UNIVERSIDAD DE SONORA

DIVISIÓN DE CIENCIAS EXACTAS Y NATURALES

DEPARTAMENTO DE INVESTIGACIÓN EN FÍSICA

INGENIERÍA EN TECNOLOGÍA ELECTRÓNICA

DISEÑO Y FABRICACIÓN DE UNA INTERFAZ ELECTRÓNICA PARA UNA PANTALLA DE OLEDS.

TESIS

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN TECNOLOGÍA ELECTRÓNICA

PRESENTA:

ÁVILA AVENDAÑO JESÚS ALBERTO

VALDEZ ALMIRUDIS CARLOS DOMINGO

DIRECTOR DE TESIS:

DR. LUIS ALFREDO GONZÁLEZ LOPEZ

DR. BENITO RAFAEL NORIEGA LUNA

Hermosillo, Sonora, México Junio del 2008

Universidad de Sonora

Repositorio Institucional UNISON

Agradecimientos…

… a mi abuela Inés y a mi abuelo Clemente, que la vida no me dio la oportunidad de agradecerles en persona, por el gran cariño y amor que siempre me demostraron, y su interés siempre por el bienestar de los demás.

A mis papás que siempre me han demostrado su apoyo, por ser un gran ejemplo para mí, por siempre estar ahí cuando los necesito, por dejarme la mejor herencia que puedo pedir, la educación para la vida, los quiero mucho, y gracias a ustedes, estoy terminando esta etapa de mi vida.

A mis hermanos, que siempre me han apoyado, por su confianza, por ayudarme a crecer, Nadiehzda y Carlos Hugo, se los agradeceré por siempre.

A toda mi familia, por compartir muy buenos momentos conmigo, por estar al pendiente siempre de mí, muchas gracias.

A mi novia Fabiola, por ser un gran apoyo para mí, por comprenderme en los momentos difíciles que se dieron durante este proyecto, por ser también mi mejor amiga, gracias.

A mis asesores de tesis, Dr. Luís González y Dr. Benito Noriega, por la disposición y su tiempo dedicado, por su invaluable ayuda para terminar este proyecto.

A mis maestros, por la paciencia que me tuvieron durante el curso en la universidad.

A CONACYT por su apoyo brindado para la realización de este proyecto, por la beca por el proyecto 4407-A “Investigación y Desarrollo de sistemas Óptico Digitales Para Procesamiento de información Tridimensional”

A todos mis compañeros que han estado conmigo, me han ayudado a salir adelante y han ofrecido su amistad.

Jesús Alberto Ávila Avendaño

Agradecimientos…

A mi familia que me apoyó en esta etapa de mi vida y la cual me llena de alegría haber concluido satisfactoriamente. A mi padre por apoyarme en los momentos difíciles que viví al terminar mis estudios, a mi madre por su paciencia, a mi hermanos que son mis eternos compañeros.

A mis maestros que compartieron sus conocimientos conmigo, así como también por esos buenos y no tan buenos momentos que me hicieron pasar.

A mis asesores de tesis, Dr. Luís González y Dr. Benito Noriega, por su gran apoyo en este proyecto.

A CONACYT por su apoyo brindándome una la beca por el proyecto 4407-A “Investigación y Desarrollo de sistemas Óptico Digitales Para Procesamiento de información Tridimensional”

A mis compañeros y amigos que tuve el honor de conocer durante mis años como universitario, que me apoyaron en mi desarrollo profesional, así como también a mis demás amistades que estuvieron conmigo durante estos años y que ellos mejor que nadie han vivido mi desarrollo personal.

Carlos Domingo Valdez Almirudis

ÍNDICE

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUCCIÓN

En la actualidad, la electrónica mueve al mundo, la vemos en todas partes, en las computadoras, teléfonos celulares, automóviles, televisores, refrigeradores, y en una gran cantidad de aparatos que hacen nuestra vida más sencilla o más placentera, en todos ellos está presente la electrónica.

La electrónica es tan común hoy en día, que la mayoría de las personas no percibe que está ahí, además, la tecnología avanza tan rápido, que a mucha gente no le sorprenden los nuevos avances que la electrónica produce. Sin embargo, el desarrollo tan acelerado que ha tenido la electrónica en los últimos años, aunado a la necesidad de fabricar componentes cada vez más pequeños, ha provocado que se esté llegando al límite donde la electrónica que conocemos ya no es funcional.

La electrónica actual se basa en semiconductores inorgánicos, y es gracias a las propiedades de estos materiales que la electrónica puede existir. Pero hoy en día, los componentes son tan pequeños, que los semiconductores inorgánicos empiezan a dejar de funcionar a esta escala. Debido a esto, los investigadores están en la búsqueda de nuevos semiconductores que reemplacen los que son utilizados hoy en día, y una opción muy viable parecen ser, los semiconductores orgánicos.

Muchos centros de investigación alrededor del mundo están desarrollando componentes electrónicos basados en semiconductores orgánicos, uno de ellos es el grupo de investigación en ciencias de los materiales, en la Universidad de Texas en Dallas (UTD).

Entre los diferentes componentes electrónicos que son desarrollados en UTD, se encuentran los OLEDs (Diodos orgánicos emisores de luz), una de las aplicaciones posibles para este componente es el despliegue de imágenes. Es

por esto, que UTD requiere el desarrollo de un sistema electrónico que le permita a los OLEDs funcionar como una pantalla.

El objetivo de esta tesis es desarrollar un sistema que reciba imágenes desde la tarjeta de video de una computadora, en formato VGA de 640x480 pixeles a una frecuencia de 60Hz, y las despliegue en una pantalla de 64x48 OLEDs.

Para realizar esta tarea se diseñó un proceso que está compuesto por tres pasos principalmente: la conversión de los datos analógicos, que provienen desde la computadora, a datos digitales mediante el convertidor analógico digital ADC08100; posteriormente la conversión del formato de video VGA a un formato más adecuado para la pantalla de OLEDs, esta manipulación de datos es realizada por el FPGA Spartan 2E; y por último, el despliegue de las imágenes en la pantalla formada por una matriz de 64x48 OLEDs.

1.1 ORGANIZACIÓN DEL DOCUMENTO

El presente trabajo de tesis esta constituido por seis capítulos. El primer capítulo presenta una introducción al panorama general que lleva a la realización de esta tesis, así como su objetivo. En el capítulo dos se describe la conversión analógica digital, así como la interfaz que se desarrolló basado en este concepto. El capítulo tres describe el módulo de desarrollo D2SB que se utilizó para este proyecto, incluyendo el FPGA que contiene, así como la manera en que éste, cambia el formato de video VGA al formato diseñado para la pantalla de OLEDs. Para el capítulo cuatro se decidió escribir acerca de la pantalla que fue fabricada para pruebas del sistema, así como los módulos que se desarrollaron para adecuar la señal proveniente del FPGA. En el capítulo cinco se muestran los resultados de las pruebas que fueron hechos al sistema que se diseñó. Por último, en el capítulo seis se presentan las conclusiones que se obtuvieron al finalizar el diseño y realizar las pruebas.

CAPÍTULO 2

MÓDULO DE CONVERSIÓN ANALÓGICO DIGITAL

2.1 INTRODUCCIÓN

Una de las etapas fundamentales para el desarrollo del sistema propuesto es la conversión analógica digital. Para la realización de esta etapa se utilizan convertidores analógico digitales, ADC por sus siglas en inglés (Analog Digital Converter). Para explicar la función de estos dispositivos, es necesario abordar el tema desde su raíz, explicando cuál es la diferencia entre el mundo analógico y el digital.

El término analógico lo podemos definir, en pocas palabras, como todas las medidas o magnitudes que varían de manera continua, por ejemplo, la temperatura que se mide con un termómetro de mercurio, cuando sube de 34°C a 35°C. La medición de este termómetro pasa por todos los puntos que existen en ese rango, es decir, pasa por una cantidad infinita de valores de temperatura, aunque por nuestras limitaciones visuales o por las limitaciones del dispositivo, solo podamos percibir quizá décimas o centésimas de grados centígrados.

Por otro lado tenemos el término digital, como su nombre lo indica, proviene de dígito, lo cual entendemos como referente a nuestros dedos. Este término se usó porque con los dedos de las manos solo podemos contar cantidades exactas y discretas. Al contrario de las medidas analógicas, las medidas digitales varían en cantidades que tienen un número finito de decimales. Dicho de otra manera, las cantidades digitales van dando brincos para pasar de una medida a otra, y los brincos que les toma en llegar de un punto a otro, se definen como cantidades enteras y contables. Por ejemplo, un reloj digital en un instante determinado estará marcando 22 segundos, y un segundo después dará un brinco y estará marcando 23 segundos, y así sucesivamente con cambios discretos de un segundo.

De lo expuesto anteriormente podemos deducir que la electrónica analógica, es la que se encarga de los dispositivos donde el voltaje y la corriente varían de manera continua, pero los datos analógicos son muy difíciles de almacenar, manipular, comparar, calcular y recuperarlos con exactitud cuando éstos han sido guardados. Por otro lado la electrónica digital se encarga de los dispositivos donde el voltaje y la corriente se toman en valores discretos, y se tiene la información de manera digital. La información que se procesa con la electrónica digital puede almacenarse, manipular, transmitir y realizar muchas tareas de manera digital a una gran velocidad y en grandes cantidades. Este es el caso de las computadoras digitales que actualmente empleamos. Sin embargo, quizá nos preguntamos, ¿por qué si la tecnología digital presenta tantas ventajas sobre la tecnología analógica, no sustituimos todo lo analógico por lo digital? La respuesta es muy simple, el mundo es analógico, todo lo que está en la naturaleza es analógico, por lo tanto, siempre tendremos que convivir con la tecnología analógica.

De esta manera podemos decir que la conversión analógico digital se realiza cuando se transforman señales eléctricas continuas de información en datos digitales. Esta conversión es ejecutada mediante un ADC; cuyo funcionamiento de describe a continuación.

2.2 CONVERTIDOR ANALÓGICO DIGITAL

En referencia a la sección anterior entendemos que la señal de entrada de un convertidor es una señal continua (voltaje) que es convertida en un número binario de *n* cantidad de bits (valores discretos de voltaje) para poder manipularse fácilmente por un dispositivo digital, en nuestro caso, un FPGA [1].

La manera en que opera un ADC es mediante la definición de un voltaje de referencia; es decir, entre que valores de voltaje debe variar la señal de entrada de información a convertir. Para realizar esto, se define con señales de voltaje directo el límite superior y el límite inferior del voltaje de referencia. Estos límites nos indican que, si la señal de entrada está en el límite inferior, los bits de la señal

de salida estarán todos en cero; por el contrario, si la señal de entrada está en el límite superior, los bits de la señal de salida, en este caso, estarán todos en uno. Por otro lado, si la señal de entrada se encuentra en un valor entre el límite superior y el límite inferior, el valor de la salida será un cociente entre el valor del voltaje de entrada y el voltaje de referencia [1] [3]. Para ilustrar esto, consideremos el límite inferior y el límite superior de nuestro voltaje de referencia en 1V y 3V respectivamente. Si nuestra salida digital, es de 3 bits, la relación entrada – salida sería como la que se muestra en la gráfica de la Figura 2.1 [2]; en donde la línea punteada representa el voltaje analógico de entrada. Si este voltaje de entrada empieza en 1V, la salida de nuestro convertidor es 000, cuando está en 1.25V la salida que corresponde es 001, y así sucesivamente. Cada vez que la entrada aumenta .25V, la salida digital aumenta en 1 su valor. Mientras la entrada analógica, aumenta de 1 a 1.25V, puede obtenerse un error de medición, ya que no existe un valor digital que represente estos valores intermedios, a este error se le da el nombre de **error de cuantización**. Lo mismo sucede cuando la señal de entrada aumenta de 1.25 a 1.5V, mientras la señal atraviesa estos puntos continuos, la salida digital tiene un valor que puede ser 001 ó 010. El error de cuantización no puede eliminarse, pero se reduce empleando una cantidad mayor de bits para la salida digital [1] [2].

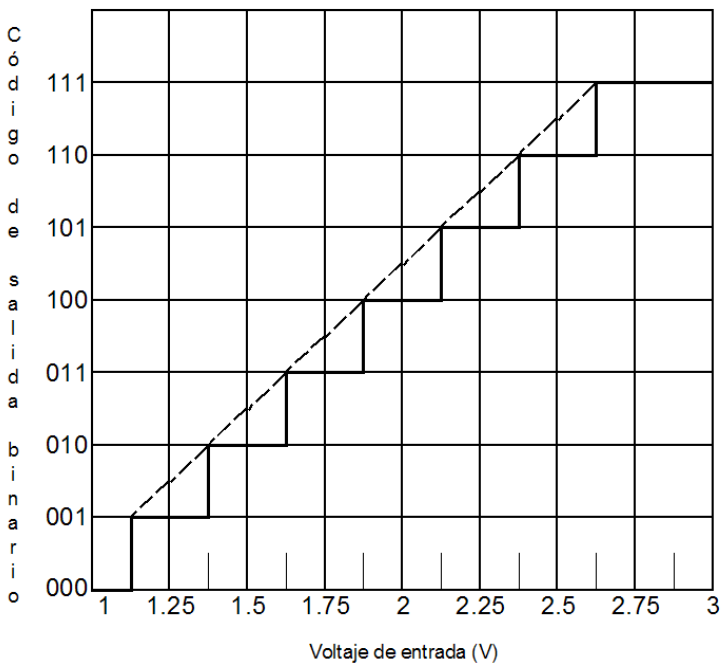


Figura 2.1 Ejemplo de relación entrada – salida de un convertidor analógico digital.

Para definir los saltos o escalones digitales, varios convertidores emplean un comparador analógico en su esquema básico, donde el voltaje de entrada del convertidor se conecta a una de las entradas del comparador, y en la otra entrada del comparador se conecta un voltaje de referencia dependiente del tiempo. A la salida del comparador, si el voltaje de entrada es mayor al voltaje de referencia, se obtiene un estado alto, es decir, un uno lógico; por el contrario, si el voltaje de entrada del comparador es menor al voltaje de entrada, la salida del comparador, será un estado bajo, o un cero lógico [1].

Después de la conversión, se obtiene un valor binario como ya mencionamos. Idealmente se espera que en los convertidores el punto de transición del dato de salida, es decir, el punto donde cambia la salida de un estado a otro, sea lo más cercano al punto medio de los valores analógicos que tienen una representación digital. Es decir, en el ejemplo que ya mencionamos, los valores analógicos de 1.25V y 1.5V son representados digitalmente por 001 y 010, respectivamente. Sin embargo para que el valor de la salida cambie de 001 a 010, lo ideal es que el punto de cambio sea el punto medio entre 1.25V y 1.5V, el cual sería, 1.375V. La diferencia fundamental entre las operaciones de diferentes convertidores es la estrategia que se usa para variar la señal del voltaje de referencia y determinar el conjunto de bits que componen la salida digital. Algunas de estas estrategias o técnicas básicas de Conversión Analógica Digital se describen con la arquitectura de los siguientes convertidores.

2.2.1 Convertidor de conteo continuo

El convertidor de conteo, funciona comparando una señal de un contador que aumenta consecutivamente desde 0 hasta su cuenta máxima, por ejemplo, si es de 3 bits, se contaría: 000, 001, 010, 011, 100, 101, 110 y 111, estos valores se convierten a señales analógicas con un convertidor digital analógico (DAC – Digital Analog Converter) y se comparan con el voltaje de entrada que queremos convertir. En el momento en que el valor del contador sea mayor que el del voltaje de entrada, se toma ese valor como conversión [1].

La principal desventaja de este método es su baja velocidad, ya que el tiempo que le toma hacer la conversión es proporcional al valor del dato

analógico. Con este procedimiento, un valor pequeño es rápidamente alcanzado por el contador, mientras que un dato analógico grande es alcanzado por el contador en un tiempo grande. Además el dato digital de entrega, es el dato inmediatamente mayor al dato que queremos convertir, pero no el más próximo [1]. Si retomamos el ejemplo anterior para ilustrar esta desventaja nos damos cuenta que, si el dato que queremos convertir es 1.51V, el ADC lo convertirá a 011, cuyo valor es referente a 1.75V, el cual está mucho más alejado de 1.5V.

Entre las ventajas, podemos mencionar que el desarrollo de este convertidor requiere muy poco hardware como se muestra en la Figura 2.2 [1], donde se puede observar que un contador, un convertidor digital analógico y un comparador son suficientes para fabricar un convertidor analógico digital de este tipo. Debido a esto, se convierte en un convertidor muy económico y suficientemente efectivo para aplicaciones de baja velocidad de conversión.

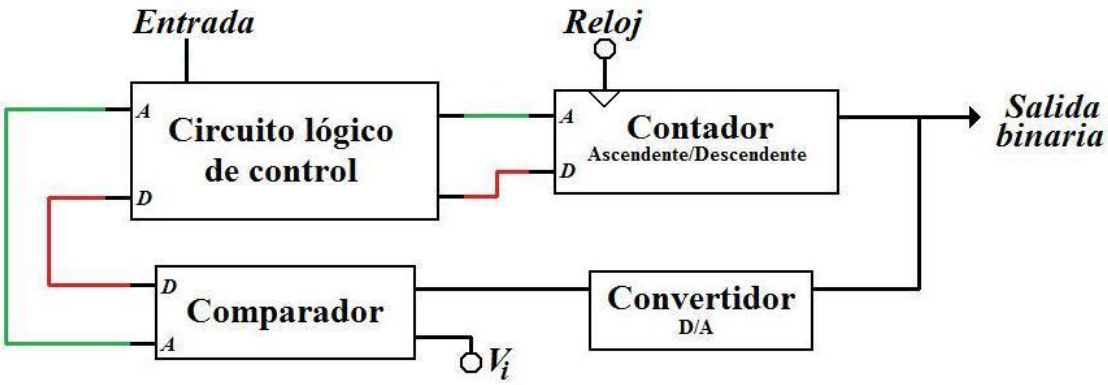


Figura 2.2 Arquitectura de un ADC de conteo continuo.

2.2.2 Convertidor de aproximaciones sucesivas

Este es otro procedimiento sencillo para convertir señales analógicas a digitales, en donde los convertidores tienen un circuito lógico para ir comparando bit por bit, desde el más significativo, al menos significativo. Para explicar esto, un convertidor de 3 bits primero compara 100, si el voltaje de entrada es mayor, entonces se decide que el bit más significativo es 1, y se compara el siguiente valor. Una vez que se ha decidido cual es el valor del bit más significativo, el siguiente valor a comparar es 110. Si el voltaje de entrada fue menor en la primera comparación, entonces el bit más significativo es 0. Así, el siguiente valor

a comparar es 010. Suponiendo que en la primera comparación, el voltaje de entrada fue mayor, entonces el siguiente valor en comparar es 110. Si al comparar este valor, el voltaje de entrada es mayor se procede a comparar 111, pero si el voltaje de entrada es menor, se compara 101. Esto se realiza sucesivamente, bit por bit, variando desde el más significativo al menos significativo. Por lo tanto, en cada ciclo de reloj se decide un bit del valor de conversión. Consecuentemente, el tiempo de conversión depende solamente de la cantidad de bits del convertidor. Este convertidor es muy popular por su velocidad en convertidores de 8 a 16 bits. Sin embargo, uno de los factores que limita su velocidad es el tiempo que le toma al comparador y al DAC en estabilizarse [4].

En la Figura 2.3 [4] podemos ver con más detalle la lógica que sigue este tipo de convertidor para aproximarse al valor a convertir. Quizá esta técnica tiene más sentido si lo explicamos en sistema decimal, ya que es lo que hacemos las personas para aproximarnos a un valor, por ejemplo, si queremos adivinar un número entre 0 y 63, preguntamos por 32, si es mayor, preguntamos por 48, y si es menor preguntamos por 16, y así sucesivamente vamos preguntando por los valores que están en medio del rango de valores posibles para obtener el valor más cercano de una manera rápida.

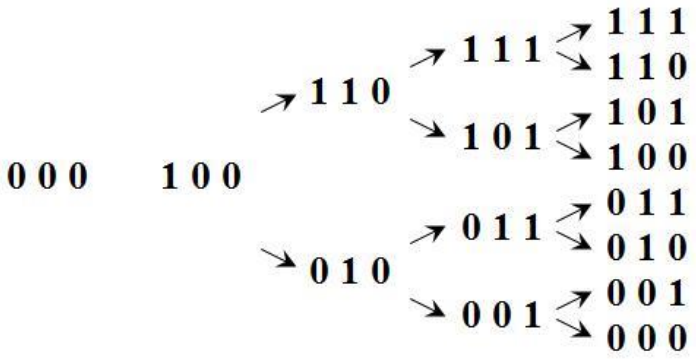


Fig. 2.3 Diagrama de la lógica que sigue el ADC de aproximaciones sucesivas.

Para complementar este tipo de convertidores, en la Figura 2.4 [2] se muestra la arquitectura básica para fabricar este tipo de convertidores. Donde se puede observar que con un comparador, un convertidor digital analógico y un circuito que desarrolla la lógica de aproximación sucesiva son suficientes para

diseñar este tipo de convertidores. Por su arquitectura, es utilizado cuando se requieren velocidades de conversión entre medias y altas del orden de algunos microsegundos o décimas de microsegundos.

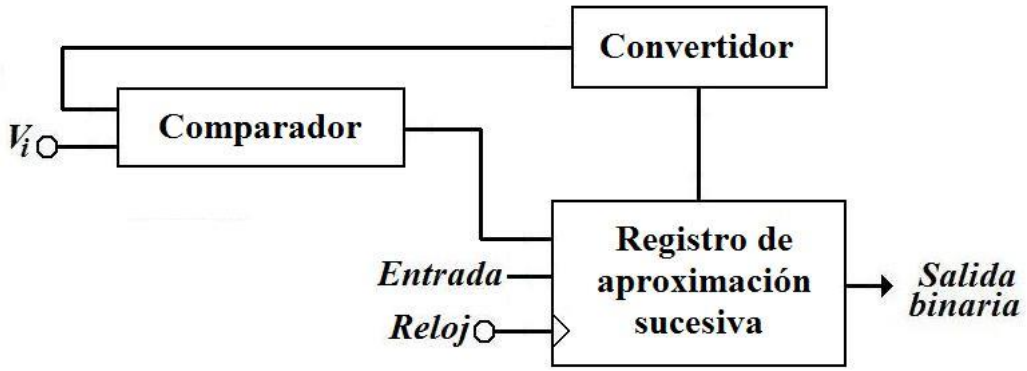


Figura 2.4 Arquitectura de un ADC de aproximaciones sucesivas.

Además de los convertidores antes mencionados existen actualmente otros en el mercado, como el convertidor de rampa (pendiente), el convertidor de doble rampa (doble pendiente), los convertidores delta sigma (que son convertidores de alta velocidad); entre otros. La decisión de cual convertidor se debe emplear dependerá del proyecto que se desee desarrollar. Para el desarrollo de nuestro proyecto hemos seleccionado un convertidor tipo flash debido a que sus características son más adecuadas al proyecto que aquí desarrollamos, principalmente su alta velocidad y bajo consumo. A continuación presentamos una breve descripción del proceso de conversión de estos convertidores.

2.2.3 Convertidor en paralelo (flash)

Este tipo de convertidor puede utilizar una variedad de esquemas de hardware bastante amplio, y su velocidad se debe a que realiza la conversión en paralelo, en lugar de hacerlo en serie como lo hacen comúnmente lo otros convertidores. Por su velocidad, también es llamado convertidor rápido (flash). En la Figura 2.5 [3] se muestra un convertidor en paralelo de 3 bits, en el cual, el voltaje de entrada se compara con siete voltajes de referencia distintos, y a su vez, las salidas de los comparadores entran a un circuito lógico que codifica directamente las señales a tres bits binarios que representan la conversión del voltaje de entrada. La velocidad de estos convertidores es muy alta, ya que solo

es limitada por el retardo de los comparadores y el circuito lógico. Además, la salida en estos convertidores es continúa pero retrasada respecto a la entrada por los comparadores y el circuito lógico [1] [3].

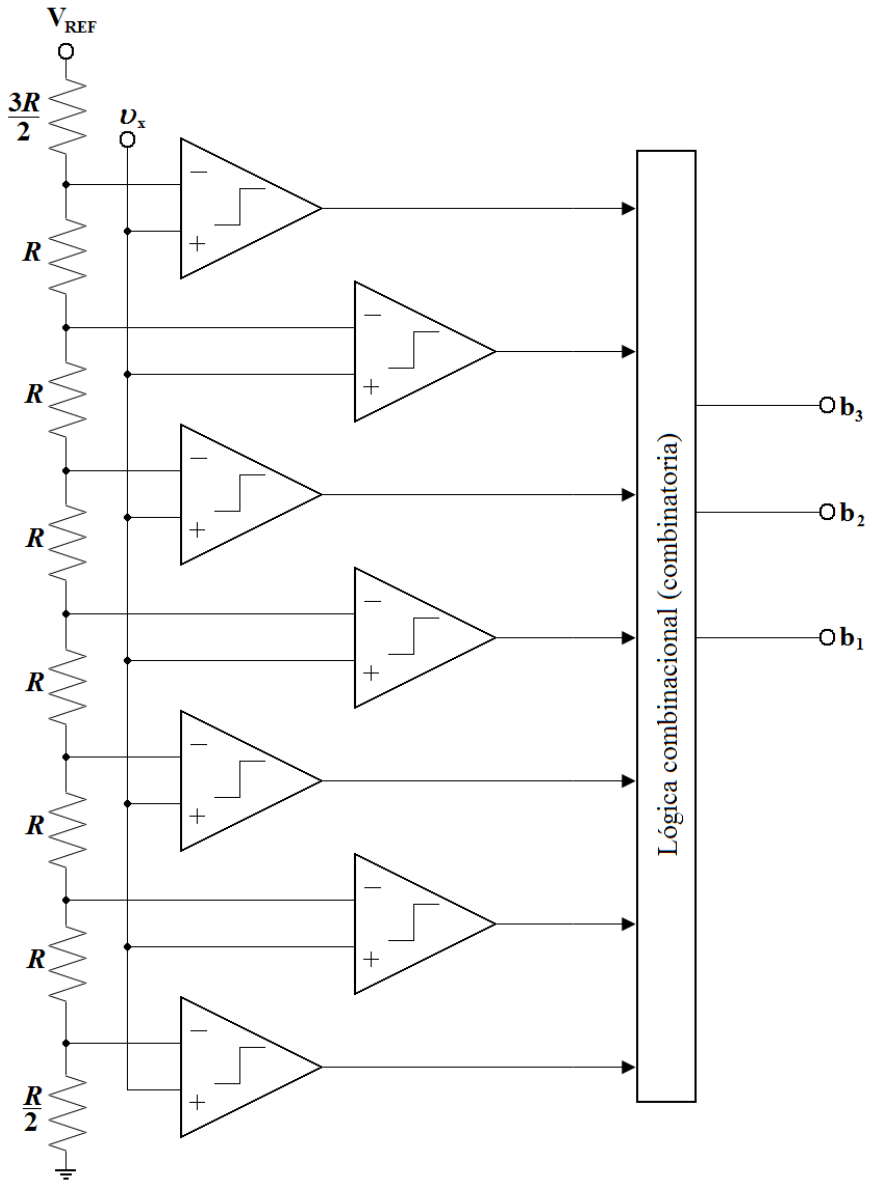


Figura 2.5 Arquitectura de un ADC en paralelo (flash).

El convertidor en paralelo se usa comúnmente en sistemas para procesar datos o señales donde se requiere alta velocidad, y no necesariamente mucha resolución. A estos convertidores se les puede encontrar hasta de 10 bits. De acuerdo al esquema de la Figura 2.5, la cantidad de resistencias y comparadores que se requieren son proporcionales a la resolución del convertidor, lo cual hace que el costo de implementar un convertidor de este tipo aumente

considerablemente por cada bit de resolución agregada. Considerando que el número de comparadores y resistencias requeridas para implementar un convertidor de n bits es de 2^n-1 para los comparadores y 2^n para las resistencias; por ejemplo, un convertidor de 10 bits ocuparía 1024 resistencias y 1023 comparadores [1]. Pero como ya mencionamos, su baja resolución se recompensa en su velocidad, ya que con tecnología de CI monolíticos se pueden alcanzar velocidades de conversión efectivas desde 100MHz hasta 1GHz [1].

2.2.4 Circuitos de muestreo y retención

Para finalizar este tema, no puede quedar fuera una parte muy importante de los ADC, los circuitos de muestreo y retención, los cuales son empleados por todos los convertidores, independientemente del tipo de arquitectura que posean. El principal objetivo de implementar estos circuitos es evitar que la señal de entrada cambie mientras se efectúa la conversión. La forma más sencilla de hacerlo es usar un interruptor y un capacitor. El interruptor cierra el circuito brevemente para cargar el capacitor con el voltaje de entrada. Una vez que el capacitor se ha cargado, el interruptor abre el circuito para que el voltaje almacenado en el capacitor no varíe. Si la señal de entrada llegara a variar, mientras se efectúa el proceso de conversión de la muestra que se almacenó en el capacitor, el interruptor vuelve a cerrar el circuito para cargar el capacitor con una nueva muestra, y así sucesivamente.

Existen otros circuitos de muestreo y retención más complejos, pero la base sigue siendo el capacitor y el interruptor, en otros circuitos se usan amplificadores operacionales (OpAmps) en su configuración de seguidores de voltaje que operan como buffers. En otros circuitos aun más complejos se utilizan lazos de retroalimentación con OpAmps para mejorar el desempeño de los circuitos [1].

Para darnos una mejor idea de cómo se realiza esta parte fundamental de la conversión, en la Figura 2.6 [1] se muestra una gráfica que ilustra el proceso de muestreo y retención en función del tiempo, definiendo los diferentes procesos que se realizan para efectuar esta función.

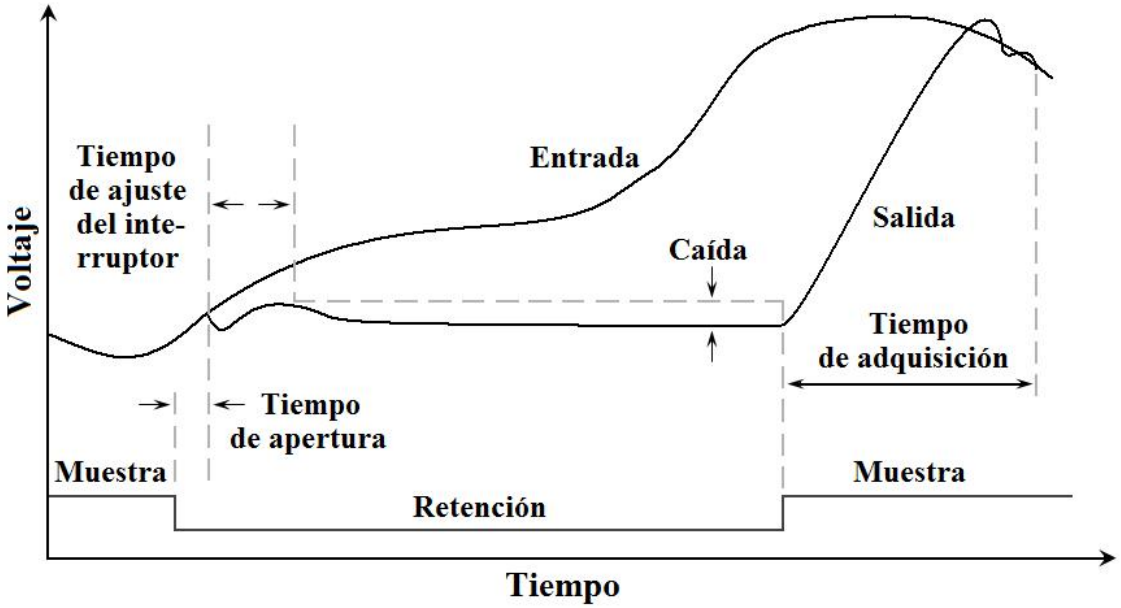


Figura 2.6 Proceso de muestreo y retención.

En la gráfica, el tiempo de apertura representa el tiempo que necesitan los dispositivos de conmutación para cambiar de estado entre los modos de muestreo y retención. En este caso se requiere un tiempo de estabilización para que los circuitos de retroalimentación se recuperen de los periodos de conmutación. Durante el modo de retención, el voltaje almacenado en el capacitor puede cambiar un poco debido a las corrientes en el módulo del interruptor y la polarización del amplificador operacional. Este cambio se muestra en la gráfica como una ligera caída de voltaje. Finalmente, se requiere un tiempo de adquisición para que el circuito recupere el voltaje de entrada después que el circuito cambia del modo de retención al modo de muestreo [1].

2.3 DESARROLLO DEL MÓDULO DE CONVERSIÓN ANALÓGICO DIGITAL

El primer paso para elaborar nuestro módulo de conversión analógica digital consiste en definir que señales vamos a recibir o convertir, que señales vamos a entregar y como se van a manipular con la interfaz. De acuerdo con nuestro propósito, la entrada será una señal VGA proveniente de la tarjeta de video de una computadora y la salida serán señales digitales que irán a un dispositivo programable para su procesamiento. La tarjeta de video de la

computadora nos provee su señal analógica mediante un conector de 15 patillas como el que se ilustra en la Figura 2.7 [14]. Como el propósito de la interfaz es manejar una pantalla de OLEDs monocromática tomaremos la señal analógica de video de color rojo (aunque también pueden ser el verde o azul ya que en todos los casos se tendrán los mismos datos) como señal de entrada de datos a nuestro ADC. Mientras que las señales VSync y HSync pasarán directamente al dispositivo programable, ya que son señales digitales. Como la pantalla de OLEDs es de baja resolución emplearemos una configuración de baja resolución, que en este caso es producir 60 cuadros de 640x480 pixeles por segundo (señal de video de 60Hz). Este es un estándar internacional, donde la frecuencia de píxel o reloj que debe ser empleada es de 25MHz (ver especificaciones en Apéndice A). La magnitud de la señal de datos varía entre 200mV y 1.5V. Una vez defina la señal característica a convertir, el siguiente paso es elegir el convertidor ADC apropiado para lograr nuestro propósito.

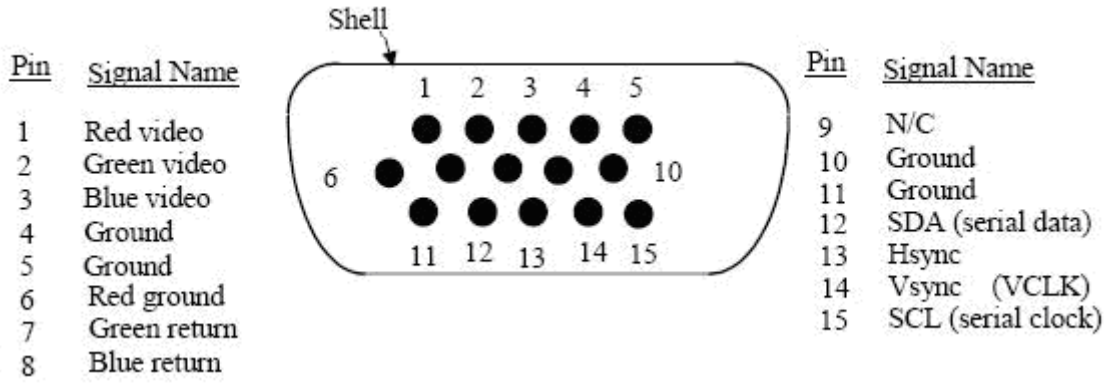


Figura 2.7 Conector VGA (Disposición de patillas)

2.3.1 Convertidor ADC08100

Para realizar la conversión analógico digital utilizamos el circuito ADC08100, que es un convertidor monolítico de 8 bits de bajo consumo y de alta velocidad. Este dispositivo trabaja a velocidades desde 20 MSPS hasta 100 MSPS. Para ilustrar la disposición de patillas y señales de este dispositivo consideremos la Figura 2.8 [5], en donde las patillas V_A son de alimentación analógica, las patillas AGND son la tierra analógica, las patillas V_{RT}, V_{RM} y V_{RB} son patillas para especificar el voltaje de referencia para la conversión, la patilla V_{IN} es la entrada de la señal analógica, las patillas D7 a D0 son los bits de salida

digitales, del más significativo al menos significativo; respectivamente. La patilla DRV_D es la patilla de alimentación digital y su correspondiente tierra digital se encuentra en la patilla DRGND. Por último, la patilla CLK es la patilla de entrada del reloj de muestreo.

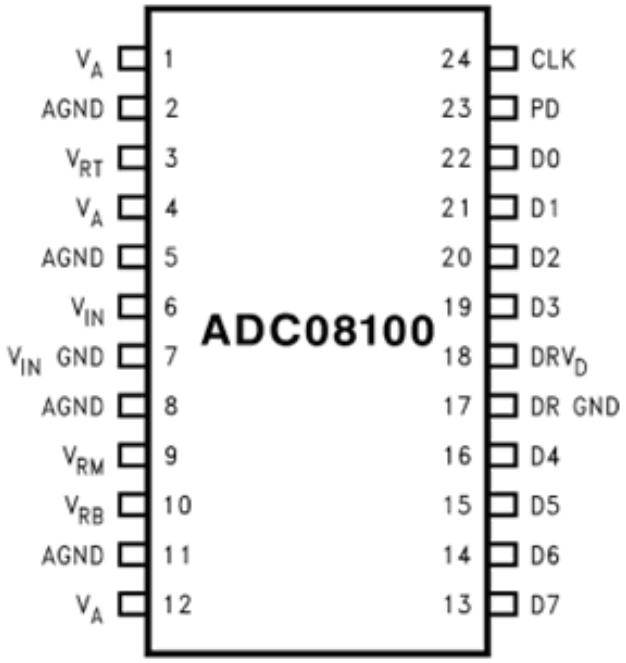


Figura 2.8 Convertidor ADC08100 (Disposición de patillas).

Una descripción detallada de las características de estas señales se encuentra en su hoja de especificaciones [5]. Adicionalmente, en ese documento podemos encontrar algunas recomendaciones de aplicaciones de este circuito integrado con sus respectivos esquemáticos. Como en nuestro proyecto no se requiere una alta resolución de video, optamos por elegir una configuración simple cuyos requerimientos son suficientes para tratar las señales a convertir. El diagrama esquemático de tal configuración se presenta en la Figura 2.9 [5].

