IMPLEMENTACIÓN FÍSICA DEL SISTEMA

5.1. INTRODUCCIÓN

Este capítulo, como su propio nombre indica, aborda la parte más física del proyecto: en primer lugar, el desarrollo de la PCB de la tarjeta de adquisición de las imágenes. En segundo lugar, la implementación hardware del sistema de visión. La fase de diseño y realización de la placa de circuito impreso o PCB es la más delicada del proyecto, ya que se debe cometer el menor número de errores posibles ya que repararlos resulta más costoso. La implementación hardware del sistema de visión se realiza sobre una FPGA de la familia Virtex, la *Virtex 300*.

5.2. REALIZACIÓN DEL PCB

Una vez decidido el diseño del sistema de adquisición de imágenes que se deseaba implementar, se procedió al diseño mediante técnicas CAD. Utilizando las

herramientas proporcionadas por ACCEL se dio comienzo al desarrollo del diseño de la placa de circuito impreso.

5.2.1. ESQUEMÁTICO

En el editor de esquemas ACCEL SCH se dibuja el esquema completo del circuito, donde aparecerán los distintos tipos de componentes así como la interconexión entre ellos.

Para realizar un diseño ordenado, en primer lugar se creará una **librería personalizada** donde se almacenarán todos los componentes que formen parte del circuito que se quiere diseñar. En dicha librería, a cada componente se le asociará un **símbolo** o esquemático y un **patrón** o huella. Además se definen todas las características mecánicas de los componentes, los tipos de encapsulado, números de pines, conexionado entre elementos, y todos aquellos aspectos relacionados con el esquema de un circuito (estilos de pads, pines, etc.). Es necesario pues, crear una librería (fichero **.lib*) donde se almacenen los componentes y sus correspondientes características. Para ello, dependiendo de cómo sea el componente, se tendrán dos posibilidades:

- Si dicho componente aparece en algunas de las librerías proporcionadas por ACCEL (incluidas en una carpeta denominada Lib) entonces sólo habría que copiarlo, junto con su símbolo y su patrón, desde esa librería a la librería personalizada.
- Si dicho componente no se encuentra en ninguna de las librerías que incluye ACCEL, o bien sólo aparece su símbolo y no su patrón o viceversa, será necesario crear los símbolos y patrones que posteriormente se asociarán al componente.

En nuestro caso, la mayor parte de los símbolos y patrones se obtuvieron de las librerías *Discrete.lib*, *Connect.lib* y *Pcbmain.lib* (incluidas en ACCEL). Sin embargo, para la captura del esquema, se tuvieron que crear una serie de componentes que no

existían en las librerías proporcionadas por ACCEL, como era el caso de los conectores de vídeo RCA JACK y S-VIDEO JACK. Haciendo uso de las aplicaciones *ACCEL Symbol Editor* y *ACCEL Pattern Editor* se diseñaron los patrones de dichos conectores de vídeo y además se crearon los símbolos de los circuitos integrados como el decodificador de vídeo SAA7113H. Una vez creados todos los componentes, se almacenaron en la librería principal del diseño, denominada *inma.lib*. (Para más detalles ver apartado I.2 del "Tutorial de ACCEL" que aparece en el anexo). Los símbolos y patrones utilizados en el diseño del circuito aparecen en las *figuras 5.1* y *5.2* respectivamente.

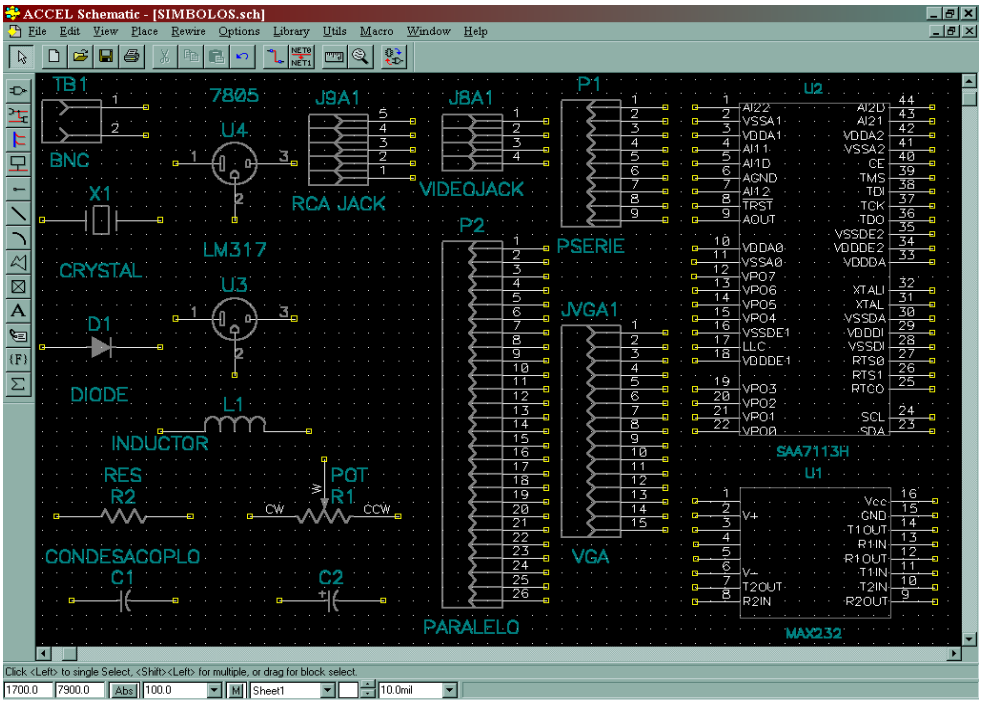


Figura 5.1: *Símbolos de los componentes del circuito*

Una vez que se tienen todos los componentes que se van a utilizar almacenados en la librería correspondiente, se procede a la colocación de los mismos en el esquemático y a la interconexión entre estos. Para ello se ejecutará el programa *ACCEL Schematic*. De este modo se obtiene el esquemático del circuito diseñado (ver *figura 5.3*).

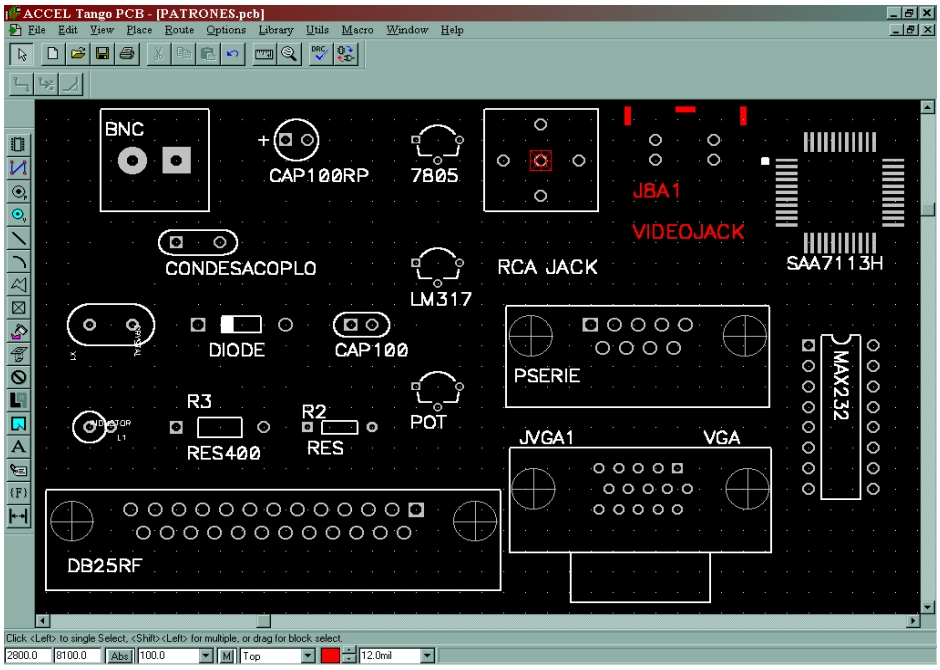


Figura 5.2: *Patrones o huellas de los componentes del circuito*

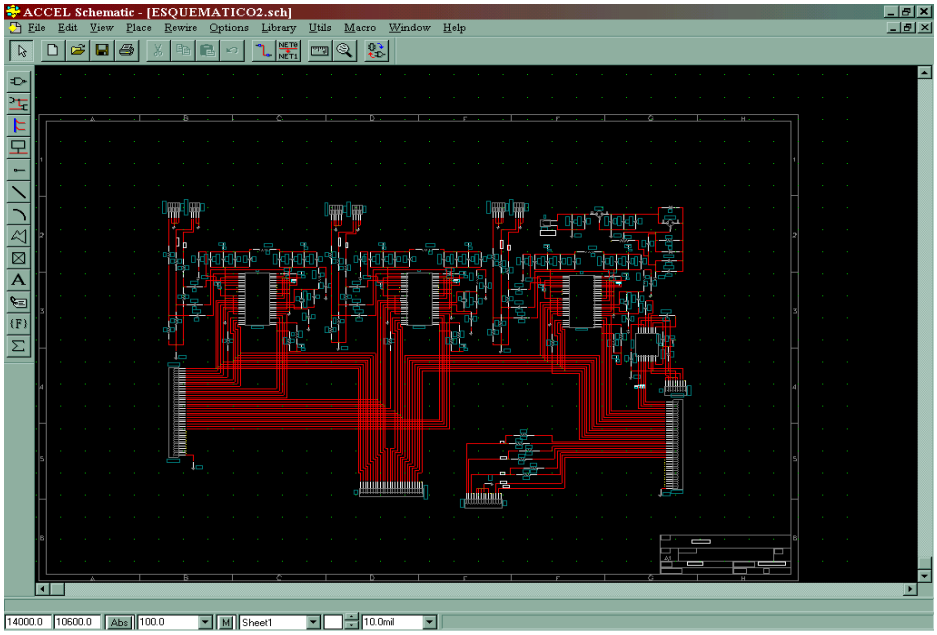


Figura 5.3: *Esquemático del circuito diseñado*

Después de realizar el esquemático y comprobar si existen o no errores de tipo eléctrico con ayuda de ERC, el siguiente paso es la obtención del llamado *Netlist*: un fichero **.net* que posteriormente se cargará en el editor de PCB.

5.2.2. EDITOR DE PCB: ACCEL TANGO

El editor de PCB o editor de placa de circuito impreso, permite el diseño de las conexiones o pistas entre los patrones o huellas, que representan los encapsulados de los componentes que integran el sistema.

Desde el editor de PCB (ACCEL Tango PCB), se cargará el fichero *Netlist*, generándose los patrones y la interconexión entre ellos además de los posibles errores y *warnings* producidos en el diseño. Los errores deben ser solucionados antes de continuar con el diseño, sin embargo se puede proseguir aunque exista algún *warning*.

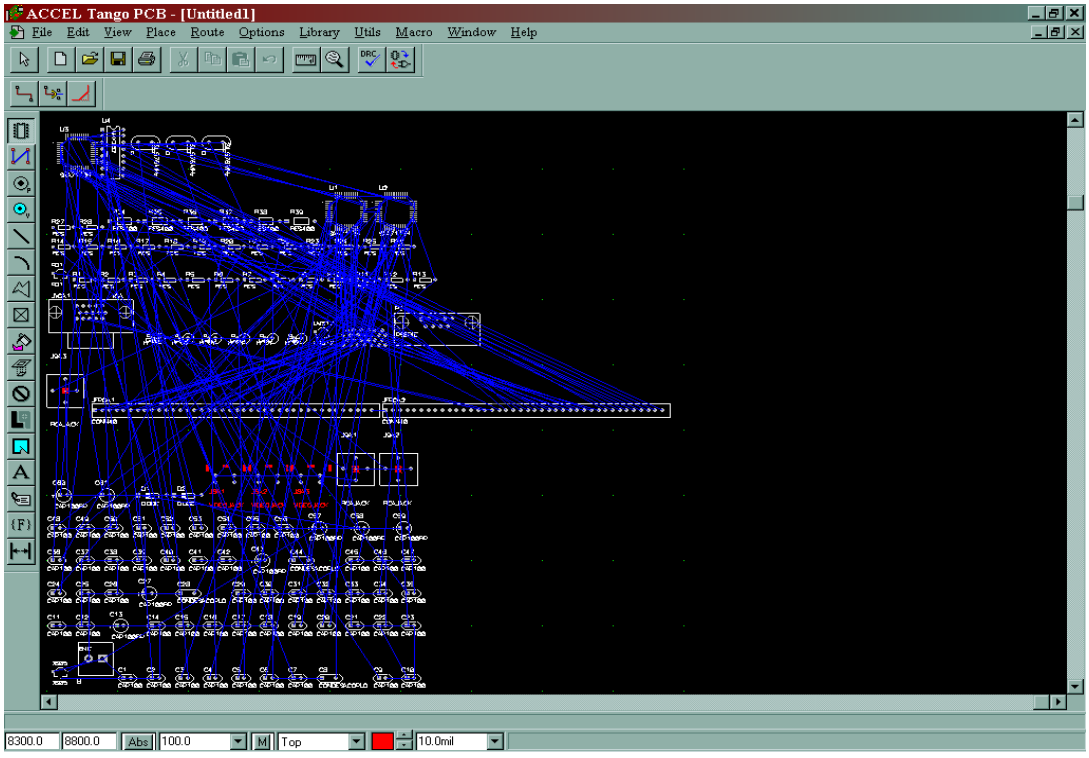


Figura 5.4: *Resultado de cargar el fichero Netlist*

5.2.2.1. EMPLAZAMIENTO DE COMPONENTES (*PLACEMENT*)

Mientras que en el esquemático no era necesario la colocación de los componentes en sus emplazamientos reales, en el editor de PCB sí lo es, ya que el diseño obtenido con el ACCEL Tango PCB será el que se imprima en papel vegetal para su posterior revelado (fotolito). Además se deben seguir con cautela unas determinadas normas de diseño de PCBs.

Muchos componentes deben estar a unas determinadas distancias del resto de los elementos para evitar la influencia no deseada de unos sobre otros. Además de guardar esas distancias mínimas, existen componentes que deben situarse cercanos a aquellos elementos con los que se conectarán. Tal es el caso de los *condensadores de desacoplo*, que deben ubicarse lo más cerca posible del CI que pretende proteger, entre la alimentación y tierra (en nuestro caso los CI son el MAX232 y los SAA7113H, que tal y como se observa en la *figura 5.6* están rodeados de condensadores de desacoplo).

Todos los componentes se colocarán paralelos a los bordes de la placa como se observa en la figura siguiente:

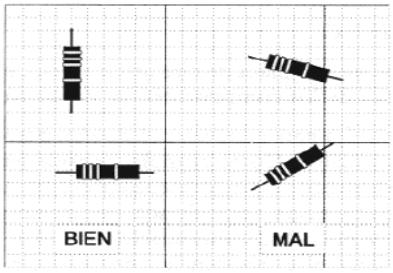


Figura 5.5: *Colocación de elementos vertical u horizontalmente*

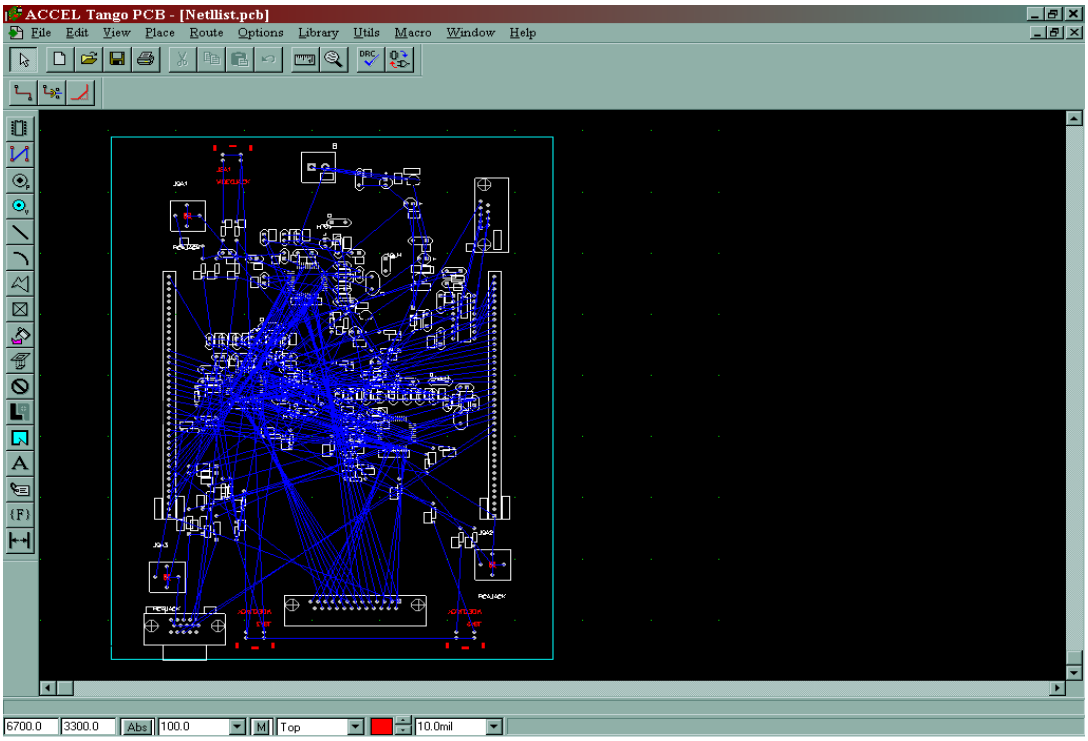


Figura 5.6: *Emplazamiento de los componentes*

5.2.2.2. TRAZADO DE LAS PISTAS (*ROUTING*)

Se denomina *Routing* al trazado del recorrido que tendrán las pistas por las que irán las señales eléctricas. Estas pistas son como cauces por los que discurren las señales eléctricas entre los distintos componentes. El rutado debe cumplir unas normas que se detallarán a continuación para evitar conflictos a la hora de la impresión de las capas en papel vegetal.

Las normas que se deben seguir a la hora de rutar un circuito impreso son:

- Distribución simétrica.
- Las pistas deberán formar un ángulo de 45º.

