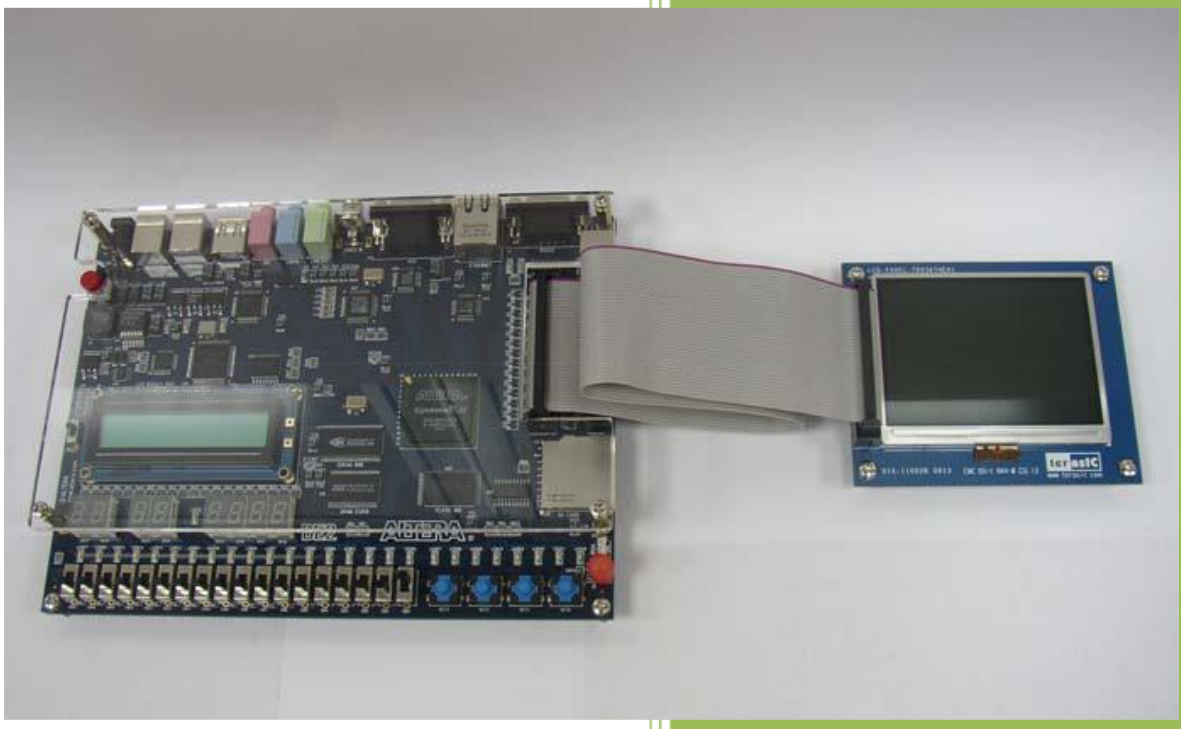
INSTITUTO TECNOLOGICO DE TUXTLA GUTIERREZ

2009

NOMBRE DEL PROYECTO: DISEÑO DE LA ETAPA DE CONTROL DE UNA PANTALLA LCD DE MATRIZ DE PUNTOS A COLOR USANDO FPGA EN UN SISTEMA EMBEBIDO.



AUTOR: SERGIO GOMEZ LOPEZ

REVISOR: ING. ALVARO HERNANDEZ SOL

02/11/2009

Índice

CAPITULO 1

1.1 INTRODUCCIÓN

Una pantalla de cristal líquido o LCD (acrónimo del inglés Liquid Crystal Display) es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora. A menudo se utiliza en dispositivos electrónicos de pilas, ya que utiliza cantidades muy pequeñas de energía eléctrica.

La implementación de la etapa de control de una pantalla TRDB_LCM con circuitos integrados comerciales estándares como compuertas or, and, not, flip-flops, multiplexores, contadores, etc. sería muy difícil. Además de que tendrían que utilizarse más de un componente de cada uno, habría que interconectarlos entre sí para generar las señales de control de la pantalla. Este método resultaría al final muy complejo de implementar.

Una manera de resolver este problema es la utilización de lenguaje HDL como herramienta de trabajo, ya que la implementación de esta tecnología facilita el diseño de los componentes necesarios de manera facil . En lenguaje HDL se pueden diseñar compuertas and, or, contadores, filp-flops, memorias. La ventaja de utilizar lenguaje HDL es que se puede simular el diseño antes de implementarlo, esto permite hacer los cambios necesarios a nivel de software en caso de fallas, el cual ahorrara tiempo y esfuerzo antes de implementarlo.

La utilización de un FPGA y lenguaje HDL para el diseño de la etapa de control de una pantalla LCD de matriz de puntos a color en un sistema embebido, es más viable de implementar que con circuitos integrados estándares.

1.2 INFORMACIÓN GENERAL DE LA INSTITUCIÓN O EMPRESA DONDE SE DESARROLLO El PROYECTO

Este proyecto se desarrollo en el INSTITUTO TECNOLOGICO DE TUXTLA GUTIERREZ donde actualmente se imparte la carrera de ingeniería electrónica con el objetivo de formar profesionistas con capacidad creativa, emprendedora, de análisis y liderazgo, que realicen actividades de diseño, innovación, adaptación y transferencia de tecnología para resolver problemas en forma competitiva y atender las necesidades de su entorno con una conciencia social y un compromiso con el desarrollo sustentable. Actualmente en el departamento de posgrados, edificio Z, donde se imparte la maestría en Mecatrónica se forman grupos de trabajo con la finalidad de desarrollar nuevos proyectos

con alumnos que cursan la carrera en ingeniería electrónica. Este proyecto se realiza con la colaboración del Doctor Madain Pérez Patricio quien integra el departamento de posgrado.

El departamento de posgrado cuenta con un kit que contiene un FPGA cyclone II de la marca Altera DE2 y una pantalla TRDB LCM de la marca Terasic el cual facilito para la realización de este proyecto.

El trabajo de realizo en el laboratorio de electrónica que se encuentra en el edificio I del instituto tecnológico de Tuxtla Gutiérrez.

1.3. ÁREA ESPECÍFICA RELACIONADA DIRECTAMENTE CON EL PROYECTO

El presente trabajo está directamente al área de diseño de sistemas digitales a nivel de software. Más directamente con las materias de sistemas digitales I y II que se imparten en el Instituto Tecnológico de Tuxtla Gutiérrez y en el departamento de posgrado como proyectos de investigación en el área de visión artificial.

1.4 ANTECEDENTES

El diseño de sistemas digitales con software es un campo que busca explotarse en el ITTG, existen trabajos donde se controla un monitor VGA, y se programan sus señales de control y sincronía para desplegar imágenes, procesamiento de imágenes, procesamiento de video, visión artificial, además existen trabajos similares con tarjetas de otras compañías lideres en la fabricación de FPGA (como son las tarjetas Virtex), utilizando lenguajes de programación a nivel de software Verilog y VHDL.

1.5. PLANTEAMIENTO DEL PROBLEMA

¿Diseñar los módulos para etapa de control de la pantalla TRBD-LCM con lenguaje Verilog es más rentable en comparación con circuitos digitales estándares en el mercado?

1.6. NOMBRE DEL PROYECTO

Diseño de la etapa de control de una pantalla LCD de matriz de puntos a color usando FPGA en un sistema embebido.

1.7 OBJETIVOS

OBJETIVOS GENERALES

- ✓ Diseñar en lenguaje Verilog HDL componentes and, or, contadores, flip-flops, memorias, para la implementación de la etapa de control de la pantalla TRDB_LCM.

OBJETIVOS ESPECIFICOS

- ✓ Diseñar la circuitería digital para la generación de las señales de control requerida por la pantalla LCD usando Verilog.
- ✓ Diseñar la circuitería digital para la generación de las señales de control requeridas por la memoria RAM disponible en el kit.
- ✓ Generar los datos requeridos para el despliegue de imágenes digitales.

1.8 JUSTIFICACION

La implementación de la etapa de control para la pantalla TRDB_LCM permitirá posteriormente desarrollar proyectos de mayor complejidad, como visión artificial, procesamiento de imágenes, control de un vehículo teledirigido y permitirá a nuestra institución estar a la vanguardia y así poder competir con otras instituciones que utilicen estas mismas tecnologías.

1.9 ALCANCES Y LIMITACIONES

Al generar la circuitería digital para la comunicación entre la pantalla LCD y la cámara digital nos permitirá posteriormente desarrollar un sistema más complejo de procesamiento de imágenes, visión artificial, así como un vehículo teledirigido utilizando un FPGA y Verilog HDL.

La pantalla solo podrá desplegar imágenes sencillas definidas por el programador, los cuales serán rectángulos, cuadros, líneas verticales, horizontales, que podrán ser en diferentes colores.

CAPITULO 2

FUNDAMENTO TEÓRICO

2.1 IMAGEN DIGITAL

Al igual que con otros muchos elementos, las imágenes pueden ser analógica o digital. Las imágenes analógicas se crean a partir de dos factores: brillo (luminosidad) y color (que tiene tres subfactores: rojo, verde, azul).

Las imágenes digitales, por el contrario, no necesitan toda esta información, porque se crean a partir de unos datos básicos y el resto de calcula a partir de éstos. Esto hace que cada vez se tienda más a elementos digitales en cada uno de los procesos relacionados con el tratamiento de las imágenes.

El proceso de digitalización consiste en dividir la imagen continua a(x, y) en N filas y M columnas, donde cada uno de los M x N elementos generados se denomina píxel. El valor asignado a cada uno de esos píxeles puede ser considerado como la evaluación de la señal física que choca contra el sensor de dos dimensiones que captura la imagen, la cual es una función de diferentes variables, incluyendo la profundidad, el color y el tiempo.

Hay diferentes valores para los parámetros M y N que usualmente vienen determinados por los estándares de video, por los requerimientos de los algoritmos o por el deseo de hacer los circuitos digitales un poco más simples. Los valores comúnmente encontrados para el número de filas (N) son: 256, 512, 525, 625, 1024 y 1035, y para el número de columnas (M): 256, 512, 768, 1024 y 1320. Cada **imagen** individual se considera como la suma aditiva de colores siendo los tres básicos cuando hablamos de televisión, el rojo (RED), el verde (GREEN) y el azul (BLUE). Por eso los dispositivos que tratan con este tipo de imágenes se denominan **dispositivos RGB**.

Como ya hemos visto, tenemos imágenes que se consiguen a través de dispositivos analógicos (cine, cámaras de fotos, televisión) y otras, a través de técnicas digitales (ordenadores personales, escáneres).

2.1.1. COMPOSICION DE UNA IMAGEN DIGITAL

En el entorno digital, las imágenes están formadas por elementos mínimos con unas determinadas características físicas: colores (rojo, verde y azul) y brillo. Cada uno de estos puntos de la imagen se llama **píxel** y cuanto mayor sea el número de ellos que haya en un determinado área de la imagen, mayor será la **resolución** figura 2.1.

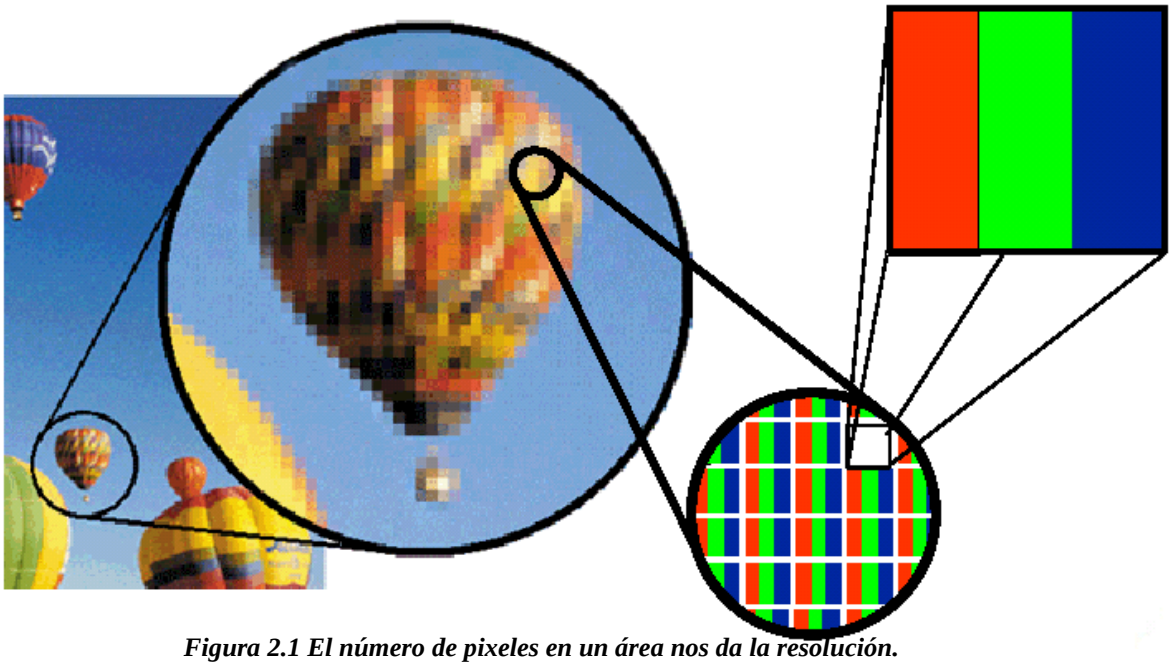


Figura 2.1 El número de pixeles en un área nos da la resolución.

Para cada uno de estos píxeles, se tiene que almacenar cierta información que permita recuperarlo después. Lo más habitual es aprovechar la descomposición de los colores para almacenar la información completa del píxel teniendo en cuenta el valor para cada una de los componentes de color. En función del número de bits que se le asigne a cada uno de estos componentes (generalmente es el mismo para las tres), se tiene un mayor número de colores para representar. Al conjunto de colores representables, se le suele denominar paleta de colores.

Generalmente, se habla de "**color real**" cuando tenemos una paleta de 16'8 millones de colores, para lo que necesitamos manejar 24/32 bits para cada uno de los píxeles de nuestra imagen (8 bits para cada una de las componentes). Sin embargo, dada la enorme cantidad de información que esto requiere, se suelen utilizar otras aproximaciones, que se centran básicamente en la reducción del número de bits que le dedicamos a cada una de las componentes y entre las que hay que destacar la que asigna 5:5:5 (ó 5:6:5) bits para cada una de ellas (en total 15/16 bits) y que nos permite alcanzar paletas de 65 mil colores.

Merece una mención especial una aproximación que reduce enormemente el número de información a manejar (únicamente requiere 8 bits por píxel, por lo que tiene una paleta de 256 colores) a costa de añadir una serie de cálculos adicionales y que ha recibido el nombre de **pseudo color** figura 2.2.

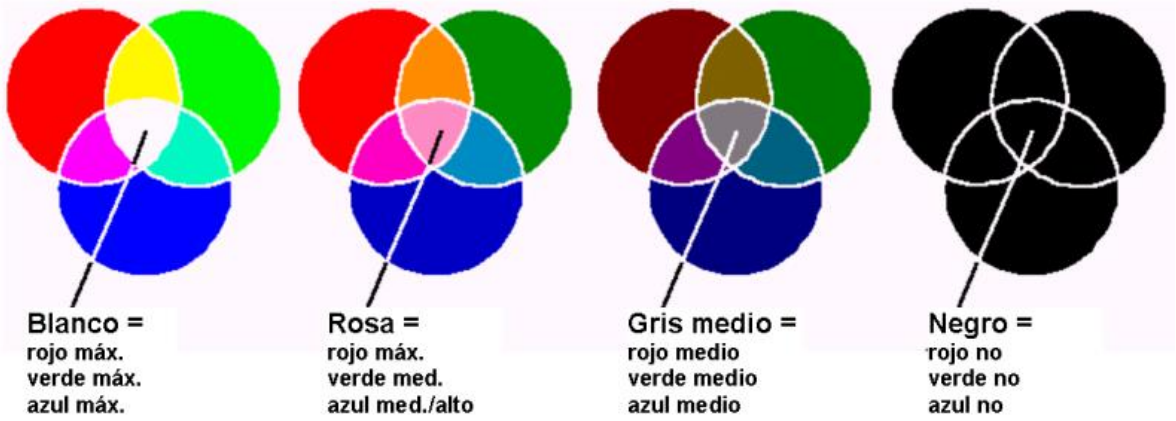


Figura 2.2 Muestra como se obtiene colores a partir de los colores RGB.

El tamaño requerido por una imagen (en bits) estará directamente relacionado con el número de píxeles que haya (resolución) y por la cantidad de información que necesite cada uno:

Tamaño de la Imagen = Número total de píxeles x Número de bits por píxel

Por lo tanto, a la hora de decidir qué formatos se va a elegir (resolución, paleta de colores), tiene que tener en cuenta el sistema del que se dispone, para ver si va a ser capaz de manejar las cantidades de información requeridas, ya no sólo espacialmente, sino temporalmente.

2.1.2. RESOLUCIÓN DE UNA IMAGEN DIGITAL

La resolución en los sistemas de televisión se especifica en términos del *número de líneas*. Este parámetro se determina observando un *test* de un patrón que consiste en alternar líneas blancas y negras que se ponen muy cerca entre sí; el par de líneas con el espaciado más cercano que pueda distinguirse como líneas separadas determina la resolución. Las líneas que pueden ser extrapoladas de un lado a otro de la pantalla con un ancho igual a la altura de una imagen son las *líneas de resolución*.

La resolución en los formatos de los ordenadores se especifica normalmente en función del número visible de píxeles en las dimensiones horizontal y vertical. Por ejemplo, una señal en formato VGA tiene 640 píxeles visibles en la dirección horizontal y 480 píxeles visibles en la dirección vertical, mientras que una señal en formato XGA tiene 1024 píxeles visibles en la dirección horizontal y 768 píxeles visibles en la dirección vertical.

A la hora de hablar de estas resoluciones, se nombran dando su número de píxeles en horizontal y en vertical (o con el nombre del estándar que representa). Así, algunas de las más comunes son: 648x480 (VGA), 800x600 (SVGA), 1280x1024 (SXGA), etc. Haciéndolo así, se consigue abstraer la idea de resolución y hacerla independiente del dispositivo que se esté usando para representar la imagen, pues únicamente hace referencia al número de píxeles que se usará en cada una de las dos dimensiones.

2.1.3. FORMATOS NTSC, PAL Y SECAM

Hay muchos tipos diferentes de señales de video, se puede dividir en dos grandes grupos: de televisión y de computación. **Los formatos de las señales de televisión** varían de país a país y se distinguen tres subgrupos:

- En los Estados Unidos y Japón se usa el formato **NTSC**, que significa *National Television Systems Committee*, que es el nombre de la organización que desarrolló el estándar.
- El formato más común en Europa es el **PAL** (*Phase Alternating Line*), que se desarrolló con base en el NTSC, y contiene algunas mejoras sobre este último.
- El formato **SECAM** se usa en Francia, y significa *SEquential Coleur Avec Memoire* (con memoria).

Existen cerca de 15 subformatos contenidos en estos tres generales, donde cada uno, normalmente es incompatible con los demás. Aunque todos ellos utilizan el mismo sistema de barrido y representan el color con un tipo de modulación en fase, difieren específicamente en las frecuencias de barrido, el número de líneas y la técnica de modulación del color, entre otras. Los diversos **sistemas de computación** como VGA, XGA o UXGA también difieren sustancialmente, donde la principal diferencia son las frecuencias de barrido. Estas diferencias no causan mucha preocupación, ya que la mayoría de los equipos de computación están siendo diseñados para soportar diferentes frecuencias de barrido.

En tabla 2.1 se puede ver las frecuencias y resoluciones típicas para los principales formatos de televisión y computación.

Tabla 2.1 características estándares de algunos sistemas.

FORMATO DE VIDEO	NTSC	PAL	HDTV/SDTV	VGA	XGA
Descripción	Formato de televisión para Norte América, Centro América, parte de sur América y Japón	Formato de televisión para la mayoría de Europa y Sur América	High Definition/ Standard Definition Digital Television Format	Video Graphics Array (PC)	Extended Graphics Array (PC)
Resolución Vertical (líneas visible por *frame*)	Aprox. 480 (525 líneas en total)	Aprox. 575 (625 líneas en total)	18 formatos diferentes: 1080 o 720 o 480	480	768
Resolución Horizontal (píxeles visibles por línea)	Determino por el ancho de banda, rangos desde 320 hasta 650	Determino por el ancho de banda, rangos desde 320 hasta 720	18 formatos diferentes: 1920 o 704 o 640	640	1024
Frecuencia Horizontal (KHz)	15.734	15.625	33.75-45	31.5	60
Frecuencia Vertical (Hz)	29.97	25	30-60	60-80	60-80
Frecuencia más elevada (MHz)	4.2	5.5	25	15.3	40.7

2.1.4. INTERFACES DE VIDEO

Existen tres niveles básicos de señales de interfaz de banda base. En el orden de incremento de la calidad son: **compuesto** (o CVBS), que utiliza un par de cables; **Y/C**, que emplea dos pares de cable; y **por componentes**, que emplea tres pares de cables; donde cada par de cables consiste en una señal y una tierra.

Estas tres interfaces difieren en sus niveles de combinación de la información (codificación). Usualmente a mayor codificación se degrada la calidad, pero permite que la señal se transmita con menos cables y tenga menos requisitos. La interfaz por componentes tiene la menor codificación, mientras que la interfaz compuesta posee la mayor.

INTERFAZ COMPUESTA O CVBS

Las señales de esta interfaz son las usadas en el video analógico. El **video compuesto** también se llama **CVBS (*Color, Video, Blanking y Sync*)**, o señal banda base de video compuesto. Esta interfaz combina la información de brillo (luminancia), la información de color (croma) y las señales de sincronización en un solo cable. El conector de estos cables es usualmente una toma RCA. Cada línea esta formada por el *video activo* y el intervalo de *blanking*. El video activo es la parte de la señal que contiene la información del brillo (luminancia) y color (croma) de la imagen. La luminancia es la amplitud instantánea de cualquier punto en el tiempo. La amplitud se mide en términos de una unidad llamada IRE, que es una unidad arbitraria donde 140 IRE son iguales a un voltio pico a pico. El voltaje durante la parte del video activo produce una imagen con brillo totalmente blanco para esta línea, mientras que la parte del intervalo de *blanking* se mostraría como negro, por lo tanto no se vería en la pantalla.

En la siguiente figura2.3 se muestra una señal de una línea de video que presenta una barra de diferentes luminosidades, y la relación de estas en unidades IRE.

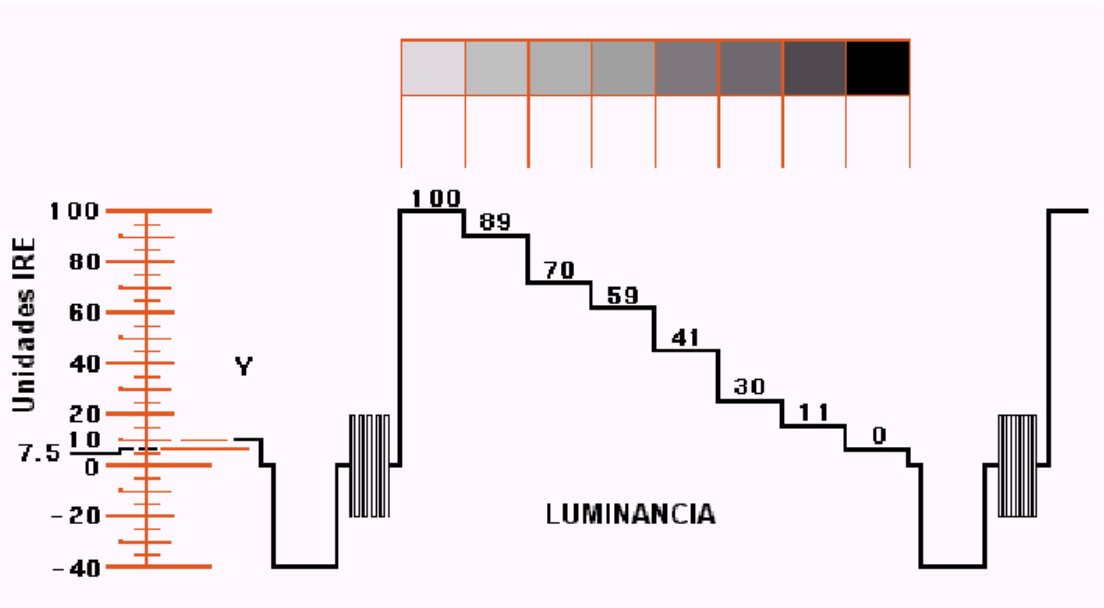


Figura 2.3 Línea de video con diferentes unidades de IRE.

La señal de luminancia es suficiente para obtener una imagen en escala de grises. La información de color se añade en la parte superior de la señal de luminancia, y es una onda senoidal con los colores especificados por la diferencia de fase entre ésta y la fase de la señal de referencia llamada *burst*.

En figura 2.4 se muestran los principales colores y el *burst* con su correspondiente ángulo de fase.

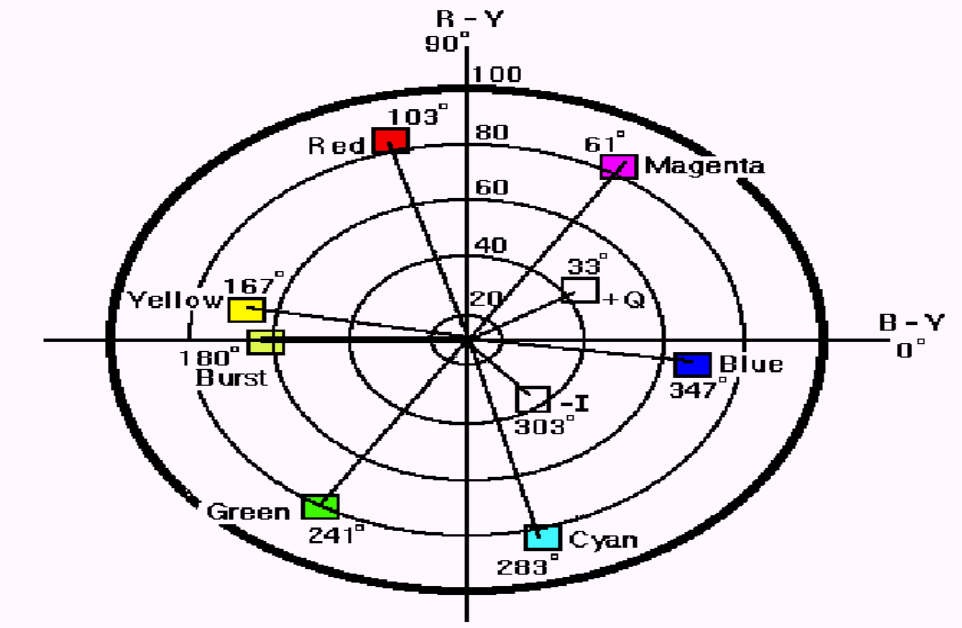


Figura 2.4 Principales colores y el burst

La amplitud de la modulación es proporcional a la cantidad de color (o saturación), y la información de la fase indica el tono (o *hue*) del color.
La parte del *blanking* contiene el pulso de sincronización horizontal, así como la referencia del color *(burst*), que está ubicado justo después del flanco de subida del pulso de sincronización horizontal. Es importante notar que la parte de la señal correspondiente al intervalo de *blanking* horizontal está ubicada en el tiempo en el cual la señal no es visible, por lo que no se observa en la pantalla.