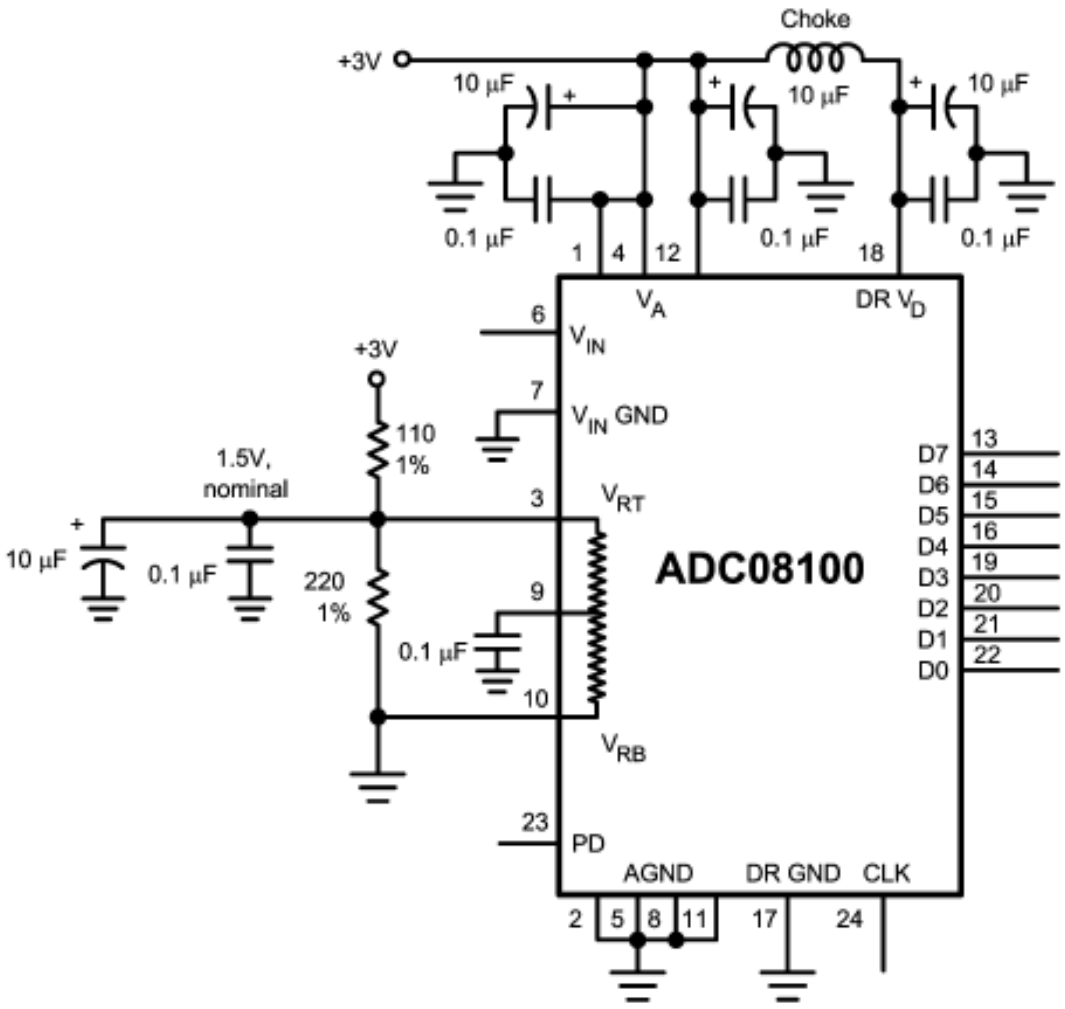


Figura 2.9 Diagrama esquemático del circuito como será conectado el convertidor ADC08100.

Como características principales de este circuito, primeramente vemos una bobina Choke, la cual es usada para desacoplar la fuente de alimentación analógica, de la fuente de alimentación digital. Esto se realiza para dar más estabilidad a la fuente digital, evitando que el ruido de la parte analógica lo afecte. Obsérvese la cantidad de capacitores que se tiene, que como podemos ver, están conectados a las patillas de la fuente y del voltaje de referencia. Con este arreglo de capacitores se mantendrá un voltaje lo más rectificado posible eliminando pequeños ruidos que pudieran ser agregados por la fuente o las línea de transmisión. Por último, vemos que las resistencias que proporcionan el voltaje de referencia son de baja tolerancia, y esto se debe a la misma estabilidad que debe de tener el voltaje de referencia, ya que entre menos varíe este voltaje, la señal de salida será más exacta o repetible. El convertidor es capaz de funcionar a una

frecuencia de reloj de 100MHz, pero los datos en este formato VGA son enviados desde la computadora a una frecuencia de 25MHz, por lo tanto, la señal de reloj que utilizará el convertidor será enviada desde el FPGA a una frecuencia de 25MHz.

Para el desarrollo de este módulo ahora es necesario unir las partes que constituyen el sistema de conversión para desarrollar una tarjeta de circuito impreso que incluya el conector VGA, el convertidor ADC con todos los componentes que se requieren, y para la salida, un conector de 40 patillas, el cual se conectará a un dispositivo programable. Un diagrama esquemático de los componentes se muestra en la Figura 2.10.

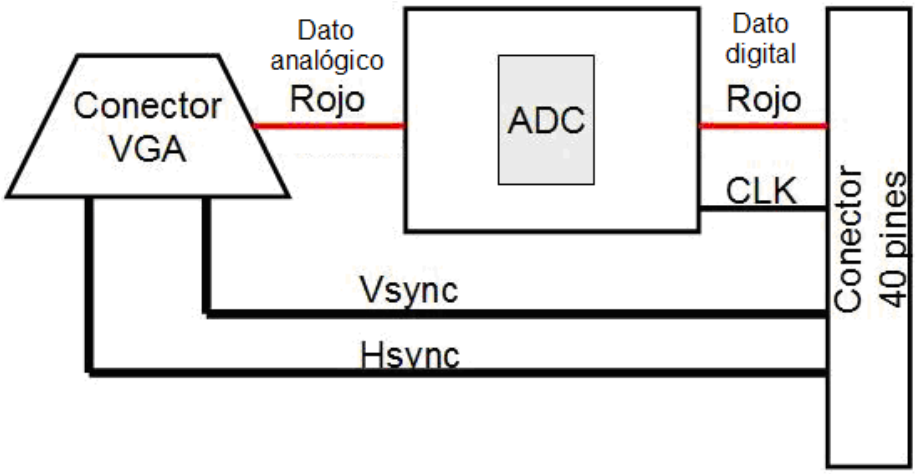


Figura 2.10 Diagrama del módulo completo de conversión analógica digital.

2.3.2 Diseño de tarjeta de circuito impreso

En la hoja de datos del convertidor ADC08100 [5], también encontramos recomendaciones acerca del diseño de la tarjeta de circuito impreso, las cuales es importante seguir para optimizar el desempeño del convertidor. Como el circuito resultante manejará señales de alta velocidad, existen muchos factores que afectan el funcionamiento del convertidor que se minimizan si se siguen las recomendaciones apropiadas. Para ello consideramos las siguientes recomendaciones:

- Se deben separar la parte analógica de la parte digital, incluyendo el camino de retorno a tierra analógica, debe ser diferente al camino de

retorno a tierra digital, para evitar que el ruido analógico afecte la parte digital del circuito.

- Otra recomendación consiste en que los capacitores de desacoplo de fuente se pongan lo más cerca posible a las patillas de alimentación del circuito integrado. De esta manera se minimiza el ruido agregado por las líneas de transmisión [5].
- Por último, al trazar las pistas de los bits de salida, debemos tomar en cuenta que los datos digitales provenientes del convertidor salen en forma paralela, y a la velocidad que trabajan estos circuitos, una pista más larga que otras, puede ocasionar un retraso de bits. Para evitar este problema, las líneas de transmisión deben tener la misma longitud.

Para el diseño de la tarjeta de circuito impreso usamos el programa OrCad Layout, comenzando por diseñar los footprints de los diferentes componentes. El footprint de un componente es la imagen del área que ocupará en la tarjeta impresa, incluyendo las perforaciones de patillas si así lo requiere el componente. El programa OrCad Layout contiene una librería donde ya se encuentran los footprints de resistencias, capacitores, conectores, circuitos integrados, etc. por lo tanto, solo fue necesario diseñar los footprints de la bobina de Choke, del conector de 40 patillas y del convertidor ADC.

Los footprints de todos los componentes que usaremos se agregan a un área de trabajo donde se desarrolla el diseño de la tarjeta y se especifican las conexiones entre los componentes. Para simplificar las dimensiones del circuito, en nuestro caso, realizamos un diseño de dos capas. El diseño resultante de ambas caras se muestra en el Apéndice C. Para la fabricación de la tarjeta impresa empleamos una máquina de impresos LPKF que se encuentra en las instalaciones del laboratorio de Mecanismos y circuitos impresos de la Universidad de Sonora. La imagen del circuito resultante con sus componentes se muestra en la Figura 2.11.

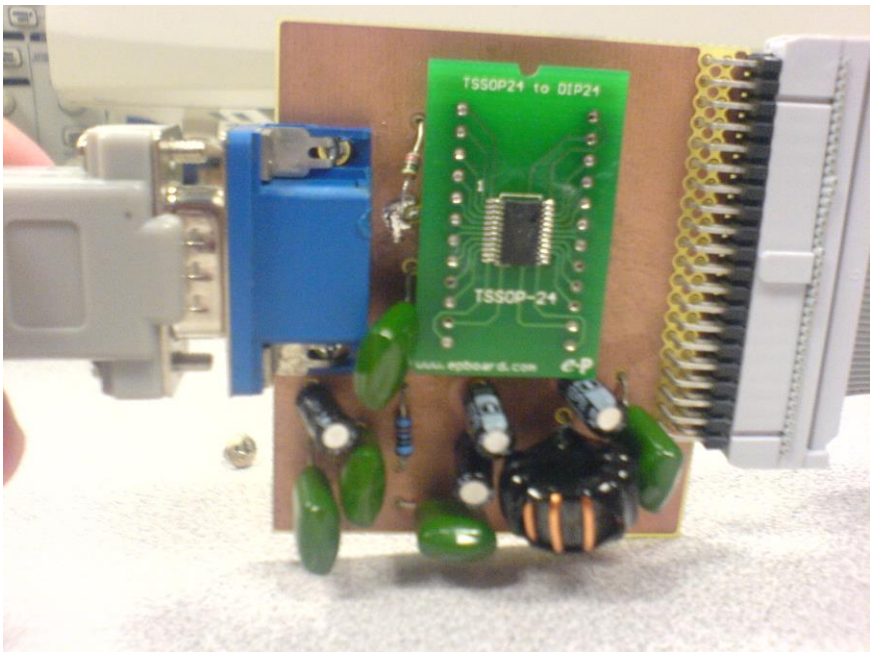


Figura 2.11 Módulo de conversión analógica digital terminado.

2.4 RESULTADOS

El procedimiento para probar el módulo de conversión analógica digital fue muy sencillo, primeramente se configuró una computadora con una señal de video de 640X480 a 60Hz que fuera enviada mediante el conector VGA de nuestro módulo. Para la señal de reloj externa, usamos un tren de pulsos de 25MHz con una amplitud de 3.3V a partir de un generador de funciones, ya que la señal de reloj que enviará el dispositivo programable será una señal similar. Para comprobar el resultado, analizamos con un osciloscopio la señal de entrada analógica y la señal de salida digital del bit más significativo, ya que en el proyecto final, será el único que usaremos para lograr nuestro propósito. De esta prueba se espera que cuando la señal de entrada sea mayor que un nivel de voltaje de 1V, la señal de salida esté en un nivel alto. Si la señal de entrada es menor a 1V, la señal de salida se encuentre en un nivel bajo. Es decir, lo que obtendremos es una imagen binaria. El diagrama de prueba se muestra en la Figura 2.12:

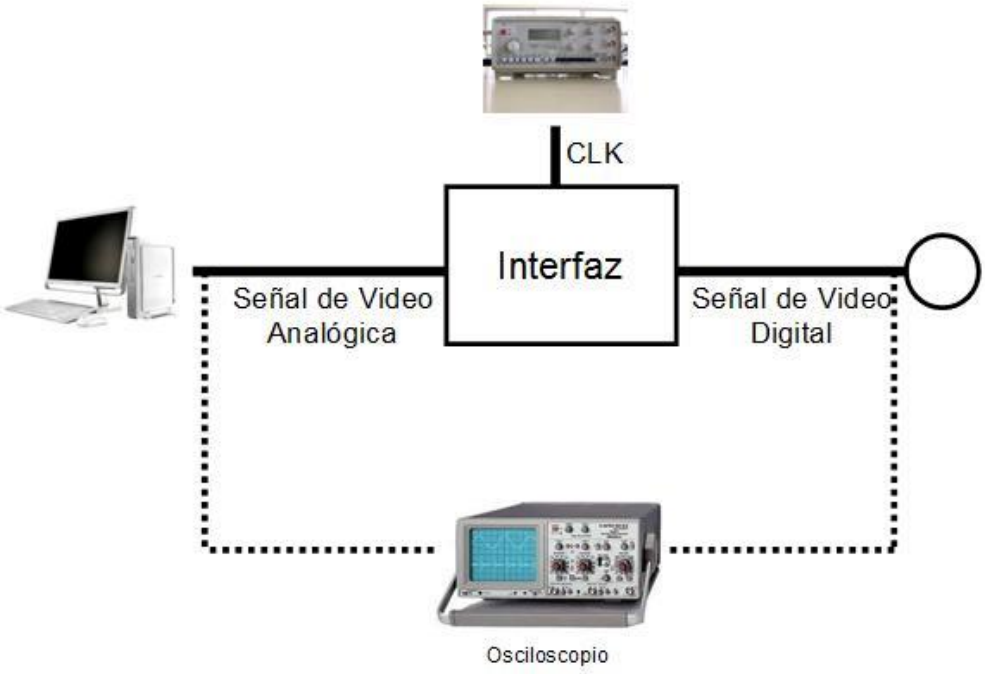


Figura 2.12 Diagrama de prueba 1.

Los resultados de esta prueba fueron favorables, ya que como vemos en la gráfica que presenta el osciloscopio en la Figura 2.13, se obtuvo una señal binaria de acuerdo a lo explicado anteriormente.

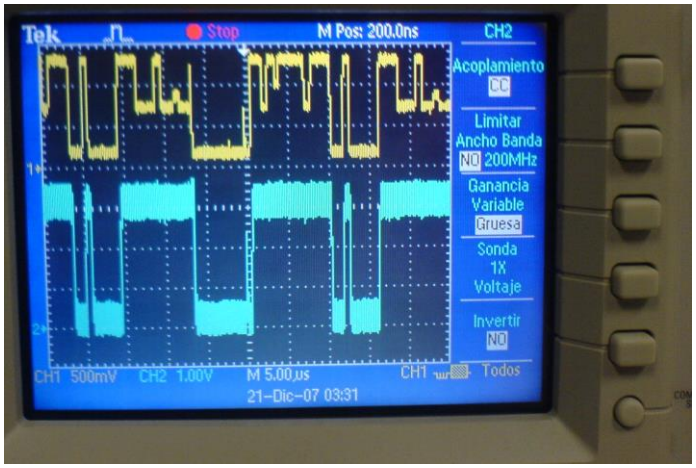


Figura 2.13 Resultados de prueba 1 (Vistos en el osciloscopio).

Como prueba final de este módulo se usó un monitor para desplegar la imagen binaria, para esto usamos el módulo D2SB de Xilinx que tiene el dispositivo programable para generar la señal de reloj de nuestro sistema. Para obtener la imagen en niveles blanco y negro triplicamos mediante el uso del

FPGA la señal digitalizada de color rojo para enviarlas como señales de tres colores a un convertidor RGB. Los resultados de está prueba fueron concluyentes para nuestra interfaz, ya que como podemos ver en las imágenes que se muestran en la Figura 2.14, el funcionamiento obtenido fue el deseado. En la figura, el monitor izquierdo recibe la señal directa desde la computadora y el monitor derecho recibe la señal digitalizada y solo despliega blanco o negro dependiendo del nivel de rojo de la imagen.

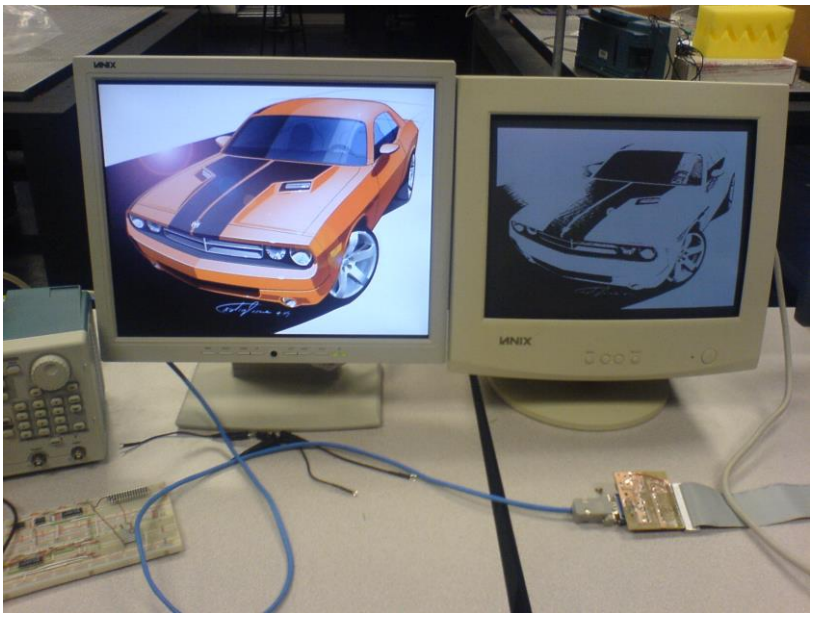


Figura 2.14 Resultados de la prueba 2.

2.5 CONCLUSIONES

Hemos diseñado y fabricado el módulo interfaz de conversión analógica digital para digitalizar señales de video provenientes de una tarjeta de video de una computadora. Nuestro módulo esta diseñado para recibir como señal de entrada de datos RGB de un formato de video VGA 640X480 a 60Hz, aunque este formato puede ser superior siempre y cuando se use para una señal monocromática. Las señales digitalizadas tienen niveles lógicos de 0V y 3.3V que son los niveles usados por el FPGA, pero las señales de VSync y HSync tiene niveles lógicos de 0V y 5V, aun así son toleradas por el dispositivo programable que permite el manejo de estos datos.

CAPÍTULO 3

MÓDULO DIGITAL PARA CONVERSIÓN DE VIDEO

3.1 INTRODUCCIÓN

Las señales provenientes de una tarjeta de video de una computadora son formatos estandarizados que podemos emplear para desplegar información visual en casi cualquier monitor de video. El objetivo en este proyecto, como ya se ha dicho anteriormente, es desplegar información visual en una pantalla de OLEDs monocromática. Sin embargo, la resolución de esta pantalla es muy baja (64x48 pixeles) por lo que es necesario tener un proceso de conversión para reducir el formato de información proveniente de la tarjeta de video de la computadora, que en este caso es de 640x480. Para llevar a cabo esta conversión diseñamos un módulo digital de conversión de video empleando un dispositivo digital programable contenido en una tarjeta de desarrollo. La tarjeta de desarrollo es la D2SB de Digilent Inc. que tiene un FPGA (Field Programmable Gate Array) Spartan II de Xilinx con 200,000 compuertas lógicas [7]. La descripción de los dispositivos; así como el desarrollo de este módulo se presenta en este capítulo.

3.2 FPGA (Field Programmable Gate Array)

Un FPGA es un dispositivo semiconductor que contiene bloques lógicos cuya interconexión y funcionalidad se pueden programar. La lógica programable puede reproducir funciones tan sencillas como las de una compuerta lógica hasta un sistema digital complejo. Estos dispositivos surgieron gracias a dos tecnologías: los PLDs (Programmable Logic Devices) o dispositivos lógicos programables y los ASIC (Application-Specific Integrated Circuit) o circuitos integrados de aplicación específica. A pesar de tener aplicaciones similares a los ASICs, los FPGAs son más lentos, además tienden a consumir más potencia. De igual manera, los FPGAs también tiene limitaciones en cuanto a complejidad en

los sistemas [6]. A pesar de esto, estos dispositivos tienen como ventaja su reprogramabilidad, la cual, les da mayor flexibilidad en comparación con los ASICs. El costo de desarrollo y adquisición de los FPGAs son bajos para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor.

Los primeros FPGAs fueron usados principalmente para hacer diseños de hardware de componentes que no existían en el mercado, ya que éste nos ahorra una buena cantidad de chips empleados. Esto se ve reflejado en la inversión de capital que tiene que realizarse al desarrollar un producto. Ciertos fabricantes usan para sus productos FPGAs que sólo se pueden programar una vez, por lo que sus ventajas e inconvenientes no se comparan con los ASICs y los FPGAs reprogramables.

Otra parte importante en la evolución de los FPGAs han sido los CPLDs (Complex Programmable Logic Device) o dispositivo complejo lógico programable. A diferencia de un PLD, estos dispositivos tienen un mayor nivel de integración ya que permite implementar sistemas más eficaces que utilizan menor espacio y mejoran la fiabilidad del diseño, reduciendo costos. Tanto los CPLDs como los FPGAs contienen un gran número de elementos lógicos programables. Si medimos la densidad de los elementos lógicos programables en compuertas lógicas equivalentes, es decir, el número de compuertas NAND equivalentes que podríamos programar en un dispositivo, podríamos decir que en un CPLD hallaríamos del orden de decenas de miles de compuertas lógicas equivalentes; mientras que un FPGA del orden de cientos de miles hasta millones de ellas.

Otra diferencia entre estos dispositivos es su arquitectura. Un CPLDs cuenta con una arquitectura más rígida basada en la suma de uno o más productos programables cuyos resultados nos reducen el número de biestables síncronos o flip-flops [6]. En cambio la arquitectura de los FPGAs, esta basada en un gran número de pequeños bloques utilizados para reproducir sencillas operaciones lógicas que cuentan a su vez con biestables síncronos. Esto representa una mayor flexibilidad debido a que nosotros podemos interconectar estos bloques de la manera que creamos más conveniente.

La arquitectura básica de un FPGA consiste en una matriz de bloques lógicos programables o CLB (Configurable Logic Block), así como canales de interconexión entre ellos. Existe varias configuraciones de bloques lógicos, en la Figura 3.1 [6] se muestra una configuración que es usada en los FPGA Spartan 2E de Xilinx. Por otro lado, los canales de interconexión están compuestos por un gran número de pequeñas pistas en paralelo de tal modo que cada entrada o salida puede conectarse a cualquiera de dichos segmentos del canal de interconexión.

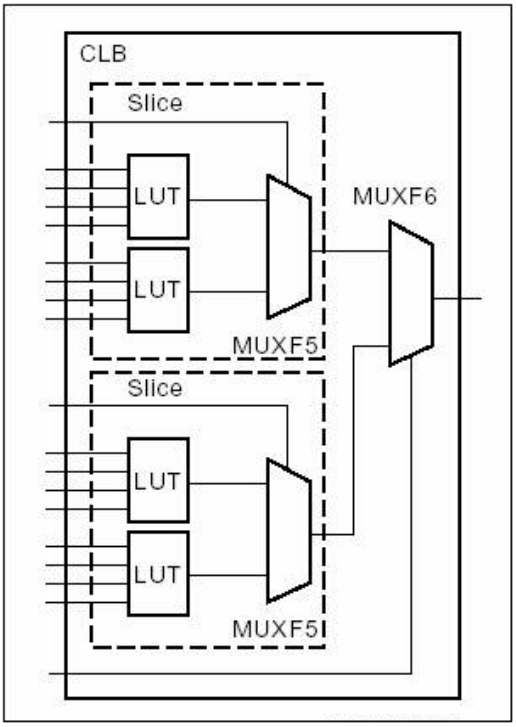


Figura 3.1 Esquema del bloque lógico programable de una FPGA Spartan 2E de Xilinx

Normalmente cada pequeña pista de un canal de interconexión comienza en un bloque lógico y termina en una caja de conmutación situada justo antes del comienzo del siguiente bloque lógico. Al programar los interruptores de la caja de conmutación se pueden construir pistas más largas. Hay diferentes líneas de conexión, por ejemplo para una línea de conexión que requiere velocidades más altas existen segmentos de pista de mayor longitud que cubren varios bloques lógicos. Además, existe un tercer tipo de recursos exclusivos de conexión: las líneas de cables dedicados a la transmisión de las señales de reloj. Esto se debe a que las señales de reloj tienen la característica especial de que han de estar

conectadas a un gran número de bloques por lo que han de llegar a todos los rincones del FPGA en el menor tiempo posible. En la Figura 3.2 [6] podemos observar a grandes rasgos la forma en que está constituido un FPGA con los bloques lógicos, las interconexiones programables y los bloques de entrada/salida.

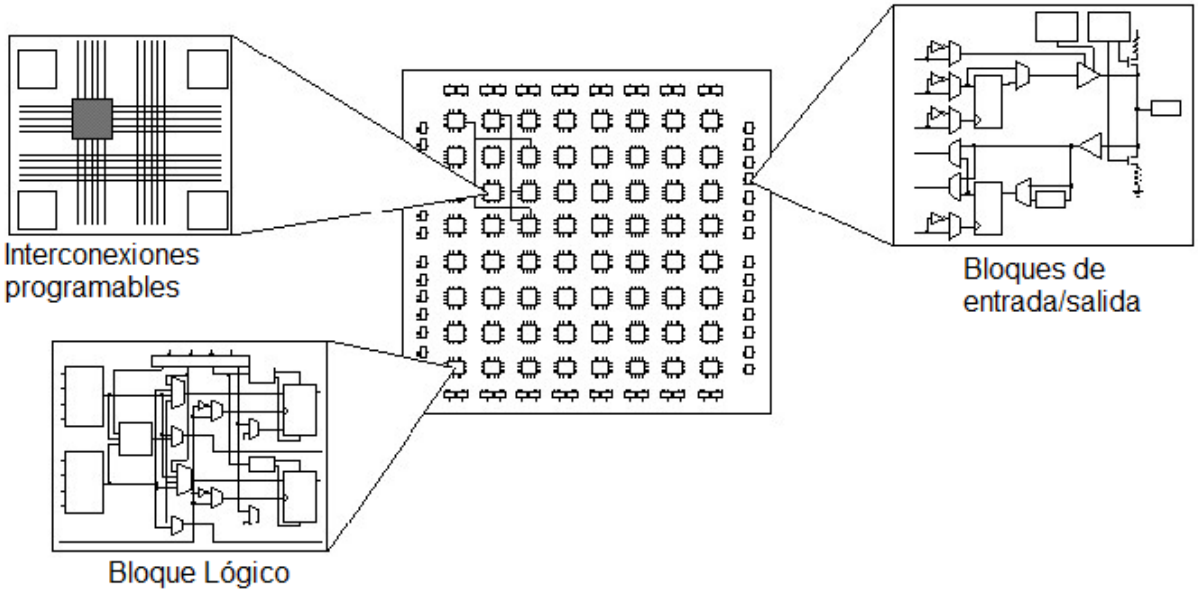


Figura 3.2 Diagrama completo de un FPGA.

Una jerarquía de interconexiones programables permite a los bloques lógicos de un FPGA estar interconectados según la necesidad del diseñador del sistema y a su vez ser reprogramados después del proceso de manufactura por el mismo diseñador, así el FPGA puede desempeñar cualquier función lógica necesaria. Una tendencia reciente ha sido combinar los bloques lógicos e interconexiones de los FPGAs con microprocesadores y periféricos relacionados para formar un sistema programable en un chip.

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. Estos dispositivos han ido creciendo en tamaño lo que nos ha permitido poder realizar un mayor numero de aplicaciones, así como estructuras mas complejas como un procesador. Un ejemplo de esto, es que muchos de los fabricantes de procesadores construyen

sus prototipos con FPGAs ya que el costo de un ASIC en algunos casos puede ser injustificable.

Llevado al extremo, el uso de los FPGAs coloca al software directamente sobre el layout del chip. Actualmente 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 (Hardware Description Language) o lenguajes de descripción de hardware. Los HDLs más utilizados son:

- VHDL
- Verilog
- ABEL

Cualquier circuito de aplicación específica puede ser implementado en un FPGA. 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. Sin embargo muchas otras áreas han empezado a emplear su uso cada vez con mayor frecuencia, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo.

Para el desarrollo de este módulo hemos empleado el lenguaje de descripción de hardware VHDL (V*ery High Speed Integrated Circuit*, H*ardware* D*escription* L*anguaje*). VHDL es un lenguaje que está diseñado para describir casi cualquier tipo de hardware [10]. Aunque puede ser usado de forma general para describir cualquier circuito, se usa principalmente para programar PLD, FPGA, ASIC y similares.

3.3 TARJETA DIGILENT D2-SB

La tarjeta empleada fue Digilent D2-SB la cual nos provee un desarrollo completo de circuitos centrada en un FPGA Xilinx Spartan 2E que incluye, según su hoja manual de referencia [7]:

· Un FPGA Xilinx XC2S200E-200 con 200,000 compuertas y 350MHz en operación.
· 143 entradas/salidas interconectadas a un conector de expansión de 40 patillas.
· Un puerto para JTAG 18V02 configurable a una Flash ROM.
· Dos reguladores de potencia 1.5A (1.8V y 3.3V).
· Un oscilador SMD 50MHz y puerto para un segundo oscilador.
· Un puerto programable para JTAG.
· Un LED de estado y un pushbutton para entrada/salida.

La D2-SB esta diseñada para trabajar con todas las versiones de Xilinx ISE CAD, así como una gran colección de tarjetas de expansión las cuales nos permiten agregar más entradas y salidas, también cuenta con varios puertos como USB y Ethernet. La D2-SB cuenta con su propia fuente de alimentación y su propio cable de programación es por eso que los diseños pueden ser implementados inmediatamente sin la necesidad de cualquier hardware adicional.

3.3.1 Descripción de operación

La Digilab D2-SB provee solamente el soporte esencial de los dispositivos para un FPGA Spartan 2E, incluye la fuente de poder y de reloj. Todas sus entradas y salidas disponibles estas enlazadas a sus conectores de expansión de 40 patillas, 100 mil ($^{1}/_{1000}$ pulgadas) de distancia entre cada entrada de patilla. También incluye un LED y un pushbuttom. La Figura 3.3 [7] muestra un diagrama de bloques de la tarjeta D2-SB.

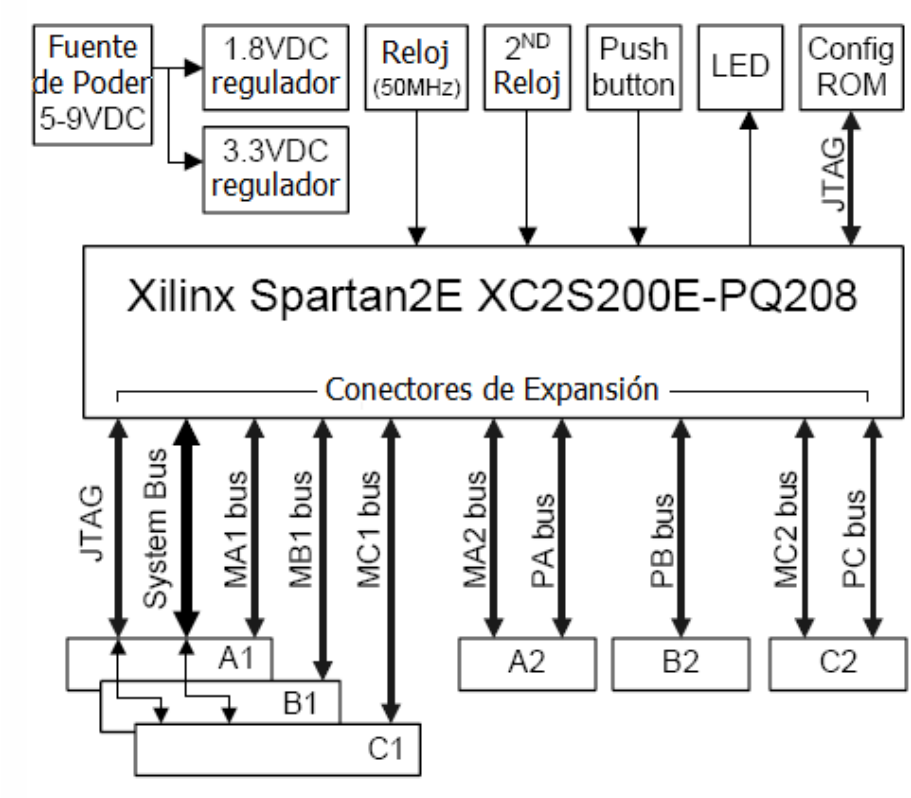


Figura 3.3 Diagrama de bloques de la tarjeta D2-SB.

Esta tarjeta esta diseñada principalmente para que tarjetas periféricas puedan ser colocadas ya que cada uno de sus seis conectores de expansión provee su voltaje, el cual es de 3.3V, su tierra, así como 32 entradas y salidas de FPGA. No solo cuenta con los conectores para las patillas del FPGA, también cuenta con más conectores como A1, B1 y C1 que comparten 18 patillas del bus del sistema. Sin embargo no todas las patillas del puerto B son usados. Las señales de JTAG están también conectadas a A1, B1 y C1. Esto permite a las tarjetas periféricas poder ser configuradas junto con el FPGA.

Para la programación, el FPGA Sparta 2E, la 18V00 ROM en la D2-SB y cualquier otro dispositivo programable en cualquier tarjeta adjuntada a la D2-SB, puede ser programada a través del puerto JTAG. Este puerto esta conectado al FPGA y a la ROM alrededor de la tarjeta por 4 puertos como se puede ver en la Figura 3.4 [7].

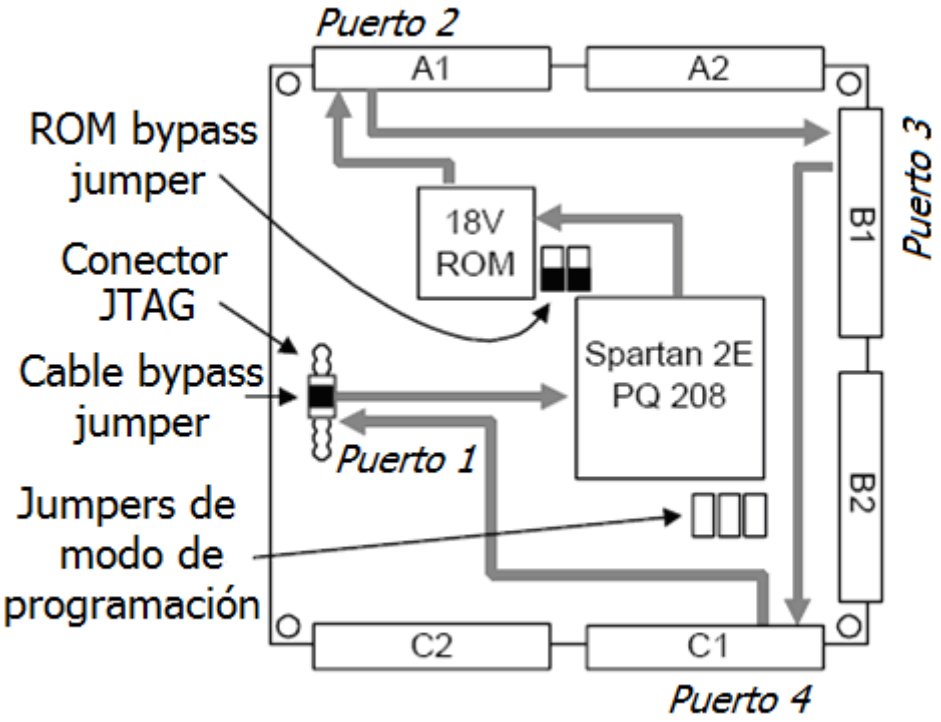


Figura 3.4 Descripción de los puertos de la tarjeta D2-SB.

El puerto de configuración principal usa un conector JTAG estándar de 6 patillas, los otros puertos disponibles son A1, B1 y C1, estos a su vez son puertos bidireccionales. En caso de que no haya nada conectado en un puerto, el buffer de la D2-SB remueve el conector del JTAG.

Por otro lado, si hay una tarjeta periférica con un dispositivo JTAG conectado, la tarjeta hace un tipo de escaneo, el cual en este caso se deshabilita para que cualquier JTAG pueda ser configurado.

Si un módulo (Ethernet, USB, EPP paralelo y seriales) está conectado a uno de los tres puertos de expansión JTAG, entonces el módulo puede correr el escaneo de JTAG para programar todos los dispositivos.

El escaneo puede ser activado a través del puerto primario conectando la D2-SB a la computadora y programando el JTAG usando el cable de programación. Después se corre la configuración del software usando “auto-detect”. La configuración del software nos permite programar cualquier dispositivo de manera selectiva con cualquier archivo de configuración disponible. Si no hay

ninguna ROM cargada en el puerto IC5, el jumper debe de estar puenteado en JP1 y JP2 en la localización “Bypass ROM” para conectar el JTAG en el puerto de la ROM. En caso de que sí haya una ROM en el puerto IC5, esta puede ser activada cambiando los jumpers JP1 y JP2 en la posición “Include ROM”. Si un programa se encuentra presente en el puerto IC5, el FPGA automáticamente tendrá acceso a la ROM para configurarla si es que los jumpers se encuentran en sus 3 posiciones de J8 de manera correcta (M2, M1, M0). Para poder activar el escaneo a través de un módulo en cualquier puerto A1, B1 o C1, tenemos que poner un jumper en el cabezal principal de JTAG en las patillas TD1 y TD0.

En cuestión de alimentación, la Tarjeta D2-SB usa dos reguladores de voltaje LM317 que producen 1.6V de DC para alimentar a Spartan 2E y una fuente para todas sus entradas y salidas de 3.3V de DC. Ambos reguladores tienen buena capacitancia bypass, lo que permite su trabajo a mas de 1.5A de corriente o menos de 50mV de ruido. La alimentación puede ser administrada por un transformador de bajo costo y esta debe de ser entre los 6 y 12V de DC.

La D2-SB es una PCB de cuatro capas con sus propios planos de VCC y GND. La mayor parte del plano VCC es para 3.3V, con una isla debajo del FPGA a 1.8V. El FPGA y los demás circuitos integrados de la tarjeta tienen una capacitancia bypass lo más cerca posible de cada patilla de VCC.

La corriente total de la tarjeta depende de la configuración del FPGA, la frecuencia de reloj y cualquier conector externo.

Todas las señales de entrada y salida del FPGA usan el voltaje de VCC0 administrado por la fuente de voltaje de 3.3V. Si otro voltaje de VCC0 es requerido, la salida del regulador puede ser modificada cambiando R12 de acuerdo a la siguiente manera: VCC0 = 1.25 (1 + R12/R11).

Para las señales de reloj, la D2-SB provee un oscilador primario de 50MHz y un conector para un segundo oscilador.

El oscilador primario esta conectado en la entrada GLK2 del Spartan 2E (patilla 182) y el secundario en GCLK3 o patilla 185. Ambas entradas de reloj pueden correr la DLL del Spartan 2E, permitiendo frecuencias internas cuatro veces mayor a la señal de reloj externa. Cualquier oscilador de 3.3V de medio tamaño con un empaquetado DIP, puede ser cargado en el conector del segundo oscilador.

