Práctica 7

Proyecto de diseño completo e integración en un SoC

Objetivos

Realizar en VHDL un diseño completo desde cero, con un nivel de complejidad más elevado que en prácticas anteriores.

Familiarizarse con la estructura de diseños electrónicos System-on-Chip (SoC) mediante un ejemplo sencillo.

NOTA IMPORTANTE: En la parte de la práctica dedicada a la integración del diseño creado en una estructura SoC se trabaja sobre un diseño incompleto, entregándose a los alumnos ficheros VHDL listos para su uso. Los alumnos deberán comprender todo el código. A la hora de evaluar la práctica se podrán hacer preguntas sobre cualquier parte del diseño.

Descripción del bloque a diseñar: controlador VGA

El diseño a realizar completamente por los alumnos es un controlador gráfico VGA (resolución 640 píxeles horizontales x 480 píxeles verticales) con 15 colores por píxel. Este diseño se implementará en la FPGA de la placa *Spartan 3 Starter Kit* que se ha usado ya para otras prácticas, que dispone de un conector VGA. La forma de ejercitar este controlador será integrarlo en un diseño simple tipo "System-on-Chip", es decir con un microprocesador integrado en la FPGA, como veremos más abajo.

La siguiente figura muestra los puertos que tendrá nuestro controlador VGA:

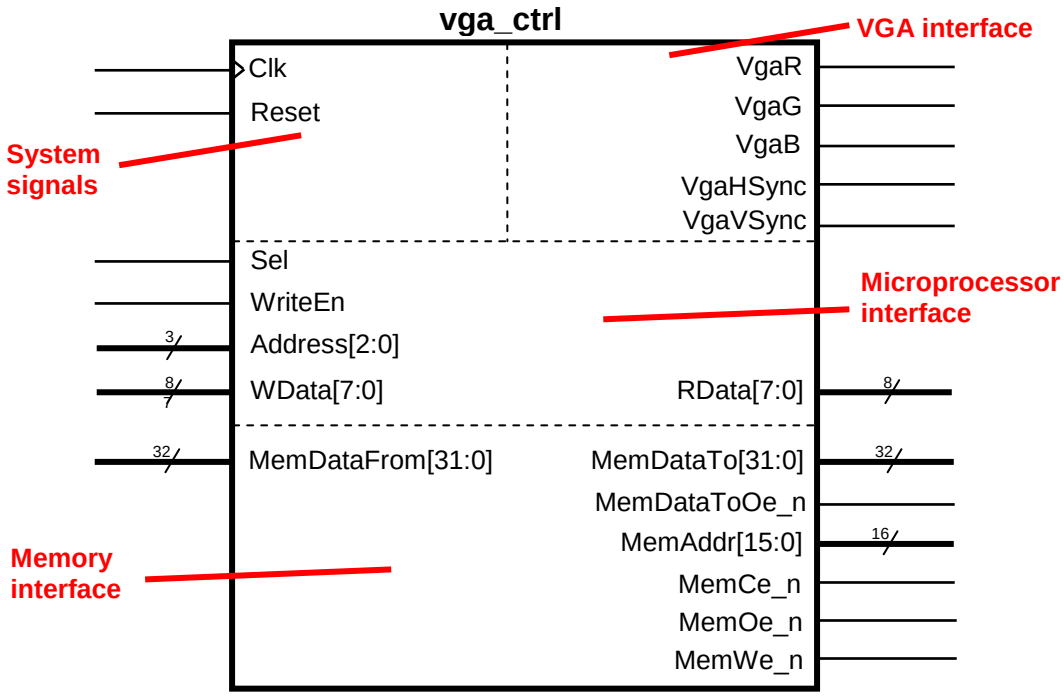


Figura 1 : Puertos del bloque controlador VGA, *vga_ctrl*

Como se muestra en la figura, el bloque tiene cuatro conjuntos de puertos diferenciados:

- Señales "de sistema": reloj (activo en flanco de subida) y reset asíncrono (activo a nivel alto), que deberán usar todos los flip-flops del diseño.
- Interfaz con el microprocesador: sirve para que el microprocesador escriba píxeles en la pantalla, definiendo la posición de un píxel y su color (ver más abajo). El interfaz es de 8 bits y funciona de la siguiente manera:
    - Si el bloque está seleccionado (*Sel* = 1) y la señal de escritura está activada (*WriteEn* = 1), se escribirá el dato *WData* en el registro interno de dirección *Address*.
    - En todo momento el bloque proporcionará en *RData* el valor del registro interno de dirección *Address*.
- Interfaz con la memoria que juega el papel de buffer gráfico (memoria de vídeo), donde se almacena el valor de color de cada píxel en la pantalla. Esta memoria no estará localizada en la FPGA (es demasiado grande), sino en los chips de memoria SRAM (Static RAM) de la placa. El funcionamiento de sus puertos se corresponde con el de estos chips de memoria, pero en este caso el interfaz se simplifica al presentar sólo las señales necesarias:
    - *MemDataTo* y *MemDataFrom* son los datos hacia y desde la memoria, de 32 bits.
    - *MemAddr* es la dirección de la memoria.
    - *MemCe_n* es el "chip enable" de la memoria, activo a nivel bajo. Cuando está a '0' la memoria lee o escribe, cuando está a '1' se sitúa en un estado inactivo de bajo consumo.
    - *MemWe_n* es el enable de escritura en la memoria, un '0' le dice a la memoria que escriba.
    - *MemOe_n* es el enable de salida de datos de la memoria, que presenta un interfaz de datos bidireccional. Si *MemOe_n* es '0' la memoria excitará su salida de datos permitiendo lecturas, mientras que si es '1' esos pines quedarán en alta impedancia, listos para funcionar como entrada en las escrituras.
    - *MemDataToOe_n* es una señal que no está pensada para conectarse a la memoria, sino para uso interno a la FPGA, siendo la contrapartida de *MemOe_n* (de hecho una implementación alternativa podría ser que una fuera la otra negada): en la FPGA se ha de implementar un buffer bidireccional, de forma que se excite la salida con el dato *MemDataTo* si *MemDataToOe_n* es '0' y se deje la salida flotar (bits a 'Z') si esta señal de control es '1'.
- Interfaz VGA: consiste en una salida por cada uno de los tres colores básicos (*R*ed, *G*reen, *B*lue) más dos señales de sincronismo, horizontal (*VgaHSync*) y vertical (*VgaVSync*).

En las siguientes secciones se describirán más detalles sobre estos interfaces.

Interfaz con el microprocesador: registros de usuario

Esta interfaz permitirá al micro encender un pixel en una posición y color determinados.

Se implementarán los siguientes registros:

Tabla 1 : Registros de usuario

Dirección (*Address[2:0]*)	**Nombre**	**Función**
000	POSX_L	Parta baja de la posición horizontal del pixel.
001	POSX_H	Parte alta de la posición horizontal del pixel (sólo los bits [1:0] se utilizan).
010	POSY_L	Parta baja de la posición vertical del pixel.
011	POSY_H	Parte alta de la posición vertical del pixel (sólo el bit [0] se utiliza).
100	PIXDATA	Color del pixel. Al escribir este registro se produce la escritura en el buffer gráfico (memoria SRAM).

Los dos primeros registros concatenados forman un valor de 10 bits que permite especificar la coordenada X (válida entre 0 y 639). Igualmente los dos siguientes registros permiten especificar la coordenada Y (válida entre 0 y 479). Las coordenadas en la pantalla se disponen como muestra la siguiente figura:

Figura 2 : Coordenadas en la pantalla

Una vez escritas las coordenadas, el micro "encargará" la escritura del pixel en memoria gráfica escribiendo en el registro PIXDATA el color deseado.

No es necesario escribir siempre los 5 los registros. Si por ejemplo se desea después escribir el pixel inmediatamente a la derecha (suponiendo que no se salga de la pantalla) bastará cambiar uno o ambos registros POSX* y volver a escribir en PIXDATA.