INTERFACES Y/C

La señal Y/C es una señal de video un poco menos codificada. A menudo se llama incorrectamente *S-video*, pero este término en la actualidad se emplea para referirse al formato de grabación de cintas para video, y no a una señal de interfaz. El brillo (luminancia) que es la señal Y, y el color (croma), la señal C, se llevan en dos pares de cables diferentes. El conector es usualmente un *mini DIN*, que parece una versión pequeña de un conector de teclado.

INTERFAZ POR COMPONENTES

Con estas interfaces se logra el más alto rendimiento, debido a que tienen la menor codificación, pues las señales están casi en su formato natural. Siempre emplean tres pares de cables que son típicamente cualquiera de estos dos: un formato que emplea una señal para la luminancia (Y) y dos señales diferenciales de color (R-Y, B-Y); o emplea una señal para el color rojo (R), otra para el verde (G) y otra para el azul (B), el cual comúnmente es llamado RGB.

Los formatos RGB casi siempre se emplean en aplicaciones de computación, mientras que los formatos diferenciales de color son generalmente usados en aplicaciones de televisión.

En los formatos diferenciales de color, la señal Y contiene la luminancia y la información de sincronización, y las señales diferenciales de color contienen el rojo (R) menos la señal Y, y el azul (B) menos la señal Y. La teoría detrás de estas combinaciones es que cada componente de color base (R, G o B) puede ser obtenida de estas señales diferenciales.

Existen algunas variaciones de estas interfaces diferenciales, como:

- ⇨ Y, B-Y, R-Y: Señales de luminancia y diferenciales de color.
- ⇨ Y, Pr, Pb: Pr y Pb son versiones escaladas de B-Y y R-Y. Comúnmente se encuentran en equipos de alto rendimiento.
- ⇨ Y, Cr, Cb: Señal digital equivalente a Y, Pr, Pb. Algunas veces se usan incorrectamente en lugar de Y, Pr, Pb.
- ⇨ Y, U, V: No es una interfaz estándar. Estas son intermediarias, entre las señales en cuadratura usadas en la formación de video compuesto y señales Y/C.

INTERFACES PARA ORDENADORES

Prácticamente todas las interfaces de ordenador emplean el formato RGB. La información de la imagen se transporta separadamente en las tres componentes bases de rojo, verde y azul. Además, la información de sincronización horizontal (H) y sincronización vertical (V) se lleva separadamente en dos señales. Las cinco señales, R, G, B, H y V, se llevan en un cable que está compuesto de un manojo de cables blindados; y el conector es casi siempre un tipo D de 15 pines. En algunas ocasiones, la información de sincronización horizontal y vertical se combina con una de las señales RGB, comúnmente la señal del verde, y a esto se le ha llamado *sync on green*. En raras ocasiones se encuentra la información de sincronización mezclada con las señales del rojo o el azul.

2.1.5. VÍDEO DIGITAL

En realidad, el movimiento de las imágenes es una "**ilusión**". Lo que ocurre realmente es que las imágenes fijas pasan a una velocidad muy rápida dando la impresión que se mueven. Este efecto tiene lugar porque las imágenes son retenidas por el ojo humano, que las mantiene durante un tiempo y de esta forma, cuando llega la segunda, todavía se tiene la anterior, por lo que realmente se están percibiéndolas como si se estuviesen produciendo de manera continúa.

Para una buena calidad de imagen en movimiento, se necesita una velocidad mínima de **24 imágenes por segundo**. Si se reproduce a menor velocidad, será capaz de percibir el parpadeo de las imágenes y, por encima, estará gastando inútilmente recursos, pues nuestro ojo no va a ser capaz de percibir más de 24. Por esto, la televisión analógica Europea emite 25 imágenes por segundo (en la Americana son 30 imágenes por segundo). Una imagen se dibuja en un televisor o en una pantalla de un ordenador pasando una señal eléctrica horizontalmente a través de la pantalla (una línea a la vez). La amplitud de la señal en función del tiempo representa el brillo instantáneo en ese punto físico de la pantalla. Hay dos formas de hacer estos barridos: entrelazado y progresivo.

En el **barrido entrelazado**, divide la imagen en dos campos, reproduciendo por separado y de manera consecutiva, primero las líneas impares y luego las pares, antes de pasar a la siguiente imagen. Por el contrario, en el **barrido progresivo**, reproduce línea a línea de manera secuencial la imagen de manera completa antes de pasar a la siguiente. Las diferencias entre ambos se ven claramente en la siguiente figura.

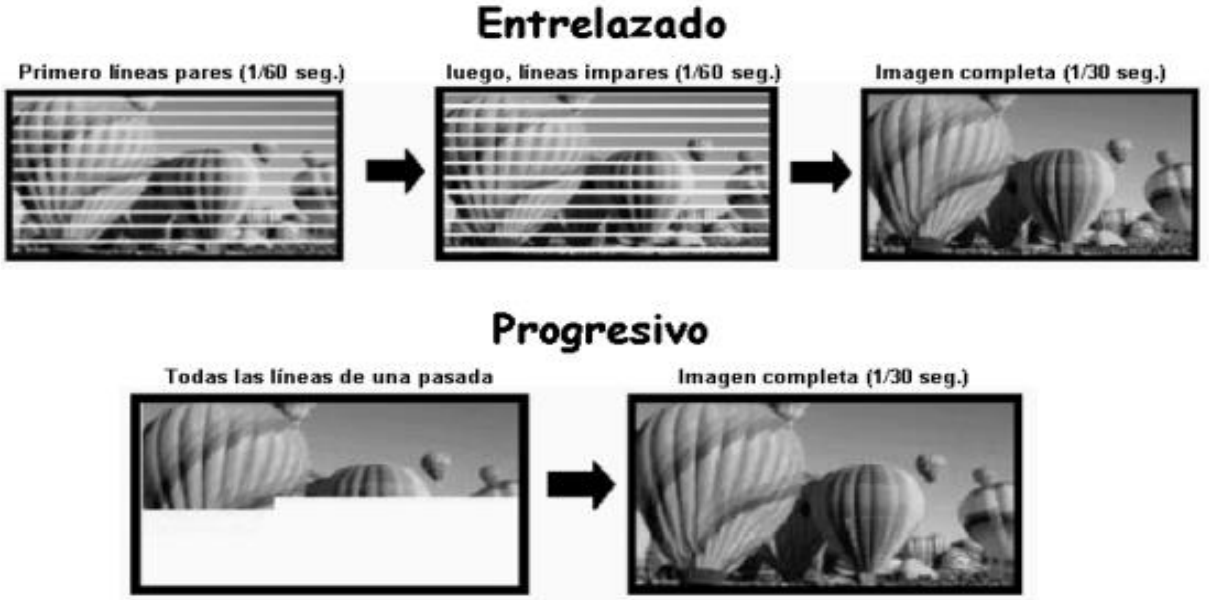


Figura 2.5 Formas de barridos: entrelazado y progresivo.

2.2 MONITORES

TEORÍA DE LOS CRT (CATHODE-RAY TUBE).

Un monitor VGA básicamente se controla con cinco señales básicas: R, G, B, V_S, H_s, las primeras: R, G, B, sirven para seleccionar el color en la pantalla y son de naturaleza analógica entre 0 y 0.7 V, las señales H_S y V_S son de sincronización horizontal y vertical respectivamente.

Es el enlace visual del usuario con los programas que ejecuta un sistema. Una forma muy usada para el despliegue de imágenes en la pantalla es un monitor de tubo de rayos catódicos (CRT, Cathode-Ray Tube). El CRT es básicamente, un gran tubo catódico al vacío en forma de botella. Por la parte trasera del tubo se encuentra un cañón de electrones el cual dispara un haz de electrones, cuando éstos hacen contacto con la parte interna de la pantalla se emite luz.

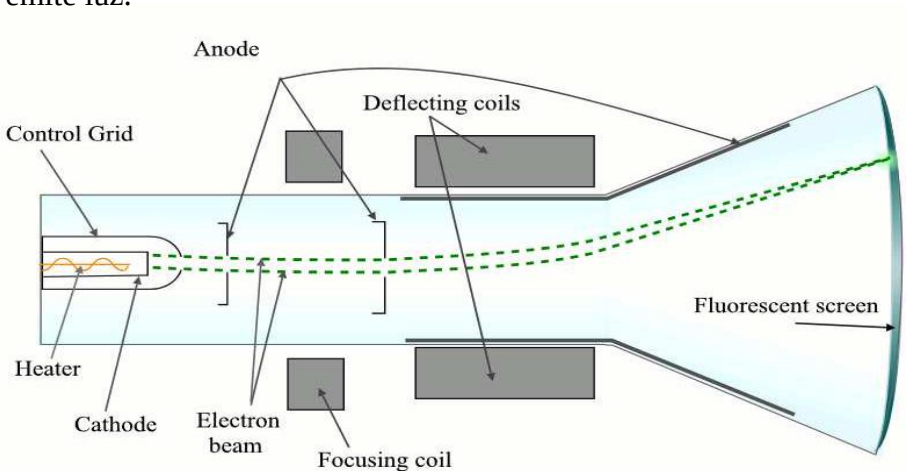


Figura 2.6 Componentes de un CRT.

El interior de la pantalla está recubierto con un tipo especial de fósforo, así el color de la luz emitido por la pantalla está determinado por el fósforo particular usado. La figura 2.6 ilustra algunos componentes de un CRT.

La visualización mediante barrido

En el caso de los televisores y de los monitores de ordenador modernos, todo el frontal del tubo se obtiene por escáner según un recorrido definido, y se crea la imagen haciendo variar la intensidad del flujo de electrones (el haz) a lo largo del recorrido. El flujo en todas las TV modernas es desviado por un campo magnético aplicado sobre el cuello del tubo por un yugo magnético (magnetic yoke en ingles) figura 2.7, que está formado por bobinas (a menudo dos) envueltas sobre ferrita y controladas por un circuito electrónico. Este sería un barrido por desviación magnética.

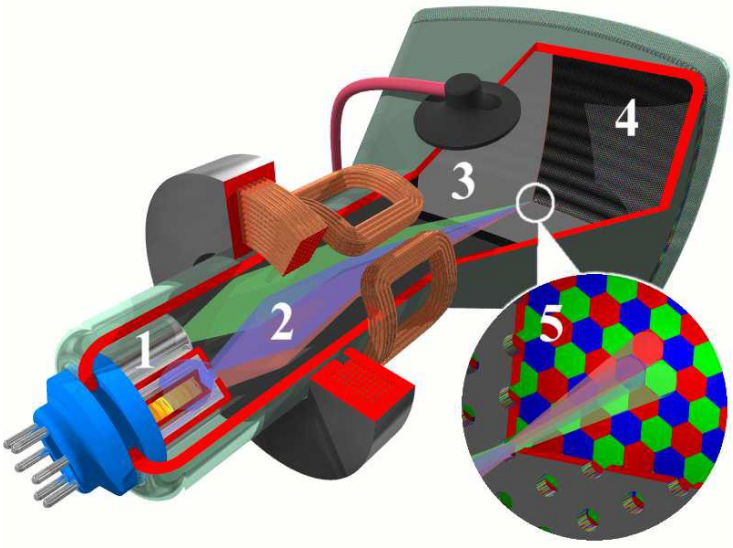


Figura 2.7 se puede apreciar un tubo de barrido en color.

1. Cañones de electrones.
2. Haces de electrones.
3. Mascara para separar los rayos rojos, azules y verdes de la imagen visualizada.
4. Capa fosforescente con zonas receptivas para cada color.
5. Gran superficie plana sobre la cara interior de la pantalla cubierta de fosforo.

Señales para producir una imagen

Una imagen en la pantalla del monitor no permanece fija, es decir, haciendo una analogía, no es como una hoja impresa sino que es una imagen "parpadeando" a tal velocidad que es imperceptible al ojo humano. En tales "parpadeos" la imagen que se muestra en el monitor es nula, por lo que la imagen que percibimos se está refrescando continuamente.

La imagen no es producto de una sola señal, de hecho es el resultado de la combinación de 5 señales independientes, que forman la llamada Señal de Video Compuesta. Las señales que participan en esta señal combinada son:

1.-Señal de Luminancia (señales RGB).

2.- señales de sincronía vertical

- Pulsos de Sincronización vertical
- Pulsos de Blanqueo vertical.

3.- señales de sincronía horizontal

- Pulsos de Sincronización vertical
- Pulsos de Blanqueo vertical

Se puede considerar la pantalla como una malla de posiciones, por lo que una imagen está compuesta por los elementos de la malla. Cada elemento de la imagen representa un punto o píxel (también llamado pel). Para mostrar un elemento de la imagen en la pantalla, el haz de electrones excita al elemento generándose una señal que es proporcional a la intensidad de la luz que incide sobre él, a esta señal se le llama señal de luminancia.

La señal de luminancia es la señal de video, pero sólo representa un punto de la imagen. Para desplegar toda la imagen en la pantalla se debe mostrar cada elemento de la imagen, uno a la vez. Para llevar a cabo esto, se debe realizar la exploración. La exploración es un método para desplegar imágenes en una pantalla de CRT.

Su nombre se debe a que el haz de electrones explora la superficie de la pantalla, en otras palabras, realiza un barrido de la superficie de la pantalla comenzando de la parte izquierda a la derecha y desde arriba hacia abajo, de esta manera se muestran todos los puntos de la malla. La exploración se realiza de la misma manera en que se lee un libro, es por eso que se le llama exploración horizontal secuencial figura 2.8.

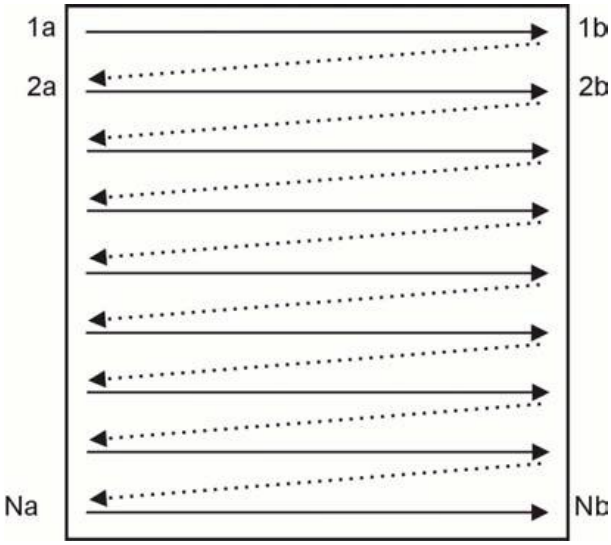


Figura 2.8 Exploración horizontal.

El origen de la exploración es en la parte superior izquierda de la pantalla (punto 1a) figura 2.8. Para desplegar la primera línea de la pantalla, el haz de electrones se mueve en forma horizontal hasta llegar a la parte superior derecha (punto 1b). En ese momento se ha realizado el despliegue de la primera línea de la pantalla. Para desplegar toda la imagen se deben desplegar todas las líneas que conforman la imagen, por ello, el haz de electrones debe continuar con la siguiente línea en la pantalla, por lo que regresa a la parte izquierda pero ahora se posiciona en la segunda línea (punto 2a). El haz de electrones continua realizando el barrido y cuando llega a la parte derecha de la última línea de la pantalla (punto Nb), el haz regresa al origen de la pantalla (punto 1a). De esta manera la pantalla parecerá desplegar toda la imagen al mismo tiempo. A la imagen completa, compuesta por todas las líneas de exploración, se le llama trama (frame).

Para controlar la trayectoria del haz de electrones, los deflectores magnéticos utilizan los pulsos de sincronización y, como es de esperarse, hay pulsos de sincronización horizontal y vertical. Normalmente estas señales se encuentran en un nivel lógico alto, cuando se aplica el pulso de sincronización pasan a un nivel lógico bajo.

El pulso de sincronización horizontal marca el inicio y final de una línea de exploración, con esto garantiza el despliegue de píxeles entre los límites de izquierda a derecha del área visible de la pantalla. De igual forma, el pulso de sincronización vertical marca el inicio y fin de la imagen pero de manera vertical, es decir, garantiza que el monitor despliegue las líneas de exploración entre la parte superior e inferior de la pantalla.

Por otra parte, cuando el haz de electrones regresa de la parte derecha de la imagen a la parte izquierda de la siguiente línea, ocurre un lapso de tiempo llamado retraso horizontal.

De la misma manera, al tiempo en que el haz regresa del extremo derecho inferior de la pantalla a la parte superior izquierda se llama retraso vertical. Durante los tiempos de retraso horizontal y vertical no se despliega la imagen en la pantalla. Es en ese momento en el que se aplican los pulsos de blanqueo (llamados blankings en inglés).

El objetivo de los pulsos de blanqueo es poner la pantalla en negro cuando se presenta algún tiempo de retraso. Esto debido a que en ese lapso el haz de electrones no realiza ninguna exploración, es decir, no despliega imágenes en la pantalla por lo que no es necesario mostrar algo en ella.

Así, un monitor debe contar con 3 señales de entrada para desplegar una imagen: pulso de sincronización horizontal, pulso de sincronización vertical y señal RGB.

Temporización de las señales

En la exploración horizontal el tiempo máximo en que se despliegan los píxeles es de 25.17 µs (dentro de ese tiempo el haz de electrones se desplaza horizontalmente). Después de desplegar el último píxel de la línea horizontal, se debe aplicar el blanqueo (llamado blanqueo horizontal), es decir, poner la pantalla en negro por 6.6 µs como mínimo.

Igualmente, al terminar los 25.17 µs se debe esperar un mínimo de 0.94 µs para aplicar el pulso de sincronización horizontal en un nivel lógico bajo y mantenerlo por 3.77 µs.

Después de aplicar el pulso de sincronización horizontal, se debe esperar por lo menos 1.89 µs antes de comenzar una nueva línea, es decir, antes de enviar los píxeles de la siguiente línea. Así, el total de tiempo en que se explora la línea horizontal y antes de desplegar la siguiente es de 31.77 µs.

Para el pulso de sincronización vertical el tiempo máximo en que el monitor debe desplegar líneas es de 15.25 ms. Después de ese tiempo, se aplica el blanqueo (blanqueo vertical) por 1.534 ms para poner la pantalla oscura. Antes de aplicar el pulso de sincronización vertical, se debe esperar 0.45 ms después de la última línea. El pulso de sincronización vertical permanece en un nivel lógico bajo por 64 µs, después de ese tiempo, se debe esperar por lo menos 1.02 ms antes de comenzar de nuevo con la primera línea. Así, el tiempo total en que se despliegan todas las líneas y antes de desplegar la primera línea nuevamente es de 16.784 ms.

La figura 2.9 ilustra los tiempos de las señales que participan en el despliegue de la imagen. El blanqueo vertical se efectúa después de un lapso de varias líneas de video (líneas de exploración horizontal). Normalmente para un monitor VGA el número de píxeles desplegados horizontalmente es de 640 y el número de líneas visibles es de 480, por lo que su resolución es de 640x480.

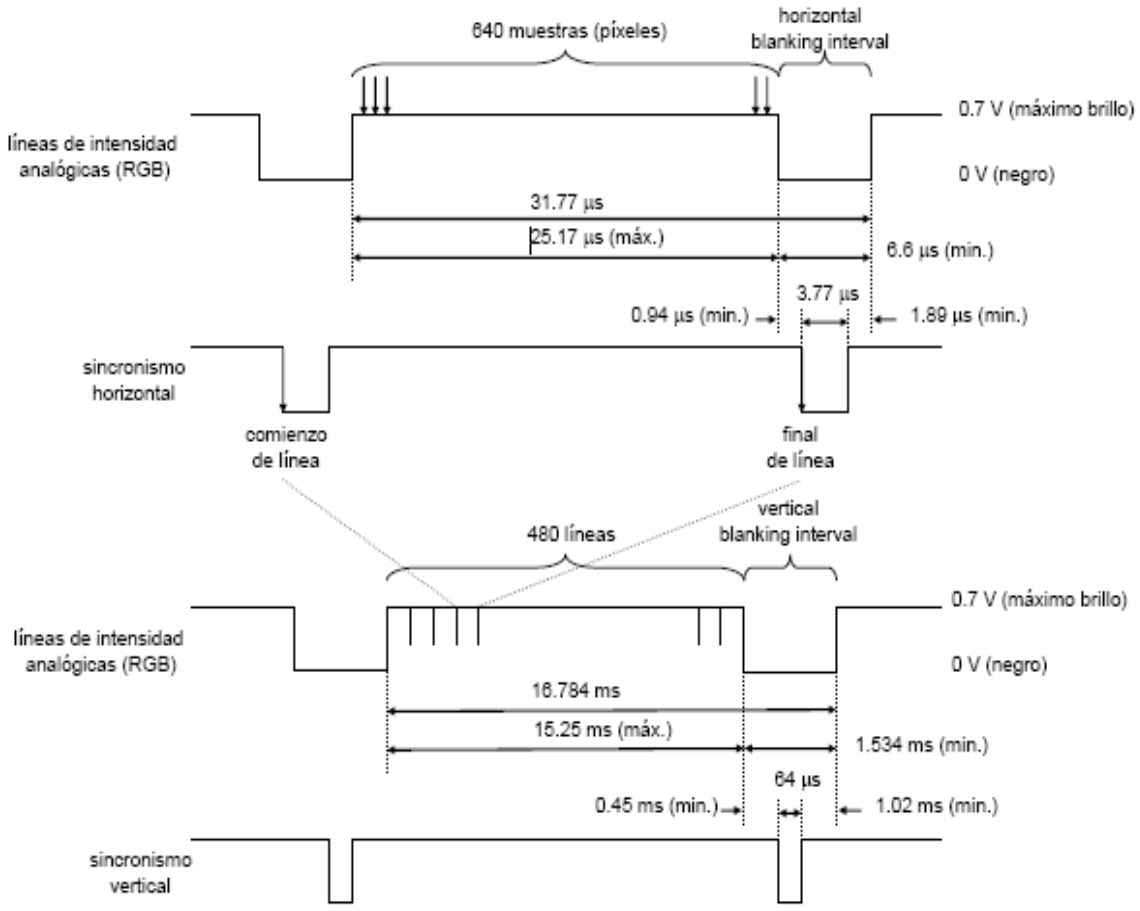


Figura 2.9 Temporización de las señales de video

Frecuencia de reloj y resolución

La resolución que puede ofrecer el controlador depende de la resolución del monitor (por ejemplo 640x480) y de la frecuencia de reloj a la que trabaja. Los tiempos que se emplean para aplicar las señales en una exploración horizontal (tiempo de blanqueo horizontal, tiempo de sincronización horizontal, tiempo de despliegue de píxeles, etc.) pueden expresarse en términos de cantidad de ciclos de reloj. Para esto, se divide cada uno de los tiempos que participan en la exploración horizontal (THor) entre el periodo de la señal de reloj a la que funciona el controlador (Ec.2.1). Determinar la cantidad de ciclos es importante para aplicar el pulso de sincronización horizontal, enviar la cantidad correcta de píxeles al monitor y realizar el blanqueo horizontal.

$$cantidad\ de\ ciclos = \frac{THor}{periodo\ de\ frecuencia\ principal}$$ Ec. 2.1

Asimismo, para obtener la cantidad de líneas correspondientes a los tiempos de las señales que participan en la exploración vertical (tiempo de despliegue de líneas, tiempo de blanqueo vertical, tiempo de sincronización vertical, etc.), se divide cada uno de los tiempos que participan en la exploración vertical (TVert) entre el tiempo total invertido en una exploración horizontal (Ec.2.2). Al determinar el número de líneas se conoce el número de líneas de exploración se pueden desplegar en la pantalla, además el número de líneas para aplicar la señal de sincronismo y de blanqueo vertical.

$$cantidad\ de\ ciclos = \frac{TVert}{tiempo\ en\ que\ se\ realiza\ la\ exploracion\ horizontal}$$ Ec. 2.2

VGA INDUSTRY STANDARD 640x480 PIXELES.

- *FRECUENCIAS:*
  - *Reloj de muestreo del monitor = 25.175 MHz*
  - *De sincronización horizontal (line. freq) = 31.469 KHz*
  - *De sincronización vertical = 59.94 HZ*
- *800 pixeles por línea (640 visible) = 25.174 MHz/31.469KHz*
  - *8 pixeles de porche delantero.*
  - *96 pixeles de sincronización horizontal.*
  - *40 pixeles de porche trasero.*
  - *8 pixeles de borde izquierdo.*
  - *640 pixeles de video.*
  - *8 pixeles de borde derecho.*

- *525 líneas por pantalla (480 visibles) = 31.466 KHz/59.94Hz.*
  - *2 líneas de porche delantero.*
  - *2 líneas de sincronización vertical.*
  - *25 líneas de porche trasero.*
  - *8 líneas de borde superior.*
  - *480 líneas de video.*
  - *8 líneas de borde inferior.*

La figura 2.10 muestra de manera grafica cada uno de los elementos por línea horizontal descritos con anterioridad.

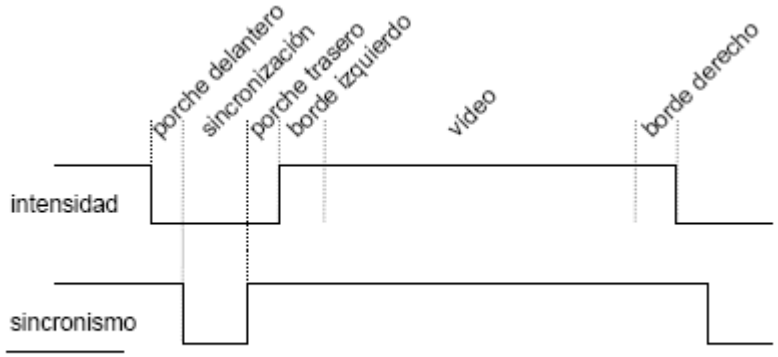


Figura 2.10 Numero de pixeles por línea horizontal.

2.3 PANTALLA LCD-TFT

Características de los monitores LCD-TFT.