Si se desea trabajar con tarjetas adicionales, la D2SB cuenta con seis conectores de expansión, estos conectores de expansión A1-A2, B1-B2 y C1-C2 son de un ángulo derecho de 2x20 cabezales con un espacio entre ellos de 100 mils. Los 6 conectores cuentan con su propia tierra en la patilla 1, VU en la patilla 2 y 3.3V en la patilla 3. Las patillas 4-35 son señales de entrada y salida conectadas al FPGA y las patillas 36-40 están reservadas para el JTAG y o las señales de reloj. La Figura 3.5 [7] muestra como están enlazados los conectores de expansión.

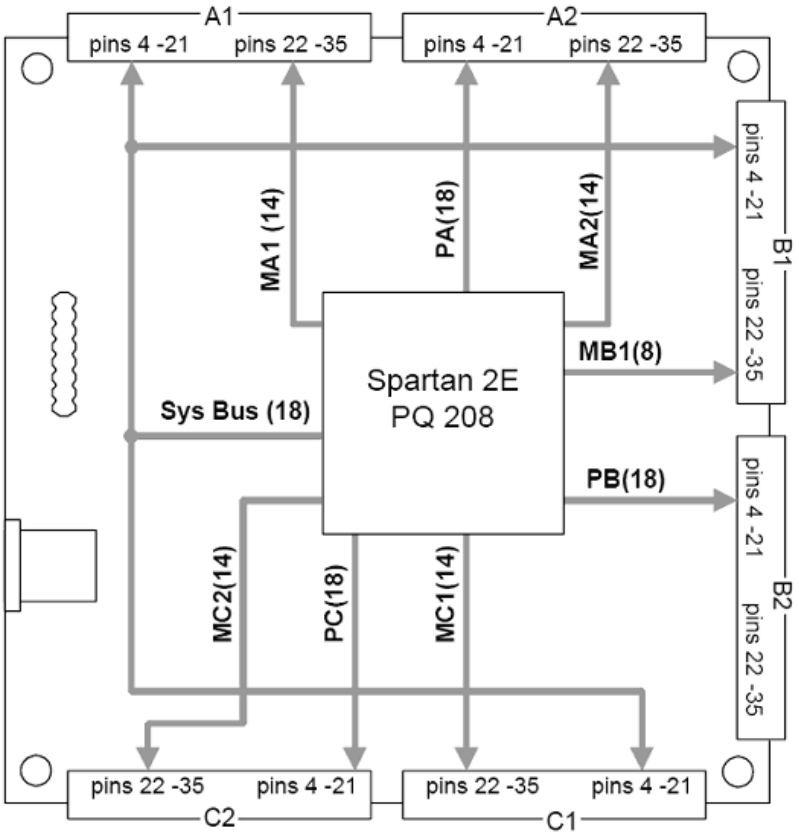


Figura 3.5 Descripciones de conectores de expansión de la tarjeta D2-SB.

Los cabezales de expansión provén 192 posibles señales, pero Spartan 2E-PQ208 solo cuenta con 143. Así, algunas señales del FPGA esta conectadas a más de un conector. En particular, las patillas menores a la 18 (4-21) de los conectores A1, B1 y C1 están conectadas a la misma patilla del FPGA la cual es “system bus” o patilla 18. Las demás señales permanecen disponibles para cualquier posición individual deseada.

La patilla 18 de los conectores A2, B2 y C2, esta designada para “buses periféricos” y cada uno de estos buses (PA, PB y PC) usan únicamente 18 señales.

Las patillas mayores a la 14 en cada conector de expansión (22-35) están diseñadas como “buses de módulo”. Los conectores A1, A2, C1 y C2 cuentan con un bus completo de módulos (MA1, MA2, MC1 y MC2) como se pueden ver en la Figura 3.5.

Para el conector B, el FPGA no cuenta con suficientes patillas para poder conectar completamente el bus de módulo, solo se encuentran conectadas las 8 patillas del MB1 y no hay ninguna otra para la mitad del conector B2.

En la D2SB el bus de sistema es un protocolo usado por ciertas tarjetas de expansión que asemejan un bus de microprocesador de 8 bits. Este usa ocho líneas de datos, seis líneas de direcciones, un habilitador de escritura (WE), un habilitador de salida (OE) para funciones de lectura, un chip de selección y un reloj para sincronizar las transferencias. En la Figura 3.6 [7] se muestra el diagrama de los ciclos de lectura y escritura con los que funciona el bus de sistema de la tarjeta. Sin embargo, cualquier modelo de bus o de tiempos puede ser usado, modificando el circuito en el FPGA y adjuntando el dispositivo periférico.

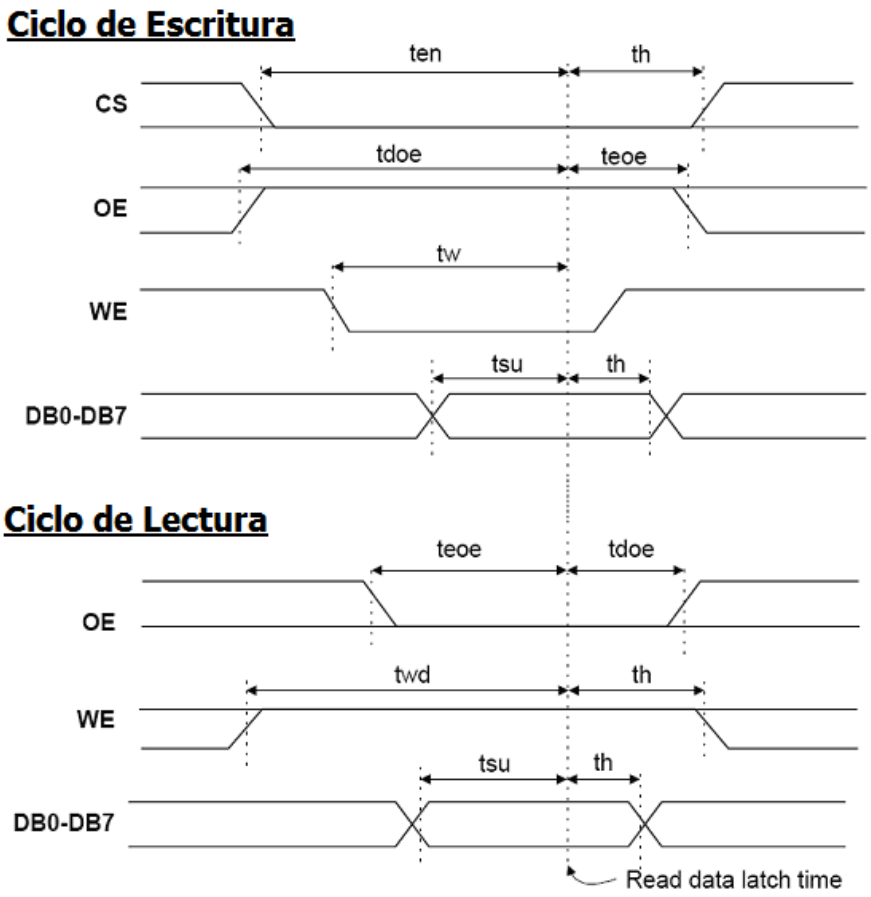


Figura 3.6 Diagrama de ciclo de lectura y escritura del bus de sistema.

3.4 DISEÑO E IMPLEMENTACIÓN DEL MÓDULO

Este módulo toma la información que proviene de la computadora, en formato VGA, y la convierte en el formato de nuestra pantalla de 64x48 OLEDs. Este formato fue elegido porque los OLEDs son fabricados en matrices, además la simplicidad de este formato no requiere mucha manipulación de la información, requiriendo muy pocos recursos de los sistemas digitales.

Para el diseño de este módulo digital, se debe entender primeramente las características de la entrada de información, en este caso, serán utilizadas tres señales de una interfaz de video en formato VGA. Dos señales son de sincronía, VSync y HSync; la otra es una señal con información de la imagen, Data RED, la cual es recibida después de un proceso de conversión analógica a digital. La

razón por la cual son tomadas tan pocas señales de la interfaz de video es que, con éstas es suficiente para poder desplegar una imagen monocromática.

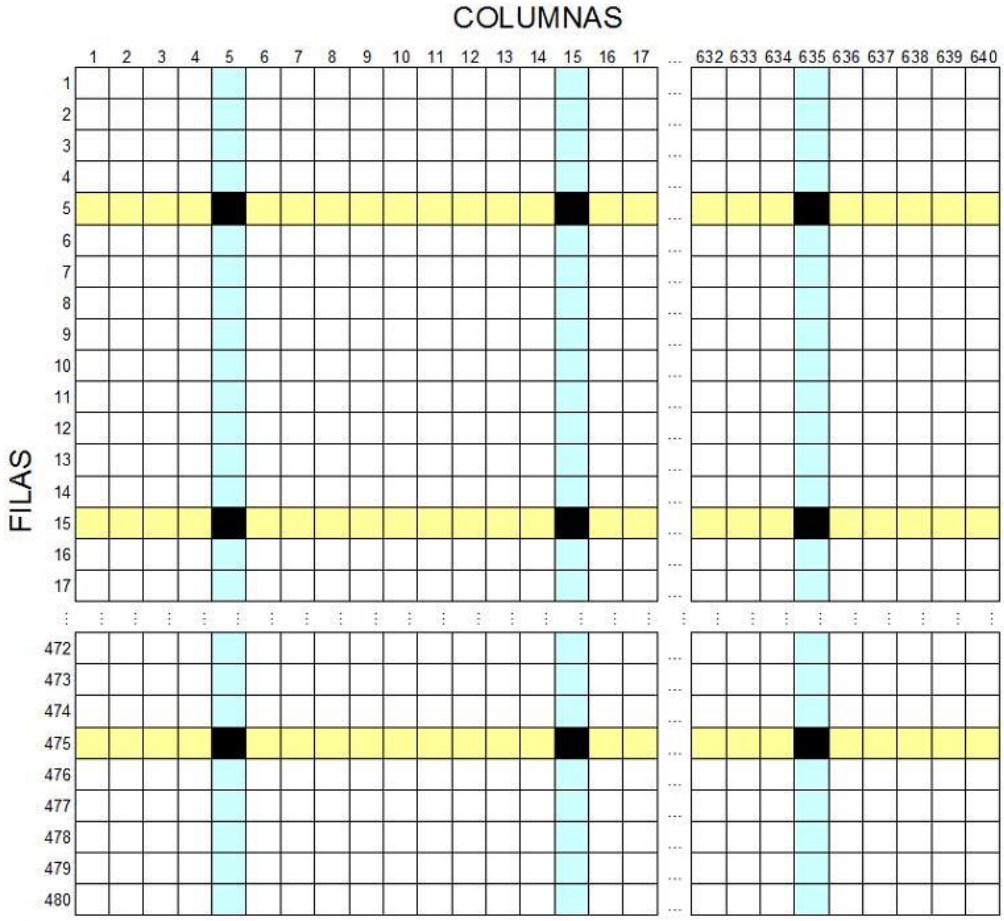


Figura 3.7 Muestreo de la imagen para convertirla de 640x480 a 64x48.

Ahora bien, sabemos que nuestra imagen de entrada al sistema es de 640X480 pixeles, pero para nuestra pantalla solo requerimos una resolución de 64X48. Por lo tanto se debe reducir la cantidad de pixeles en una décima parte en las filas y en las columnas de la imagen. Para obtener en nuestra pantalla una imagen similar a la imagen que proviene de la computadora (aunque con mucha menor resolución), decidimos realizar el diseño de un sistema digital que realiza un muestreo a la imagen VGA consistente en enumerar las columnas en orden desde 1 hasta 640. De tal modo que de cada diez pixeles que recibimos, tomemos solo uno en el punto medio de esa sección de la imagen. Dicho en otras palabras, de los pixeles del 1 al 10, solo tomaremos el píxel 5, de los pixeles del 11 al 20 solo tomaremos el número 15, y así sucesivamente. De esta manera, de los 640 pixeles que tenemos en la imagen original, solo nos quedaremos con 64.

Lo mismo se realiza para las filas de la imagen, de manera que solo nos queda una imagen de 64x48 pixeles. Un esquema que ilustra el procedimiento de muestreo de la imagen representada en una cuadrícula se muestra en la Figura 3.7. Como podemos apreciar, los pixeles de color negro son los que pertenecerán a la figura que se desplegará en nuestra pantalla de baja resolución.

Toda vez que la imagen original es muestreada, definimos el modo de desplegar la nueva imagen reducida en nuestra pantalla. Como es conocido, el formato VGA envía las imágenes de manera serial, fila por fila de arriba a abajo, y en cada fila, envía también los pixeles de manera serial de izquierda a derecha. Para nuestro diseño decidimos que en nuestra pantalla usaríamos un despliegue de fila por fila de arriba abajo, en donde pixeles de una misma fila se desplieguen de manera paralela. En otras palabras, en nuestra pantalla, primeramente se desplegará la primera fila completa, después la segunda fila completa, y así sucesivamente será el barrido de filas hasta completar las 48 filas que conforman nuestra pantalla y formar la imagen completa. Para lograr esto, se requerirán dispositivos de almacenaje que provean los datos de forma paralela. Bajo este panorama general de la operación de este módulo se realizó el diseño del sistema con VHDL.

3.4.1. Descripción y simulación del diseño.

Para poder empezar, recordaremos en forma breve como envía las imágenes el formato VGA con resolución de 640X480. Primeramente un pulso de la señal VSync, nos indica que la imagen va a empezar a desplegarse desde el principio, después del pulso VSync empieza el envío de datos, se empieza a enviar la primera línea en la señal de datos, que en nuestro caso, es la señal de datos del color rojo (Data RED), cuando termina de enviar la primera línea, un pulso en la señal de HSync nos indica que empezará a enviar la segunda línea, al finalizar la segunda línea, nuevamente un pulso en la señal de HSync nos indica que empezará a transmitir la tercera línea, y así sucesivamente, hasta completar todas las líneas de la imagen, cuando todas las líneas se han transmitido, nuevamente un pulso en la señal de VSync nos indica que se va a empezar a enviar una imagen nueva [15].

Por otra parte, como recordamos, la señal de datos es una señal analógica, por lo que antes de ser transmitida al FPGA, debe ser convertida a digital, y esta conversión se lleva a cabo en la interfaz, que ya está diseñada, pero el programa debe enviar a la interfaz la señal de reloj para el muestreo del convertidor por lo que se debe tomar en cuenta, que la señal de datos Data RED envía pixeles a una frecuencia de 25MHz, por lo que la frecuencia a la cual debe trabajar el programa y la frecuencia que debe ser enviada al convertidor es de 25MHz. Esto con el fin de que programa pueda estar trabajando píxel por píxel.

Otra característica a tomar en cuenta es que, en el formato VGA, como ya se hizo mención en la introducción, no solo envía datos de la imagen a desplegar, si no que en cada imagen, las primeras y las últimas líneas que envía no tienen información sobre la imagen, por lo que para formar una imagen de 480 filas, no manda 480 pulsos de HSync por cada pulso de VSync, si no que envía más de 480, por lo que se puede decir que envía más líneas que las 480 que conforman la imagen, unas antes y otras después de las líneas efectivas que pertenecen a la imagen. Pasa lo mismo con los pixeles que envía en cada línea, envía más pixeles que los 640 que conforman la imagen, es decir, después del pulso de HSync que indica que se empezará a desplegar una nueva línea, los primeros pixeles que envía no pertenecen a la imagen, después de estos, envía los 640 pixeles pertenecientes a la imagen, y por último, antes del siguiente HSync, envía otra cantidad de pixeles que no pertenecen a la imagen. Esta cantidad de líneas y pixeles adicionales, son siempre un valor constante, por lo que no representa problema a la hora de encontrar la imagen en los datos que se envían por Data RED.

Para llevar a cabo el objetivo del proyecto de una manera más adecuada, fue necesario desarrollar un conjunto de módulos, cada uno con un objetivo particular dentro del programa de VHDL. En la Figura 3.8 se observa el diagrama de bloques que conforman el proyecto completo.

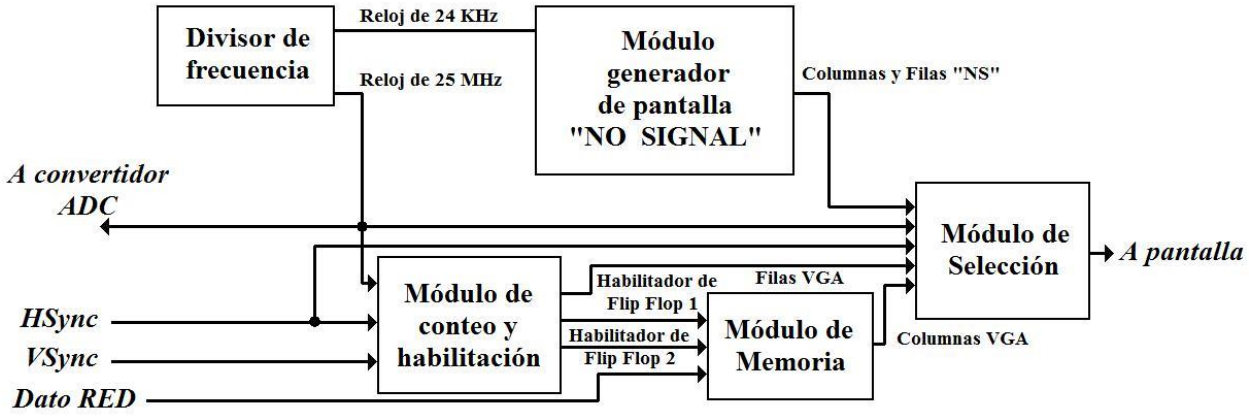


Figura 3.8 Diagrama de bloques del proyecto completo de VHDL.

En el diagrama se puede observar primeramente un divisor de frecuencia que se encarga de generar a partir de la señal de reloj que recibe el FPGA de 50MHz, las señales de reloj necesarias para el diseño completo. Siguiendo con el diagrama de bloques en la Figura 3.8, se observa también un módulo de conteo y habilitación, el cuál se encarga de contar pulsos en la señal de reloj de 25MHz y en la señal HSync con el objetivo de saber en que momento se está recibiendo desde la computadora una de las filas que se desean desplegar, así como cada uno de sus pixeles, con esta información se generan señales para habilitar memorias y almacenar solo los pixeles a desplegar en la pantalla, además, en este módulo se genera la señal que activa las filas en la pantalla cuando se despliega una imagen desde VGA. De la misma manera, se observa en la Figura 3.8 un módulo de memoria, el cual recibe la señal de los datos de color rojo digitalizados y almacena el valor de los pixeles que se desean desplegar, con esto se genera en su salida la información de una línea completa de pixeles de una imagen proveniente de VGA que se va a desplegar en la pantalla. Otro módulo de gran importancia es el generador de pantalla “NO SIGNAL”, ya que la pantalla que se está diseñando tiene como característica adicional la función de desplegar un mensaje de “NO SIGNAL” (sin señal) en caso de que no se reciba información desde VGA, y este módulo cumple con esa parte del programa, generar la imagen con el mensaje de “NO SIGNAL” y enviarlo a la pantalla línea por línea, a una frecuencia de 24KHz cada línea. Por último, se tiene el módulo de selección, el cual recibe las señales que despliegan en la pantalla las imágenes, una proveniente desde VGA y otra desde el generador de pantalla “NO SIGNAL”, y en base a si está presente la señal de HSync decide enviar a la pantalla una imagen

u otra, es decir, si está presente la señal de HSync, el módulo de selección envía a la pantalla las imágenes que provienen desde VGA, si la señal de HSync no está presente, le módulo envía a la pantalla la imagen con el mensaje “NO SIGNAL”.

Divisor de frecuencia

Primeramente, en la Figura 3.8 se puede observar un divisor de frecuencia, el cual genera dos señales de reloj, una de 24KHz y otra de 25MHz, ambas a partir de la señal del reloj externo de 50MHz que recibe el FPGA. En las Figuras 3.9 y 3.10 se pueden observar los resultados de la simulación de este componente. La Figura 3.9 se presenta en un periodo de tiempo pequeño, por lo que se puede distinguir que la frecuencia de la señal de salida de 25MHz (slow_clk) es la mitad que la señal de entrada de 50MHz (clk). Por otra parte, en la Figura 3.10 solo se observa adecuadamente la señal de reloj de 24KHz (slow_clkNS) ya que en ese periodo de tiempo, las otras dos señales realizan una cantidad muy grande de transiciones y el programa solo las representa como una barra continua.

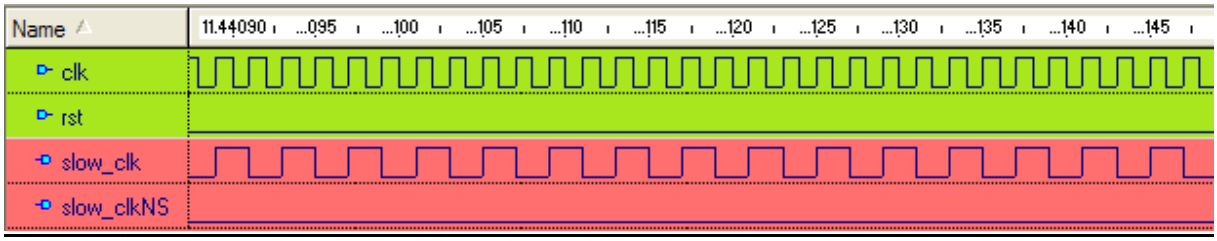


Figura 3.9 Divisor de frecuencia.

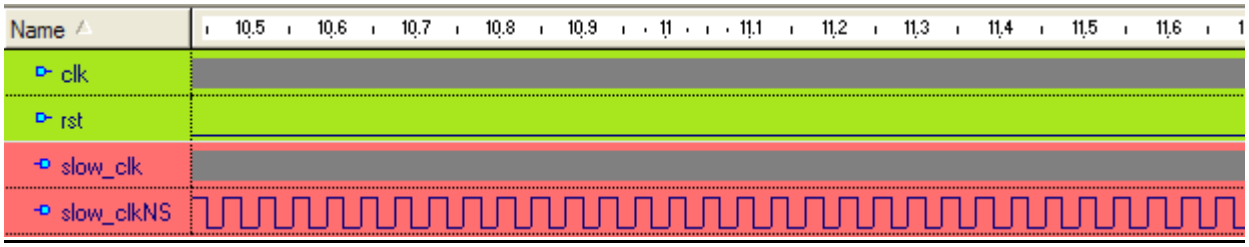


Figura 3.10 Divisor de frecuencia.

Módulo de conteo y habilitación

Este módulo está compuesto por 2 contadores y 3 codificadores como se muestra en el diagrama de bloques de la Figura 3.11.

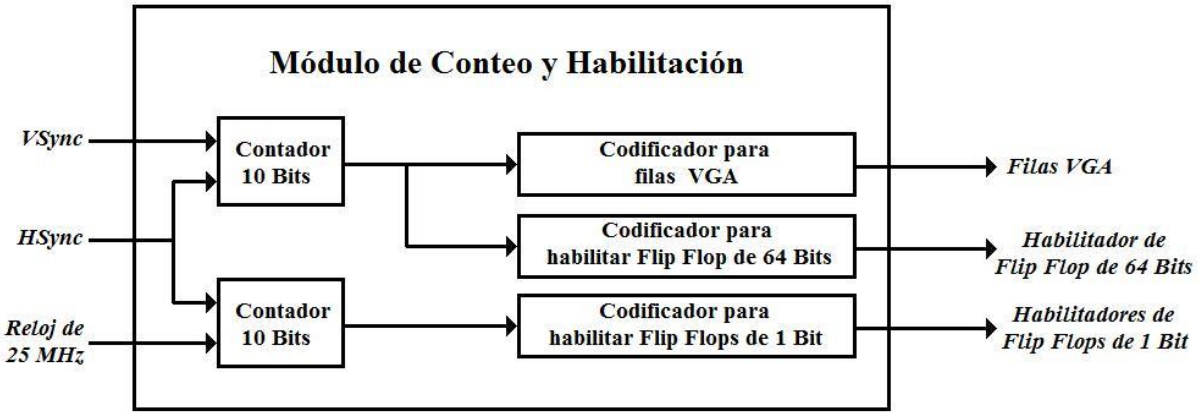


Figura 3.11 Diagrama de bloques del módulo de conteo y habilitación.

El primer contador recibe las señales VSync y Hsync, este contador reestablece su salida en 0 cuando hay un pulso bajo de VSync, a partir de ese momento, empieza a contar los pulsos HSync, y es reestablecido nuevamente con otro pulso de VSync, su objetivo es saber cual fila está siendo enviada en cada momento desde VGA.

Las Figuras 3.12 y 3.13 muestran los resultados de las simulaciones de este componente, en el caso de la Figura 3.12 se puede observar que el valor de la salida del contador está en 0 mientras el valor de VSync permanece en estado bajo, cuando la señal de VSync cambia a un estado alto, el valor del contador empieza a aumentar en 1 por cada pulso de la señal de HSync. En la Figura 3.13, se muestra que el valor del contador restablece su valor a 0 cuando la señal de VSync cambia su valor a un estado bajo, y se mantiene en 0 hasta que el valor de VSync cambie nuevamente a un estado alto.

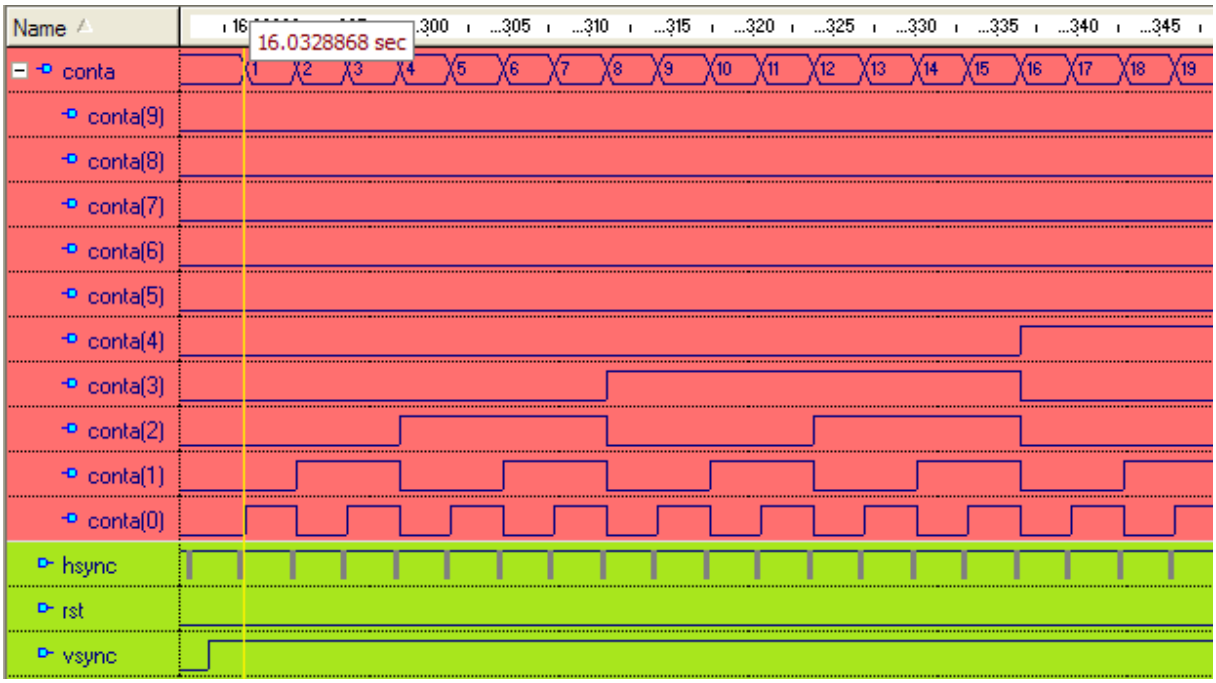


Figura 3.12 Contador de pulsos HSync (inicio).

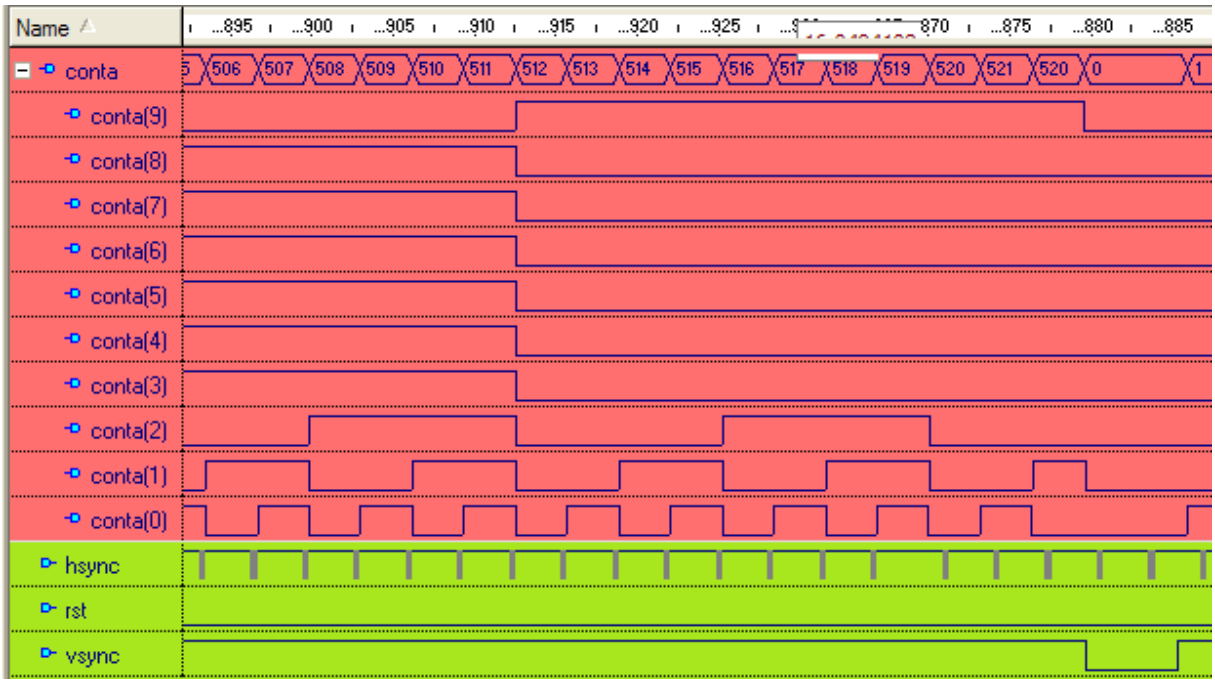


Figura 3.13 Contador de pulsos HSync (final).

El valor que se obtiene a la salida de este contador de HSync es utilizado como entrada para dos codificadores, como se muestra en el diagrama de la Figura 3.11, llamados codificador para filas VGA y codificador para habilitar flip-flop de 64 bit, por sus respectivas funciones.

El codificador para filas VGA recibe el valor del contador, y produce la señal que activa las filas en la pantalla para desplegar una imagen desde VGA, esto es posible porque es sabido que la fila número 38 después del pulso VSync es la fila número 5 de la imagen a desplegar, por lo tanto, en el periodo en que el contador de HSync tiene un valor de 38, el FPGA estará almacenando la información de los 64 pixeles de esa línea de imagen que se van a desplegar, por este motivo, ninguna de las filas de la pantalla debe estar activada. Cuando el contador de HSync cambie su salida a 39, significa que la línea 5 de la imagen ha sido enviada completamente desde la computadora y que las columnas de la pantalla ya están activadas con la información de la primera línea que se debe desplegar, por lo tanto, desde el pulso 39 hasta el pulso 47 del contador de HSync, el codificador activa la primera fila de la pantalla. Posteriormente en el pulso 48 del contador de HSync desactiva nuevamente todas las filas para que el FPGA almacene en la memoria los 64 pixeles de la fila 15 de la imagen, usando la misma secuencia, desde el pulso número 49 al pulso 57 del contador de HSync, el codificador activará la fila número 2 de la pantalla, y continúa este proceso hasta completar toda la imagen en la pantalla. En la Figura 3.14 se observa el resultado de la simulación de este codificador de filas VGA, donde es posible ver que desde el pulso 39 al pulso 47, está activada en la pantalla la fila 1, desde el pulso 49 al pulso 57 estará activada la fila 2, desde el pulso 59 al pulso 67 lo estará la fila 3, y así sucesivamente, mientras que en los valores intermedios a estos rangos, las 48 filas permanecen inactivas.

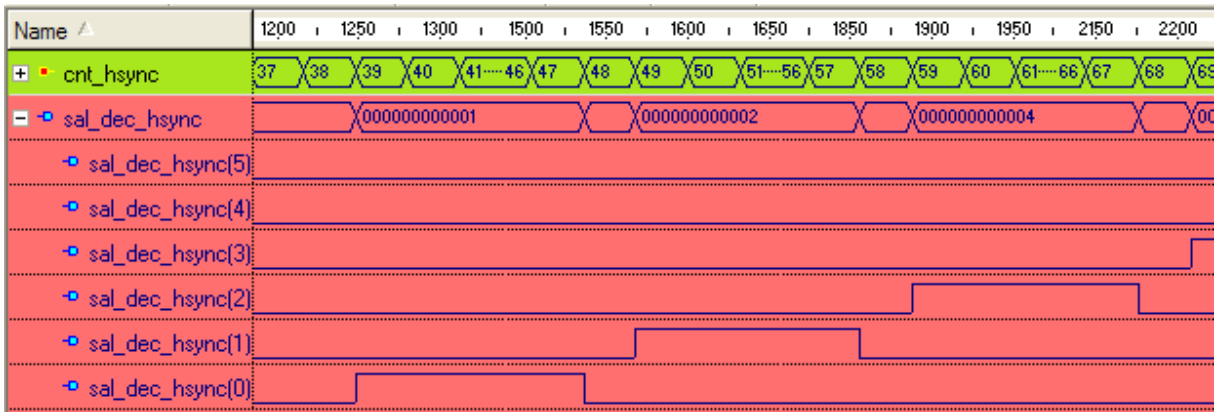


Figura 3.14 Codificador para filas VGA.

En la Figura 3.15 se observa un panorama más amplio de este codificador, con lo cual se puede ver mejor que funciona de una manera favorable para el objetivo que debe cumplir.

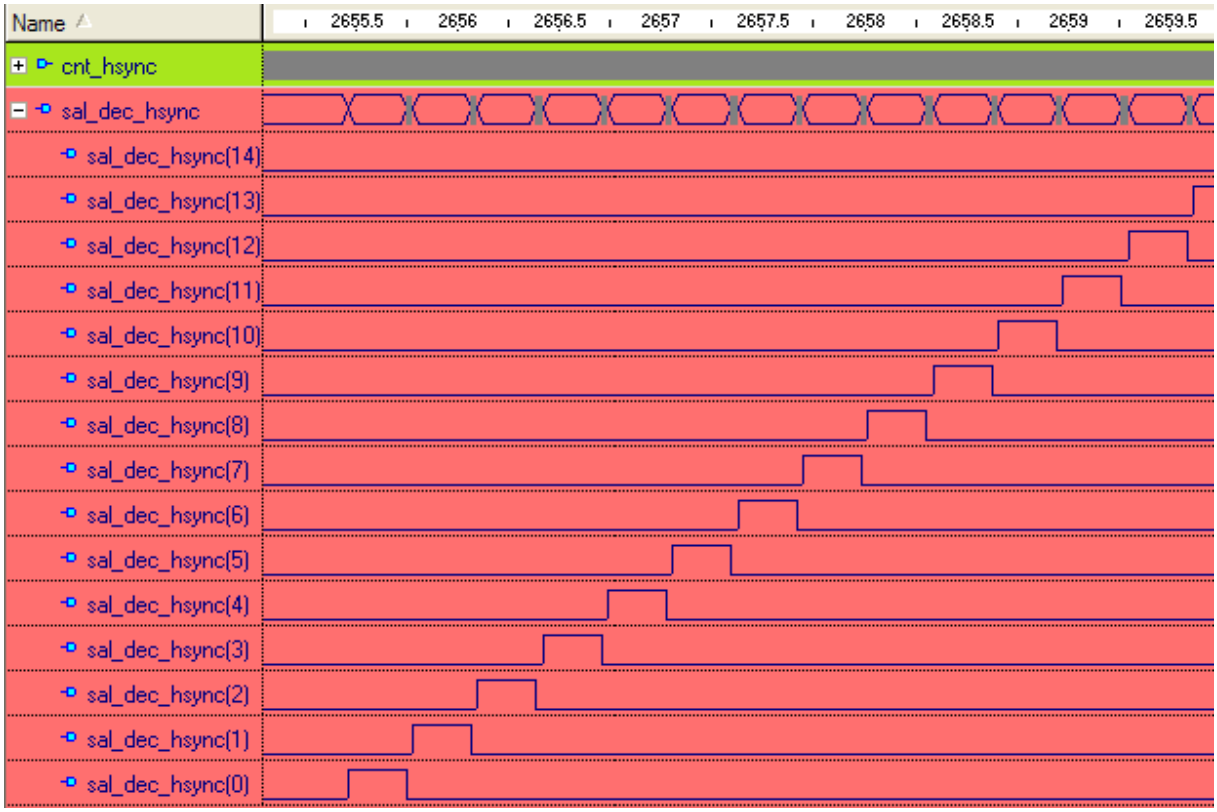


Figura 3.15 Codificador para filar VGA (Panorama más amplio).

El otro codificador que recibe el valor del contador de HSync, el codificador para habilitar flip-flop de 64 bit, como su nombre lo indica, habilita un flip-flop de 64 bit que se encuentra en el módulo de memoria, el cual será bien definido posteriormente, con el objetivo de que los pixeles que se desean desplegar, cuando estén siendo transmitidos desde la computadora, sean almacenados en este flip-flop, por lo tanto, el flip-flop solo es habilitado cuando la computadora está enviando una fila que se desea desplegar en la pantalla, como se mencionó, esto ocurre cuando el contador de HSync tiene valores de 38, 48, 58, etc. debido a esto, la Figura 3.16 muestra el resultado de la simulación donde es visible que solo es activada la salida del codificador en los valores adecuados del contador.

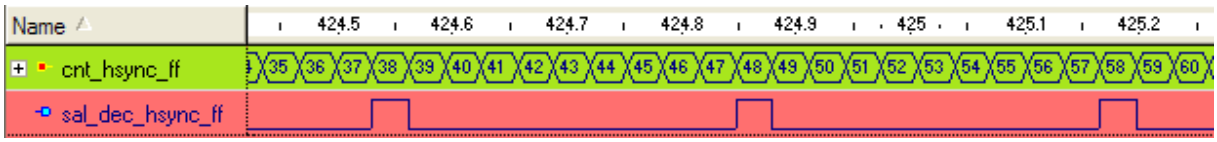


Figura 3.16 Codificador para habilitar flip-flop de 64 bits.