En esta práctica, en principio, sólo debemos preocuparnos de que estas funciones de los registros se efectúen correctamente, no de hacer ninguna escritura en estos registros. Las escrituras las hará el microprocesador con un programa proporcionado. Lo que sí podremos hacer es ver en la simulación cómo se van realizando estas escrituras.

Los registros deberán ser todos de "lectura/escritura", es decir, el micro podrá leer el último valor escrito. Sin embargo este interfaz NO está pensado para que el micro lea el contenido de un pixel en la memoria gráfica. La lectura de PIXDATA devolverá el último valor escrito en este registro, no necesariamente el valor de color del píxel en la posición definida por POSX* y POSY*. En todos los registros los bits no usados se leerán como '0'.

Codificación de color

En el registro PIXDATA y en la memoria gráfica el color de un píxel se codificará mediante un nibble (4 bits):

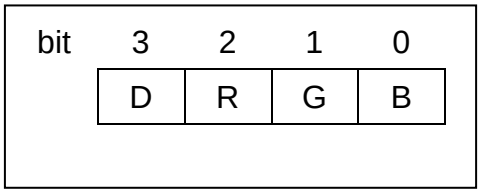


Figura 3 : Codificación de color

En PIXDATA este nibble estará situado en la parte menos significativa del byte (bits 3:0), mientras que la parte más significativa (bits 7:4) no será usada (no afectará en escrituras y los bits se leerán como '0').

Los bits R, G y B corresponden a los tres colores básicos: rojo (*R*ed), verde (*G*reen) y azul (*B*lue). El bit D (*D*ark) permitirá generar colores adicionales oscureciendo las combinaciones de los tres colores básicos[1]. La siguiente tabla muestra los colores resultantes.

Tabla 2 : Colores

D	R	G	B	Color
0	0	0	0	Negro
0	0	0	1	Azul
0	0	1	0	Verde
0	0	1	1	Cian
0	1	0	0	Rojo
0	1	0	1	Magenta
0	1	1	0	Amarillo
0	1	1	1	Blanco
1	0	0	0	Negro oscuro = Negro
1	0	0	1	Azul oscuro
1	0	1	0	Verde oscuro
1	0	1	1	Cian oscuro
1	1	0	0	Rojo oscuro
1	1	0	1	Magenta oscuro
1	1	1	0	Amarillo oscuro
1	1	1	1	Blanco oscuro (gris)

Organización de los píxeles en la memoria de video

La memoria de video tiene que tener suficiente capacidad de almacenamiento para albergar 640 x 480 píxeles x 4 bits / píxel, es decir más de 1.2 Mbits. Esta memoria no está disponible en la FPGA pero sí en la placa, que contiene dos chips de memoria RAM estática (SRAM) de 256k x 16 con los buses dispuestos en paralelo, de forma que se puede considerar como una memoria de 256k x 32.

[1] Con la placa utilizada está bastante limitada la capacidad de generar colores, ya que las líneas R, G y B del conector VGA son digitales (en vez de usarse un DAC por línea), con lo que la generación de colores adicionales sólo puede hacerse mediante algún promediado de píxeles con colores a '0' y '1'.

Utilizaremos estas palabras de 32 bits para almacenar 8 píxeles en cada una, según la disposición de la siguiente figura:

Bits	31…28	27…24	23…20	19…16	15…12	11…8	7…4	3…0
Contenido	píxel X+7	píxel X+6	píxel X+5	píxel X+4	píxel X+3	píxel X+2	píxel X+1	píxel X

Figura 4 : Disposición de los píxeles en las palabras de 32 bits de la memoria de vídeo

En la figura "píxel X" es el valor de color (según la Figura 3) de un píxel en una coordenada horizontal X y "píxel X+N" es el valor correspondiente al píxel en la coordenada horizontal X+N. (Dado que cada palabra define 8 píxeles, la coordenada X será siempre múltiplo de 8).