LCD son las siglas de "Liquid Cristal Display", pantalla de cristal líquido. Fue descubierto por Fredeich Rheinizer en 1888. A mediados de los años 60, con la técnica más avanzada, los científicos llegaron a dominar esta tecnología al comprobar que, gracias a un estímulo externo, cambiaba las propiedades de la luz al atravesarlo. Estos cristales comenzaron a utilizarse en las calculadoras, relojes digitales y posteriormente en ordenadores portátiles. El funcionamiento es relativamente sencillo: en la pantalla LCD hay dos filtros que polarizan, con filas de cristales líquidos que forman 90º entre ellas. Según apliquemos o no una corriente eléctrica, la luz pasará o no a través de ellos (siendo el segundo filtro el que permitirá el paso de la luz que haya filtrado previamente el primero de ellos: dos filtros de este tipo colocados perpendicularmente no permiten el paso de la luz, así que el segundo debe girar para permitir el paso de la luz en las zonas necesarias). Para conseguir color, además se emplean tres filtros adicionales (rojo, verde y azul) y las diferentes variedades de los colores se obtienen aplicando diferentes voltajes a los filtros. La técnica más utilizada en los monitores es TFT "Thin Film Transistor". Se basa en transistores individuales (conmutadores) que gobiernan cada píxel del display. La luz es emitida por los transistores y, mediante el bloqueo de unos filtros RGB activos colocados de forma lineal sobre las superficies mencionadas, se consigue el tono del color del punto.

CARACTERÍSTICAS A TENER EN CUENTA

Tamaño de pantalla

Para las aplicaciones más usuales se suelen utilizar monitores de 15" ó 17" pudiendo llegar hasta las 22". Recordemos que dado que se utiliza el área total de la pantalla estas dimensiones son completamente útiles.

Resolución

Es el número de puntos que el monitor puede representar en pantalla. La forma de nombrarlo es mediante dos números separados por el signo de multiplicar, refiriéndose el primero a los puntos en horizontal y el segundo a los puntos en vertical. Cuanto mayor es la resolución, mayor es la calidad de la imagen. En cualquier caso la resolución del monitor ha de ser consecuente con el tamaño de éste. Para 15" la resolución más indicada es de 1024x768 y para el monitor de 17" de 1280x1024 puntos.

Tiempo de respuesta

Podría entenderse como el equivalente al refresco de la pantalla en un monitor CRT. Se refiere al tiempo que tarda cada celda en responder a los cambios del campo eléctrico

aplicado, renovando de este modo la imagen en pantalla. Un tiempo de respuesta no debería superar en ningún caso los 70ms.

Angulo de Visión

En los principios de esta tecnología este era el principal problema. Resultaba casi imposible ver la imagen de la pantalla si no se miraba de frente a ella. Los valores mínimos admisibles hoy en día serían de 45º hacia arriba y hacia abajo y de 60º a la derecha y a la izquierda.

Brillo y Contraste

El brillo hace referencia a la intensidad luminosa de una fuente de luz en un área concreta. Se mide en candelas por metro cuadrado. Como mínimo debe ser de 150 cd/m2. El contraste es la relación existente entre la intensidad del punto más claro y el más oscuro. Cuanto mayor sea este valor más nítido será la imagen en el monitor TFT. Como mínimo debe exigirse un valor de 100:1.

Medidas

Las medidas del TFT suelen darse en diagonal y en pulgadas. De este modo:

Tabal 2.2 medidas de TFT LCD

Pulgadas	Diagonal
15	38.1 cm
17	43.18 cm
18	45.72 cm
19	48.26 cm
21	53.34 cm

2.3.1 PANTALLA TRDB-LCM

Características de la pantalla

La característica establecida del TRDB se encuentra enumerada debajo:

1. Acondicionado con módulo compacto Toppoly TD036THEA1 de la pantalla de cristal líquido TFT.

2. Maneja señales digitales en un bus de 8 bits (RGB o YUV).

3. Soporta formatos NTSC y PAL

4. Contiene tres líneas de registro para el manejo del display y uno para selección.

5. Contiene un contraste incorporado, la claridad y la modulación de la gama de colores.

6. El filtro de color de la tira soporta 960x240 (a través del modo, RGB sustituya por una variable ficticia, aporte YUV).

7. Las especificaciones generales de la pantalla se muestran en la tabla 2.3

Tabla 2.3 especificaciones generales del panel

Item	Description	Unit
Display Size (Diagonal)	3.6	Inch
Display Type	Transmissive	-
Active Area (HxV)	72.96 x 54.72	mm
Number of Dots (HxV)	320 x RGB x 240	dot
Dot Pitch (HxV)	0.076 x 0.228	mm
Color Arrangement	RGB Stripe	-
Color Numbers	8 bit RGB (16M color)	-

Los fabricantes de pantallas LCD desarrollan sus equipos independientes de las otras compañías, es por eso que para esta pantalla se presenta la siguiente tabla 2.4 donde se puede observar sus características

Tabla 2.4 datos requeridos por la pantalla para el despliegue de imágenes.

Pin Numbers	Name	Direction	Description
1~10	NC	N/A	Not connect
11	VCC5	N/A	Power 5V
12	GND	N/A	Ground
13~20	NC	N/A	Not connect
21	DIN6	Input	LCD data bus bit 7
22	DIN7	Input	LCD data bus bit 6
23	DIN4	Input	LCD data bus bit 4
24	DIN5	Input	LCD data bus bit 5
25	DIN2	Input	LCD data bus bit 2
26	DIN3	Input	LCD data bus bit 3
27	DIN0	Input	LCD data bus bit 0
28	DIN1	Input	LCD data bus bit 1
29	VCC33	N/A	Power 3.3V
30	NC	N/A	Not connect
31	VSYNC	Input	Vertical sync input
32	NC	N/A	Not connect
33	SCL	Input	3-wire serial interface clock
34	DCLK	Input	LCD data clock
35	GRESTB	Input	Global reset, low active
36	SHDB	Input	Shutdown control, low active
37	CPW	N/A	Reserved
38	SCEN	Input	3-wire serial interface enable
39	SDA	Input/Output	3-wire serial interface data
40	HSYNC	Input	Horizontal sync input

2.4 LOS FPGA

Una FPGA (del inglés Field Programmable Gate Array) es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad se puede programar. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional, o implementar hasta complejos sistemas en un chip.

Las FPGAs se utilizan en aplicaciones similares a los ASICs sin embargo son más lentas, tienen un mayor consumo de potencia y no pueden abarcar sistemas tan complejos como ellos. A pesar de esto, las FPGAs tienen las ventajas de ser reprogramables (lo que añade una enorme flexibilidad al flujo de diseño), sus costes de desarrollo y adquisición son mucho menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor.

Ciertos fabricantes cuentan con FPGAs que sólo se pueden programar una vez, por lo que sus ventajas e inconvenientes se encuentran a medio camino entre los ASICs y las FPGAs reprogramables.

Históricamente las FPGAs surgen como una evolución de los conceptos desarrollados en las PLAs y los CPLDs.

Las FPGAs fueron inventadas en el año 1984 por Ross Freeman y Bernard Vonderschmitt, co-fundadores de Xilinx, y surgen como una evolución de los CPLDs.

Tanto los CPLDs como las FPGAs contienen un gran número de elementos lógicos programables. Si medimos la densidad de los elementos lógicos programables en puertas lógicas equivalentes (número de puertas NAND equivalentes que podríamos programar en un dispositivo) podríamos decir que en un CPLD hallaríamos del orden de decenas de miles de puertas lógicas equivalentes y en una FPGA del orden de cientos de miles hasta millones de ellas.

Aparte de las diferencias en densidad entre ambos tipos de dispositivos, la diferencia fundamental entre las FPGAs y los CPLDs es su arquitectura. La arquitectura de los CPLDs es más rígida y consiste en una o más sumas de productos programables cuyos resultados van a parar a un número reducido de biestables síncronos (también denominados flip-flops). La arquitectura de las FPGAs, por otro lado, se basa en un gran número de pequeños bloques utilizados para reproducir sencillas operaciones lógicas, que cuentan a su vez con biestables síncronos. La enorme libertad disponible en la interconexión de dichos bloques confiere a las FPGAs una gran flexibilidad.

Otra diferencia importante entre FPGAs y CPLDs es que en la mayoría de las FPGAs se pueden encontrar funciones de altos niveles (como sumadores y multiplicadores) embebidas en la propia matriz de interconexiones, así como bloques de memoria.

2.4.1 HISTORIA DE LOS FPGA

Las FPGA son el resultado de la convergencia de dos tecnologías diferentes, los dispositivos lógicos programables (PLDs [Programmable Logic Devices]) y los circuitos integrados de aplicación específica (ASIC [application-specific integrated circuit]). La historia de los PLDs comenzó con los primeros dispositivos PROM (Programmable Read-Only Memory) y se les añadió versatilidad con los PAL (Programmable Array Logic) que permitieron un mayor número de entradas y la inclusión de registros. Esos dispositivos han continuado creciendo en tamaño y potencia. Mientras, los ASIC siempre han sido potentes dispositivos, pero su uso ha requerido tradicionalmente una considerable inversión tanto de tiempo como de dinero. Intentos de reducir esta carga han provenido de la modulizacion de los elementos de los circuitos, como los ASIC basados en celdas, y de la estandarización de las máscaras, tal como Ferranti fue pionero con la ULA (Uncommitted Logic Array). El paso final era combinar las dos estrategias con un mecanismo de interconexión que pudiese programarse utilizando fusibles, antifusibles o celdas RAM, como los innovadores dispositivos Xilinx de mediados de los 80. Los circuitos resultantes son similares en capacidad y aplicaciones a los PLDs más grandes, aunque hay diferencias puntuales que delatan antepasados diferentes. Además que en computación reconfigurable, las FPGAs se utilizan en controladores, codificadores/decodificadores y en el prototipado de circuitos VLSI y microprocesadores a la medida.

El primer fabricante de estos dispositivos fue Xilinx y los dispositivos de Xilinx se mantienen como uno de los más populares en compañías y grupos de investigación. Otros vendedores en este mercado son Atmel, Altera, AMD y Motorola.

2.4.2 CARACTERISTICAS DE LOS FPGAs

Características

Una jerarquía de interconexiones programables permite a los bloques lógicos de un FPGA ser interconectados según la necesidad del diseñador del sistema, algo parecido a un breadboard programable. Estos bloques lógicos e interconexiones pueden ser programados después del proceso de manufactura por el usuario/diseñador, así que el FPGA puede desempeñar cualquier función lógica necesaria.

Una tendencia reciente ha sido combinar los bloques lógicos e interconexiones de los FPGA con microprocesadores y periféricos relacionados para formar un «Sistema programable en un chip». Ejemplo de tales tecnologías híbridas pueden ser encontradas en los dispositivos Virtex-II PRO y Virtex-4 de Xilinx, los cuales incluyen uno o más

procesadores PowerPC embebidos junto con la lógica del FPGA. El FPSLIC de Atmel es otro dispositivo similar, el cual usa un procesador AVR en combinación con la arquitectura lógica programable de Atmel. Otra alternativa es hacer uso de núcleos de procesadores implementados haciendo uso de la lógica del FPGA. Esos núcleos incluyen los procesadores MicroBlaze y PicoBlaze de Xlinx, Nios y Nios II de Altera, y los procesadores de código abierto LatticeMicro32 y LatticeMicro8.

Muchos FPGA modernos soportan la reconfiguración parcial del sistema, permitiendo que una parte del diseño sea reprogramada, mientras las demás partes siguen funcionando. Este es el principio de la idea de la «computación reconfigurable», o los «sistemas reconfigurables».

Programación

La tarea del programador es definir la función lógica que realizará cada uno de los CLB, seleccionar el modo de trabajo de cada IOB e interconectarlos.

El diseñador cuenta con la ayuda de entornos de desarrollo especializados en el diseño de sistemas a implementarse en un FPGA. Un diseño puede ser capturado ya sea como esquemático, o haciendo uso de un lenguaje de programación especial. Estos lenguajes de programación especiales son conocidos como HDL o Hardware Description Language (lenguajes de descripción de hardware). Los HDLs más utilizados son:

- VHDL
- Verilog
- ABEL

En un intento de reducir la complejidad y el tiempo de desarrollo en fases de prototipaje rápido, y para validar un diseño en HDL, existen varias propuestas y niveles de abstracción del diseño. Entre otras, National Instruments LabVIEW FPGA propone un acercamiento de programación gráfica de alto nivel.

Aplicaciones

Cualquier circuito de aplicación específica puede ser implementado en un FPGA, siempre y cuando esta disponga de los recursos necesarios. Las aplicaciones donde más comúnmente se utilizan los FPGA incluyen a los DSP (procesamiento digital de señales), radio definido por software, sistemas aeroespaciales y de defensa, prototipos de ASICs, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora, entre otras. Cabe notar que su uso en otras áreas es cada vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo.

Existe código fuente disponible (bajo licencia GNU GPL)1 de sistemas como microprocesadores, microcontroladores, filtros, módulos de comunicaciones y memorias, entre otros. Estos códigos se llaman cores.

Tecnología de la memoria de programación

Las FPGAs también se pueden diferenciar por utilizar diferentes tecnologías de memoria: Volátiles: Basadas en RAM. Su programación se pierde al quitar la alimentación. Requieren una memoria externa no volátil para configurarlas al arrancar (antes o durante el reset).

- No Volátiles: Basadas en ROM. Hay de dos tipos, las reprogramables y las no reprogramables.
- Reprogramables: Basadas en EPROM o flash. Éstas se pueden borrar y volver a reprogramar aunque con un límite de unos 10.000 ciclos.
- No Reprogramables: Basadas en fusibles. Solo se pueden programar una vez, lo que las hace poco recomendables para trabajos en laboratorios.

Fabricantes

A principios de 2007, el mercado de los FPGA se ha colocado en un estado donde hay dos productores de FPGA de propósito general que están a la cabeza del mismo, y un conjunto de otros competidores quienes se diferencian por ofrecer dispositivos de capacidades únicas.

- Xilinx es uno de los dos grandes líderes en la fabricación de FPGA.
- Altera es el otro gran líder.
- Lattice Semiconductor lanzó al mercado dispositivos FPGA con tecnología de 90nm. En adición, Lattice es un proveedor líder en tecnología no volátil, FPGA basadas en tecnología Flash, con productos de 90nm y 130nm.
- Actel tiene FPGAs basados en tecnología Flash reprogramable. También ofrece FPGAs que incluyen mezcladores de señales basados en Flash.
- QuickLogic tiene productos basados en antifusibles (programables una sola vez).
- Atmel es uno de los fabricantes cuyos productos son reconfigurables (el Xilinx XC62xx fue uno de estos, pero no están siendo fabricados actualmente). Ellos se enfocaron en proveer microcontroladores AVR con FPGAs, todo en el mismo encapsulado.

- Achronix Semiconductor tienen en desarrollo FPGAs muy veloces. Planean sacar al mercado a comienzos de 2007 FPGAs con velocidades cercanas a los 2GHz.
- MathStar, Inc. ofrecen FPGA que ellos llaman FPOA (Arreglo de objetos de matriz programable).

2.4.3 CYCLONE II

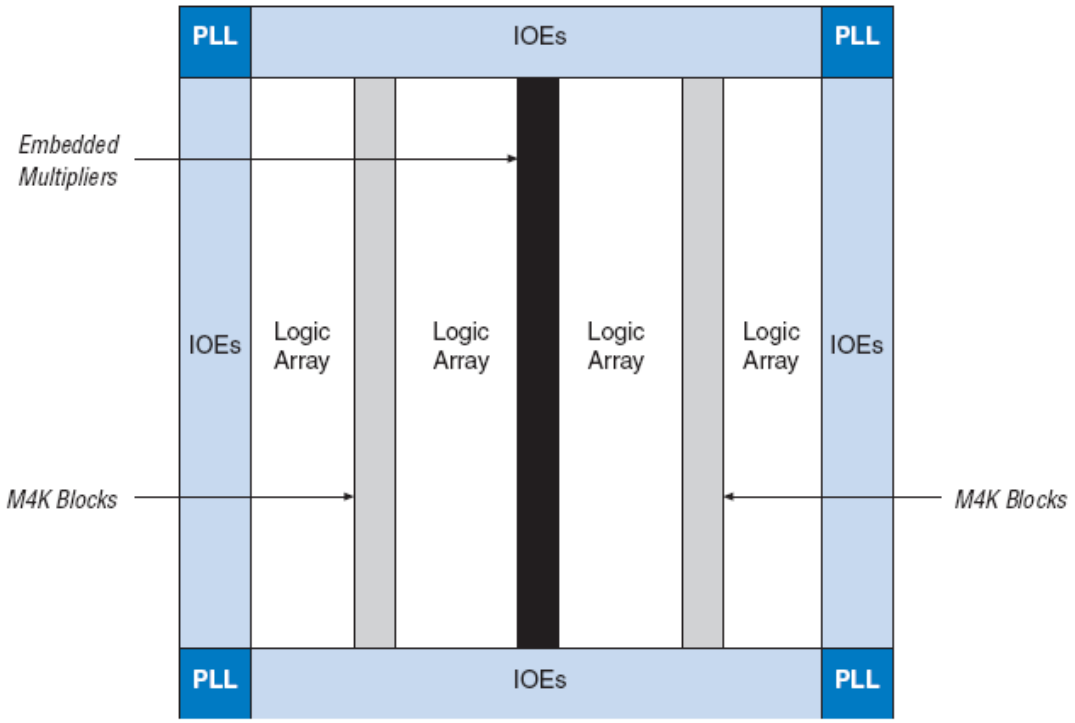


Figura 2.11 A rquitectura del FPGA Cyclone II de Altera

Descripción funcional:

Este circuito muestra a detalle la estructura interna de un FPGA, de cada uno de los componentes principales que conforman este dispositivo figura 2.11.

Los dispositivos Cyclone ® II contienen una fila de dos dimensiones y arquitectura basada en columnas para la implementación de la lógica basica. La columna y fila se interconecta a los relojes (PLLs) que proveen señales a los bloques lógicos (LABs), bloques de memoria y multiplicadores incrustados.

El arreglo lógico consiste en LABs, con 16 elementos lógicos (LEs) en cada LAB. Un LE es una pequeña unidad lógica con implementación de funciones lógicas eficientes. Los

bloque de arreglo lógicos (LABs) son agrupadas en filas y columnas a través del dispositivo. El dispositivo Cyclone II tiene un rango en densidad de 4,608 a 68416 LEs.

El dispositivo Cyclone II proporciona una red global de reloj y hasta cuatro bucles de enganche de fase (*Phase Locked-Loop*: PLL). La red global de reloj se compone de 16 líneas que controla totalmente el dispositivo, puede proporcionar relojes para todo el recurso necesario dentro del dispositivo, como entradas/salidas (IOEs), LEs, multiplexores y memorias de bloques embebidas, etc. Las líneas de reloj global también pueden ser utilizadas para otras señales de fan-out altas. Cyclone II PLLs provee de manera general, sincronización con la síntesis de reloj y corrimiento de fase, así como también, salidas externas de alta velocidad con soporte diferencial de E/S.

Los bloques de memoria MK4 son bloques de memorias de doble puerto real, con 4K bits de memoria, mas la paridad (4,608 bits). Estos bloques dedicados proveen doble puerto real, doble puerto simple, o puerto singular de memoria hasta 36-bits de ancho y hasta 260Mhz. Estos bloques son organizados en columna a través del dispositivo y están ubicados entre cierto LABs. El dispositivo Cyclone II ofrece entre 119 a 1,152 Kbits de memoria embebida.

Cada bloque de multiplicador embebido puede implementar ya sea dos multiplicador 9x9-bits, y uno multiplicador 18x18-bits con hasta 250Mhz de función. Los multiplicadores embebidos son ordenados en columnas a través del dispositivo.

El numero de bloques de memoria MK4, bloques de multiplicador embebido, PLLs, columnas y fila varia por dispositivo.

Elemento lógico (LE)

Es la unidad más pequeña de la arquitectura Cyclone II, el LE, es compacta y ofrece funciones avanzadas con una utilización eficiente de la lógica. Cada LE cuenta con:

- Una tabla de búsqueda (look-up table: LUT) de cuatro entradas, son las encargadas de generar las funciones de cuatro variables.
- Un registro programable.
- Una conexión de cadena de acarreo.
- Una conexión de cadena de registro.
- La capacidad para controlar cualquier tipo de conexión: local, fila, columna, cadena de registro y enlace directo interconexiones.
- Soporte para empaquetado y retroalimentación de registro.

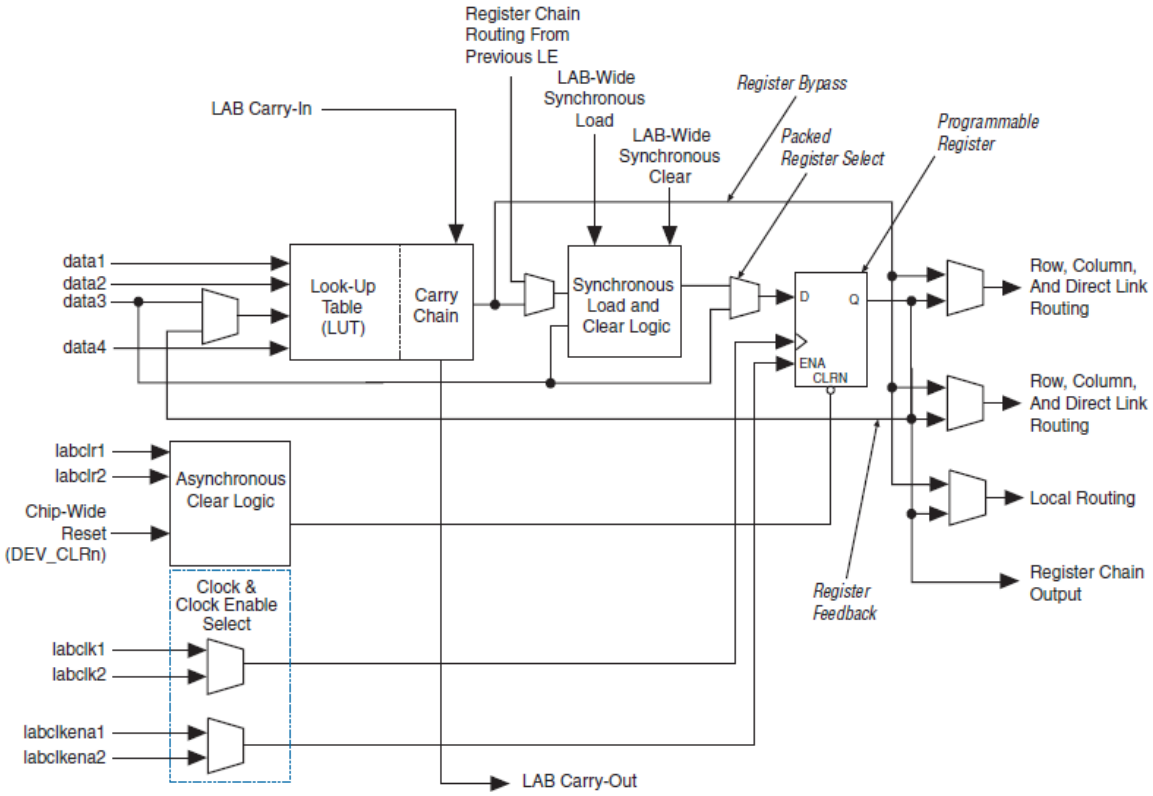


Figura 2.12 look-up table: LUT

El diseño lógico se implementa mediante bloques conocidos como generadores de funciones o LUT (Look Up Table: tabla de búsqueda) figura 2.12, los cuales permiten almacenar la lógica requerida, ya que cuentan con una pequeña memoria interna por lo general de 16 bits. Cuando se aplica alguna combinación en las entradas de la LUT, el circuito la traduce en una dirección de memoria y envía fuera del bloque el dato almacenado en esa dirección.

Cada registro programable LE´s puede ser configurado para D, T, JK, u operación SR. Cada registro cuenta con datos, reloj, habilitación de reloj, y entradas.

Cada **LE** tiene tres salidas que controlan las interconexiones local, columna, fila y columna de recurso de enrutamiento. La LUT o registro de salida puede controlar estas tres salidas de forma independiente. Dos salidas LE controlan filas o columnas y enlaces directo de conexión de enrutamiento y la otra controla las interconexiones de recursos, permitiendo que la LUT controle una salida mientras el registro controle otra salida. Esta característica mejora la utilización de los dispositivos porque el dispositivo puede utilizar el registro y la LUT para funciones no relacionadas.

LE Modos de funcionamiento

La LE Cyclone II opera en uno de los siguientes modos:

1. Modo Normal
2. Modo Aritmético

Cada modo utiliza los recursos LE diferente. En cada modo, seis entradas a disposición de la LE—Las cuatro entradas de datos desde el LAB local de interconexión, el LAB de entrada-acarreo de la anterior cadena-acarreo LAB, y la conexión de la cadena de registro—son dirigidas a diferentes destinos para implementar la función de la lógica deseada. Las señales LAB-Wide proporcionan reloj, clear asíncrono, clear síncrono, carga síncrona, y el reloj de habilitación para control de el registro. Estas señales LAB-Wide están disponible en todos los modos LE.

El software Quartus ® II, en relación con las funciones de parámetros tales como la biblioteca de módulos con parámetros de funciones (LPM), elige automáticamente el modo apropiado para las funciones comunes, tales como contadores, sumadores, restadores, y funciones aritméticas. Si es necesario, también puede crear funciones especiales que especifican cual modo de operación LE utilizar para un rendimiento óptimo.

Modo Normal:

El modo normal es adecuado para aplicaciones lógicas generales y funciones combinacionales. En modo normal, cuatro datos de entrada de LAB local de interconexión son necesarios para un periodo de cuatro LUT de entrada Figura 2-13. El compilador Quartus II selecciona automáticamente el retardo o la señal de Data3 como una de las entradas a la LUT. Él LE en modo normal soporta registros de retroalimentación y empaquetados.