Retomando la Figura 3.11, se observa que faltan por definir dos componentes de este módulo, un contador y su respectivo codificador. Este contador tiene un funcionamiento muy similar al primer contador, la estructura del código en VHDL es la misma, las únicas diferencias que existen entre ambos contadores son, que en este nuevo contador su salida es reestablecida en 0 con un pulso bajo de la señal de HSync y que cuenta pulsos de reloj de 25MHz. Por este motivo, los resultados de las simulaciones de este contador que se presentan en las Figuras 3.17 y 3.18, son muy similares a los resultados del contador de HSync, con la diferencia del origen de sus señales de entrada, es decir, en este contador el reloj de 25MHz (clk) ocupa la entrada que Hsync ocupaba en el contador anterior, como objeto de cuenta y, de la misma manera, HSync ocupa la entrada VSync como pulso de reestablecimiento. El objetivo de este contador de pulsos de reloj es conocer en que momento está siendo enviado cada píxel de cada línea de la imagen desde la computadora. Cabe resaltar que, con los valores de los dos contadores de este módulo, se conoce que número de píxel está siendo enviado a cada momento y a que número de línea pertenece, información indispensable al momento de realizar el muestreo de la imagen.

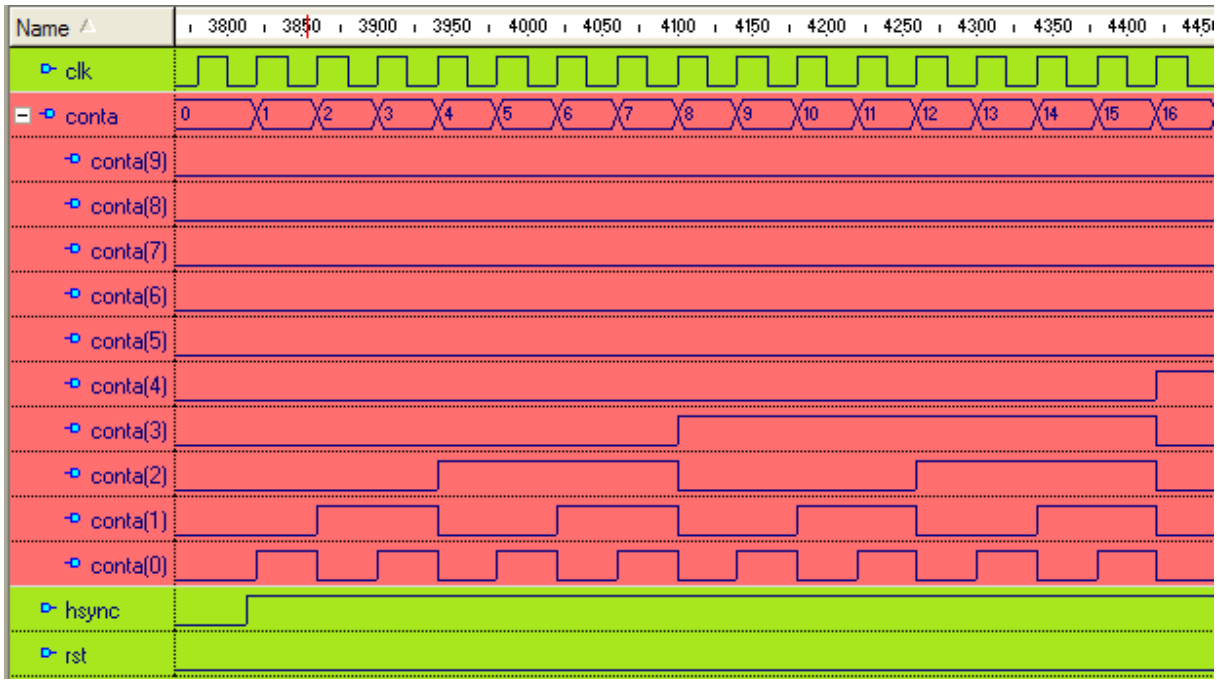


Figura 3.17 Contador de pulsos de reloj (inicio).

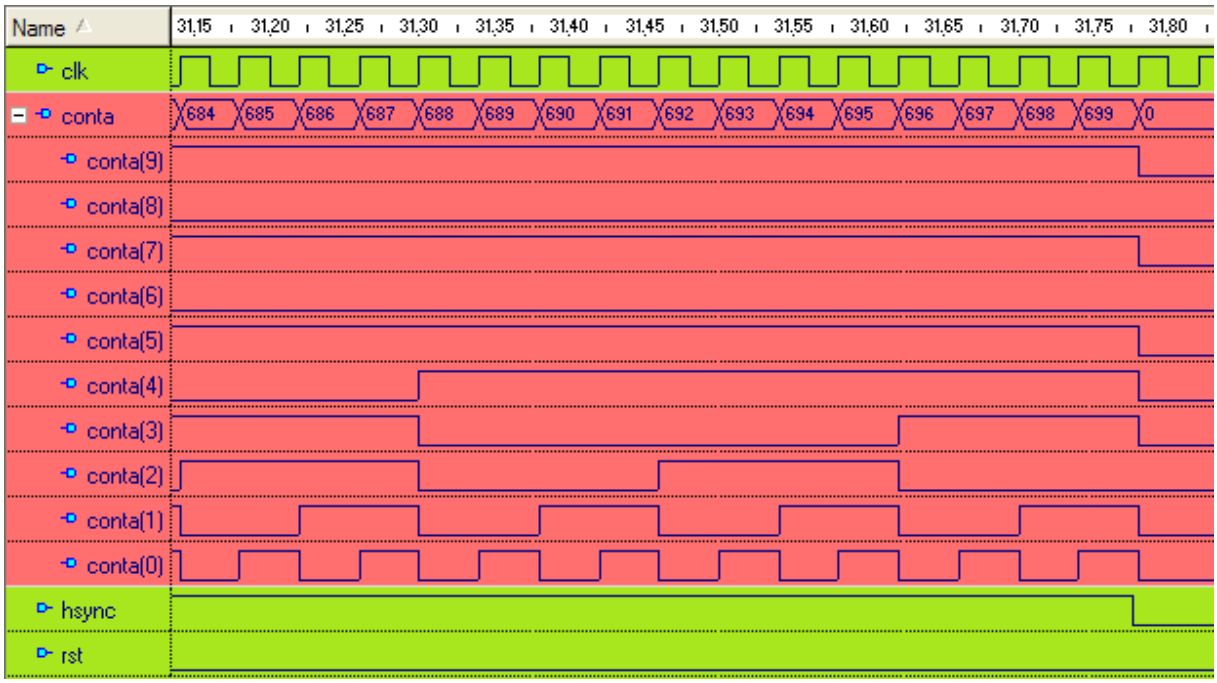


Figura 3.18 Contador de pulsos de reloj (final).

Como información complementaria en la simulación de los contadores, la Figura 3.19 muestra el resultado de la simulación del contador de datos de reloj, pero en un panorama muy amplio, con el fin de poder observar como funciona este contador en el pulso completo de HSync, donde el contador solo aumenta su

valor en el pulso en alto de HSync y es restablecido cada vez que HSync está en un estado bajo.

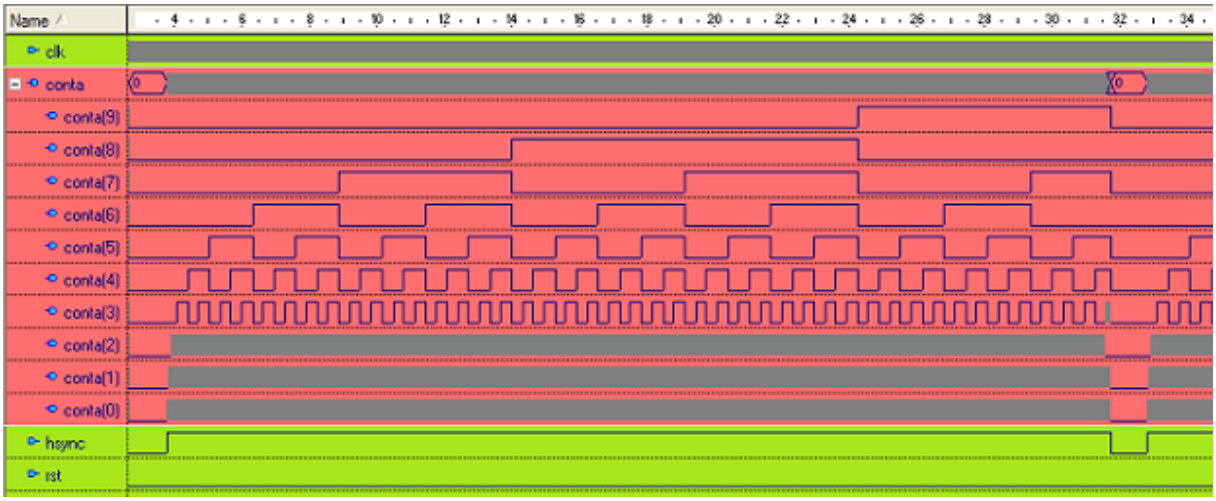


Figura 3.19 Contador de pulsos de reloj (pulso HSync completo).

Para concluir con este módulo de conteo y habilitación, el último componente por definir es el llamado codificador para habilitar flip-flops de 1 bit. Su función es recibir la señal del contador anterior y habilitar, cada uno en el momento adecuado, los 64 flip-flops de un bit, que se encuentran en el módulo de memoria, para que en conjunto almacenen la información de los 64 pixeles que conforman una línea completa de la imagen que se va a desplegar en la pantalla. Esto es posible, porque la salida del contador de pulsos de 25MHz nos indica en que momento está siendo enviado desde VGA cada píxel de cada línea de imagen, por lo tanto, cuando el contador nos indica que está siendo enviado el píxel número 5 de la imagen desde VGA, correspondiente al número 53 de esa línea completa, el codificador habilita el flip-flop número 1 para que se almacene el píxel 1 de la imagen que se va a desplegar en la pantalla. Cuando se recibe el píxel 15 de la imagen desde VGA, correspondiente al número 63 desde VGA, el codificador habilita el flip-flop número 2 para almacenar este nuevo píxel, y continúa de la misma manera hasta almacenar los 64 pixeles de la línea completa que se va a desplegar en la pantalla.

En la Figura 3.20 se observa el resultado de la simulación de este componente, en la imagen se puede ver que el codificador no habilita los flip-flops de 1 bit en el momento en que son enviados desde VGA los pixeles 5, 15, 25, etc.

de la línea de imagen, sino que los habilita 3 pulsos después, este desfasamiento se debe a que el módulo de conversión analógico digital, retrasa la información de los bits en 3 pulsos de reloj, es decir, el FPGA recibe la información del píxel 5 de la imagen cuando la computadora ya está enviando el píxel 8, y lo mismo ocurre en cada línea y en cada píxel. Por lo tanto, para corregir ese desplazamiento el codificador habilita 3 pulsos después de lo debido. Por último, en la Figura 3.21 se muestra el mismo resultado, pero de una vista más general.

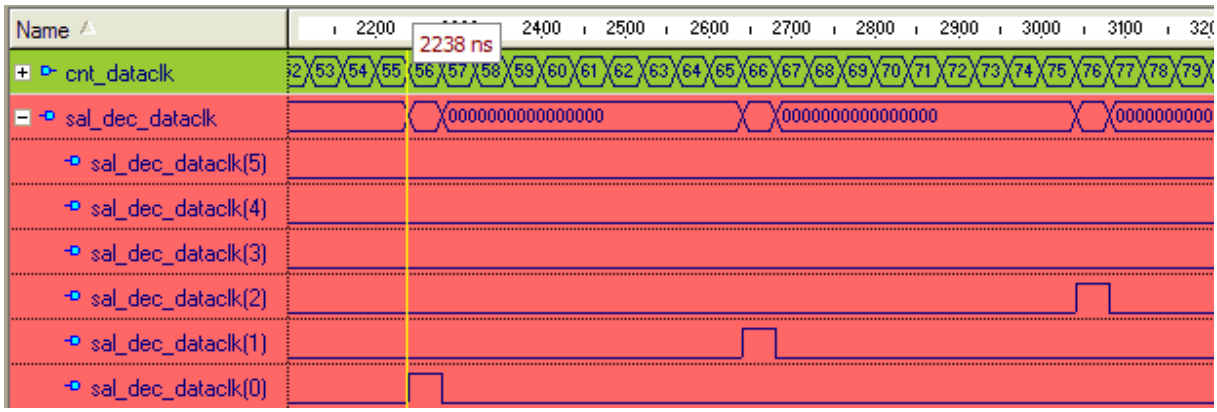


Figura 3.20 Codificador para habilitar flip-flops de 1 bit.

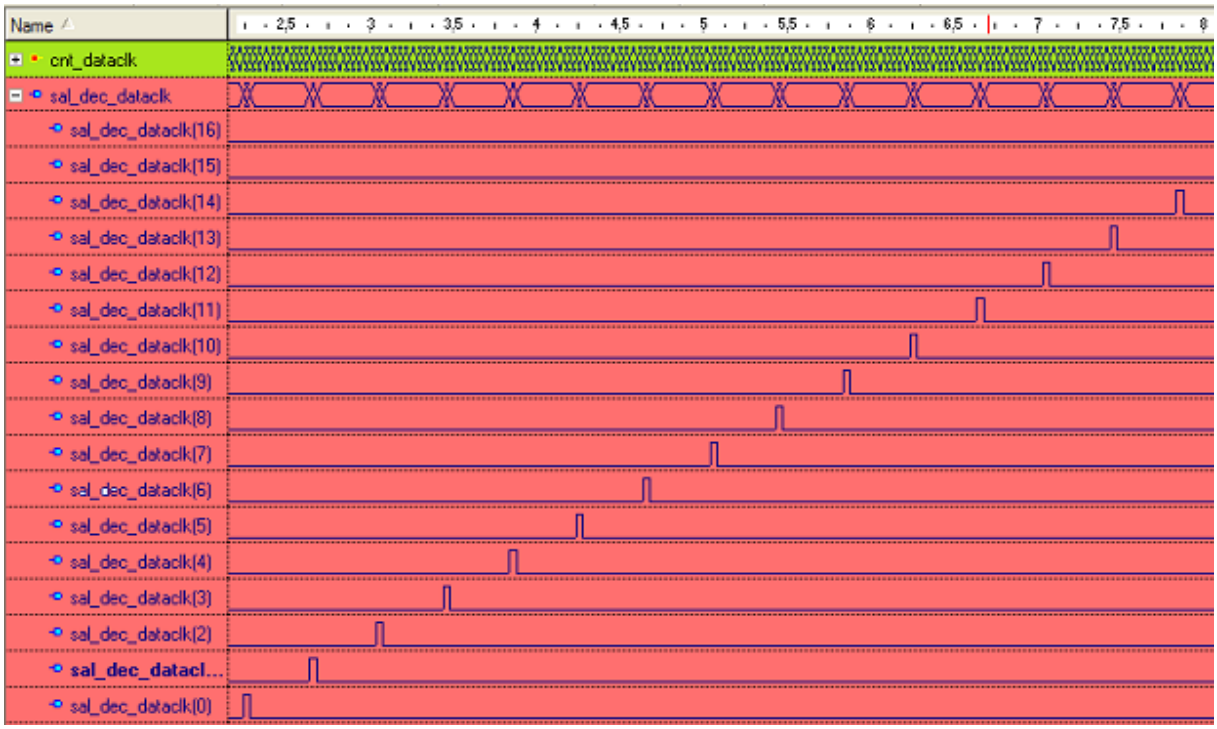


Figura 3.21 Codificador para habilitar flip-flops de 1 bit (panorama más amplio).

Módulo de memoria

Este módulo de memoria recibe la información de la señal de color rojo desde VGA ya digitalizada, y con señales de habilitación del módulo de conteo y habilitación, entrega en su salida la señal de 64 bits que habilita las 64 columnas en la pantalla. Como se observa en la Figura 3.22 este módulo está compuesto por 64 flip-flops de 1 bit y 1 flip-flop de 64 bits.

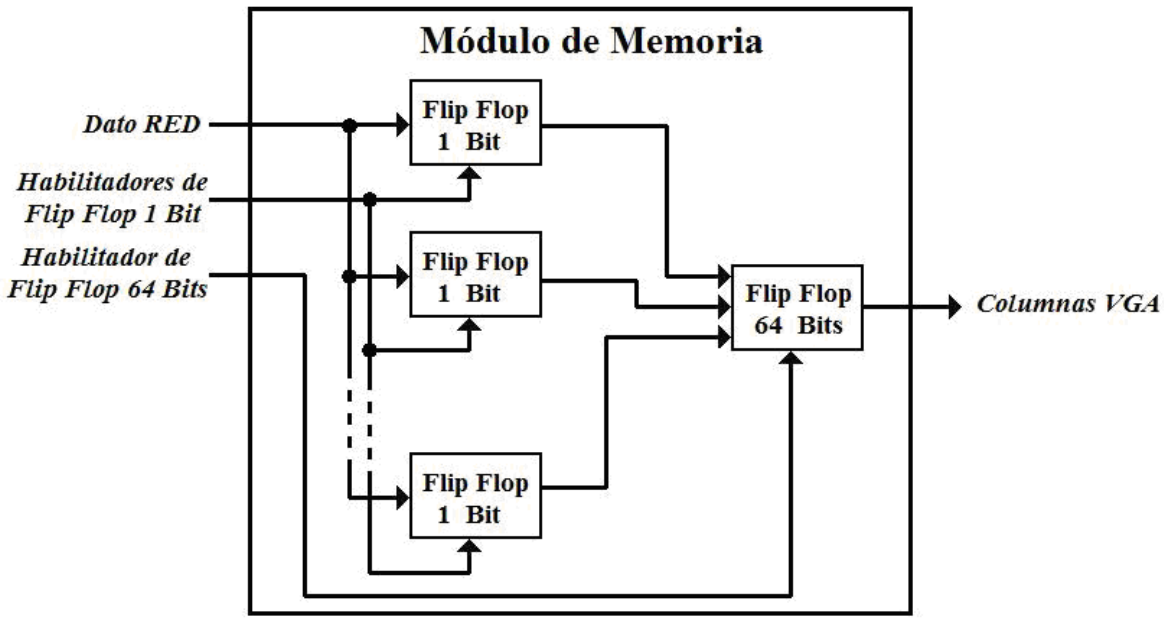


Figura 3.22 Diagrama de bloques del módulo de memoria.

En este módulo, primeramente se encuentran 64 flip-flops de 1 bit, los cuales reciben en todo momento, todos los pixeles digitalizados que son enviados desde la interfaz analógica digital, pero cada uno de los flip-flops solo es habilitado cuando recibe el píxel que le corresponde de cada línea.

Cuando en la señal dato RED se está enviando el píxel 5 de la imagen desde VGA, se habilita el flip-flop 1 para que almacene el valor correspondiente a ese píxel, mientras los demás se encuentran deshabilitados, y no es hasta que se envía el píxel 15 de la imagen cuando se habilita solamente el flip-flop 2, para almacenar el valor correspondiente al segundo píxel que se va a desplegar, pero esto sin afectar el valor almacenado en el flip-flop 1. Siguiendo el mismo procedimiento, cuando VGA finaliza el envío completo de una línea de imagen, en

los flip-flops de 1 bit se encuentran almacenados solo los 64 pixeles de una línea que corresponden al muestreo de la imagen.

En la Figura 3.23 se observa el funcionamiento de cada uno de estos flip-flops, en los cuales, la salida q toma el valor de la entrada d cuando se presenta un pulso en la señal enable, después la salida q mantiene ese valor almacenado aunque la entrada varíe, solo hasta que aparece otro pulso en la señal de enable, la salida q cambia para almacenar el valor que está presente en la entrada d en el momento del nuevo pulso.

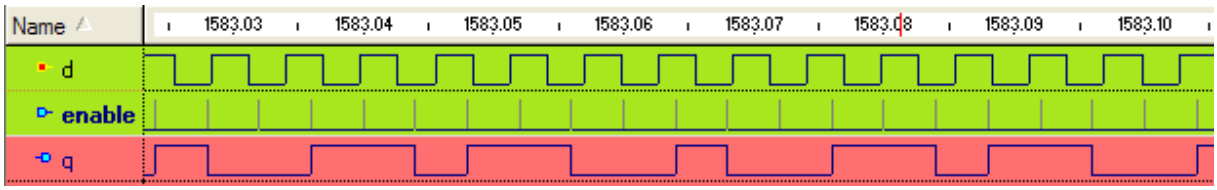


Figura 3.23 Flip-flop de 1 bit.

Los flip-flops de 1 bit son capaces de almacenar toda la línea completa de pixeles a desplegar en la pantalla, pero sus salidas no son usadas para activar las columnas en la pantalla ya que presentan dos inconvenientes: almacenan 64 pixeles de todas las filas, incluyendo las que no serán desplegadas, además los pixeles son almacenados en la forma en que son enviados desde VGA, es decir, de una manera secuencial, por lo que los primeros pixeles están más tiempo almacenados que los últimos, lo que provocaría en la pantalla que el lado izquierdo de la imagen tenga más brillo que la parte derecha.

Para evitar los efectos mencionados, el módulo de memoria cuenta con un flip-flop adicional de 64 bits, el cual es habilitado por el módulo de conteo y habilitación cuando está siendo enviada una fila que se va a desplegar. Por lo tanto, mientras los flip-flop de 1 bit almacenan cada uno los pixeles de la línea, el flip-flop recaba la información de todos en una sola señal de 64 bits y los almacena. Cuando la línea completa que se va a desplegar ha sido enviada, el flip-flop de 64 bit tiene en su salida la señal para activar las columnas de la pantalla, y es deshabilitado por el módulo de conteo y habilitación para que los pixeles de las líneas que no se van a desplegar no afecten los datos

almacenados. El flip-flop mantiene en su salida los 64 pixeles que se despliegan en la pantalla mientras la computadora envía las nueve líneas de imagen que no se van a desplegar, para posteriormente volver a habilitar el flip-flop de 64 bit y almacenar una nueva línea que si se desplegará en la pantalla.

En la Figura 3.24 se muestra la simulación de este componente, se observa que su funcionamiento es similar al de un flip-flop de un bit, difiere solamente en la cantidad de bits que almacena.

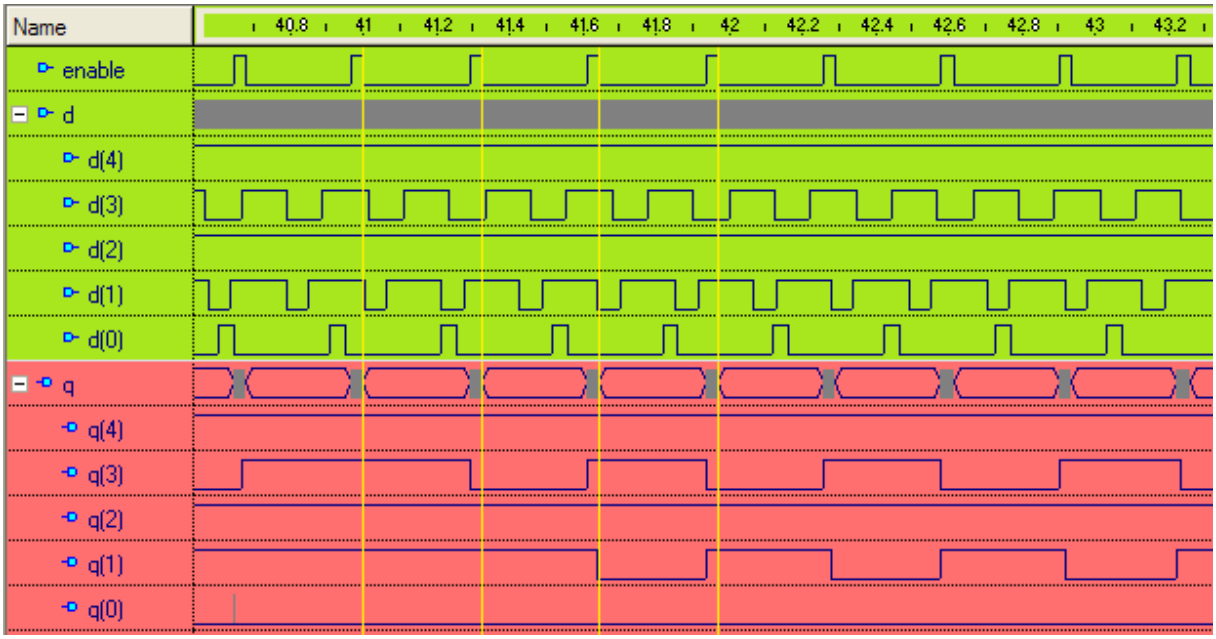


Figura 3.24 Flip-flop de 64 bits.

Módulo generador de pantalla "NO SIGNAL"

Este módulo genera la imagen con el mensaje "NO SIGNAL" ("SIN SEÑAL"), en el formato adecuado para la pantalla, que es desplegado cuando no se detecta señal VGA. En el diagrama de bloques de la Figura 3.25 se muestra que esta función es realizada solo con un contador y dos codificadores.

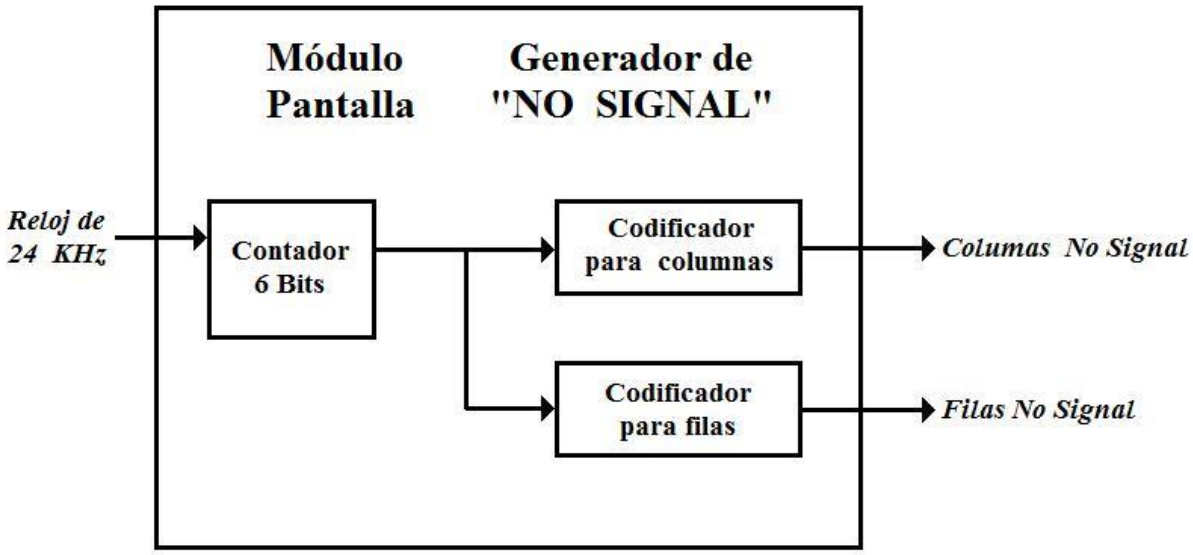


Figura 3.25 Diagrama de bloques del módulo generador de pantalla "NO SIGNAL".

El contador presente en este módulo recibe en su entrada una señal de reloj de 24KHz generada por el divisor de frecuencia y aumenta su salida en 1 bit por cada pulso de reloj. Este contador no tiene señal de reestablecimiento por lo que la cuenta es continúa desde 0 hasta su valor máximo, y posteriormente regresa a 0, en el siguiente pulso de reloj, como se muestra en el resultado de la simulación de la Figura 3.26. Por ser un contador de 6 bits, su salida va desde 0 hasta 63.

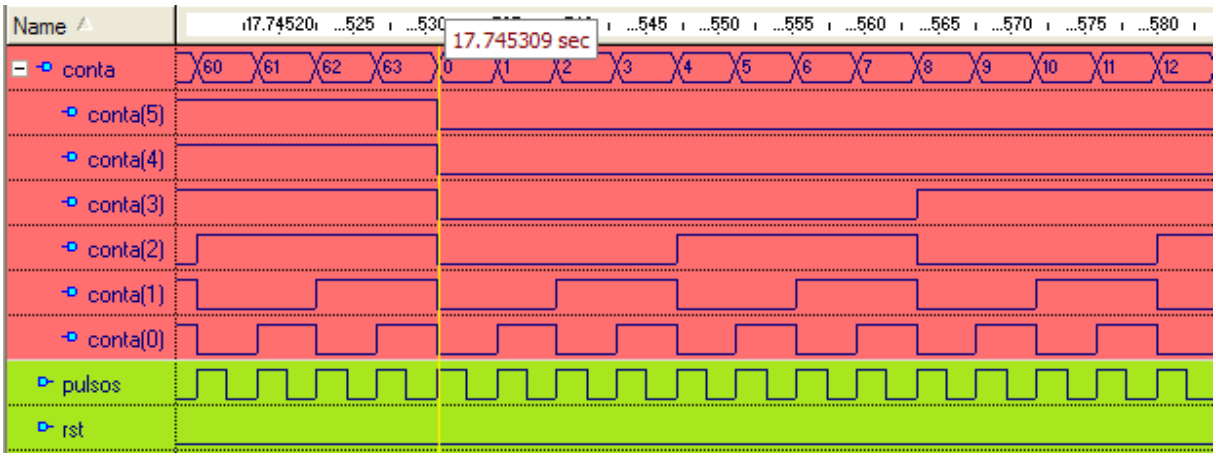


Figura 3.26 Contador para el generador de pantalla "NO SIGNAL".

En cada uno de los 64 posibles valores del contador, que corresponden al mismo número de filas de la pantalla, se generan con ayuda de dos codificadores

las señales que activan las filas y las columnas para desplegar en la pantalla la imagen con el mensaje “NO SIGNAL”.

Para el codificador de columnas se diseñó una imagen en blanco y negro de 64x48 pixeles con el mensaje “NO SIGNAL”, posteriormente la imagen se convirtió en una matriz de unos y ceros. El codificador de columnas retoma esta matriz y en cada valor del contador activa las columnas con la información de cada fila de la matriz con la imagen “NO SIGNAL”, es decir, en el valor 0 del contador, el codificador activa las columnas con la información de la primer fila de la matriz; en el valor 1 del contador, el codificador activa las columnas con la información de la segunda fila de la matriz, y continúa así hasta completar las 64 filas de la matriz. En la Figura 3.27 se muestra el resultado de la simulación de este codificador, y es posible distinguir una parte del mensaje que genera en la pantalla este componente.

Por su parte, el codificador de columnas activa en cada uno de los valores del contador cada una de las filas de la pantalla, es decir, en el valor 0 activa solo la fila 1 de la pantalla, en el valor 1 del contador activa solamente la fila 2 de la pantalla, y así sucesivamente activa las 64 filas como se muestra en los resultados de la Figura 3.28. Con la combinación de ambos codificadores se genera en cada valor del contador las señales para desplegar una fila en la pantalla, y con los 64 valores del contador se despliegan las 64 filas que la conforman a una frecuencia de 24KHz cada fila.

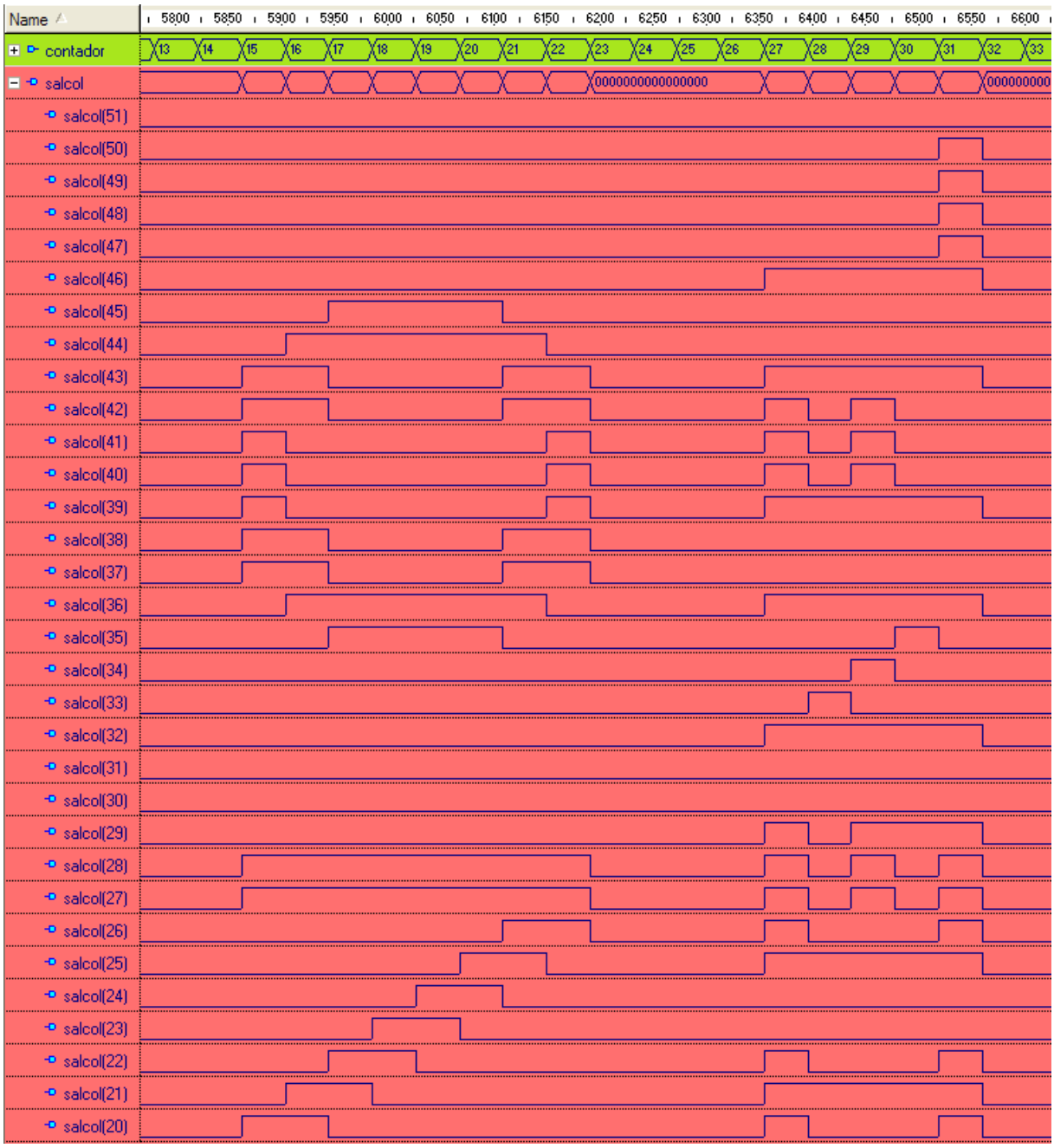


Figura 3.27 Codificador de columnas del generador de pantalla "NO SIGNAL".

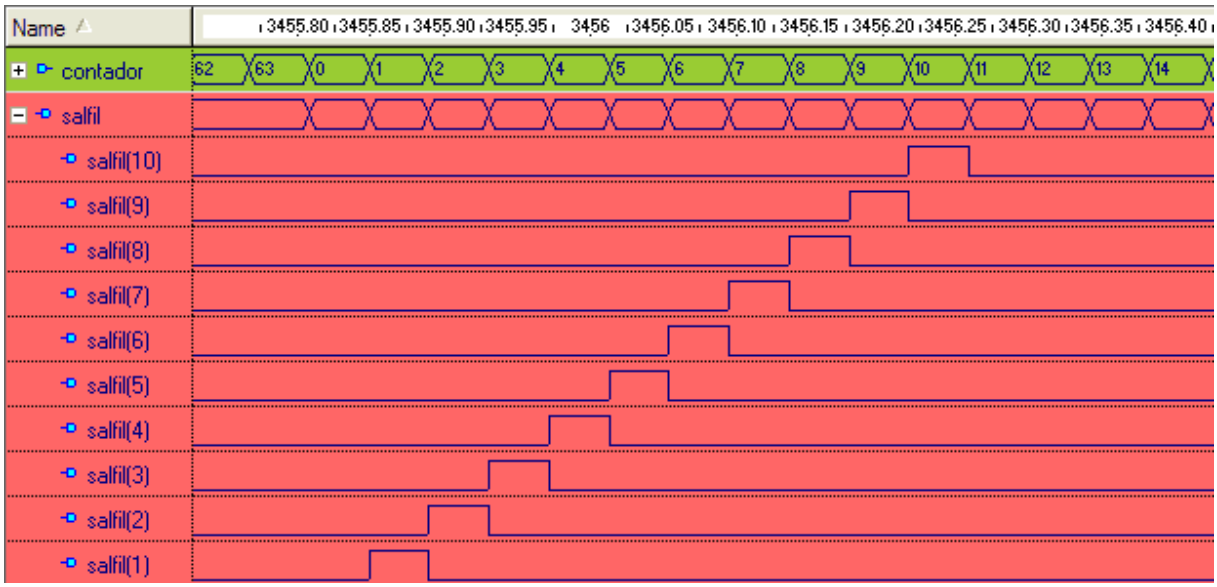


Figura 3.28 Codificador de filas del generador de pantalla "NO SIGNAL".

Módulo de selección

En el diseño se generan señales para desplegar en la pantalla dos imágenes diferentes, unas provenientes de VGA y otras con el mensaje "NO SIGNAL". Este módulo recibe en dos multiplexores, uno para filas y otro para columnas, las señales para generar ambas imágenes, y con un contador y un comparador detecta si está presente la señal VGA para decidir cual de las imágenes desplegar en la pantalla, como se muestra en la Figura 3.29.

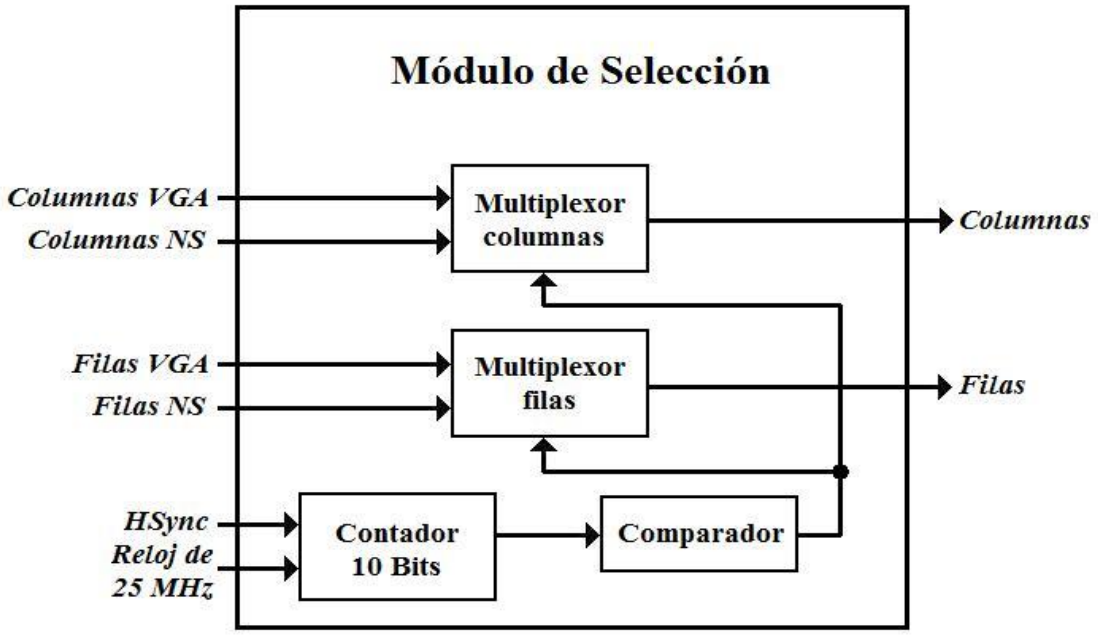


Figura 3.29 Diagrama de bloques del módulo de selección.