Palabras consecutivas definirán píxeles sucesivos a lo largo del eje X, hasta completar una línea de vídeo. Con el objeto de facilitar el direccionamiento de la memoria de vídeo, y dado que tenemos capacidad más que suficiente, las líneas de video estarán alineadas a direcciones múltiplo de una potencia de 2, en concreto cada línea empezará en una dirección de palabra múltiplo de 128 (0x80), como se muestra en la siguiente figura:

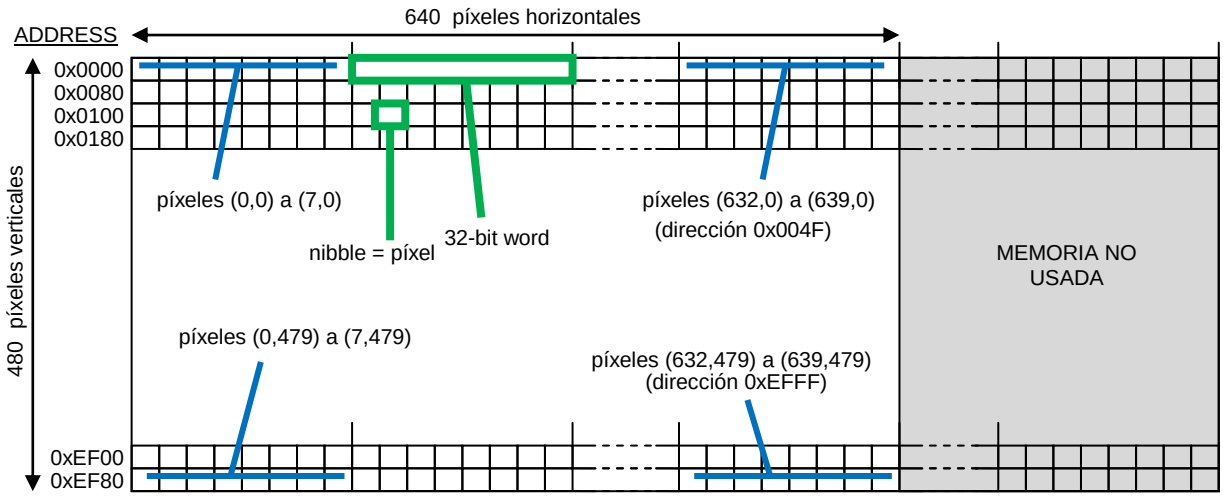


Figura 5 : Mapeado de los píxeles en la memoria de vídeo

Interfaz con la memoria de vídeo

La placa tiene montados dos chips de memoria con la siguiente disposición:

- Dirección, señal de escritura (WE) y señal de habilitación de salida (OE) comunes.
- Datos, chip enable (CE) y señales de enable para la escritura/lectura del byte alto/bajo (UB, LB), independientes.

Esta disposición permite manejar los chips como una memoria conjunta de 256k x 32 en la que cada byte se puede escribir por separado sin alterar el contenido de los otros bytes en la misma palabra (por ejemplo si UB no está activo no se escribirá el byte alto). Por desgracia esto no nos vale directamente, ya que nosotros no manejamos como unidad básica el byte sino el nibble. Así pues utilizaremos la memoria de la siguiente manera:

- Para dibujar en pantalla el contenido de la memoria, el controlador leerá la memoria palabra a palabra, recogiendo de esta forma simultáneamente los datos de 8 píxeles consecutivos. Esta palabra quedará almacenada en registros internos desde los cuales se irá pasando cada uno de los 8 píxeles hacia la salida de vídeo.

- Cuando el microprocesador ordene la escritura de un píxel determinado lo que haremos será ejecutar una operación "Read-Modify-Write", es decir:
    - Calcularemos la dirección de memoria correspondiente al píxel solicitado (según los registros POSX* y POSY*) y leeremos esa palabra, almacenándola en un registro interno.
    - En el registro interno sustituiremos el nibble del píxel deseado por el nuevo dato (PIXDATA).
    - Volveremos a escribir la palabra entera en memoria a partir del registro interno.

