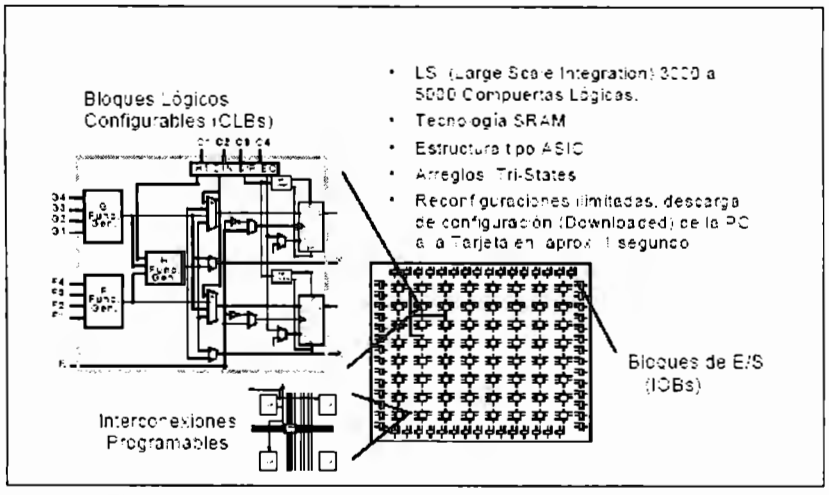


Figura 6.2 Arquitectura de un FPGA XC4003E de Xilinx.

La serie Spartan surgió como una opción para sustituir diseños probados de menos de 15,000 compuertas por dispositivos de bajo costo y alto desempeño (además incluyen el soporte de los CORES prediseñados), pero sacrificando algunas características que manejan las series estándar de Xilinx, como la XC4000 tradicional o la serie Virtex. Las series estándar XC3000 y su sucesora, la 4000, son series que muestran la mayor parte de las características funcionales de los FPGAs de Xilinx, pero con la gran desventaja que son dispositivos de baja densidad de compuertas. Sin embargo, el uso de la serie XC4000 (en especial el XC40003E) es muy favorecido para el diseño e implementación de prototipos de bajo impacto.

III. DESARROLLO Y RESULTADOS

1. Interfaz RS232 – I^2C

La primera etapa de trabajo del proyecto consistió el desarrollo de la interfaz RS232 – I^2C. Para esto, lo primero que hicimos fue una investigación sobre los recursos y dispositivos que se necesitarían para llevar a cabo el diseño e implementación de la interfaz. Esta investigación fue básicamente sobre el funcionamiento y configuración del puerto RS232 de la computadora, así como el envío y recepción de datos a través de este puerto serial. En la figura 1.1 se muestra la configuración de pines del conector estándar RS232 que se encuentra en las computadoras personales con puerto serie.

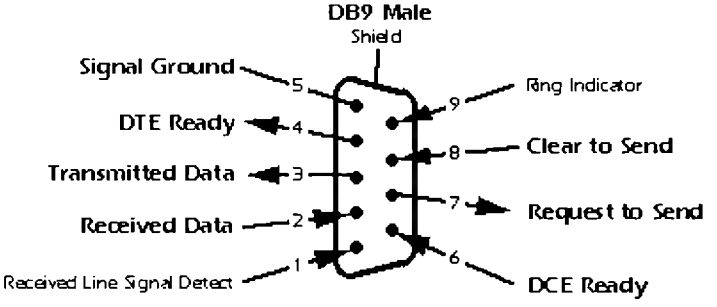


Figura 1.1. Configuración del conector RS232

Para la transmisión y recepción de datos, lo que hicimos fue utilizar el software XTALK, el cual simplemente envía una cadena de pulsos correspondientes al código ASCII del carácter que se esté tecleando. Cabe mencionar, que con el XTALK también se pueden controlar algunos parámetros de la configuración del puerto serie de la computadora, como lo son la tasa de transmisión (en bits / seg), el puerto que se está utilizando, etc. Esto es algo importante, ya que nos permite tener un mayor control sobre lo que queremos hacer con el puerto y de esta manera entender mejor su funcionamiento.

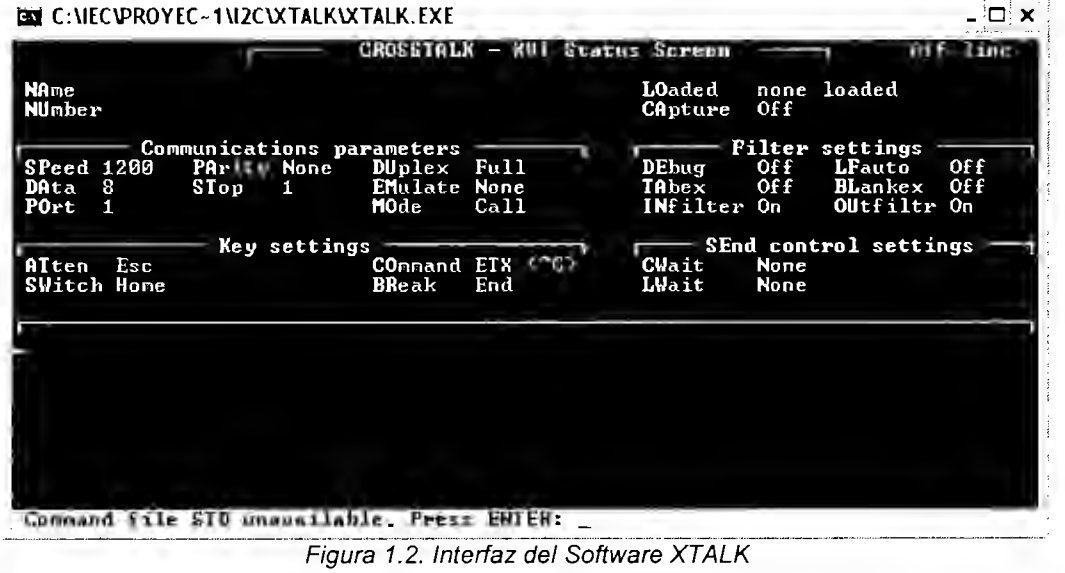


Figura 1.2. Interfaz del Software XTALK

Para visualizar esto, utilizamos el osciloscopio, con el cual pudimos observar los trenes de impulsos que se generaban cuando se enviaban los datos por el puerto. En

la figura 1.3 se muestra los trenes de pulsos generados con diferentes señales enviadas a través del puerto.

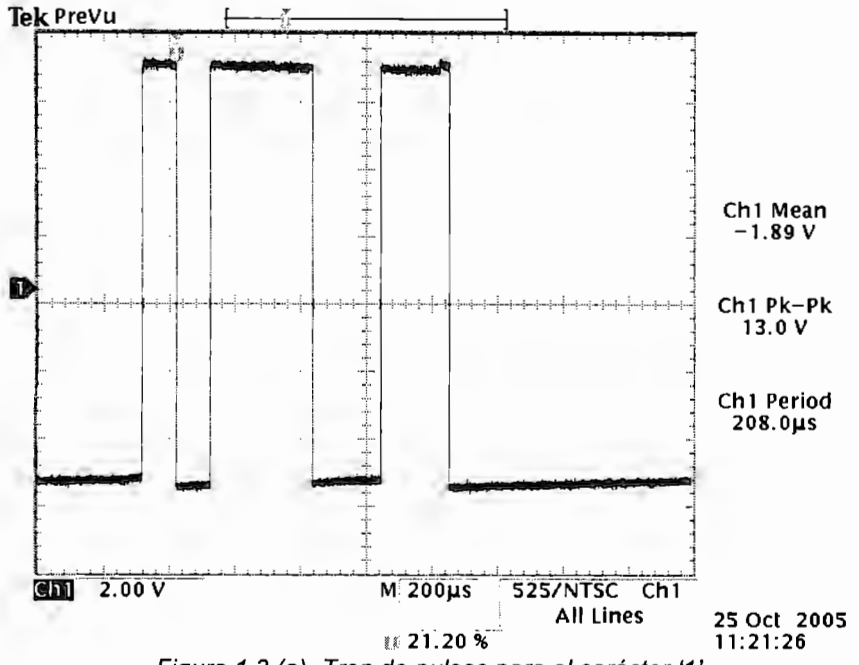


Figura 1.3 (a). Tren de pulsos para el carácter '1'

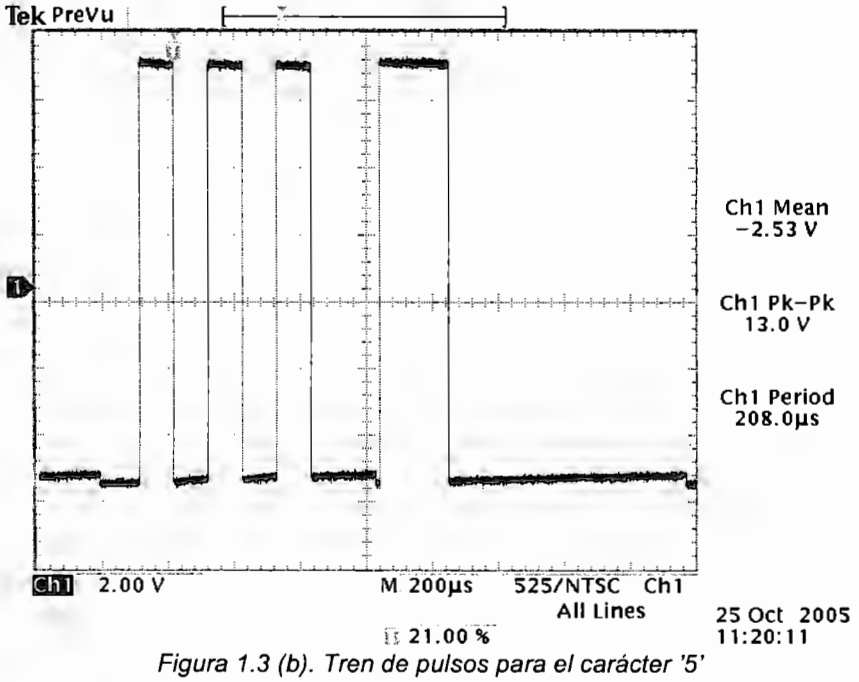


Figura 1.3 (b). Tren de pulsos para el carácter '5'

Una vez que logramos entender el funcionamiento del puerto RS232, desarrollamos un programa en Borland C++ Builder con el cual pudiéramos enviar y recibir datos a la computadora, a través del puerto RS232.

En una primera etapa, este programa consistía simplemente de una ventana de texto, en la cual, cada carácter que fuera introducido desde el teclado era enviado a través del puerto serial hacia otra computadora. Si este programa se ejecutaba en ambas computadoras al mismo tiempo, las dos computadoras podían enviar y recibir

caracteres a través de los puertos seriales y los caracteres que fueran escritos en una computadora aparecerían en la ventana de texto de la otra. Es decir, se tenía comunicación bidireccional de tipo half duplex entre ambas computadoras, ya que existía comunicación en ambos sentidos, pero no al mismo tiempo, sino que hasta que una terminara de transmitir, la otra podía comenzar a hacerlo.

Una vez que se tuvo esta comunicación bidireccional, pudimos desarrollar el programa que utilizaríamos en nuestra interfaz RS232 – I^2C. El desarrollo de este nuevo sistema simplemente consistió en agregar al sistema de envío y recepción de caracteres, los campos de texto con sus respectivos botones para cada uno de los parámetros de imagen que se deseaban controlar y que eran los siguientes: Brillo, Contraste, Tipo de Video, Frame Rate, Número de Bits, Resolución, Configuración de Fábrica y Estado.