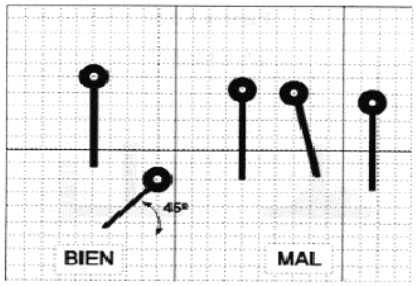
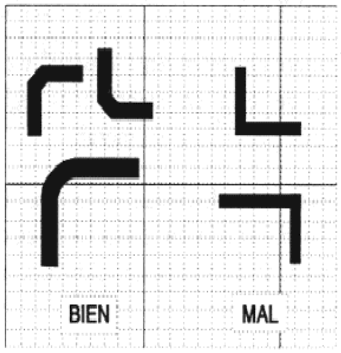
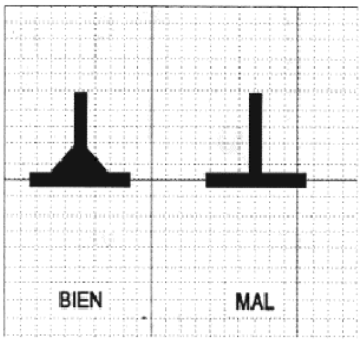


Figura 5.7: *Ángulo óptimo de las pistas*

- Evitar que las pistas formen ángulos rectos o agudos entre sí. Por ejemplo, para crear un ángulo de 90º se utilizarán dos ángulos de 135º (ver *figura 5.8*). Si es necesario ejecutar una bifurcación en una pista, se hará suavizando los ángulos con sendos triángulos a cada lado (*figura 5.9*).





Figuras 5.8 y 5.9: *Ángulos de pistas*

- Los puntos donde se realizarán las soldaduras serán círculos cuyo diámetro será al menos el doble del ancho de la pista que en él termina.
- La distancia mínima entre pistas y el borde de la placa será de 2 décimas de pulgada, aproximadamente 5 mm.
- Se debe dejar como norma general, una o dos décimas de pulgada de patilla entre le cuerpo de los componentes y el punto de soldadura correspondiente.

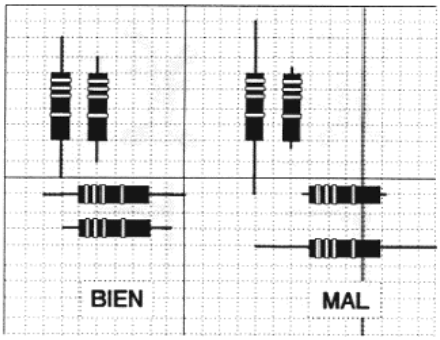
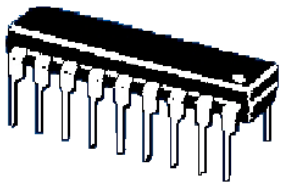


Figura 5.10: *Longitud de patillas de componentes*

Para el diseño del sistema de adquisición de datos fue necesaria la realización de una placa a dos caras debido al alto número de redes y elementos que se tenían que conectar.

Como ya se ha indicado, se tendrán dos capas o *layers*, el *Top* y el *Bottom*, por las que se podrán rutar y soldar los componentes. La mayoría de los patrones se definen por defecto en el *Top*, como es el caso de los circuitos integrados (CI). Los tipos fundamentales de encapsulados de CI son:

- De inserción (DIP = *Dual-In-Line Package*).



- De montaje superficial (SMT :Surface-Mount Technology; SMD : Surface-Mount Device).

En el diseño se tienen ambos tipos de CI:

- De inserción: MAX232.
- De montaje superficial: SAA7113H.

Es importante tener cuidado de qué pistas se rutan por cada capa, puesto que posteriormente se deberán soldar los componentes del sistema, y no todos pueden ir en

ambas capas indistintamente. Generalmente, los CI deben situarse en la capa *Top*. Sin embargo, mientras que los CI de inserción se pueden soldar por ambas capas, pudiéndose rutar por ambas las pistas, los de montaje superficial sólo se podrán soldar por el *Top*, debiendo rutar las pistas por el *Top*. Las pistas que salen de los CI deben hacerlo sin ángulo alguno, propiciando de esta forma que la superficie de unión entre el pin y la pista sea lo mayor posible.

Como se señaló anteriormente, la colocación de los patrones es muy importante, puesto que será la ubicación que finalmente tendrán en la placa. Por ejemplo, las bornas deben situarse los más cerca posible del borde de la placa, evitando así dificultar la entrada de cables para la inserción/extracción de señales.

Entre pistas próximas y entre pistas y puntos de soldadura, deberá existir una distancia que dependerá de la tensión eléctrica que se prevea exista entre ellas. Normalmente se dejará una distancia de 0,8 mm. En diseños complejos se podrá disminuir hasta 0,4 mm (*figura 5.11*).

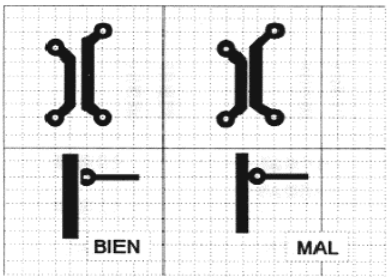


Figura 5.11: *Separación entre pistas*

En nuestro diseño, las distancias entre la mayoría de las pistas son algo superiores a los 0,8mm (1mm, 0,93mm) salvo en el caso de las pistas de los decodificadores de vídeo, donde la distancia media es de unos 0,41mm. El ancho de las pistas dependerá de la intensidad que vaya a circular por ellas. Un ancho de 0,8 mm, dependiendo del espesor de la pista, podrá soportar alrededor de 2 A, 2 mm soportará 5 A y 4,5 mm, unos 10 A.

Debido a que las corrientes del circuito son bajas, en el diseño la mayoría de las pistas son de 1mm *(39,4mil)* de grosor, aunque en determinadas zonas fue conveniente disminuirlos por estar las pistas demasiado cerca de pads, vías u otras pistas. Las pistas de alimentación y tierra tienen unos 2 mm *(78,7mil)*. Las pistas más problemáticas son las de los SAA7113H, con un ancho de 0,30 a 0,40mm *(11.8 mil- 15,7 mil)*.[17]

El trazado de las pistas puede realizarse de modo automático o manual. Para el modo automático *("Route Autorouters Pro route")* se pueden especificar al programa que trace las pistas en la capa "*Top*" horizontal o verticalmente, y las del "*Bottom*" a la inversa, siendo de esta manera el diseño algo más estructurado y sencillo a la hora de visualizar las pistas (ver *figura 5.12*).

Figura 5.12: *Rutado de pistas por "Top" horizontalmente (rojo) y de "Bottom" verticalmente (verde)*

En algunas ocasiones, sobre todo en el diseño de circuitos complejos, como el que se describe en el proyecto, el rutado automático no es capaz de resolver todas la conexiones (líneas azules). En estos casos es necesario continuar con el rutado manual

[17] *(0,64 mm =25 mil)*

(Route Manual) que aunque puede ser más tedioso y lento, se ajustará mejor a las características que se pretende tenga la placa. También se suele emplear cuando el número de vías obtenidas es demasiado elevado y se pretende disminuir, ya que éstas son posibles fuentes de errores. En nuestro caso, mediante el rutado manual, se consigue pasar de 233 vías iniciales a 66 en el PCB final.

Las vías se definirán según un estilo que se debe crear puesto que ofrece mejores prestaciones. Las características de las vías utilizadas en el diseño son las siguientes:

- *Type*: Tru.
- *Width*: 1.30mm.
- *Heigth*: 1.30mm.
- *Shape*:Ellipse.
- *Hole diameter*: 0.40mm.

Los *pads* empleados tienen las siguientes características:

- *Type*: Tru.
- *Width*: 1.60mm.
- *Heigth*: 1.60mm.
- *Shape*:Rounded Rectangle.
- *Hole diameter*: 0.40mm.

Para poder tener una idea de los resultados estadísticos del rutado se puede, mediante la opción *Design Info* del menú *File*, obtener el número de vías, redes, componentes, pads, etc.

- Dimensiones de la placa: 165.40mm x 217.47mm.
- Nº de componentes: 125.
- Nº de vías: 66.
- Nº de pads: 533.

Además, para verificar la correcta unión entre pistas y componentes, se puede hacer uso de la opción *Highlight* (ver apartado I.5.3 del "Tutorial de ACCEL", donde se analizan todas las posibilidades que ofrece Accel).

5.2.2.3. PCB Y FOTOLITO

Una vez que se ha concluido el diseño y se han comparado el *Netlist* del esquema y del PCB, se puede imprimir el ***fotolito***, que es una copia del circuito impreso, por una o las dos caras. Posteriormente, mediante las técnicas de serigrafía se obtendrá la placa sobre la que se montarán los componentes.

El PCB (*Top* y *Bottom*) de la placa resultante se muestra a continuación:

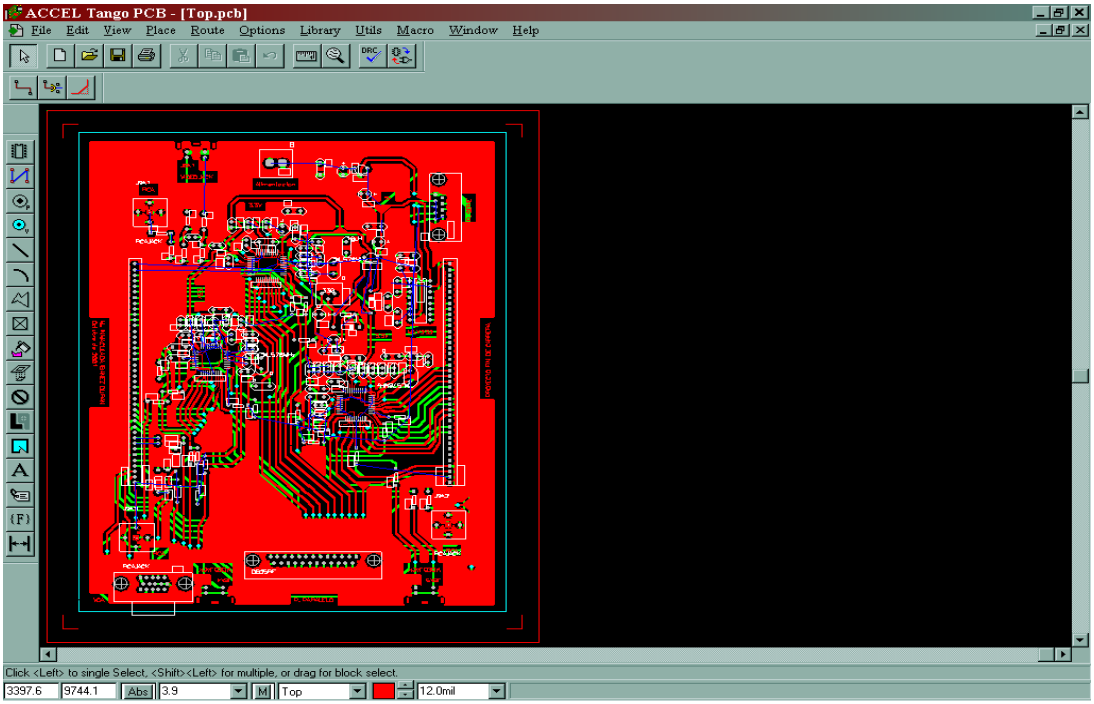


Figura 5.13: *Cara Top de la placa del circuito impreso*

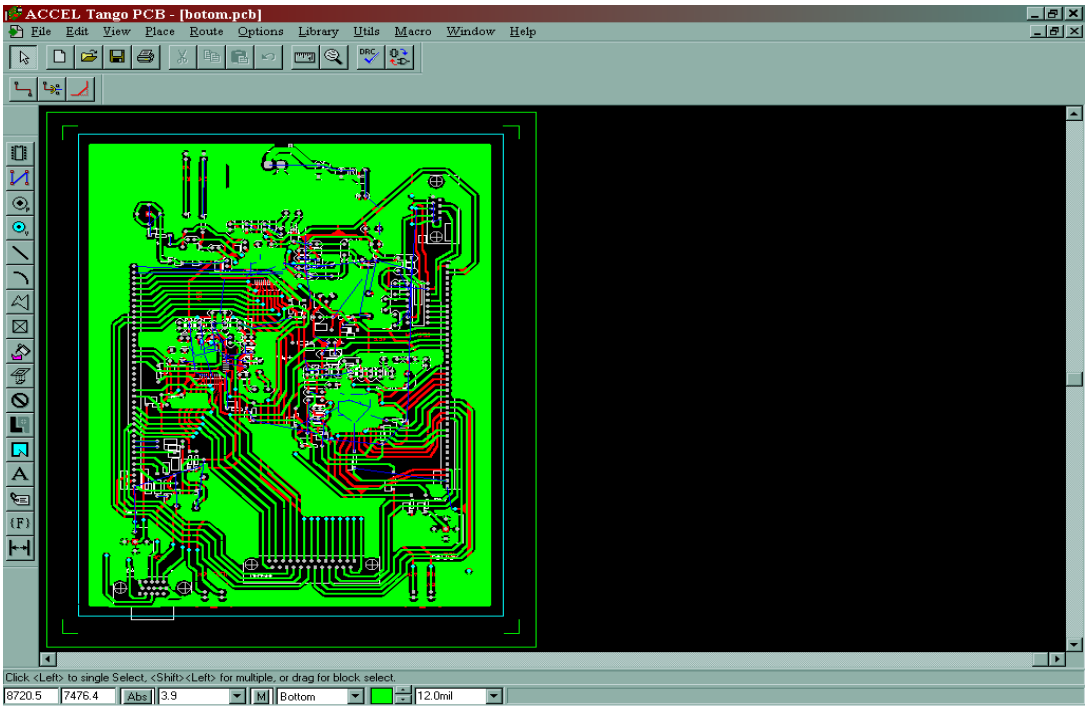


Figura 5.14: *Cara Bottom de la placa del circuito impreso*

5.2.2.3.1. Ficheros de Impresión

La *placa fotosensible* tiene un barniz que es sensible a la luz, que se impresiona mediante una insoladora o cualquier otro foco luminoso adecuado. Normalmente, es más sensible a la luz que contenga UVA (ultravioleta tipo A) que es el que tienen los rayos de sol.

Los ficheros de impresión son los que proporcionan en un papel vegetal, con una impresora láser, el **fotolito**, que es una copia del circuito impreso, por una o las dos caras.

Para la exposición, se prepara la transparencia de las pistas o **fotolito**, que puede ser en negativo o en positivo, aunque éste último es el más utilizado. Tras la exposición, se introduce la placa en un líquido revelador que destruirá el barniz que no forma parte de las pistas, de forma que el restante actúa de protector contra la corrosión.

En la placa positiva las partes insoladas perderán el barniz durante el proceso de revelado, y por tanto la transparencia será también positiva. En ella las pistas van en negro para proteger el barniz de la insolación, que en este caso permanecerá tras el proceso de revelado de la placa.

Para cada una de las capas interesa imprimir las pistas, las vías, pads y los pads/vias hole, además del borde (*board*). A la hora de imprimir hay que ser muy cuidadoso, hacerlo con calidad óptima y elegir un tipo de papel específico. Para ello se utilizó una impresora láser y papel vegetal.

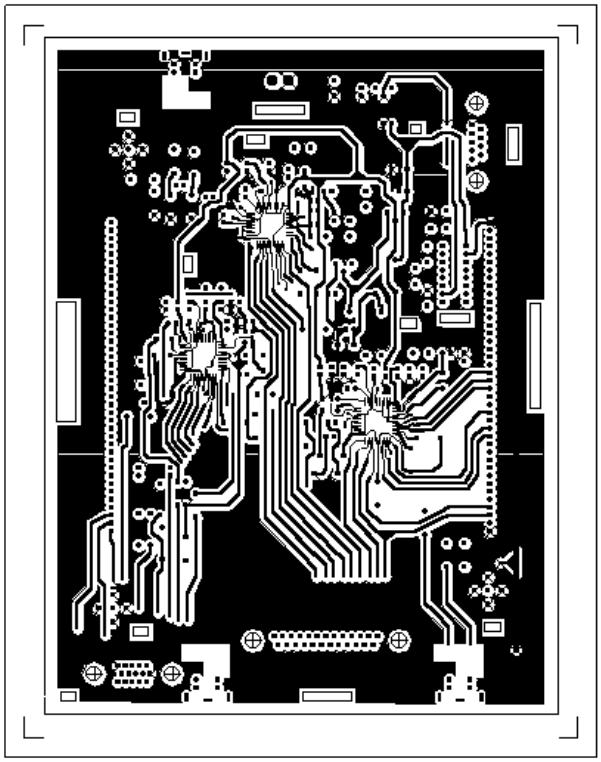


Figura 5.15: *Cara Top de la placa del circuito impreso*

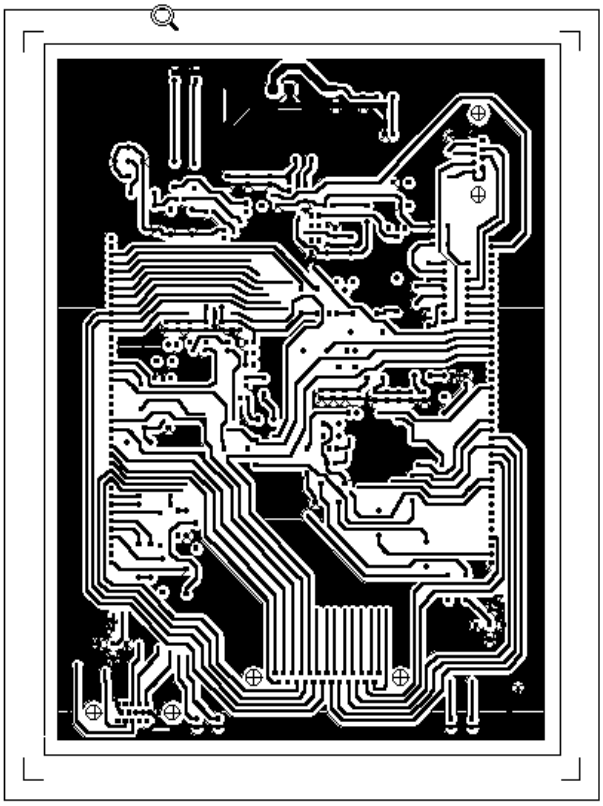


Figura 5.16: *Cara Bottom de la placa del circuito impreso*

La placa debe ser limpiada con alcohol muy bien para retirar los posibles residuos de resina que tenga.