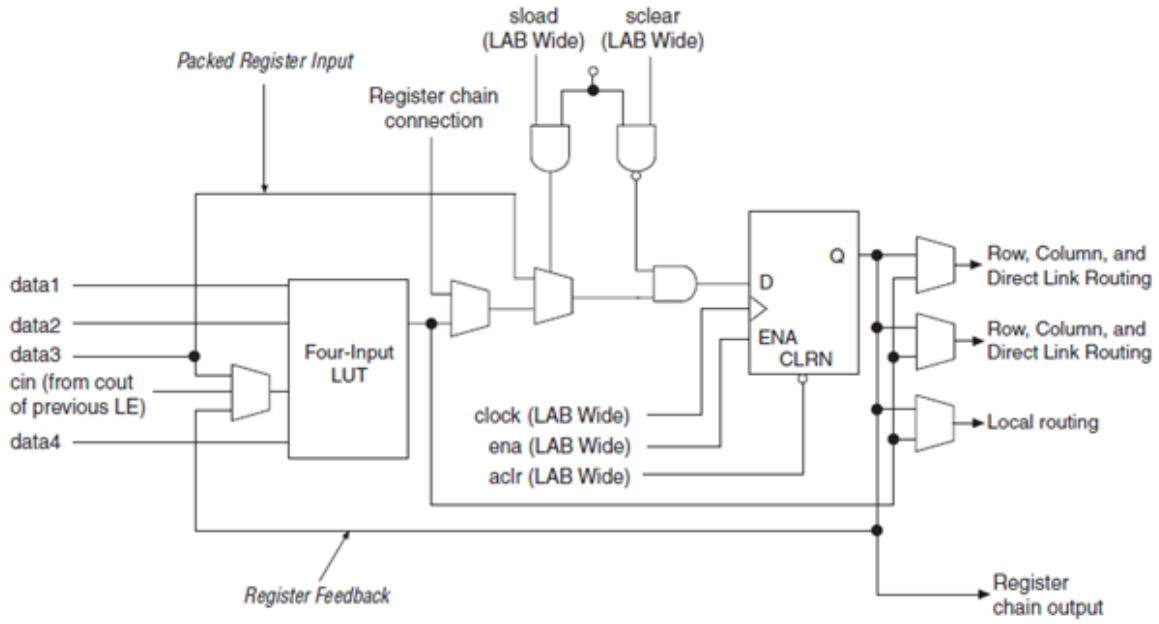


Figura 2.13 LE en modo normal

Modo Aritmético

El modo aritmético es ideal para la implementación de contadores, sumadores, acumuladores, y comparadores. Un LE en modo aritmético implementa un sumador total de 2 bits y una cadena básica de acarreo Figura 2.14. LEs en modo aritmético puede controlar salida de versiones registrada y no registrada de la LUT de salida. La retroalimentación y paquete de registro son soportados cuando la LEs se utiliza en modo aritmético.

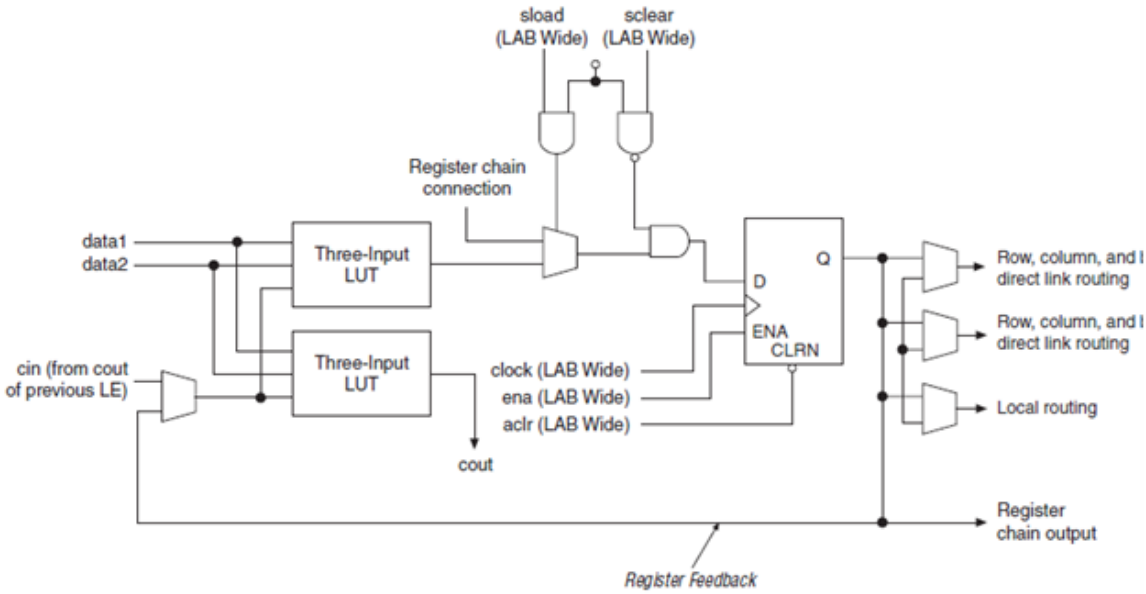


Figura 2.14 LE (elemento logico) en modo aritmético

El compilador Quartus II crea automáticamente cadenas de acarreos lógicos durante el proceso de diseño, o se puede crear de forma manual durante el diseño de entrada. Funciones parametrizable tales como funciones LPM automáticas, aprovechando cadenas de acarreo para las funciones apropiadas.

También crea cadenas de acarreos más de 16 LEs para vincular automáticamente los LAB en una misma columna. Para una mejor adaptación, una larga cadena de acarreos corre verticalmente, que permite rápidas conexiones horizontales mente de los bloques de memoria M4K o multiplicadores integrados a través de enlace directo de interconexiones. Por ejemplo, si un diseño tiene una larga cadena de acarreos en una columna LAB junto a una columna de bloques de memoria M4K, cualquier salida LE puede alimentar a un bloque de memoria M4K contiguo a través de enlace directo de interconexiones. Mientras si la cadena de acarreos corre horizontalmente, cualquier LAB a lado de la columna de bloques de memoria M4K usaría otra fila o columna de interconexiones para controlar un bloque de memoria M4K.

Arreglos de bloques lógicos (LABs)

Los Arreglos de bloques lógicos (LABs) (llamados también celdas generadoras de funciones) están configurados para procesar cualquier aplicación lógica. Estos bloques tienen la característica de ser funcionalmente completos; es decir, permiten la implementación de cualquier función booleana representada en la forma de suma de productos.

Cada LABs consiste en los siguientes:

- 16 LEs
- Control de señal LAB
- Acarreo de cadena LE
- Cadena de Registro
- Interconexión local

La interconexión local de las señales de transferencia entre las LEs en el mismo LAB. Las conexiones de cadena de registro transfiere registro de un LE a otro LE adyacente dentro de un LAB figura 2.15. El compilador Quartus II

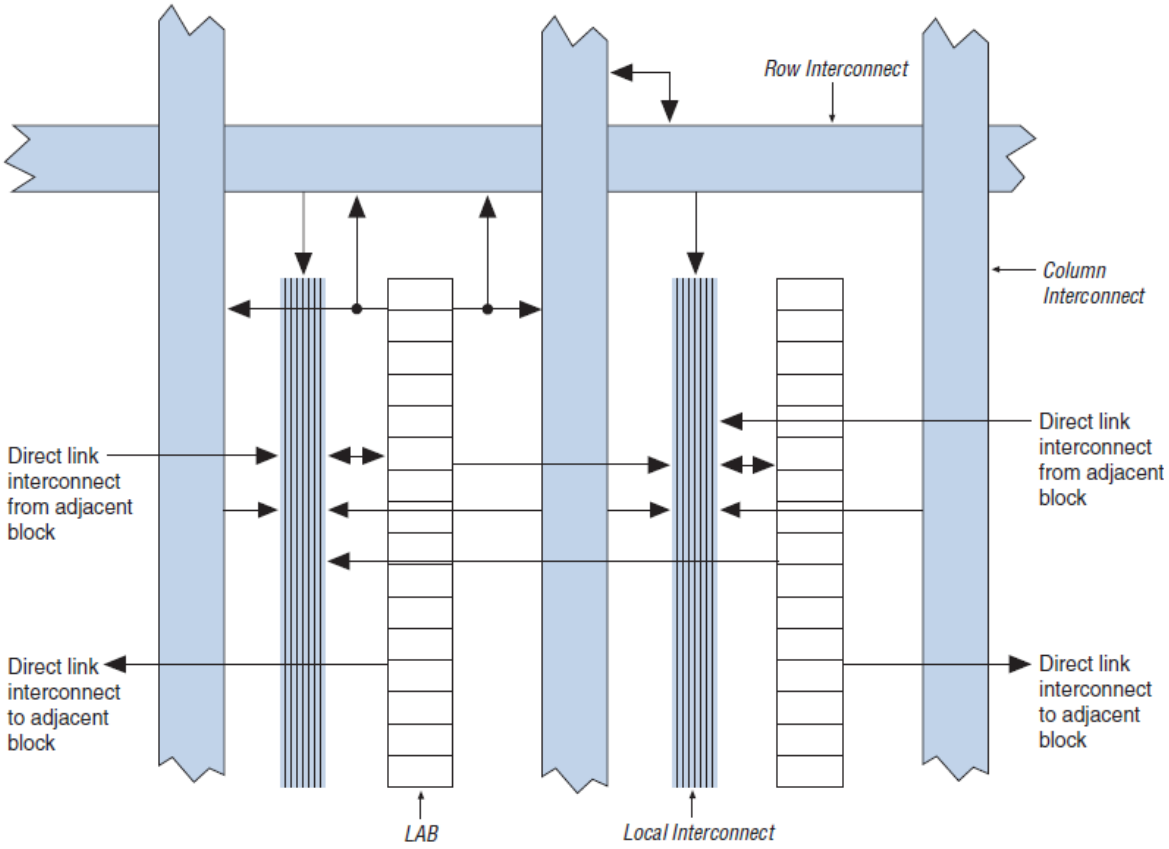


Figura 2.15 arquitectura de un bloque lógico configurable FPGA

Interconexiones LAB:
La interconexión local LAB puede controlar LEs dentro del mismo LAB.

Red global de reloj y bucle de enganche de fase

Los dispositivos Cyclone II proporcionan red global de reloj y hasta cuatro PLL para una solución completa de administración de reloj figura 2.16. La red de reloj incluye:

- 16 redes de reloj global
- Cuatro PLL
- Red global de reloj dinámico de selección de fuente de reloj
- Red global de reloj dinámico de habilitación y deshabilitación

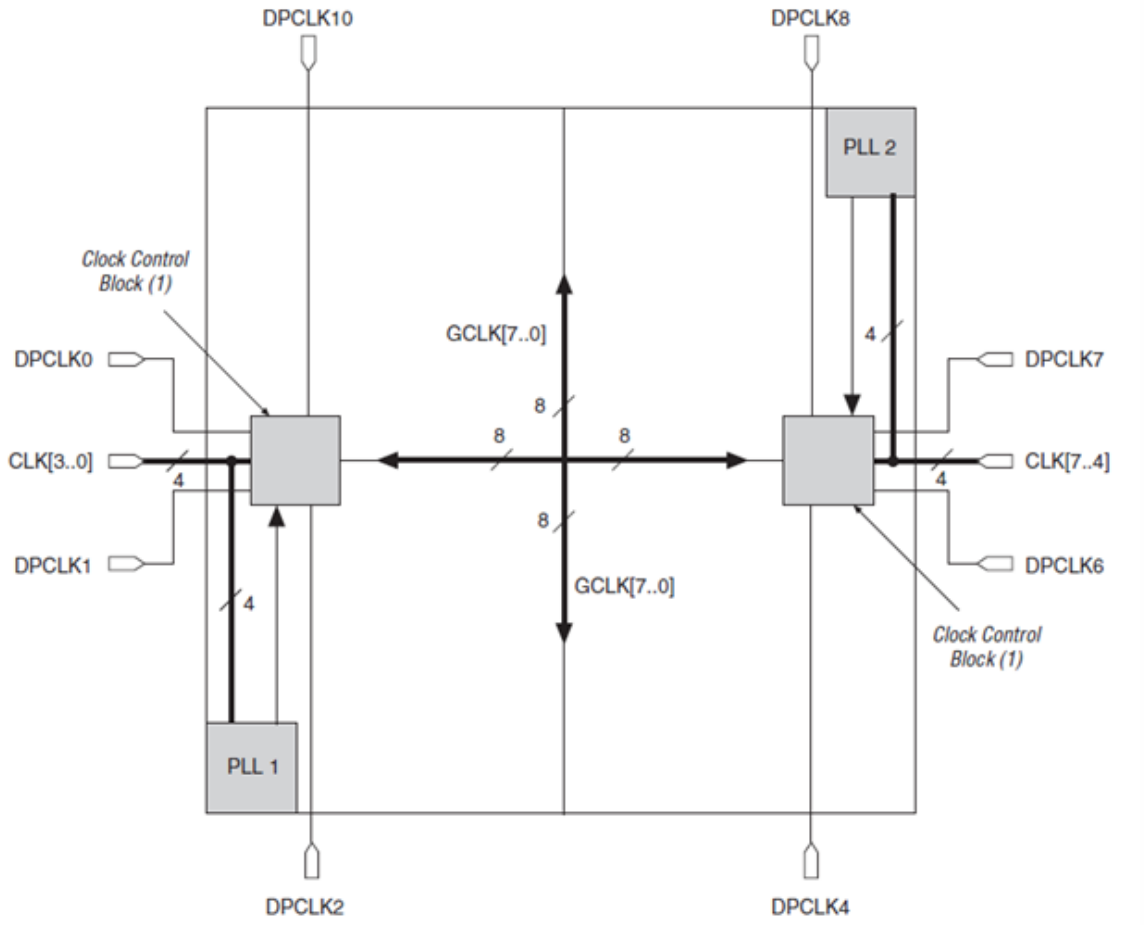


Figura 2.16 localización de los bloques de controlan el reloj

2.4.4 EL KIT ALTERA DE2

La figura 2.17 muestra la tarjeta DE2 y sus componentes:

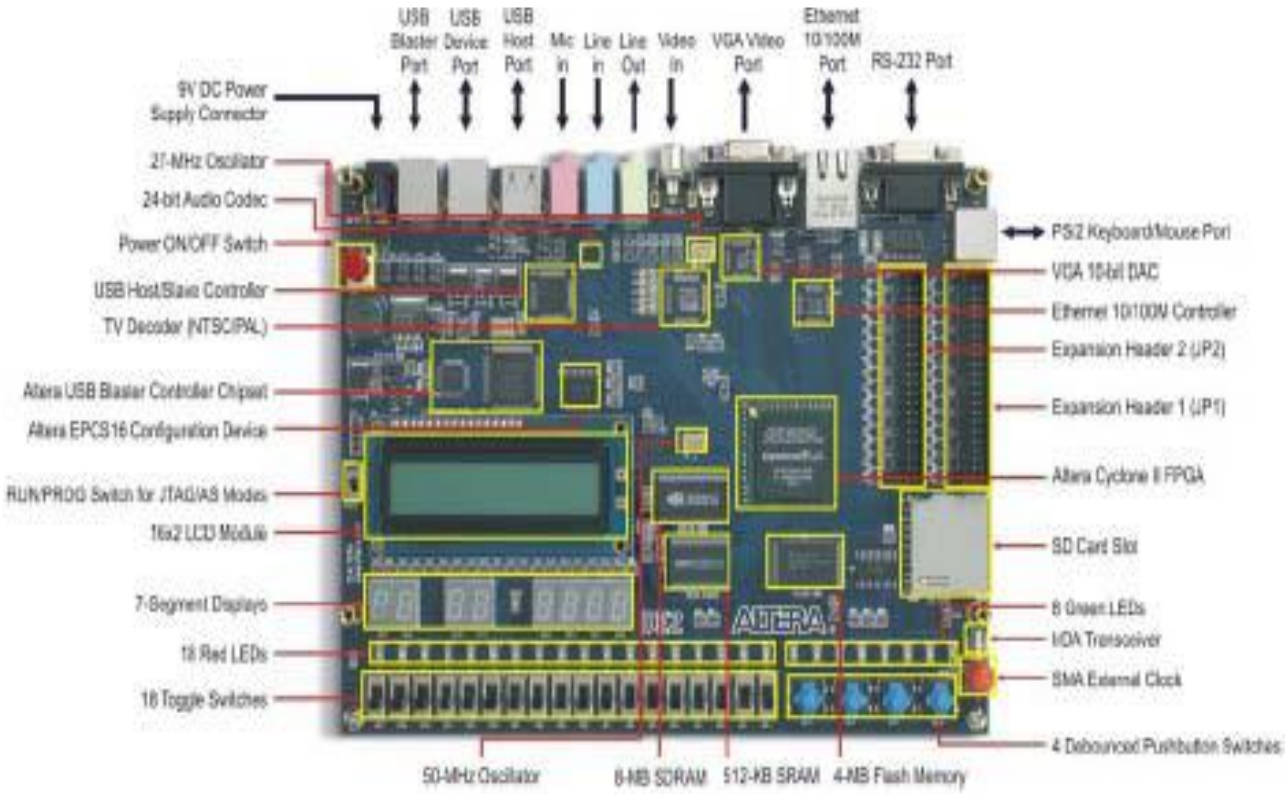


Figura 2.17 kit altera DE2 físicamente

La tarjeta de desarrollo DE2 contiene muchos accesorios, que le permiten al usuario implementar una gran variedad de diseños digitales, como contadores con display de 7 segmentos, entrada de video y audio , conexión a internet entre otras.

• FPGA Altera Cyclone® II 2C35
• Dispositivo de configuración serial Altera EPCS16
• USB Blaster (sobre la tarjeta) para programar el circuito bajo los protocolos JTAG y Active Serial
• SRAM de 512-Kbyte
• SDRAM de 8-Mbyte
• DE2 User Manual versión 5
• Memoria FLASH de 4-Mbyte
• Conector para SD Card
• 4 pushbutton switches
• 18 toggle switches
• 18 LEDs color rojo
• 9 LEDs color verde

• Oscilador de 50-MHz y 27-MHz como señales de reloj utilizables dentro de la tarjeta
• Codec de audio calidad CD de 24-bit con línea de entrada, línea de salida, y entrada de micrófono
• Codec VGA DAC (10-bit high-speed triple DACs) con un conector VGA de salida
• TV Decoder (NTSC/PAL) y un conector de entrada para TV
• Controlador y conector para 10/100 Ethernet
• Controlador de USB Host/Slave con conectores USB tipo A y tipo B
• Interface y Conector RS-232 de 9 terminales
• Conector PS/2 mouse/keyboard
• Interface IrDA
• Dos expansiones de 40 terminales con protección de diodo

El dispositivo FPGA de la tarjeta tiene las siguientes características: Cyclone II EP2C35
• 33,216 Logic Elements
• 105 M4K RAM blocks
• 483,840 bits de RAM embebidos
• 35 multiplicadores embebidos
• 4 PLLs
• 475 I/O pins configurables
• Empaquetado de tipo FineLine BGA de 672 (F672)

Asignación de terminales de Entrada/salida para usar Switches y Leds sobre la DE2.
Como ya vimos anteriormente, la DE2 cuenta con los siguientes elementos:
• 4 pushbutton switches
• 18 toggle switches
• 18 LEDs color rojo
• 9 LEDs color verde

2.5 LENGUAJE VERILOG

2.5.1 SINTAXIS DEL LENGUAJE VERILOG

Verilog es un lenguaje de descripción hardware (Hardware Description Language, HDL) utilizado para describir sistemas digitales, tales como procesadores, memorias o un simple flip-flop. Esto significa que realmente un lenguaje de descripción hardware puede utilizarse para describir cualquier hardware (digital) a cualquier nivel.

Se puede diseñar sistemas sencillos como una compuerta AND o hasta sistemas complejos de más de un millón de transistores, tal es el caso de un procesador. Verilog es uno de los estándares HDL disponibles hoy en día en la industria para el diseño hardware. Este lenguaje nos permite la descripción del diseño a diferentes niveles, denominados niveles de abstracción.

Para implementar un código en Verilog se necesita especificar los componentes básicos para la sintaxis, en el ejemplo 2.1 muestra del lado izquierdo la sintaxis básica, y del lado derecho representa en código un simple flip-flop.

El nombre del modulo es ff y las señales son (d,clk,q,q_bar), cada vez que haya un pulso de reloj **q** tomara el valor de **d** y **q_bar** tomara el valor negado de **d**.

Ejemplo 2.1

module <nombredelmodulo> (<señales>));	**module** ff (d,clk,q,q_bar);
<declaraciones de señales>	**input** d,clk; **output** q,q_bar;

<funciones del modulo>	**always @**(**posedge** clk)
	begin
	q <= d;
	q_bar <= !d;
	end

endmodule	**endmodule**

NIVELES DE ABSTRACCIÓN EN VERILOG

Verilog soporta el diseño de un circuito a diferentes niveles de abstracción, entre los que destacan:

Nivel de compuerta. Corresponde a una descripción a bajo nivel del diseño, también denominada modelo estructural. El diseñador describe el diseño mediante el uso de primitivas lógicas (AND, OR, NOT, etc.), conexiones lógicas y añadiendo las propiedades de tiempo de las diferentes primitivas. Todas las señales son discretas, pudiendo tomar únicamente los valores '0', '1', 'X' o 'Z' (siendo 'X' estado indefinido y 'Z' estado de alta impedancia) el ejemplo 2.2 muestra un circuito en código Verilog y la figura 2.18 muestra el circuito esquematico.

Ejemplo 2.2

```
// descripción del circuito simple
module circuito_ejemplo(A,B,C,x,y);
input A,B,C;
output x,y;
wire e;
and  #(30) U1(e,A,B);
not  #(20) U2(y,C);
or   #(10) U3(x,e,y);

endmodule
```

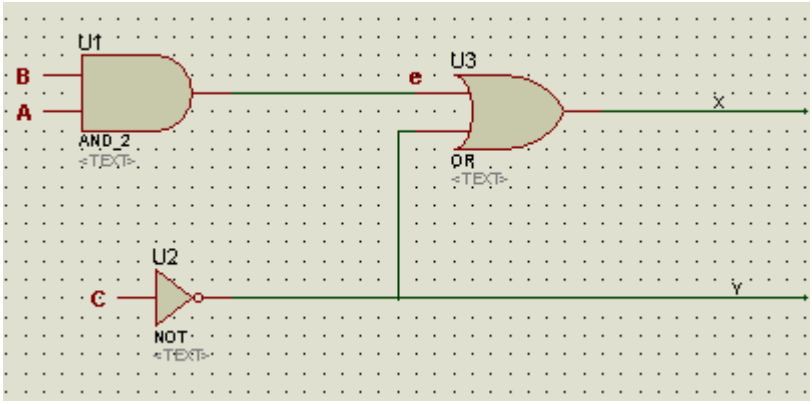
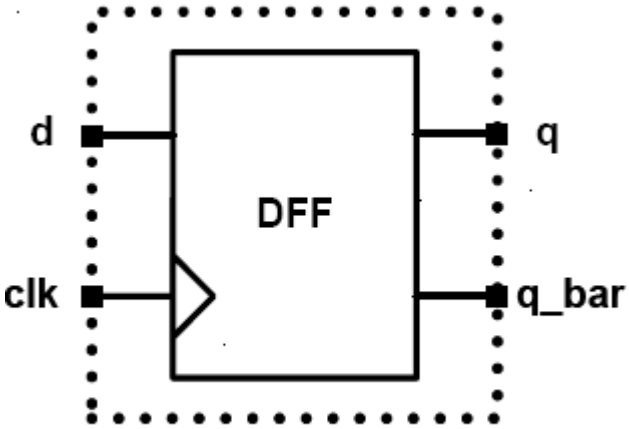


Figura 2.18 circuito para ilustrar el ejemplo anterior

NIVEL DE TRANSFERENCIA DE REGISTRO O NIVEL RTL.

Los diseños descritos a nivel RTL especifican las características de un circuito mediante operaciones y la transferencia de datos entre registros. Mediante el uso de especificaciones de tiempo las operaciones se realizan en instantes determinados. La especificación de un diseño a nivel RTL le confiere la propiedad de diseño sintetizable, por lo que hoy en día una moderna definición de diseño a nivel RTL es todo código sintetizable se denomina código RTL.
La figura 2.19 corresponde a la descripción a nivel RTL de un flip-flop. Este nivel de descripción, por la propiedad de ser sintetizable, será el nivel utilizado por excelencia en el diseño HDL.




```
module ff(d,clk,q,q_bar);
input d,clk;
output q,q_bar;
reg q,q_bar;
always @(posedge clk)
begin
q <= #1 d; // Retardo de 1 unidad
q_bar <= #1 !d; // Retardo de 1
//unidad
end
endmodule
```

Figura 2.19 Descripción a nivel RTL de un flip-flop.

NIVEL DE COMPORTAMIENTO (BEHAVIORAL LEVEL)

La principal característica de este nivel es su total independencia de la estructura del diseño. El diseñador, más que definir la estructura, define el comportamiento del diseño. En este nivel, el diseño se define mediante algoritmos en paralelo. Cada uno de estos algoritmos consiste en un conjunto de instrucciones que se ejecutan de forma secuencial. La descripción a este nivel pude hacer uso de sentencias o estructuras no sintetizables, y su uso se justifica en la realización de los denominados test bench (definidos más adelante).

CAPITULO 3

PROCESO DE DISEÑO

El objetivo de este primer acercamiento al manejo de una pantalla de VGA desde la perspectiva de los sistemas digitales es adquirir conocimientos elementales sobre el funcionamiento de un controlador básico de video, no se pretende desplegar imágenes complicadas sino sencillas líneas que ilustren como estos dispositivos procesan la información para exhibirla. El despliegue de una sola línea bastara para intuir como combinar muchas de ellas y obtener sencillas figuras geométricas o letras. La herramienta a utilizar es el lenguaje de programación de sistemas digitales de alta escala de integración conocido como Verilog HDL.

Verilog HDL permite describir hardware de manera estructural y funcional con un alto nivel de abstracción.

El kit ALTERA DE2 cuenta con un reloj de 25 MHz, el cual será nuestra frecuencia principal y en base a ello sacaremos el número de pixeles por línea horizontal y la cantidad de línea vertical que conforman el frame.

3.1 DISEÑO DE SISTEMAS DIGITALES

DISEÑO DE UN PRODUCTO CON HARDWARE DIGITAL

Los productos con hardware digital contienen varios PCB con muchos chips y componentes como resistencias, capacitores, etc. El desarrollo de estos productos tiene inicio con la definición de la estructura global. Después se tienen que elegir los chips y componentes necesarios y se procede al diseño de PCB que los contendrá y conectara.

Si los chips elegidos son PLD o chips diseñados a la medida, entonces se tendrá que diseñar y simularlos antes de implementarlos en le PCB. En la actualidad hay muchos circuitos demasiados complejos como los que contiene una computadora, tarjetas madres, etc. Una manera de diseñar estos circuitos es dividirlos por bloques, esto nos permitirá diseñar bloques por bloques y luego se interconectan entre sí para formar el circuito completo. Este enfoque recibe el nombre de *divide y vencerás.*

Una vez establecidos los bloques el siguiente paso es el diseño con los chips adecuados, y luego de estar completos los bloques se simula el proyecto completo. Esto permite saber si las interconexiones entre ellos son correctas y si cumple con las especificaciones establecidas por el cliente. La figura 3.1 permite entender de manera visual el procedimiento de diseño en sistemas digitales, los pasos que deben seguirse para cumplir adecuadamente el objetivo de producto.