La interfaz gráfica de este nuevo sistema es la siguiente:

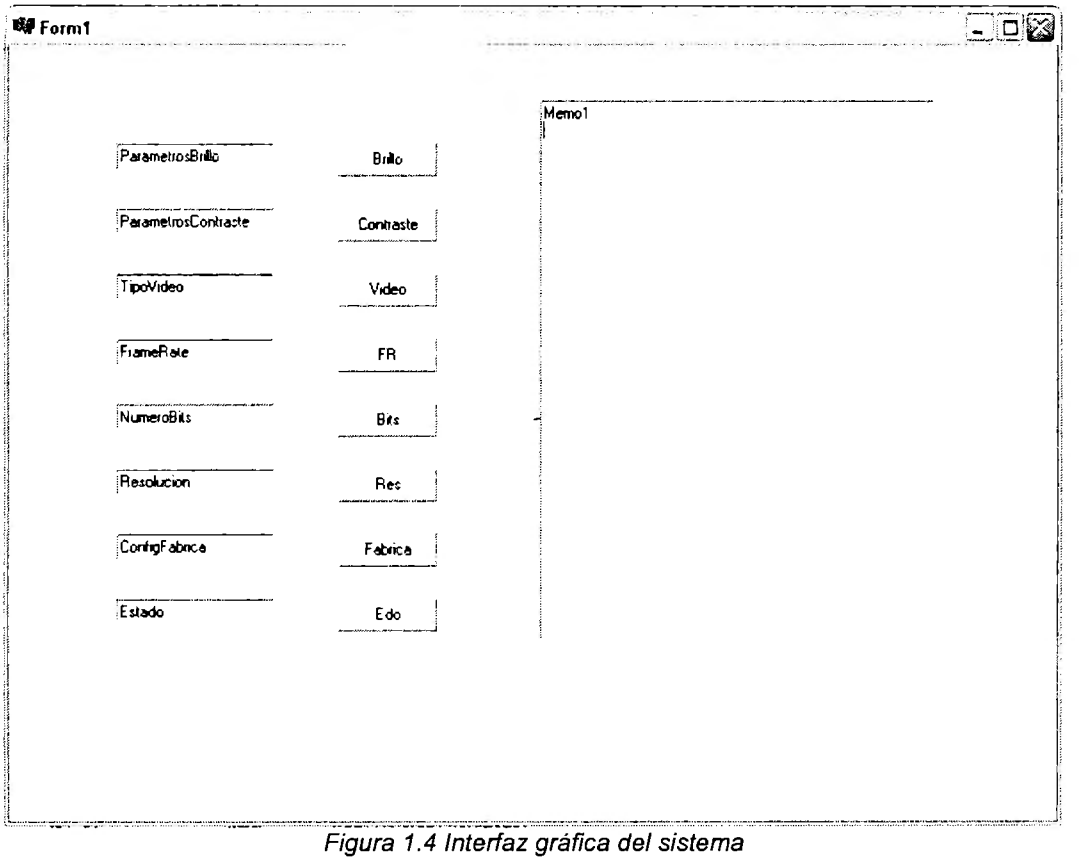


Figura 1.4 Interfaz gráfica del sistema

*El código de este programa se presenta en el Anexo A.

Terminado el desarrollo de esta interfaz gráfica, lo siguiente que debíamos hacer era comenzar con la programación del PIC 16F877, con el cual se haría la conversión de las instrucciones, de formato serial RS232 a formato I^2C. Para esto, lo primero que hicimos fue desarrollar dos programas en lenguaje C, uno que controlara las funciones del dispositivo maestro, que en nuestro caso sería el PIC, y otro que controlara al dispositivo esclavo, que sería el DSP.

Como una primera aproximación, lo que se hizo fue desarrollar, en conjunto con el equipo del proyecto de Captura y Digitalización de Video, un sistema que permitiera desplegar caracteres en una matriz de LEDs. Lo que hacía este programa era enviar el

carácter deseado desde la computadora, por puerto serial y en una primera tarjeta (dispositivo maestro), se convertía a bus I^2C con un PIC 16F877. Este PIC enviaba la información a una segunda tarjeta (dispositivo esclavo) que era la encargada de manejar la matriz de LEDs. Es importante señalar que, en este punto, el sistema tenía la capacidad para convertir de formato RS232 a I^2C y transmitir los datos del maestro al esclavo por medio de este bus, sin embargo, aún no contaba con la capacidad de leer datos del dispositivo esclavo para transmitirlos hacia el maestro, lo cual era un requisito del sistema, ya que necesitábamos poder recuperar el estado del DSP en cualquier momento.

*El código de los programas para maestro y esclavo se presenta en el anexo B.

Al comprobar que el sistema era capaz de realizar la conversión entre formatos y controlar al dispositivo esclavo, nos encontrábamos listos para adecuarlo a nuestro dispositivo esclavo, el DSP C6416. Una vez que esta parte estuviera lista, el paso siguiente sería implementar la parte de lectura y recepción de datos desde el dispositivo esclavo.

En este punto, nos encontramos con una de las mayores dificultades de integración de nuestro proyecto, ya que, después investigar y realizar varias pruebas, descubrimos que el DSP C6416 no contaba con un puerto I^2C. Esto detuvo nuestro avance de manera considerable, ya que la integración de la interfaz que habíamos desarrollado iba a ser mucho más complicada de lo que se esperaba, puesto que había que desarrollar una nueva interfaz, que permitiera al DSP recibir datos vía bus I^2C. Después de aproximadamente dos semanas de buscar alternativas y posibilidades en conjunto con el personal de Kokai, se llegó a la conclusión de que el desarrollo de esta nueva interfaz se encontraba fuera de los alcances del proyecto y por cuestiones de tiempo, quedó descartada. Por este motivo, a partir de este momento dejamos de lado todo lo relacionado con el desarrollo de la interfaz RS232 – I^2C, ya que finalmente no se iba a poder integrar con el proyecto y necesitábamos seguir adelante.

Una vez que se dio por concluida la etapa de la interfaz RS232 - I^2C, el paso siguiente consistía en enfocarnos de lleno al procesamiento digital de imágenes, para poder implementar los algoritmos de redimensionamiento en el DSP, que era el objetivo central de nuestro proyecto.

2. Procesamiento Digital de Imágenes en Matlab

Al realizar nuestra investigación y marco teórico, nos dimos cuenta de que el primer paso en el procesamiento de video digital es el procesamiento de imágenes fijas. Debido a que el video no es más que una secuencia de imágenes a una velocidad tal que el ojo humano no perciba los cambios entre imágenes, los mecanismos para procesar video son exactamente los mismos que se utilizan para procesar imágenes. Esto es, para procesar video, a cada una de las imágenes (o cuadros) de una secuencia de video, se le aplican los mismos algoritmos y funciones que se aplican a una imagen fija.

Al principio del proyecto se nos proporcionaron 2 algoritmos de redimensionamiento de imágenes; el de Interpolación Bilineal y el del Vecino más Cercano, estos dos ya implementados en Matlab. Al simular ambos algoritmos, nos dimos cuenta de que la complejidad de ambos era muy similar y sin embargo, los resultados obtenidos con el algoritmo de Interpolación Bilineal eran considerablemente mejores que los del Vecino más Cercano. Por este motivo, desde un principio decidimos enfocarnos al algoritmo de Interpolación Bilineal y dejar de lado el del Vecino más Cercano.

Al ver que el propósito general de los algoritmos era remover píxeles, decidimos desarrollar un algoritmo que fuera mucho más sencillo, el cual solo eliminara píxeles de una imagen original, para compararlo con los algoritmos que se nos proporcionaron. El motivo por el cual desarrollamos este nuevo algoritmo, fue que, si los resultados lo permitían, el trabajar con un algoritmo mucho más corto y sencillo nos facilitaría considerablemente la implementación del mismo.

Al comparar los resultados de los algoritmos, pudimos ver que la imagen redimensionada con los algoritmos proporcionados tiene ligeramente mejor calidad que la imagen a la cual solo se le quitan los píxeles. Por este motivo decidimos llevar el proyecto con el algoritmo que desarrollamos nosotros ya que es mucho más corto y sencillo de implementar en el DSP que los algoritmos que se nos proporcionaron.

2.1 Interpolación bilineal (algoritmo proporcionado por Kokai Electrónica)

Este algoritmo asigna al píxel en cuestión un valor medio ponderado de las intensidades de los píxeles que lo rodean. Los factores de ponderación vienen dados por la distancia entre el píxel y los del entorno.
El código se presenta en el anexo C.

Los resultados de la imagen redimensionada con el algoritmo de interpolación bilineal, de 768x1024 a 384x512 se muestran en la figura 2.2

2.2 Eliminación de píxeles (algoritmo desarrollado por el equipo de trabajo)

Lo que hace este algoritmo es tomar la medida original de la matriz, que deberá ser de MxNx3, posteriormente entra a una lógica en donde se eliminaran n píxeles del vector M y posteriormente se eliminaran n píxeles del vector N. La forma de descartar los bits es intercalada y dependiendo del numero n que se ponga en el programa. La matriz que tiene el valor de 3 se deja exactamente igual.

El algoritmo tiene el nombre de Half. El código se presenta en el anexo D.

Los resultados de la imagen redimensionada con el algoritmo de eliminación de bits, de 1024x768 a 512x384 se muestran en la figura 2.3.

Fig. 2.1 Imagen original (1024x768 píxeles)

Fig. 2.2 Imagen redimensionada con interpolación bilineal (512x384)

Fig. 2.3 Imagen redimensionada con el método Half (512x384)

En el primer algoritmo se hace una interpolación bilineal entre píxeles, por lo que podemos ver que la calidad de la imagen (Fig 2.2) es relativamente buena, en comparación con el método de Half ya que los bordes están más suavizados, así como los detalles de la imagen.

La diferencia entre la imagen original y la de la figura 2.3 es principalmente la calidad, ya que el algoritmo Half al no hacer un promedio y solo eliminar píxeles se pierden muchos detalles y los bordes no están suavizados.

Además de utilizar la imagen de Angelina Jolie, se aplicaron los algoritmos de Half y de interpolación bilineal a una textura de *Brodatz*, que son imágenes diseñadas específicamente para ser procesadas y poder observar los resultados del procesamiento de manera clara y estandarizada. Las imágenes originales son de 640x640 píxeles, las imágenes redimensionadas por ambos métodos son de 320x 320 píxeles.