El contador de este módulo aumenta su valor en 1 por cada pulso de la señal de reloj de 25MHz y es reestablecido en 0 cuando la señal HSync está en estado alto. Este contador funciona en dos situaciones distintas, primeramente, si no hay presencia de HSync, el contador aumenta desde 0 hasta 1023, por ser de 10 bit, como se muestra en la Figura 3.30.

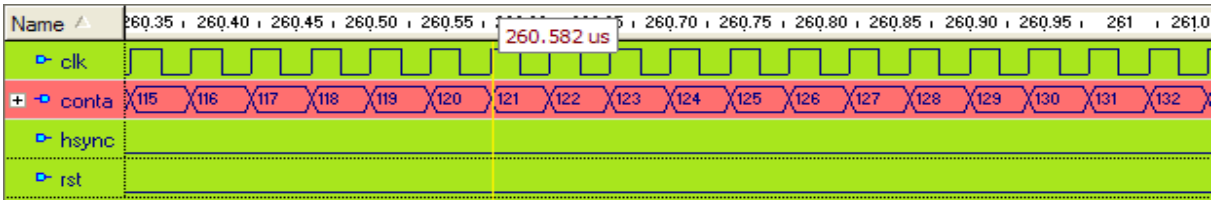


Figura 3.30 Contador sin presencia de HSYNC.

La otra situación en la que puede trabajar este contador es con presencia de la señal HSync, por lo tanto, el contador aumenta solamente cuando HSync está en estado bajo y al llegar a un valor de 79 es reestablecido en 0, como se muestra en la Figura 3.31. Debido a esto, el contador estará la mayor parte del tiempo en 0, lo cual se puede ver en la Figura 3.32, y su valor nunca alcanzará un valor mayor a 79.

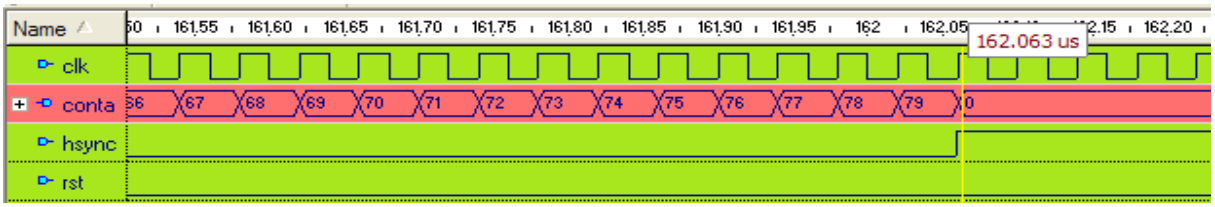


Figura 3.31 Contador en presencia de HSYNC.

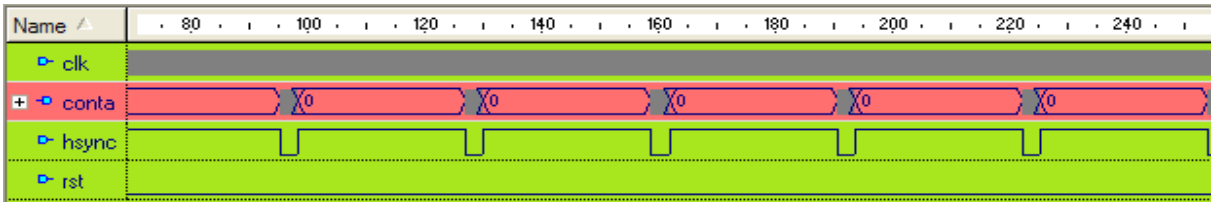


Figura 3.32 Contador en presencia de HSYNC.

El comparador tiene como función tomar el valor del contador y compararlo con un valor constante de 120, si el valor del contador es menor o igual a 120, la salida del comparador mantiene un estado alto, pero si el valor del contador

rebasa 120, la salida del comparador cambia al estado bajo como se muestra en el resultado de la simulación de la Figura 3.33.

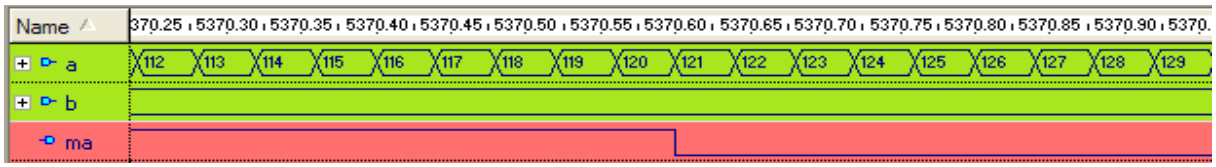


Figura 3.33 Comparador.

Tomando en cuenta las dos posibles situaciones del contador de este módulo, si está presente la señal de VGA, el comparador siempre estará en estado alto, ya que la salida del contador estará en todo momento entre 0 y 79. En el caso en que no esté presente la señal de VGA, el comparador estará la mayor parte del tiempo en estado bajo, ya que entre los valores 0 y 120 del contador, el comparador estará en estado alto, pero desde los valores 121 hasta 1023 del contador, el comparador estará en estado bajo. Debido a esto, la señal de salida del comparador es usada como señal de selección en este módulo, ya que su salida representa la presencia de la señal de VGA, está en 1 cuando hay señal de VGA y está en 0 cuando no hay señal de VGA.

Por último, en este módulo se encuentran dos multiplexores, el primero recibe las señales que activan las columnas para desplegar la imagen desde VGA o para desplegar la imagen con el mensaje de “NO SIGNAL”. Este multiplexor envía a la pantalla únicamente una de las dos señales, para elegir cual enviar usa la señal de salida del comparador, si está en 1, envía a la pantalla la señal de columnas para desplegar la imagen de VGA y si la salida del comparador está en 0, envía a la pantalla la señal de columnas para desplegar la imagen con el mensaje “NO SIGNAL”, un ejemplo del funcionamiento de este multiplexor se encuentra en la Figura 3.34.

El multiplexor de filas realiza la misma función que el de columnas, pero como su nombre lo indica, recibe las señales que activan las filas para desplegar la imagen desde VGA o la imagen con el mensaje de “NO SIGNAL”, así mismo, con la señal de salida del comparador, decide cuál de las dos señales enviar a la pantalla, el resultado de su simulación se encuentra en la Figura 3.35.

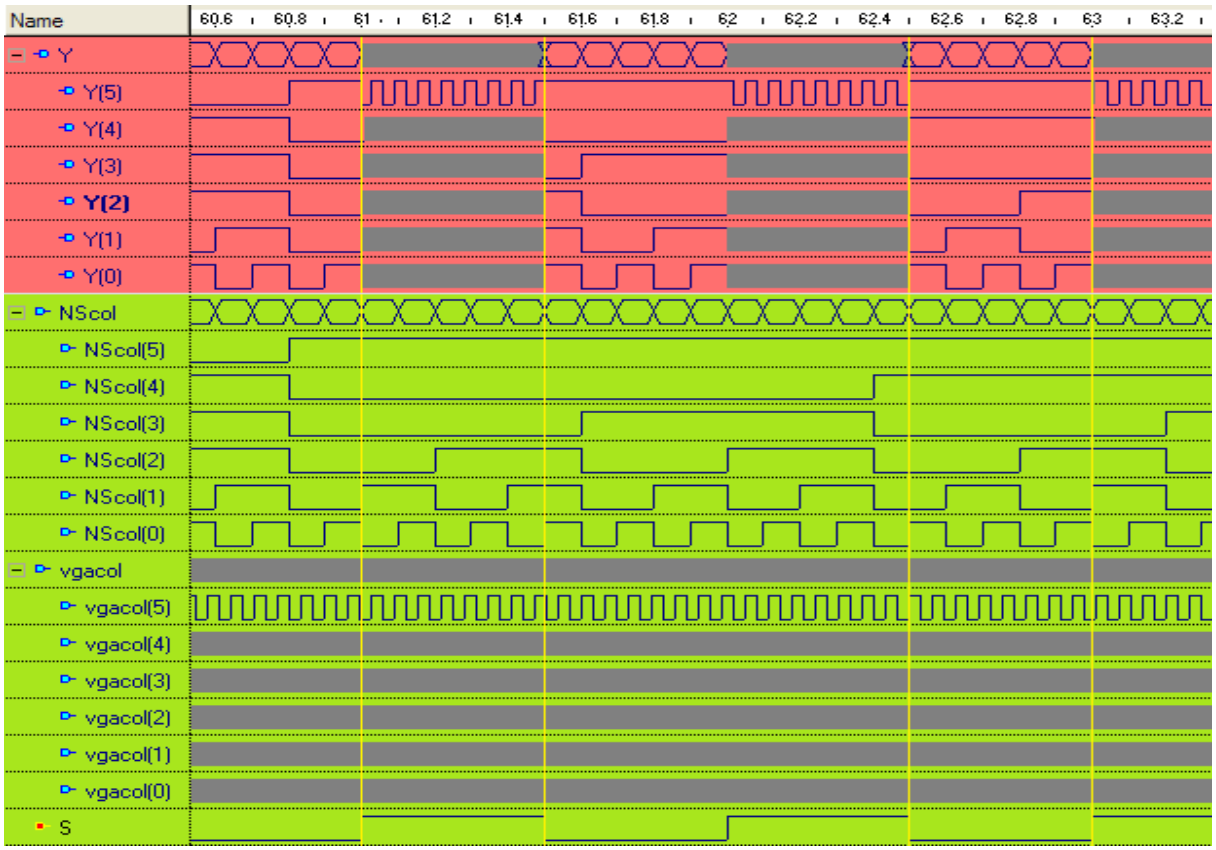


Figura 3.34 Multiplexor para columnas.

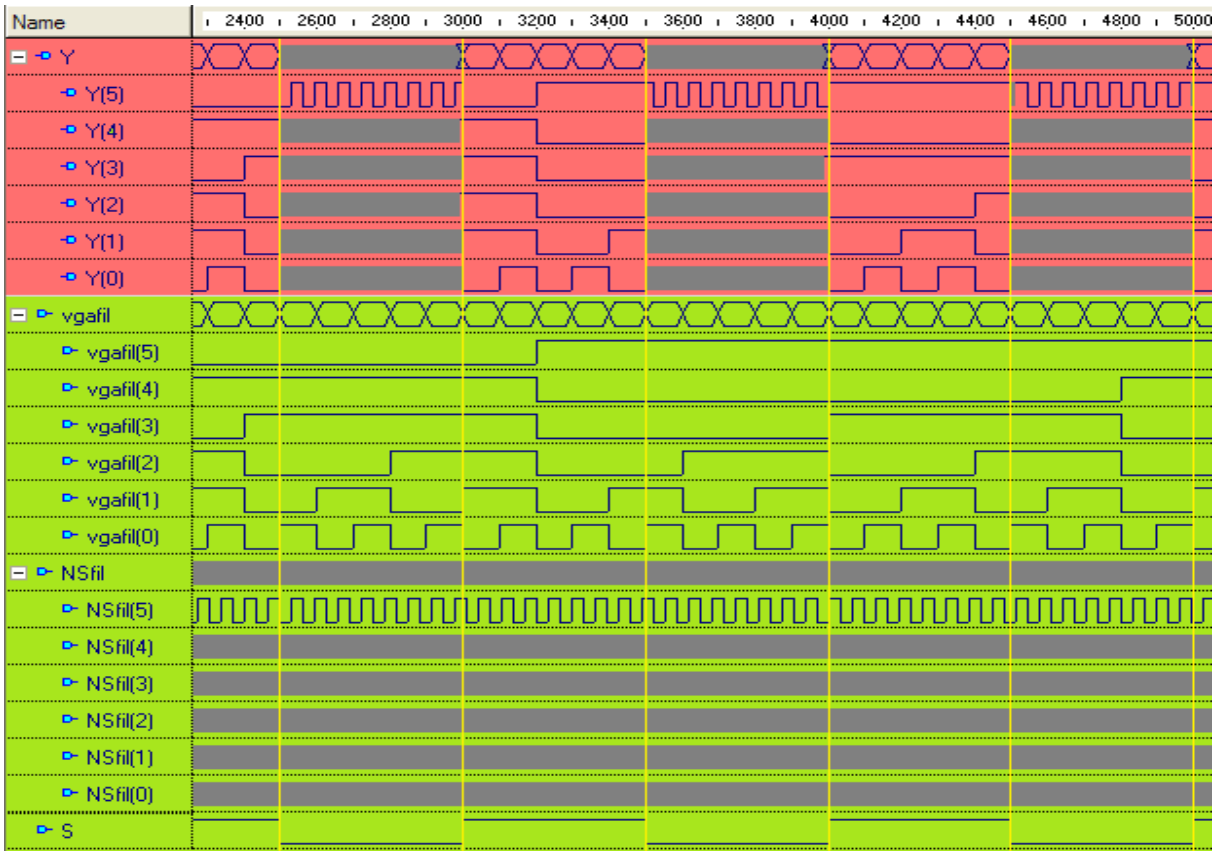


Figura 3.35 Multiplexor para filas.

3.5 CONCLUSIONES

Hemos diseñado y simulado un programa en VHDL para funcionar en un módulo de desarrollo D2SB, capaz de recibir la información de una tarjeta de video en un formato VGA 640X480 a 60Hz, pero con la señal de datos de color rojo ya digitalizada y convertir las señales a un formato de despliegue más apropiado para una pantalla de OLEDs de 64x48.

Con los resultados obtenidos en las simulaciones, se concluye que los componentes por separado funcionan adecuadamente, a pesar de que no es posible simularlos todos juntos, ya que el simulador no acepta diseños que contengan componentes, se espera que en conjunto no presenten problemas.

La funcionalidad de este diseño se basa en que los componentes son unidades muy básicas, contadores, flip-flops, multiplexores y codificadores, como se puede comprobar en los resultados. La diferencia entre algunos de ellos son solo parámetros, pero el código sigue siendo el mismo, por esta razón, los resultados de las simulaciones son sencillos de interpretar.

Los problemas que se presentan en el diseño son que, el generador de pantalla “NO SIGNAL” siempre está en función, pero esto no afecta a las otras partes del programa, el otro problema es que para detectar la presencia de la señal de VGA solo es analizada la señal de HSync y no se toman en cuenta las demás señales, pero por las características principales del diseño están todas cumplidas.

CAPÍTULO 4

MÓDULO DE PANTALLA DE DIODOS EMISORES DE LUZ

4.1 INTRODUCCIÓN

La última parte del proyecto es el despliegue de video en una pantalla de 64x48 OLEDs (Organic Light Emisor Diode – Diodo orgánico emisor de luz). Pero al estar los OLEDs en una etapa experimental se tomó la decisión de elaborar una pantalla de LEDs inorgánicos convencionales, que ya se encuentran disponibles en el mercado, para probar que las dos etapas previas del proyecto funcionen adecuadamente, para posteriormente sustituir la pantalla de LEDs inorgánicos por una pantalla de OLEDs.

Habiendo incluido un nuevo componente al proyecto, los LEDs, es necesario tener una idea de cómo funciona un LED [4]. Un LED o Diodo Emisor de Luz es un dispositivo semiconductor que emite luz monocromática, es decir, con un espectro muy angosto cuando es polarizado de forma directa, y se le inyecta una corriente eléctrica.

El color o longitud de onda que emite el LED, depende del material semiconductor empleado en la construcción del diodo, pudiendo variar desde el ultravioleta hasta el infrarrojo.

Como se mencionó, los LEDs son básicamente pequeños diodos que producen luz cuando una corriente eléctrica pasa a través del material semiconductor del que están hechos, pero a diferencia de una bombilla eléctrica convencional, éstos no tienen una resistencia que pueda romperse o quemarse, lo cual los hace muy durables y confiables, además no producen calor, lo que les da una mayor eficiencia.

Un diodo es el dispositivo semiconductor más simple que hay, ya que básicamente es un material semiconductor, hecho de un material de conducción pobre al que le han agregado “impurezas”, este proceso se conoce como “dopaje”, estas impurezas no son más que átomos de otro elemento el cual modifica las propiedades de conducción del material.

El funcionamiento físico consiste en que, un electrón pasa de la banda de conducción a la de valencia, perdiendo energía. Esta energía se manifiesta en forma de un fotón desprendido con una amplitud, una dirección y una fase aleatoria.

Para encender un LED es necesario polarizarlo directamente, es decir, tienen un polo positivo y uno negativo. Además poseen un voltaje mínimo de polarización, el cual está relacionado con el semiconductor del cual están fabricados, si este voltaje no es superado, el LED no enciende.

Por último, en un LED, el dispositivo semiconductor está comúnmente encapsulado en una cubierta de plástico de mayor resistencia que las de vidrio que usualmente se emplean en las lámparas incandescentes. Aunque el plástico puede estar coloreado, es sólo por razones estéticas, ya que ello no influye en el color de la luz emitida. Usualmente un LED es una fuente de luz compuesta con diferentes partes, razón por la cual el patrón de intensidad de la luz emitida puede ser bastante complejo.

4.2 DISEÑO DE LA PANTALLA

Para el diseño de la pantalla se debe tener muy claro el formato en que el FPGA envía las imágenes, en el cual se deben activar en un mismo momento, una fila completa y las respectivas columnas que corresponden a los pixeles del renglón de la imagen que se está desplegando. Con esta información se procede a diseñar una tarjeta de circuito impreso donde será montada la matriz de 64x48 LEDs.

Para el diseño de la tarjeta de circuito impreso se utilizó el programa OrCAD Layout, el mismo que fue usado para diseñar la interfaz analógica digital. Para facilitar la construcción de la pantalla, fue adquirido un encapsulado, con referencia LPT-757G [8], que contiene una matriz de 5x7 LEDs.

En este encapsulado cada LED está hecho en base a GaP, es decir, fosfuro de galio, el cual ofrece una emisión de luz color verde, cuya longitud de onda corresponde a 565nm. Además es importante conocer la disipación promedio de cada LED, que corresponde a 32mW. En la Figura 4.1 [8] se muestra el diagrama del circuito interno del empaquetado.

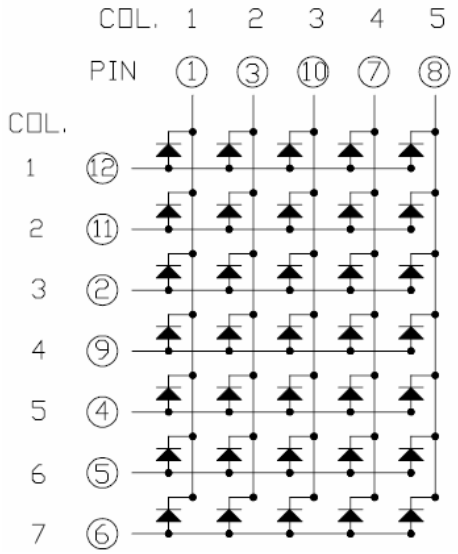


Figura 4.1 Diagrama interno de matriz de 5x7 LEDs.

La forma de encender cada LED es mediante una combinación de patillas, es decir, para prender un LED en particular, es necesario polarizar de manera directa su patilla de columna y su patilla de fila a la vez. Dichas conexiones se muestran en la siguiente tabla:

Numero de Patilla	Conexión
1	Cátodo Columna 1
2	Ánodo Fila 3
3	Cátodo Columna 2

4	Ánodo	Fila 5
5	Ánodo	Fila 6
6	Ánodo	Fila 7
7	Cátodo	Columna 4
8	Cátodo	Columna 5
9	Ánodo	Fila 4
10	Cátodo	Columna 3
11	Ánodo	Fila 2
12	Ánodo	Fila 1

Con el componente seleccionado, el primer paso fue diseñar el footprint adecuado para el encapsulado. Posteriormente, fueron contempladas dos configuraciones para interconectar la matriz de LEDs, las cuales se muestran en la Figura 4.2.

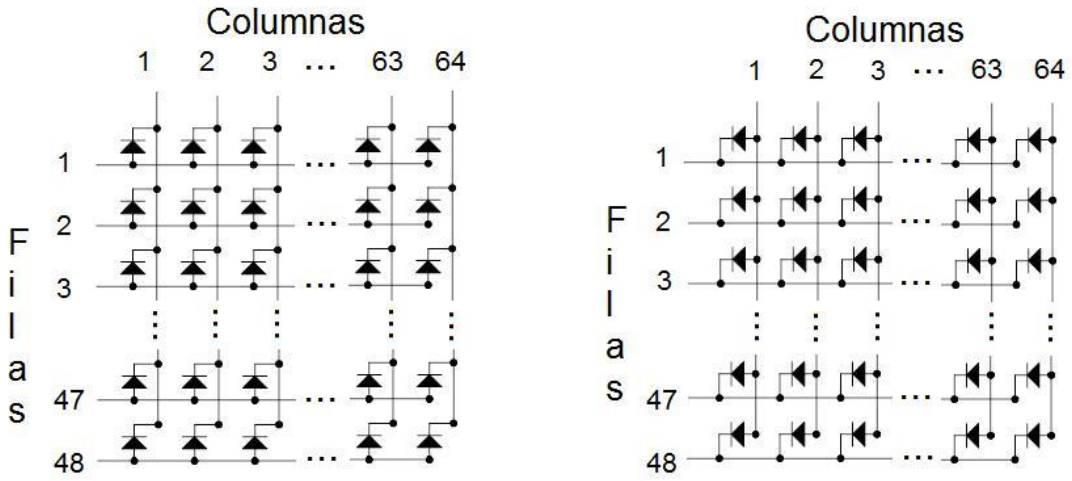


Figura 4.2 Configuraciones posibles para la elaboración de la matriz de LEDs.

En ambas configuraciones, se mantiene la forma de encender un LED en particular, polarizar de manera directa la fila y la columna a la cual pertenecen. La diferencia es que en la primera configuración, el cátodo está asociado a las columnas, y el ánodo a las filas, en cambio, en la segunda configuración ocurre lo contrario.

Fue elegida la primera configuración con el objetivo de disminuir la cantidad de LEDs inactivos en la pantalla. En la primera configuración se forma la matriz con bloques de 5x7, por lo tanto, son usados 13x7 encapsulados, lo que hace un total de 65x49 LEDs, y solo queda una fila y una columna inactiva en la pantalla y son usados 91 encapsulados. Por otra parte, la segunda configuración hace que la matriz sea completada con bloques de 7x5, lo que requiere 10x10 encapsulados, es decir, un total de 70x50 LEDs, de los cuales 6 columnas y dos filas quedan inactivas, y son usados un total de 100 encapsulados.

Posteriormente, fue elaborado en Layout un diseño de una tarjeta de circuito impreso, donde se interconectaron las filas y las columnas de 91 footprint, correspondientes al encapsulado de la matriz de LEDs de 5x7, de manera que se formó una matriz de mucho mayor tamaño, de 65x49 LEDs, con la configuración correspondiente a la primera matriz de la Figura 4.2. Además de los footprint del encapsulado LPT-757G, fueron añadidos al diseño los footprints de 5 conectores machos de 40 patillas, 3 para conectar al exterior de la tarjeta las 64 columnas y los otros 2 para conectar, también al exterior de la tarjeta, las 48 filas.

Después de la elaboración del diseño en OrCAD Layout, se fabricó la tarjeta, con el procedimiento ya explicado para la interfaz analógica digital, y se montaron los componentes en sus respectivas posiciones, con lo cual se finalizó el diseño y la elaboración de la pantalla donde serán desplegadas las imágenes de video enviadas por el FPGA.

4.3 INTERFAZ ANALÓGICA

Para desplegar imágenes en la pantalla fabricada, es necesaria la alimentación de corriente desde una fuente externa, ya que con un pequeño análisis, es posible comprender que no es suficiente la alimentación que provee una patilla de salida del FPGA.

En la pantalla, la corriente máxima que puede ser requerida corresponde al momento en que en una línea se enciendan todos sus LEDs. Por la hoja de

especificaciones LPT-757G se puede conocer que cada LED se alimenta en promedio con 11mA, por lo tanto, la máxima corriente que es requerida por una fila es de 700mA aproximadamente.

Por la forma en que son desplegadas las imágenes por el FPGA, una patilla de éste se encarga de alimentar una fila completa, pero una patilla no provee la corriente suficiente para encender los 64 LEDs, además, el riesgo de dañar el dispositivo, si esto llegara a intentarse, es muy grande.

Para poder cubrir los requerimientos de corriente de la pantalla, y eliminar los riesgos de daños al FPGA, se diseñó un sistema de interruptores usando transistores, este sistema que fue agregado al proyecto, es llamado interfaz analógica, y su esquemático se muestra en la Figura 4.3. En la imagen se puede observar que los transistores actúan como interruptores, y el FPGA solo provee una pequeña corriente para activar estos interruptores.

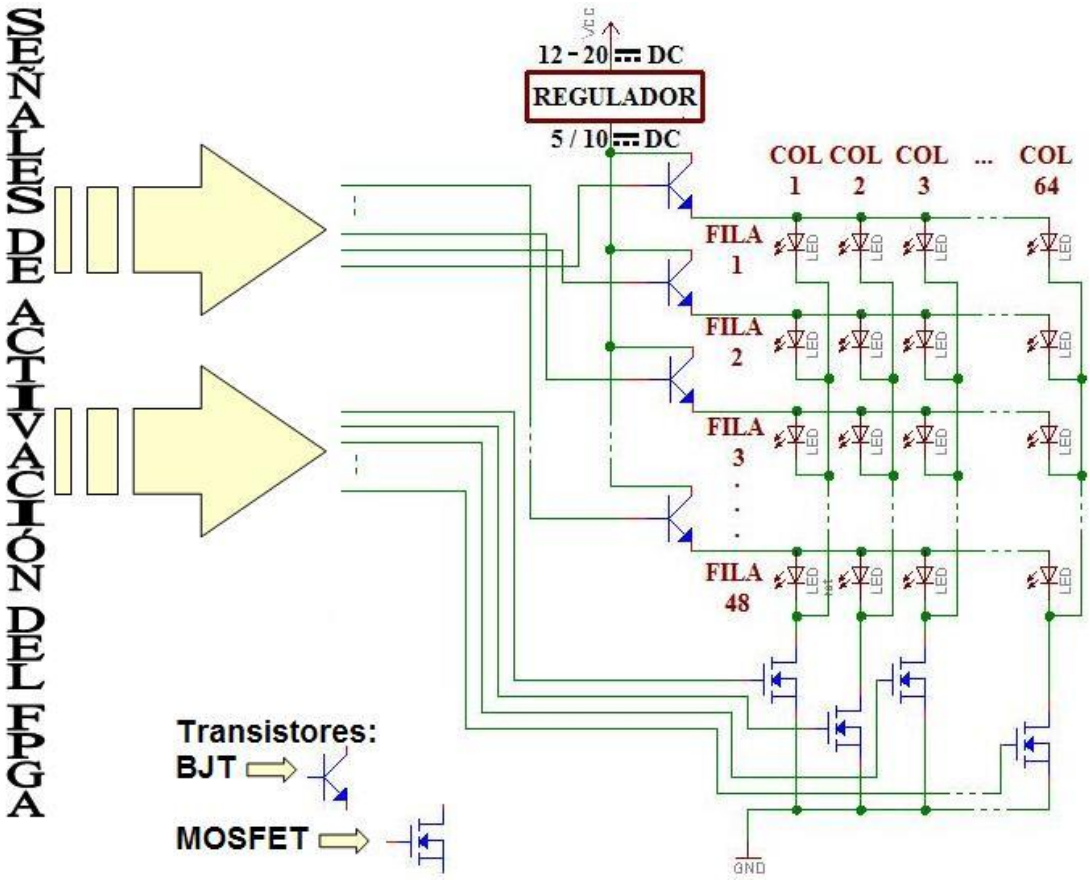


Figura 4.3 Esquemático del circuito de la tercera etapa.

Con este nuevo diseño, el encargado de proveer la corriente necesaria para encender una fila completa es ahora un transistor, por lo tanto, para esta función se eligió el transistor BJT 2N2222A, ya que entre sus características, está la de soportar los 700mA que puede llegar a requerir la pantalla. En este punto, es necesario hacer una pausa y explicar un poco acerca de los transistores BJT, así como hacer una breve descripción del transistor que será usado para alimentar las filas de LEDs en la pantalla.

4.3.1 Transistores BJT

Un transistor de unión bipolar o BJT [11], se fabrica básicamente sobre un monocristal de Germanio, Silicio o Arseniuro de Galio, que tienen cualidades de semiconductores, estado intermedio entre conductores y aislantes. Sobre el sustrato de cristal se dopa en forma muy controlada tres zonas, dos de las cuales son del mismo tipo, es decir, NPN o PNP, por lo tanto quedan dos uniones NP.

La zona N con elementos donantes de electrones o cargas negativas, y la zona P de huecos o cargas positivas. Los elementos mas utilizados en la zona P son Indio (In), Aluminio (Al) o Galio (Ga) y donantes N al Arsénico (As) o Fósforo (P).

La configuración de uniones PN, dan como resultado transistores PNP o NPN, donde la letra intermedia siempre corresponde a la característica de la base, y las otras dos al emisor y al colector que, si bien son del mismo tipo y de signo contrario a la base, tienen diferente contaminación entre ellas. En la Figura 4.4 [9], se puede observar la estructura básica de este tipo de transistores.

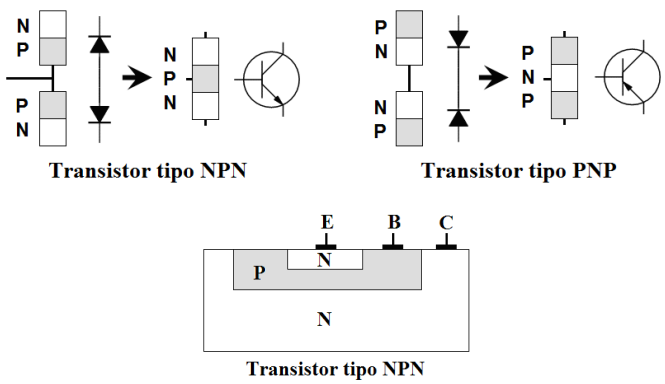


Figura 4.4 Estructura básica de transistores BJT.

Este tipo de transistores tiene tres estados o regiones de operación: región de corte, región de saturación y región activa.

Región de Corte

Esta se produce cuando la corriente del colector es igual a la del emisor. Este caso normalmente se presenta cuando la corriente de base es igual a cero $I_b=0$.

Región de Saturación

El transistor esta saturado cuando la corriente del colector es igual a la corriente del emisor y a su corriente máxima, es decir, $I_c=I_e=I_{máxima}$. Este caso se presenta normalmente cuando la corriente de base es lo suficientemente grande como para inducir una corriente de colector es β veces más grande, es decir, $I_c=\beta x I_b$.

Región Activa

Cuando el transistor no se encuentra en ninguna de las regiones o estados anteriores, este se encuentra en una región intermedia, la cual es la región activa. En esta región la corriente del colector depende principalmente de la corriente de la base, la ganancia de corriente de un amplificador (es decir, β) y de las resistencias que puedan estar conectadas en el colector y en el emisor. Cada destacar, que esta región es la mas importante cuando queremos utilizar nuestro transistor como un amplificador.

En el proyecto, los transistores que son usados solo funcionan en los estados de corte o de saturación, es decir, funcionan como interruptores que activan o desactivan una línea completa de la pantalla, en ningún momento trabajan en la región activa.

4.3.2 Transistor BJT 2N2222A

El transistor 2N2222A [12] está diseñado para trabajar a alta velocidad y con una corriente de colector mayor a 500mA. Esta característica es útil para la ganancia de corriente sobre un amplio rango de corriente en el colector, poca fuga de corriente y bajo voltaje de saturación.

Las características más importantes, por las cuales se considero utilizar este transistor fueron las siguientes:

Voltaje máximo Colector-Base…….75V

Voltaje máximo Colector-Emisor…….40V

Voltaje máximo Emisor-Base…….6V

Corriente máxima de Colector…….0.8A

Tiempo de Transición……..……..300MHz

Tiempo de Subida……..……..25nS

En la Figura 4.5 [12] se puede observar el diagrama esquemático interno y el modelo físico del transistor adquirido para la pantalla.

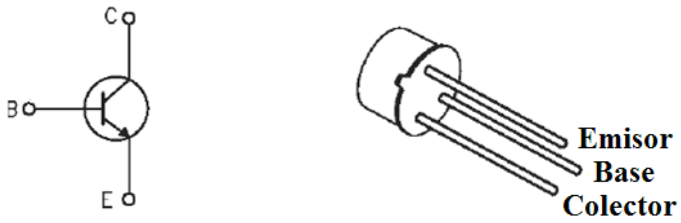


Figura 4.5 Diagrama esquemático interno y modelo físico del transistor BJT 2N2222A.

Por último, se observa en las características del transistor BJT 2N2222A, que cumple con los requisitos de voltaje, y lo más importante, de corriente, ya que soporta hasta 800mA, lo cual es suficiente para alimentar 64 LEDs en un mismo momento.

Por otro lado, los transistores que activan las columnas solo encienden un LED a la vez, por lo tanto, no es necesario que soporten demasiada corriente. Debido a esto, para activar las columnas se eligió un transistor tipo MOSFET, cuyo encapsulado corresponde al número de referencia es SD5400CY, de los cuales se hace una breve descripción en la siguiente sección.

4.3.3 Transistores MOSFET

El transistor MOSFET [11] es fabricado a base de sustrato de material semiconductor dopado mediante técnicas de difusión de dopantes. Se crean dos

islas de tipo opuesto separadas por un área sobre la cual se hace crecer una capa de dieléctrico culminada por una capa de conductor. Los transistores MOSFET se dividen en dos tipos fundamentales, los cuales se muestran en la Figura 4.6 [9], dependiendo de cómo se haya realizado el dopaje:

- Tipo nMOS: Sustrato de tipo p y difusiones de tipo n.
- Tipo pMOS: Sustrato de tipo n y difusiones de tipo p.

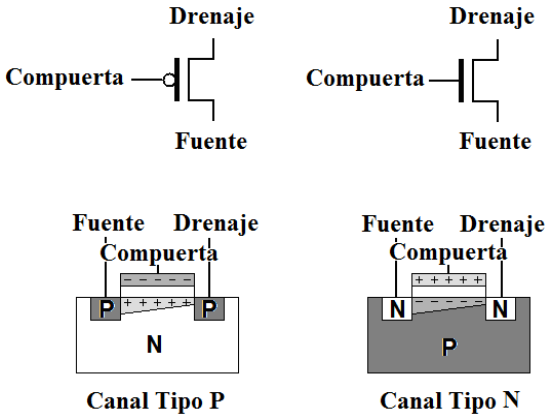


Figura 4.6 Tipos de transistores MOSFET.

Las áreas de difusión se denominan fuente y drenaje, y el conductor entre ellos es llamado compuerta.

El transistor MOSFET tiene tres estados de funcionamiento:

Estado de corte

Cuando el voltaje de la compuerta y la del sustrato son iguales, el MOSFET está en estado de no conducción, es decir, ninguna corriente fluye entre fuente y drenaje aunque se aplique una diferencia de potencial entre ambos.

Conducción lineal

Al polarizarse la compuerta con un voltaje negativo para uno de tipo pMOS o un voltaje positivo para uno de tipo nMOS, se crea una región de deplexión en la región que separa la fuente y el drenaje. Si este voltaje crece lo suficiente, se crea un canal de conducción en la región de deplexión mencionada anteriormente. Esto debido a que aparecerán portadores minoritarios, es decir,

electrones en nMOS y huecos en pMOS. El transistor pasa entonces a un estado de conducción, de modo que una diferencia de potencial entre la fuente y el drenaje dará lugar a una corriente. El transistor se comporta como una resistencia controlada por el voltaje de compuerta.

Saturación

Cuando el voltaje entre el drenaje y la fuente supera cierto límite, el canal de conducción bajo la compuerta sufre un estrangulamiento en las cercanías del drenaje y desaparece. La corriente entre fuente y drenaje no se interrumpe, ya que se debe al campo eléctrico entre ambos, pero se hace independiente de la diferencia de potencial entre ambos terminales.

Al igual que el transistor BJT, el transistor MOSFET solo recibe una señal digital de 0 ó de 3.3V que proviene del FPGA, por lo tanto el transistor solo estará funcionando en el estado de corte o de saturación, es decir, el transistor MOSFET solo trabajará como un interruptor que habilita o deshabilita una columna, y nunca estará trabajando en la región de conducción lineal.

4.3.4 Transistor MOSFET SD5400CY

La serie SD5000 [13] cuenta con cuatro arreglos monolíticos de cuatro interruptores bidireccionales de alto desempeño, como se muestra en la Figura 4.7 [13], los cuales son fabricados para alta velocidad, alto voltaje y resistencia baja de doble difusión MOS (DMOS). Con un umbral máximo de 2V, permite usar TTL y CMOS simples en aplicaciones de señales pequeñas.

Las características por lo que fue considerado este transistor son:

Señal Analógica…….....……..... ±10V
Voltaje máximo Drenaje-Fuente……20V
Voltaje máximo Drenaje-Sustrato……..25V
Voltaje máximo Fuente-Sustrato……25V
Voltaje máximo Compuerta-Fuente……...30 / -25V
Voltaje máximo Compuerta-Sustrato……30 / -0.3V
Voltaje máximo Compuerta-Drenaje…….30 / -25V
Corriente máxima de Drenaje……..50mA

Tiempo de Encendido……..……..1nS

Tiempo de Subida……..……1nS

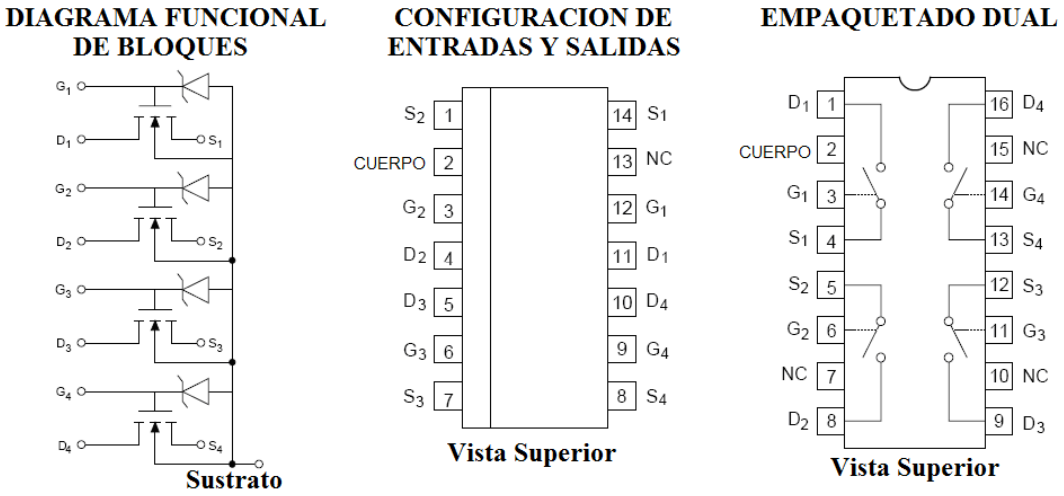


Figura 4.7 Diagramas de transistor MOSFET SD5400CY.