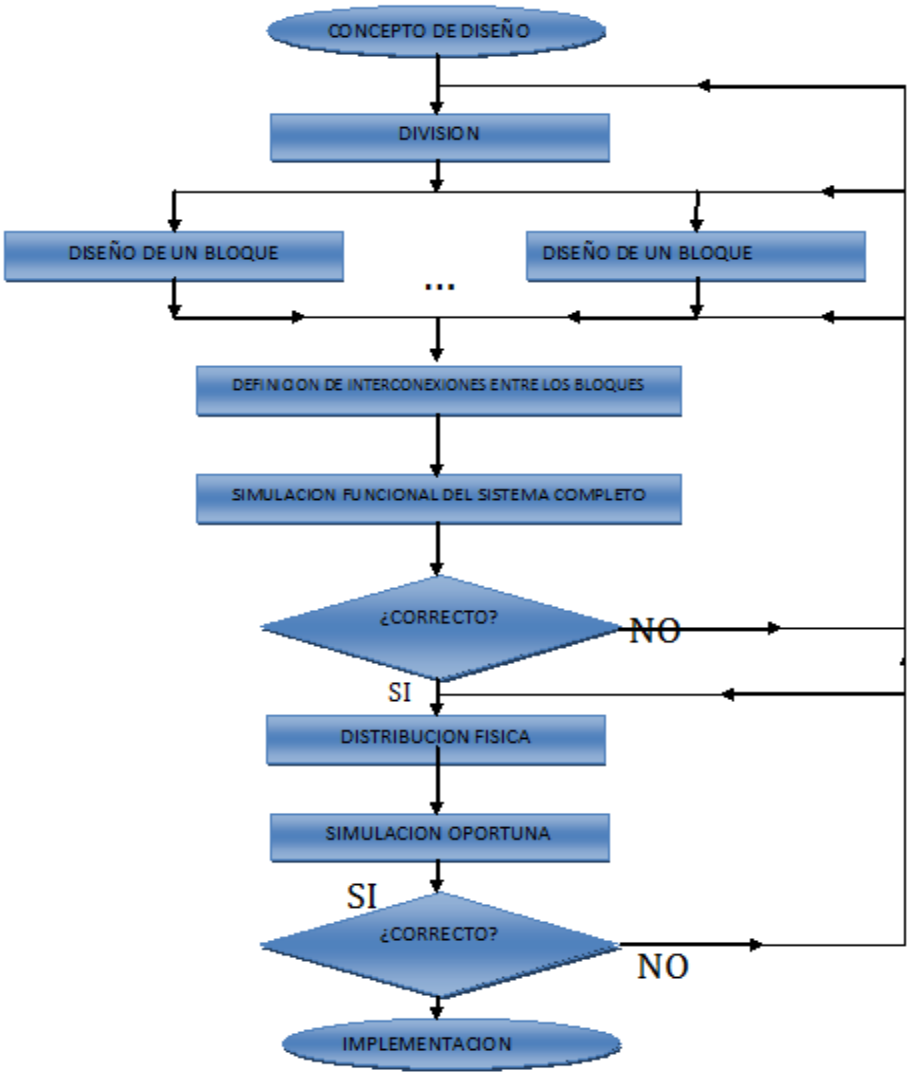


Figura 3.1 flujo diseño para circuitos lógicos.

3.2 DISEÑO Y CONTROL DE UN MONITOR VGA

Una de las opciones para el despliegue de imágenes utilizando FPGA es un monitor VGA, cuando se trabaja en procesamiento de imágenes, procesamiento de video, visión artificial, se necesita un equipo para visualizar las imágenes o video. Existen múltiples equipos para este propósito como pueden ser un monitor VGA, televisores de plasma, cañones y pantallas LCD entre otras.

Es por eso que para la realización de este proyecto se opto por una pantalla TFT de matriz de puntos, para poder desplegar imágenes sencillas en esta pantalla fue necesario, primero programar un monitor VGA para poder entender cómo se generan las señales de sincronía horizontal y vertical, así como las señales de blanqueo horizontal y vertical.

La figura 3.2 muestra los tiempos exactos para cada señal, con las ecuaciones 1.1 y 1.2 y a una frecuencia de 25Mhz se calculo el número de pixeles que hay en cada ciclo de tiempo.

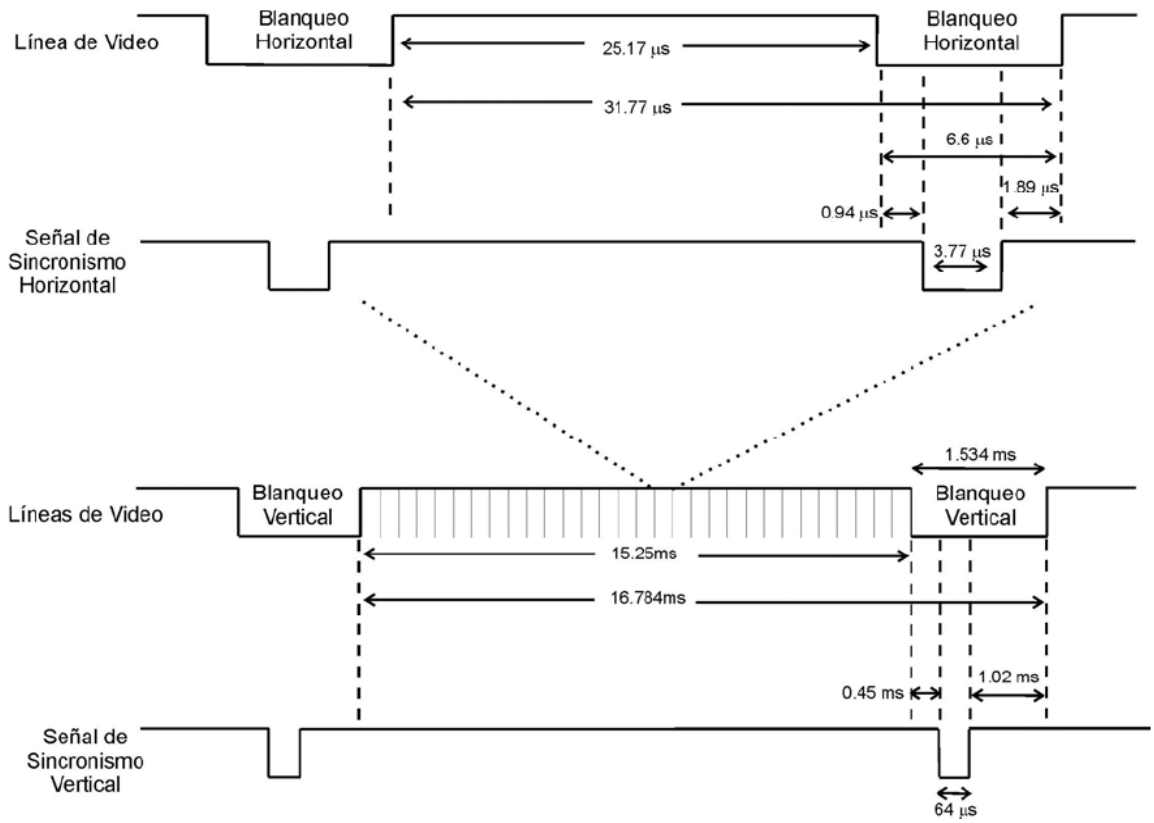


Figura 3.2 temporizacion de las señales

Con esta ecuación se calculo el número de pixeles que hay en la línea de video horizontal, a una frecuencia de 25 MHz, por tanto el periodo es de 40 ns.

$$cantidad\ de\ ciclos = \frac{THor}{periodo\ de\ frecuencia\ principal}$$

Tabla 3.1 Numero de pixeles en la línea horizontal.

Tiempo (µs)	Descripción	Tiempo dividido entre 40 ns	Cantidad de pixeles
31.77	Tiempo de exploración horizontal	794.25 pixeles	794 pixeles
25.17	Tiempo máximo en que se puede desplegar pixeles en una línea	629.25 pixeles	629 pixeles
0.94	Tiempo mínimo antes de aplicar el pulso de sincronización horizontal y después de desplegar el ultimo pixel	23.5 pixeles	24 pixeles
3.77	Tiempo en que el pulso de sincronización horizontal debe permanecer en un nivel lógico bajo.	94.2594 pixeles	94 pixeles
1.89	Tiempo mínimo antes de terminar el blanqueo horizontal	47.25 pixeles	47 pixeles

Para calcular el número de pixeles en la línea vertical se sigue el mismo procedimiento que se hizo con la línea horizontal, en este caso el periodo es tiempo que le lleva la exploración horizontal.

$$cantidad\ de\ ciclos = \frac{TVert}{tiempo\ en\ que\ se\ realiza\ la\ exploracion\ horizontal}$$

Tabla 3.2 numero de pixeles en el barrido vertical

Tiempo (ms)	Descripción	Tiempo dividido entre 31.77µs	Cantidad de Ciclos
16.784	Tiempo de exploración vertical	528.29 líneas	528 líneas
15.25	Tiempo máximo en que se puede desplegar todas las líneas en la pantalla	480.01 líneas	480 líneas
0.45	Tiempo mínimo antes de aplicar el pulso de sincronización vertical y después de desplegar el ultimo pixel	14.16 líneas	14 líneas
0.064	Tiempo en que el pulso de sincronización vertical debe permanecer en un nivel lógico bajo.	2.01 líneas	2 líneas
1.02	Tiempo mínimo antes de terminar el blanqueo vertical	32.1 líneas	32 líneas

El kit Altera DE2 cuenta con un DAC que nos permitirá obtener señales analógicas RGB, el kit incluye un conector D-SUB de 16 pines para las salidas al VGA figura 3.4. Las señales de sincronización del VGA provienen directamente del Cyclone II FPGA y el DAC ADV7123. El DAC de 10 bits se usa para producir las señales de datos analógicos (el verde, rojo y azul) figura3.3.

La figura 3.3 muestra el diagrama esquemático del DAC y puede soportar resoluciones de hasta de 1600 x 1200 pixeles en 100 MHZ.

LA DESCRIPCIÓN GENERAL

El ADV7123 (ADV ®) es un chip de velocidad alta, digital-analógico.
El convertidor en un solo chip monolítico.
Consiste en tres velocidades altas, convertidores D/A de 10 bits, de vídeo con salida complementaria, una interfaz estándar de aporte de lógica transistor-transistor y una impedancia alta, La fuente analógica de la corriente de salida.

El ADV7123 tiene tres terminales separadas de entrada de 10 bits para cada puerto.
Un solo suministro de voltaje de +5 V / +3.3 V y una reloj todo lo que requiere para hacer la parte funcional.
El ADV7123 tiene señales de control de vídeo adicionales, *SYNC y BLANK*.
El ADV7123 también tiene un power-save modo.
El ADV7123 es fabricado en un proceso del semiconductor complementario de óxido metálico +5 V.
Su construcción monolítica del semiconductor complementario de óxido metálico asegura mayor funcionabilidad
Con disipación más bajo de poder.
El ADV7123 está disponible en un paquete LQFP de 48 patitas.

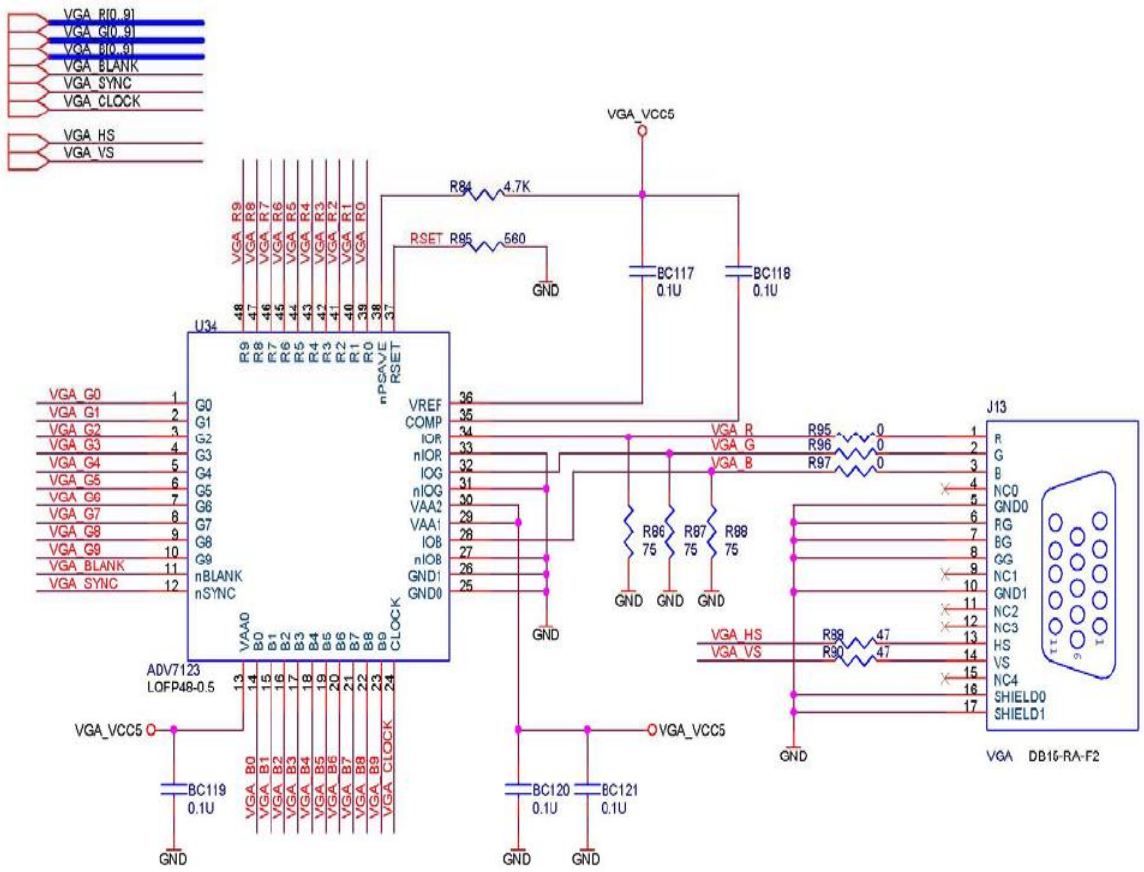


Figura 3.3 diagrama esquemático del DAC ADV7123

BLANK

Compuesto de entrada de control en blanco (TTL compatible). Un cero lógico en su entrada de control de las unidades de las salidas analógicas, IOR, IOB y GSI, en el nivel de blanking. La señal en blanco es normalmente, en el flanco de subida de reloj. Mientras

En blanco es un cero lógico, el R0-R9, G0-G9 y R0-insumos R9 píxel son ignorados.

SYNC

De control compuesto de sincronización de entrada (TTL compatible). Un cero lógico en los interruptores de entrada SYNC fuera IRE a 40 fuente de corriente. Esto está conectado internamente a la salida analógica Grupo de Supervisión Interna. SYNC no anula cualquier otra el control o la entrada de datos, por lo tanto, sólo debe hacerse valer durante el intervalo de supresión. SYNC está trabado en la flanco ascendente de reloj.

Si la información de sincronización no es necesaria en el canal verde, la entrada SYNC debe estar vinculada a cero lógico.

RELOJ

Entrada de reloj (TTL compatible). El flanco de subida del reloj asegura el R0-R9, G0-G9, B0-B9, SYNC y de píxeles en blanco y entradas de control. Es típicamente el tipo de reloj de píxeles del sistema de vídeo. Reloj debe impulsado por un buffer TTL dedicado.

R0-R9,

Rojo, verde y azul entradas de datos de píxeles (TTL compatible). Datos de los píxeles está trabado en el flanco de subida de reloj.

G0-G9,

R0, G0 y B0 son los bits menos significativos de datos. No utilizados píxel de los datos introducidos deben estar conectados a cualquiera de los

B0-B9

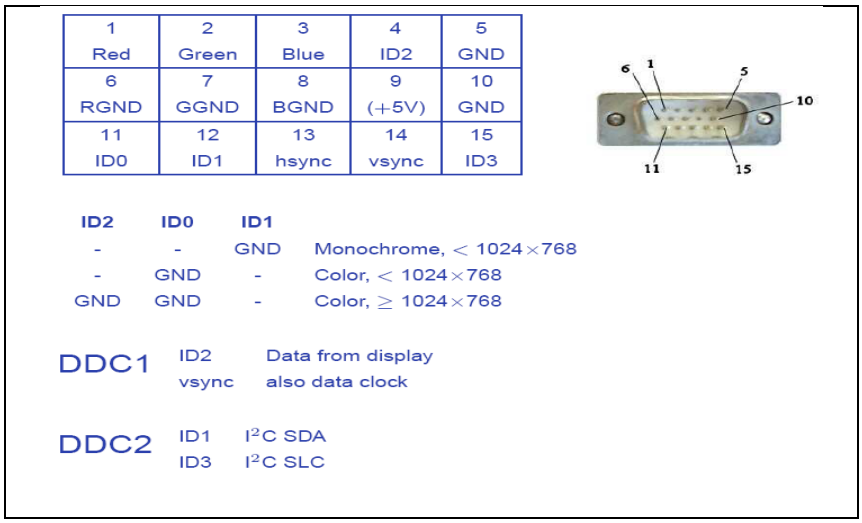
Regular el poder de PCB o plano del suelo.

IOR, GSI, IOB

Rojo, verde, azul y salidas de corriente. Estas fuentes de alta impedancia actuales son capaces de controlar un determinado cable coaxial de 75 Ω. Las tres salidas de corriente deben tener las cargas de producción similares o no que están siendo utilizados.

IOR, GSI, IOB

Salidas de corriente diferencial de rojo, verde y azul (las fuentes de corriente de alta impedancia). Estas salidas de vídeo RGB de se especifican directamente a la unidad RS-343A y RS-170 niveles de vídeo en una carga de 75 Ω determinado. Si el productos complementarios no son necesarios, estos productos deben estar atados a tierra.



1 Red	2 Green	3 Blue	4 ID2	5 GND
6 RGND	7 GGND	8 BGND	9 (+5V)	10 GND
11 ID0	12 ID1	13 hsync	14 vsync	15 ID3

ID2	ID0	ID1	
-	-	GND	Monochrome, < 1024×768
-	GND	-	Color, < 1024×768
GND	GND	-	Color, ≥ 1024×768

DDC1 ID2 Data from display
vsync also data clock

DDC2 ID1 I²C SDA
ID3 I²C SLC

Figura 3.4 descripción física de la salida VGA

Para diseñar el código tuvimos que definir los parámetros que serán las constantes del programa, estos parámetros son homólogos a los que se utilizan en la industria. El número total de pixeles en una solo línea son de 800 pixeles de los cuales 640 son activos por linea. El número de líneas totales son 525 líneas de frame, de los cuales 480 líneas son visibles en la pantalla.

- **Parámetros del programa**

```
//Parámetros horizontales      ( Pixel )
parameter      H_SYNC_CYC   =      96;      // pixeles de sincronización horizontal
parameter      H_SYNC_BACK  =      45+3;    // pixels porche trasero + porche delantero
parameter      H_SYNC_ACT   =      640;     // pixels de video
parameter      H_SYNC_FRONT=       13+3;    // borde derecho + borde izquierdo
parameter      H_SYNC_TOTAL=       800;     // pixeles por línea

//       Parámetros verticales   ( Line )
parameter      V_SYNC_CYC   =      2;       // líneas de porche delantero
parameter      V_SYNC_BACK  =      35+8;    // porche trasero + borde inferior
parameter      V_SYNC_ACT   =      480;     // lines visible
parameter      V_SYNC_FRONT=       8+2;     //borde superior + porche delantero
parameter      V_SYNC_TOTAL=       525;     // líneas por pantalla

//       rectangulo pequeño
parameter      X_START                =      H_SYNC_CYC+H_SYNC_BACK;
parameter      Y_START                =      V_SYNC_CYC+V_SYNC_BACK;
```

Con estos datos se empezó a diseñar el código para generar imágenes sencillas en el monitor VGA. El diseño del código se presenta en el capitulo siguiente.

Se presenta parte del código, donde se generan las señales de sincronía horizontal y vertical.

El generador de H_Sync es un proceso que sirvió para contar el número de pixeles por línea.

El generador de V_Sync es un proceso que sirvió para contar el número de líneas verticales.

3.2.1 SEÑALES DE SINCRONÍA VERTICAL Y HORIZONTAL PARA EL MONITOR VGA

Se diseño un proceso para generar las señales de control horizontal dentro del código para contar el numero de pixeles por línea horizontal, el contador está definido dentro del proceso como **H_Cont**, esto permitió que se comportara como un reloj para la línea horizontal y poner en nivel alto o bajo la señal de sincronía horizontal, cada vez que el contador llega a 800 empieza a contar desde cero e incrementa en uno a **V_Cont,** que es un contador de líneas verticales.

Cada vez que se presente un pulso alto de reloj en **iCLK** o un pluso bajo de **iRST** el proceso se activa y ejecuta las ordenes que están dentro de este proceso.

Si **iRST** tiene un valor de cero, entonces **H_Cont** toma el valor de cero y **VGA_H_SYNC** que es la señal de sincronía horizontal tomara el valor de cero, en caso contrario **H_Cont** se incrementara en uno.

Si **H_Cont** es menor que **H_SYNC_TOTAL** entonces se incrementa en uno **H_Cont** en caso contrario hace a **H_Cont** igual a cero.

Si **H_Cont** es menor que **H_SYNC_CYC** entonces **VGA_H_SYNC** toma un nivel lógico 0, en caso contrario toma el valor lógico 1.

Este es el proceso que se realiza en el siguiente código:

```
// Generador de H_Sync , Ref. 25MHz Clock

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        begin
                H_Cont<=0;
                VGA_H_SYNC<=0;
        end
        else
        begin

//      Contador de H_Sync
                if( H_Cont < H_SYNC_TOTAL )
                H_Cont<=H_Cont+1;
                else
                H_Cont<=0;



//      Generador de H_Sync
                if( H_Cont < H_SYNC_CYC )
                VGA_H_SYNC<=0;
                else
                VGA_H_SYNC<=1;
                endend
```

Se diseño otro proceso que generara las señales de sincronía vertical **V_Sync,** este proceso permitió contar el número de líneas verticales, cada vez que **H_Cont** toma el valor de 800, se incrementa en uno **V_Cont.** La señal de sincronía vertical **VGA_V_SYNC** toma el valor lógico 0 y valor lógico 1.

Cada vez que se presente un pulso alto de reloj en **iCLK** o un pluso bajo de **iRST** el proceso se activa y ejecuta las ordenes que están dentro de este proceso.

Si **iRST** tiene un valor de cero, entonces **V_Cont** toma el valor de cero y **VGA_V_SYNC** que es la señal de sincronía vertical toma el valor de cero, en caso contrario **V_Cont** se incrementara en uno.

Cuando **H_Cont** tome el valor de 0, se realiza el siguiente paso:

Si **V_Cont** es menor que **V_SYNC_TOTAL** entonces se incrementa en uno **V_Cont** en caso contrario hace a **V_Cont** igual a cero.

Si **V_Cont** es menor que **V_SYNC_CYC** entonces **VGA_H_SYNC** toma un nivel lógico 0, en caso contrario toma el valor lógico 1.

Esto es lo que hace el siguiente código para generar las señales de sincronía vertical.

```
//Generador de V_Sync, Ref. H_Sync

always@(posedge iCLK or negedge iRST)
begin
        if(!iRST)
        begin
                V_Cont<=0;
                VGA_V_SYNC<=0;
        end
        else
        begin

//      Cuando H_Sync Re-start
                if(H_Cont==0)
                begin
//      Contador V_Sync
                if( V_Cont < V_SYNC_TOTAL )
                        V_Cont <= V_Cont+1;
                        else
                        V_Cont<=0;
//      Generador V_Sync
        if(     V_Cont < V_SYNC_CYC )
                VGA_V_SYNC<=0;
                        else
                VGA_V_SYNC<=1;
                end
        end
end
```

3.3 PANTALLA TRDB-LCM DE TERASIC

Descripción de los pines de los conectores

Para poder desplegar imágenes digitales en la pantalla se tienen que programar los parámetro que requiere la pantalla, algunos de los cuales son necesarios para poderse comunicar con el kit altera DE2. A continuación se muestran una tabla donde se resumen los pines que van conectados a kit y las señales de que deben programarse.

Una vez entendido como generar las señales de sincronía vertical y horizontal, así como las señales de blanqueo, nuestro siguiente paso es el diseño del código para generar imágenes

sencillas en la pantalla LCD, en este caso para generar dichas imágenes fue necesario leer la hoja de especificaciones de la pantalla para poder entender cómo funciona.

La pantalla contiene unos drivers diseñados por el fabricante para poder configurar la pantalla, fue necesario entender cómo configurar la pantalla, ya que en base a ello se diseño el código en Verilog.

Características

La tecnología LTPS TFT (temperatura baja Poly Silicon) usa driver verticales y horizontales para conectar el panel la figura 3.5 muestra el diagrama a boloques.
El escaneo puede ser horizontal, de izquierda a derecha o de derecha a izquierda, y el escaneo Vertical también se hace de abajo hacia hasta arriba.

Manejo de la pantalla y diagramas de bloques.

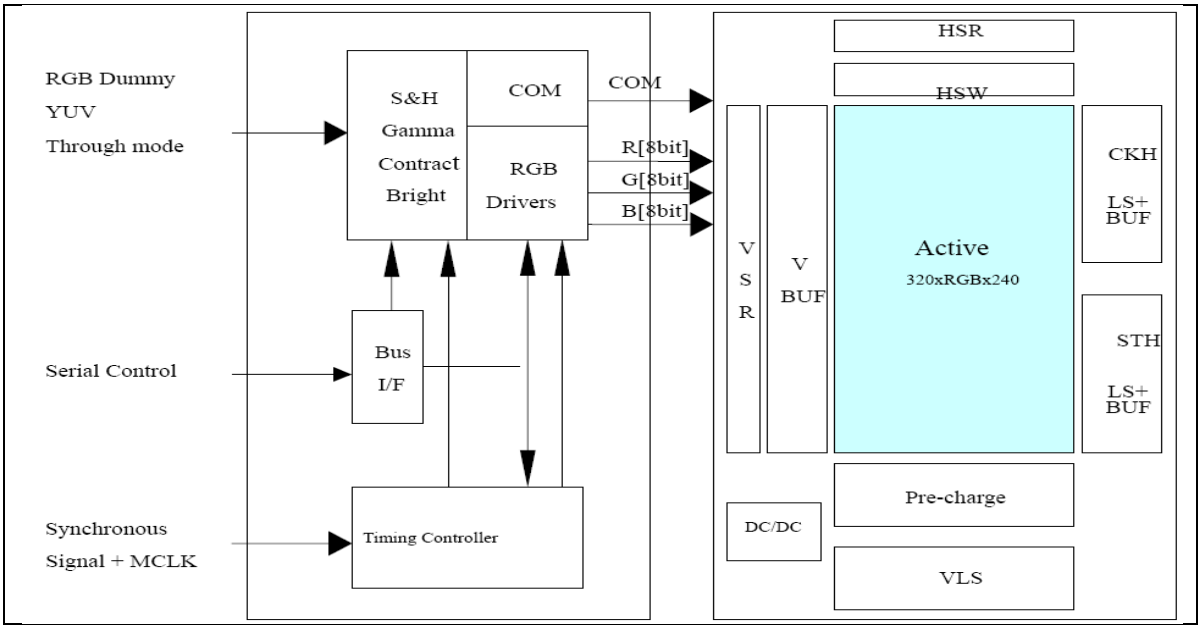


Figura 3.5 Diagrama a bloques de la pantalla LCD

En el diagrama a bloques se puede ver un puerto de control serial para comunicar y enviar, datos, un puerto para las señales de sincronía que requiere la pantalla y el periodo de la frecuencia principal de reloj, la pantalla soporta el tres modos para la configuración de imágenes; RGB dummy, YUV, through mode. Los cuales van conectados al drivers RGB para la proyección de imágenes.