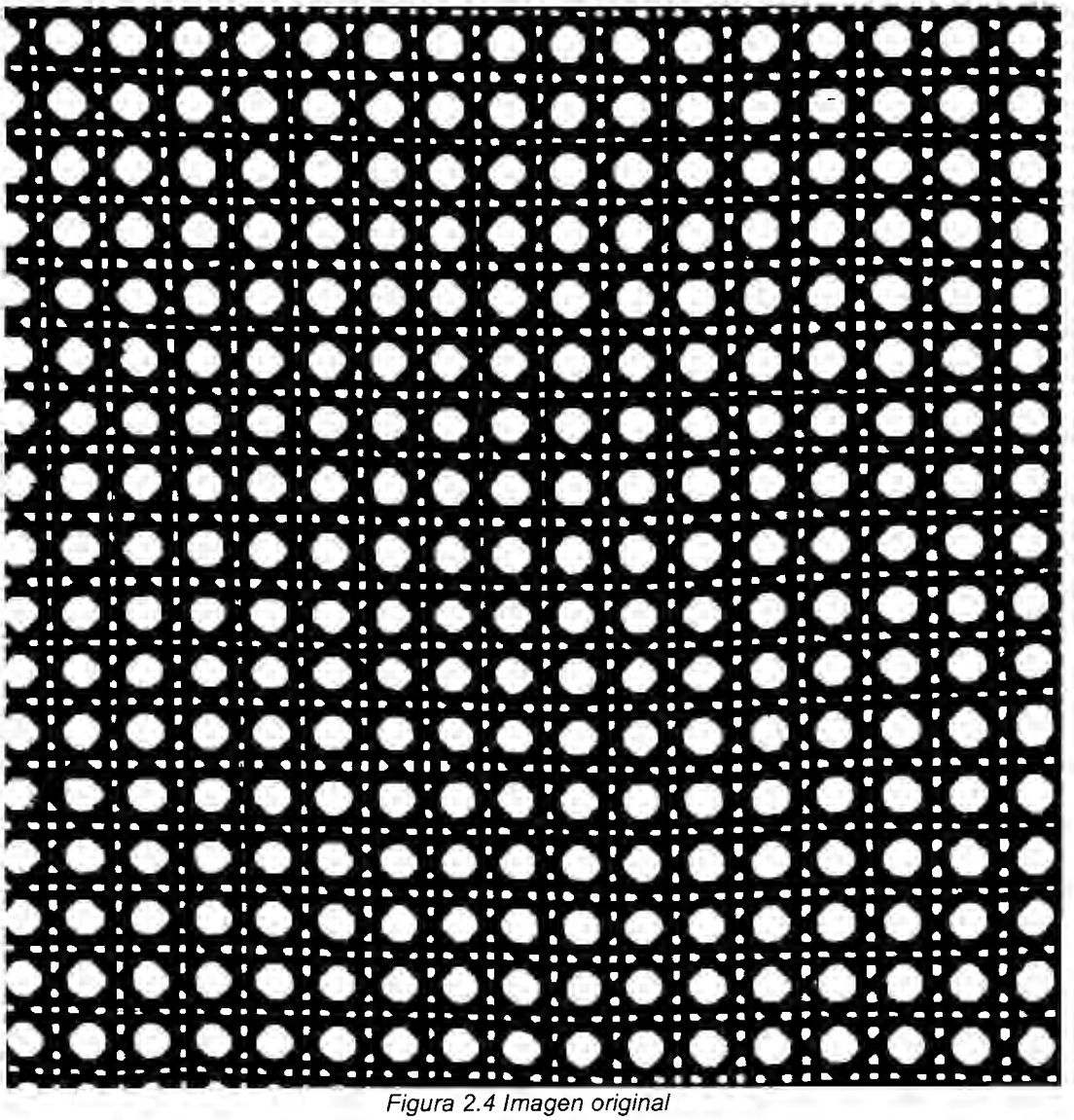


Figura 2.4 Imagen original

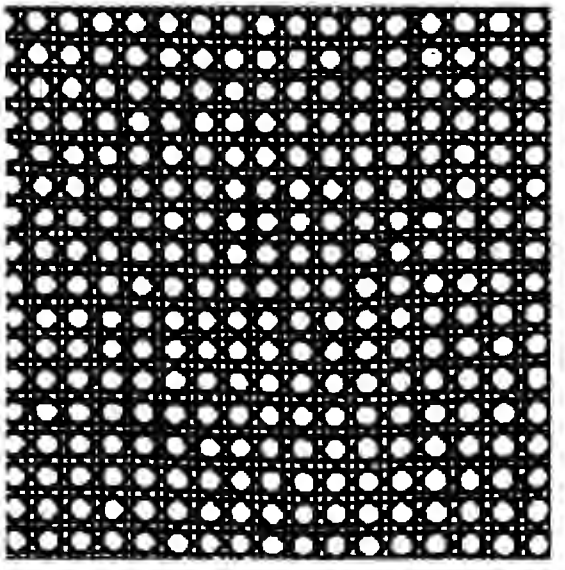


Figura 2.5 Imagen redimensionada con algoritmo Bilinieal

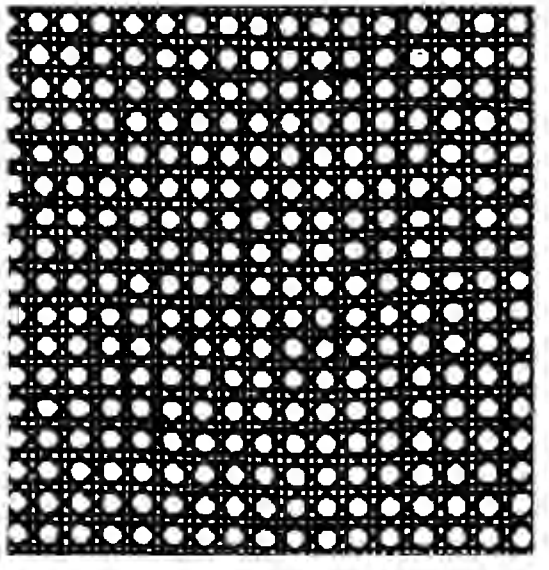


Figura 2.6 Imagen redimensionada con algoritmo Half

Como se puede observar, la calidad de las imágenes al ser redimensionadas ya sea removiendo píxeles o haciendo un promedio entre sus píxeles, es bastante aceptable y no existe una diferencia considerable entre ambos métodos. Únicamente si una imagen redimensionada se amplia al tamaño original de esta misma, se puede observar la diferencia.

2.3 Separación de una imagen en sus componentes R, G y B.

Después de realizar la simulación de los algoritmos y tomar la decisión de trabajar sobre el método Half, el siguiente paso era separar una imagen en sus componentes R,G y B, ya que el procesamiento de video se iba a hacer en formato RGB, sobre cada uno de los tres canales por separado.

Los algoritmos que desarrollamos para separar una imagen en sus componentes RGB están basados en el algoritmo Half, pero en vez tener como resultado los tres arreglos (RGB) en la misma imagen, ahora los tendremos separados. Cabe destacar que, como están basados en el algoritmo de Half, estos tres algoritmos, aparte de separar la imagen en sus componentes R, G y B, también hacen la función de redimensionar la imagen.

En el anexo E se muestran los algoritmos para obtener cada una de las tres componentes de una imagen por separado.

2.3.1 Componente R de una imagen

Figura 3.1 Componente rojo (R) de la imagen

2.3.2 Componente G de una imagen

Figura 3.2 Componente verde (G) de la imagen

2.3.3 Componente B de una imagen

Figura 3.3 Componente azul (B) de la imagen

Estas tres imágenes son las componentes R, G y B de la imagen original y también fueron redimensionadas por los algoritmos. Ahora, el siguiente paso, para poder pasar nuestras ideas al DSP es pasar estas imágenes a archivos de tipo hexadecimal.

*2.3.4 Algoritmo para pasar los arreglos R, G y B a un archivo *.dat*

En la interfaz grafica del DSP para procesamiento de imágenes se piden tres buffers de entrada, uno para la componente R, otro para la componente G y un último para la componente B, en tipo de archivo hexadecimal *.dat. Esto es, una sola columna de información de la imagen en formato hexadecimal, precedida por un encabezado.

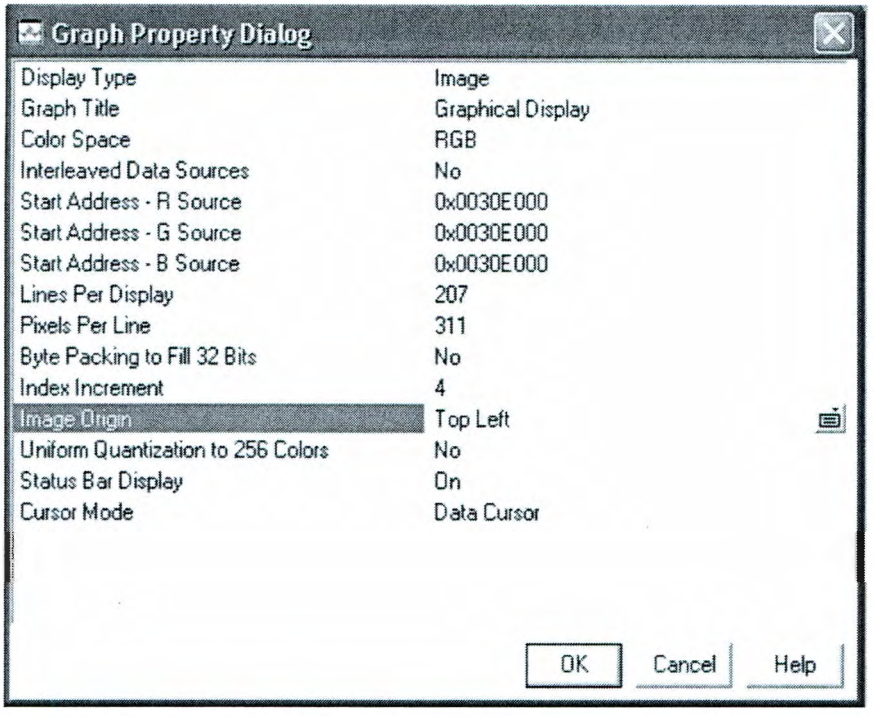


Figura 4.1 Interfaz de imagen del CCS:

También se tienen ahora 3 algoritmos diferentes, uno para cada arreglo de color. En el anexo F se presenta el código.

Nota: los archivos hexadecimales que despliegan estos algoritmos son de aproximadamente 3328 datos, ya que la imagen con la que se trabaja en el DSP es de 52x64 píxeles.

3. Procesamiento Digital de Imágenes en el DSP C6416

Una vez que logramos separar las imágenes en sus componentes RGB y hacer el redimensionamiento de cada una de las componentes, habíamos concluido con la etapa de simulación en Matlab y nos encontrábamos listos para implementar estos algoritmos en el DSP C6416.

El kit de desarrollo que utilizamos fue el TMS320C6416T DSK, de Spectrum Digital. A continuación, se presenta una descripción general del mismo:

Características Generales

EL C6416T DSK viene con un gran número de dispositivos que permiten desarrollar una gran variedad de ambientes de aplicación. Las características principales son las siguientes:

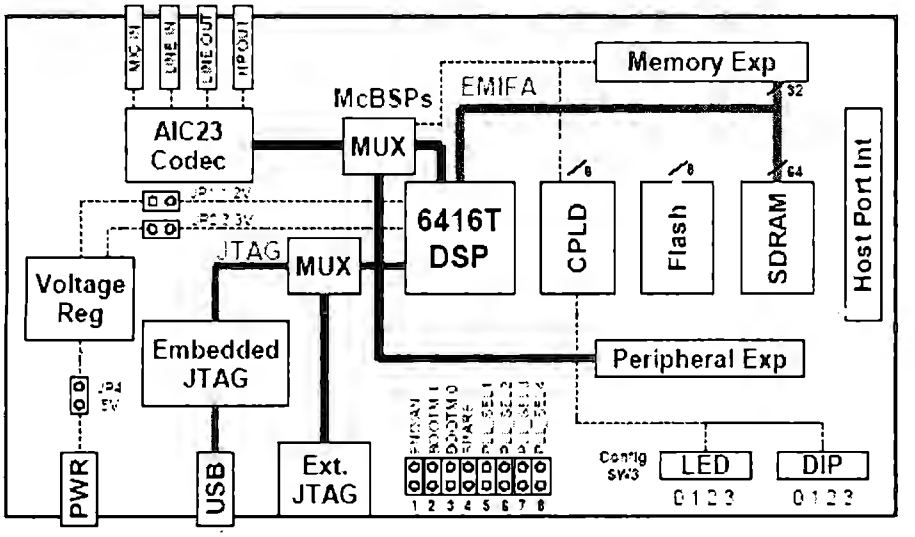


Figura 3(a) Diagrama a bloques del DSK C6416T

- EL DSP TMS320C6416T de Texas Instruments opera a 1Gigahert
- Un codificador estéreo AIC23
- 16 Mbytes de DRAM síncrona
- 512 Kbytes de memoria no volátil FLASH
- 4 LEDs y DIP switches accesibles por el usuario
- Configuración de la tarjeta por Software por medio de registros implementados en el CPLD
- Opciones configurables de "boot" y una entrada de selección de reloj
- Conectores Standard para el uso de una tarjeta hija
- Emulación JTAG por medio de un emulador JTAG ubicado en la tarjeta con interfase USB o algún emulador externo
- Una fuente sencilla de voltaje (+5V)

Panorama Funcional del TMS320C6416T DSK

El DSP en el 6416T DSK interacciona con los periféricos en la tarjeta por medio de uno de los dos buses, el EMIFA de 64 bits y el EMIFB de 8 bits. La SDRAM, Flash y el CPLD están cada uno conectados por uno de los buses. El EMIFA está también conectado a los conectores de la tarjeta hija de expansión que es usada para una adición extra de una tarjeta.

Un codificador AIC23 en la tarjeta permite al DSP transmitir y recibir señales analógicas. El McBSP1 es usado para la interfase de codificación de control y el McBSP2 es usado para los datos. Una I/O está compuesta por cuatro jacks de audio de 3.5mm que corresponden a una entrada de audifonos, línea de entrada, línea de salida y salida de audífonos. El codificador puede seleccionar el micrófono o la línea de entrada como entrada activa. La salida analógica es manejada por conectores de línea de salida (ganancia fija) y una de audífono (ganancia ajustable). El McBSP1 y McBSP2 pueden ser nuevamente enrutados por los conectores de expansión de software.

Un dispositivo lógico programable llamado CPLD es usado para implementar "blue logic" que une a los componentes de la tarjeta. El CPLD también tiene un registro basado en una interfase de usuario que permite al usuario configurar la tarjeta leyendo y escribiendo los registros del CPLD.

El DSK incluye 4 LEDs y 4 posiciones de DIP switch como una manera simple de proveer al usuario una retroalimentación interactiva, pero ambos son accesados escribiendo y leyendo los registros del CPLD.

Mapa de Memoria

La familia C64XX de DSPs tiene un byte de espacio de dirección. El código de programa y los datos pueden ser puestos en cualquier lugar del espacio de dirección unificado. Las direcciones siempre son de un ancho de 32 bits.

En la parte izquierda de la figura se muestran los espacios de dirección de un procesador genérico 6416T con detalles específicos de cómo cada región es usada, esto en la parte derecha. Por default, la memoria interna empieza al principio del espacio de direcciones. Algunas partes de la memoria pueden ser mapeados nuevamente por medio de software como L2 cache más que por RAM fija

Cada EMIF (External Memory Interface) tiene 4 regiones direccionables separadas llamadas espacios habilitados de chip (CE0-CE3). La SDRAM ocupa CE0 de EMIFA mientras que el CPLD y la FLASH son mapeadas al CE0 y CE1 del EMIFG respectivamente. Las tarjetas hijas usan CE2 y CE3 del EMIFA.