El proceso de taladrar la placa debe realizarse con sumo cuidado y destreza, procurando que quede suficiente cobre alrededor del agujero del pad, ya que posteriormente al soldar el componente se debe aplicar estaño y dado que éste se fija al cobre, mientras más superficie de cobre exista, mejor será la conexión y menos probabilidades habrá de que dicho componente quede suelto.

Para taladrar las vías y los pads se utilizaron brocas de 0,8 mm, por lo general. También fue necesario emplear brocas de 1mm y 1,5mm para los pads de los conectores de vídeo y de la fuente de tensión.

5.2.3. MONTAJE DEL CIRCUITO

Una vez taladrada la placa, se lijará muy bien empleando una goma de borrar tinta o bien un estropajo no muy duro para no eliminar cobre. Posteriormente se soldarán los componentes siguiendo este orden:

1º Vías.

2º Circuitos de montaje superficial: decodificadores de vídeo SAA7113H.

3º Resistencias.

4º Condensadores de desacoplo (muy próximos a los SAA7113H).

5º Reguladores de tensión y conectores (VGA, puerto serie, puerto paralelo).

De este modo, se sueldan primero los componentes que quedan más próximos a la placa y en último lugar aquellos que tienen más "altura", para no dificultar así el proceso de soldadura. Las zonas de superficies muy amplias por ejemplo, plano de masa, suelen presentar problemas puesto que el calor aplicado con el soldador, para que se derrita el estaño, se disipa.

5.2.3.1. MÉTODOS APLICADOS PARA SOLDAR COMPONENTES

Hay que diferenciar dos métodos para soldar, puesto que en el diseño se tienen componentes de *montaje superficial* (decodificadores de vídeo) y de *inserción* (resistencias, condensadores, conectores,...)

5.2.3.1.1. Componentes SMT o de montaje superficial

Lo más conveniente para soldar este tipo de componentes es estañar la placa. De este modo, será suficiente con colocar el componente sobre sus pads y aplicar calor a

cada una de las patas para que se peguen al estaño. No se debe aplicar mucho calor ya que si el CI no tiene protección el posible que éste se funda y se tenga que cambiar.

Soldar los decodificadores de vídeo SAA7113H resultó bastante complejo ya que no se estañó la placa y no se disponía de un soldador de punta fina. Con mucha destreza, excelente vista y buen pulso se consiguieron soldar dichos CI.

5.2.3.1.2. Componentes THD o de inserción

En primer lugar, con el soldador se suministra calor tanto al pad como a la pata del componente. Luego se aplica un poco de estaño sobre el pad con el componente insertado. El estaño debe quedar conectando el pin con el cobre del pad, evitando la existencia de "globos" que no permiten la correcta conexión.

Para el CI de inserción del diseño: el MAX232, se optó por utilizar un *zócalo de pines torneados* que queda permanentemente soldado a la placa. De esta forma, si el CI no funcionase correctamente, se evitaría el engorro de tener que desoldarlo y soldarlo de nuevo, ya que al estar insertado sobre el zócalo, bastaría con extraerlo y cambiarlo.

5.2.4. REALIZACIÓN DE PRUEBAS

Las pruebas realizadas fueron las de comprobar:

- La continuidad de las pistas.
- La existencia de una única referencia de tierra.
- Que se tiene una alimentación adecuada en todos y cada uno de los componentes que la precisan.
- Las frecuencias de los relojes son las adecuadas.
- Las formas de onda de ciertas señales de control se corresponden con las que aparecen en los *datasheets*.

Montados ya todos los componentes en la placa, se debe comprobar la continuidad, tanto en las vías como en los pads, para localizar malas soldaduras. Para realizar esta prueba se dispone de un polímetro[18]. Este aparato emite un pitido si existe continuidad entre los dos puntos sometidos a estudio. Será fundamental realizar correctamente este estudio que facilitará la localización de posibles conexiones en mal estado, evitando posteriormente quebraderos de cabeza ante un mal funcionamiento.

Los planos de masa de ambas capas deben estar conectados entre ellos para proporcionar una misma referencia de tierra a todo el circuito. Por lo tanto, no deben quedar zonas aisladas o islas que no tengan la misma referencia que el resto del circuito. Todo esto también se comprueba con el polímetro.

Con ayuda del osciloscopio[19] y de la fuente de tensión[20] del laboratorio se miden los niveles de tensión. La placa se alimenta a 12V, al igual que las cámaras. A partir de esa tensión, mediante dos reguladores, se consiguen los 5V para el MAX232 y los 3.3V requeridos por los decodificadores SAA7113H. Las medidas obtenidas en el laboratorio aparecen en la siguiente tabla:

HM 8040-2 (V)	*LM 7805 (V)*	*LM 317 (V)*
5	*2.20*	*2.00*
6	*4.40*	*2.40*
7	*4.88*	*2.72*
8	*4.96*	*3.12*
9	*4.98*	*3.20*
10-11-12	*4.98*	*3.28*

Tabla 5.1: *Datos experimentales*

[18] F Freak my-63
[19] Tektronix TDS220 Two Channel Digital Real-Time Oscilloscope 100MHz 1GS/s
[20] Tripple Power Supply HM 8040-2

5.3. DISEÑO E IMPLEMENTACIÓN HARDWARE

5.3.1. INTRODUCCIÓN

Una vez obtenido un sistema de adquisición de imágenes operativo se comenzó su implementación hardware. Como primer paso se realizó una primera división del sistema en bloques y se evaluaron los recursos necesarios. El diseño se realizó según la metodología *top-down*, es decir, se realizaron divisiones sucesivas de los bloques hasta llegar a las unidades mínimas de diseño. Una vez claro el diseño se pasó a escribir el código VHDL. El depurado del código se realizó en paralelo a la escritura del código: chequeo de sintaxis, síntesis de bloques y simulación lógica, implementación y simulación temporal, también se realizaron pruebas sobre la placa. Utilizando un sistema de propósito general basado en FPGA's, llamado UNSHADES-1, se ha desarrollado y probado el sistema completo.

5.3.2. ENTORNO DE TRABAJO: UNSHADES-1

Toda la lógica que implementa la parte digital del sistema de posicionamiento del punto de soldadura se aloja en una FPGA, la Virtex modelo V300PQ240. Se trabaja con la placa del proyecto UNSHADES, ver *figura 5.17*, desarrollada por *M. A. Aguirre* del Departamento de Tecnología Electrónica de la Universidad de Sevilla y tutor de este proyecto, que permite explotar las nuevas prestaciones que ofrecen estas FPGAs de última generación del fabricante Xilinx.

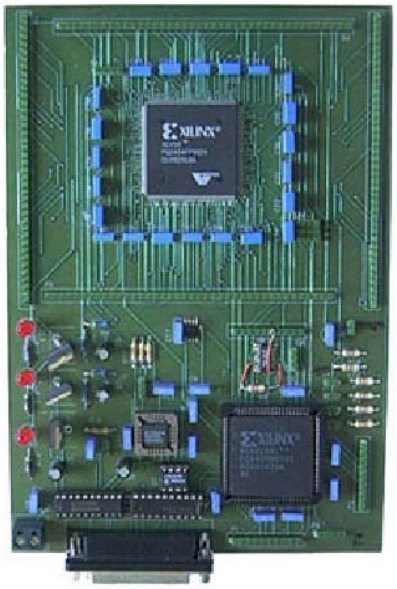


Figura 5.17: *Placa del proyecto UNSHADES-1*

Un depurador hardware es un sistema completamente programable que puede ser adaptado para diferentes aplicaciones. La principal ventaja es que el sistema completo puede ser monitorizado durante el tiempo de ejecución. Los procesadores principales son sustituidos por una FPGA avanzada y corren insertados en el mismo sistema. Si se desea construir un sistema totalmente adaptado la propia descripción del circuito sirve para construirlo.

UNSHADES-1 es un entorno de trabajo de propósito general que se basa en una FPGA avanzada del fabricante Xilinx para emular el procesador adaptado para, en este caso, el procesamiento de imagen. La principal ventaja del sistema UNSHADES es que el usuario tiene total control sobre la ejecución de la FPGA debido a las características especiales de este dispositivo. UNSHADES significa *Universidad de Sevilla Hardware Debugger System*, y permite un rápido prototipado y el depurado del circuito procesador durante la ejecución.

Los principales componentes del sistema UNSHADES se enuncian a continuación:

- Una FPGA de alta gama de la familia Virtex de Xilinx (desde la XCV50 hasta la XCV800). Esta FPGA soporta la configuración del diseño.

- Una herramienta de software, la interfaz hombre-máquina que proporciona un medio para manipular la información de la FPGA a bajo nivel.
- Un oscilador programable de Dallas Semiconductor DS-1075, que genera el reloj del sistema. Este dispositivo se puede ajustar desde el PC.
- Existe otra FPGA que actúa como interfaz entre el PC y la FPGA avanzada. Todas las comunicaciones, adaptación de protocolo y generación de señales son soportadas por el dispositivo, que es de menor tamaño que la FPGA avanzada y está preprogramada. La comunicación entre el PC y UNSHADES se realiza utilizando el protocolo EPP que alcanza velocidades de hasta 2 Mbytes/s, suficiente para nuestro propósito.

UNSHADES software es un sistema completo que permite el manejo de la configuración a muy bajo nivel y que proporciona algunas opciones interesantes: programación de la FPGA, inspección de registros internos, modificación de registros internos, control total del oscilador y selección de diseño, estando totalmente integrado en el flujo de diseño estándar que proporciona Xilinx. La transferencia de información se ha optimizado mediante el uso de técnicas de reconfiguración parcial.

Como se ha comentado anteriormente, la comunicación entre el PC y UNSHADES-1 no la administra ningún microcontrolador como cabría esperar, sino que se hace a través de una FPGA **XC40000**. La configuración de esta FPGA está guardada en una PROM y se carga en la FPGA durante el arranque de la misma. Es la encargada de gestionar el transvase de la información entre la Virtex 300 y el PC a través del puerto paralelo usando el protocolo EPP. La información se transmite en los dos sentidos, en el sentido PC→UNSHADES-1 se transmite el diseño que queremos probar sobre la FPGA Virtex, es decir, los bits para configurar los bloques de lógica, las conexiones entre ellos, y los bloques dedicados de la Virtex; y en el sentido UNSHADES-1→PC se transmite la información que se guarda en la memoria de configuración de la Virtex cuando configuremos el puerto de descarga para que permanezca activo tras finalizar la configuración de la Virtex y la condición de disparo que hallamos impuesto se verifique.

El proyecto UNSHADES-1 puede trabajar con todas las FPGAs de la familia Virtex, siempre que su encapsulado sea el PQ240. En el momento de la realización del proyecto se disponían de placas con la de cincuenta mil y trescientas mil puertas, dadas las dimensiones del sistema que se iba a desarrollar la más adecuada es la segunda opción, la **V300PQ240**. Esta FPGA no sólo dispone de más circuitería para la lógica del sistema sino que también dispone de más memoria interna: 16 bloques de RAM de 4096 bits cada uno.

Device	SystemGates	CLB Array	Logic Cells	Max. Available I/O	Block RAM bits	Max. SelectRAM bits
XCV50	57,906	16x24	1,728	180	32,768	24,576
XCV100	108,904	20x30	2,700	180	40,960	38,400
XCV150	164,674	24x36	3,888	260	49,152	55,296
XCV200	236,666	28x42	5,292	284	57,344	75,264
XCV300	322,970	32x48	6,912	316	65,536	98,304
XCV400	468,252	40x60	10,800	404	81,920	153,600
XCV600	661,111	48x72	15,552	512	98,304	221,184
XCV800	888,439	56x84	21,168	512	114,688	301,056
XCV1000	1,124,022	64x96	27,648	512	131,072	393,216

Tabla 5.2: *FPGAs de la familia Virtex*

Estas FPGAs de última generación disponen de una serie de nuevas características, muy especialmente en la lógica de configuración de la FPGA (lógica que permite la carga de un diseño en la FPGA, es decir, recibe el flujo de bits que se le envía y lo guarda en la memoria de configuración de la Virtex, esta era la única funcionalidad que hasta ahora presentaba), dado que permite la lectura de la memoria de configuración de la FPGA. La lectura de la memoria de configuración de la Virtex se puede emplear tanto para comprobar que la configuración de la FPGA es correcta como para leer el estado las CLBs, registros de las IOBs, los valores de las LUT RAMs y de los bloques de RAM. Para esto segundo se emplea el elemento *capture*, como se verá más adelante en este capítulo. El sistema para prototipado rápido UNSHADES-1 hace uso de todas estas nuevas características, por lo que permite aprovecharlas para el desarrollo rápido de prototipos. Este entorno fue el que se nos propuso para el desarrollo del proyecto.

5.3.3. DESCRIPCIÓN GENERAL DEL FUNCIONAMIENTO DEL CIRCUITO LÓGICO

El primer paso es la búsqueda de un método adecuado para determinar las coordenadas de un punto 3D. Para lo cual hay que tener muy en cuenta que el objetivo final es el desarrollo de un sistema hardware y más concretamente, que el prototipo se va a implementar sobre una FPGA de la familia Virtex. La principal preocupación es que el método para calcular las coordenadas sea realizable en el entorno propuesto y en tiempo real. Por todo lo anterior se ha buscado la sencillez en el diseño del sistema.

El prototipo ha sido diseñado e implementado sobre la FPGA citada anteriormente. Nuestro circuito externo sólo necesita los sistemas de adquisición de vídeo y los digitalizadores. Dentro de la FPGA se programan los siguientes bloques:

- Un circuito para configurar los decodificadores de vídeo para tener el formato de imagen adecuado.
- Un bloque para capturar los datos procedentes de los decodificadores de forma sincronizada.
- Un circuito para determinar el valor del umbral.
- Un sistema que se encarga de calcular las coordenadas finales.
- Un bloque para representar el punto calculado en un monitor VGA.
- Un interfaz entre el PC y la FPGA, para proporcionarle al PC los resultados del procesamiento de las imágenes realizado en la FPGA.

Las imágenes analógicas captadas por las cámaras son digitalizadas y enviadas a la FPGA para su procesamiento. Una vez obtenidos los resultados, es decir, las coordenadas del punto de soldadura, se pueden representar en un monitor VGA, eligiendo la perspectiva de una de las cámaras, además se transfieren a un ordenador donde tras compararlos con la trayectoria deseada, se generarán los comandos de gobierno del robot, de forma autónoma y en tiempo real.

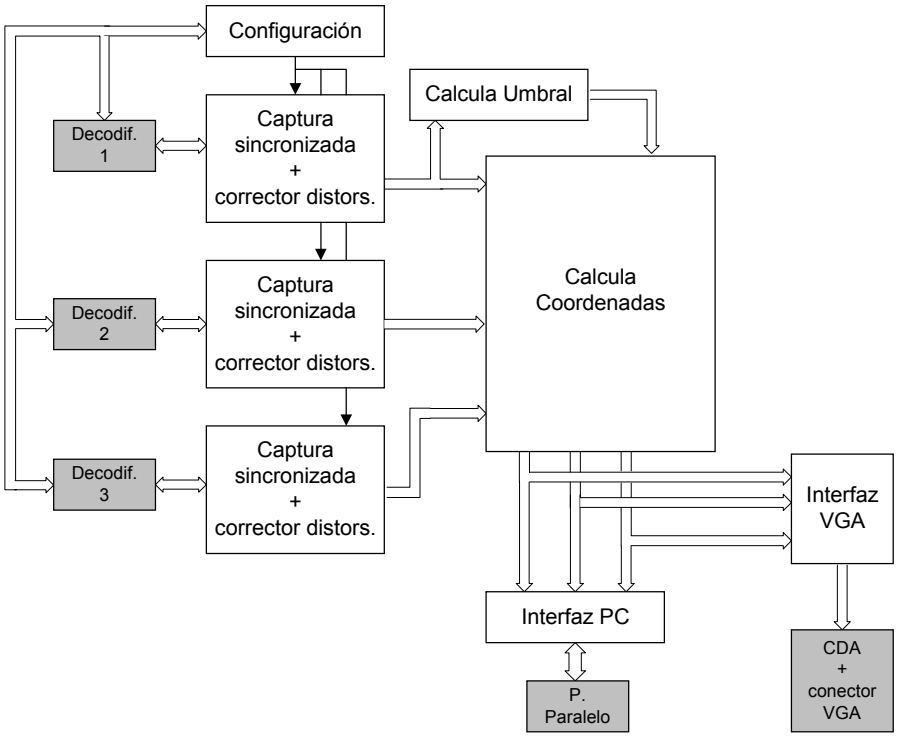


Figura 5.18: *Diagrama de bloques del diseño hardware*

A continuación se describe la funcionalidad de los bloques del diseño *top* del proyecto:

En primer lugar es necesario configurar los decodificadores de vídeo para que funcionen en modo esclavo y proporcionen los datos de salida en formato RAW, es decir, valores de la luminancia de cada uno de los píxeles, codificados en muestras de 8 bits, cuyo rango va de 0 a 255.

Como la precisión es un requisito indispensable del sistema, sobre los datos procedentes de los decodificadores y capturados a la frecuencia adecuada, se realiza una primera operación que consiste en aplicar una función de corrección para corregir la distorsión provocada por las lentes de las cámaras. A continuación se lleva a cabo una selección, de la información útil e indispensable de cada imagen. Para ello, cada uno de esos datos (valores de luminancia de los píxeles) se compara con un valor umbral previamente calculado. Si resulta ser menor que el umbral entonces pasará a valer 0, que se corresponde con el valor mínimo de luminancia (negro). Si el valor del píxel es igual o mayor que el umbral, pasará a valer 255 lo cual corresponde al máximo valor de luminancia (blanco).

Serán las coordenadas de los píxeles cuyo valor supera el umbral, los únicos datos que se almacenen en memoria. Hay que tener en cuenta que cada una de las cámaras tendrá una posición fija y conocida, y hará referencia a un par de coordenadas: (x,y), (y,z) o (z,y). Los decodificadores de vídeo proporcionan imágenes digitalizadas compuestas de 625 líneas de 1440 bytes cada una, es decir, transforman la imagen en una matriz de 625 filas y 1440 columnas, o lo que es lo mismo, cada imagen estará formada por 625x1440 píxeles. Cada uno de esos píxeles se caracterizará por un par de coordenadas. De este modo, considerando un contador de filas y otro de columnas para cada uno de los decodificadores de vídeo, se van obteniendo las coordenadas del *i-ésimo* píxel: (x_{i1},y_{i1}), (y_{i2},z_{i2}) o (z_{i3},x_{i3}), según proceda la imagen de la cámara 1, de la 2 ó de la 3.