Como se puede observar, los transistores SD5400CY cumplen con los requerimientos de tiempo de encendido y soporta el suficiente voltaje. Además que su empaquetado cuenta con 4 transistores en un mismo componente, lo que reduce el tamaño de las tarjetas de circuito impreso, ya que por los 64 transistores MOSFET que se requieren, se necesitan solo 16 circuitos integrados.

4.4 DISEÑO DE LA INTERFAZ ANALÓGICA

Para la fabricación de las tarjetas de circuito impreso que contienen los transistores, se realiza el mismo proceso que con la interfaz analógica digital y la pantalla de LEDs. El programa en el cual se realizan los diseños es OrCAD Layout, en este caso, no es necesario elaborar footprints, ya que las librerías de Layout ya contienen footprints adecuados para los componentes que son utilizados.

Las tarjetas de esta etapa deben conectarse directamente a los puertos con que cuenta la tarjeta de desarrollo D2SB. Estos puertos tienen conectores de 40 entradas, como los que se muestran en la Figura 4.8 [7]. Pero en cada

conector, no todas las entradas son funcionales, ya que existen algunas entradas que están desconectadas del FPGA Spartan 2E.

Figura 4.8 Diagrama de conector de 40 entradas de la tarjeta SPARTAN 2E.

Los puertos en la tarjeta Spartan 2E se encuentran distribuidos en pares en tres lados de la tarjeta, por lo que forman un total de seis, como se muestra en el diagrama de la Figura 4.9 [7], y son nombrados A1, A2, B1, B2, C1 y C2.

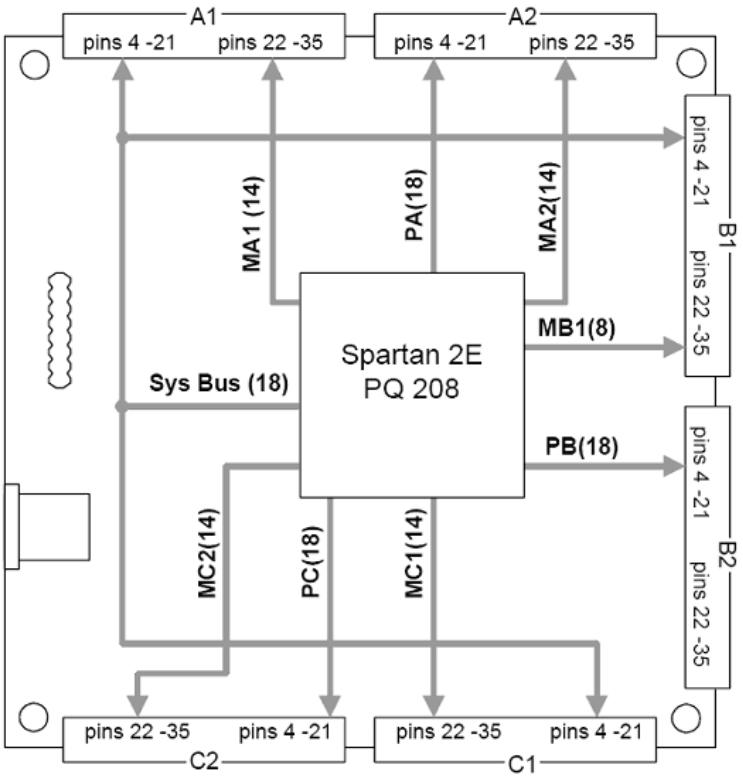


Figura 4.9 Diagrama SPARTAN 2E.

En el diagrama, se observa también la distribución de las entradas que son funcionales, es decir, que sí están conectadas al FPGA. El puerto A1 cuenta con 14 entradas propis y 18 compartidas con B1 y C1; el puerto A2 cuenta con 32 entradas propias; el puerto B1 cuenta con 18 entradas propias y 18 compartidas con A1 y C1; el puerto B2 cuenta con 18 entradas propias; el puerto C1 cuenta

con 14 entradas propios y 18 compartidas con A1 y B1; y por último, el puerto C2 cuenta con 32 entradas propios.

En base al diagrama de la Figura 4.9 [7], fueron distribuidos los 6 puertos de la tarjeta D2SB de la siguiente manera: Las 46 entradas propios que suman los puertos A1 y A2, más 2 entradas compartidos de A1, en total 48, fueron destinados para activar las filas de la pantalla; las entradas propias de los puertos B2, C1 y C2, en total 64, se destinaron para activar las columnas de la pantalla; y por último, el puerto B1 fue destinado para conectar el Convertidor Analógico Digital, que ya fue definido en el capítulo 1.

Posteriormente, fueron diseñadas en Layout 4 tarjetas de circuitos impresos para la interfaz analógica, cada una con sus características particulares. La primera tarjeta contiene todos los transistores BJT 2N2222A, que activan las 48 filas, y cuatro puertos. Dos puertos funcionan como entradas que se conectan directamente a través de los conectores de 40 patillas a la tarjeta D2SB, en los puertos A1 y A2. Los otros dos puertos serán de salida que se conectan a través de cable plano a la pantalla de LEDs.

La segunda tarjeta de circuito impresa contiene 5 circuitos integrados SD5400CY cada uno con 4 transistores MOSFET, de los 20 que suman en total se usan solo 18 para activar 18 de las 64 columnas que conforman la pantalla. Esta tarjeta fue diseñada con un puerto de entrada que contendrá un conector de 40 patillas, para ser conectado directamente al puerto B2 de la tarjeta D2SB y un puerto más de salida, para conectarse a través de cable plano a la pantalla de LEDs.

Después se diseñó la tercera tarjeta, la cual requiere activar las 46 columnas que restan de la pantalla, por lo que contiene 12 circuitos integrados SD5400CY, lo que da un total de 48 transistores MOSFET. Esta tarjeta se conecta a los puertos C1 y C2 de la tarjeta D2SB, por lo que tiene por entradas dos conectores de 40 patillas, y por salidas conectores para cable plano, que son conectados a la pantalla de LEDs.

Por último, como caso adicional se diseñó un regulador de voltaje capaz de regular una entrada que puede ser desde 12 hasta 20V de voltaje directo y convertirlo en una salida muy estable de 5 ó 10V de voltaje directo. Este regulador se diseñó con la finalidad de adaptar una fuente de voltaje regulado de 12V al proyecto y con solo manipular un jumper, poder controlar el voltaje de entrada a la interfaz analógica, para que ésta sea de 5 ó 10V.

Posteriormente, después del diseño de las 4 tarjetas correspondientes a la interfaz analógica en OrCAD Layout, se procedió a la fabricación y posteriormente al montaje de los componentes. El proceso de fabricación completo de tarjetas de circuito impreso que se realizó, en resumen es el siguiente: Se seleccionan los footprints adecuados para los componentes, los que no se encuentren en las librerías de OrCAD Layout pueden ser diseñados en el mismo programa; posteriormente todos los footprints seleccionados se agregan a una plataforma de diseño, donde se acomodan en la posición que van a ser montados en la tarjeta impresa; se hacen las conexiones entre los componentes y después se trazan las pistas que serán impresas. El siguiente paso en OrCAD Layout es obtener del diseño los archivos Gerber para posteriormente exportarlos a otro programa llamado Circuit Cam, en este otro programa se agregan al diseño de la tarjeta impresa características necesarias para la impresión. Por último, se exporta el diseño creado en Circuit Cam al programa Board Master, el cual controla la máquina que imprime la tarjeta, para finalmente montar los componentes.

4.5 RESULTADOS

Los resultados de esta etapa son las mismas tarjetas de circuito impreso, las cuales en conjunto realizan la tarea de desplegar imágenes. Primeramente, en la Figura 4.10 y 4.11 se muestran la vista frontal y la vista posterior de la pantalla de LEDs. En la fotografía de la Figura 4.10 se puede observar que la pantalla está formada por una matriz de 64x49 LEDs, hecha con módulos de 5x7 LEDs. Adicionalmente a estos pequeños módulos de matriz, en la tarjeta solo se encuentran 5 conectores de 40 patillas, 3 colocados de manera horizontal que son usados para conectar los cables planos provenientes de los transistores MOSFET

y activar las columnas, y otros dos conectores colocados de manera vertical donde se conectarán los cables planos provenientes de los transistores BJT que activan las filas.

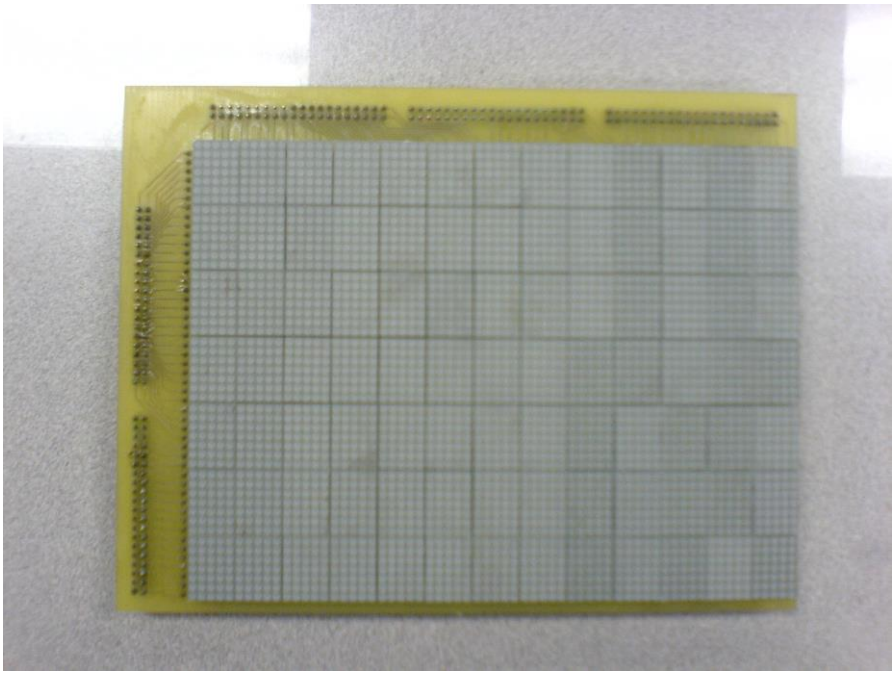


Figura 4.10 Pantalla de LEDs (vista frontal)

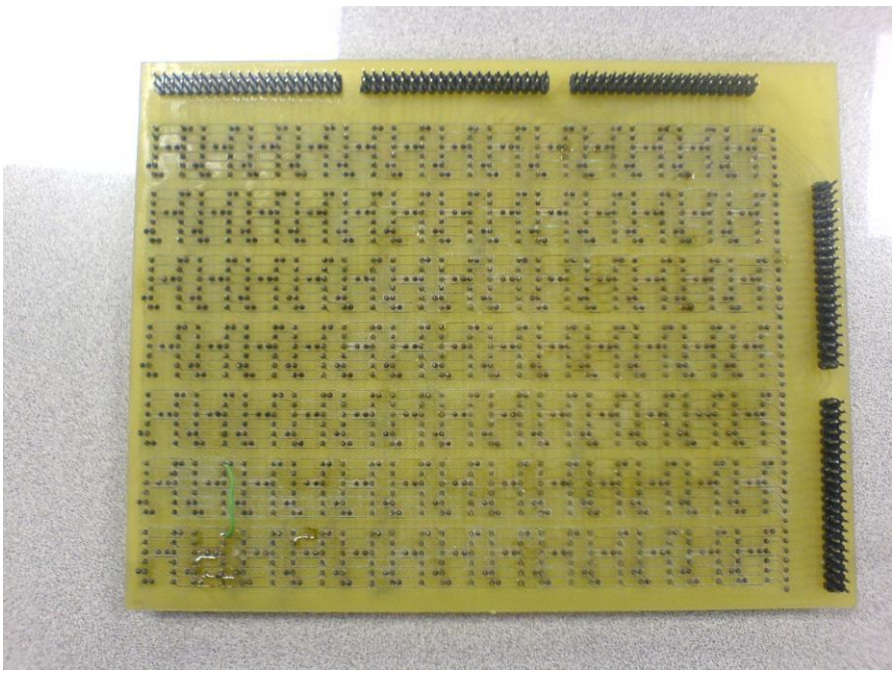


Figura 4.11 Pantalla de LEDs (Vista trasera)

En la Figura 4.12 se muestra una fotografía de la tarjeta que contiene los 48 transistores que activan las filas de la pantalla, en la imagen se puede ver que al diseño le fue agregado en cada transistor la posibilidad de conectar el emisor directamente a la pantalla a través de un puente o controlar la corriente agregando una resistencia entre el emisor y la pantalla. También se observan los conectores de 40 patillas en forma de L, los cuales se conectan al FPGA y los conectores en forma de I que son conectados a la pantalla a través de cable plano.

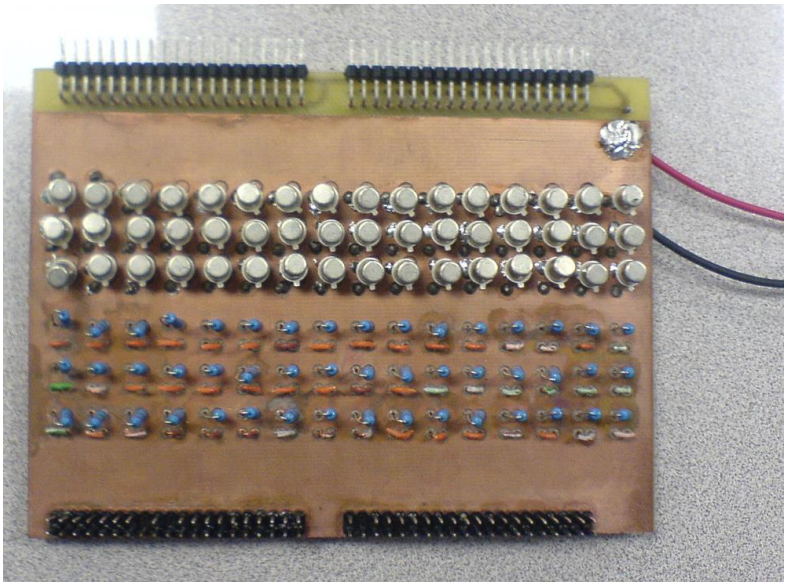


Figura 4.12 Tarjeta completa de transistores BJT de filas.

De la misma manera, la tarjeta con los 5 circuitos integrados, que contienen 20 transistores MOSFET se muestra en la fotografía de la Figura 4.13, donde se puede observar que no requiere resistencias, solamente transistores, que son conectados hacia el FPGA y hacia tierra por un conector de 40 patillas en forma de L, y con otro conector también de 40 patillas, pero este en forma de I, se conecta a través de cable plano a la pantalla.

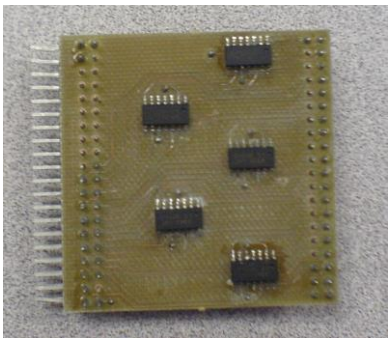


Figura 4.13 Tarjeta con transistores MOSFET para 18 columnas

La última tarjeta que contiene transistores, se muestra en la fotografía de la Figura 4.14. Se observa que contiene 12 circuitos integrados para completar los 46 transistores que hacen falta para el resto de las columnas. Los transistores están conectados de la misma manera que la tarjeta anterior, pero para esta tarjeta se destinaron dos puertos del módulo de desarrollo D2SB, ya que con uno no es suficiente para controlar las 46 columnas. Por esta razón esta tarjeta fue diseñada con cuatro puertos, los dos en forma de L para conectarse al FPGA y los otros dos, en forma de I, para conectarse a la pantalla mediante cable plano.

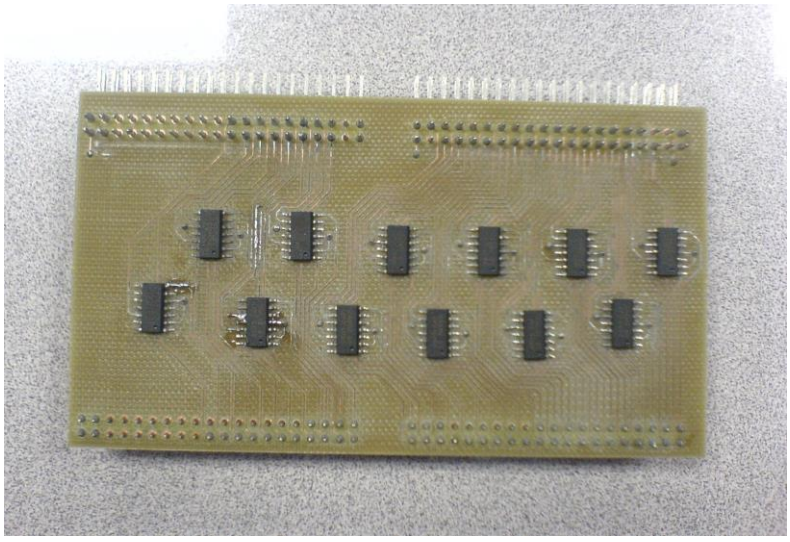


Figura 4.14 Tarjeta de transistores MOSFET de columnas.

Para terminar los resultados de las tarjetas fabricadas, la última corresponde a la fuente de 5V ó 10V en voltaje directo. El diseño se realizó con un regulador de voltaje variable y se elaboró con las recomendaciones de la hoja de datos del regulador. La Figura 4.15 muestra la fotografía de la fuente diseñada, donde pueden verse los componentes usados, así como la forma en que se

conectan. El jumper que se muestra en la figura tiene la función de controlar el voltaje de salida, si el jumper está puesto la salida es de 5V, y si el jumper es retirado, la salida cambia a 10V.

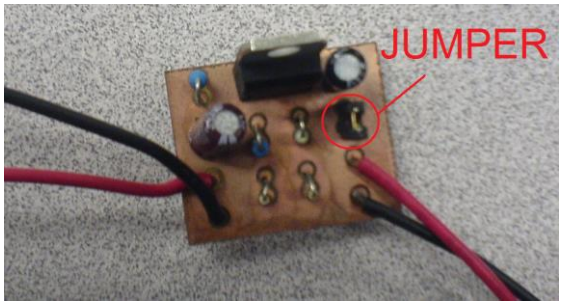


Figura 4.15 Fuente de $5V_{DC}$ ó $10V_{DC}$.

4.5.1 Prueba de eficiencia de LEDs

Se diseñó una sencilla prueba para comprobar que tanta luminiscencia perderá un LED cuando esté funcionando en la pantalla. Para realizar esta prueba, se conectó un LED en voltaje directo y se midió su luminiscencia, posteriormente, se reemplazó la fuente de voltaje directo por una fuente de pulsos, cuyo ciclo de trabajo corresponde al ciclo de trabajo con que cada LED funciona en la pantalla, alrededor del 1.7%, y se midió nuevamente su luminiscencia. El resultado de esta prueba mostró que el LED enciende de una manera muy tenue con este ciclo de trabajo, ya que solo tiene una eficiencia del 3.2%.

4.6 CONCLUSIONES

Hemos fabricado las tarjetas de circuito impreso que nos permiten desplegar imágenes que son enviadas desde la tarjeta de video de una computadora en formato VGA, y convertidas a otro formato por un FPGA Spartan 2E. Estás tarjetas constan de una pantalla de 65x49 LEDs, tres tarjetas con transistores para aumentar la corriente de las señales de VGA y una fuente de voltaje directo variable entre 5 ó 10V.

CAPÍTULO 5

RESULTADOS

Después de ser diseñados y fabricados cada uno de los tres módulos que componen este proyecto, y habiendo obtenido resultados satisfactorios en las pruebas que fueron realizadas a cada uno de ellos, se conectó cada uno de estos módulos en sus respectivas posiciones, para ensamblar el sistema completo, como se muestra en la Figura 5.1.

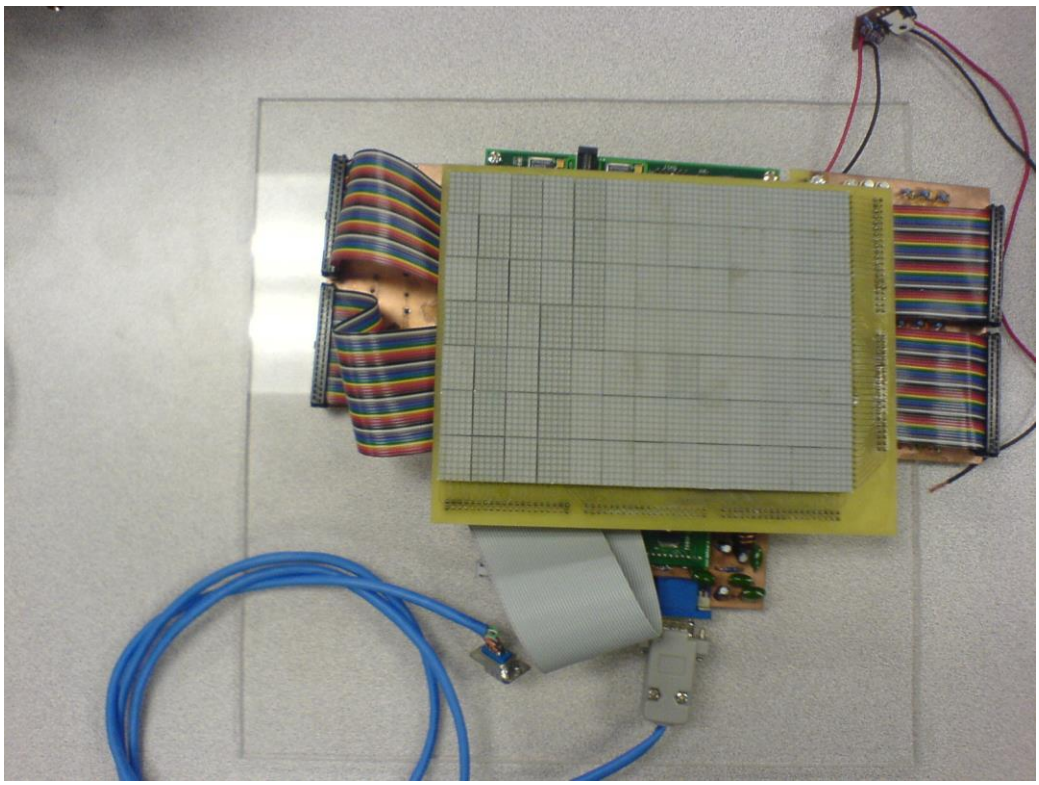


Figura 5.1 Interfaz electrónica para una pantalla de OLEDs.

En la fotografía de la Figura 5.1 se observa que cinco de los seis puertos del módulo de desarrollo D2-SB se encuentran conectados a las tarjetas con transistores BJT y MOSFET, que activan las filas y las columnas de la pantalla, a su vez, estas tarjetas están conectadas a la pantalla de LEDs mediante cable plano. El otro puerto del módulo D2-SB es usado por el módulo de conversión analógico digital, el cual está conectado a la tarjeta de video de una computadora.

Para probar el funcionamiento de la interfaz, fue elaborado en el programa Power Point un conjunto de diapositivas para ser desplegadas en la pantalla, principalmente con imágenes en blanco y negro, y algunas imágenes con movimiento, éstas son mostradas en la Figura 5.2. Posteriormente se conectado a su fuente de alimentación el módulo Spartan 2E, y de la misma manera, la tarjeta con los transistores BJT se alimentó con la fuente diseñada a 5V.

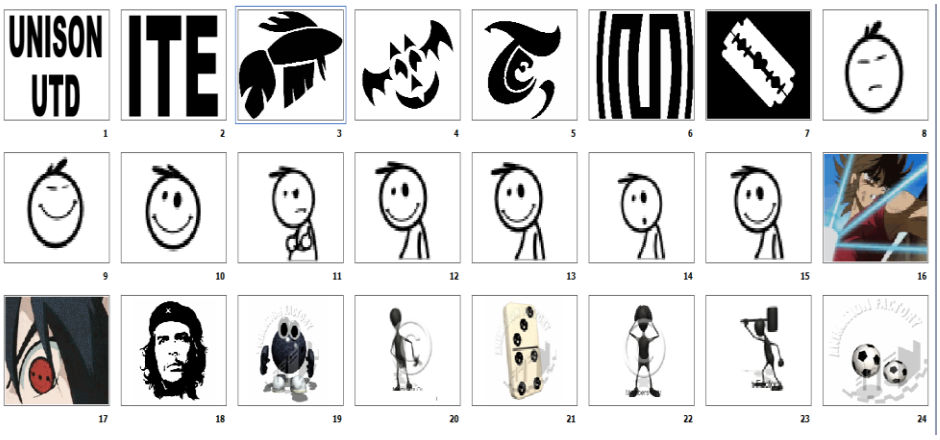
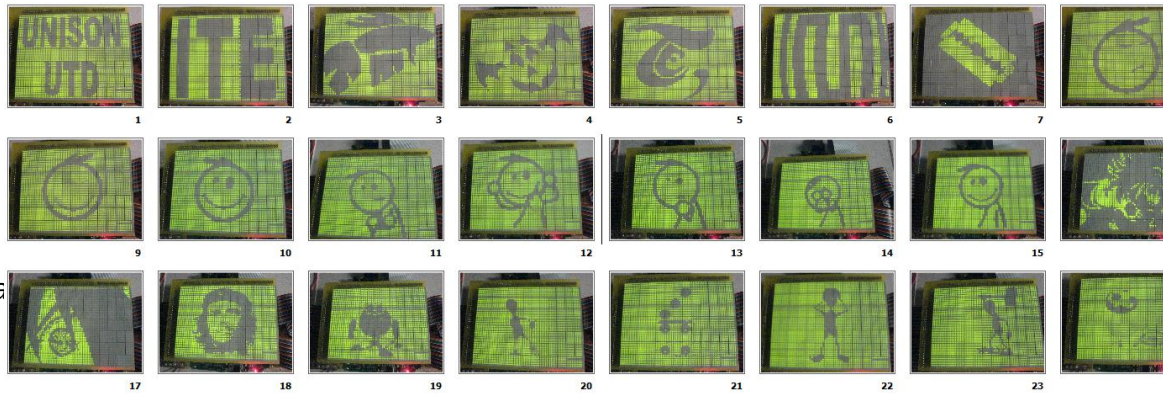


Figura 5.2 Diapositivas elaboradas para desplegar en la pantalla de OLEDs.

Los resultados obtenidos se pueden observar en la fotografía de la Figura 5.3. Como se puede ver, la pantalla desplegó las imágenes de cada una de las diapositivas de la manera como se esperaba, incluyendo las imágenes con movimiento. Además como prueba adicional, se desconectó el cable de la señal VGA, y en la pantalla se desplegó la imagen con el mensaje de “NO SIGNAL”.



Figura

5.1 Observaciones a los resultados

El funcionamiento de esta interfaz electrónica, consiste en recibir mediante un módulo de conversión analógica digital las señales que provienen de la tarjeta de video, y transmite solo las señales que son usadas por el FPGA, no sin antes digitalizar la señal de datos que es enviada por la tarjeta de video en forma analógica. Posteriormente el FPGA recibe la información de las imágenes en formato VGA, y las convierte a un formato diseñado para la pantalla, el cual consiste en desplegar de forma serial las líneas completas de la imagen. Para lograr esto, el FPGA manda señales que activan las filas y las columnas de la pantalla, mediante el uso de transistores, los cuales actúan como interruptores que alimentan una línea o una columna completa de la pantalla. Y finalmente, en la pantalla se encuentra una matriz de 64x48 OLEDs, la forma de activar cada uno de ellos es polarizar la fila y la columna en un mismo momento, es así como se forman las imágenes en la pantalla.

En el despliegue de imágenes en la pantalla de LEDs, existen dos efectos negativos debido al diseño que fue elegido. El primero es que las imágenes se despliegan con muy poco brillo, esto se debe a que el ciclo de trabajo generado por el barrido de las filas es muy pequeño, alrededor del 1.7%, lo cual produce una baja eficiencia en el encendido de cada LED.

El otro efecto negativo es el parpadeo de algunos LED, que se encuentran en los bordes de las imágenes. Esto es debido a que en los niveles medios del color rojo, el convertidor analógico digital se vuelve inestable, ya que alrededor de estos valores se encuentra el punto de transición, es decir, donde el ADC pasa de 0 a 1. Por este motivo, en los bordes de la imagen, el color rojo toma valores cercanos al punto de transición, es por ellos que el ADC en ocasiones tomará un valor de 1, y en otras ocasiones, tomará un valor de 0, generando así el parpadeo de algunos LEDs en los bordes de las imágenes.

CAPÍTULO 6

CONCLUSIONES

Se diseñó y se fabricó una interfaz electrónica capaz de recibir imágenes desde la tarjeta de video de una computadora, en formato VGA de 640x480 pixeles a una frecuencia de 60Hz, y desplegar las imágenes en una pantalla de 64x48 OLEDs. La imagen que se despliega en la pantalla es monocromática y de baja resolución.

Aunque en esta ocasión se trabajó con una pequeña matriz de 64x48 OLEDs, este proyecto no está restringido a estas pequeñas pantallas, ya que el tamaño de la pantalla puede escalarse y amplificarse. También se podrían formar pantallas mucho más grandes uniendo pequeñas pantallas de 64x48 OLEDs, las aplicaciones pueden resultar muy variadas.

Por otro parte, el programa está diseñado de manera que si se desea recibir otro formato de video VGA, que no sea el de 640x480 pixeles, con solo algunas modificaciones al código VHDL pueda adecuarse para cualquier otro formato.

Este proyecto se puede calificar como innovador, ya que trabaja con OLEDs, que aun están en una etapa de desarrollo, pero con los cuales posiblemente se fabricarán pantallas más delgadas, baratas y portables, y probablemente en un futuro llegarán a suplir las pantallas de plasma y cristal líquido.

En lo personal, este proyecto nos ayudó a reforzar temas como el procesamiento de imágenes, programación en VHDL, diseño y fabricación de tarjetas impresas, que después de esta experiencia aplicaremos con más confianza en futuros proyectos.

Por otra parte, la responsabilidad que implicó realizar un proyecto para UTD, nos ha motivado a seguir buscando y desarrollando nuevos proyectos y

nuevas experiencias. Además, el habernos expresado su satisfacción con el proyecto nos ha dejado la seguridad que somos capaces de competir contra cualquier otro ingeniero de cualquier parte del mundo.

Trabajo futuro:

El reto inmediato en este proyecto, consiste en el diseño de una pantalla activa, para reemplazar la pantalla actual que funciona de manera pasiva, lo que puede mejorar la eficiencia de los OLEDs en la pantalla, para que enciendan de una manera más brillante y constante.

Los siguientes pasos posiblemente pudieran ser, aumentar el tamaño de la pantalla; diseñar el código en VHDL de manera que pueda detectar el formato de video que se está recibiendo, como lo hacen los monitores actuales; dejar las pantallas monocromáticas y fabricar pantallas capaces de desplegar en escala de grises; posteriormente pasar a pantallas de OLEDs a color; entre muchas otras cosas, el camino es largo, y el trabajo es mucho, los OLEDs nos ofrecen una gran cantidad de aplicaciones que pudieran ser incluidas en esta lista, pero ya depende del ingenio de los que continúen en este proyecto.

APÉNDICE A

EL FORMATO VGA 640x480 a 60Hz

El formato de video VGA consiste en desplegar imágenes mediante un barrido de pixeles en la pantalla. El barrido es realizado línea por línea de arriba abajo, hasta completar las 480 líneas de la pantalla. A su vez, cada línea es desplegada pixel por pixel de izquierda a derecha hasta completar los 640 puntos que conforman la línea de imagen. En la Figura A.1 [14] se puede observar el orden en el cual se van desplegando los pixeles en un cuadro de imagen.

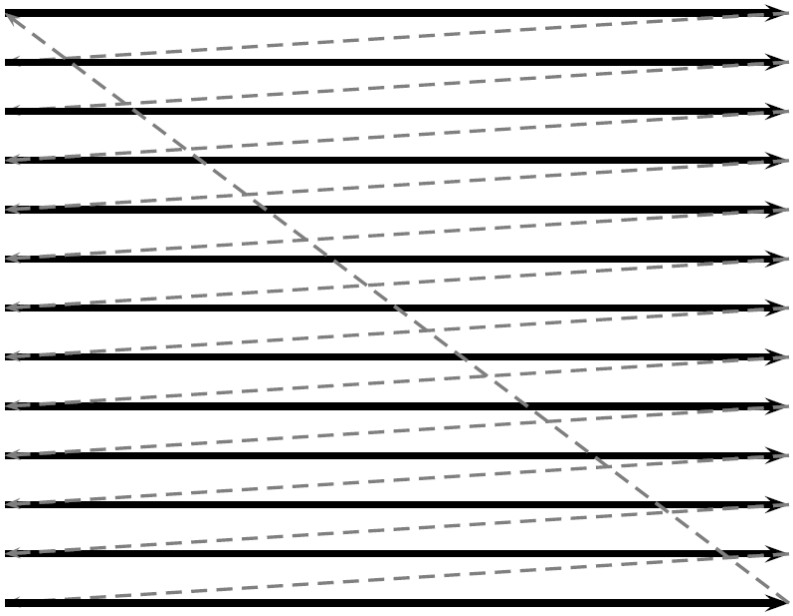


Figura A.1 Modo de barrido en el formato VGA

Para realizar de una forma precisa el barrido de la pantalla, se utilizan señales de sincronía, llamadas VSync y HSync. La señal de sincronía vertical VSync se activa en estado bajo, e indica que se va a empezar a enviar un cuadro o una imagen. En la Figura A.2 [15] se puede observar un diagrama de tiempo de esta señal, como se puede ver el pulso de sincronía tiene una duración de 64μs, posteriormente, hay un tiempo de 1.02ms antes de empezar a enviarse la imagen, este periodo está compuesto por el porche frontal y el borde izquierdo de la imagen. Después son enviados los datos que corresponden a la imagen durante 15.24ms, la imagen es formada por 3 colores, rojo, verde y azul, que corresponden a las tres señales de datos llamadas RGB (Red – Green – Blue).

Antes del siguiente pulso de sincronía hay una duración de 1.35ms, que son llamados borde derecho de la imagen y porche posterior. En el siguiente ciclo, ocurre lo mismo, pero con una nueva imagen.

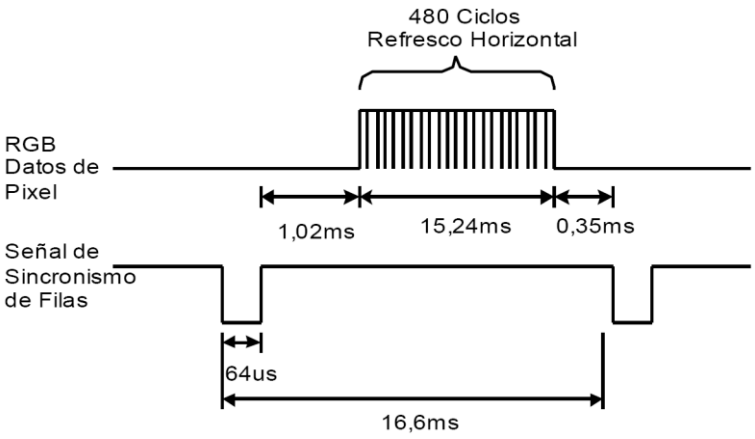


Figura A.2 Diagrama de tiempo de la señal de sincronía VSync.

La señal de sincronía HSync actúa de una manera muy similar a la señal VSync. Esta señal envía un pulso bajo cada vez que se va a enviar una línea nueva. La Figura A.3 [15] muestra un diagrama de tiempo de la señal de sincronía horizontal, se puede observar que después del pulso de sincronía, transcurre un tiempo que corresponde al Porche frontal y al borde izquierdo, posteriormente se envía por las señales RGB la información con los 640 píxeles que conforman la línea de imagen, y posteriormente, hay un tiempo de 0.94μs antes del siguiente pulso de sincronía, el cual pertenece al borde derecho y al porche posterior.

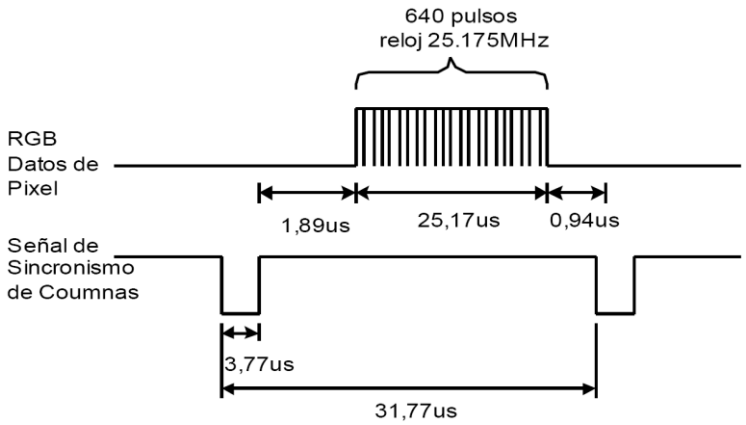


Figura A.3 Diagrama de tiempo de la señal de sincronía HSync.

En base a los diagramas anteriores, se puede decir que la señal VGA envía 800 píxeles en 525 líneas, pero la imagen está formada por 640 pixeles en 480 líneas. En las líneas donde no existe información acerca de la imagen, es donde se generan las señales de sincronía VSync, y en los píxeles que no corresponden a la imagen, se generan los pulsos de sincronía de Hsync. La Figura A.4 [15] muestra una forma de expresar lo que es recibido desde VGA, es decir, una imagen de mayor tamaño a la que es desplegada en el monitor.

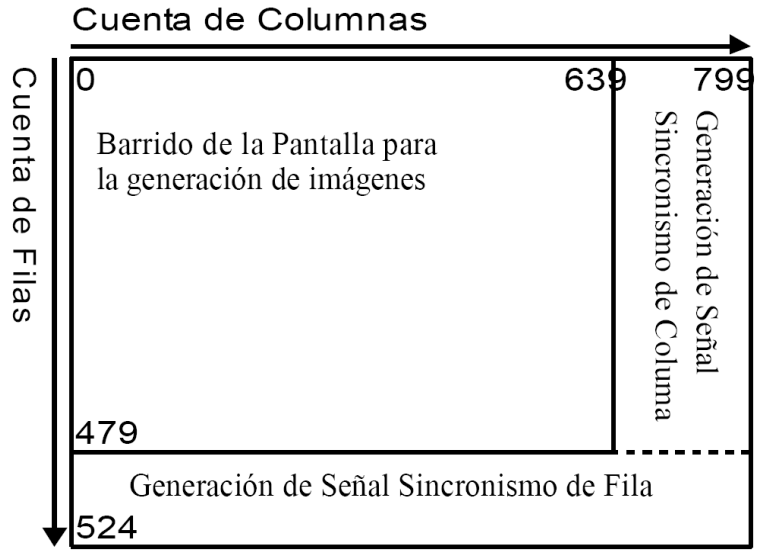


Figura A.4 Zonas donde deben generarse los pulsos de sincronía.