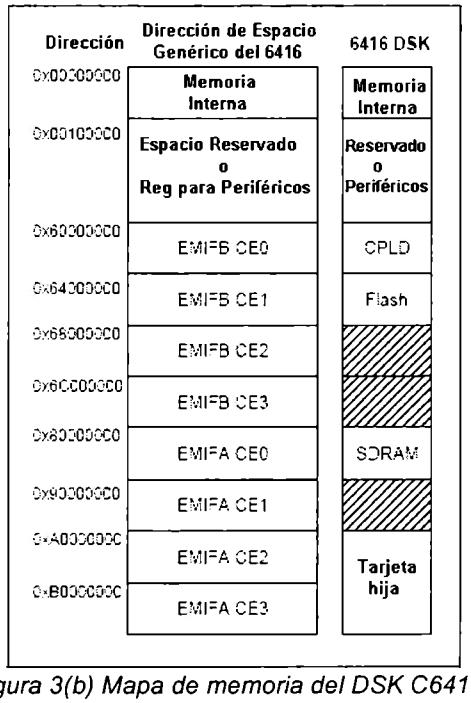


Figura 3(b) Mapa de memoria del DSK C6416T

Una vez que comenzamos el desarrollo, la primera dificultad a la que nos enfrentamos en esta etapa, fue que, por el tipo de datos con el que se estaba trabajando, archivos *.dat, y porque la programación en lenguaje Matlab es diferente a C, no pudimos implementar los algoritmos idénticamente a como los teníamos en Matlab, sin embargo, el nuevo algoritmo se baso en la misma idea de la eliminación de píxeles.

El algoritmo se basa en el uso de apuntadores para leer los buffers de entrada R, G y B, de esta forma se podrá manipular esta información, tomar los píxeles que se necesiten y eliminar los que no se necesiten de nuestra imagen original.

* El código del programa se presenta en el Anexo G al final del trabajo.

En la figura 3.1 se muestran los resultados obtenidos con este algoritmo.

La primera imagen es la original, cuyas dimensiones son de 52x64 píxeles, la última imagen es el resultado final, es decir, la imagen redimensionada a 26x32 píxeles. (La imagen de en medio es una imagen de prueba, es decir, un paso intermedio entre la imagen original y la imagen redimensionada en su número de píxeles por línea y su número de líneas).

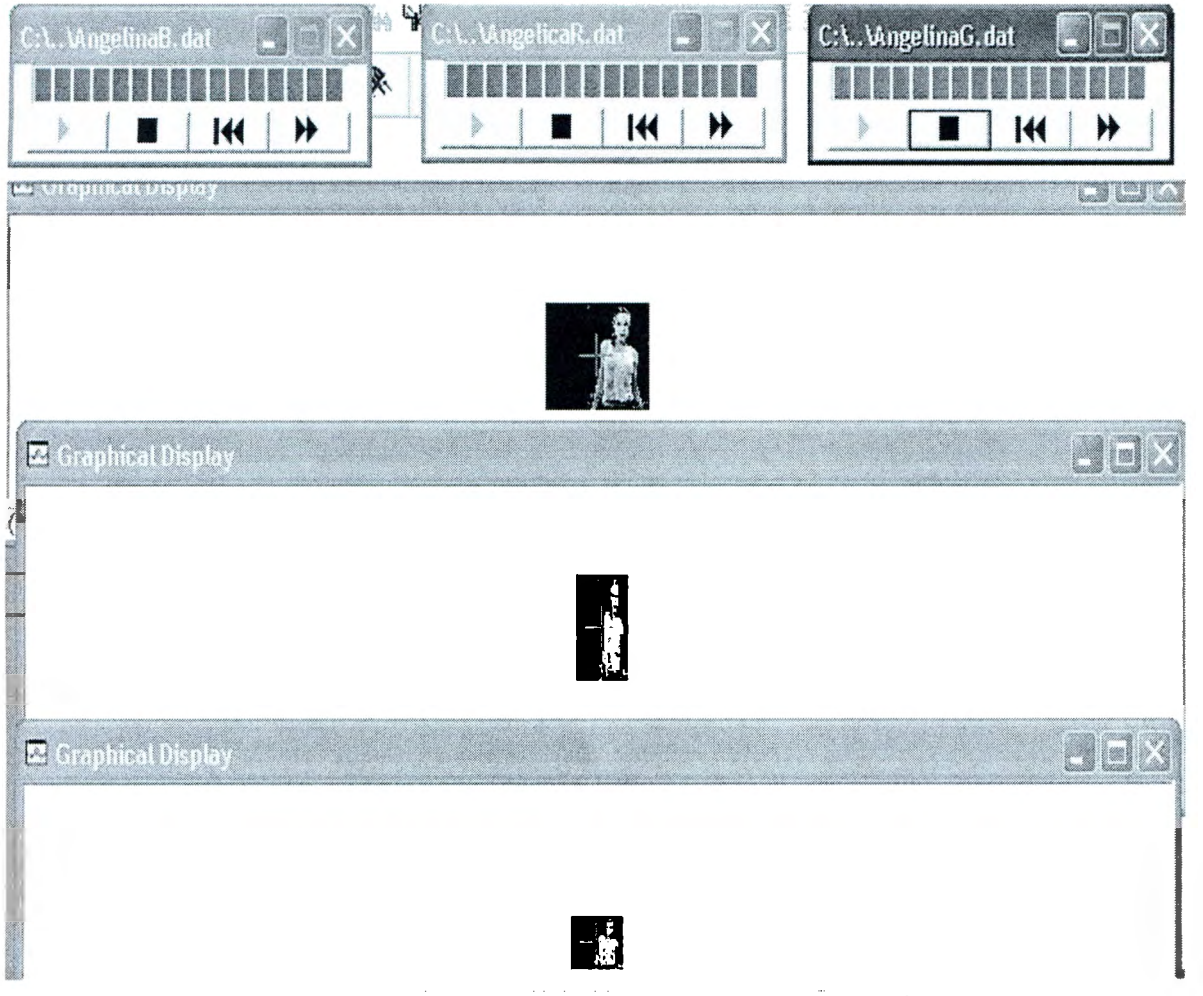


Figura 3.1 Resultados del algoritmo de eliminación en el DSP

A la vez que se trabajaba en los algoritmos de redimensionamiento en el DSP, se estuvo trabajando en cómo ingresar imágenes (lectura) por algunos de los puertos del DSP. Desgraciadamente la idea que se planteó al principio del semestre, de entrar por el puerto HPI (Host Port Interface), ya que era el único puerto paralelo de hasta 32 bits, resultó no ser adecuada. Esto debido a que implementar este tipo de interfaz no es nada fácil y se usa para conectar en paralelo dispositivos que deseen tener acceso a la memoria del DSP, lo que hace que su uso no sea sencillo.

Otra opción que planteamos, fue la de usar el puerto de video del DSP, pero el gran problema fue que este puerto solo aceptaba formato YCrCb y fue por eso que nuestra propuesta no fue aceptada por Kokai. Por ultimo, la siguiente opción propuesta fue la de usar los pines de entrada y salida de propósito general del DSP (GPIO´s) pero desafortunadamente, solo se contaba con 16 pines de ese tipo y nosotros necesitábamos un mínimo de 24 pines para los canales RGB, mas otros 3 para sincronía vertical, horizontal y el reloj. Es por eso que ahora el gran problema era cómo íbamos a hacer la lectura de bits si no teníamos un puerto por el cual accesar los datos y es por esta razón por la que se nos propuso cambiar a un FPGA, el cual cuenta con más de 100 pines disponibles para entradas y salidas. Una de las grandes ventajas que tuvimos fue que ya se contaba con el FPGA, así que no tuvimos que esperar para adquirirlo, lo que nos ayudó mucho, puesto que ya teníamos muy poco tiempo para trabajar en él.

4. Adquisición y Redimensionamiento de Video en el FPGA Spartan-3

Una vez acordado que la mejor solución para obtener video digital redimensionado en tiempo real era el FPGA, se nos proporcionó el Kit de Desarrollo SPARTAN-3 por Kokai para que desarrolláramos una aplicación de redimensionamiento y finalmente pudiéramos hacer las pruebas correspondientes. El gran problema que se tuvo en esta etapa fue el tiempo, puesto que estaba a casi un mes de la entrega final del proyecto. El Kit de desarrollo que utilizamos fue el Xilinx Spartan-3 XC3S200. A continuación se presenta la descripción del mismo:

Componentes y Características

La figura 4(a) muestra la tarjeta de desarrollo Spartan-3 Starter Kit, la cual incluye los siguientes componentes y características.

- 200,000-gate Xilinx Spartan-3 XC3S200 FPGA ①
  - 4,320 compuertas lógicas
  - Doce bloques de RAM cada uno de 18Kbit
  - Doce 18x18 multiplicadores en hardware
  - Cuatro Manejadores de Reloj Digital (DCMs)
  - Más de 173 señales I/O definidas por el usuario
- Una Plataforma Flash Xilinx XCF02S de 2Mbit en un sistema programable de configuración PROM ②
  - 1Mbit de información no volátil o aplicaciones de código de almacenamiento disponible para la configuración del FPGA
  - Opciones Jumper que permiten al FPGA la posibilidad de leer datos de la PROM o configuraciones FPGA de otras fuentes ③
- 1Mbyte de "Fast Asynchronous SRAM" ④
- Puerto VGA ⑤
- Puerto Serial RS-232 ⑥
  - Un traductor RS232 "transceiver/level" ⑦
  - Un segundo canal RS232 transmisor y receptor disponible para puntos de pruebe en la tarjeta ⑧
- Un Puerto "PS/2-style mouse/keyboard" ⑨

- Cuatro Displays de caracteres de siete segmentos (12)
- Ocho switches (11)
- Ocho salidas individuales de LEDs (12)
- Cuatro botones para switches momentarios (13)
- Un cristal Oscilador de 50MHz (14)
- Una ranura para un oscilador adicional externo (15)
- Un selector vía jumpers para configurador de FPGA (16)
- Un botón de switch para forzar la reconfiguración del FPGA (17)
- LEDS indicadores para la indicación de una configuración éxitosa (18)
- Tres expansores de pines, cada uno con 40 pines (19) (20) (21)
- Un puerto JTAG (22)
- Un puerto Paralelo – JTAG (23)
- Un puerto JTAG compatible para el cable paralelo Xilinx (24)
- Una entrada para adaptador de AC (25)
- Un LED indicador de prendido (26)
- Reguladores en la tarjeta de 3.3V(27), 2.5V (28) y 1.2V(29)

Localización de componentes

En las siguientes figuras se indican la localización de los componentes tanto en el la parte inferior como en la superior de la tarjeta de desarrollo:

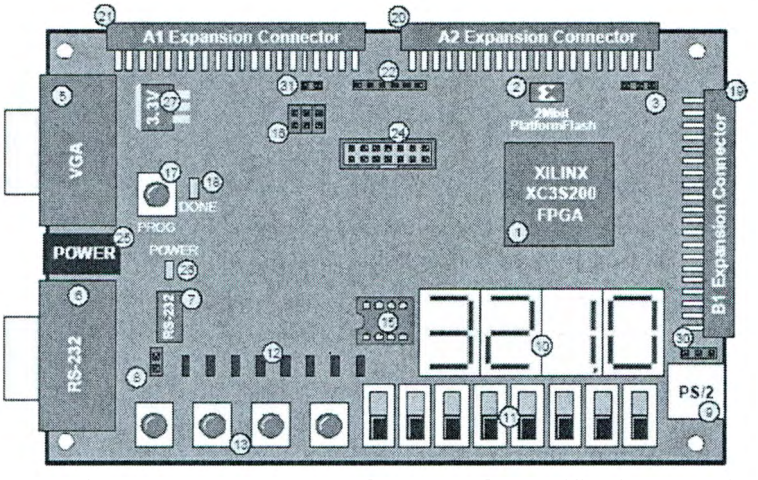


Figura 4(a) Tarjeta de Desarrollo Spartan-3 Starter Kit, vista superior

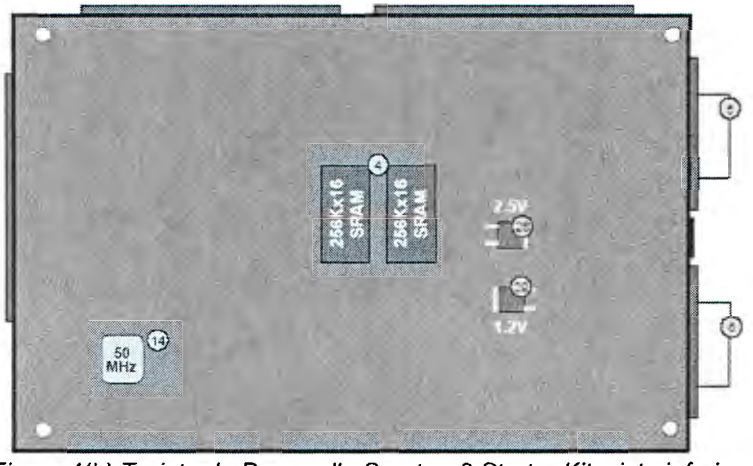


Figura 4(b) Tarjeta de Desarrollo Spartan-3 Starter Kit, vista inferior

Durante esta etapa, una vez más partimos de cero, ya que nadie del equipo había trabajado antes con el lenguaje VHDL, así que tuvimos que empezar a documentarnos en este lenguaje y buscar diferentes asesorías para adquirir los conocimientos necesarios que nos permitiera desarrollar una aplicación de redimensionamiento de video digital en formato RGB.