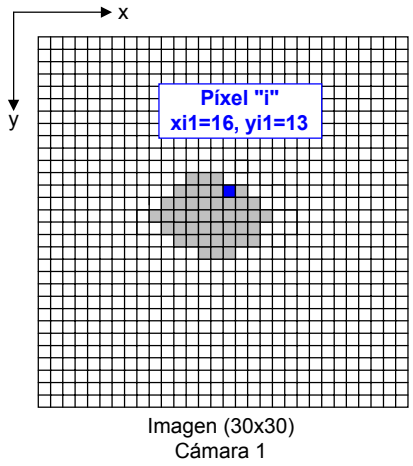


Figura 5.19: *Ejemplo gráfico de las coordenadas del píxel i-ésimo en la imagen proporcionada por la cámara 1*

En el ejemplo mostrado en la *figura 5.19* se observa cómo el punto luminoso está formado por un conjunto de píxeles (cuadraditos grises) que forman un área cuyo centro de gravedad $(\bar{x}_1, \bar{y}_1)$ se calcula como:

$$\bar{x}_1 = \frac{1}{n}\sum_i x_{i1} \qquad \bar{y}_1 = \frac{1}{n}\sum_i y_{i1}$$

Del mismo modo y de forma **paralela**, al finalizar cada pantalla, se calculan los centros de gravedad de las áreas que aparecen en las imágenes dadas por la cámara 2 y por la 3, y que corresponden al mismo punto luminoso. Es decir:

$$\bar{y}_2 = \frac{1}{m}\sum_i y_{i2} \qquad \bar{z}_2 = \frac{1}{m}\sum_i z_{i2}$$

$$\bar{z}_3 = \frac{1}{l}\sum_i z_{i3} \qquad \bar{x}_3 = \frac{1}{l}\sum_i x_{i3}$$

Siendo *l*, *m* y *n* el área de dichas figuras o lo que es lo mismo, el número de píxeles cuyo valor supera al valor umbral.

Se tendrá, por tanto, información redundante[21]:

- Coordenadas x: $\bar{x}_1$ y $\bar{x}_3$
- Coordenadas y: $\bar{y}_1$ e $\bar{y}_2$
- Coordenadas z: $\bar{z}_2$ y $\bar{z}_3$

Las coordenadas finales del punto real de soldadura 3D vendrán dadas por el promedio de los valores anteriores:

$$X = \frac{\bar{x}_1 + \bar{x}_3}{2} \qquad Y = \frac{\bar{y}_1 + \bar{y}_2}{2} \qquad Z = \frac{\bar{z}_2 + \bar{z}_3}{2}$$

Por último, dichas coordenadas serán transferidas a un PC a través de la placa de circuito impreso diseñada. La comunicación con el PC podrá llevarse a cabo mediante el puerto serie o el puerto paralelo, siendo éste el más apropiado debido a su mayor velocidad. Por otra parte, también cabe la posibilidad de mostrar, desde el enfoque de una de las tres cámaras (elegido previamente), el punto luminoso en un monitor VGA. Para ello, el módulo *Interfaz VGA* se encarga de enviar píxeles hacia el PCB, donde existe una circuitería que realizará la conversión digital-analógica de esos datos. Dicho módulo sólo transfiere píxeles de valor 255 (blanco) cuando las coordenadas del punto

[21] Dicha redundancia de información servirá para aumentar la precisión del sistema, resolviéndose además posibles problemas de oclusiones.

de la pantalla que se está enviando en ese momento coincida con las coordenadas (X,Y), (Y,Z) o (Z,X), según la cámara elegida. En los demás casos se transfieren píxeles correspondientes al color negro. El envío de los píxeles se realiza según las normas de generación de imágenes VGA.

5.3.3.1. CONFIGURACIÓN

El primer paso que hay que realizar es el de configurar los decodificadores de vídeo para que proporcionen la información en el formato que mejor se adapte a las necesidades del proyecto. Así pues, los datos de configuración están definidos por:

- Las características de la señal analógica de vídeo que sirve de entrada:
  - Señal analógica de vídeo compuesto (CVBS).
  - Compatible estándar de color PAL.
  - Sincronización horizontal: 15625 Hz.
  - Sincronización vertical: 50 Hz.

- Las características del procesador de vídeo que se quieran utilizar:
  - Amplificador y filtro anti-aliasing activos.
  - Funcionamiento Modo 3: la señal analógica de vídeo es de formato CVBS y entra por el pin AI22 del decodificador de vídeo.
  - Control automático de ganancia.
  - Detección automática de campo, 50 Hz, 625 líneas.
  - Salida de la señal de sincronismo de línea HREF a través del pin RTS0.
  - Salida de la señal de sincronismo de imagen VREF a través del pin RTS1.
  - Los datos de salida del bus VPO serán los 8 bits más significativos de los convertidores analógico-digital (recordemos que son de 9 bits).
  - Formato de salida RAW, es el más adecuado para señales de vídeo en blanco y negro.

La configuración la realiza la FPGA programando, con los valores adecuados, los 99 registros internos que posee cada decodificador. Este proceso se realiza a través

del bus I^2C, que es el encargado de la comunicación entre la FPGA y cada procesador de vídeo[22].

El hecho de que el control de todo el sistema lo llevará a cabo el circuito implementado en la FPGA permite realizar una serie de simplificaciones sobre la parte de control I^2C que habrá que implementar sobre la FPGA:

- En primer lugar, no será necesario ejercer ningún tipo de arbitraje ya que nuestro sistema tiene un **único** dispositivo **maestro**: la FPGA Virtex 300 de la placa del proyecto UNSHADES.
- En segundo lugar, el sistema tiene **tres** dispositivos **esclavos**: los procesadores de vídeo.
- Por último, la FPGA sólo tendrá que realizar operaciones de **escritura** mediante el I^2C sobre los registros internos de los decodificadores de vídeo, y estas operaciones las llevará a cabo una sola vez.

Por estas tres razones podremos implementar un circuito de control I^2C más sencillo, sin necesidad de tener que implementar el circuito que se ajuste al protocolo completo.

El circuito diseñado para llevar a cabo la configuración estará ampliamente basado en uno de los bloques realizado en un Proyecto Fin de Carrera anterior al nuestro, cuyo esquema es el siguiente:

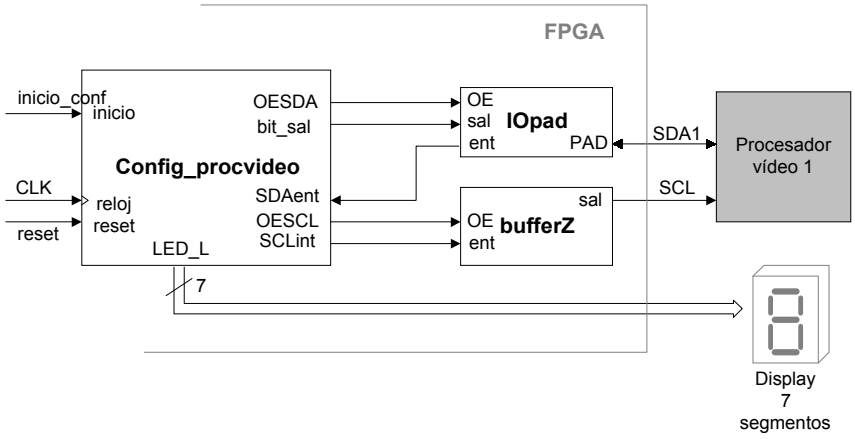


Figura 5.20: *Interconexión entre el decodificador y su circuito de configuración*

En nuestro caso, no se dispone de ningún display en el que se visualice el estado del proceso de configuración. En cambio, se tendrá una señal de salida, llamada

[22] Para una información más detallada sobre el bus I^2C, ver apartado 4.3.1.2.1 del Capítulo IV de la presente memoria.

cofiguración_ok, que pasará a nivel alto cuando la configuración se haya terminado con éxito y permanecerá en ese estado hasta que se reciba un *reset*.

El hecho de tener que configurar **tres** decodificadores en lugar de uno sólo, puede conducir, en un principio, a considerar tres bloques como el que aparece en la *figura 5.20*, uno para cada procesador de vídeo. Pero la forma óptima de abordar el problema es considerar un único circuito para la configuración de los tres decodificadores. Esta solución es la más ventajosa, no sólo respecto a los recursos utilizados por el sistema (menor área), sino también desde el punto de vista de la sincronización de los tres decodificadores, uno de los requisitos que debe cumplir nuestro sistema.

Cada uno de los procesadores de vídeo tiene su oscilador de cristal independiente. Todos oscilan a la misma frecuencia pero no están sincronizados. Una vez configurados, cada uno de ellos comienza a proporcionar los datos de forma continua. Por tanto, será muy importante que los tres empiecen a proporcionar los datos en el mismo instante, es decir, sincronizarlos. Para ello habrá que **configurarlos simultáneamente**.

El esquema del circuito de configuración empleado en el diseño es el siguiente:

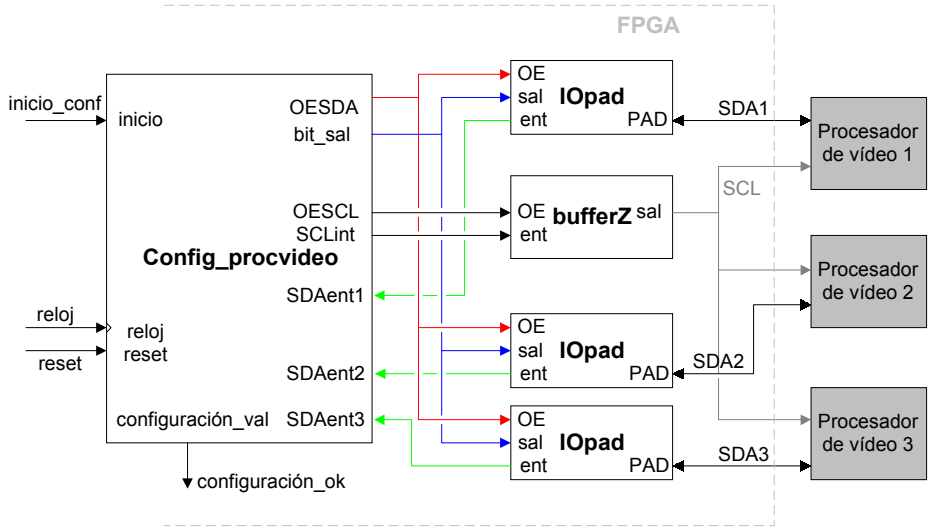


Figura 5.21: *Top de Configuración de los decodificadores de vídeo*

El bloque principal es el denominado ***Config_procvideo***, que se encarga de la configuración simultánea de los tres decodificadores. Entre este circuito y los procesadores de vídeo se encuentran los elementos triestado: ***IOpad*** y ***bufferZ***, que controlarán la bidireccionalidad y el estado de alta impedancia de las líneas *SDA1*, *SDA2*, *SDA3* (líneas de datos) y *SCL* (reloj). Dichas líneas formarán el bus I^2C, que en nuestro caso, como se verá a continuación, estará multiplexado en tres buses.

Tal y como se explicó anteriormente, la FPGA (maestro) sólo realizará operaciones de escritura en los registros internos de los decodificadores (esclavos). En cada proceso de escritura se distinguen las siguientes fases:

El circuito ***Config_procvideo*** genera la condición de *START*.

Config_procvideo envía a los tres decodificadores la dirección hexadecimal: X"4A" para indicarles que ellos serán los esclavos en escritura, es decir, que se va a escribir un valor en el registro correspondiente. El byte X"4A" se envía bit a bit (según el protocolo I^2C) por la línea *bit_sal*, y habilitando simultáneamente los bloques ***IOpad*** mediante la señal *OESDA*, se tendrán dichos bits en las líneas *SDA1*, *SDA2* y *SDA3* a la vez.

Cada uno de los decodificadores envía un asentimiento si recibe correctamente el byte enviado por el maestro, el cual esperará a recibir los **tres** asentimientos para continuar con el siguiente paso. Los asentimientos son recibidos por la FPGA a través de las líneas *SDA1*, *SDA2* y *SDA3*, cambiando su direccionalidad. Estos asentimientos llegan al circuito ***Config_procvideo*** por las entradas: *SDAent1*, *SDAent2* y *SDAent3*.

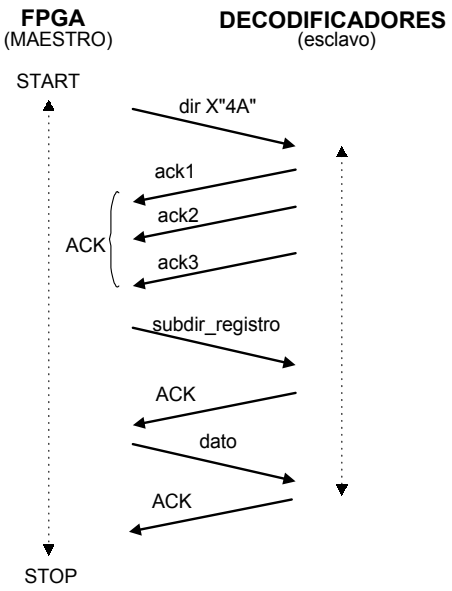


Figura 5.22: *Etapas del proceso de escritura*

El bloque ***Config_procvideo*** envía un byte que contiene la subdirección del registro interno del decodificador de vídeo que se va a programar. Esta transferencia se realiza como en el paso 2.

Los decodificadores vuelven a asentir el byte recibido correctamente.

El maestro espera recibir los tres asentimientos.

Config_procvideo envía un byte con el valor que se desea escribir en el registro indicado anteriormente.

Vuelve a esperar los asentimientos y finalmente, genera la condición de *STOP*.

El circuito ***Config_procvideo*** está formado principalmente por los siguientes bloques:

- ***FSM_I2C***: máquina de estados que se encarga de realizar una operación de escritura (según el protocolo I^2C) sobre un registro interno de cada procesador de vídeo:

- Cuando recibe la señal *ini_trans* a nivel alto, en primer lugar, genera la señal *carga_datos* de permiso de carga de los registros que mantendrán la subdirección y el dato.

- A continuación genera la condición de *START*. Para ello es necesario que controle las líneas SDA y SCL según indica el protocolo I^2C. Esto lo consigue gracias a que:
  1. Mediante el bloque ***divisor***, que es un contador de 8 bits, se genera el reloj SCL a partir del reloj del sistema (50 MHz). La frecuencia obtenida es:

$$f_{SCL} = \frac{50MHz}{256} = 195.312KHz < 400\frac{Kbits}{s} \quad (\text{fast-mode } I^2C)$$

Para introducir esperas y respetar la temporización del protocolo I^2C se emplean los contadores:

- ***Cont_espera64*** que cuenta 64 periodos de reloj del sistema.
- ***Cont_espera128*** que cuenta 128 periodos de reloj del sistema.

- El siguiente paso es enviar la *dirección esclavo* X"4A" empezando por el bit más significativo. Para evaluar si se ha enviado el byte completo, se comprueba si el valor de *punt_bit* (bus de 3 bits) es 7. Una vez enviado el último bit, se comprueba si el asentimiento se ha recibido correctamente por la línea *SDAent*. Si es así, entonces se evalúa si se ha enviado el último byte, es decir, el correspondiente al dato. Si no es esa la situación, se enviaría el siguiente byte: subdirección del registro. Y se repetiría hasta enviar el dato. Si por el contrario, el asentimiento no fuese recibido, entonces se generaría la condición de *STOP*. También se genera esta condición cuando el valor de *punt_byte* (bus de 2 bits) es 3, indicando que se ha enviado el último byte, ya que en cada proceso de escritura se transfieren 3 bytes: dirección X"4A", subdirección del registro y el dato o valor a escribir en el registro. Para llevar a cabo dicha transferencia de bits en serie se emplean los siguientes bloques:
  - ***mux_byte***: módulo que da como salida uno de los tres bytes que tiene a su entrada dependiendo del valor de *punt_byte*.

- ***mux***: bloque en cuya salida se obtienen en serie los bits del byte de entrada. Para ello emplea el valor *punt_bit*.
- ***tx_bit***: biestable para mantener estable el bit a transmitir.

Hasta ahora hemos visto cómo ***FSM_I2C*** se encarga de programar *un registro* interno de los decodificadores de vídeo. Pero éstos tienen 99 registros internos, la mayoría de ellos de escritura, por tanto se necesita un circuito que controle las *sucesivas* operaciones de escritura necesarias para configurar los procesadores de vídeo. De esto se encarga la máquina de estados ***FSM_gen_datos***.