Por último, se muestra un resumen con el tiempo detallado de cada una de las etapas que se realizan en los pulsos de sincronía, tanto para VSync como para HSync.

Resolución	640x480
Frecuencia	60Hz
Frecuencia de pixel	25.175MHz
Frecuencia de línea	31.469KHz
Frecuencia de cuadro	59.94Hz

Función	Pixeles	Función	Líneas
Porche frontal	8	Porche frontal	2
Pulso de sincronía horizontal	96	Pulso de sincronía vertical	2
Porche posterior	40	Porche posterior	25
Borde izquierdo	8	Borde izquierdo	8
Zona activa de la imagen	640	Zona activa de la imagen	480
Borde derecho	8	Borde derecho	8
Total por línea	800	Total por cuadro	525

APÉNDICE B

TARJETAS DE CIRCUITO IMPRESO FABRICADAS (DISEÑO LAYOUT)

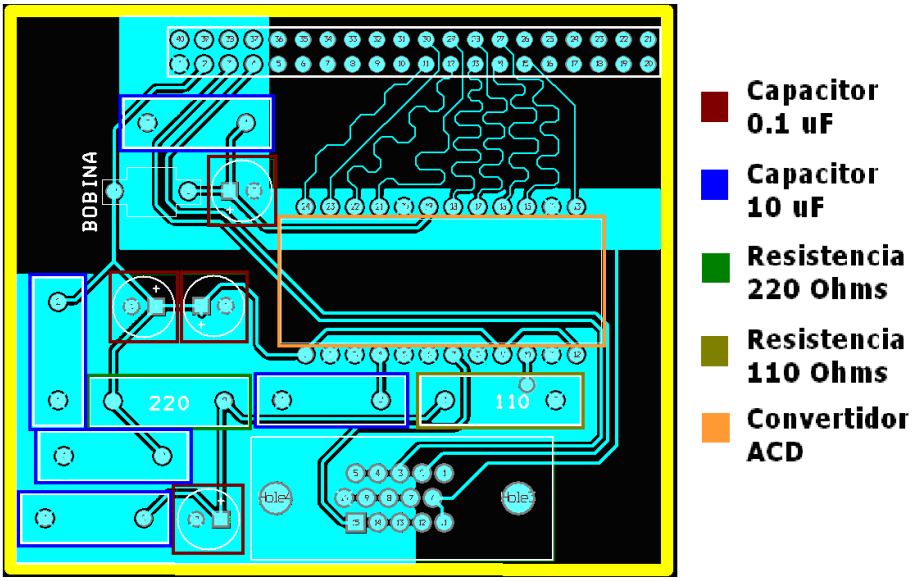


Figura B.1 Capa frontal del módulo de conversión analógica digital.

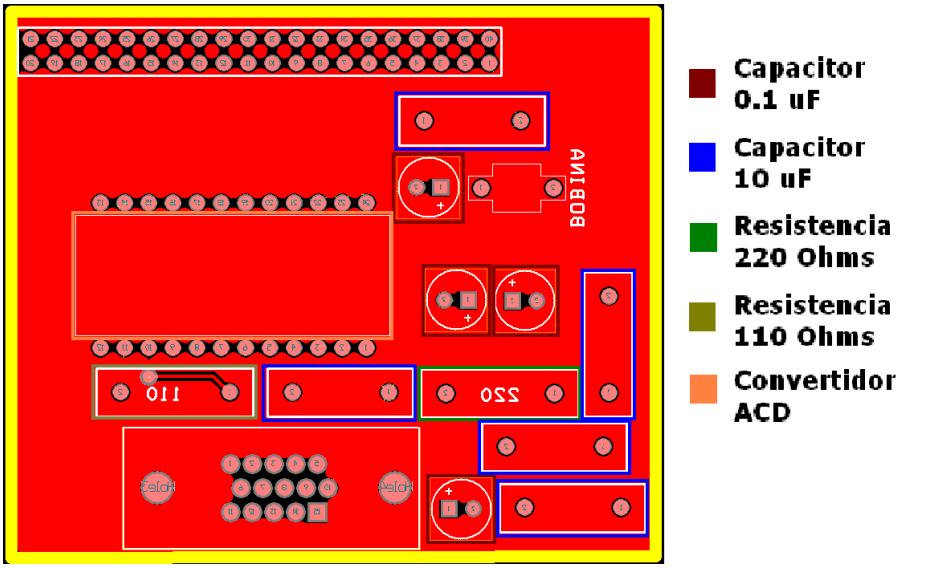


Figura B.2 Capa trasera del módulo de conversión analógica digital.

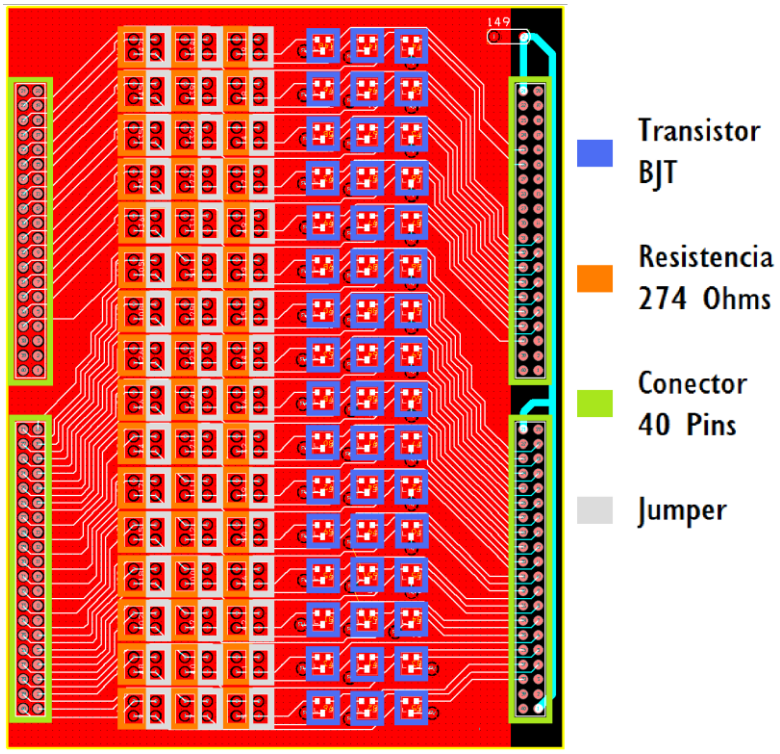


Figura B.3 Tarjeta para activar las 48 filas de la pantalla.

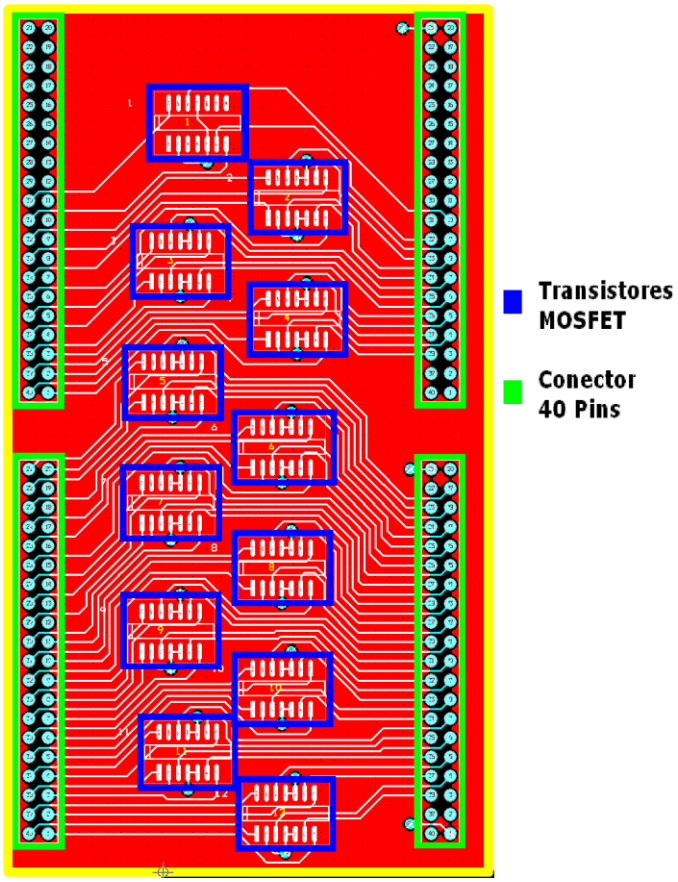


Figura B.4 Tarjeta para activar 46 columnas de la pantalla.

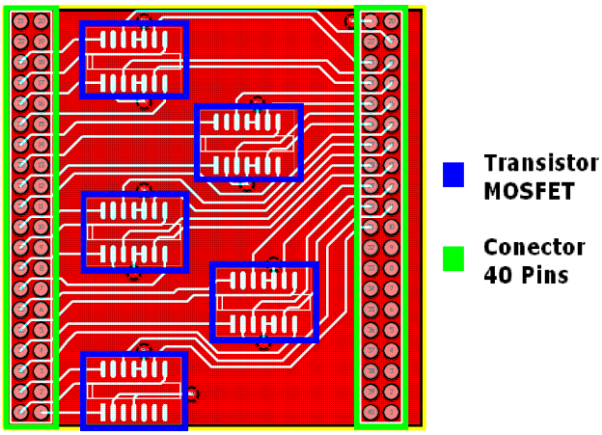


Figura B.5 Tarjeta para activa 18 columnas de la pantalla.

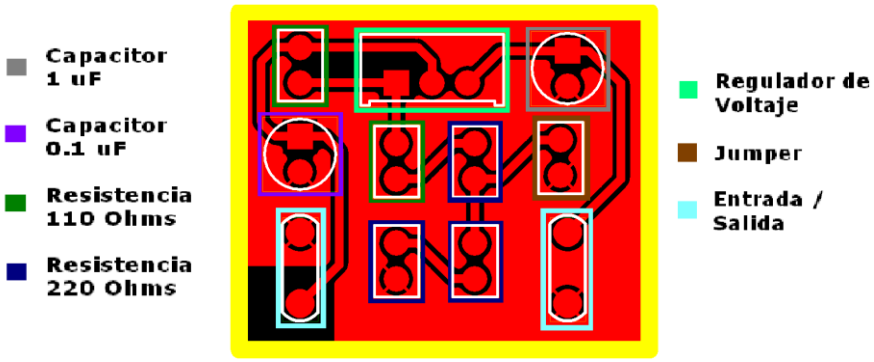


Figura B.6 Fuente de voltaje directo de 5V/10V.

APÉNDICE C

DISEÑO EN VHDL DEL MÓDULO DIGITAL CONVERSIÓN PARA CONVERSIÓN DE VIDEO

Programa principal

```
-- VHDL Programa principal para despliegue de imágenes
-- en pantalla de OLEDs.
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que une todos los componentes y los
-- interconecta, para poder desplegar imágenes en una
-- pantalla de OLEDs, a partir de una señal VGA.

----------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

----------------------------------------------------

-- Declaración de la entidad
entity vga is
      port(
            clk                  : in std_logic;
            rst,hsync,dato,vsync : in std_logic;
            clk_adc              : out std_logic;
            sal                  : out std_logic_vector(63 downto 0);
            sal_vert             : out std_logic_vector(47 downto 0)
      );
end vga;

----------------------------------------------------

-- Declaración de la arquitectura
architecture arqvga of vga is

--Declaración de los componentes a usar
component divisor_freq is
      port(
            clk        : in  std_logic;
            rst        : in  std_logic;
            slow_clk   : out std_logic;
            slow_clkNS : out std_logic
      );
end component;

component cont_dataclk is
      port(
            clk, rst, hsync : in std_logic;
            conta           : out std_logic_vector(9 downto 0)
      );
end component;

component dec_dataclk is
      port(
            cnt_dataclk     : in std_logic_vector(9 downto 0);
```

```
            sal_dec_dataclk : out std_logic_vector(63 downto 0)
      );
end component;

component ffd1 is
      port(
            enable,d : in std_logic;
            q        : out std_logic
      );
end component;

component ffd64 is
      port(
            enable : in std_logic;
            d      : in std_logic_vector(63 downto 0);
            q      : out std_logic_vector(63 downto 0)
      );
end component;

component cont_hsync is
      port(
            rst,hsync,vsync : in std_logic;
            conta           : out std_logic_vector(9 downto 0)
      );
end component;

component dec_hsync is
      port(
            cnt_hsync     : in std_logic_vector(9 downto 0);
            sal_dec_hsync : out std_logic_vector(47 downto 0)
      );
end component;

component dec_hsync_ff is
      port(
            cnt_hsync_ff     : in std_logic_vector(9 downto 0);
            sal_dec_hsync_ff : out std_logic
      );
end component;

component comp is
      port(
            a,b : in std_logic_vector (9 downto 0);
            ma  : out std_logic
      );
end component;

component cont_clk is
      port(
            clk,rst,hsync : in std_logic;
            conta         : out std_logic_vector(9 downto 0)
      );
end component;

component MUX48 is
      port(
            vgafil,NSfil : in std_logic_vector(47 downto 0);
            S            : in std_logic;
            Y            : out std_logic_vector(47 downto 0)
      );
end component;
```

```
component MUX64 is
      Port(
            vgacol,NScol : in std_logic_vector(63 downto 0);
            S            : in std_logic;
            Y            : out std_logic_vector(63 downto 0)
      );
end component;

component pruebaleds is
      port(
            slow_clk,rst : in std_logic;
            fil          : out std_logic_vector(47 downto 0);
            col          : out std_logic_vector(63 downto 0)
      );
end component;

----------------------------------------------------

-- Declaración de las señales a usar

-- Reloj del divisor de frecuencia al contador de dataclk
signal div_cont : STD_LOGIC;

-- Divisor de frecuencia al contador de pulsos para el NS
signal div_NS : std_logic;

-- Señal de cuenta de dataclk al decodificador
signal cont_dec : std_logic_vector(9 downto 0);

-- Señal del decodificador a cad auno de los ff
signal dec_ffd1 : std_logic_vector(63 downto 0);

-- Señal de los ffd1 al ffd64
signal ffd1_ffd64 : std_logic_vector(63 downto 0);

-- Señal del cont_hsync al dec_hsync
signal cont_dec_hsync : std_logic_vector(9 downto 0);

-- Señal de enable del ff64
signal enable_ff64 : std_logic;

-- Señal del contador de reloj para el NS al comparador
signal contclk_comp : std_logic_vector (9 downto 0);

-- Señal de salida del comparador
signal compout : std_logic;

-- Señal de vga 48 bits al mux
signal vga48mux  : std_logic_vector(47 downto 0);

-- Señal de vga 64 bits al mux
signal vga64mux  : std_logic_vector(63 downto 0);

-- Señal NS 48 bits al mux
signal ns48mux   : std_logic_vector(47 downto 0);

-- Señal NS 64 bits al mux
signal ns64mux : std_logic_vector(63 downto 0);

----------------------------------------------------
```

```
-- Inicio del cuerpo del programa
begin

-- Mapeo por nombre del divisor de frecuencia
idivisor : divisor_freq
      port map(
            CLK        => CLK,
            RST        => RST,
            slow_clk   => div_cont,
            slow_clkNS => div_NS
      );

-- Mapeo por nombre del Contador de Pulsos de datos
icontador_dataclk : cont_dataclk
      port map(
            clk   => div_cont,
            rst   => rst,
            hsync => hsync,
            conta => cont_dec
      );

-- Mapeo por nombre del decodificador de dataclk
idec_dataclk : dec_dataclk
      port map(
            cnt_dataclk     => cont_dec,
            sal_dec_dataclk => dec_ffd1
      );

--Mapeo por nombres de los flip-flop de 1 bit
iffd1_0 : ffd1
port map(enable => dec_ffd1(0), d => dato, q => ffd1_ffd64(0));

iffd1_1 : ffd1
port map(enable => dec_ffd1(1), d => dato, q => ffd1_ffd64(1));

iffd1_2 : ffd1
port map(enable => dec_ffd1(2), d => dato, q => ffd1_ffd64(2));

iffd1_3 : ffd1
port map(enable => dec_ffd1(3), d => dato, q => ffd1_ffd64(3));

iffd1_4 : ffd1
port map(enable => dec_ffd1(4), d => dato, q => ffd1_ffd64(4));

iffd1_5 : ffd1
port map(enable => dec_ffd1(5), d => dato, q => ffd1_ffd64(5));

iffd1_6 : ffd1
port map(enable => dec_ffd1(6), d => dato, q => ffd1_ffd64(6));

iffd1_7 : ffd1
port map(enable => dec_ffd1(7), d => dato, q => ffd1_ffd64(7));

iffd1_8 : ffd1
port map(enable => dec_ffd1(8), d => dato, q => ffd1_ffd64(8));

iffd1_9 : ffd1
port map(enable => dec_ffd1(9), d => dato, q => ffd1_ffd64(9));
```

```
iffd1_10 : ffd1
port map(enable => dec_ffd1(10), d => dato, q => ffd1_ffd64(10));

iffd1_11 : ffd1
port map(enable => dec_ffd1(11), d => dato, q => ffd1_ffd64(11));

iffd1_12 : ffd1
port map(enable => dec_ffd1(12), d => dato, q => ffd1_ffd64(12));

iffd1_13 : ffd1
port map(enable => dec_ffd1(13), d => dato, q => ffd1_ffd64(13));

iffd1_14 : ffd1
port map(enable => dec_ffd1(14), d => dato, q => ffd1_ffd64(14));

iffd1_15 : ffd1
port map(enable => dec_ffd1(15), d => dato, q => ffd1_ffd64(15));

iffd1_16 : ffd1
port map(enable => dec_ffd1(16), d => dato, q => ffd1_ffd64(16));

iffd1_17 : ffd1
port map(enable => dec_ffd1(17), d => dato, q => ffd1_ffd64(17));

iffd1_18 : ffd1
port map(enable => dec_ffd1(18), d => dato, q => ffd1_ffd64(18));

iffd1_19 : ffd1
port map(enable => dec_ffd1(19), d => dato, q => ffd1_ffd64(19));

iffd1_20 : ffd1
port map(enable => dec_ffd1(20), d => dato, q => ffd1_ffd64(20));

iffd1_21 : ffd1
port map(enable => dec_ffd1(21), d => dato, q => ffd1_ffd64(21));

iffd1_22 : ffd1
port map(enable => dec_ffd1(22), d => dato, q => ffd1_ffd64(22));

iffd1_23 : ffd1
port map(enable => dec_ffd1(23), d => dato, q => ffd1_ffd64(23));

iffd1_24 : ffd1
port map(enable => dec_ffd1(24), d => dato, q => ffd1_ffd64(24));

iffd1_25 : ffd1
port map(enable => dec_ffd1(25), d => dato, q => ffd1_ffd64(25));

iffd1_26 : ffd1
port map(enable => dec_ffd1(26), d => dato, q => ffd1_ffd64(26));

iffd1_27 : ffd1
port map(enable => dec_ffd1(27), d => dato, q => ffd1_ffd64(27));

iffd1_28 : ffd1
port map(enable => dec_ffd1(28), d => dato, q => ffd1_ffd64(28));

iffd1_29 : ffd1
port map(enable => dec_ffd1(29), d => dato, q => ffd1_ffd64(29));

iffd1_30 : ffd1
```

```
port map(enable => dec_ffd1(30), d => dato, q => ffd1_ffd64(30));

iffd1_31 : ffd1
port map(enable => dec_ffd1(31), d => dato, q => ffd1_ffd64(31));

iffd1_32 : ffd1
port map(enable => dec_ffd1(32), d => dato, q => ffd1_ffd64(32));

iffd1_33 : ffd1
port map(enable => dec_ffd1(33), d => dato, q => ffd1_ffd64(33));

iffd1_34 : ffd1
port map(enable => dec_ffd1(34), d => dato, q => ffd1_ffd64(34));

iffd1_35 : ffd1
port map(enable => dec_ffd1(35), d => dato, q => ffd1_ffd64(35));

iffd1_36 : ffd1
port map(enable => dec_ffd1(36), d => dato, q => ffd1_ffd64(36));

iffd1_37 : ffd1
port map(enable => dec_ffd1(37), d => dato, q => ffd1_ffd64(37));

iffd1_38 : ffd1
port map(enable => dec_ffd1(38), d => dato, q => ffd1_ffd64(38));

iffd1_39 : ffd1
port map(enable => dec_ffd1(39), d => dato, q => ffd1_ffd64(39));

iffd1_40 : ffd1
port map(enable => dec_ffd1(40), d => dato, q => ffd1_ffd64(40));

iffd1_41 : ffd1
port map(enable => dec_ffd1(41), d => dato, q => ffd1_ffd64(41));

iffd1_42 : ffd1
port map(enable => dec_ffd1(42), d => dato, q => ffd1_ffd64(42));

iffd1_43 : ffd1
port map(enable => dec_ffd1(43), d => dato, q => ffd1_ffd64(43));

iffd1_44 : ffd1
port map(enable => dec_ffd1(44), d => dato, q => ffd1_ffd64(44));

iffd1_45 : ffd1
port map(enable => dec_ffd1(45), d => dato, q => ffd1_ffd64(45));

iffd1_46 : ffd1
port map(enable => dec_ffd1(46), d => dato, q => ffd1_ffd64(46));

iffd1_47 : ffd1
port map(enable => dec_ffd1(47), d => dato, q => ffd1_ffd64(47));

iffd1_48 : ffd1
port map(enable => dec_ffd1(48), d => dato, q => ffd1_ffd64(48));

iffd1_49 : ffd1
port map(enable => dec_ffd1(49), d => dato, q => ffd1_ffd64(49));

iffd1_50 : ffd1
port map(enable => dec_ffd1(50), d => dato, q => ffd1_ffd64(50));
```

```
iffd1_51 : ffd1
port map(enable => dec_ffd1(51), d => dato, q => ffd1_ffd64(51));

iffd1_52 : ffd1
port map(enable => dec_ffd1(52), d => dato, q => ffd1_ffd64(52));

iffd1_53 : ffd1
port map(enable => dec_ffd1(53), d => dato, q => ffd1_ffd64(53));

iffd1_54 : ffd1
port map(enable => dec_ffd1(54), d => dato, q => ffd1_ffd64(54));

iffd1_55 : ffd1
port map(enable => dec_ffd1(55), d => dato, q => ffd1_ffd64(55));

iffd1_56 : ffd1
port map(enable => dec_ffd1(56), d => dato, q => ffd1_ffd64(56));

iffd1_57 : ffd1
port map(enable => dec_ffd1(57), d => dato, q => ffd1_ffd64(57));

iffd1_58 : ffd1
port map(enable => dec_ffd1(58), d => dato, q => ffd1_ffd64(58));

iffd1_59 : ffd1
port map(enable => dec_ffd1(59), d => dato, q => ffd1_ffd64(59));

iffd1_60 : ffd1
port map(enable => dec_ffd1(60), d => dato, q => ffd1_ffd64(60));

iffd1_61 : ffd1
port map(enable => dec_ffd1(61), d => dato, q => ffd1_ffd64(61));

iffd1_62 : ffd1
port map(enable => dec_ffd1(62), d => dato, q => ffd1_ffd64(62));

iffd1_63 : ffd1
port map(enable => dec_ffd1(63), d => dato, q => ffd1_ffd64(63));

--Mapeo por nombre del flip-flop de 64 bits
iffd64 : ffd64
      port map(
            enable => enable_ff64,
            d      => ffd1_ffd64,
            q      => vga64mux
       );

-- Mapeo por nombre del contador de hsync
icont_hsync : cont_hsync
      port map(
            rst   => RST,
            hsync => hsync,
            vsync => vsync,
            conta => cont_dec_hsync
      );

-- Mapeo por nombre del decodificador hsync
idec_hsync : dec_hsync
      port map(
            cnt_hsync     => cont_dec_hsync,
```

```
            sal_dec_hsync => vga48mux
      );

-- Mapeo por nombre del decodificador que activa el flip-flop de 64 bit
idec_hsync_ff : dec_hsync_ff
      port map(
            cnt_hsync_ff     => cont_dec_hsync,
            sal_dec_hsync_ff => enable_ff64
      );

-- Mapeo por nombre del comparador
comparador : comp
      port map(
            a  => contclk_comp,
            b  => "0001111000",
            ma => compout
      );

-- Mapeo por nombre del contador de datos para el NS
icontadorclk : cont_clk
      port map(
            clk   => div_cont,
            rst   => rst,
            hsync => hsync,
            conta => contclk_comp
      );

-- Mapeo por nombre del multiplexor de 48 bits
imux48 : MUX48
      port map(
            vgafil => vga48mux,
            NSfil  => ns48mux,
            S      => compout,
            Y      => sal_vert
      );

-- Mapeo por nombre del multiplexor de 64 bits
imux64  :  MUX64
      Port map(
            vgacol => vga64mux,
            NScol  => ns64mux,
            S      => compout,
            Y      => sal
      );

-- Mapeo por nombre del generador de pantalla "NO SIGNAL"
ipruebaleds : pruebaleds
      port map(
            slow_clk => div_NS,
            rst      => rst,
            fil      => ns48mux,
            col      => ns64mux
      );

-- Señal de 25MHz conectada a la salida del ADC
clk_adc <= div_cont;

end arqvga;
```

Divisor de frecuencia

```
-- VHDL Divisor de frecuencia
```

```
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que genera dos frecuencias, una de 25MHz
-- y otra de 24KHz a partir de la frecuencia de entrada
-- de 50MHz

----------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

----------------------------------------------------

-- Declaración de la entidad
entity divisor_freq is
      port(
            clk                 : in  std_logic;
            rst                 : in  std_logic;
            slow_clk,slow_clkNS : out std_logic
      );
end divisor_freq;

----------------------------------------------------

-- Declaración de la arquitectura
architecture Behavioral of divisor_freq is

-- Señal para realizar conteo
signal mycount : std_logic_vector(10 downto 0);

begin

      process(CLK, RST)
      begin
            if(RST = '1') then
                  mycount <= (others => '0');
            elsif(rising_edge(CLK)) then
                  mycount <= mycount + 1;
            end if;
      end process;

-- Lógica concurrente
slow_clk <= mycount(0);
slow_clkNS <= mycount(10);

end Behavioral;
```

MÓDULO DE CONTEO Y HABILITACIÓN

Contador de pulsos HSync

```
-- VHDL Contador para pulsos de HSync
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que cuenta pulsos bajos de Hsync y
-- representa el valor de conteo en una señal de
-- 10 bit, pero además su salida es reestablecida
-- con un pulso bajo de VSync
```

```
----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

----------------------------------------------------

-- Declaración de la entidad
entity cont_hsync is
      port(
            rst,hsync,vsync : in std_logic;
            conta           : out std_logic_vector(9 downto 0)
      );
end cont_hsync;

----------------------------------------------------

-- Declaración de la arquitectura
architecture arq_cont_hsync of cont_hsync is

-- Señal con el valor del conteo
signal cuenta: std_logic_vector(9 downto 0);

begin

      process (rst,hsync,vsync)
      begin
            if rst = '1' then
                  cuenta <= (others=>'0');
            elsif (hsync'event and hsync= '1') then
                  if vsync = '0' then
                        cuenta <= (others=>'0');
                  else cuenta <= cuenta + 1;
                  end if;
            end if;
      end process;

conta <= cuenta;

end arq_cont_hsync;
```

Codificador para filas VGA

```
-- VHDL Codificador para activar las filas para
-- desplegar imágenes desde VGA
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que genera la señal que activa las columnas
-- en la pantalla, cuando se va a desplegar una
-- imagen desde VGA, recibe de entrada la señal del
-- contador de pulsos HSync

----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------------

-- Declaración de la entidad
```

```
entity dec_hsync is
      port(
            cnt_hsync     : in std_logic_vector(9 downto 0);
            sal_dec_hsync : out std_logic_vector(47 downto 0)
      );
end dec_hsync;

----------------------------------------------------

-- Declaración de la arquitectura
architecture arq_dec_hsync of dec_hsync is

begin

      decode: process (cnt_hsync)
      begin
            case cnt_hsync is
            when "0000100111" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000001";
            when "0000101000" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000001";
            when "0000101001" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000001";
            when "0000101010" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000001";
            when "0000101011" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000001";
            when "0000101100" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000001";
            when "0000101101" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000001";
            when "0000101110" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000001";
            when "0000101111" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000001";

            when "0000110001" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000010";
            when "0000110010" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000010";
            when "0000110011" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000010";
            when "0000110100" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000010";
            when "0000110101" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000010";
            when "0000110110" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000010";
            when "0000110111" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000010";
            when "0000111000" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000010";
            when "0000111001" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000010";

            when "0000111011" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000100";
            when "0000111100" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000100";
            when "0000111101" => sal_dec_hsync <=
"000000000000000000000000000000000000000000000100";
```

```
            when "0000111110" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000000100";
            when "0000111111" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000000100";
            when "0001000000" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000000100";
            when "0001000001" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000000100";
            when "0001000010" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000000100";
            when "0001000011" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000000100";

            when "0001000101" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000001000";
            when "0001000110" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000001000";
            when "0001000111" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000001000";
            when "0001001000" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000001000";
            when "0001001001" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000001000";
            when "0001001010" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000001000";
            when "0001001011" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000001000";
            when "0001001100" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000001000";
            when "0001001101" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000001000";

            when "0001001111" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000010000";
            when "0001010000" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000010000";
            when "0001010001" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000010000";
            when "0001010010" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000010000";
            when "0001010011" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000010000";
            when "0001010100" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000010000";
            when "0001010101" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000010000";
            when "0001010110" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000010000";
            when "0001010111" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000010000";

            when "0001011001" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000100000";
            when "0001011010" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000100000";
            when "0001011011" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000100000";
            when "0001011100" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000100000";
            when "0001011101" => sal_dec_hsync <=
"00000000000000000000000000000000000000000000100000";
```

```
            when "0001011110" => sal_dec_hsync <=
"00000000000000000000000000000000000000000100000";
            when "0001011111" => sal_dec_hsync <=
"00000000000000000000000000000000000000000100000";
            when "0001100000" => sal_dec_hsync <=
"00000000000000000000000000000000000000000100000";
            when "0001100001" => sal_dec_hsync <=
"00000000000000000000000000000000000000000100000";

            when "0001100011" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000";
            when "0001100100" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000";
            when "0001100101" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000";
            when "0001100110" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000";
            when "0001100111" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000";
            when "0001101000" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000";
            when "0001101001" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000";
            when "0001101010" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000";
            when "0001101011" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000";

            when "0001101101" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000";
            when "0001101110" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000";
            when "0001101111" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000";
            when "0001110000" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000";
            when "0001110001" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000";
            when "0001110010" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000";
            when "0001110011" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000";
            when "0001110100" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000";
            when "0001110101" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000";

            when "0001110111" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000";
            when "0001111000" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000";
            when "0001111001" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000";
            when "0001111010" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000";
            when "0001111011" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000";
            when "0001111100" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000";
            when "0001111101" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000";
```

```
            when "0001111110" => sal_dec_hsync <=
"00000000000000000000000000000000000000000100000000";
            when "0001111111" => sal_dec_hsync <=
"00000000000000000000000000000000000000000100000000";

            when "0010000001" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000000";
            when "0010000010" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000000";
            when "0010000011" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000000";
            when "0010000100" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000000";
            when "0010000101" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000000";
            when "0010000110" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000000";
            when "0010000111" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000000";
            when "0010001000" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000000";
            when "0010001001" => sal_dec_hsync <=
"00000000000000000000000000000000000000001000000000";

            when "0010001011" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000000";
            when "0010001100" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000000";
            when "0010001101" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000000";
            when "0010001110" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000000";
            when "0010001111" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000000";
            when "0010010000" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000000";
            when "0010010001" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000000";
            when "0010010010" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000000";
            when "0010010011" => sal_dec_hsync <=
"00000000000000000000000000000000000000010000000000";

            when "0010010101" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000000";
            when "0010010110" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000000";
            when "0010010111" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000000";
            when "0010011000" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000000";
            when "0010011001" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000000";
            when "0010011010" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000000";
            when "0010011011" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000000";
            when "0010011100" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000000";
            when "0010011101" => sal_dec_hsync <=
"00000000000000000000000000000000000000100000000000";
```

```
            when "0010011111" => sal_dec_hsync <=
"00000000000000000000000000000000000010000000000000";
            when "0010100000" => sal_dec_hsync <=
"00000000000000000000000000000000000010000000000000";
            when "0010100001" => sal_dec_hsync <=
"00000000000000000000000000000000000010000000000000";
            when "0010100010" => sal_dec_hsync <=
"00000000000000000000000000000000000010000000000000";
            when "0010100011" => sal_dec_hsync <=
"00000000000000000000000000000000000010000000000000";
            when "0010100100" => sal_dec_hsync <=
"00000000000000000000000000000000000010000000000000";
            when "0010100101" => sal_dec_hsync <=
"00000000000000000000000000000000000010000000000000";
            when "0010100110" => sal_dec_hsync <=
"00000000000000000000000000000000000010000000000000";
            when "0010100111" => sal_dec_hsync <=
"00000000000000000000000000000000000010000000000000";

            when "0010101001" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010101010" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010101011" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010101100" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010101101" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010101110" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010101111" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010110000" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010110001" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";

            when "0010110011" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010110100" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010110101" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010110110" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010110111" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010111000" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010111001" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010111010" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
            when "0010111011" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";

            when "0010111101" => sal_dec_hsync <=
"00000000000000000000000000000000000100000000000000";
```

```
            when "0010111110" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0010111111" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011000000" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011000001" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011000010" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011000011" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011000100" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011000101" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";

            when "0011000111" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011001000" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011001001" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011001010" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011001011" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011001100" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011001101" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011001110" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011001111" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";

            when "0011010001" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011010010" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011010011" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011010100" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011010101" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011010110" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011010111" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011011000" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011011001" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";

            when "0011011011" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011011100" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
            when "0011011101" => sal_dec_hsync <=
"00000000000000000000000000000000100000000000000";
```

```
            when "0011011110" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011011111" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011100000" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011100001" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011100010" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011100011" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";

            when "0011100101" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011100110" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011100111" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011101000" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011101001" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011101010" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011101011" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011101100" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011101101" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";

            when "0011101111" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011110000" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011110001" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011110010" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011110011" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011110100" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011110101" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011110110" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011110111" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";

            when "0011111001" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011111010" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011111011" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011111100" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
            when "0011111101" => sal_dec_hsync <=
"00000000000000000000000000000010000000000000000000";
```

```
            when "0011111110" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0011111111" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100000000" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100000001" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";

            when "0100000011" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100000100" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100000101" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100000110" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100000111" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100001000" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100001001" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100001010" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100001011" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";

            when "0100001101" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100001110" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100001111" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100010000" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100010001" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100010010" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100010011" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100010100" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100010101" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";

            when "0100010111" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100011000" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100011001" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100011010" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100011011" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100011100" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
            when "0100011101" => sal_dec_hsync <=
"00000000000000000000000000010000000000000000000000";
```

```
            when "0100011110" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100011111" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";

            when "0100100001" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100100010" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100100011" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100100100" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100100101" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100100110" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100100111" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100101000" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100101001" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";

            when "0100101011" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100101100" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100101101" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100101110" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100101111" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100110000" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100110001" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100110010" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100110011" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";

            when "0100110101" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100110110" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100110111" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100111000" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100111001" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100111010" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100111011" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100111100" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
            when "0100111101" => sal_dec_hsync <=
"00000000000000000000000010000000000000000000000000";
```

```
            when "0100111111" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101000000" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101000001" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101000010" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101000011" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101000100" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101000101" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101000110" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101000111" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";

            when "0101001001" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101001010" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101001011" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101001100" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101001101" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101001110" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101001111" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101010000" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101010001" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";

            when "0101010011" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101010100" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101010101" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101010110" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101010111" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101011000" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101011001" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101011010" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
            when "0101011011" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";

            when "0101011101" => sal_dec_hsync <=
"00000000000000000001000000000000000000000000000000";
```

```
            when "0101011110" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101011111" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101100000" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101100001" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101100010" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101100011" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101100100" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101100101" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";

            when "0101100111" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101101000" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101101001" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101101010" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101101011" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101101100" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101101101" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101101110" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101101111" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";

            when "0101110001" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101110010" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101110011" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101110100" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101110101" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101110110" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101110111" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101111000" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101111001" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";

            when "0101111011" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101111100" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
            when "0101111101" => sal_dec_hsync <=
"00000000000000001000000000000000000000000000000000";
```

```
            when "0101111110" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0101111111" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110000000" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110000001" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110000010" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110000011" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";

            when "0110000101" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110000110" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110000111" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110001000" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110001001" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110001010" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110001011" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110001100" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110001101" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";

            when "0110001111" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110010000" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110010001" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110010010" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110010011" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110010100" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110010101" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110010110" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";
            when "0110010111" => sal_dec_hsync <=
"00000000000001000000000000000000000000000000000000";

            when "0110011001" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110011010" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110011011" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110011100" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110011101" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
```

```
            when "0110011110" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110011111" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110100000" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110100001" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";

            when "0110100011" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110100100" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110100101" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110100110" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110100111" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110101000" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110101001" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110101010" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110101011" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";

            when "0110101101" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110101110" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110101111" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110110000" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110110001" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110110010" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110110011" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110110100" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";
            when "0110110101" => sal_dec_hsync <=
"00000000001000000000000000000000000000000000000000";

            when "0110110111" => sal_dec_hsync <=
"00000001000000000000000000000000000000000000000000";
            when "0110111000" => sal_dec_hsync <=
"00000001000000000000000000000000000000000000000000";
            when "0110111001" => sal_dec_hsync <=
"00000001000000000000000000000000000000000000000000";
            when "0110111010" => sal_dec_hsync <=
"00000001000000000000000000000000000000000000000000";
            when "0110111011" => sal_dec_hsync <=
"00000001000000000000000000000000000000000000000000";
            when "0110111100" => sal_dec_hsync <=
"00000001000000000000000000000000000000000000000000";
            when "0110111101" => sal_dec_hsync <=
"00000001000000000000000000000000000000000000000000";
```

```
            when "0110111110" => sal_dec_hsync <=
"00000001000000000000000000000000000000000000000000";
            when "0110111111" => sal_dec_hsync <=
"00000001000000000000000000000000000000000000000000";

            when "0111000001" => sal_dec_hsync <=
"00000010000000000000000000000000000000000000000000";
            when "0111000010" => sal_dec_hsync <=
"00000010000000000000000000000000000000000000000000";
            when "0111000011" => sal_dec_hsync <=
"00000010000000000000000000000000000000000000000000";
            when "0111000100" => sal_dec_hsync <=
"00000010000000000000000000000000000000000000000000";
            when "0111000101" => sal_dec_hsync <=
"00000010000000000000000000000000000000000000000000";
            when "0111000110" => sal_dec_hsync <=
"00000010000000000000000000000000000000000000000000";
            when "0111000111" => sal_dec_hsync <=
"00000010000000000000000000000000000000000000000000";
            when "0111001000" => sal_dec_hsync <=
"00000010000000000000000000000000000000000000000000";
            when "0111001001" => sal_dec_hsync <=
"00000010000000000000000000000000000000000000000000";

            when "0111001011" => sal_dec_hsync <=
"00000100000000000000000000000000000000000000000000";
            when "0111001100" => sal_dec_hsync <=
"00000100000000000000000000000000000000000000000000";
            when "0111001101" => sal_dec_hsync <=
"00000100000000000000000000000000000000000000000000";
            when "0111001110" => sal_dec_hsync <=
"00000100000000000000000000000000000000000000000000";
            when "0111001111" => sal_dec_hsync <=
"00000100000000000000000000000000000000000000000000";
            when "0111010000" => sal_dec_hsync <=
"00000100000000000000000000000000000000000000000000";
            when "0111010001" => sal_dec_hsync <=
"00000100000000000000000000000000000000000000000000";
            when "0111010010" => sal_dec_hsync <=
"00000100000000000000000000000000000000000000000000";
            when "0111010011" => sal_dec_hsync <=
"00000100000000000000000000000000000000000000000000";

            when "0111010101" => sal_dec_hsync <=
"00001000000000000000000000000000000000000000000000";
            when "0111010110" => sal_dec_hsync <=
"00001000000000000000000000000000000000000000000000";
            when "0111010111" => sal_dec_hsync <=
"00001000000000000000000000000000000000000000000000";
            when "0111011000" => sal_dec_hsync <=
"00001000000000000000000000000000000000000000000000";
            when "0111011001" => sal_dec_hsync <=
"00001000000000000000000000000000000000000000000000";
            when "0111011010" => sal_dec_hsync <=
"00001000000000000000000000000000000000000000000000";
            when "0111011011" => sal_dec_hsync <=
"00001000000000000000000000000000000000000000000000";
            when "0111011100" => sal_dec_hsync <=
"00001000000000000000000000000000000000000000000000";
            when "0111011101" => sal_dec_hsync <=
"00001000000000000000000000000000000000000000000000";
```

```
            when "0111011111" => sal_dec_hsync <=
"00010000000000000000000000000000000000000000000000";
            when "0111100000" => sal_dec_hsync <=
"00010000000000000000000000000000000000000000000000";
            when "0111100001" => sal_dec_hsync <=
"00010000000000000000000000000000000000000000000000";
            when "0111100010" => sal_dec_hsync <=
"00010000000000000000000000000000000000000000000000";
            when "0111100011" => sal_dec_hsync <=
"00010000000000000000000000000000000000000000000000";
            when "0111100100" => sal_dec_hsync <=
"00010000000000000000000000000000000000000000000000";
            when "0111100101" => sal_dec_hsync <=
"00010000000000000000000000000000000000000000000000";
            when "0111100110" => sal_dec_hsync <=
"00010000000000000000000000000000000000000000000000";
            when "0111100111" => sal_dec_hsync <=
"00010000000000000000000000000000000000000000000000";

            when "0111101001" => sal_dec_hsync <=
"00100000000000000000000000000000000000000000000000";
            when "0111101010" => sal_dec_hsync <=
"00100000000000000000000000000000000000000000000000";
            when "0111101011" => sal_dec_hsync <=
"00100000000000000000000000000000000000000000000000";
            when "0111101100" => sal_dec_hsync <=
"00100000000000000000000000000000000000000000000000";
            when "0111101101" => sal_dec_hsync <=
"00100000000000000000000000000000000000000000000000";
            when "0111101110" => sal_dec_hsync <=
"00100000000000000000000000000000000000000000000000";
            when "0111101111" => sal_dec_hsync <=
"00100000000000000000000000000000000000000000000000";
            when "0111110000" => sal_dec_hsync <=
"00100000000000000000000000000000000000000000000000";
            when "0111110001" => sal_dec_hsync <=
"00100000000000000000000000000000000000000000000000";

            when "0111110011" => sal_dec_hsync <=
"01000000000000000000000000000000000000000000000000";
            when "0111110100" => sal_dec_hsync <=
"01000000000000000000000000000000000000000000000000";
            when "0111110101" => sal_dec_hsync <=
"01000000000000000000000000000000000000000000000000";
            when "0111110110" => sal_dec_hsync <=
"01000000000000000000000000000000000000000000000000";
            when "0111110111" => sal_dec_hsync <=
"01000000000000000000000000000000000000000000000000";
            when "0111111000" => sal_dec_hsync <=
"01000000000000000000000000000000000000000000000000";
            when "0111111001" => sal_dec_hsync <=
"01000000000000000000000000000000000000000000000000";
            when "0111111010" => sal_dec_hsync <=
"01000000000000000000000000000000000000000000000000";
            when "0111111011" => sal_dec_hsync <=
"01000000000000000000000000000000000000000000000000";

            when "0111111101" => sal_dec_hsync <=
"10000000000000000000000000000000000000000000000000";
```

```
            when "0111111110" => sal_dec_hsync <=
"1000000000000000000000000000000000000000000000000";
            when "0111111111" => sal_dec_hsync <=
"1000000000000000000000000000000000000000000000000";
            when "1000000000" => sal_dec_hsync <=
"1000000000000000000000000000000000000000000000000";
            when "1000000001" => sal_dec_hsync <=
"1000000000000000000000000000000000000000000000000";
            when "1000000010" => sal_dec_hsync <=
"1000000000000000000000000000000000000000000000000";
            when "1000000011" => sal_dec_hsync <=
"1000000000000000000000000000000000000000000000000";
            when "1000000100" => sal_dec_hsync <=
"1000000000000000000000000000000000000000000000000";
            when "1000000101" => sal_dec_hsync <=
"1000000000000000000000000000000000000000000000000";
            when others  => sal_dec_hsync <= (others => '0');
            end case;
      end process decode;

end arq_dec_hsync;
```