Una vez que logramos comprender la arquitectura del leguaje VHDL se continuó con la aplicación que hiciera la adquisición de video en formato RGB digital de AXB píxeles y pudiera ser redimensionado a la mitad de su tamaño. Este proceso consistió en obtener video en formato RGB de la tarjeta Silicon Image, la cual recibe video digital por medio de un conector DVI y lo convierte en video digital en formato RGB, el cual a su vez conectamos a la entrada del FPGA. En esta configuración se reciben ocho bits por cada canal (R, G y B) a una frecuencia de 165 MHz, una señal de sincronía vertical y otra de sincronía horizontal, las cuales son las encargadas de señalar el inicio y fin de una línea, así como de un campo.

Nuestro programa lo que hace es capturar esos 27 bits de formato RGB por los pines de entrada con los que cuenta el Kit de desarrollo, además de un bit de RESET, el cual habilita o deshabilita la operación del redimensionamiento. Una vez que se tienen las entradas, lo que se hace para redimensionar el video a la mitad es detectar los flancos de subida del reloj, de tal forma que sólo en los flancos de subida los bits de entrada puedan cambiar, es decir, que sólo habrá cambios en los bits de entrada cuando se detecte un flanco de subida en el reloj. De esta forma se obtiene un redimensionamiento a la mitad del video entrada. Por otra parte también se programaron dos contadores, uno se encarga de detectar los flancos de subida en la señal de sincronía horizontal para poder detectar el número de líneas que se registran y el otro hace uno para contar 24 bits de entrada para que de esta forma podamos llevar el conteo del ancho del video.

Finalmente, se obtienen a la salida los canales R, G y B ya redimensionados a la mitad, los bits de sincronía vertical y horizontal, el reloj de entrada y los contadores de líneas y de ancho del video.

En el anexo H se muestra el código VHDL del redimensionamiento:

Se realizó un diagrama a bloques de nuestro sistema, en él se muestran las entradas y salidas de nuestra aplicación

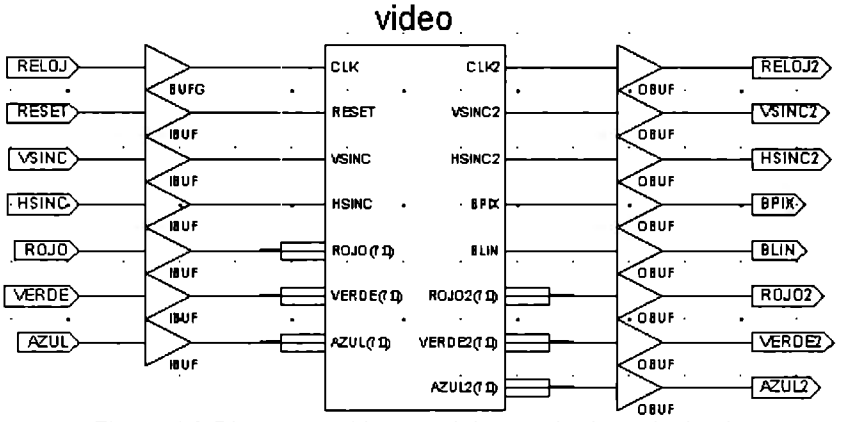


Figura 4.2 Diagrama a bloques del contador hexadecimal

Cabe mencionar que debido al poco tiempo que tuvimos para aprender a programar en VHDL y poder realizar una aplicación de redimensionamiento actualmente nos encontramos en las pruebas para poder tener la aplicación en la tarjeta de desarrollo, por lo que aún estamos considerando los pines de entrada y salida con los que cuenta la tarjeta de desarrollo.

IV. CONCLUSIONES

1. Conclusiones Generales

A principios del semestre pasado, cuando tuvimos que seleccionar el proyecto de ingeniería en el cual íbamos a trabajar durante los siguientes ocho meses, realmente no teníamos una idea muy clara sobre qué era lo que queríamos desarrollar como proyecto. Lo único que sabíamos era que este tenía que ser un proyecto que tuviera aplicaciones prácticas reales, que fuera más allá de los conocimientos teóricos que habíamos adquirido en las aulas y los libros, es decir, que los resultados del proyecto realmente sirviera para satisfacer una necesidad.

Fue por este motivo que decidimos tomar la propuesta de proyecto de Kokai Electrónica. Para nosotros era muy importante tener la oportunidad de trabajar con una empresa mexicana líder en su área, en un proyecto de innovación y desarrollo en una tecnología que hasta la fecha no se había desarrollado en nuestro país. Esto fue muy importante para nosotros, ya que, además de los conocimientos y capacidades que desarrollaríamos en el proyecto, también representaba una excelente oportunidad de enfrentarnos por primera vez a lo que era el mundo laboral y sobre todo dentro de una empresa dedicada 100% a la ingeniería electrónica.

Después de ocho meses de trabajo, creemos que en verdad fue muy enriquecedor para nuestra formación como ingenieros el haber participado en este proyecto. A pesar de todas las dificultades que se tuvieron y a pesar de que no fue posible realizar la integración completa del proyecto como se había planteado originalmente, el trabajo realizado a lo largo de estos dos semestres siempre se mantuvo acorde a los objetivos y metas establecidas desde el principio, respetando, hasta donde nos fue posible, la calendarización de actividades definida al inicio de cada semestre.

Durante la primera etapa del proyecto tuvimos que documentarnos en la parte del redimensionamiento de imágenes, específicamente en los métodos de Interpolación bilineal y el Vecino más Cercano, ya que la idea que nos propusimos fue trabajar en estos algoritmos para que se pudieran implementar en un DSP y poder tener el redimensionamiento del video digital en formato RGB. Sin embargo, conforme fuimos conociendo el funcionamiento de estos algoritmos, pudimos darnos cuenta de que la implementación de estos iba a ser bastante complicada y que nos iba a llevar mucho tiempo, además de que estos estaban en Matlab y se necesitaban pasar a C para poderlos trabajar en el DSP.

Debido a este primer problema al que nos enfrentamos, se decidió crear otro algoritmo que trabajara de manera similar a los antes mencionados, el resultado de este algoritmo creado por nosotros y llamado Half, realmente no difería mucho a los otros dos en la cuestión de la imagen obtenida, además se tomó en cuenta de que el video sería implementado en pantallas de LED's Full Color, por lo usar cualquiera de los tres algoritmos no crearía resultados notorios en cuanto a la calidad de la imagen desplegada. Por este motivo, se decidió trabajar sobre este algoritmo, pues resultaba mucho más fácil de implementar en lenguaje C para poderlo trabajar en el DSP. Esta dedición fue tomada junto con el asesor del primer semestre, el Dr. Emmanuel Moya.

Una vez que se decidió trabajar con este algoritmo y que ya se tenía implementado en Matlab, además de que se hicieron diferentes pruebas con él para poder comparar los resultados con los otros dos algoritmos, se continúo con la realización de la interfaz RS232-I2C. Para poder realizar esta interfaz se necesito usar un PIC, del cual tampoco teníamos conocimiento, sin embargo, lo que fue de gran ayuda fue que sí

Tecnológico de Monterrey, Campus Ciudad de México
Biblioteca

teníamos conocimiento en microprocesadores, los cuales nos sirvieron para poder entender el funcionamiento del PIC. A partir de esto fue como se empezó a realizar, en conjunto con el equipo de Captura y Digitalización de Video, el desarrollo de esta interfaz, la cual, desafortunadamente, se tuvo que dejar inconclusa debido a que el DSP con el que contamos no tenía la entrada de un bus I^2C.

De esta etapa se desprende una de las conclusiones más importantes del trabajo, la cual es que es muy importante hacer una investigación completa y extensiva sobre los dispositivos y recursos disponibles en el mercado, para poder seleccionar el más adecuado a las necesidades del proyecto. En nuestro caso, esto representó una seria dificultad en el desarrollo del proyecto, ya que el DSP que se adquirió no cumplía con las características necesarias para lo que nosotros lo requeríamos, es decir, es de los DSPs mas rápidos que hay en el mercado lo cual era una característica bastante importante sin embargo los problemas fueron los puertos (bus I2C y puerto de mínimo 27 pines en paralelo).

En primer lugar, el DSP no contaba con un puerto I^2C, por lo que, como ya se ha explicado, no nos fue posible llevar a cabo la integración de la interfaz RS232 - I^2C, lo cual representó que la mayor parte del trabajo realizado durante el 1er semestre no se viera reflejada en el resultado final del proyecto.

La segunda y tal vez más importante cuestión, fue el hecho de que el DSP no contara con los suficientes puertos de entrada y salida para poder manipular video digital en formato RGB de 24 bits en tiempo real. Como se explicó en la parte de desarrollo y resultados, no fue sino hasta el mes de marzo cuando nos dimos cuenta de que los puertos de entrada/salida con los que contaba nuestro kit de desarrollo no serían suficientes para poder llevar a cabo la adquisición y entrega de video en tiempo real, como se había planteado en un principio. Y a pesar de que se contaba con un puerto de 32 bits del HPI, este puerto no funcionaba para nuestro proyecto.

El hecho de trabajar con un equipo que no se ajustara por completo a las necesidades del proyecto fue, sin duda, la mayor dificultad en el desarrollo de nuestro trabajo ya que implicó una serie de complicaciones que detenían nuestro avance y nos limitaban aún más en cuanto al tiempo de entrega del proyecto.

En este sentido, otra conclusión muy importante que podemos obtener del desarrollo de nuestro proyecto, es que es esencial contar con el equipo con el cual se va a trabajar en el proyecto desde el inicio del mismo. En nuestro caso, el tiempo fue siempre la principal limitante, ya que, siendo objetivos, el tiempo de trabajo neto que tuvimos fue de menos de cuatro meses con el DSP y de apenas un mes con el FPGA.

Evidentemente, para un trabajo de la magnitud de un proyecto de ingeniería como el nuestro, el cual estaba basado en gran medida en el manejo de dispositivos como el DSP o el FPGA, y considerando que ninguno de los integrantes del equipo había utilizado ninguno de estos equipos con anterioridad, el tiempo disponible era muy limitado. Si a esto le agregamos todas las dificultades que se tuvieron debido a la incompatibilidad del DSP con los requisitos del proyecto, el resultado es que el tiempo con el que contábamos fue insuficiente para alcanzar todas las metas establecidas.

Por otro lado, creemos que algo muy importante es que la realización de este proyecto nos permitió conocer y programar varios lenguajes como lo fue C en tres diferentes plataformas, VHDL y Lenguaje Matlab. El problema que se tuvo al programar sobre diferentes lenguajes fue la integración, pues tuvimos que hacer diferentes versiones de nuestros algoritmos para poder pasar de un lenguaje a otro. Esto se vio más notoriamente entre Matlab y C, que fue el lenguaje utilizado para programar el DSP, ya

que en Matlab se tienen funciones específicas y que resultan muy complicadas de programar manualmente en C, como fue la parte de generar los bits de la imagen, pasarlos a base hexadecimal y poder tener en una columna todos los bits, de tal forma que pudiéramos generar el archivo *.dat necesario para poder trabajar en el DSP. El problema más grande que enfrentamos en esta parte fue que Matlab sólo almacena cierta cantidad de valores en su ventana de resultados, por lo que obtener los bits de una imagen grande fue complicado.

En cuanto al trabajo realizado en el FPGA al principio creímos que iba a ser muy complicado debido al tiempo que teníamos para poder desarrollar la aplicación y al desconocimiento del lenguaje VHDL, pero conforme nos fuimos involucrando con este lenguaje pudimos darnos cuenta que no sería tan difícil como lo era trabajar en el DSP, además de que también contamos con asesorías de tal forma que pudimos familiarizarnos rápidamente con el lenguaje y de esta forma poder empezar a trabajar con el FPGA, no obstante, durante el desarrollo de la aplicación nos encontramos con varios problemas, sobre todo con la herramienta para poder compilar el programa.