3.3.1 RGB DRIVER/TIMING CONTROLLER IC PARA LA PANTALLA LTPS TFT LCD

Descripción

Éste es circuito integrado para el proceso de señales digitales (polisilicio de baja temperatura TFT-LCD).
Contiene 8 bits de datos de entrada (los datos son en la escala de grises de 256 niveles) para cada uno de los formatos RGB o YUV colores.
La salida RGB analógico son señales internas que entregan los Opamp.

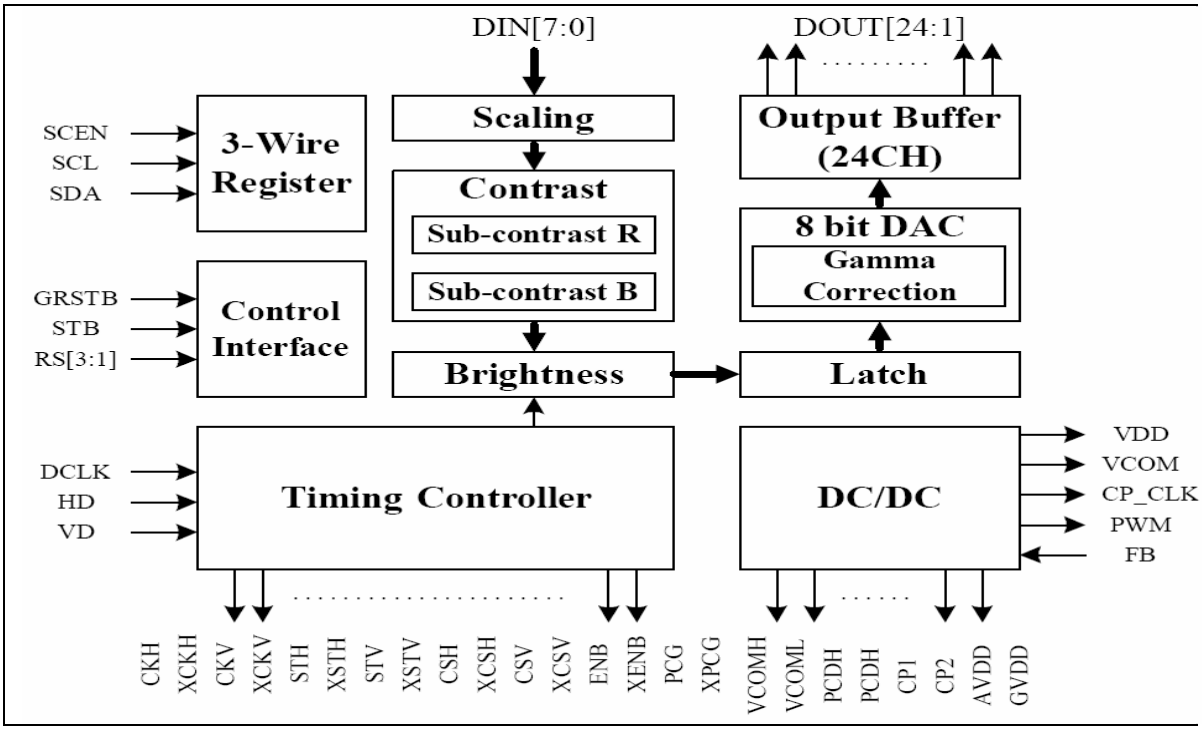


Figura 3.6 Diagrama a bloques de los controladores de la pantalla

.
Se tuvo que revisar datashet de la pantalla para poder generar las señales de sincronía horizontal y vertical, así como las señales de blanqueo. En el siguiente diagrama de tiempos explica con detalle los números de pixeles para cada señal.

Horizontal

Con información que se obtuvo del datashet de la pantalla se encontró el siguiente diagrama de tiempos, los valores son dados por el fabricante, lo que se hizo fue interpretar los datos; donde un pulso de reloj representa un pixel. HD es el periodo para una línea horizontal, donde una línea horizontal (t_h) tiene un total de 1171 pixeles, de los cuales 960 pixeles conforman el área activa (t_{hd}), el ancho de pulso de HSYNC (t_{hpw}) es de de 1 pulso de reloj, la señal de blanqueo horizontal (t_{hb}) es de 152 pixeles y por último el porche delantero (t_{hfp}) contiene 59 pixeles. La tabla 3.3 muestra los valores que toman estos parámetros a una frecuencia de 18.42 MHz

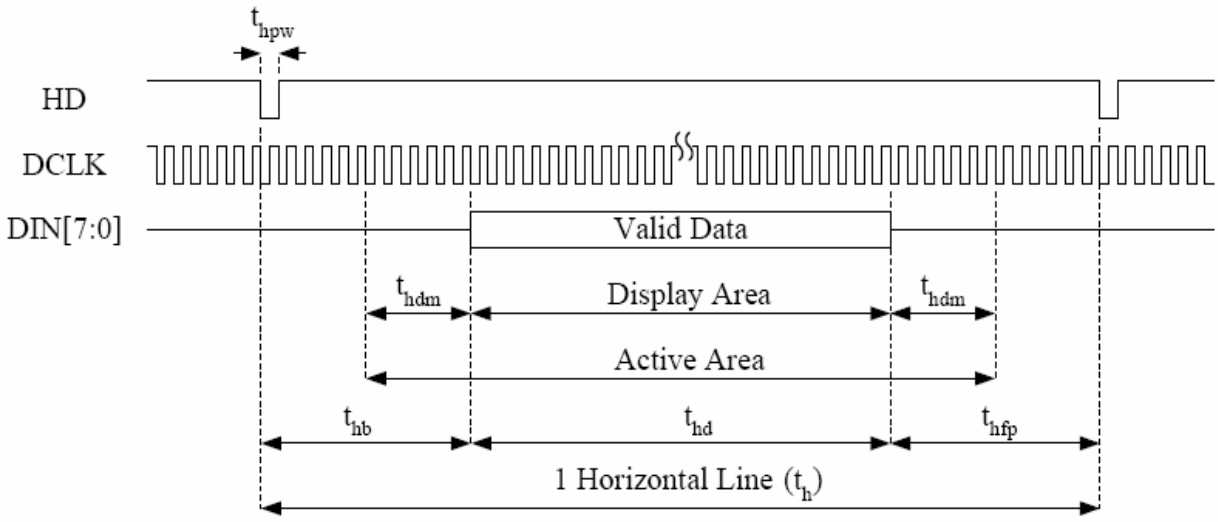


Figura 3.7 Diagrama de tiempos de la pantalla LCD.

La siguiente tabla 3.3 muestra los números de pixeles de cada señal respecto al diagrama de tiempo que se ve en la parte superior, se trabajo con una frecuencia de reloj de 18.42 MHz que se puede ver donde esta **panel resolution** de la tabla 3.3

Tabla 3.3 numero de pixeles para cada una de las señales de la pantalla LCD.

Parameter		Symbol	Panel Resolution					Unit
DCLK Frequency		F_{DCLK}	10.36	11.63	12.90	14.18	18.42	MHz
Horizontal valid data		t_{hd}	492	558	640	720	960	DCLK
1 Horizontal Line		t_h	659	739	820	901	1171	DCLK
HSYNC Pulse Width	Min.	t_{hpw}	1					DCLK
	Typ.		1					
	Max.		-					
Hsync blanking		t_{hp}	102	113	117	122	152	DCLK
Hsync front porch		t_{hfp}	65	68	63	59	59	DCLK
Horizontal dummy time		t_{hdm}	6	9	4	0	0	DCLK

Vertical

Para las señales verticales también se hizo lo mismo, revisar el datashet para las temporizaciones de las señales.

ODD Field: guarda la fase de VD (periodo vertical) y HD (periodo horizontal)

EVEN Field: guarda la fase VD y Half-H

Un frame figura 3.8 está compuesta por un periodo de Odd Field (t_v) y un periodo de Even Field (t_v) cada uno de estos tiene 262 periodos HD, los datos verticales validos (t_{vd}) contiene 240 periodos HD, el ancho de pulso (t_{vpw}) es de un pulso de reloj, la señal de blanqueo vertical es de 14 HD y la señal de sincronía vertical de porche delantero (t_{vfpe}) es de 8 HD.

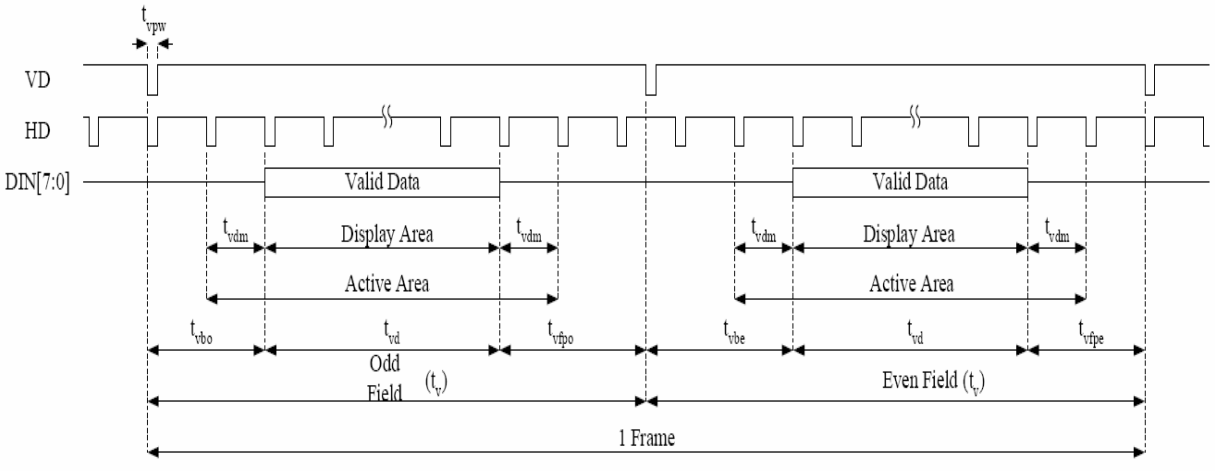


Figure 3.8 Diagrama de tiempos para las señales verticales

Todos estos datos fueron sacados de datashet de la pantalla y se trabajo con los datos que se encuentra la columna **non interlace** por ser números enteros, esto facilito la programación.

La siguiente tabla 3.4 muestra los números de periodos HD para las señales verticales en la columna **Non-interlace.**

Tabla 3.4 numero de pixeles para las señales verticales

Parameter		Symbol	Interlace	Non-interlace	Unit
Vertical valid data		t_{vd}	240	240	H
1 Vertical field		t_v	262.5	262	H
Vsync pulse width	Min.	t_{vpw}	1	1	DCLK
	Typ.		1	1	DCLK
	Max.		-	-	H
Vsync blanking	Odd field	t_{vbo}	14	14	H
	Even field	t_{vbe}	14.5	14	H
Vsync front porch	Odd field	t_{vfpo}	8.5	8	H
	Even field	t_{vfpe}	8	8	H
Vertical dummy time		t_{vdm}	0	0	H

El siguiente diagrama a bloque muestra las conexiones entre una cámara y la pantalla, se uso este diagrama para diseñar el código.

El **CCD sensor** es el bloque de la cámara, se hizo referencia a esto ya que el bloque muestra la parte **RAW to RGB** que es donde se partirá para hacer el proyecto, se muestra el bloque de la cámara **CCD sensor** en este diagrama debido a que se hará trabajos posteriores con la cámara y la pantalla, por eso es de vital importancia familiarizarse con ello.

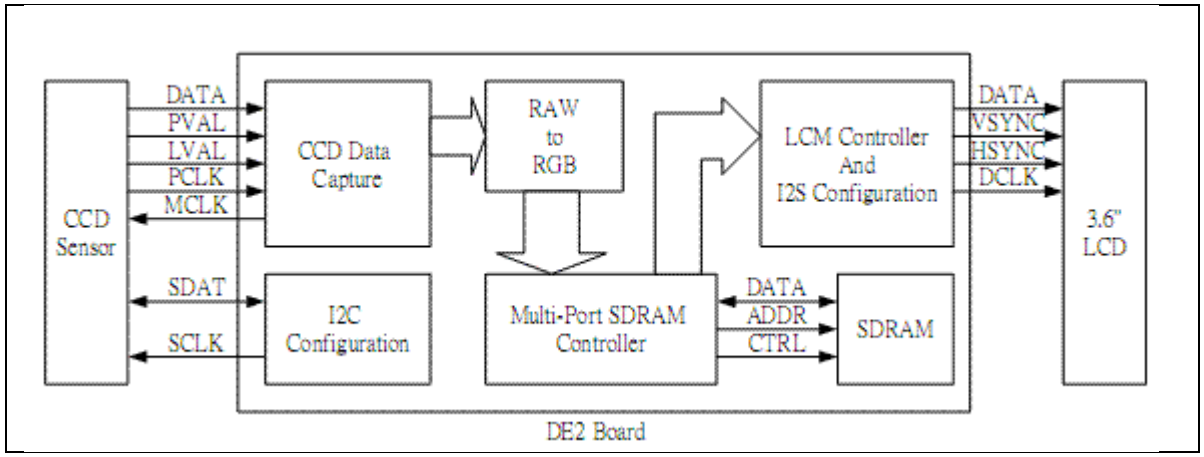


Figura 3.9 Diagrama a bloques de la pantalla y su configuración.

El bloque donde esta **LCM controller and I2S configuration** son los controladores de la pantalla para que pueda configurarse y pueda ver comunicación entre la pantalla y el kit. Como puede observar este bloque permitio el envió de las señales de sincronía horizontal y vertical, así como la señal de reloj.

El bloque **Multi-Port SDRAM Controller** permite el refresco de la pantalla y guardar los datos para enviarlos a la pantalla.

En este trabajo lo que se pretende es mandar imágenes sencillas RGB a la pantalla, para posteriormente utilizar para monitores de sistemas. Semejante a las cámara fotográficas que son capaces de detectar los rostros, detectores de movimiento.

3.3.2 CÓDIGO PARA LAS SEÑALES DE SINCRONÍA HORIZONTAL Y VERTICAL

Se diseño el siguiente código es para la generación de las señales de sincronía horizontal y vertical que necesita la pantalla para poder mandar imágenes, en esta parte se explica cómo generar esas señales con contadores. Este contador que se utiliza tiene una frecuencia de 18.42 MHz para poder enviar bien los datos, para lograr esto se hizo uso de las librerías que nos facilita el software para poder diseñar módulos específicos.

```
//      parametros horizontales      ( Pixel )
parameter     H_SYNC_CYC   =     1;
parameter     H_SYNC_BACK  =     151;
parameter     H_SYNC_ACT   =     960;
parameter     H_SYNC_FRONT=      59;
parameter     H_SYNC_TOTAL=      1171;

//      parametros verticales        ( Line )
parameter     V_SYNC_CYC   =     1;
parameter     V_SYNC_BACK  =     13;
parameter     V_SYNC_ACT   =     240;
parameter     V_SYNC_FRONT=      8;
parameter     V_SYNC_TOTAL=      262;

//      Start Offset
parameter     X_START              =     H_SYNC_CYC+H_SYNC_BACK;
parameter     Y_START              =     V_SYNC_CYC+V_SYNC_BACK;
```

Se diseño un proceso **H_Sync Generator** para generar las señales de sincronía horizontal con una señal de reloj de 18.42 MHz, cuando se presente un pulso alto en **iCLK** o un pulso bajo de **iRST_N** se ejecuta las ordenes que están dentro del proceso.

Si **iRST_N** tiene un valor lógico 0, entonces hace a **H_Cont** y **oVGA_H_SYNC** igual a cero .

Si **H_Cont** es menor que **H_SYNC_TOTAL** (1171) entonces **H_Cont** se incrementa en uno, en caso contrario **H_Cont** toma el valor de 0.

Si **H_Cont** es menor que **H_SYNC_CYC** entonces **oVGA_H_SYNC** toma un valor lógico 0, en caso contrario toma un valor lógico 1.

código para generar señales de sincronía horizontal y vertical

```
//H_Sync Generator, Ref. 18.42 MHz Clock

always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                H_Cont<=0;
                oVGA_H_SYNC <=0;
        end
        else
        begin
//      H_Sync Counter
                if( H_Cont < H_SYNC_TOTAL )
                H_Cont<=H_Cont+1;
                else
                H_Cont<=0;
//      H_Sync Generator
                if( H_Cont < H_SYNC_CYC )
                oVGA_H_SYNC <=0;
                else
                oVGA_H_SYNC <=1;
        end
end
```

Se diseño un proceso **V_Sync Generator** para generar las señales de sincronía vertical, cuando se presente un pulso alto en **iCLK** o un pulso bajo de **iRST_N** se ejecuta las ordenes que están dentro del proceso.

Si **iRST_N** tiene un valor lógico 0, entonces hace a **V_Cont** y **oVGA_V_SYNC** igual a cero .

Si **H_Cont** es igual a cero entonces se realiza lo siguiente:

Si **V_Cont** es menor que **V_SYNC_TOTAL** (262) entonces **V_Cont** se incrementa en uno, en caso contrario **V_Cont** toma el valor de 0.

Si **V_Cont** es menor que **V_SYNC_CYC** entonces **oVGA_V_SYNC** toma un valor lógico 0, en caso contrario toma un valor lógico 1.

```
//V_Sync Generator, Ref. H_Sync

always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                V_Cont <=0;
                oVGA_V_SYNC <=0;
        end
        else
        begin
//      When H_Sync Re-start
                if(H_Cont==0)
                begin
//      V_Sync Counter
                if( V_Cont < V_SYNC_TOTAL )
                V_Cont <=V_Cont+1;
                        else
                        V_Cont <=0;
//      V_Sync Generator
        if(     V_Cont < V_SYNC_CYC )
                oVGA_V_SYNC <=0;
                        else
                oVGA_V_SYNC <=1;
                end
        end
end
```

Este código sirvió para generar las señales de sincronía necesarias para la pantalla, con esto se tuvo el control de las líneas verticales y de los pixeles por línea. Para generar el reloj de 18.42 MHz se utilizo la herramienta ALTPLL. Donde se puede diseñar contadores, con solo especificar la frecuencia del reloj deseado.

Se diseñó un contador con una frecuencia de 18.42 MHz, para esto se utilizo la herramienta **MegaWizard Plug-in Manager.**

Para generar esta señal de reloj se siguió estos pasos:

Abrir **MegaWizard Plug-in Manager**

Luego en la casilla **what is the frecuency of the inclocko input?** Se especificó el valor de la frecuencia principal del kit, en este caso es de 50 MHz figura 3.10.

En este caso **.inclk0** es la entrada y se selecciono **c0** que está en la parte inferior como salida

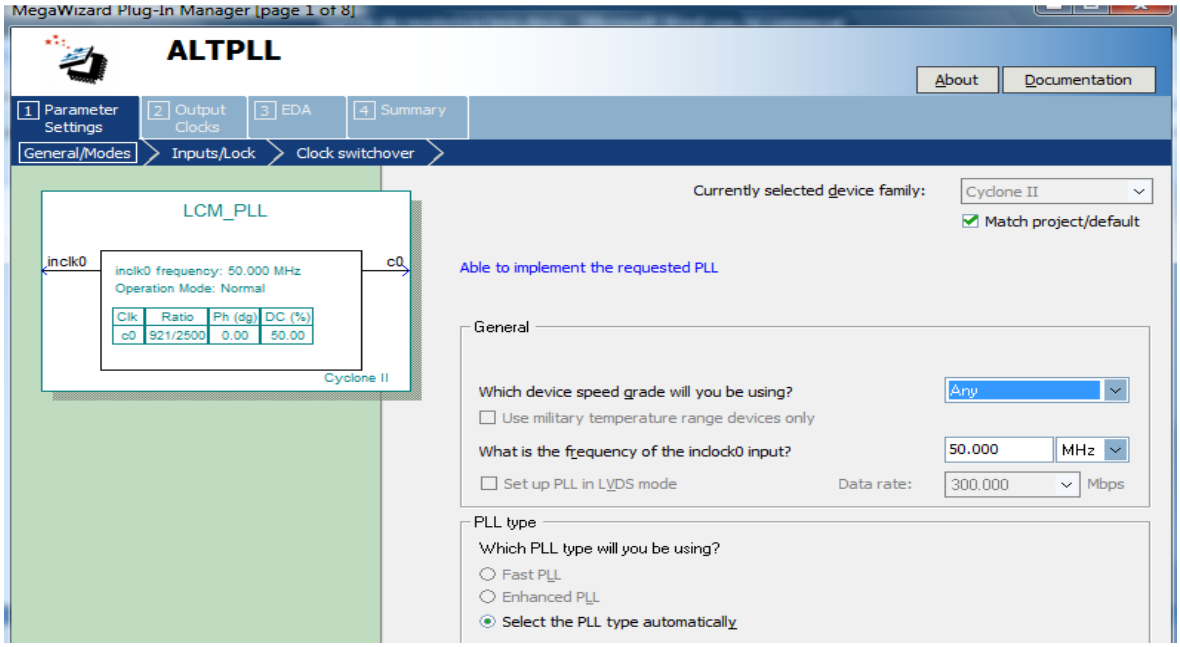


Figura 3.10 Utilización de la librería ALTPLL

Después de haber llenado la casilla le damos click en el icono con la leyenda **next.** Y la figura 3.11 aparece, en la casilla **Enter output clock frecuency** se puso el valor de la frecuencia de reloj de **18.42MHz** figura 3.11

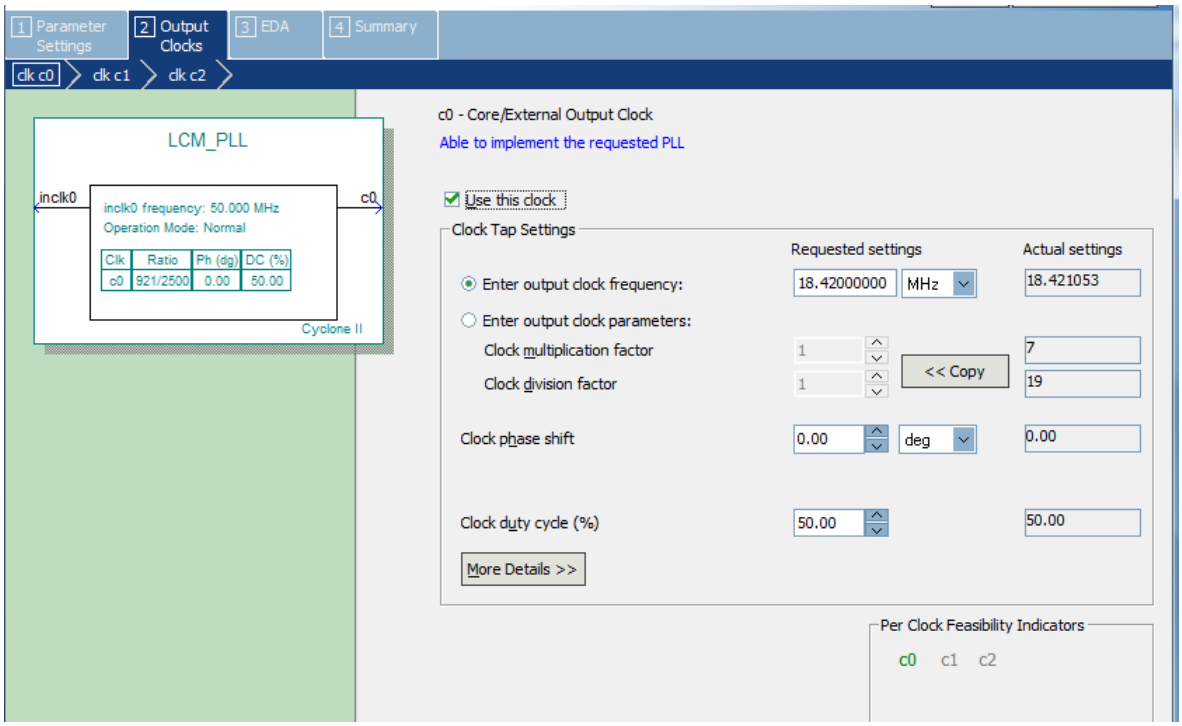


Figura 3.11 Diseño de un reloj con frecuencia de 18.42 MHz

Esta parte de código sirvió para asignar la señal de entrada de reloj, la variable **inclk0** toma el valor de **CLOCK_50** que es la frecuencia principal del kit. El valor que tiene la salida **c0** pasara a la variable **CLK_25.**

```
LCM_PLL      u0      (.inclk0(CLOCK_50),.c0(CLK_25));
```

3.3.3 GENERACIÓN DE FONDO PARA LA PANTALLA LCD

Se diseño un código de tal manera que se pudiera desplegar una variedad de colores a partir de los colores ROJO, AZUL Y VERDE (RGB), estos colores son las bases para la generación de una gran cantidad de colores mediante software. Para este caso se genero un fondo de pantalla de color verde. Para generar otros colores solo se tiene que modificar los valores para:

```
imagen[0] <= 8'b11111111; //G
imagen[1] <= 8'b00000000; //B,
imagen[2] <= 8'b00000000; //R
```

Esta parte de código sirvió para generar un fondo de pantalla de color verde, cuando **iRST_N** tome el valor de cero, **Tmp_DATA** que es una variable de 8 bits tomara los valores de cero "**00000000**".