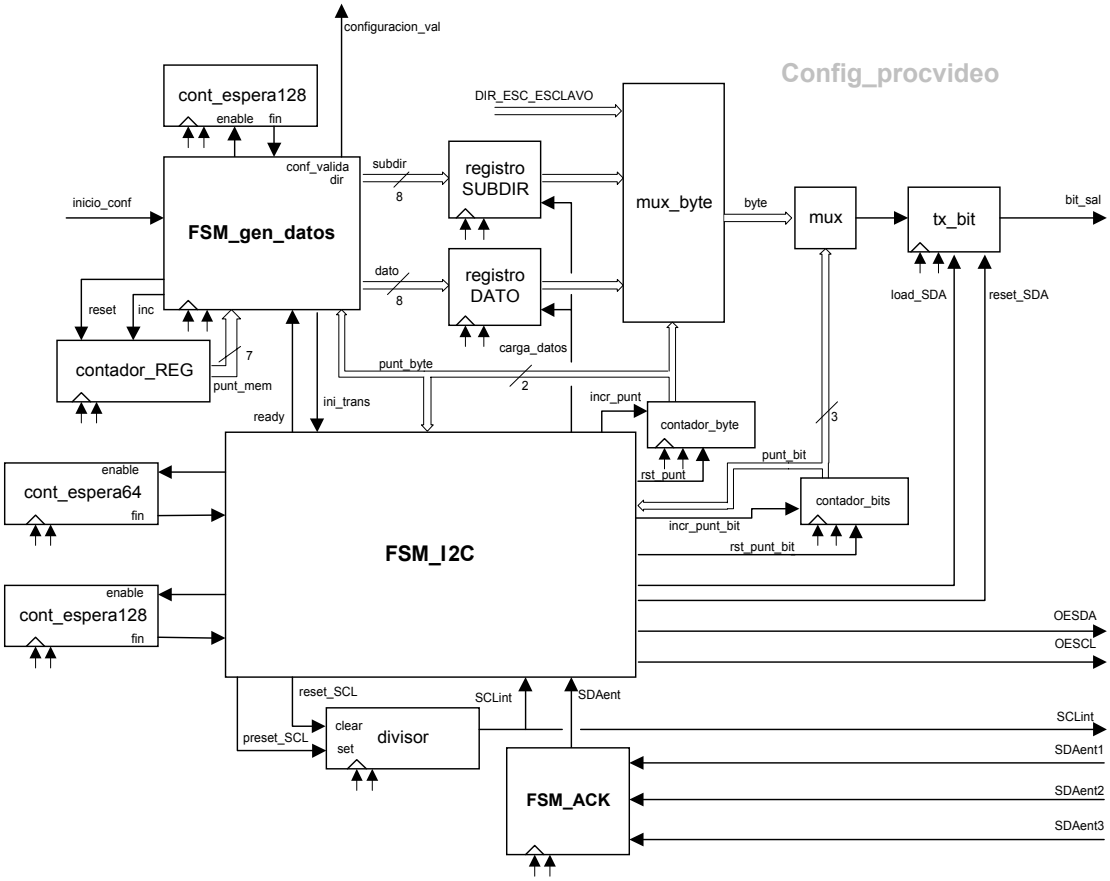


Figura 5.23: *Diagrama de bloques de* ***Config_procvideo***

- ***FSM_gen_datos***: máquina de estados que controla el proceso de configuración de los decodificadores de vídeo, es decir, la escritura de los registros internos de los decodificadores.

El usuario es quien inicia el proceso de configuración mediante la señal *inicio_conf*. En ese momento, se realiza la primera operación de escritura, sobre el primer registro configurable, el registro de subdirección X"01".Dicho registro será programado con el valor X"08". Si la escritura se realiza con éxito, se continúa con la escritura del dato X"C3" en el siguiente registro, el de subdirección X"02". Y así sucesivamente, mientras las operaciones de escritura se realicen correctamente y no se haya programado el último registro, cuya subdirección es X"5F", lo cual significaría que se ha concluido con éxito la configuración y se indicaría con la señal *configuracion_val*.

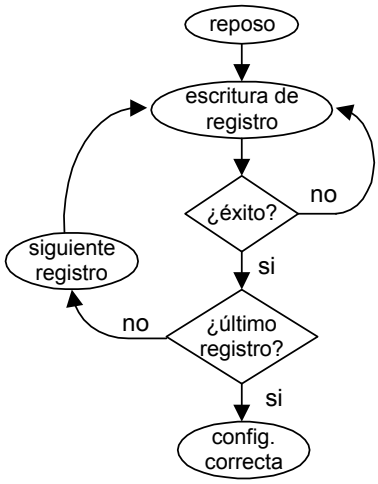


Figura 5.24: *Diagrama de flujo de* ***FSM_gen_datos***

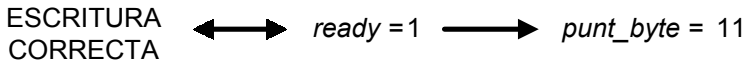
Cuando alguna operación de escritura no se realiza con éxito se repite hasta que se consiga ejecutar de forma correcta.

FSM_gen_datos reconoce que una operación de escritura se ha llevado a cabo con éxito gracias a las señales:

- *ready*: procedente de ***FSM_I2C***; indica a ***FSM_gen_datos*** que puede activar el inicio de otra escritura.
- *punt_byte*: bus (2 bits) de salida de ***contador_byte***. Indica si se han enviado con éxito los tres bytes que componen una operación de escritura, es decir: dirección esclavo X"4A" , subdirección del registro y el dato. Cada vez que uno de esos bytes se transfiere correctamente, ***FSM_I2C*** incrementa *punt_byte* en una unidad. En caso contrario, ***FSM_I2C*** genera la condición de *STOP* y no

incrementa el puntero. De este modo, *punt_byte* se queda apuntando al byte cuyo asentimiento no se ha recibido de forma correcta.

Por tanto:



Cuando la operación de escritura es correcta, ***FSM_gen_datos*** activa una nueva operación enviándole a ***FSM_I2C*** la señal *ini_trans*.

Los valores con los que se programan los registros internos así como las subdirecciones de éstos, están almacenados en una memoria definida en ***FSM_gen_datos***. Dicha memoria se compone de 94 registros de 2 bytes cada uno. En el byte superior estará la subdirección del registro que se va a programar. En el inferior estará el dato que desea escribir en dicha subdirección.

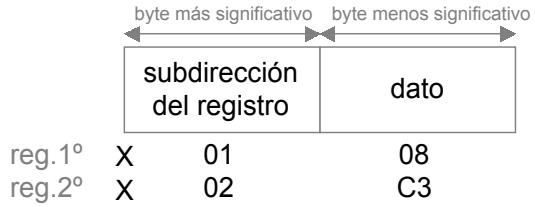


Figura 5.25: *Ejemplo dos primeros registros de la memoria*

Cada uno de esos registros se direcciona mediante el puntero *punt_mem*. Es un bus de 7 bits proporcionado por el bloque ***contador_REG***, el cual está controlado por ***FSM_gen_datos*** mediante las señales *reset_punt_mem* (resetear y apuntar al primer registro) e *inc_punt_mem* (incrementar y apuntar al siguiente registro).

- ***FSM_ACK***: máquina de estados que activa (a nivel bajo) su señal de salida *SDAent* sólo cuando recibe los tres asentimientos de los decodificadores de vídeo: *SDAent1*, *SDAent2* y *SDAent3*.

5.3.3.2. CAPTURA: SINCRONIZACIÓN

Una vez configurados los decodificadores de vídeo, éstos se sincronizan con las imágenes analógicas que reciben y comienzan a proporcionar los datos digitales (formato RAW) a una frecuencia de 27 MHz. Las principales restricciones a tener en cuenta en el diseño de esta parte del sistema son:

- En primer lugar, la frecuencia del reloj del sistema es 50 MHz mientras que los datos aparecen cada 27 MHz. Esta falta de sincronismo entre la frecuencia de captura del sistema y los datos proporcionados puede ocasionar problemas que tendrán que ser considerados.
- En segundo lugar, no se puede empezar a capturar los datos en cualquier instante, ya que existiría la posibilidad de capturar datos no válidos (zona de *blanking)*.
- Por último, habrá que tener en cuenta que cada imagen completa está formada por dos campos[23] y por tanto habrá que respetar el orden de éstos.

Como se explicó en el Capítulo IV, cada decodificador proporciona los datos a través de su bus VPO (8 bits). La señal LLC, mediante su flanco de subida, indica la validez de esos datos. El problema surge porque LLC es una señal de reloj de 27 MHz mientras que el reloj de nuestro sistema sólo puede programarse para que trabaje a frecuencias iguales a: $\frac{1}{n}$x100 MHz, con n = 1,2,3,4,...En nuestro caso se programa a 50 MHz. Con esta frecuencia de reloj no podríamos capturar los datos directamente del bus VPO con la condición: LLC = ‘1’, ya que habría instantes en que capturaríamos datos no válidos aunque detectásemos que LLC = ‘1’.

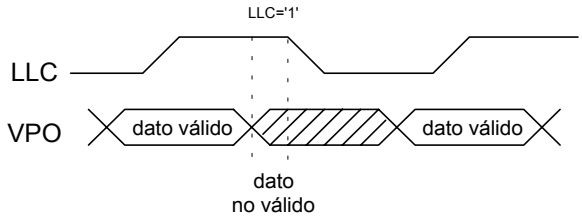


Figura 5.26: *LLC y bus VPO*

[23] Ver apartado “Interfaces y Estándares” del Capítulo II

Esto se debe a que, tal y como se observa en la figura anterior, son los *flancos de subida* y no los niveles 'alto' o 'bajo' de la señal LLC, los que indican la presencia de datos válidos en VPO.

Para resolver este problema de sincronismo, los datos del bus VPO pasan, en primer lugar, por un *latch* activo a nivel bajo de la señal LLC (ver *figura 5.28*). De este modo, en el bus *aux_VPO* se tendrán sólo los datos de VPO cuando LLC = '0' y también justo cuando se produce el flanco de subida de LLC. Es decir, se tendrán datos válidos y no válidos, tal y como se indica en la siguiente figura.

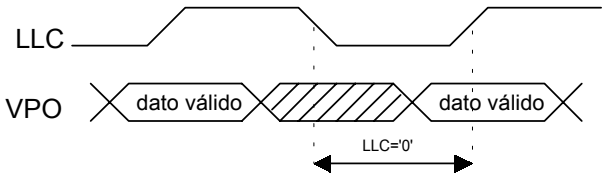


Figura 5.27: *LLC='0' y bus VPO*

Sin embargo, mientras LLC = '1', *aux_VPO* mantiene el valor que hay en VPO justo en el flanco de subida de LLC. Es decir, en el bus *aux_VPO* se tendría un dato **válido**. Esto se consigue con la sentencia VHDL:

auxVPO<=VPO when (LLC='0') else VPOreg;

donde *VPOreg* es la salida de un registro cuya entrada es *aux_VPO*.

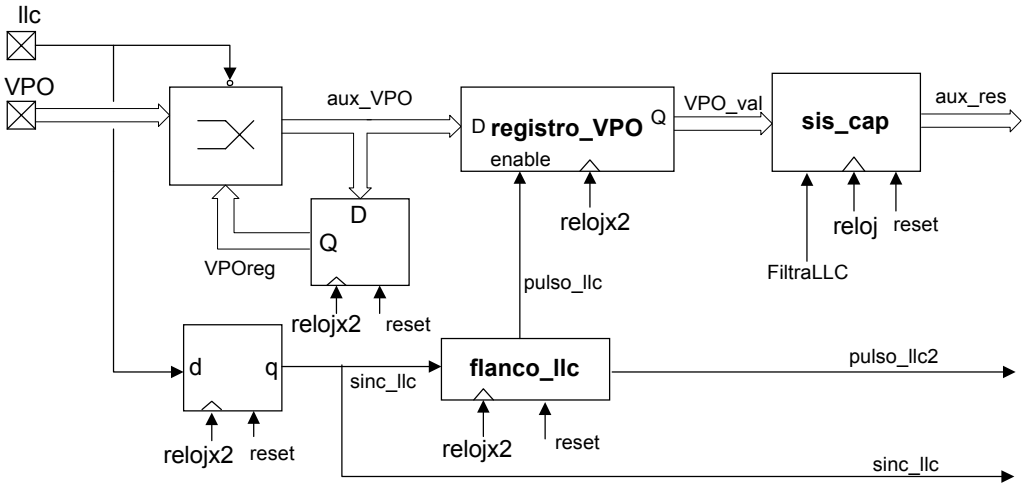


Figura 5.28: *Bloque para sincronizar la captura de los datos. Existe un bloque como éste para cada uno de los decodificadores de vídeo*

Para asegurar la captura del dato válido se considera el bloque ***flanco_llc***, que se encarga de detectar los flancos de subida de la señal *sinc_llc*, que no es más que la línea de reloj LLC sincronizada con un reloj de **100 MHz** (*relojx2)*. Es muy importante señalar que en esta etapa de captura y sincronización de las señales de entrada, además del reloj maestro de 50 MHz, se emplea una señal de reloj del doble de frecuencia. Y el motivo principal es el de evitar que existan flancos de subida de LLC sin detectar.

Una vez detectado el flanco de subida de *sinc_llc*, el byte presente en *aux_VPO*, que será un dato válido, se carga en el registro ***registro_VPO*** y a partir de ese instante se puede considerar que el dato con el que se trabajará en el resto del sistema es correcto.

Por otra parte, para sincronizarnos con la imagen digitalizada habrá que tener en cuenta las señales *href* y *vref*, que son proporcionadas por cada uno de los decodificadores:

- *href*: indica con cada uno de sus flancos de subida, el comienzo de la zona de vídeo activo de cada una de las 625 **líneas** que componen la imagen. Permitirá determinar el comienzo de cada línea y su final.
- *vref*: indica con sus flancos de subida el comienzo de un nuevo **campo**. Permitirá distinguir entre los dos campos que forman cada imagen, así como, identificar el comienzo y el fin de la misma.

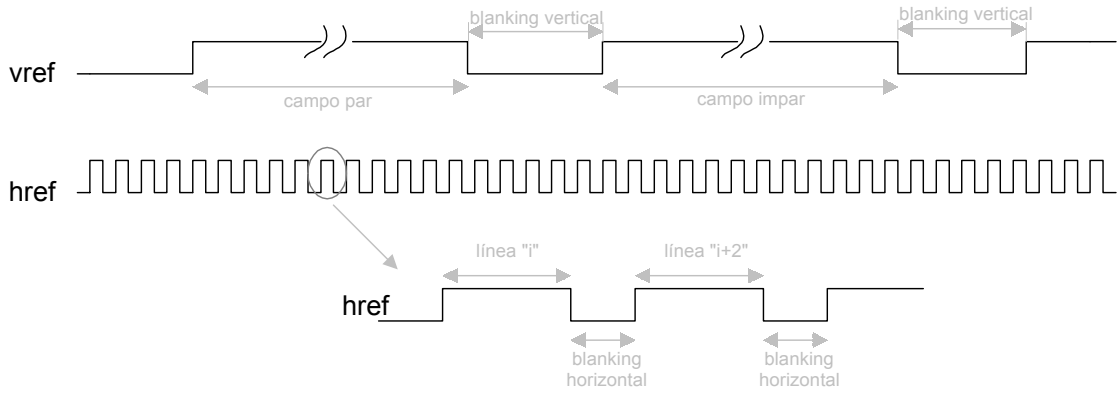


Figura 5.29: *Señales de sincronismo vertical y horizontal (vref , href)*

Estas señales, una vez sincronizadas con el reloj de 100 MHz, son utilizadas por el circuito ***coordenadas_xy*** para determinar, entre otras cosas, el instante en que el dato almacenado en el ***registro_VPO*** debe transferirse al resto del sistema, a través del registro ***sis_cap***, a la frecuencia del reloj maestro, que es de 50MHz. Tal y como se puede ver en el siguiente diagrama:

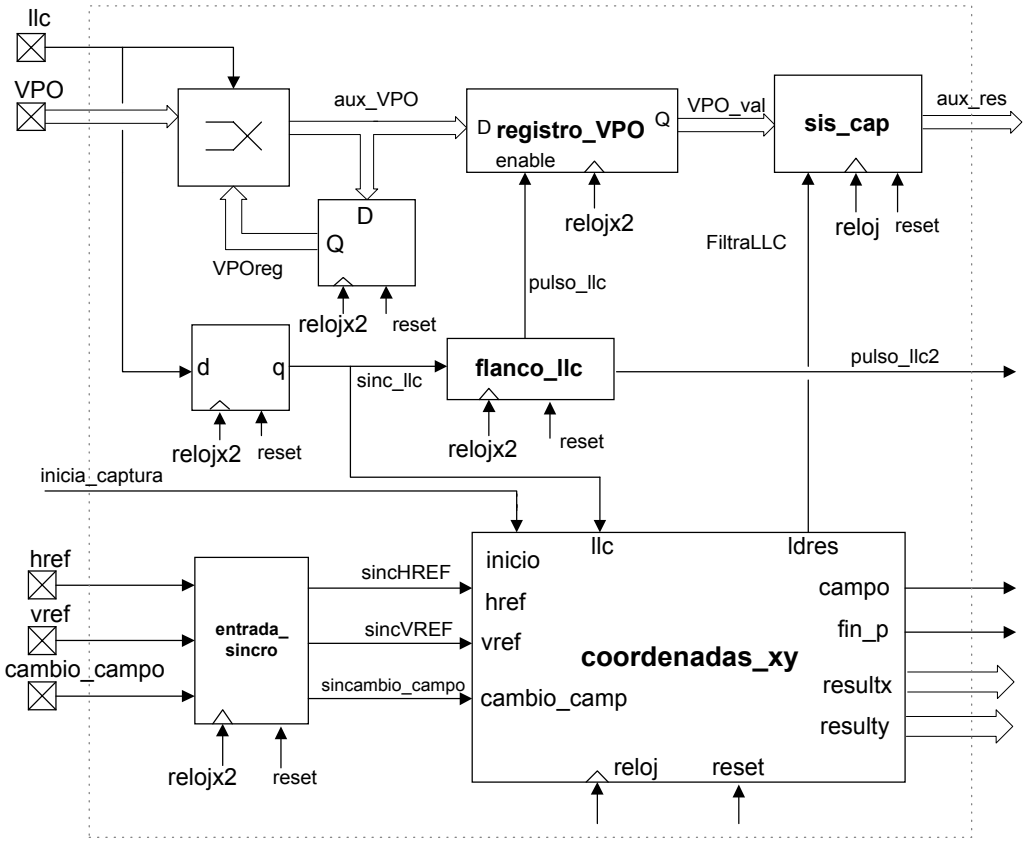


Figura 5.30: *Diagrama de bloques del sistema de captura y sincronización*

Otras de las funciones del bloque ***coordenadas_xy*** son:

- Cambiar el orden de los campos cuando el usuario lo indique mediante la señal *cambio_campo*.
- Determinar, para el valor del píxel capturado (*aux_res*), la posición que ocupa dentro de la línea de vídeo, es decir, su coordenada x: *resultx*.
- Del mismo modo, determinar en qué línea se encuentra dicho píxel, es decir, cuál es su coordenada y: *resulty*.