Codificador para habilitar flip-flop de 64 bits

```
-- VHDL Codificador para habilitar el flip-flop de 64 bits
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que genera la señal que habilita el flip-flop
-- de 64 bits, el cual almacena un vector completo con la
-- señal que activa las columnas de la pantalla para
-- desplegar imágenes desde VGA

----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------------

-- Declaración de la entidad
entity dec_hsync_ff is
      port(
            cnt_hsync_ff     : in std_logic_vector(9 downto 0);
            sal_dec_hsync_ff : out std_logic
      );
end dec_hsync_ff;

----------------------------------------------------

-- Declaración de la arquitectura
architecture arq_dec_hsync_ff of dec_hsync_ff is

begin

      decode: process (cnt_hsync_ff)
      begin
            case cnt_hsync_ff is
            when "0000100110" => sal_dec_hsync_ff <= '1';
            when "0000110000" => sal_dec_hsync_ff <= '1';
            when "0000111010" => sal_dec_hsync_ff <= '1';
            when "0001000100" => sal_dec_hsync_ff <= '1';
            when "0001001110" => sal_dec_hsync_ff <= '1';
```

```
            when "0001011000" => sal_dec_hsync_ff <= '1';
            when "0001100010" => sal_dec_hsync_ff <= '1';
            when "0001101100" => sal_dec_hsync_ff <= '1';
            when "0001110110" => sal_dec_hsync_ff <= '1';
            when "0010000000" => sal_dec_hsync_ff <= '1';
            when "0010001010" => sal_dec_hsync_ff <= '1';
            when "0010010100" => sal_dec_hsync_ff <= '1';
            when "0010011110" => sal_dec_hsync_ff <= '1';
            when "0010101000" => sal_dec_hsync_ff <= '1';
            when "0010110010" => sal_dec_hsync_ff <= '1';
            when "0010111100" => sal_dec_hsync_ff <= '1';
            when "0011000110" => sal_dec_hsync_ff <= '1';
            when "0011010000" => sal_dec_hsync_ff <= '1';
            when "0011011010" => sal_dec_hsync_ff <= '1';
            when "0011100100" => sal_dec_hsync_ff <= '1';
            when "0011101110" => sal_dec_hsync_ff <= '1';
            when "0011111000" => sal_dec_hsync_ff <= '1';
            when "0100000010" => sal_dec_hsync_ff <= '1';
            when "0100001100" => sal_dec_hsync_ff <= '1';
            when "0100010110" => sal_dec_hsync_ff <= '1';
            when "0100100000" => sal_dec_hsync_ff <= '1';
            when "0100101010" => sal_dec_hsync_ff <= '1';
            when "0100110100" => sal_dec_hsync_ff <= '1';
            when "0100111110" => sal_dec_hsync_ff <= '1';
            when "0101001000" => sal_dec_hsync_ff <= '1';
            when "0101010010" => sal_dec_hsync_ff <= '1';
            when "0101011100" => sal_dec_hsync_ff <= '1';
            when "0101100110" => sal_dec_hsync_ff <= '1';
            when "0101110000" => sal_dec_hsync_ff <= '1';
            when "0101111010" => sal_dec_hsync_ff <= '1';
            when "0110000100" => sal_dec_hsync_ff <= '1';
            when "0110001110" => sal_dec_hsync_ff <= '1';
            when "0110011000" => sal_dec_hsync_ff <= '1';
            when "0110100010" => sal_dec_hsync_ff <= '1';
            when "0110101100" => sal_dec_hsync_ff <= '1';
            when "0110110110" => sal_dec_hsync_ff <= '1';
            when "0111000000" => sal_dec_hsync_ff <= '1';
            when "0111001010" => sal_dec_hsync_ff <= '1';
            when "0111010100" => sal_dec_hsync_ff <= '1';
            when "0111011110" => sal_dec_hsync_ff <= '1';
            when "0111101000" => sal_dec_hsync_ff <= '1';
            when "0111110010" => sal_dec_hsync_ff <= '1';
            when "0111111100" => sal_dec_hsync_ff <= '1';
            when others  => sal_dec_hsync_ff <= '0';
            end case;
      end process decode;

end arq_dec_hsync_ff;
```

Contador de pulsos de 25MHz

```
-- VHDL Contador para pulsos de 25MHz
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que cuenta pulsos de 25MHz y representa
-- el valor de conteo en una señal de 10 bit, pero
-- además su salida es reestablecida con un pulso
-- bajo de HSync

-------------------------------------------------------

library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

----------------------------------------------------

-- Declaración de la entidad
entity cont_dataclk is
      port(
            clk,rst,hsync : in std_logic;
            conta           : out std_logic_vector(9 downto 0)
      );
end cont_dataclk;

----------------------------------------------------

-- Declaración de la arquitectura
architecture arq_cont_dataclk of cont_dataclk is

-- Señal con el valor del conteo
signal cuenta: std_logic_vector(9 downto 0);

begin

      process (clk,rst,hsync)
      begin
            if rst = '1' then
                  cuenta <= (others=>'0');
            elsif (clk'event and clk= '1') then
                  if hsync = '0' then
                        cuenta <= (others=>'0');
                  else cuenta <= cuenta + 1;
                  end if;
            end if;
      end process;

conta <= cuenta;

end arq_cont_dataclk;
```

Codificador para habilitar flip-flops de 1 bit

```
-- VHDL Codificador para habilitar flip-flops de 1 bit
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que genera la señal habilita los flip-flops
-- de un bit, para que almacenen los pixeles de cada
-- línea de imagen que corresponden a los que se
-- desean muestrear

----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------------

-- Declaración de la entidad
entity dec_dataclk is
      port(
            cnt_dataclk     : in std_logic_vector(9 downto 0);
            sal_dec_dataclk : out std_logic_vector(63 downto 0)
```

```
      );
end dec_dataclk;

-------------------------------------------------

-- Declaración de la arquitectura
architecture arq_dec_dataclk of dec_dataclk is

begin

      decode: process (cnt_dataclk)
      begin
            case cnt_dataclk is
            when "0000111000" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000000000000001";
            when "0001000010" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000000000000010";
            when "0001001100" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000000000000100";
            when "0001010110" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000000000001000";
            when "0001100000" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000000000010000";
            when "0001101010" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000000000100000";
            when "0001110100" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000000001000000";
            when "0001111110" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000000010000000";
            when "0010001000" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000000100000000";
            when "0010010010" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000001000000000";
            when "0010011100" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000010000000000";
            when "0010100110" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000000100000000000";
            when "0010110000" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000001000000000000";
            when "0010111010" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000010000000000000";
            when "0011000100" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000000100000000000000";
            when "0011001110" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000001000000000000000";
            when "0011011000" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000010000000000000000";
            when "0011100010" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000000100000000000000000";
            when "0011101100" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000001000000000000000000";
            when "0011110110" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000010000000000000000000";
            when "0100000000" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000000100000000000000000000";
            when "0100001010" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000001000000000000000000000";
            when "0100010100" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000010000000000000000000000";
            when "0100011110" => sal_dec_dataclk <=
"00000000000000000000000000000000000000000000000000000000100000000000000000000000";
```

```
            when "0100101000" => sal_dec_dataclk <=
"00000000000000000000000000000000000000001000000000000000000000000";
            when "0100110010" => sal_dec_dataclk <=
"00000000000000000000000000000000000000010000000000000000000000000";
            when "0100111100" => sal_dec_dataclk <=
"00000000000000000000000000000000000000100000000000000000000000000";
            when "0101000110" => sal_dec_dataclk <=
"00000000000000000000000000000000000001000000000000000000000000000";
            when "0101010000" => sal_dec_dataclk <=
"00000000000000000000000000000000000010000000000000000000000000000";
            when "0101011010" => sal_dec_dataclk <=
"00000000000000000000000000000000000100000000000000000000000000000";
            when "0101100100" => sal_dec_dataclk <=
"00000000000000000000000000000000001000000000000000000000000000000";
            when "0101101110" => sal_dec_dataclk <=
"00000000000000000000000000000000010000000000000000000000000000000";
            when "0101111000" => sal_dec_dataclk <=
"00000000000000000000000000000000100000000000000000000000000000000";
            when "0110000010" => sal_dec_dataclk <=
"00000000000000000000000000000001000000000000000000000000000000000";
            when "0110001100" => sal_dec_dataclk <=
"00000000000000000000000000000010000000000000000000000000000000000";
            when "0110010110" => sal_dec_dataclk <=
"00000000000000000000000000000100000000000000000000000000000000000";
            when "0110100000" => sal_dec_dataclk <=
"00000000000000000000000000001000000000000000000000000000000000000";
            when "0110101010" => sal_dec_dataclk <=
"00000000000000000000000000010000000000000000000000000000000000000";
            when "0110110100" => sal_dec_dataclk <=
"00000000000000000000000000100000000000000000000000000000000000000";
            when "0110111110" => sal_dec_dataclk <=
"00000000000000000000000001000000000000000000000000000000000000000";
            when "0111001000" => sal_dec_dataclk <=
"00000000000000000000000010000000000000000000000000000000000000000";
            when "0111010010" => sal_dec_dataclk <=
"00000000000000000000000100000000000000000000000000000000000000000";
            when "0111011100" => sal_dec_dataclk <=
"00000000000000000000001000000000000000000000000000000000000000000";
            when "0111100110" => sal_dec_dataclk <=
"00000000000000000000010000000000000000000000000000000000000000000";
            when "0111110000" => sal_dec_dataclk <=
"00000000000000000000100000000000000000000000000000000000000000000";
            when "0111111010" => sal_dec_dataclk <=
"00000000000000000001000000000000000000000000000000000000000000000";
            when "1000000100" => sal_dec_dataclk <=
"00000000000000000010000000000000000000000000000000000000000000000";
            when "1000001110" => sal_dec_dataclk <=
"00000000000000000100000000000000000000000000000000000000000000000";
            when "1000011000" => sal_dec_dataclk <=
"00000000000000001000000000000000000000000000000000000000000000000";
            when "1000100010" => sal_dec_dataclk <=
"00000000000000010000000000000000000000000000000000000000000000000";
            when "1000101100" => sal_dec_dataclk <=
"00000000000000100000000000000000000000000000000000000000000000000";
            when "1000110110" => sal_dec_dataclk <=
"00000000000001000000000000000000000000000000000000000000000000000";
            when "1001000000" => sal_dec_dataclk <=
"00000000000010000000000000000000000000000000000000000000000000000";
            when "1001001010" => sal_dec_dataclk <=
"00000000000100000000000000000000000000000000000000000000000000000";
```

```
            when "1001010100" => sal_dec_dataclk <=
"00000000010000000000000000000000000000000000000000000000000000000000000000";
            when "1001011110" => sal_dec_dataclk <=
"00000000100000000000000000000000000000000000000000000000000000000000000000";
            when "1001101000" => sal_dec_dataclk <=
"00000001000000000000000000000000000000000000000000000000000000000000000000";
            when "1001110010" => sal_dec_dataclk <=
"00000010000000000000000000000000000000000000000000000000000000000000000000";
            when "1001111100" => sal_dec_dataclk <=
"00000100000000000000000000000000000000000000000000000000000000000000000000";
            when "1010000110" => sal_dec_dataclk <=
"00001000000000000000000000000000000000000000000000000000000000000000000000";
            when "1010010000" => sal_dec_dataclk <=
"00010000000000000000000000000000000000000000000000000000000000000000000000";
            when "1010011010" => sal_dec_dataclk <=
"00100000000000000000000000000000000000000000000000000000000000000000000000";
            when "1010100100" => sal_dec_dataclk <=
"01000000000000000000000000000000000000000000000000000000000000000000000000";
            when "1010101110" => sal_dec_dataclk <=
"10000000000000000000000000000000000000000000000000000000000000000000000000";
            when others  => sal_dec_dataclk <= (others => '0');
            end case;
      end process decode;

end arq_dec_dataclk;
```

MÓDULO DE MEMORIA

Flip-flop de 1 bit

```
-- VHDL Flip-flop de 1 bit
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que funciona como un flip-flop tipo D,
-- es decir, su salida toma el valor de la entrada
-- solo cuando la señal de habilitación está en
-- estado alto

-------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

-------------------------------------------------------

-- Declaración de la entidad
entity ffd1 is
      port(
            enable,d : in std_logic;
            q        : out std_logic
      );
end ffd1;

-------------------------------------------------------

-- Declaración de la arquitectura
architecture arq_ffd1 of ffd1 is

begin

      process (enable, d)
```

```
      begin
            if enable='1' then
                  q <= d;
            end if;
      end process;

end arq_ffd1;
```

Flip-flop de 64 bits

```
-- VHDL Flip-flop de 64 bits
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que funciona como un flip-flop tipo D,
-- es decir, su salidas toman el valor de las entradas
-- solo cuando la señal de habilitación está en
-- estado alto

----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------------

-- Declaración de la entidad
entity ffd64 is
      port(
            enable : in std_logic;
            d      : in std_logic_vector(63 downto 0);
            q      : out std_logic_vector(63 downto 0)
      );
end ffd64;

----------------------------------------------------

-- Declaración de la arquitectura
architecture arq_ffd64 of ffd64 is

begin

      process (enable, d)
      begin
            if enable='1' then
                  q <= d;
            end if;
      end process;

end arq_ffd64;
```

MÓDULO GENERADOR DE PANTALLA “NO SIGNAL”

Programa principal

```
-- VHDL Generador de pantalla "NO SIGNAL"
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que conecta 1 contador y dos codificadores
-- para generar la imagen con el mensaje "NO SIGNAL"
-- en la pantalla

----------------------------------------------------
```

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

----------------------------------------------------

--Declaración de la entidad
entity pruebaleds is
      port(
            slow_clk,rst : in std_logic;
            fil          : out std_logic_vector(47 downto 0);
            col          : out std_logic_vector(63 downto 0)
      );
end pruebaleds;

----------------------------------------------------

--Declaración de la arquitectura
architecture arq_prueba of pruebaleds is

--Declaración de los componentes a usar

component cont is
      port(
            pulsos,rst : in std_logic;
            conta      : out std_logic_vector(5 downto 0)
      );
end component;

component dec_fil is
      port(
            contador : in std_logic_vector(5 downto 0);
            salfil   : out std_logic_vector(47 downto 0)
      );
end component;

component dec_col is
      port(
            contador : in std_logic_vector(5 downto 0);
            salcol   : out std_logic_vector(63 downto 0)
      );
end component;

--Señal del contador al decodificador de columnas
signal salcont : std_logic_vector(5 downto 0);

begin

  -- Mapeo por nombre del Contador
icont : cont
      port map(
            pulsos            =>   slow_clk,
            rst               =>   rst,
            conta(5 downto 0) => salcont
      );

--Mapeo por nombre del decodificador de filas
idecfilas : dec_fil
      port map(
```

```
            contador => salcont,
            salfil   => fil
      );

-- Mapeo por nombre del decodificador de columnas
ideccolumnas : dec_col
      port map(
            contador => salcont,
            salcol   => col
      );

end arq_prueba;
```

Contador de pulsos de 24KHz

```
-- VHDL Contador para pulsos de 24KHz
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que cuenta pulsos de 25MHz y representa
-- el valor de conteo en una señal de 10 bit

----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

----------------------------------------------------

-- Declaración de la entidad
entity cont is
      port(
            pulsos,rst : in std_logic;
            conta      : out std_logic_vector(5 downto 0)
      );
end cont;

----------------------------------------------------

-- Declaración de la arquitectura
architecture arq_cont of cont is

-- Señal con el valor del conteo
signal cuenta: std_logic_vector(5 downto 0);

begin

      process (pulsos,rst)
      begin
            if rst = '1' then
                  cuenta <= (others=>'0');
            elsif (pulsos'event and pulsos='1') then
                  cuenta <= cuenta + 1;
            end if;
      end process;

conta <= cuenta;

end arq_cont;
```

Codificador para columnas

```
-- VHDL Codificador para activar columnas en pantalla
-- "NO SIGNAL"
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa en cada valor de entrada, envía un vector
-- para activar las columnas de cada línea, y formar
-- la imagen con el mensaje "NO SIGNAL"

----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------------

-- Declaración de la entidad
entity dec_col is
      port(
            contador : in std_logic_vector(5 downto 0);
            salcol   : out std_logic_vector(63 downto 0)
      );
end dec_col;

----------------------------------------------------

-- Declaración de la arquitectura
architecture arq_dec_col of dec_col is

begin

      decode: process (contador)
      begin
            case contador is
            when "000000" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
            when "000001" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
            when "000010" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
            when "000011" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
            when "000100" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";--5
            when "000101" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
            when "000110" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
            when "000111" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
            when "001000" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
            when "001001" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";--10
            when "001010" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
            when "001011" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
            when "001100" => salcol <=
"0000000000000000000000000000000000000000000000000000000000000000";
```

```
            when "001101" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "001110" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";--15
            when "001111" => salcol <=
"00000000000000000000111111100000000110000001110000000000000000000";
            when "010000" => salcol <=
"00000000000000000000111000111000000011000001111000000000000000000";
            when "010001" => salcol <=
"00000000000000000001100000011000000110000110110000000000000000000";
            when "010010" => salcol <=
"00000000000000000001100000011000000110001100110000000000000000000";
            when "010011" => salcol <=
"00000000000000000001100000011000000110011000110000000000000000000";--20
            when "010100" => salcol <=
"00000000000000000001100000011000000110110000110000000000000000000";
            when "010101" => salcol <=
"00000000000000000001110001110000000111100000110000000000000000000";
            when "010110" => salcol <=
"00000000000000000001111111100000000111000000110000000000000000000";
            when "010111" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "011000" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";--25
            when "011001" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "011010" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "011011" => salcol <=
"00000000000000001001111100100010011111001110011111000000000000000";
            when "011100" => salcol <=
"00000000000000001001000100100110000001000100000001000000000000000";
            when "011101" => salcol <=
"00000000000000001001111100101010011101000100011111000000000000000";--30
            when "011110" => salcol <=
"00000000000000001001000100110010010001000100010000000000000000000";
            when "011111" => salcol <=
"00000000000001111100100010010001001111001110011111000000000000000";
            when "100000" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "100001" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "100010" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";--35
            when "100011" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "100100" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "100101" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "100110" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "100111" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";--40
            when "101000" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "101001" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "101010" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
```

```
            when "101011" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "101100" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";--45
            when "101101" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "101110" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when "101111" => salcol <=
"00000000000000000000000000000000000000000000000000000000000000000";
            when others  => salcol <= (others => '0');
            end case;
      end process decode;

end arq_dec_col;
```

Codificador para filas

```
-- VHDL Codificador para activar filas en pantalla
-- "NO SIGNAL"
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa en cada valor de entrada, activa cada una
-- de las filas de pantalla, para generar la imagen
-- con el mensaje "NO SIGNAL"

----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------------

-- Declaración de la entidad
entity dec_fil is
      port(
            contador : in std_logic_vector(5 downto 0);
            salfil   : out std_logic_vector(47 downto 0)
      );
end dec_fil;

----------------------------------------------------

-- Declaración de la arquitectura
architecture arq_dec_fil of dec_fil is

begin

      decode: process (contador)
      begin
            case contador is
            when "000000" => salfil <=
"00000000000000000000000000000000000000000000000000000001";
            when "000001" => salfil <=
"00000000000000000000000000000000000000000000000000000010";
            when "000010" => salfil <=
"00000000000000000000000000000000000000000000000000000100";
            when "000011" => salfil <=
"00000000000000000000000000000000000000000000000000001000";
            when "000100" => salfil <=
"00000000000000000000000000000000000000000000000000010000";
```

```
            when "000101" => salfil <=
"00000000000000000000000000000000000000000100000";
            when "000110" => salfil <=
"00000000000000000000000000000000000000001000000";
            when "000111" => salfil <=
"00000000000000000000000000000000000000010000000";
            when "001000" => salfil <=
"00000000000000000000000000000000000000100000000";
            when "001001" => salfil <=
"00000000000000000000000000000000000001000000000";
            when "001010" => salfil <=
"00000000000000000000000000000000000010000000000";
            when "001011" => salfil <=
"00000000000000000000000000000000000100000000000";
            when "001100" => salfil <=
"00000000000000000000000000000000001000000000000";
            when "001101" => salfil <=
"00000000000000000000000000000000010000000000000";
            when "001110" => salfil <=
"00000000000000000000000000000000100000000000000";
            when "001111" => salfil <=
"00000000000000000000000000000001000000000000000";
            when "010000" => salfil <=
"00000000000000000000000000000010000000000000000";
            when "010001" => salfil <=
"00000000000000000000000000000100000000000000000";
            when "010010" => salfil <=
"00000000000000000000000000001000000000000000000";
            when "010011" => salfil <=
"00000000000000000000000000010000000000000000000";
            when "010100" => salfil <=
"00000000000000000000000000100000000000000000000";
            when "010101" => salfil <=
"00000000000000000000000001000000000000000000000";
            when "010110" => salfil <=
"00000000000000000000000010000000000000000000000";
            when "010111" => salfil <=
"00000000000000000000000100000000000000000000000";
            when "011000" => salfil <=
"00000000000000000000001000000000000000000000000";
            when "011001" => salfil <=
"00000000000000000000010000000000000000000000000";
            when "011010" => salfil <=
"00000000000000000000100000000000000000000000000";
            when "011011" => salfil <=
"00000000000000000001000000000000000000000000000";
            when "011100" => salfil <=
"00000000000000000010000000000000000000000000000";
            when "011101" => salfil <=
"00000000000000000100000000000000000000000000000";
            when "011110" => salfil <=
"00000000000000001000000000000000000000000000000";
            when "011111" => salfil <=
"00000000000000010000000000000000000000000000000";
            when "100000" => salfil <=
"00000000000000100000000000000000000000000000000";
            when "100001" => salfil <=
"00000000000001000000000000000000000000000000000";
            when "100010" => salfil <=
"00000000000010000000000000000000000000000000000";
```

```
            when "100011" => salfil <=
"00000000000010000000000000000000000000000000000000";
            when "100100" => salfil <=
"00000000000100000000000000000000000000000000000000";
            when "100101" => salfil <=
"00000000001000000000000000000000000000000000000000";
            when "100110" => salfil <=
"00000000010000000000000000000000000000000000000000";
            when "100111" => salfil <=
"00000000100000000000000000000000000000000000000000";
            when "101000" => salfil <=
"00000001000000000000000000000000000000000000000000";
            when "101001" => salfil <=
"00000010000000000000000000000000000000000000000000";
            when "101010" => salfil <=
"00000100000000000000000000000000000000000000000000";
            when "101011" => salfil <=
"00001000000000000000000000000000000000000000000000";
            when "101100" => salfil <=
"00010000000000000000000000000000000000000000000000";
            when "101101" => salfil <=
"00100000000000000000000000000000000000000000000000";
            when "101110" => salfil <=
"01000000000000000000000000000000000000000000000000";
            when "101111" => salfil <=
"10000000000000000000000000000000000000000000000000";
            when others  => salfil <= (others => '0');
            end case;
      end process decode;

end arq_dec_fil;
```

MÓDULO DE SELECCIÓN

Contador de pulsos de 25MHz

```
-- VHDL Contador para pulsos de 25MHz
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que cuenta pulsos de 25MHz y representa
-- el valor de conteo en una señal de 10 bit, y su
-- salida es reestablecida con un pulso alto de HSync

----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

----------------------------------------------------

-- Declaracion de la entidad
entity cont_clk is
      port(
            clk,rst,hsync : in std_logic;
            conta           : out std_logic_vector(9 downto 0)
      );
end cont_clk;

----------------------------------------------------
```

```
-- Declaracion de la arquitectura
architecture arq_cont_clk of cont_clk is

-- Señal con el valor del conteo
signal cuenta: std_logic_vector(9 downto 0);

begin

      process (clk,rst,hsync)
      begin
            if rst = '1' then
                  cuenta <= (others=>'0');
            elsif (clk'event and clk= '1') then
                  if hsync = '1' then
                        cuenta <= (others=>'0');
                  else cuenta <= cuenta + 1;
                  end if;
            end if;
      end process;

conta <= cuenta;

end arq_cont_clk;
```

Comparador

```
-- VHDL Comparador
-- Hecho por Jesús Ávila A. 09/2007
--
-- Programa que pone en alto su salida, solo si su
-- entrada b es mayor a su entrada a.

----------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

----------------------------------------------------

-- Declaración de la entidad
entity comp is
      port(
            a,b : in std_logic_vector (9 downto 0);
            ma  : out std_logic
            );
end comp;

----------------------------------------------------

-- Declaración de la arquitectura
architecture simple of comp is

begin

      ma<='0' when (a>b) else '1';

end simple;
```

Multiprexor columnas

```
-- VHDL Multiprexor columnas
```

```
-- Hecho por Jesús Ávila A. 09/2007
--
-- Recibe las señales que habilitan las columnas para
-- generar imágenes desde VGA o la imagen con el
-- mensaje "NO SIGNAL", y con una señal de selección
-- decide cual de las dos señales enviar a la pantalla

----------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

----------------------------------------------------

-- Declaración de la entidad
entity MUX64 is
      Port(
            vgacol, NScol : in std_logic_vector(63 downto 0);
            S             : in std_logic;
            Y             : out std_logic_vector(63 downto 0)
      );
end MUX64;

----------------------------------------------------

-- Declaración de la arquitectura
architecture simple of MUX64 is

begin

      process(S,vgacol,nscol)
      begin
            case S is
            when '1'=>Y<=vgacol;
            when others =>Y<=NScol;
            end case;
      end process;

end simple;
```

Multiplexor para filas

```
-- VHDL Multiprexor filas
-- Hecho por Jesús Ávila A. 09/2007
--
-- Recibe las señales que habilitan las filas para
-- generar imágenes desde VGA o la imagen con el
-- mensaje "NO SIGNAL", y con una señal de selección
-- decide cual de las dos señales enviar a la pantalla

----------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

----------------------------------------------------

-- Declaración de la entidad
```

```
entity MUX48 is
      Port(
            vgafil, NSfil : in std_logic_vector(47 downto 0);
            S             : in std_logic;
            Y             : out std_logic_vector(47 downto 0)
      );
end MUX48;

----------------------------------------------------

-- Declaración de la arquitectura
architecture simple of MUX48 is

begin

      process(S,vgafil,nsfil)
      begin
            case S is
            when '1'=>Y<=vgafil;
            when others =>Y<=NSfil;
            end case;
      end process;

end simple;
```

Archivo UCF

```
NET "CLK"         LOC="P182";
NET "RST"         LOC="P110";
NET "VSYNC"       LOC="P109";
NET "HSYNC"       LOC="P111";
NET "DATO"        LOC="P87";
NET "CLK_ADC"     LOC="P73";
NET sal_vert<0>   LOC="P114";
NET sal_vert<1>   LOC="P113";
NET sal_vert<2>   LOC="P116";
NET sal_vert<3>   LOC="P115";
NET sal_vert<4>   LOC="P121";
NET sal_vert<5>   LOC="P120";
NET sal_vert<6>   LOC="P123";
NET sal_vert<7>   LOC="P122";
NET sal_vert<8>   LOC="P126";
NET sal_vert<9>   LOC="P125";
NET sal_vert<10>  LOC="P129";
NET sal_vert<11>  LOC="P127";
NET sal_vert<12>  LOC="P132";
NET sal_vert<13>  LOC="P133";
NET sal_vert<14>  LOC="P134";
NET sal_vert<15>  LOC="P135";
NET sal_vert<16>  LOC="P136";
NET sal_vert<17>  LOC="P138";
NET sal_vert<18>  LOC="P139";
NET sal_vert<19>  LOC="P140";
NET sal_vert<20>  LOC="P141";
NET sal_vert<21>  LOC="P145";
NET sal_vert<22>  LOC="P146";
NET sal_vert<23>  LOC="P147";
NET sal_vert<24>  LOC="P148";
NET sal_vert<25>  LOC="P149";
NET sal_vert<26>  LOC="P150";
NET sal_vert<27>  LOC="P151";
NET sal_vert<28>  LOC="P152";
```

```
NET sal_vert<29> LOC="P160";
NET sal_vert<30> LOC="P161";
NET sal_vert<31> LOC="P162";
NET sal_vert<32> LOC="P163";
NET sal_vert<33> LOC="P164";
NET sal_vert<34> LOC="P165";
NET sal_vert<35> LOC="P166";
NET sal_vert<36> LOC="P167";
NET sal_vert<37> LOC="P168";
NET sal_vert<38> LOC="P169";
NET sal_vert<39> LOC="P173";
NET sal_vert<40> LOC="P174";
NET sal_vert<41> LOC="P175";
NET sal_vert<42> LOC="P176";
NET sal_vert<43> LOC="P178";
NET sal_vert<44> LOC="P179";
NET sal_vert<45> LOC="P180";
NET sal_vert<46> LOC="P100";
NET sal_vert<47> LOC="P102";
NET sal<0>       LOC="P23";
NET sal<1>       LOC="P22";
NET sal<2>       LOC="P21";
NET sal<3>       LOC="P20";
NET sal<4>       LOC="P18";
NET sal<5>       LOC="P17";
NET sal<6>       LOC="P16";
NET sal<7>       LOC="P15";
NET sal<8>       LOC="P11";
NET sal<9>       LOC="P10";
NET sal<10>      LOC="P9";
NET sal<11>      LOC="P8";
NET sal<12>      LOC="P7";
NET sal<13>      LOC="P6";
NET sal<14>      LOC="P5";
NET sal<15>      LOC="P4";
NET sal<16>      LOC="P3";
NET sal<17>      LOC="P206";
NET sal<18>      LOC="P205";
NET sal<19>      LOC="P204";
NET sal<20>      LOC="P203";
NET sal<21>      LOC="P202";
NET sal<22>      LOC="P201";
NET sal<23>      LOC="P200";
NET sal<24>      LOC="P199";
NET sal<25>      LOC="P198";
NET sal<26>      LOC="P194";
NET sal<27>      LOC="P193";
NET sal<28>      LOC="P192";
NET sal<29>      LOC="P191";
NET sal<30>      LOC="P189";
NET sal<31>      LOC="P188";
NET sal<32>      LOC="P44";
NET sal<33>      LOC="P43";
NET sal<34>      LOC="P42";
NET sal<35>      LOC="P41";
NET sal<36>      LOC="P40";
NET sal<37>      LOC="P36";
NET sal<38>      LOC="P35";
NET sal<39>      LOC="P34";
NET sal<40>      LOC="P33";
NET sal<41>      LOC="P31";
```

```
NET sal<42>      LOC="P30";
NET sal<43>      LOC="P29";
NET sal<44>      LOC="P27";
NET sal<45>      LOC="P24";
NET sal<46>      LOC="P71";
NET sal<47>      LOC="p70";
NET sal<48>      LOC="p69";
NET sal<49>      LOC="P68";
NET sal<50>      LOC="P64";
NET sal<51>      LOC="P63";
NET sal<52>      LOC="P62";
NET sal<53>      LOC="P61";
NET sal<54>      LOC="P60";
NET sal<55>      LOC="P59";
NET sal<56>      LOC="P58";
NET sal<57>      LOC="P57";
NET sal<58>      LOC="P56";
NET sal<59>      LOC="P55";
NET sal<60>      LOC="P49";
NET sal<61>      LOC="P48";
NET sal<62>      LOC="P47";
NET sal<63>      LOC="P46";
```

BIBLIOGRAFÍA

[1] *Richard C. Jaeger, Travis N. Blalock, Diseño de Circuitos Microelectrónicos, 2da. Edición McGrawHill.*

[2] *Robert F. Coughlin, Frederick F. Driscoll, Amplificadores Operacionales y Circuitos Integrados Lineales, pHH Prentice Hall*

[3] *Sergio Franco, Diseño con Amplificadores Operacionales y Circuitos Integrados Analógicos, 3ra. Edición, McGrawHill*

[4] *James W. Bignell, Robert L. Donovan, Electrónica Digital, CECSA*

[5] *Convertidor Analógico Digital ADC08100, National Semiconductor, Data Conversion, www.national.com*

[6] *Kevin Skahill, VHDL for Programmable Logic, Addison-Wesley*

[7] *Digilent D2-SB, Manual de Referencia, www.digilentinc.com*

[8] *Matriz de LEDs LTP-757G, LITE-ON Electronics, www.liteon.com*

[9] *John F. Wakerly, Diseño Digital Principios y Practicas, 3ra. Edición, Prentice Hall*

[10] *Sudhakar Yalamananchili, VHDL Starter's Guide, Prentice Hall*

[11] *Adel S. Sendra, Kenneth C. Smith, Circuitos Microelectronicos, 4ta. Edición, Oxford*

[12] *Transistor BJT 2N2222A, STMicroelectronics, www.st.com*

[13] *Transistor MOSFET SD5000, Calogic Corporation*

[14] *Edwards Stephen, Presentación Video CSEE W4840, Columbia University*

[15] *Murray Herrera Víctor, Santillán Quiñonez Gerard, Artículo Controlador de un monitor vga con resolución 640x480 sobre un cpld flex10k, Lima, Perú*