Las variables **coord_x** es un contador de pixeles y **coord_y** contador de líneas. Si **coord_x** entre el rango de (0,960) y **coord_y** este entre el rango (0,240) se genera un fondo de color verde en la pantalla determinada por **imagen[0]<= 8'b11111111; //G.**

```
begin

if(!iRST_N)
        begin
                Tmp_DATA<= 8'b0;
        end
        else
        begin
        if (coord_x >0 && coord_x < 960 &&  coord_y > 0 && coord_y < 240)
                begin

                        imagen[0]<= 8'b11111111; //G
                        imagen[1]<= 8'b00000000; //B
                        imagen[2]<= 8'b00000000; //R

                       Tmp_DATA<=          imagen[MOD_3];

                End
```

3.3.4 GENERACIÓN DE UN RECTÁNGULO SOBRE LA PANTALLA LCD

Se diseño un código de tal manera que se pudiera desplegar imágenes sencillas en la pantalla, para este caso se desplegara un rectángulo de color azul con un fondo de color verde. Para hacer esto se siguió el mismo procedimiento que se hizo para generar el fondo de color verde, para este caso se hizo el mismo procedimiento ya que es igual al anterior.

Para generar rectángulos con una gama variada de colores solo hay que modificar los valores

```
imagen[0]<= 8'b00000000; //G
imagen[1]<= 8'b11111111; //B
imagen[2]<= 8'b00000000; //R
```

Cabe señalar que, los valores que puede tomar cada color está entre 0-256 para cada color. Si **coord_x** está entre el rango (180,780) y **coord_y** esta entre el rango (35,205), la variable **imagen[1]** toma lo valores **"111111111"** esto genera un rectángulo de color azul.

```
if (coord_x >180 && coord_x < 780 &&  coord_y > 35 && coord_y < 205)
                begin


                    imagen[0]<= 8'b00000000; //G
                    imagen[1]<= 8'b11111111; //B
                    imagen[2]<= 8'b00000000; //R

                    Tmp_DATA<=   imagen[MOD_3];

                end
```

3.4 IMAGEN FINAL DESPLEGADA EN LA PANTALLA LCD

Por último se implemento un código que desplegara un rectángulo de color rojo dentro del rectángulo de color azul que se diseño con anterioridad, en este caso estará compuesto por un fondo de color verde que contendrá un rectángulo de color azul, este a su vez contendrá un rectángulo rojo, por último se desplegara una línea vertical y horizontal que dividirá la pantalla en cuatro zonas. Con este código se logro desplegar un rectángulo de color rojo con líneas horizontales y verticales que dividían la pantalla en cuatro zonas.

Si **coord_x** está entre el rango (380,580) y **coord_y** esta entre el rango (70,170) la variable **imagen[2]** toma los valores **"11111111"** que es el generador del cuadro rojo figura 4.4.

Si **coord_x** está entre el rango (0,960) y **coord_y** esta entre el rango (118,122), las variables **imagen[0], imagen[1], imagen[2],** toman el valor de **"00000000"**dando como resultado una línea vertical de color blanco figura 4.4. Si **coord_x** está entre el rango (476,484) y **coord_y** esta entre el rango (0,240), las variables **imagen[0], imagen[1], imagen[2],** toman el valor de **"00000000"**dando como resultado una línea horizontal de color blanco figura 4.4.

```
if (coord_x > 380 && coord_x < 580 &&  coord_y > 70 && coord_y < 170) ,
                begin

                    imagen[0]<= 8'b00000000; //G
                    imagen[1]<= 8'b00000000; //B
                    imagen[2]<= 8'b11111111; //R  generador de cuadro rojo

                    Tmp_DATA<=imagen[MOD_3];
                end

               if (coord_x > 0 && coord_x < 960 && coord_y > 118 && coord_y < 122)
               begin

                   imagen[0]<= 8'b00000000; //R  generador de lineas horizontales
                   imagen[1]<= 8'b00000000; //G
                   imagen[2]<= 8'b00000000; //B
```

```
                    Tmp_DATA<=imagen[MOD_3];
                end

        if (coord_x > 476 && coord_x < 484 && coord_y > 0  && coord_y < 240)
                begin
                    imagen[0]<= 8'b00000000; //R
                    imagen[1]<= 8'b00000000; //G
                    imagen[2]<= 8'b00000000; //B

                    Tmp_DATA<=imagen[MOD_3];
                end

        end
end
```

3.5 RESULTADOS OBTENIDOS

3.5.1 DESPLIEGUE DE IMÁGENES EN EL MONITOR VGA GENERADOR DE FONDO

Para enviar imágenes sencillas al monitor se diseño los contadores de pixeles horizontales y de líneas verticales para tener control sobre el monitor VGA.

Para lograr el color de fondo solo hay que modificar los parámetros que a continuación se muestran esta parte de código nos permite generar el color de fondo del monitor VGA. Para este caso se le puso un color de fondo de color verde (VERDE = "1111111111).

Si **H_Cont** está entre el rango (144,784) y **V_Cont** está entre el rango (45,525) la variable **ROJO** toma el numero binario **“00000000”**, en caso contrario toma el valor de cero. Para generar un color rojo la variable **ROJO** tiene que tomar el numero binario **“11111111”.**

Si **H_Cont** está entre el rango (144,784) y **V_Cont** está entre el rango (45,525) la variable **VERDE** toma el numero binario **“11111111”**, en caso contrario toma el valor de cero. Para generar un color rojo la variable **VERDE** tiene que tomar el numero binario **“11111111”.** Esta es la parte del código que genera el fondo verde.

Si **H_Cont** está entre el rango (144,784) y **V_Cont** está entre el rango (45,525) la variable **AZUL** toma el numero binario **“00000000”**, en caso contrario toma el valor de cero. Para generar un color azul la variable **AZUL** tiene que tomar el numero binario **“11111111”.**

Esta parte de código es la encargada de generar los fondos de pantalla para el monitor VGA, las variables **ROJO,VERDE, AZUL,** pueden tomar diferentes valores para dar origen a cualquier color las combinaciones son 256 x 256 x 256.

Parte del código para generar fondo

```
always@(H_Cont  or  V_Cont) begin

 // Generador de Fondo
if (H_Cont >= X_START && H_Cont<X_START + H_SYNC_ACT && V_Cont >= Y_START &&
V_Cont<Y_START + V_SYNC_ACT)
 ROJO = "0000000000";
 else
 ROJO = 0;

if (H_Cont >= X_START && H_Cont < X_START + H_SYNC_ACT &&V_Cont >= Y_START && V_Cont
< Y_START + V_SYNC_ACT)
VERDE = "1111111111";
else
VERDE = 0;

if (H_Cont >= X_START && H_Cont < X_START + H_SYNC_ACT && V_Cont >= Y_START && V_Cont
< Y_START + V_SYNC_ACT)
AZUL = "0000000000";
else
AZUL = 0;
```

Para lograr un rectángulo de menor dimensión sobre el fondo generado de la pantalla. Es necesario otro contador de pixeles y de líneas verticales, para este caso se generara un rectángulo de menor dimensión de color azul de una dimensión menor.

La variable **coord_x es** es un contador de pixel horizontal y **coord_y** es un contador de línea vertical.
Si **coord_x** esta entre el rango (220,420) y **coord_y** esta entre el rango (190,290) las variables **ROJO, VERDE, AZUL,** toman el numero binario **"11111111"**.

Si **coord_x** esta entre el rango (120,520) y **coord_y** esta entre el rango (210,270) las variables **ROJO, VERDE, AZUL,** toman el numero binario **"11111111"**.

Este código es para generar un rectángulo de menor dimensión en la pantalla VGA. Las variables **ROJO, VERDE, AZUL** pueden tomar el valor entre 0-256, por lo que el numero de colores que se puede desplegar sobre la pantalla es 256 x 256 x 256.

El codigo siguiente es para generar un rectángulo de menor tamaño de color blanco.

Parte del código para generar un rectángulo blanco

```
// Generador de imagenes rectangulo color blanco
  if (coord_x > 220 && coord_x < 420 &&  coord_y > 190 && coord_y < 290) begin
     ROJO = "1111111111";
     VERDE = "1111111111";
     AZUL = "1111111111";
  end
 if (coord_x > 120 && coord_x < 520 &&  coord_y > 210 && coord_y < 270) begin
     ROJO = "1111111111";
     VERDE = "1111111111";
     AZUL = "1111111111";
  end
end
```

El resultado obtenido se presenta en la figura 3.12, se puede ver el monitor con un fondo de color verde y rectángulo de menor dimensión de color blanco.

Figura 3.12 Pantalla con fondo de color verde y rectángulo de color blanco.

La figura 3.13 muestra otro color para el fondo de pantalla, en este caso el color de fondo es azul y el rectángulo es de color rojo. El procedimiento fue el mismo lo que se cambio fue solo el valor de los parámetros;

Para el fondo:

*VERDE = "1111111111"; **por** VERDE = "0000000000";*

y

*AZUL = "0000000000"; **por** AZUL = "1111111111";*

Para el rectángulo:

ROJO = "1111111111"; *ROJO = "1111111111";*
*VERDE = "1111111111"; **por** VERDE = "0000000000";*
AZUL = "1111111111"; *AZUL = "0000000000";*

Figura 3.13 Monitor VGA con fondo azul y rectángulo rojo.

3.5.2 DESPLIEGUE DE IMÁGENES EN LA PANTALLA LCD TRDB-LCM

FONDO DE COLOR VERDE PARA LA PANTALLA LCD

Lo primero que se hizo fue generar un fondo para la pantalla LCD, para este caso implemento el código para generar un fondo de color verde y el resultado se muestra en la figura 3.14.

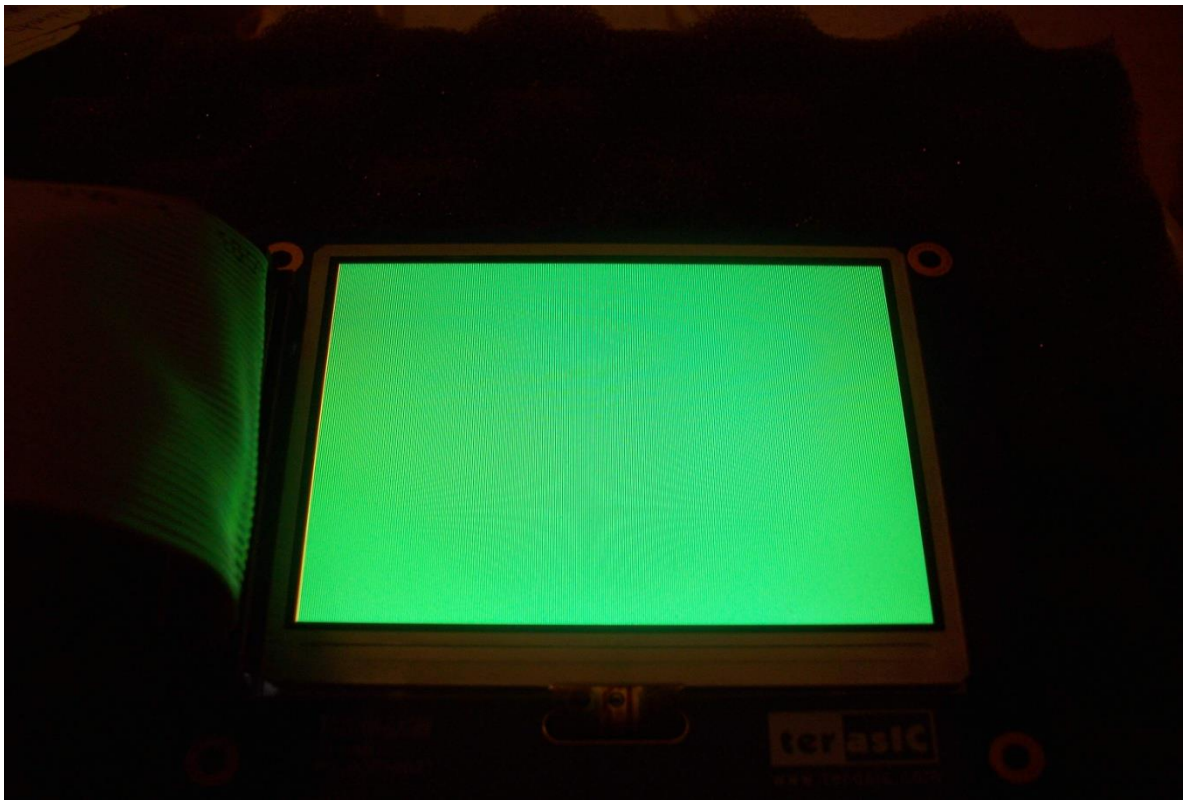


Figura 3.14 Pantalla LCD con un fondo de pantalla de color verde.

Para cambiar el fondo de pantalla solo hay que modificar los parámetros de los valores RBG que se explicaron anteriormente. Los colores que soporta esta pantalla esta en los de rangos de R [9:0] x G [9:0] x B [9:0] bits. Para tener un fondo de pantalla de color azul por ejemplo se tendrían que poner en 1 todos los valores de B y los valores de RG en ceros.

3.5.3 RESULTADO DE LA IMAGEN FINAL DESPLEGADA SOBRE LA PANTALLA

Como resultado final se obtuvo un código diseñado en Verilog HDL que fuera capaz de desplegar imágenes sencillas sobre una pantalla figura 3.15, el código completo esta en el apéndice A, ahí está el código completo para obtener las figuras plasmadas en la pantalla LCD. Para desplegar otras imágenes de diferentes tamaños y de diferente color. Solo hay que modificar los parámetros del código del apéndice A.

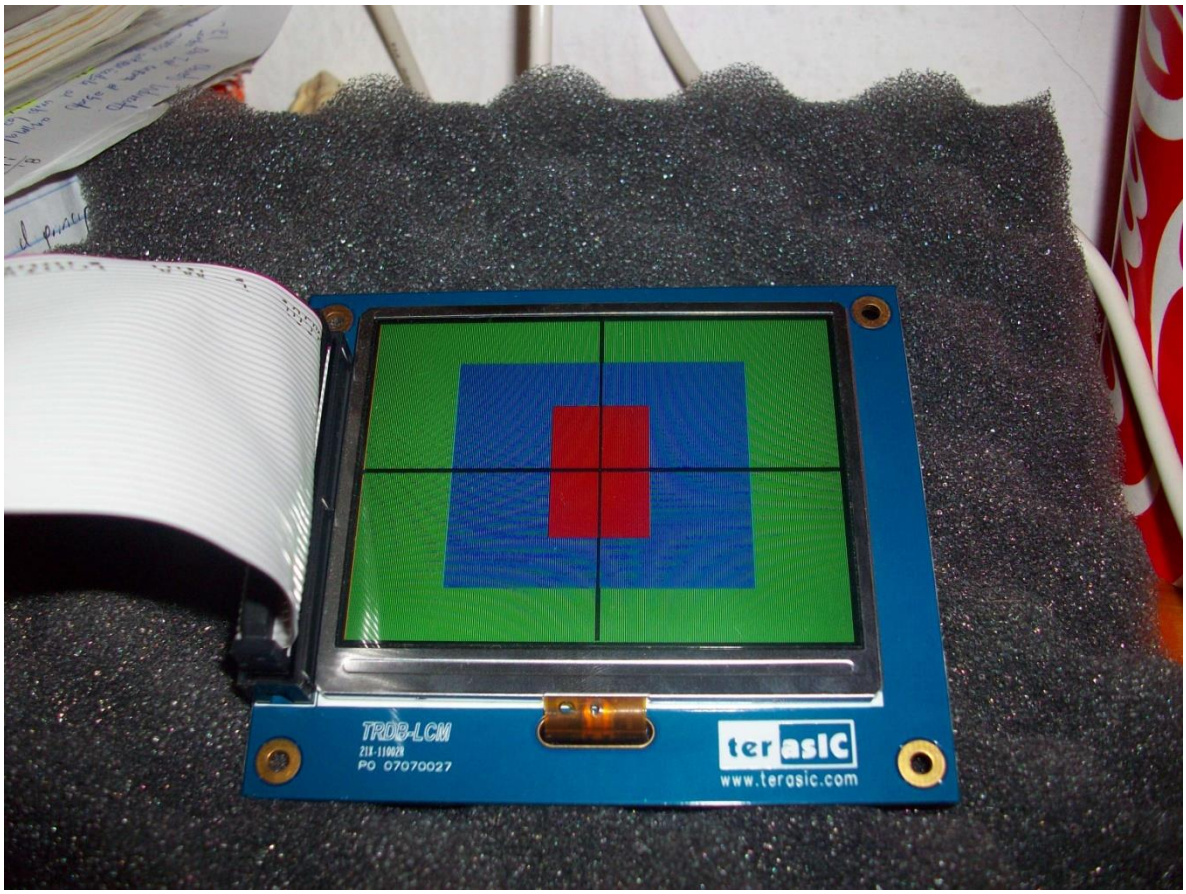


Figura 3.15 Imagen final obtenida sobre la pantalla LCD

OBSERVACIONES Y SUGERENCIAS

El diseño de sistemas digitales mediante software es un campo muy amplio en el área de Ingeniería Electrónica, con este trabajo se pretende que nuevos alumnos del Instituto Tecnológico de Tuxtla Gutiérrez se interesen en este campo de trabajo. El ITTG es una institución que está a la vanguardia y por eso que es necesario que en los laboratorios de electrónica se pueda contar con materiales para realizar prácticas y proyectos utilizando kits de trabajo, como puede ser los de la compañía Altera que están completos para realizar prácticas sencillas. Los kits de Altera están compuestos por LCDs, dispaly de 8 segmentos. LEDs, entrada de video y audio, así como conexión a internet e infrarrojo.

Otro factor importante es la bibliografía disponible en la biblioteca de la escuela, esta es muy reducida, ya que solo cuenta con tres libros de autores diferentes, seria conveniente ampliar la bibliografía que trate sobre sistemas digitales mediante software.

CONCLUCIONES

Cuándo se trabaja con el procesamiento de imágenes y de video, es necesario comprender como están compuestas las imágenes, videos, y que características hay que tomar en cuenta para su manipulación. El trabajo expuesto en este trabajo nos da las bases para la manipulación de imágenes y video utilizando un FPGA en un sistema embebido y el lenguaje Verilog HDL. La ventaja de utilizar el lenguaje Verilog y un FPGA es que nos permite trabajar varios procesos al mismo tiempo, haciendo que los procesos no sean concurrentes. Para el procesamiento de video y de imágenes es necesario determinar la frecuencia de reloj de la cámara y de la pantalla donde será desplegada la imagen, con el objeto de tener sincronía entre los dos equipos, esto nos permitirá tener una mejor visualización de la imagen. Una de las aplicaciones de este trabajo seria la comparación de imágenes tomadas por una cámara para reconocimiento de objetos, o de movimiento para luego ser desplegadas en un equipo de visualización. Para manipular imágenes o videos la técnica que se sigue es comparar pixel por pixel y línea por línea. Como resultado de este trabajo se logro el despliegue de imágenes sencillas sobre la pantalla y la programación de los tiempos de sincronía horizontal y vertical para un monitor VGA y una pantalla TRDB-LCM, que son dos equipos para el despliegue de imágenes y de video.

La continuación de este proyecto nos permitirá el tratamiento de imágenes aplicando las diferentes técnicas que hay, por ejemplo binarizacion, sobel, etc.

Por otro lado el dispositivo FPGA cuenta con mayores posibilidades frente a un GAL y un CPLD, teniendo una enorme masa de componentes internos disponibles para desarrollar diseños de sistemas digitales.

REFERENCIAS

Libros

Charles H, Roth, Jr (2005). *Fundamentos de diseño lógico* (5ª ed.). México: THOMSON

D. Gajski Daniel (1997). *Principios de diseño digital.* España, Madrid: PRENTICE HALL.

G. Martínez David. Alcalá Jessica (2003). *VHDL. El arte de programar sistemas digitales.* México: CECSA

Morris, Mano, M (2003). *Diseño digital* (3ª ed.). México: Pearson Education.

Links de interés

www.altera.com o **ltm.terasic.com** para más información del kit ALTERA DE2.

www.terasic.com para más información sobre la pantalla TRDB-LCM.

Digital Video and Image Processing
http://www.xilinx.com/esp/prof_brdcst/collateral/videoprocessing.pdf

Apéndice A

PROGRAMA EN VERILOG DE LA PANTALLA TRDBLCM

```
module LCM_clok
        (
                ////////////////////    Clock Input             ////////////////////
                CLOCK_27,                                       //      27 MHz
                CLOCK_50,                                       //      50 MHz
                EXT_CLOCK,                                      //      External Clock
                ////////////////////    Push Button             ////////////////////
                KEY,                                            //      Pushbutton[3:0]
                ////////////////////    DPDT Switch             ////////////////////
                SW,                                             //      Toggle
Switch[17:0]
                ////////////////////    GPIO    ////////////////////////////
                GPIO_0,                                         //      GPIO
Connection 0
                ////////////////////  I2C  ////////////////////////////////
                I2C_SDAT,
                I2C_SCLK,

        );



//////////////////////        Clock Input             //////////////////////
input                   CLOCK_27;                       //      27 MHz
input                   CLOCK_50;                       //      50 MHz
input                   EXT_CLOCK;                      //      External Clock
//////////////////////        Push Button             //////////////////////
input   [3:0]   KEY;                                    //      Pushbutton[3:0]
//////////////////////        DPDT Switch             //////////////////////
input   [17:0]  SW;                                     //      Toggle Switch[17:0]
//////////////////////        I2C             ////////////////////////////////
inout                   I2C_SDAT;                       //      I2C Data
output                  I2C_SCLK;                       //      I2C Clock
//////////////////////        GPIO    ////////////////////////////////
inout   [35:0]  GPIO_0;                                 //      GPIO Connection 0
```

```
assign  GPIO_0          =       36'hzzzzzzzzz;


wire    [7:0]   LCM_DATA;               //      LCM Data 8 Bits
wire            LCM_GRST;               //      LCM Global Reset
wire            LCM_SHDB;               //      LCM Sleep Mode
wire            LCM_DCLK;               //      LCM Clcok
wire            LCM_HSYNC;              //      LCM HSYNC
wire            LCM_VSYNC;              //      LCM     VSYNC
wire            LCM_SCLK;               //      LCM I2C Clock
wire            LCM_SDAT;               //      LCM I2C Data
wire            LCM_SCEN;               //      LCM I2C Enable


assign  GPIO_0[18]      =       LCM_DATA[6];
assign  GPIO_0[19]      =       LCM_DATA[7];
assign  GPIO_0[20]      =       LCM_DATA[4];
assign  GPIO_0[21]      =       LCM_DATA[5];
assign  GPIO_0[22]      =       LCM_DATA[2];
assign  GPIO_0[23]      =       LCM_DATA[3];
assign  GPIO_0[24]      =       LCM_DATA[0];
assign  GPIO_0[25]      =       LCM_DATA[1];
assign  GPIO_0[26]      =       LCM_VSYNC;
assign  GPIO_0[28]      =       LCM_SCLK;
assign  GPIO_0[29]      =       LCM_DCLK;
assign  GPIO_0[30]      =       LCM_GRST;
assign  GPIO_0[31]      =       LCM_SHDB;
assign  GPIO_0[33]      =       LCM_SCEN;
assign  GPIO_0[35]      =       LCM_HSYNC;

assign  LCM_GRST        =       KEY[0];
assign  LCM_DCLK        =       ~CLK_25;
assign  LCM_SHDB        =       1'b1;

wire                    iCLK;
wire                    iRST_N;
reg             [10:0]  H_Cont;
reg             [10:0]  V_Cont;
reg             [7:0]   Tmp_DATA;
reg                             CLK_25;
reg     oVGA_V_SYNC;
reg     oVGA_H_SYNC;
reg         [10:0]      coord_x;
reg         [10:0]      coord_y;

reg             [1:0]   MOD_3;
```

```
//memoria 3 palabras de 8 bits c/u
reg    [7:0]       imagen [0:2];

wire     [1:0]     mSEL;

assign  iCLK              =        CLK_25;
assign  iRST_N            =        KEY[0];
assign  LCM_VSYNC         =        oVGA_V_SYNC;
assign  LCM_HSYNC         =        oVGA_H_SYNC;

assign  LCM_DATA = Tmp_DATA;


//        Horizontal Parameter   ( Pixel )
parameter       H_SYNC_CYC    =        1;
parameter       H_SYNC_BACK   =        152;
parameter       H_SYNC_ACT    =        960;
parameter       H_SYNC_FRONT=          59;
parameter       H_SYNC_TOTAL=          1171;
//        Virtical Parameter             ( Line )
parameter       V_SYNC_CYC    =        1;
parameter       V_SYNC_BACK   =        13;
parameter       V_SYNC_ACT    =        240;
parameter       V_SYNC_FRONT=          8;
parameter       V_SYNC_TOTAL=          262;
//        Start Offset
parameter       X_START                =        H_SYNC_CYC+H_SYNC_BACK;
parameter       Y_START                =        V_SYNC_CYC+V_SYNC_BACK;

LCM_PLL         u0        (.inclk0(CLOCK_50),.c0(CLK_25));


//generador de color de fondo para la pantalla lcd

always@(posedge iCLK or negedge iRST_N)

begin
if(!iRST_N)
        begin
                Tmp_DATA<= 8'b0;
        end
        else
        begin
        if (coord_x >0 && coord_x < 960 && coord_y > 0 && coord_y < 240)
                begin
```

```
                imagen[0]<= 8'b11111111; //G
                imagen[1]<= 8'b00000000; //B
                imagen[2]<= 8'b00000000; //R

                Tmp_DATA<=   imagen[MOD_3];

        end

        if (coord_x >180 && coord_x < 780 && coord_y > 35 && coord_y < 205)
        begin


                imagen[0]<= 8'b00000000; //G generador de cuadro azul
                imagen[1]<= 8'b11111111; //B
                imagen[2]<= 8'b00000000; //R

                Tmp_DATA<=   imagen[MOD_3];

        end

    if (coord_x > 380 && coord_x < 580 &&coord_y > 70 && coord_y < 170)
        begin


                imagen[0]<= 8'b00000000; //G
                imagen[1]<= 8'b00000000; //B
                imagen[2]<= 8'b11111111; //R  generador de cuadro rojo

                Tmp_DATA<=   imagen[MOD_3];

        end

        if (coord_x > 0 && coord_x < 960 &&coord_y > 118 && coord_y < 122)
        begin


            imagen[0]<= 8'b00000000; //R  generador de lineas horizontales y verticales
            imagen[1]<= 8'b00000000; //G
            imagen[2]<= 8'b00000000; //B

                Tmp_DATA<=   imagen[MOD_3];

        end

    if (coord_x > 476 && coord_x < 484 &&
coord_y > 0  && coord_y < 240)
```

```
                begin


                        imagen[0]<= 8'b00000000; //R
                        imagen[1]<= 8'b00000000; //G
                        imagen[2]<= 8'b00000000; //B

                        Tmp_DATA<=   imagen[MOD_3];

                end



        end
end


always@(posedge iCLK or negedge iRST_N)
begin
if(!iRST_N)
        begin
                MOD_3<=2'b00;
        end
        else
        begin
                if(     H_Cont>H_SYNC_BACK && H_Cont<(H_SYNC_TOTAL-H_SYNC_FRONT) )
      begin
                        if(MOD_3<2'b10)
                                MOD_3<=     MOD_3+1'b1;


                                else
                                MOD_3<=     2'b00;

                  end
                else
                begin
                        MOD_3<=     2'b00;

                end
        end
end
```

```
//      H_Sync Generator, Ref. 18.42 MHz Clock
always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                H_Cont          <=      0;
                oVGA_H_SYNC <=      0;
        end
        else
        begin
                //      H_Sync Counter
                if( H_Cont < H_SYNC_TOTAL )
                H_Cont <=       H_Cont+1;
                else
                H_Cont <=       0;
                //      H_Sync Generator
                if( H_Cont < H_SYNC_CYC )
                oVGA_H_SYNC <=      0;
                else
                oVGA_H_SYNC <=      1;
        end
end

//      V_Sync Generator, Ref. H_Sync
always@(posedge iCLK or negedge iRST_N)
begin
        if(!iRST_N)
        begin
                V_Cont          <=      0;
                oVGA_V_SYNC <=      0;
        end
        else
        begin
                //      When H_Sync Re-start
                if(H_Cont==0)
                begin
                        //      V_Sync Counter
                        if( V_Cont < V_SYNC_TOTAL )
                        V_Cont <=       V_Cont+1;
                        else
                        V_Cont <=       0;
                        //      V_Sync Generator
                        if(     V_Cont < V_SYNC_CYC )
                        oVGA_V_SYNC <=      0;
                        else
                        oVGA_V_SYNC <=      1;
```

```
                end
        end
end

//contador de pixeles
always@(posedge iCLK or negedge iRST_N) begin
        if (!iRST_N) begin
                coord_x         <=      0;
                coord_y         <=      0;
 end
        else begin
                if (H_Cont >= X_START - 2 && H_Cont < X_START + H_SYNC_ACT-2 &&
                        V_Cont >= Y_START && V_Cont < Y_START + V_SYNC_ACT)
          begin
          coord_x       <=      H_Cont - (X_START - 2);
          coord_y       <=      V_Cont - Y_START;
         end
        end
end

// Configuracion de la pantalla LCD TFT
I2S_LCM_Config          u4      (       //      Host Side
                                        .iCLK(CLOCK_50),
                                        .iRST_N(KEY[0]),
                                        //      I2C Side
                                        .I2S_SCLK(LCM_SCLK),
                                        .I2S_SDAT(GPIO_0[34]),
                                        .I2S_SCEN(LCM_SCEN)  );
endmodule
```

Apéndice B

LENGUAJE VERILOG

ALGUNAS CONSIDERACIONES ACERCA DEL LENGUAJE

Antes de proseguir con la definición del diseño es importante hacer notar las siguientes consideraciones.