Para llevar a cabo esas tareas, el circuito ***coordenadas_xy*** consta de los bloques que aparecen en la siguiente figura y que se describen a continuación:

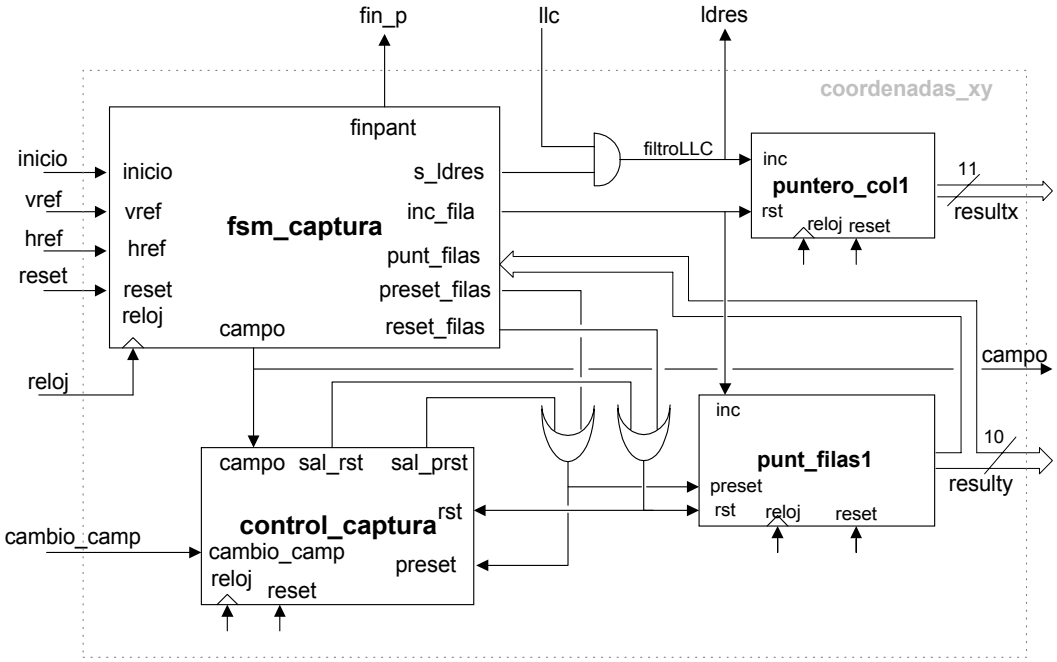


Figura 5.31: *Diagrama de bloques del sistema* ***coordenadas_xy***

- ***fsm_captura***: máquina de estados que controla la sincronización con la imagen digitalizada y marca las zonas de vídeo activo. Para ello, en primer lugar se sincroniza con las entradas *vref* y *href*. Además, controla a los punteros de fila y columna de cada píxel. También se ocupa de determinar el campo de la imagen y de cambiarlo cuando se indique desde el exterior.

  Una vez configurados los decodificadores, se activa la fase de sincronización y captura, mediante la línea *inicio*, que es una señal de entrada de ***fsm_captura***. A partir de ese momento, el proceso que sigue la máquina de estados para poder sincronizarse con la imagen es sencillo:

  En primer lugar se evalúa *vref*:

  - Si es cero, estaríamos en la zona de *blanking* vertical y habría que esperar a que valiese uno, o lo que es lo mismo, a que comience un nuevo campo.

- Si es uno, entonces primero se espera que termine el campo actual, es decir, se espera a que valga cero y luego hay que esperar que vuelva a valer uno (comienzo del nuevo campo).

Una vez sincronizados con el inicio del nuevo campo, se evalúa *href*:

- Si es cero (zona de blanking horizontal) se espera a que sea uno, detectándose así el inicio de la primera línea de vídeo del campo.
- Si es uno, hay que esperar que termine la línea actual (esperar que sea cero) y después esperaríamos a que fuese uno de nuevo (inicio de línea).

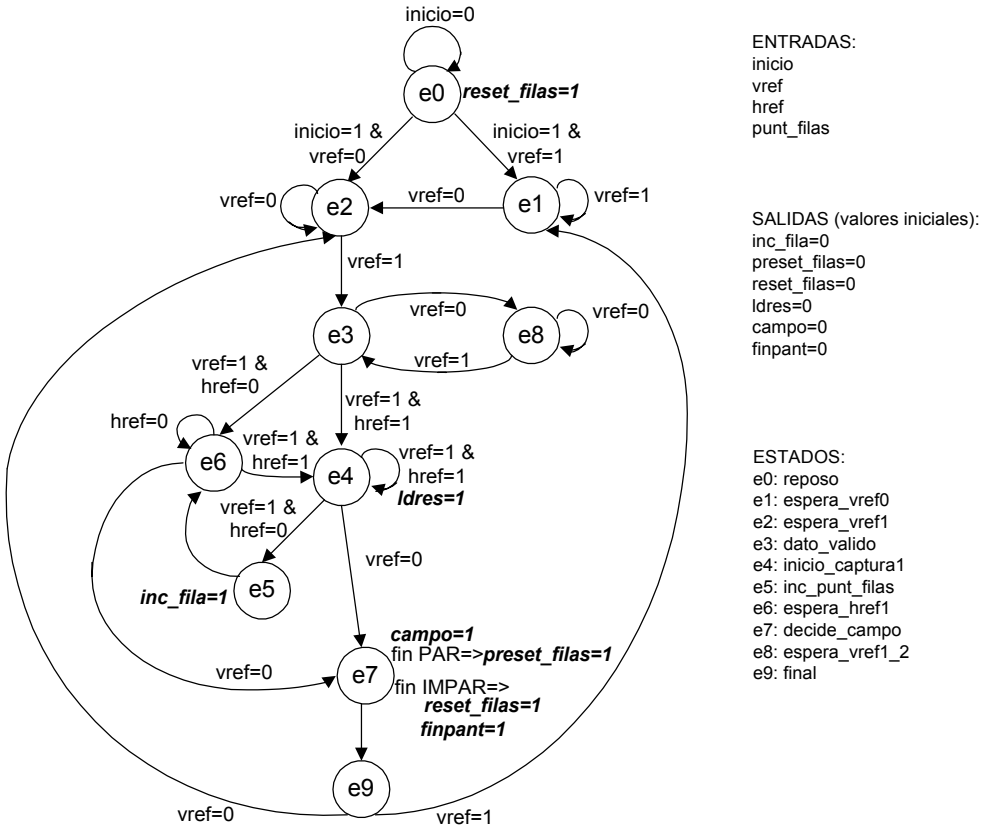


Figura 5.32: *Diagrama de bolas de* ***fsm_captura***

Una vez que *vref* y *href* están a nivel alto, se puede habilitar la captura de los datos. Para ello, ***fsm_captura*** se encarga de activar la señal *ldres*. Y cuando alguna de ellas valga cero, *ldres* también será cero, indicando que se está en algún periodo

de *blanking*, ya sea horizontal (*href* = ’0’) o vertical (*vref* = ’0’), deshabilitando la captura de los datos.

Cada flanco de bajada en la señal *href* indica el fin de una línea de vídeo activo y ***fsm_captura*** genera un pulso en la señal *inc_fila* para que el contador ***punt_filas1*** se incremente. Dicho incremento será de dos unidades debido a la exploración entrelazada que se realiza de la imagen: primero el campo par, formado por las líneas pares:0, 2, 4, 6,... y después el campo impar, compuesto por las líneas 1, 3, 5,...

Para indicar el fin de un campo (par o impar), ***fsm_captura*** genera un pulso en la señal *campo* cada vez que *vref* presenta un flanco de bajada. En ese instante, ***fsm_captura*** también se encarga de evaluar si el campo que concluye es par o impar. Para ello comprueba si el bit menos significativo del puntero de filas *punt_filas* es ‘0’ (campo par) ó ‘1’ (campo impar). En el primer caso, ***fsm_captura*** activa un la señal *preset_filas*, poniendo a ‘1’ el contador de filas ***punt_filas1***. En el segundo caso, se genera un pulso en *reset_filas* (poner a ‘0’ el puntero de filas) y además se indica el fin de una imagen[24] con la señal *finpant*. (Ver *figura 5.33*).

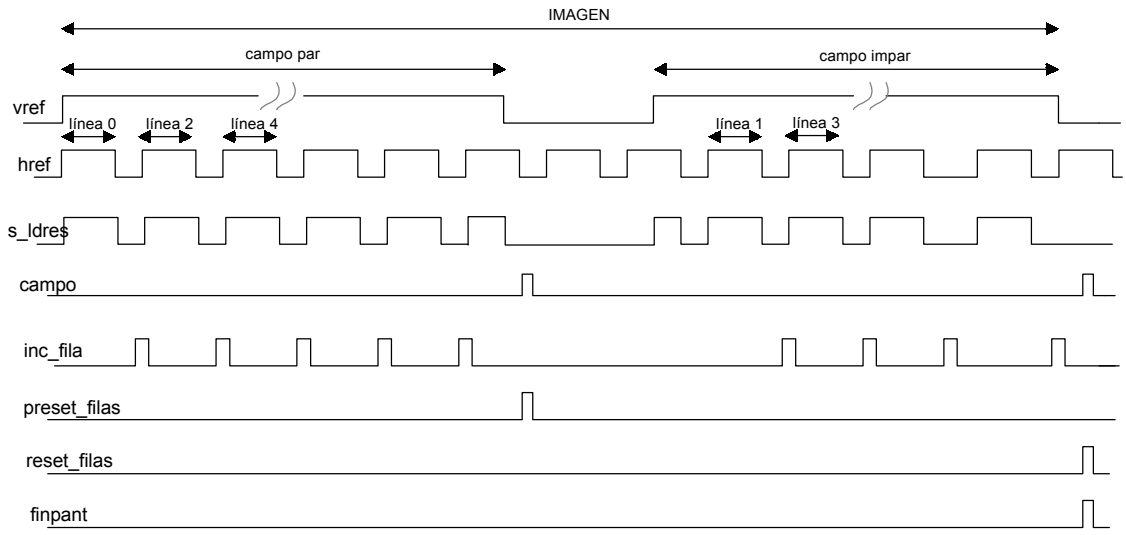


Figura 5.33: *Formas de onda de las señales de salida de* ***fsm_captura***

[24] Cada pantalla completa viene dada primero por el campo PAR y luego el IMPAR.

Hay que señalar que pueden existir otras condiciones para resetear o poner a uno el contador de filas ***punt_filas1***. Estas condiciones vendrán dadas por el propio usuario, que desde el exterior, mediante la señal *cambio_camp*, podrá provocar un cambio en el orden de los campos si este orden no es el correcto (el campo real de la imagen no se corresponde con el campo considerado en ***fsm_captura***).

De esto se encarga el circuito ***control_captura***, cuyo diagrama de bloques es el siguiente:

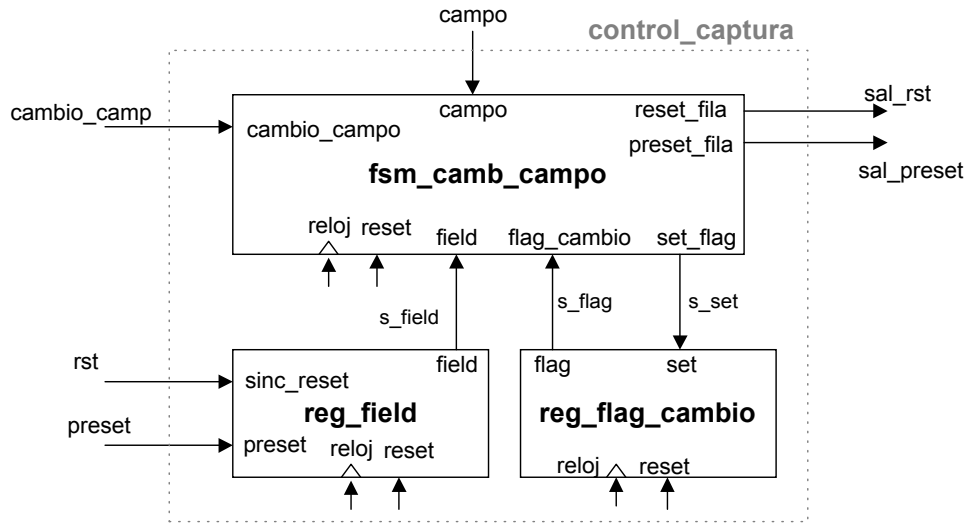


Figura 5.34: *Diagrama de bloques de* ***control_captura***

- ***fsm_camb_campo***: ante la petición de cambio del orden de los campos por parte del usuario (*cambio_camp*), esta máquina de estados analiza si se ha producido un cambio previo. Si no se ha realizado, entonces lo lleva a cabo. En caso contrario, no lo ejecuta. El cambio en el orden de los campos sólo se hará una vez.

  El procedimiento que sigue es muy simple:

  Cada vez que termina un campo, lo cual se indica con la señal de entrada *campo* = '1', ***fsm_camb_campo*** analiza, en primer lugar, si existe petición de cambio de campo. En ese caso, comprueba si ya se ha producido una solicitud previa o no. Para ello tiene en cuenta la señal *flag_cambio*:

  - Si su valor es '1'entonces ya ha existido un cambio de campo y se vuelve al reposo.

- Si *flag_cambio* = ’0’ entonces se trata de la primera solicitud de cambio, se pasa a ejecutar y se indica con *set_flag* = ’1’.

Si no hay solicitud de cambio, se vuelve al reposo.

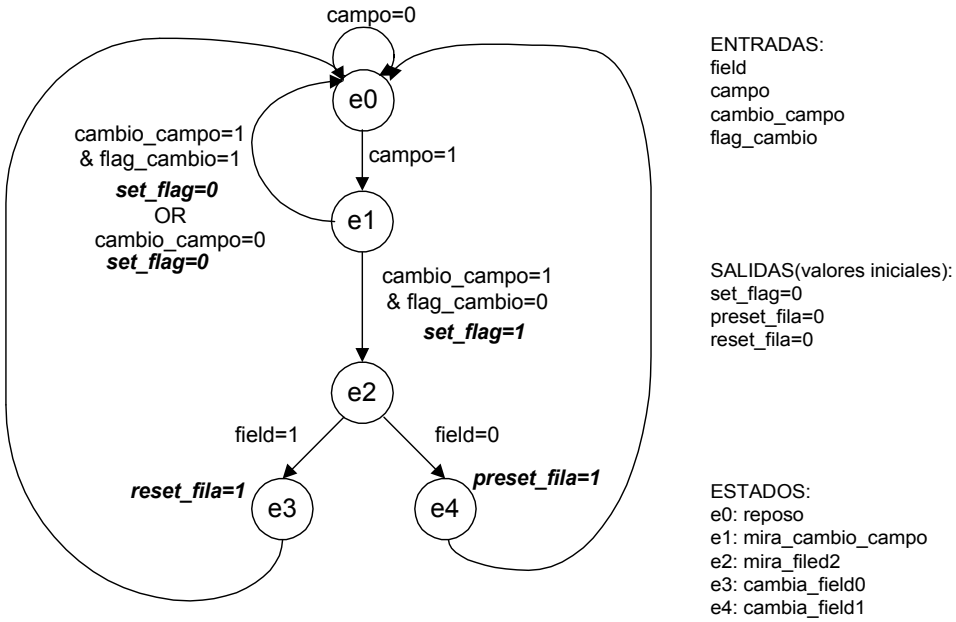


Figura 5.35: *Diagrama de bolas de* ***fsm_camb_campo***

Para realizar el cambio de campo, ***fsm_camb_campo*** evalúa la señal *field*, que indica si el campo actual es el par (*field* = ‘0’) o el impar (*field* = ‘1’). En el primer caso se activa la señal de salida *preset_fila* (poner a ‘1’ el puntero de filas), y en el segundo, se activa *reset_fila* (poner a ‘0’ el puntero de filas).

- ***reg_flag_cambio***: máquina que controla el flag que indica que se ha activado la opción de cambiar el campo, es decir, *flag_cambio*. Una vez que se active el cambio de campo, dicha señal permanecerá a nivel alto hasta que se reciba un reset global.

- ***reg_field***: bloque que a partir de las señales de entrada *rst* y *preset*, determina el campo actual, indicándolo con la señal *field*. Hay que señalar que esas señales de entrada son las resultantes de realizar la operación lógica OR a las salidas de ***fsm_camb_campo*** y ***fsm_captura*** que actúan sobre el puntero de filas.

(Ver *figura* 5.31). Serán esas mismas señales: *rst* y *preset* las que actúen como entradas de ***punt_filas1***.

Por último, y volviendo a la *figura* 5.31, se tiene el bloque ***puntero_col1***, encargado de proporcionar la posición (columna) que ocupa cada píxel dentro de la línea de vídeo. Se trata de un contador cuya señal de incremento *filtroLLC* es la misma que indica el instante en que el dato válido (valor del píxel) almacenado en ***sis_cap*** se transfiere al resto del sistema. Su forma de onda aparece en la siguiente figura:

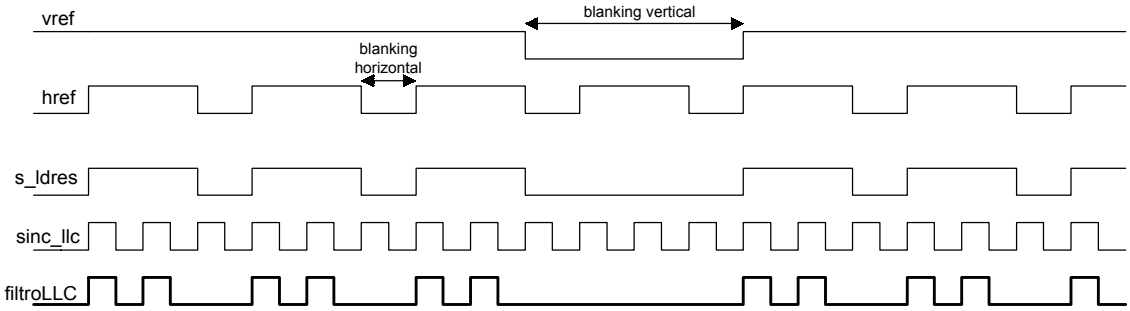


Figura 5.36: *Forma de onda de filtroLLC*

El puntero se incrementa de 1 en 1 y se resetea cada vez que finaliza una línea de vídeo.

5.3.3.3. CÁLCULO DEL UMBRAL

De todos los píxeles que componen una imagen, sólo nos interesarán los que forman parte del punto luminoso cuyas coordenadas se pretende calcular. Para ello, se compara el valor de todos y cada uno de los píxeles con un valor umbral. Este valor umbral se calcula a partir del valor del píxel más luminoso (máximo) y de un valor medio obtenido teniendo en cuenta todos los píxeles de una pantalla.