Como comentario final podemos decir que es de suma importancia para hacer un proyecto contar con la asesoría necesaria para lograrlo, ya que si no se cuenta con ello, el desarrollo del proyecto se complica mucho y esto se traduce en pérdidas de tiempo considerables. En nuestro caso, nosotros nos fuimos por el camino de Matlab ya que era una herramienta que conocíamos y por eso nos ayudó bastante al conocimiento y procesamiento de una imagen RGB. Por otra parte, la asistencia a la clase de DSPs con el profesor Alfredo Mantilla para poder conocer lo básico respecto al Code Componer Studio fue muy útil para realizar los algoritmos en el DSP, y por último la asesoría del profesor Juan Carlos Herrera (la cual nos fue conseguida por el profesor Mantilla) nos fue de mucha ayuda con el desarrollo de nuestros algoritmos en el FPGA.

En este sentido, creemos que es importante mencionar el hecho de que nosotros no contábamos con algún tipo de trabajo o avance previamente realizado sobre el proyecto. Creemos que esto hubiera sido una gran ventaja y nos hubiera ayudado a avanzar mas rápido y no perder tanto tiempo, además de que nos hubiera ayudado mucho en el desarrollo del proyecto, ya que en varias ocasiones nos encontramos sin saber qué hacer o por dónde atacar el problema y a pesar de contar con asesoría, eran cuestiones muy específicas, de las cuales no había mucha información disponible o recursos de donde basarnos.

2. Perspectivas y trabajo a futuro

Se tiene ya programado un PIC capaz de comunicarse con otro PIC usando I^2C, también se tiene un algoritmo ya probado y funcionando en el Code Componer Studio que hace un redimensionamiento de la imagen AxB a axb, así como un programa en VHDL que hace un redimensionamiento de video digital en un FPGA, de una imagen AxB a otra A/2xB/2. En términos generales, revisando todas las especificaciones que se piden en el proyecto inicial, se cumplen las más importantes y se ha logrado un gran avance y partiendo desde cero.

Hemos platicado con nuestros asesores y nos gustaría que este proyecto se volviera a dar en el campus, específicamente para alumnos de la carrera de Ingeniería en Sistemas Electrónicos (ISE), ya que ellos están más familiarizados con la programación de microprocesadores, así como el lenguaje de VHDL. Además, debido a los avances que ya se tienen en el desarrollo del proyecto, se sugiere que el trabajo sea realizado en un período de un semestre. También por este motivo resulta más conveniente que este proyecto sea retomado por alumnos de la carrera ISE. Sin

embargo, esto no significa que el proyecto no pueda ser retomado por alumnos de la carrera IEC, en este caso, sería necesario redefinir los objetivos para realizarse en un trabajo de dos semestres

Proponemos dos enfoques para continuar con el desarrollo del proyecto. El primero sería seguir utilizando un FPGA como dispositivo para llevar a cabo el redimensionamiento de video digital, de una resolución AxB a una resolución menor, axb, tal y como se especifica en los requerimientos iniciales del sistema. Sin embargo, por lo que nos comentaron nuestros asesores y sinodales, el trabajo con el FPGA podría resultar más complicado de lo que se espera. Por este motivo, la segunda opción que proponemos es continuar con el trabajo utilizando el DSP como dispositivo para realizar el redimensionamiento. En este caso, lo que se sugiere es realizar una investigación a fondo sobre los diferentes DSPs que existen en el mercado, para seleccionar el que mejor se ajuste a las necesidades del proyecto y que permita una integración completa del sistema. En este sentido, tal vez sea necesario adquirir alguna tarjeta de expansión, que permita tener una mayor cantidad de puertos de entrada y salida disponibles, que fue la principal limitante a la que nos enfrentamos con el DSP.

En cualquiera de las dos propuestas, el objetivo final de este trabajo sería lograr cumplir con todas las características que Kokai Electrónica había definido originalmente y que se especifican en la sección de requerimientos del sistema, en la introducción de este trabajo.

3. Comentarios

Alejandro López

En lo personal, me parece que fue una experiencia muy valiosa el trabajar en este proyecto. Nunca me había enfrentado a un proyecto de tal magnitud, el cual abarcara tantas áreas de conocimiento y sobre todo que tuviera una aplicación realmente práctica. Esto es muy importante, ya que me dio una idea de lo que era trabajar en un proyecto de ingeniería real, que va más allá de ser un requisito para aprobar una materia de la carrera.

Creo que el haber participado en el desarrollo del proyecto realmente me sirvió para aprender una gran cantidad de cosas que nunca había visto en la carrera, como es el uso de dispositivos como los DSPs y los FPGAs, además de conocimientos sobre procesamiento de imágenes, que es un área realmente interesante y en la cual nunca había trabajado.

Por otro lado, algo que considero como un plus que me ha dejado el proyecto, es el hecho de haber trabajado con la empresa Kokai Electrónica. Esto fue muy importante para mí, ya que fue mi primera experiencia en el ambiente laboral y me sirvió para conocer las exigencias de trabajar en una empresa, que son muy diferentes a lo que estamos acostumbrados en la escuela.

Finalmente, puedo decir que el haber trabajado con Emilio y Gabriel fue muy gratificante, ya que pudimos conformar un gran equipo de trabajo, donde los tres asumimos los compromisos con responsabilidad y a pesar de todas las dificultades a las que nos enfrentamos, fuimos capaces de desarrollar el trabajo de manera equitativa y sin complicaciones entre nosotros.

Emilio Galán

Me fue importante trabajar en una empresa tan importante como Kokai y saber el nivel de exigencia que se me va a pedir en el campo laboral. Realmente estoy satisfecho con el gran avance que mis compañeros y yo logramos en este semestre ya que no contábamos con nada programado ni en el DSP y ni mucho menos en el FPGA.

No me gustó con respecto al puerto HPI que la empresa Kokai nos dijera hasta hace poco tiempo que no debemos trabajar con el HPI ya que invertimos mucho tiempo en tratar de entenderle, sin embargo la solución que nos proporcionaron y los nuevos objetivos fueron acertados y creemos que van a ser cumplidos. Afortunadamente el equipo con el que trabaje, (Alejandro López y Gabriel López) son eficientes y responsables y eso hizo que dividirnos las tareas para poder avanzar más funcionara como estrategia para sacar este proyecto adelante.

Gabriel López

Como primera parte quiero destacar la importancia que tuvo haber trabajado en proyecto de la empresa mexicana Kokai Electrónica, pues considero que al haber hecho se tiene en primer lugar una gran ventaja pues es muy claro que lo que se desarrolló durante estos dos semestres de trabajo será la base de una nueva tecnología que se desarrollará aquí en México y pronto estará disponible en el mercado, es por eso que se adquiere un mayor compromiso y una mayor satisfacción, pues este trabajo no quedará como un trabajo más dentro de alguna biblioteca.

Con la realización de este proyecto tuve la oportunidad de poder adquirir nuevos conocimientos los cuales han complementado mi formación en la carrera, además de que tuve la oportunidad de fortalecer y desarrollar nuevas habilidades como lo fueron el trabajo en equipo, ya que sin este definitivamente no se hubiera podido lograr nada, por otra parte también pude desarrollar habilidades en las cuales enfrentar grandes problemas y los cuales se tienen que resolver buscando la mejor alternativa, sin olvidar el tiempo y lo laborioso que esta pueda ser para resolver este problema.

Por otra parte también en lo personal tuve que enfrentar diversos obstáculos en cuanto al poder trabajar de lleno en el proyecto, pues muchas las demás materias también requerían de tiempo, pero creo que la mejor solución a esto fue que supimos dividirnos tareas y que cada uno de nosotros se comprometiera con la tarea que tenía y poder tener los resultados a tiempo, es por que pienso que el trabajo en equipo fue indispensable para poder cumplir con las metas propuestas.

4. Agradecimientos

Los integrantes del equipo queremos agradecer a:

- Los profesores Emmanuel Moya y Alfredo Mantilla, quienes asesoraron este proyecto en todas sus etapas y nos brindaron su constante apoyo durante estos ocho meses de trabajo.
- Al personal de Kokai Electrónica, en especial a los Ingenieros César Galicia y Antonio Luna, por su valioso apoyo y asesoría en la realización de este proyecto.
- Al profesor Juan Carlos Herrera, quien nos brindó importante asesoría en la parte de lógica programable.
- A la Ing. Marisol Romo, por su valioso apoyo en la parte de VHDL.
- Y a todos los que de alguna u otra forma colaboraron en el desarrollo de este proyecto.

V. FUENTES DE INFORMACIÓN

1. **González Maxinez, David** *El Arte de Programar Sistemas Digitales*, Edit Compañía Editorial Continental, México 2002

2. **González, Rafael.** *Digital Image Processing*, Second Edition. Prentice Hall. USA, 2002.

3. **Grob, Bernard**. *Basic Television and Video Systems*, Sixth Edition. Glencoe McGraw-Hill. USA, 1999.

4. **Pajarez, Gonzalo.** *Visión por Computador: Imágenes Digitales y Aplicaciones*. Editorial Ra-Ma. España, 2002.

5. **Irazabal, Jean-Marc.** *AN10216-01 I^2C Manual*. Philips Semiconductors, 2003. http://www.semiconductors.philips.com/acrobat_download/applicationnotes/AN10216_1.pdf

6. **Herrera, Juan Carlos**. *Jcrls en la Web* http://mx.geocities.com/jcrls/biblioteca.htm

7. **Nicklin, David** *FPGAs – Enabling DSP in Real-Time Video Processing.* Xcell Journal, Summer 2002

8. **Springer, Alan.** *MAX220-49 DS +5V-Powered, Multichannel RS-232 Drivers/Receivers*. Maxim Integrated Products. 2004 http://pdfserv.maxim-ic.com/en/ds/MAX220-MAX249.pdf

9. *Le cours PIC de Bigonoff* http://www.abcelectronique.com/bigonoff/organisation.php?par=6f600

10. *PIC16F87X Data Sheet.* Microchip Technology Inc. 2001 http://ww1.microchip.com/downloads/en/DeviceDoc/30292c.pdf

11. *Spartan-3 Starter Kit Board User Guide UG130 (v1.1)* Xilinx Inc. May 13, 2005 http://www.xilinx.com/bvdocs/publications/ds099.pdf

12. *TDA8754 Triple 8-bit video ADC up to 270 Msps*. Philips Semiconductors, 2005. http://www.semiconductors.philips.com/acrobat_download/datasheets/TDA8754_6.pdf

13. *Texas Instruments Video & Imaging Applications* www.ti.com/video

14. *TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors Datasheet.* Texas Instruments Inc. 2005. http://focus.ti.com/lit/ds/sprs226h/sprs226h.pdf

15. *TMS320C6416T DSK Technical Reference.* DSP Development Systems. Texas Instruments Inc. 2004 http://c6000.spectrumdigital.com/dsk6416/V3/docs/dsk6416_TechRef.pdf

16. *TMS320C64x Image/Video Processing Library*. Programmer's Reference. Texas Instruments Inc. October 2003.
http://www.seeddsp.com/pdf/soft/c600002089.pdf

17. *TMS320C64x DSP Video Port/VCXO Interpolated Control (VIC) Port Reference Guide*. Texas Instruments Inc. January 2005
http://focus.ti.com/lit/ug/spru629d/spru629d.pdf

18. *Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs* Xilinx Inc. January 5, 2006
http://www.xilinx.com/bvdocs/publications/ds312.pdf

VI. Anexos

Anexo A. Código para interfaz gráfica en Borland C++ Builder

Código para el encabezado Unit1.h:

```
//---------------------------------------------------------------------------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published: // IDE-managed Components
        TMemo *Memo1;
        TButton *Brillo;
        TButton *Contraste;
        TButton *Video;
        TButton *FR;
        TButton *Bits;
        TButton *Res;
        TButton *Fabrica;
        TButton *Edo;
        TEdit *ParametrosBrillo;
        TEdit *ParametrosContraste;
        TEdit *TipoVideo;
        TEdit *FrameRate;
        TEdit *NumeroBits;
        TEdit *Resolucion;
        TEdit *ConfigFabrica;
        TEdit *Estado;
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall Memo1KeyPress(TObject *Sender, char &Key);
        void __fastcall BrilloClick(TObject *Sender);
        void __fastcall ContrasteClick(TObject *Sender);
        void __fastcall VideoClick(TObject *Sender);
        void __fastcall FRClick(TObject *Sender);
        void __fastcall BitsClick(TObject *Sender);
        void __fastcall ResClick(TObject *Sender);
        void __fastcall FabricaClick(TObject *Sender);
        void __fastcall EdoClick(TObject *Sender);

        private:    // User declarations
        public:             // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
#ifndef Unit2H
#define Unit2H
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
class TRead : public TThread
{
private:
protected:
        void __fastcall DisplayIt(void); // ADD THIS LINE
        void __fastcall Execute();
public:
```

```
        __fastcall TRead(bool CreateSuspended);
};
//---------------------------------------------------------------------------
#endif
```

Código del programa:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
char InBuff[100];
TForm1 *Form1;
HANDLE hComm = NULL;
TRead *ReadThread;
COMMTIMEOUTS ctmoNew = {0}, ctmoOld;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
    DCB dcbCommPort;

  // OPEN THE COMM PORT.
  // REPLACE "COM2" WITH A STRING OR "COM1", "COM3", ETC. TO OPEN
  // ANOTHER PORT.

  hComm = CreateFile("COM1",
                     GENERIC_READ | GENERIC_WRITE,
                     0,
                     0,
                     OPEN_EXISTING,
                     0,
                     0);

  // IF THE PORT CANNOT BE OPENED, BAIL OUT.

  if(hComm == INVALID_HANDLE_VALUE) Application->Terminate();

  // SET THE COMM TIMEOUTS IN OUR EXAMPLE.