Comentarios. De una sola línea: pueden introducirse precedidos de //. De varias líneas pueden introducirse con el formato /* comentario */.

Uso de Mayúsculas. Verilog es sensible al uso de mayúsculas. Se recomienda el uso únicamente de minúsculas.

Identificadores. Los identificadores en Verilog deben comenzar con un carácter, pueden contener cualquier letra de la **a** a la **z**, caracteres numéricos además de los símbolos "_" y "$". El tamaño máximo es de 1024 caracteres.

NÚMEROS EN VERILOG

Las constantes numéricas en Verilog pueden especificarse en decimal, hexadecimal, octal o binario. Los números negativos se representan en complemento a 2 y el carácter "_" puede utilizarse para una representación más clara del número, si bien no se interpreta. La sintaxis para la representación de una constante numérica es:

<Tamaño>'<base><valor>

Si bien el lenguaje permite el uso de números esteros y reales, por brevedad y simplicidad sólo utilizaremos la representación de constantes numéricas enteras.

Tabla B.1 manejo de números en verilog.

Entero	Almacenado como	Descripción
1	00000000000000000000000000000001	Unzised 32 bits
8'hAA	10101010	Sized hex
6'b10_0011	100011	Sized binary
'hF	00000000000000000000000000001111	Unzised hex 32 bits
6'hCA	001010	Valor truncado
6'hA	001010	Relleno de 0's a la izquierda
16'bz	zzzzzzzzzzzzzzzz	Relleno de z's a la izquierda
8'bx	xxxxxxxx	Relleno de x's a la izquierda
8'b1	00000001	Relleno de 0's a la izquierda

IDENTIFICADORES Y VARIABLES

Un identificador está formado por una letra o "_" seguido de letras, números y los caracteres "$" o "_". Se distingue entre mayúsculas y minúsculas.

VARIABLES

Existen 2 tipos fundamentales de variables:

- **Reg** es un registro y permite almacenar un valor
- **Wire** es una red que permite la conexión

Por defecto dichas variables son de un único bit, pero pueden declararse variables con un mayor número de bits:

Tipo [msb: lsb] nombre_variable;

Ejemplo:

- **reg [5:0]** C**;** // Es un registro de 6 bits, siendo C[0] el bit menos significativo.

También es posible también definir memorias, así por ejemplo:

- **reg [7:0]** mem **[0:1023];** // Es la declaración de una memoria de 1024 bytes.

Además de estos tipos existen otros tipos de datos más abstractos como:

- **Integer** (registro de 32 bits).
- **Real** (registro capaz de almacenar un número en coma flotante).
- **Time** (registro unsigned de 64 bits).

EXPRESIONES BOOLEANAS

Las expresiones booleanas se especifican en Verilog con un enunciado de asignación continuo que consiste en la palabra clave **assign** seguid de una expresión booleana. Para distinguir el mas aritmético del OR, Verilog HDL usa los símbolos (&), (|) y (~) para AND, OR, NOT (complemento).

Ejemplo B.1 muestra la descripción de un circuito que se especifica con estas dos expresiones booleanas;

x= A+BC+B'D

y= B'C+BC'D'

EJEMPLO B.1

```
//circuito especificado con expresiones booleanas
module circuito_bln (x,y,A,B,C,D);
input  A,B,C,D;
output x,y;
assign    x= A | (B&C) | (~B&D);
assign    y= (~B&C) | (B&~C&~D);
endmodule
```

PROCESOS

El concepto de procesos que se ejecutan en paralelo es una de las características fundamentales del lenguaje, siendo ese uno de los aspectos diferenciales con respecto al lenguaje procedural como el lenguaje C.

Toda descripción de comportamiento en lenguaje Verilog debe declararse dentro de un proceso, aunque existe una excepción que trataremos a lo largo de este apartado. Existen en Verilog dos tipos de procesos, también denominados bloques concurrentes.

Initial. Este tipo de proceso se ejecuta una sola vez comenzando su ejecución en tiempo cero. Este proceso NO ES SINTETIZABLE, es decir no se puede utilizar en una descripción RTL. Su uso está íntimamente ligado a la realización del testbench.

Always. Este tipo de proceso se ejecuta continuamente a modo de bucle. Tal y como su nombre indica, se ejecuta siempre. Este proceso es totalmente sintetizable. La ejecución de este proceso está controlada por una temporización (es decir, se ejecuta cada determinado tiempo) o por eventos. En este último caso, si el bloque se ejecuta por más de un evento, al conjunto de eventos se denomina lista sensible. La sintaxis de este proceso es pues:

- **Always <temporización> o <@ (lista sensible)>**

Se muestra muestra dos ejemplos de utilización de los procesos initial y always. De estos ejemplos se pueden apuntar las siguientes anotaciones:

```
initial
begin
clk = 0;
reset = 0;
enable = 0;
data = 0;
end
```

```
always @(a or b or sel)
begin //Sobra en este caso
if (sel == 1)
y = a;
else
y = b;
end //Sobra en este caso
```

Begin, end. Si el proceso engloba más de una asignación procedural (=) o más de una estructura de control (if-else, case, for, etc.), estas deben estar contenidas en un bloque delimitado por begin y end.

Initial. Se ejecuta a partir del instante cero y, en el ejemplo, en tiempo 0 (no hay elementos de retardo ni eventos, ya los trataremos), si bien las asignaciones contenidas entre begin y end se ejecutan de forma secuencial comenzando por la primera. En caso de existir varios bloques initial todos ellos se ejecutan de forma concurrente a partir del instante inicial.

Always. En el ejemplo, se ejecuta cada vez que se produzcan los eventos variación de la variable a o variación de b o variación de sel (estos tres eventos conforman su lista de sensibilidad) y en tiempo 0. En el ejemplo, el proceso always sólo contiene una estructura de control por lo que los delimitadores begin y end pueden suprimirse.

Todas las asignaciones que se realizan dentro de un proceso initial o always se deben de realizar sobre variables tipo reg y NUNCA sobre nodos tipo wire. Se muestra un ejemplo de asignación errónea sobre nodos tipo wire en un proceso initial.

A	**B**
wire clk, reset;	**reg** clk, reset;
reg enable, data;	**reg** enable, data;
initial	**initial**
begin	**begin**
clk = 0; //Error	clk = 0;
reset = 0; //Error	reset = 0;
enable = 0;	enable = 0;
data = 0;	data = 0;
end	**end**

A) Error de asignación de valores a variables tipo wire. B) Código corregido.

En general, la asignación dentro de un proceso initial o always tiene la siguiente sintaxis:

- **Variable = f (wire, reg, constante numérica)**

La variable puede ser tanto una variable interna como una señal del interfaz del módulo. La asignación puede ser de un nodo tipo wire, de una variable tipo reg, de una constante numérica o una función de todas ellas.

ASIGNACIÓN CONTINUA

La asignación continua se utiliza exclusivamente para modelar lógica combinacional. A diferencia de la asignación procedural se ejecuta de forma continua por lo que no necesita de una lista sensible. La sintaxis de este tipo de asignación es:

- **assign** variable #<delay> = f (**wire, reg, constante numérica**)

La variable sólo puede estar declarada tipo net (en nuestro caso tipo wire). La asignación continua debe estar declarada fuera de cualquier proceso y nuca dentro de bloques always o bloques initial. Se presenta una serie de ejemplos de asignaciones continuas.

```
//El siguiente ejemplo corresponde a un buffer triestado
wire [7:0] out;
……
assign #1 out = (enable)? data : 8'bz
//Asignación de una suma de operandos de 4bits a una concatenación de suma y carry de
//salida
reg [3:0] a,b;
wire cout;
wire [3:0] sum;
……
assign #1 {cout,sum} = a + b cin
```

TEMPORIZACIONES

Las temporizaciones se consiguen mediante el uso de retardos. El retardo se especifica mediante el símbolo # seguido de las unidades de tiempo de retardo. La Figura B.1 muestra el efecto del operador retardo. En ella se observa que los retardos especificados son acumulativos.

```
initial
begin
clk = 0;
reset = 0;
#5 reset = 1;
#4 clk = 1;
Reset = 0;
end
```

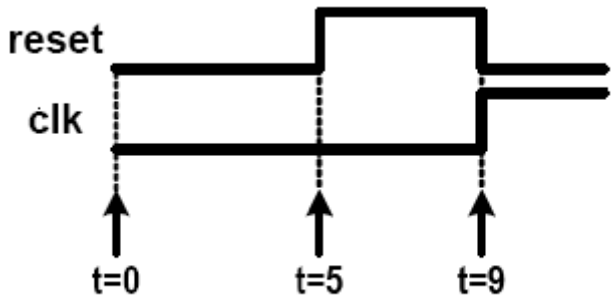


Figura B.1 Uso de temporizaciones en un bloque initial.

En el ejemplo anterior, las dos primeras asignaciones se ejecutan en tiempo 0. Cinco unidades de tiempo más tarde se realizan la tercera asignación y cuatro unidades más tarde de esta última se ejecutan la cuarta y quinta asignación. La Figura B.2 muestra un ejemplo de control de asignación en un proceso always.

```
always
#5clk= !clk;
```

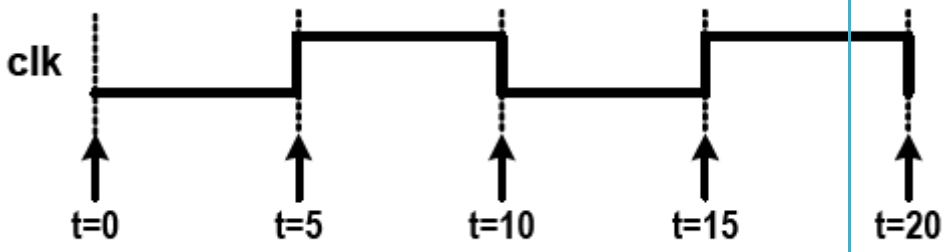


Figura B.2 Uso de temporizaciones en un bloque always.

EVENTOS

La segunda manera de controlar el instante en que se produce una asignación procedural o la ejecución de un proceso always es por el cambio de una variable, denominándose control por eventos. Para ello se emplea el carácter @ seguido del evento.

Se distinguen dos tipos de eventos:

Eventos de nivel. Este evento se produce por el cambio de valor de una variable simple o de una lista sensible. Veamos los siguientes ejemplos:

Evento	Descripción
always @ (a) b = b+c;	Cada vez que varía a se evalúa la expresión
always @(a or b or c) d = a+b;	Cada vez que varía a o b o c se evalúa la expresión. En este caso, a, b y c conforman la lista sensible.

Eventos de flanco. Este evento se produce por la combinación de flanco/s de subida y/o bajada de una variable simple o de una lista sensible. Veamos los siguientes ejemplos:

Evento	Descripción
always @(posedge clk or posedge mr) b <= b+c;	Cada vez que se produce un flanco de subida de clk o de mr se evalúa la expresión
always @(posedge clk or negedge mr) b <= b+c;	Cada vez que se produce un flanco de subida de clk o de bajada de mr se evalúa la expresión

MÓDULOS Y JERARQUÍAS

Tal y como se comentó anteriormente, el identificador **module** se utiliza para la definición de módulos/diseños. Estos módulos pueden utilizarse para construir diseños de mayor complejidad, creando lo que se denomina un diseño con jerarquía. La manera de incluir un módulo en un diseño es:

- **master_name instante_name (port_list);**

La figura B.3 muestra un ejemplo de un módulo, denominado muxt, formado por dos módulos, denominados mux, y cuya declaración de interfaz es la que a continuación se muestra:

- Module mux(a, b, sl, y); //a, b, out son señales de 8 bits

```
module
muxt(in0,in1,sel,out);
input [7:0] in0,in1;
input sel;
output [7:0] out;
wire [7:0] out;
 // nodos internas
wire [7:0] y0;
//Conexionado por orden
mux mux0(in0,in1,sel,y0);
//Conexionado por nombre
mux  mux1(.y(out),  .b(in1),
.a(y0), .sl(sel));
endmodule
```

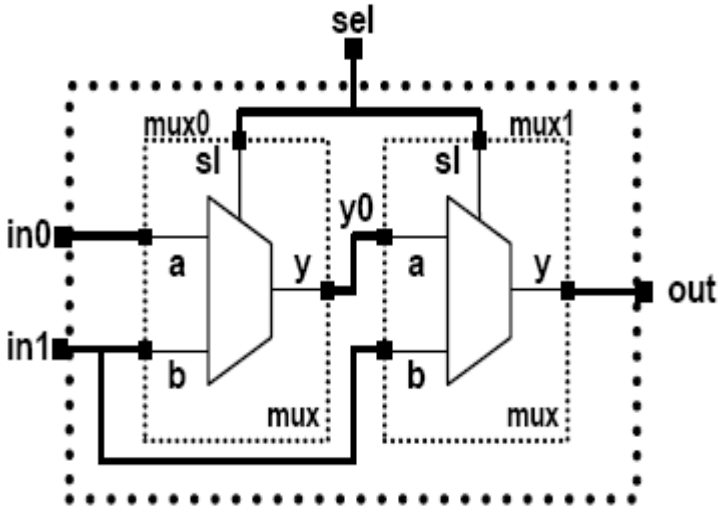


Figura B.3 Llamada a un módulo.

OPERADORES EN VERILOG

Los operadores aritméticos y/o lógicos se utilizan para construir expresiones. Estas expresiones se realizan sobre uno o más operandos. Estos últimos pueden ser nodos, registros constantes, etc.

Operadores aritméticos

De un operando:

- "+" a = +8'h01 Resultado: a = 8'h01
- "-" a = -8'h01 Resultado: a = 8'hFF

De dos operandos:

- "+" a = b + c;
- "-" a = b – c;
- "*" a = b * c;
- "/" a = b / c; Resultado: Se devuelve la parte entera de la división
- "%*" a = b % c; Resultado: Se devuelve el módulo de la división

El resultado del módulo toma el signo del primer operando. Si algún bit del operando tiene el valor "x", el resultado completo es "x". Los números negativos se representan en complemento a 2.

Operadores relacionales

- "<" a < b a es menor que b
- ">" a > b a es mayor que b
- "<=" a <= b a es menor o igual que b
- ">=" a >= b a es mayor o igual que b

El resultado que se devuelve es:

- 0 si la condición no se cumple.
- 1 si la condición se cumple.
- x si alguno de los operandos tiene algún bit a "x".

Operadores de igualdad

- "==" a == b a es igual a b.
- "!=" a!= b a es diferente de b.
- "===" a === b a es igual a b, incluyendo "x" y "z".
- "!==" a !== b a es diferente a b, incluyendo "x" y "z".

Los operandos se comparan bit a bit, rellenando con ceros para ajustar el tamaño en caso de que no sean de igual ancho. Estas expresiones se utilizan en condicionales. El resultado es:

- 0 si se cumple la igualdad
- 1 si no se cumple la igualdad
- x únicamente en las igualdades "==" o "!=" si algún operando contiene algún bit a "x" o "z"

Operadores lógicos

- "!" negación lógica
- "&&" AND lógica
- "||" OR lógica

Las expresiones con "&&" o "||" se evalúan de izquierda a derecha. Estas expresiones se utilizan en condicionales. El resultado es:

- 0 si la relación es falsa
- 1 si la relación es verdadera
- x si algún operando contiene algún bit a "x" o "z"

Operadores bit a bit (bit-wise)

- "~" negación
- "& " AND
- "|" OR
- "^" OR exclusiva

Los citados operadores pueden combinarse obteniendo el resto de operaciones, tales como ~& (AND negada), ~^ (OR exclusiva negada), etc. El cálculo incluye los bits desconocidos (x) en la siguiente manera:

- ~x = x
- 0&x = 0
- 1&x = x&x = x
- 1|x = 1
- 0|x = x|x = x
- 0^x = 1^x = x^x = x

Operadores de reducción

- "&" AND Ejemplo: &a Devuelve la AND de todos los bits de a
- "~&" AND negada Igual que el anterior
- "|" OR Ejemplo: |a Devuelve la OR de todos los bits de a
- "~|" OR negada Igual que el anterior
- "^" OR exclusiva Ejemplo: ^a Devuelve la OR exclusiva de todos los bits de a
- "~^" OR exclusiva negada Igual que el anterior

Estos operadores de reducción realizan las operaciones bit a bit sobre un solo operando. Las operaciones negadas operan como AND, OR o EXOR pero con el resultado negado.

Operadores de desplazamiento

- "<<" Desplazamiento a izquierda Ejemplo: a = b << 2;
- ">>" Desplazamiento a derecha Ejemplo: a = b >> 2;

Notas: Los bits desplazados se rellenan con ceros.

Operadores de concatenacion

- "{}" Concatenación de operandos. Los operandos se separan por comas

Ejemplos: {a, b [3:0], c} Si a y c son de 8 bits, el resultado es de 20 bits

{3{a}} Es equivalente a {a, a, a}

{b,3{c,d}} Es equivalente a {b, c, d, c, d, c, d}

Nota: No se permite la concatenación con constantes sin tamaño.

Operador condicional

- "?" El formato de uso de dicho operador es (condición) ? trae_expr : false_expr

Ejemplos: out = (enable)? a:b; La salida out toma el valor de a si enable está a 1, en caso contrario toma el valor de b

Precedencia de los operadores

El orden de precedencia de los diferentes operadores es el siguiente:

Tabla B.2 ordenación de precedencia de los operadores.

Operador	**Símbolo**
Unary, Multiplicación, División, Módulo	+,-,*,%
Suma, Resta, Desplazamientos	+,-,<<,>>
Relación, Igualdad	<,>,<=,>=,==,!=,===,!===
Reducción	&,!&,^,~^,|,~|
Lógicos	&&,||
Condicional	?:

ESTRUCTURAS DE CONTROL

El lenguaje dispone de una elevada cantidad de estructuras de control, similares a las disponibles en otros lenguajes de programación.

IF-ELSE

If (expresión) command1;
Else command2;

Si expresión se evalúa como cierto(1) se ejecuta command1 en caso contrario se ejecuta command2.

La sentencia condicional **if – else** controla la ejecución de otras sentencias y/o asignaciones (estas últimas siempre procedurales). El uso de múltiples sentencias o asignaciones requiere el uso de begin – end. La sintaxis de la expresión es:

La sentencia condicional **if – else**

```
if (condición) //Condición simple sentencias;
if (condición) //Condición doble
sentencias;
else
sentencias;
if (condición1) // Múltiples condiciones
sentencias;
else if (condición2)

 sentencias
 ;
 ....
 else
 sentencias
 ;
```

```
//Condicional simple
if (enable)
q<= #1 data;
//Condicional multiple
if (reset == 1'b1)
count<= #1 8'b0;
else if (enable == 1'b1 && up_en == 1'b1)
count<= #1 count + 1'b1;
else if (enable == 1'b1 && down_en == 1'b1)
count<= #1 count – 1'b1;
else
count<= #1 count;
```

CASE

Case (expresión)
val1: command1;
val2: command2;
...
Default: commandN;
Endcase

Se evalúa la expresión, en caso de resultar val1 se ejecuta command1, si resulta val2 se ejecuta command2. En caso de no estar reflejado el resultado se ejecuta commandN.

La sentencia case evalúa una expresión y en función de su valor ejecuta la sentencia o grupos de sentencias agrupadas en el primer caso en que coincida. El uso de múltiples sentencias o asignaciones requiere el uso de begin – end. En caso de no cubrir todos los posibles valores de la expresión a avaluar, es necesario el uso de un caso por defecto (default). Este caso se ejecutará siempre y cuando no se cumplan ninguno de los casos anteriores. La sintaxis de la expresión es:

```
case (expresión)
<caso1>: sentencias;
<caso2>: begin
sentencias;
sentencias;
end
<caso3>: sentencias;
....
....
<default>: sentencias;
endcase
```

En le siguiente ejemplo hay que destacar que no se están cubriendo todos los casos ya que no se evalúan los posibles valores de la variable sel en estado de alta impedancia (z) o desconocido (x). La sentencia case reconoce tanto el valor z como x como valores lógicos.

```
//Ejemplo
case (sel)
2'b00: y = ina;
2'b01: y = inb;
2'b10: y = inc;
2'b11: y = ind;
   default: $display("Valor de sel erróneo);
endcase
```

FOR

- **For** (init; condición1; rep) command;

Inicialmente se ejecuta el comando init, ejecutándose a continuación command mientras que la condición condicion1 resulte un valor cierto (1), al final de cada ejecución se ejecuta rep.

WHILE

- **While** (cond) command;

Mientras que la condición cond sea cierta (1), se ejecuta el comando command.

REPEAT

- **Repeat** (Ntimes) command;

Repite el comando command tantas veces como indique Ntimes

WAIT

- **Wait** (cond) command;

Mientras que la condición cond sea falsa (0), se ejecuta el comando command.
A menudo se emplea para detener la ejecución secuencial del proceso hasta que se verifique una condición.

Apéndice C

MANEJO DEL KIT ALTERA DE2 Y EL SOFTWARE QUARTUS II

En el CD que viene dentro de la caja de la DE2 rotulado como "DE2 System", se debe buscar un archivo en Excel denominado "DE2 pin Assignments". Este archivo indica cuales son las terminales del Cyclone II que están directamente conectaras a los elementos enlistados anteriormente.

Seleccionar: Assignments>>Pins, aparecerá una nueva pantalla dividida en 3 partes, en una de ellas (izquierda) aparecerá un listado de las Entradas/salidas del sumador-restador, en el lado derecho aparecerá una vista del circuito Cyclone II. En la parte inferior de esa ventana aparecerá una lista de cada una de las terminales de Entrada/Salida (bit por bit).
De esa última tabla, en la tercera columna aparece la leyenda "Location", en ese espacio se puede ir colocando la asignación para cada una de las terminales conforme a la tabla en Excel. Una vez finalizada la asignación de terminales de E/S se deberá compilar de nuevo el dispositivo. Para verificar que las conexiones se hayan realizado correctamente se pueden hacer 2 cosas:

a) Leer atentamente los mensaje de la consola

b) En la pantalla del lado izquierdo del "compilation Report" seleccionar la carpeta "Fitter" y la sub-carpeta "Pin-Out File", aparecerá un archivo texto en donde se especifican las asignaciones de Entradas y Salidas.

Configuración del dispositivo:
La DE2 esta pre-configurada con un código de muestra que permite comprobar el funcionamiento de todos los componentes de la tarjeta. Para encender la tarjeta deberán seguirse los siguientes pasos:

a) Conectar el cable USB a la computadora y al conector USB Blaster de la tarjeta. Para poder usar esta interface de configuración bajo el estándar JTAG se deberá instalar el dispositivo en la computadora siguiendo el "asistente de Hardware nuevo" de Windows para instalar el driver del USB Blaster. Si este "driver" no está instalado en la computadora, deberán seguirse los pasos del tutorial "Getting Started with Altera's DE2 Board", el cual está dentro del CD de la tarjeta o siguiendo los pasos del "Asistente de HW nuevo" de Windows una vez que se encienda la tarjeta y detecte el dispositivo, el "driver" también está en el CD, o bien se puede obtener de la página de ALTERA.

b) Conectar el adaptador de 9 Volts a la tarjeta DE2

c) Verificar que el Switch RUN/PROG se encuentre en la posición de RUN

d) En QUARTUS II, dentro del diseño desarrollado, seleccionar el menú:
TOOLS>>Programmer (o con el botón de acceso rápido)

e) Aparecerá la pantalla del programador, en caso de que esta página no contenga ningún archivo, se deberá apretar el botón "Add File" y agregar el archivo con extensión ".sof" perteneciente a este proyecto.

f) Una vez agregado el archivo, se deberá verificar apretar el botón "Hardware SetUp", se abrirá una nueva pantalla indicando que no hay un Hardware de programación activado, para añadir el USB Blaster dar doble click sobre la lista en blanco de "Available Hardware Items", aparecerá una nueva venta llamada "Add Hardware", busque el dispositivo en la lista de "Hardware type", seleccione el dispositivo y de click en OK.

g) Cierre la pantalla del "Hardware SetUp". Una vez sobre la pantalla del programador, active la casilla de la columna "program/configure", posteriormente apriete el botón: Start.