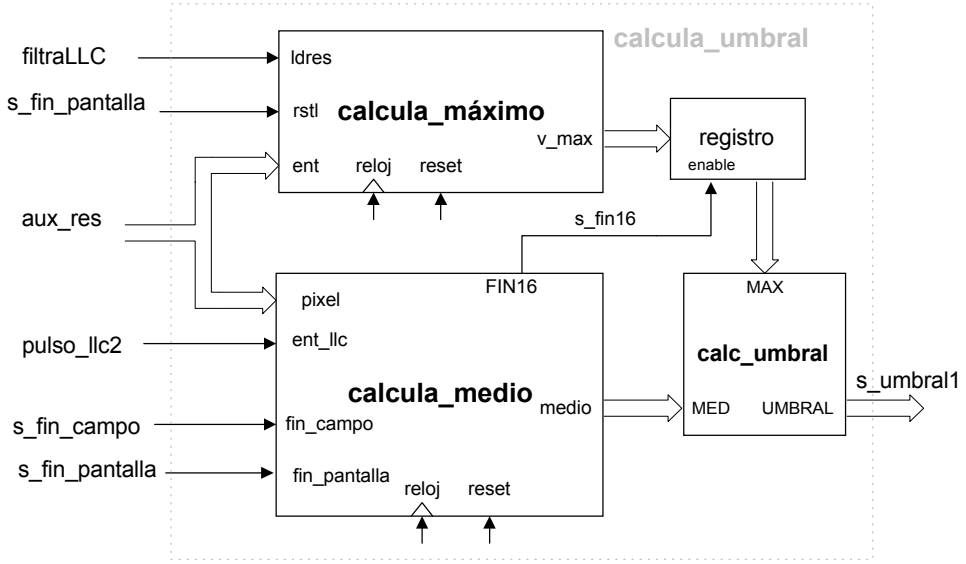


Figura 5.37: *Diagrama de bloques para calcular el umbral*

El valor umbral se calcula cada 0.32ms (16 pantallas) y según la siguiente expresión:

$$umbral = \frac{máximo + medio}{2}$$

Además hay que señalar que el umbral se va a calcular considerando sólo los datos de uno de los decodificadores de vídeo. Se toma esa decisión por simplificar el diseño[25].

5.3.3.3.1. Cálculo del valor Máximo

Bloque que da como salida en un determinado instante el valor máximo de todos los píxeles que se han ido leyendo hasta ese momento. Consta de un bloque llamado ***max1***, donde se compara cada píxel nuevo con el anterior y de un registro ***registro_rstl*** donde se guarda el píxel que resulte mayor después de la comparación.

[25] En el Capítulo VI se darán más detalles.

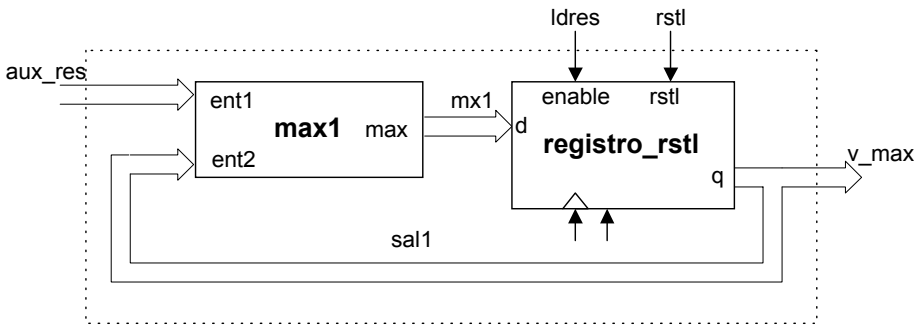


Figura 5.38: *Bloques del circuito que calcula el valor máximo*

El valor máximo estará comprendido entre 0 y 256, ya que cada píxel se codifica con 8 bits.

5.3.3.3.2. Cálculo del valor Medio

Módulo que calcula el valor medio teniendo en cuenta el control automático de ganancia de la cámara de vídeo. En cada "campo" se cuentan los píxeles que son mayores que un determinado umbral. Si el número de estos píxeles supera el 8% del total de píxeles que hay en un campo, entonces se incrementa el umbral en un 25%. Y se vuelve a repetir el proceso. El umbral de partida es de 64, y el final el de 224.

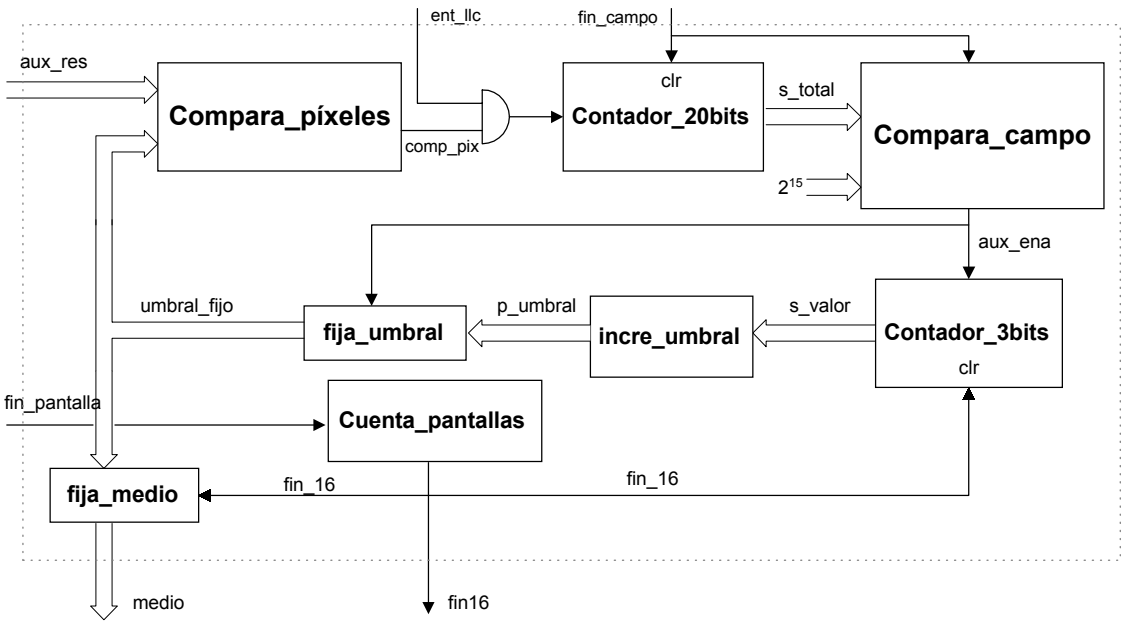


Figura 5.39: *Top de cálculo del valor medio*

En el bloque ***compara_píxeles*** se compara cada píxel *aux_res* con el valor medio fijado hasta el momento: *umbral_fijo*. El bloque ***contador_20bits*** se incrementa cada vez que *aux_res* > *umbral_fijo* y *ent_llc* = '1' (dato o píxel válido). Dicho contador se resetea cada fin de campo. En ese instante, se compara el valor alcanzado por ***contador_20bits*** con la cantidad 2^{15} = 32768 (igual al 8% del número total de píxeles que hay en cada campo, 1440 $\frac{píxeles}{línea}$ x 287 $\frac{líneas\ vídeo\ activo}{campo}$).

Si el resultado de esa comparación es mayor que '1' entonces *aux_ena* = '1' y ***contador_3bits*** se incrementa en una unidad. Su valor de salida *s_valor* será el que determine el próximo umbral de valor medio que se va a emplear, es decir, *p_umbral*:

s_valor	*próx_umbral*
0	64
1	96
2	128
3	160
4	192
5,6,7	224

Ese nuevo valor medio es el que vuelve a utilizarse en el bloque ***compara_píxeles*** cuando *aux_ena* vuelva a ser '1' y se repite todo el proceso de nuevo. Finalmente, cada 16 pantallas (*fin_16* = '1') el valor del bus *umbral_fijo* se toma como el valor medio y se transfiere al bloque que calcula el umbral de intensidad luminosa.

5.3.3.4. CÁLCULO DE LAS COORDENADAS

El punto luminoso captado por cada una de las cámaras vendrá dado por un conjunto de píxeles que forman un área cuyo centro de gravedad ($\bar{x}, \bar{y}$) se calcula como:

$$\bar{x} = \frac{1}{n}\sum_i x_i \qquad \bar{y} = \frac{1}{n}\sum_i y_i \qquad i = 1,2,...n$$

siendo n el número de píxeles, y_i la línea de vídeo en la que se encuentra el i-ésimo píxel (es decir, la *fila*) y x_i la posición que ocupa el píxel dentro de dicha línea (es decir, la *columna*).

Para determinar la posición del punto luminoso, sólo se tendrán en cuenta los valores *columna-fila* de los píxeles cuyo valor supera al umbral.

Los valores *columna-fila* de todos y cada uno de los píxeles que forman una imagen son proporcionados por el bloque ***coordenadas_xy***, a través de los buses de salida: *resultx* y *resulty*, tal y como se explicó en el apartado 5.3.3.2. del presente Capítulo.

Sólo cuando el resultado de comparar el valor del píxel con el umbral sea mayor que '1', cada uno de esos valores, se cargará por separado en sendos registros, habilitados con la señal *s_resulta* = '1' (ver *figura 5.40)*. Posteriormente, dichos valores pasan al los bloques ***calc_coord***, donde se calculan de forma paralela las coordenadas: $\bar{x}$ e $\bar{y}$, que formarán el centro de gravedad del área del punto luminoso en la pantalla.

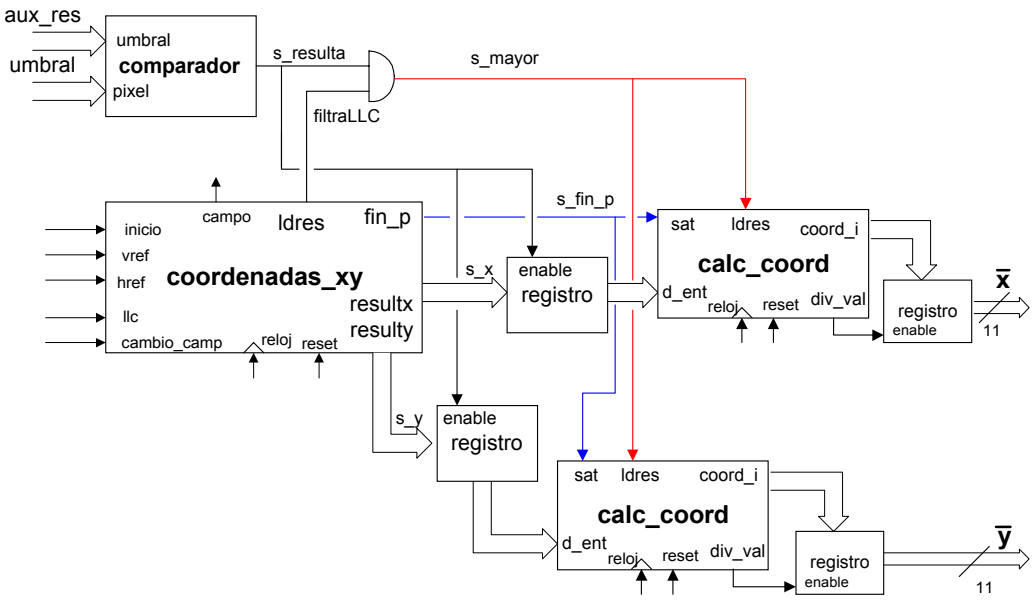


Figura 5.40: *Bloques relacionados con el cálculo de las coordenadas* $\bar{x}$ *e* $\bar{y}$

A continuación se describe con más detalle cómo el circuito ***calc_coord*** realiza el cálculo de $\bar{x}$ (o $\bar{y}$). Tal y como se observa en la *figura 5.41*, este bloque está formado por dos sumadores y un divisor, que son los circuitos necesarios para llevar a cabo la fórmula:

$$\bar{x} = \frac{1}{n}\sum_i x_i \quad (\text{ó } \bar{y} = \frac{1}{n}\sum_i y_i) \qquad i = 1,2,...n$$

Mientras un circuito sumador se encarga de calcular el área, es decir, contar el número se píxeles cuyo valor es mayor que el umbral, es decir: $n = \sum_i^n 1$, el otro, va sumando los valores de las columnas x_i (o de las filas y_i) , es decir: $\sum_i^n x_i$ (ó $\sum_i^n y_i$).

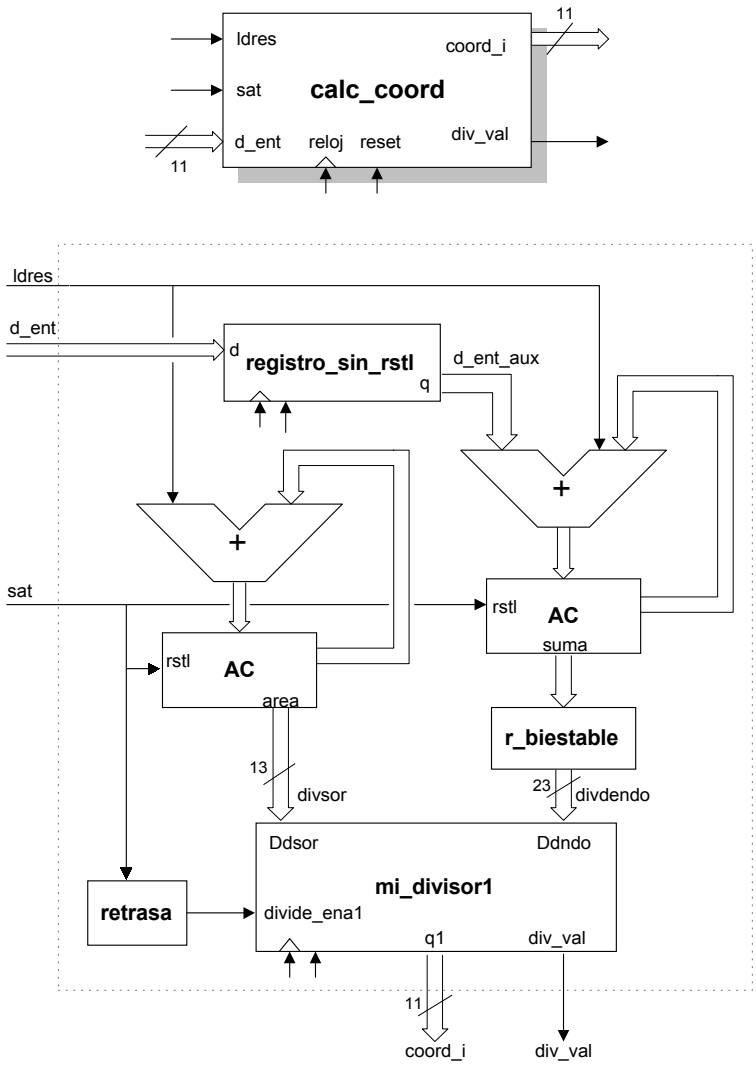


Figura 5.41: *Cálculo de la coordenada $\bar{x}$ (o $\bar{y}$)*

Al final de cada pantalla (*sat* = '1') el contenido de los acumuladores pasa al bloque divisor, donde tras 29 ciclos de reloj, o lo que es lo mismo, 29 x 20 ns = 480 ns, se obtiene el valor $\bar{x}$ (o $\bar{y}$). Teniendo en cuenta que el periodo de *blanking* o supresión del haz vertical es de 1.6 ms, podemos concluir que aunque la división requiere más ciclos de reloj que otras operaciones realizadas por el sistema, no limita en absoluto, en cuanto a velocidad, la capacidad del diseño de trabajar en tiempo real.

Una vez obtenidos, de forma paralela, los centros de gravedad:

- $(\bar{x}_1, \bar{y}_1)$ para la imagen procedente de la cámara1.
- $(\bar{y}_2, \bar{z}_2)$ para la imagen procedente de la cámara2.
- $(\bar{z}_3, \bar{x}_3)$ para la imagen procedente de la cámara3.

Se calculan las coordenadas finales del punto real de soldadura 3D como el promedio de los valores anteriores:

$$X = \frac{\bar{x}_1 + \bar{x}_3}{2} \quad Y = \frac{\bar{y}_1 + \bar{y}_2}{2} \quad Z = \frac{\bar{z}_2 + \bar{z}_3}{2}$$

Par ello se cuenta con el bloque ***promedia2***, que como se puede ver en la siguiente figura, se instancia tres veces en el diseño.

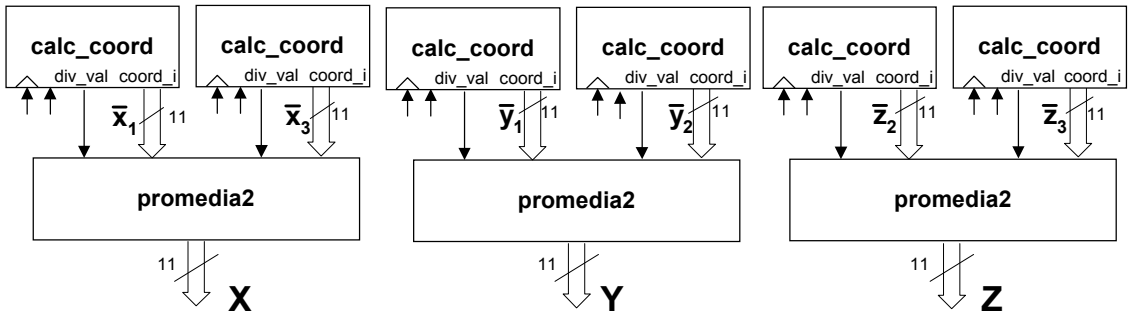


Figura 5.42: *Cálculo de las coordenadas finales X, Y, Z*

5.3.3.5. INTERFAZ VGA

Una vez calculadas las coordenadas finales: X, Y, Z, el sistema nos ofrece la posibilidad de representarlas en un monitor VGA. De esto se encarga el circuito mostrado en la siguiente figura:

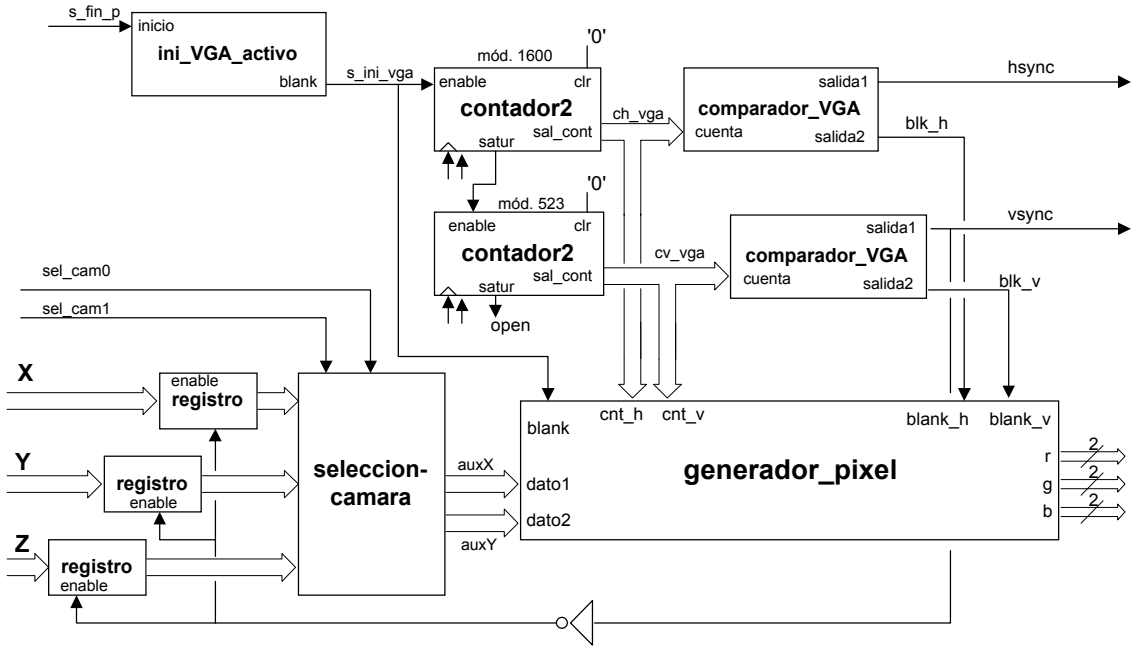


Figura 5.43: *Top para representar las coordenadas en la pantalla de un monitor VGA*

En primer lugar hay que activar el proceso. Para ello, contamos con el bloque ***ini_VGA_activo***, que tras recibir un pulso a su entrada, genera una señal de salida *s_ini_vga*, que permanecerá a nivel alto hasta que se reciba un reset. La señal que activará este bloque será la que indica el fin de pantalla.