  GetCommTimeouts(hComm,&ctmoOld);
  ctmoNew.ReadTotalTimeoutConstant = 100;
  ctmoNew.ReadTotalTimeoutMultiplier = 0;
  ctmoNew.WriteTotalTimeoutMultiplier = 0;
  ctmoNew.WriteTotalTimeoutConstant = 0;
  SetCommTimeouts(hComm, &ctmoNew);

  // SET BAUD RATE, PARITY, WORD SIZE, AND STOP BITS.
  // THERE ARE OTHER WAYS OF DOING SETTING THESE BUT THIS IS THE EASIEST.
  // IF YOU WANT TO LATER ADD CODE FOR OTHER BAUD RATES, REMEMBER
  // THAT THE ARGUMENT FOR BuildCommDCB MUST BE A POINTER TO A STRING.
  // ALSO NOTE THAT BuildCommDCB() DEFAULTS TO NO HANDSHAKING.

  dcbCommPort.DCBlength = sizeof(DCB);
  GetCommState(hComm, &dcbCommPort);
  BuildCommDCB("9600,N,8,1", &dcbCommPort);
  SetCommState(hComm, &dcbCommPort);

  // ACTIVATE THE THREAD. THE FALSE ARGUMENT SIMPLY MEANS IT HITS THE
  // GROUND RUNNING RATHER THAN SUSPENDED.

  ReadThread = new TRead(false);
}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
        // TERMINATE THE THREAD.

  ReadThread->Terminate();

  // WAIT FOR THREAD TO TERMINATE,
  // PURGE THE INTERNAL COMM BUFFER,
  // RESTORE THE PREVIOUS TIMEOUT SETTINGS,
  // AND CLOSE THE COMM PORT.

  Sleep(250);
  PurgeComm(hComm, PURGE_RXABORT);
  SetCommTimeouts(hComm, &ctmoOld);
  CloseHandle(hComm);

}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
//   Important: Methods and properties of objects in VCL can only be
//   used in a method called using Synchronize, for example:
//
//      Synchronize(UpdateCaption);
//
//   where UpdateCaption could look like:
//
//      void __fastcall TRead::UpdateCaption()
//      {
//        Form1->Caption = "Updated in a thread";
//      }
//---------------------------------------------------------------------------

__fastcall TRead::TRead(bool CreateSuspended)
        : TThread(CreateSuspended)
{
}
//---------------------------------------------------------------------------
void __fastcall TRead::DisplayIt()
{

  // NOTE THAT IN THIS EXAMPLE, THERE IS NO EFFORT TO MONITOR
  // HOW MUCH TEXT HAS GONE INTO Memo1. IT CAN ONLY HOLD ABOUT 32K.
  // ALSO, NOTHING IS BEING DONE ABOUT NON-PRINTABLE CHARACTERS
  // OR CR-LF'S EMBEDDED IN THE STRING.

  // DISPLAY THE RECEIVED TEXT.

  Form1->Memo1->SetSelTextBuf(InBuff);

}

//---------------------------------------------------------------------------
void __fastcall TRead::Execute()
{
        //---- Place thread code here ----

  DWORD dwBytesRead;

  // MAKE THE THREAD OBJECT AUTOMATICALLY DESTROYED WHEN THE THREAD
  // TERMINATES.

  FreeOnTerminate = true;

  while(1)
```

```
  {

   // TRY TO READ CHARACTERS FROM THE SERIAL PORT.
   // IF THERE ARE NONE, IT WILL TIME OUT AND TRY AGAIN.
   // IF THERE ARE, IT WILL DISPLAY THEM.
        TPoint p;

     ReadFile(hComm, InBuff, 50, &dwBytesRead, NULL);

     if(dwBytesRead)
     {
       InBuff[dwBytesRead] = 0; // NULL TERMINATE THE STRING
       Synchronize(DisplayIt);
     }
     for(int i=0; i<dwBytesRead; i++)
     {
        p=Form1->Memo1->CaretPos;
        Form1->Memo1->Lines->Strings[p.y]+InBuff; //lines es donde esta la
info del memo
     }
   }

}
//---------------------------------------------------------------------------
void __fastcall TForm1::Memo1KeyPress(TObject *Sender, char &Key)
{
// TRANSMITS ANYTHING TYPED INTO THE MEMO AREA.

  //if(Key != 13 && (Key < ' ' || Key > 'z')) Key = 0;
  //TransmitCommChar(hComm, Key);
  //pwd=Kokai
  //comandos: '1' - para brillo, '2' - para contraste,
  //'3' - seleccionar entrada de video, '4' - frame rate, '5' - numero de bits
  //'6' - resolucion, '7' - programar configuracion de fabrica
  //'8' - solicitar estado
  //fin de cadena = 03 hex

  // THIS PREVENTS TYPED TEXT FROM DISPLAYING GARBAGE ON THE SCREEN.
  // IF YOU ARE CONNECTED TO A DEVICE THAT ECHOES CHARACTERS, SET
  // Key = 0 WITHOUT THE OTHER STUFF.

}
//---------------------------------------------------------------------------

void __fastcall TForm1::BrilloClick(TObject *Sender)
{
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'o');
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'a');
        TransmitCommChar(hComm, 'i');
        TransmitCommChar(hComm, '1');
        TransmitCommChar(hComm, StrToInt(ParametrosBrillo -> Text) );
        TransmitCommChar(hComm, 03);
        //pwd=Kokai
        //comandos: '1' - para brillo, '2' - para contraste,
        //'3' - seleccionar entrada de video, '4' - frame rate, '5' - numero
de bits
        //'6' - resolucion, '7' - programar configuracion de fabrica
        //'8' - solicitar estado
        //fin de cadena = 03 hex
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ContrasteClick(TObject *Sender)
{
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'o');
```

```
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'a');
        TransmitCommChar(hComm, 'i');
        TransmitCommChar(hComm, '2');
        TransmitCommChar(hComm, StrToInt(ParametrosContraste -> Text) );
        TransmitCommChar(hComm, 03);
        //pwd=Kokai
        //comandos: '1' - para brillo, '2' - para contraste,
        //'3' - seleccionar entrada de video, '4' - frame rate, '5' - numero
de bits
        //'6' - resolucion, '7' - programar configuracion de fabrica
        //'8' - solicitar estado
        //fin de cadena = 03 hex
}
//---------------------------------------------------------------------------

void __fastcall TForm1::VideoClick(TObject *Sender)
{
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'o');
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'a');
        TransmitCommChar(hComm, 'i');
        TransmitCommChar(hComm, '3');
        TransmitCommChar(hComm, StrToInt(TipoVideo -> Text) );
        TransmitCommChar(hComm, 03);
        //pwd=Kokai
        //comandos: '1' - para brillo, '2' - para contraste,
        //'3' - seleccionar entrada de video, '4' - frame rate, '5' - numero
de bits
        //'6' - resolucion, '7' - programar configuracion de fabrica
        //'8' - solicitar estado
        //fin de cadena = 03 hex
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FRClick(TObject *Sender)
{
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'o');
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'a');
        TransmitCommChar(hComm, 'i');
        TransmitCommChar(hComm, '4');
        TransmitCommChar(hComm, StrToInt(FrameRate -> Text) );
        TransmitCommChar(hComm, 03);
        //pwd=Kokai
        //comandos: '1' - para brillo, '2' - para contraste,
        //'3' - seleccionar entrada de video, '4' - frame rate, '5' - numero
de bits
        //'6' - resolucion, '7' - programar configuracion de fabrica
        //'8' - solicitar estado
        //fin de cadena = 03 hex
}
//---------------------------------------------------------------------------

void __fastcall TForm1::BitsClick(TObject *Sender)
{
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'o');
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'a');
        TransmitCommChar(hComm, 'i');
        TransmitCommChar(hComm, '5');
        TransmitCommChar(hComm, StrToInt(NumeroBits -> Text) );
        TransmitCommChar(hComm, 03);
        //pwd=Kokai
        //comandos: '1' - para brillo, '2' - para contraste,
```

```
        //'3' - seleccionar entrada de video, '4' - frame rate, '5' - numero
de bits
        //'6' - resolucion, '7' - programar configuracion de fabrica
        //'8' - solicitar estado
        //fin de cadena = 03 hex
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ResClick(TObject *Sender)
{
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'o');
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'a');
        TransmitCommChar(hComm, 'i');
        TransmitCommChar(hComm, '6');
        TransmitCommChar(hComm, StrToInt(Resolucion -> Text) );
        TransmitCommChar(hComm, 03);
        //pwd=Kokai
        //comandos: '1' - para brillo, '2' - para contraste,
        //'3' - seleccionar entrada de video, '4' - frame rate, '5' - numero
de bits
        //'6' - resolucion, '7' - programar configuracion de fabrica
        //'8' - solicitar estado
        //fin de cadena = 03 hex
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FabricaClick(TObject *Sender)
{
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'o');
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'a');
        TransmitCommChar(hComm, 'i');
        TransmitCommChar(hComm, '7');
        TransmitCommChar(hComm, StrToInt(ConfigFabrica -> Text) );
        TransmitCommChar(hComm, 03);
        //pwd=Kokai
        //comandos: '1' - para brillo, '2' - para contraste,
        //'3' - seleccionar entrada de video, '4' - frame rate, '5' - numero
de bits
        //'6' - resolucion, '7' - programar configuracion de fabrica
        //'8' - solicitar estado
        //fin de cadena = 03 hex
}
//---------------------------------------------------------------------------

void __fastcall TForm1::EdoClick(TObject *Sender)
{
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'o');
        TransmitCommChar(hComm, 'k');
        TransmitCommChar(hComm, 'a');
        TransmitCommChar(hComm, 'i');
        TransmitCommChar(hComm, '8');
        TransmitCommChar(hComm, StrToInt(Estado -> Text) );
        TransmitCommChar(hComm, 03);
        //pwd=Kokai
        //comandos: '1' - para brillo, '2' - para contraste,
        //'3' - seleccionar entrada de video, '4' - frame rate, '5' - numero
de bits
        //'6' - resolucion, '7' - programar configuracion de fabrica
        //'8' - solicitar estado
        //fin de cadena = 03 hex

}
//---------------------------------------------------------------------------
```

Anexo B. Código para dispositivos maestro y esclavo en MPLAB

1. Código para el dispositivo maestro:

```
#include <pic.h>
#include "delay.h"
#include "i2c.h"

#define Slave 0xF0   // 0b 1010 0000       /* I2C Slave */
unsigned char dato;

void InicializarMicro()
{

       __CONFIG(0x3D72);   //hs,wd_off,POT_on,BRWN_on 3f71
       INTCON=0xA0; //1010 0000  1110 0000
       OPTION=0x81;
       ADCON1=0x06;
       TRISA=0;
       PEIE = 1;
}

void interrupt Rutina()        //rutina de interrupcion destello del led
{                              //y base de tiempo a 1ms
       static int blink = 0;

       if(T0IF)
       {
      T0IF=0;
      if(blink==500)
      {
         blink=0;
         RA4=!RA4;
      }
      blink++;
    }
}

void main(void)
{

       InicializarMicro();

TRISB=0;        // ORIGINALMENTE use a led on RB0 - set as output
PORTB=0;
RB0=0;

dato=0b10111011;