Trabajando de esta manera, el conjunto de señales de las memorias en pines del bloque controlador se simplifica un poco respecto al que podemos encontrar en la placa, como se muestra en la siguiente figura, que completa la que aparece en el manual de la placa:

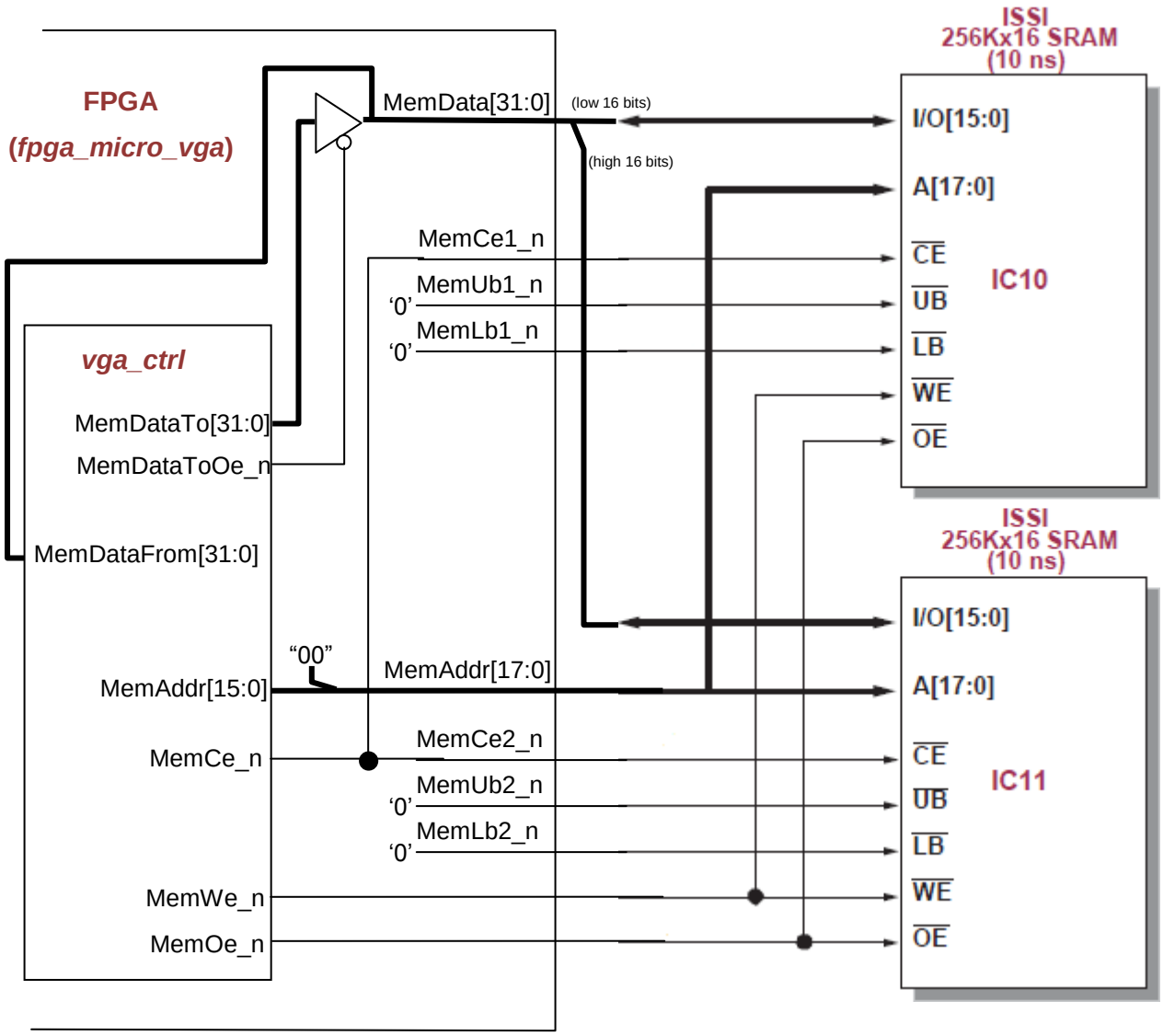


Figura 6 : Interconexión con las memorias de la placa

Los chips de memoria que están montados en la placa son bastante rápidos (tiempo de acceso de 10 ns), y un diseño cuidado permitiría hacer tanto lecturas como escrituras ciclo a ciclo. Sin embargo, para evitar problemas, vamos a seguir un esquema muy conservador para ambos tipos de acceso. Utilizaremos una secuencia de tres estados