A partir de ese instante, se pone en funcionamiento el contador de sincronismo horizontal ***contador2***. El valor de su cuenta, *ch_vga*, se emplea en el bloque ***comparador_VGA*** para generar las señales de sincronismo y blanking horizontal: *hsync* y *blk_h*, tal y como se muestra en la siguiente figura.

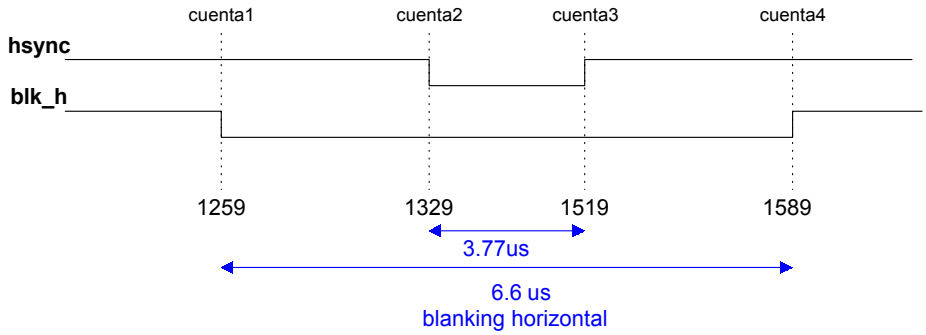


Figura 5.44: *Señales de sincronismo horizontal*

Cada vez que concluye el periodo de una línea VGA, el contador de sincronismo horizontal satura y el contador de sincronismo vertical se incrementa en una unidad. Dicho contador se encarga de contar las filas de la imagen VGA, la cual está compuesta por 528 líneas de las que sólo 480 son visibles. El valor de esta cuenta, *cv*_vga, se utiliza para generar las señales de sincronismo y blanking vertical: *vsync* y *blk_v*.

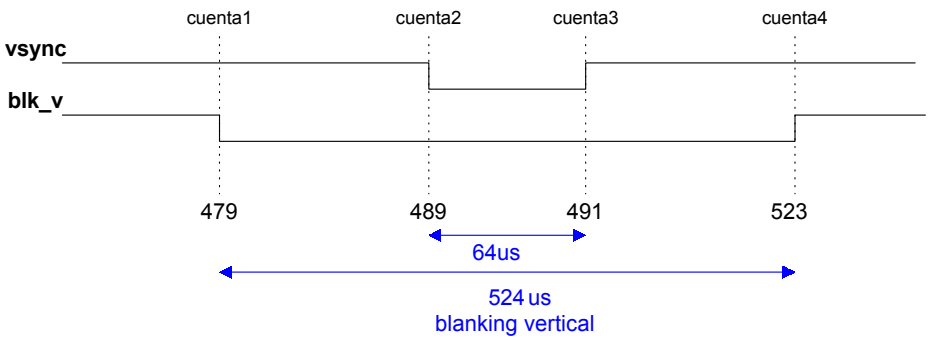


Figura 5.45: *Señales de sincronismo vertical*

El circuito ***generador_píxel*** es el encargado de determinar el color del píxel (blanco o negro) que se envía al monitor VGA. Para ello cuenta con tres buses de salida de 2 bits cada uno: *r* (color rojo), *g* (color verde), *b* (color azul).

Las entradas *s_ini_vga*, *blk_h* y *blk_v* se usan para determinar los periodos de blanking, de modo que si alguna de ellas es cero, en la pantalla del monitor sólo se dibujarán píxeles en negro. Las condiciones para que este circuito envíe píxeles de color blanco al monitor son:

- No estar en periodo de blanking.

- Los valores de las coordenadas X, Y dados por *auxX* y *auxY* coincidan con los valores del contador de sincronismo horizontal y vertical respectivamente: *ch_vga* y *cv_vga*.

Previamente, mediante las líneas *sel_cam1* y *sel_cam0*, se puede seleccionar desde qué perspectiva se quiere ver el punto, si desde el punto de vista de la cámara 1, o desde el de la cámara 2 ó la cámara 3:

- Cámara 1: *sel_cam1* = '0' → auxX = **X**, *auxY* = **Y**.
- Cámara 2: *sel_cam1* = '1' y *sel_cam0* = '0'→ *auxX* = **Y**, *auxY* = **Z**.
- Cámara 3: *sel_cam1* = '1' y *sel_cam0* = '1'→ *auxX* = **Z**, *auxY* = **X**.

Hay que señalar la utilización de la señal *vsync* negada para sincronizar este bloque de interfaz VGA con el circuito que calcula las coordenadas. Es necesario llevarlo a cabo ya que la frecuencia a la que se obtienen las coordenadas finales es menor que la frecuencia a la que se escribe una imagen VGA completa. La duración de una imagen de vídeo activo era de 18 ms. Si añadimos el tiempo empleado en calcular las coordenadas, se tienen 18.460 ms. Mientras que la duración de una imagen VGA es de 15.25 ms. Por tanto, para evitar cambios en los valores de las coordenadas X, Y, Z en mitad de un *frame* VGA, se añaden unos registros que se actualizarán cada vez que termina de representarse una imagen VGA.

5.3.3.6. COMUNICACIÓN CON EL PC

Además de representar las coordenadas finales X, Y, Z en un monitor VGA, éstas pueden ser transferidas a un PC a través del puerto paralelo, según el protocolo EPP[26]. Se trata de un enlace bidireccional, controlado por *hardware*, que soporta

[26] EPP significa *Enhanced Parallel Port* (puerto paralelo "mejorado"), y se describe con detalle en el estándar IEEE-1284.

velocidades comprendidas entre los 500Kbytes/s y los 2Mbytes/s, que es más que suficiente para nuestro proyecto.

En el diseño, la máquina de estados ***fsmepp*** será la encargada de que la comunicación entre la FPGA y el PC se lleve a cabo según el protocolo EPP. Su diseño nos fue facilitado por parte del tutor del proyecto.

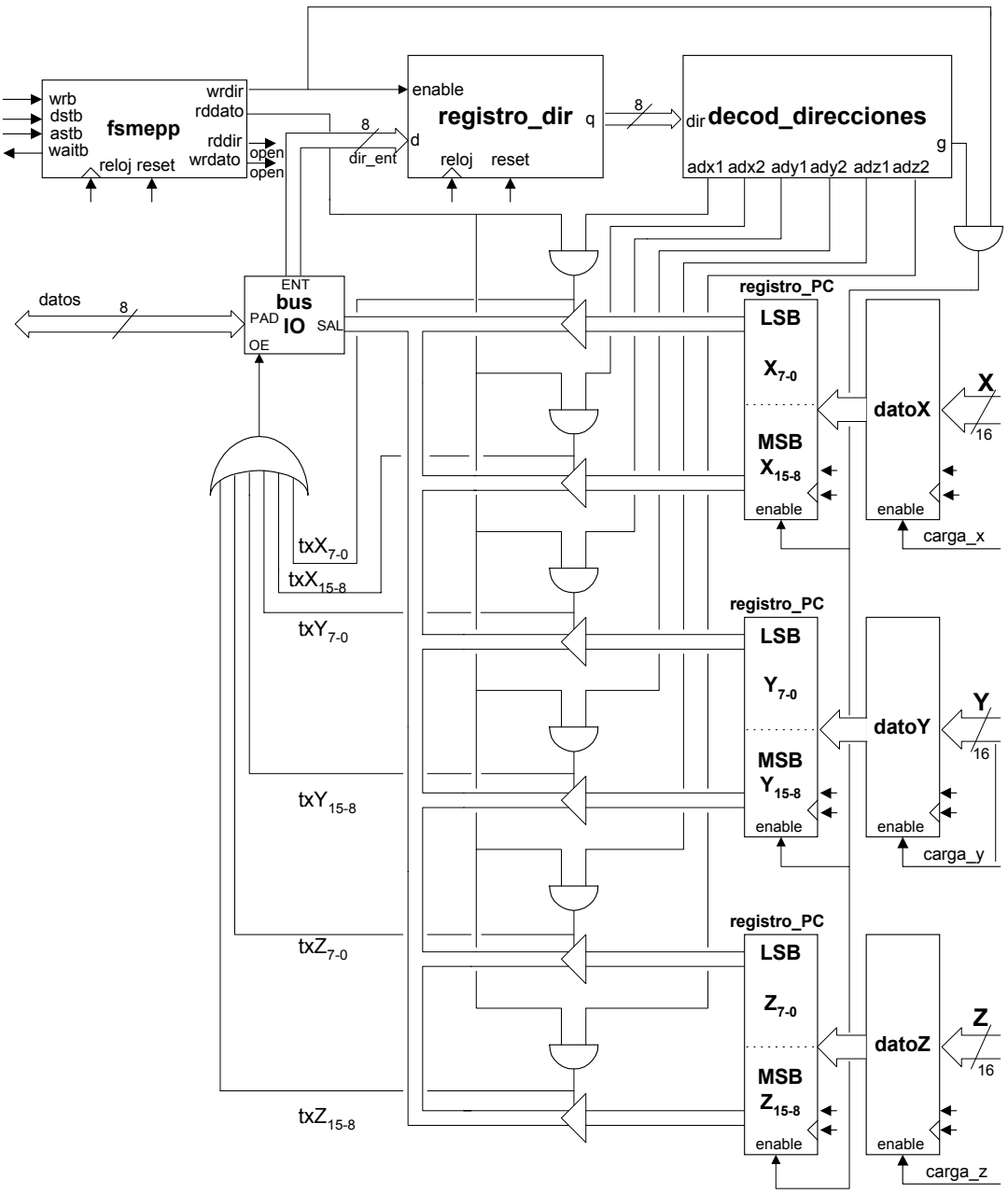


Figura 5.46: *Top del interfaz FPGA-PC*

En nuestro caso, la única operación que se realiza es la lectura de datos, por parte del PC, almacenados en determinadas direcciones conocidas por el usuario. De ahí que no utilicemos las señales *rddir* y *wrdato*. Estas señales son la que controlarían la escritura de datos PC→FPGA.

El puerto paralelo dispone de 14 bits: 8 bits para los datos I/O, 4 bits para señales de control y 2 bits libres. Los datos se transfieren de 8 en 8 bits. Como las coordenadas finales tienen un tamaño de 11 bits, lo primero que se hace es expandirlas hasta un tamaño de los 16 bits (2 bytes). De este modo, para enviar, cada coordenada completa, primero se transfiere el byte menos significativo (bits 7-0) y después el más significativo (bits 8-15).

Cada vez que se quieren leer las tres coordenadas se realizan los siguientes pasos:

- 1º) PC → FPGA: *Writ AD(g, 0XFF)* : instrucción para que se carguen a la vez las coordenadas en los registros ***registro_PC***. Con esta instrucción se activa la señal *wrdir* permitiéndose la carga de la dirección de *g* en el registro de direcciones: ***registro_dir***. Esta dirección es decodificada, generándose la señal *g* (get), que junto con *wrdir*, activan la carga de las coordenadas.
- 2º) *Read AD(Ax1, &x1)* : instrucción para leer el byte menos significativo de la coordenada X. De nuevo, *wrdir* = '1', la dirección Ax1 se carga en ***registro_dir*** y se decodifica, activándose la señal *adx1*, que junto con *rddato*, activan el bloque triestado, permitiendo que el byte menos significativo de X (x1), pase al PC.
- 3º) *Read AD(Ax2, &x2)* : igual que la anterior salvo que ahora se transfiere x2: el byte más significativo de X.
- 4º) *Read AD(Ay1, &y1)* : instrucción para enviar y1: el byte menos significativo de Y.
- 5º) *Read AD(Ay2, &y2)* : instrucción para enviar y2: el byte más significativo de Y.
- 6º) *Read AD(Az1, &z1)* : instrucción para enviar z1: el byte menos significativo de Z.

- 7º) *Read AD(Az2, &z2)* : instrucción para enviar z2: el byte más significativo de Z.

Una vez enviadas las tres coordenadas, se repetiría el proceso para seguir leyendo otro triplete nuevo.

5.3.3.7. DLL

La familia Virtex incorpora hasta ocho DLLs en las FPGAs con las que se consigue un retraso cero en el reloj y un desplazamiento del reloj muy bajo entre las salidas síncronas distribuidas a lo largo del chip. Estas DLLs se ubican en las cuatro esquinas de la FPGA. Sin partes de la FPGA dedicadas a éstas en exclusiva. Permiten disponer de hasta ocho diseños con relojes independientes.

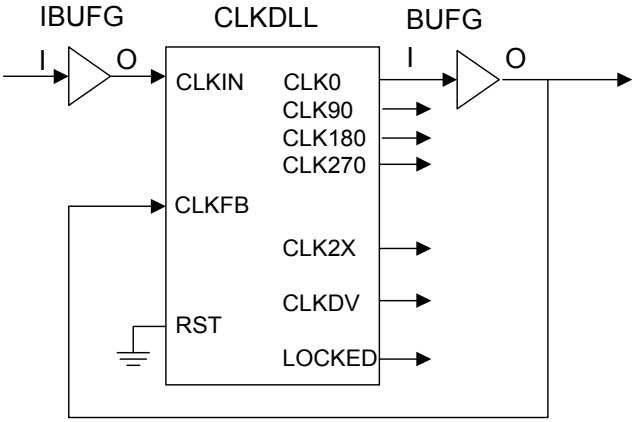


Figura 5.47: *Esquema de BUFGDLL*

La DLL resulta indispensable para el funcionamiento de los diseños descargados sobre la FPGAs de la familia Virtex, ya que es tal el tamaño de la FPGA que sin un control en los desplazamientos en el reloj los diseños no resultan viables. Es por ello que la hemos ubicado en el TOP del diseño.

Otra utilidad de las DLLs es que permite multiplicar por 2 la frecuencia del reloj o dividirla por distintos factores. En el bloque de *captura* y *sincronización* se hace uso del reloj **clk2x** (*reloj2*) de 100MHz, además del reloj maestro del sistema: **clk0** (*reloj*), de 50MHz de frecuencia.

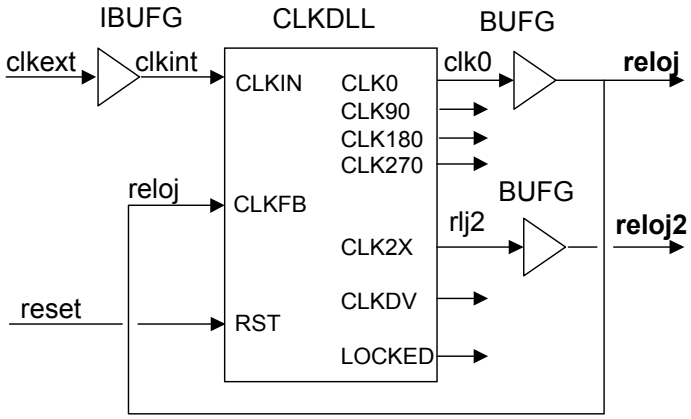


Figura 5.48: *Obtención de las señales de reloj de nuestro diseño*

5.3.3.8. *READBACK*: ELEMENTO CAPTURE_VIRTEX

El elemento ***capture*** aprovecha las nuevas características para el depurado de la familia de FPGAs Virtex. Este elemento permite imponer la condición de captura que deben cumplir las señales del diseño que se le indiquen para realizar la captura en la memoria de configuración del estado del chip. El sistema UNSHADES-1 nos permite la lectura de esta memoria de configuración y presenta los valores de las señales internas de nuestro circuito.

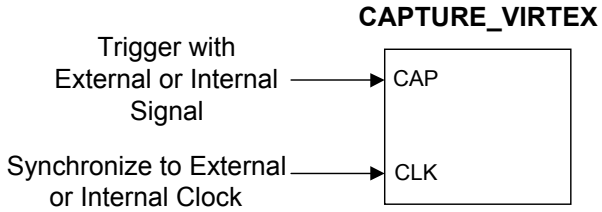


Figura 5.49: *Elemento Capture_Virtex, de la librería Virtex*

Para Permitir el *readback*, como éste se realiza a través del interfaz SelecMAP después de la configuración de la FPGA, habrá que configurar este puesto para que continúe activo, para ello hay que establecerlo en el proceso de configuración de la FPGA, con la orden: ***bitgen –g persist:X8***.

La captura e interpretación de los datos que nos envía la FPGA se realiza a través del sistema UNSHADES-1.