/* initialize i2c */
    SSPMode(MASTER_MODE);
    SSPEN = 1;
    CKP = 1;

while(1)
       {
       char continua = 0;
       i2c_Restart();
       i2c_SendAddress(Slave,I2C_WRITE);

       continua = i2c_ReadAcknowledge();

       if(continua == 0 )
        {
             i2c_PutByte(dato);
        }
```

```
        continua = i2c_ReadAcknowledge();

        i2c_Stop();

        DelayMs(3);
    }
}
```

2. Código para el dispositivo esclavo:

```
/* I2C test program that writes & reads data to an I2C EEPROM device. */

#include <pic.h>
#include "delay.h"
#include "i2c.h"

#define Slave 0xF0   /* I2C Slave */
unsigned char dato, direccion;
bit escritura = 0;

void InicializarMicro()
{

        __CONFIG(0x3F72);   //0x3D72 Para program, 0x3F72 para debugger
        INTCON=0xA0; //1010 0000  1110 0000
        OPTION=0x01; //1000 0001
        ADCON1= 6;
        TRISA=0;
        TRISB=0;
        PORTB= 0xFF;
}

void InicializarSlave()
{
        SSPSTAT = 0x80;
        TRISC = 0x18;       // seteo las SDA y SCL como entrada, el resto de las
patas del // TRISC pueden tener otros valores.
        SSPCON  = 0b00000110;      // seteo como SLAVE
        SSPADD  = Slave;                  // Dirección del I2C
        SSPEN    = 1;              // módulo SSP habilitado.
        CKP = 1;                   // Libera el clock
        SSPIE = PEIE = 1;                        //         Para         habilitar
interrupciones, esta en 0 por defecto.
        escritura = 0;

}

void interrupt Rutina()         //rutina de interrupcion destello del led
{                               //y base de tiempo a 1ms
        static int blink = 0;
        if(T0IF)
        {
       T0IF=0;
       if(blink==500)
       {
          blink=0;
          RA4=!RA4;
       }
       blink++;
    }

        else if(SSPIF)
        {
              SSPIF =0;
       if(escritura == 1)
       {
              dato = SSPBUF;
```

```
            escritura = 0;
      }

        else if(escritura == 0)
        {
            direccion = SSPBUF;
            escritura = 1;
        }

    }

}

void main(void)
{

      InicializarMicro();
      InicializarSlave();


while(1)
      {
            PORTB = dato;
    }
}
```

Anexo C. Algoritmo de Interpolación Bilineal

```
function g = Myimgrescale(data, newsize)
oldsize = size(data);

% Checa el tipo de dato en el arreglo

oldtype = [];
if ~isa(data, 'double')
	if isa(data, 'uint8')
		oldtype = class(data);
		data = double(data);
	else
		error('Only double or uint8 data types supported.');
	end
end

% Checa si los tres planos de color son dados o no

if length(oldsize)>2
	if oldsize(3) == 3
		% RGB data present, otherwise treat as gray scale data.
		RGB = 1;
	else
		error('Format not supported.');
	end
else
	RGB = 0;
end

% Cambia los factores de escala

factor = (oldsize(1:2)-1)./(newsize-1);

% Crea un grid nuevo

u = 0:newsize(1)-1;
v = 0:newsize(2)-1;

% Transforma los vectores de la rejilla al tamaño de la ultima rejilla
u = u.*factor(1) + 1;
v = v.*factor(2) + 1;

% Cambia la ubicacion de cada punto relative al mas cercano

% Vecino de la imagen original

% Hace la interpolación elemento por elemento

if ~RGB
	g=data(floor(u), floor(v));
else
	% replicate grid to all three dimensions
	g=data(floor(u), floor(v), :);
end

% Reconvierte al antiguo formato los datos, si es necesario

if ~isempty(oldtype)
	switch oldtype
	 case 'uint8'
		g = uint8(g);
	end
end
```

Anexo D. Algoritmo para Eliminación de Píxeles

```
% Obtiene una matriz dividida entre cualquier numero entero, 8 bits

function [y]=half(x)
n=size(x);
if n(1)==1;

%Se define la matriz será reducida a la mitad

    [y]=x(1:2:length(x));
else
    [y]=x(1:2:n(1),1:2:n(2),1:1:n(3));
end
```

Anexo E. Algoritmos para obtener las componentes R,G y B de una imagen

Componente R

```
R(a);
function [y]=half(x)
n=size(x);
if n(1)==1;
    [y]=x(1:2:length(x));
else
    [y]=x(1:4:n(1),1:4:n(2),1);
End
```

Componente G

```
G(a);
function [y]=half(x)
n=size(x);
if n(1)==1;
    [y]=x(1:2:length(x));
else
    [y]=x(1:4:n(1),1:4:n(2),2);
End
```

Componente B

```
B(a);
function [y]=half(x)
n=size(x);
if n(1)==1;
    [y]=x(1:2:length(x));
else
    [y]=x(1:4:n(1),1:4:n(2),3);
End
```

Anexo F. Algoritmo para pasar los arreglos a archivos *.dat

```
Bandas ()
b=B(a);
Sub. = sise (b);
lineas_b = sz_b(1);
columnas_b=sz_b(2);
cont=1;
for i=1:lineas_b
    for j=1:columnas_b
    salida(cont,1)=b(i,j);

    cont=cont+1;
    end
end
fprintf('0x%x\n',salida);

bandasG()
g=G(a);
sz_g = size(g);
lineas_g = sz_g(1);
columnas_g=sz_g(2);
cont=1;
for i=1:lineas_g
    for j=1:columnas_g
    salida(cont,1)=g(i,j);
    cont=cont+1;
    end
end
fprintf('0x%x\n',salida);

bandasB()
b=B(a);
sz_b = size(b);
lineas_b = sz_b(1);
columnas_b=sz_b(2);
cont=1;
for i=1:lineas_b
    for j=1:columnas_b
    salida(cont,1)=b(i,j);
    cont=cont+1;
    end
end
fprintf('0x%x\n',salida);
```

Anexo G. Código del programa de eliminación de píxeles en el DSP

```
/*
 *  Copyright 2003 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 */
/* "@(#) DSP/BIOS 4.90.270 01-05-04 (bios,dsk6416-f06)" */
/*************************************************************************/
/*
*//************************************************************************
/

#include <stdio.h>

#include "volume.h"

/* Global declarations */
int inp_buffer[BUFSIZE];       /* processing data buffers */
int rojo_buffer[3328];
int rojoh_buffer[1664]; //1664
int rojov_buffer[3328];   //832

int verde_buffer[3328];
int verdeh_buffer[1664];
int verdev_buffer[3328];

int azul_buffer[3328];
int azulh_buffer[1664];
int azulv_buffer[3328];
int out_buffer[1664];
int out_buff[3328];

int gain = MINGAIN;                     /* volume control variable */
unsigned int processingLoad = BASELOAD;  /* processing routine load value */

struct PARMS str =
{
    2934,
    9432,
    213,
    9432,
    &str
};

/* Functions */
extern void load(unsigned int loadValue);
static int processing(int *inputR, int *outputR,int *outbuffR,int *inputG, int
*outputG,int *outbuffG,int *inputB, int *outputB,int *outbuffB);
//static int processing(int *inputG, int *outputG,int *outbuffG);
static void dataIO(void);


/*
 * ======== main ========
 */
void main()
{
    int *inputR = &rojo_buffer[0];
    int *outputR = &rojoh_buffer[0];
    int *outbuffR = &rojov_buffer[0];

       int *inputG = &verde_buffer[0];
    int *outputG = &verdeh_buffer[0];
    int *outbuffG = &verdev_buffer[0];
```

```
      int *inputB = &azul_buffer[0];
    int *outputB = &azulh_buffer[0];
    int *outbuffB = &azulv_buffer[0];

    puts("volume example started\n");

    /* loop forever */
    while(TRUE)
    {
        /*
         *  Read input data using a probe-point connected to a host file.
         *  Write output data to a graph connected through a probe-point.
         */
          dataIO();
          dataIO();
          dataIO();

        #ifdef FILEIO
        puts("begin processing")        /* deliberate syntax error */
        #endif

        /* apply gain */
        processing(inputR,  outputR,outbuffR,inputG,  outputG,outbuffG,inputB,
outputB,outbuffB);
             //processing(inputG, outputG,outbuffG);
    }
}

/*
 *  ======== processing ========
 *
 * FUNCTION: apply signal processing transform to input signal.
 *
 * PARAMETERS: address of input and output buffers.
 *
 * RETURN VALUE: TRUE.
 */

static int processing(int *inputR, int *outputR, int *outbuffR,int *inputG,
int *outputG, int *outbuffG,int *inputB, int *outputB, int *outbuffB)
//static int processing(int *inputG, int *outputG, int *outbuffG)
{

    int y=1;
    int j=1;
    int size1 = 3328;
     int size2 = 3328;
      int size3 = 3328;
       int p;

    while(size1--){
        *outputR++ = *inputR++;
             *inputR++;

    }
       while(size2--){


    *outputG++ = *inputG++;
             *inputG++;

    }
       while(size3--){


    *outputB++ = *inputB++;
             *inputB++;
```

```
    }

outputR =&rojoh_buffer[0];
outputG =&verdeh_buffer[0];
outputB =&azulh_buffer[0];

for( j=1;j<=52;j++){

             for( y=1;y<=32;y++){
             *outbuffR++ = *outputR++;
                    *outbuffG++ = *outputG++;
                    *outbuffB++ = *outputB++;

         }
       for(p=1;p<=32;p++){
       outputR++;
             outputG++;
                    outputB++;
    }

    }

 load(processingLoad);

    return(TRUE);
}

/*
 *  ======== dataIO ========
 *
 * FUNCTION: read input signal and write processed output signal.
 *
 * PARAMETERS: none.
 *
 * RETURN VALUE: none.
 */
static void dataIO()
{
    /* do data I/O */

    return ;
}
```

Anexo H. Código para Redimensionamiento en el FPGA

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
ENTITY video IS
  PORT(
        -- Entrada:
        CLK: in STD_LOGIC;
        RESET: in STD_LOGIC;
        VSINC: in STD_LOGIC;
        HSINC: in STD_LOGIC;
        ROJO: in STD_LOGIC_VECTOR (7 downto 0);
        VERDE: in STD_LOGIC_VECTOR (7 downto 0);
        AZUL: in STD_LOGIC_VECTOR (7 downto 0);

        -- Salidas:
        CLK2: out STD_LOGIC;
        VSINC2: out STD_LOGIC;
        HSINC2: out STD_LOGIC;
        ROJO2: out STD_LOGIC_VECTOR (7 downto 0);
        VERDE2: out STD_LOGIC_VECTOR(7 downto 0);
        AZUL2:     out STD_LOGIC_VECTOR(7 downto 0);
        -- Bandera fin de pixel:
        BPIX: out STD_LOGIC;
        -- Bandera fin de linea:
        BLIN: out STD_LOGIC
        );
  END video;

ARCHITECTURE myvideo OF video IS

-- BANDERA FIN DE LINEA
SIGNAL CLIN: STD_LOGIC_VECTOR(7 downto 0);
SIGNAL CPIX: STD_LOGIC_VECTOR(2 downto 0);

begin
contalin: process (HSINC, RESET)
begin
      if RESET='1' then
            CLIN<="00000000";
            BLIN<='0';
      elsif HSINC='1' and HSINC'EVENT then
            if CLIN<="11111111" then
                  BLIN<='1';
            else
                  CLIN<=CLIN+1;
            end if;
      end if;
end process contalin;

-- BANDERA FIN DE PIXEL

contapix: process (CLK, RESET)
begin
      if RESET='1' then
            CPIX<="000";
            BPIX<='0';
      elsif CLK='1' and CLK'EVENT then
            if
             CPIX<="111" then
                  BPIX<='1';
            else
                  CPIX<=CPIX+1;
            end if;
```

```
      end if;
end process contapix;

-- REDUCCION DE CUADROS A LA MITAD
redux: process (CLK, RESET, VSINC, HSINC, ROJO, VERDE, AZUL)
begin
      if RESET='0' then
            CLK2<=CLK;
            VSINC2<=VSINC;
            HSINC2<=HSINC;
            if CLK='1' and clk'event then
                  ROJO2<=ROJO;
                  AZUL2<=AZUL;
                  VERDE2<=VERDE;
            end if;
      else
            CLK2<='0';
            VSINC2<='0';
            HSINC2<='0';
            ROJO2<="00000000";
            AZUL2<="00000000";
            VERDE2<="00000000";
      end if;
end process redux;
END myvideo;
```