de las señales de control para las escrituras y otros tres estados para las lecturas, generando unas ondas como las que se pueden ver en la siguiente figura:

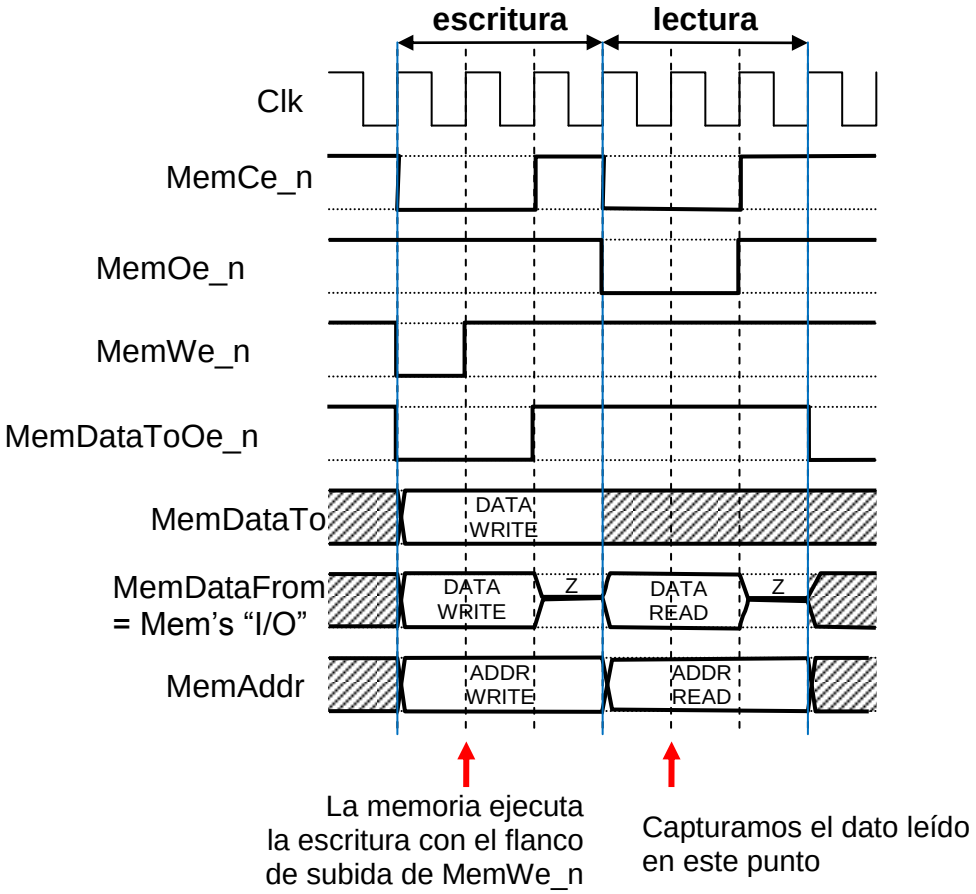


Figura 7 : Ciclos de escritura y lectura en la memoria.

Como se puede ver en la figura, las secuencias propuestas garantizan la estabilidad de dirección y dato en los puntos de escritura o captura del dato leído. También garantizan que no hay colisiones en el bus bidireccional, al guardar siempre un ciclo de alta impedancia entre la excitación eléctrica en distintas direcciones. En estado de reposo (cuando ni se lee ni se escribe), la memoria se deselecciona (*MemCe_n* = '1'), para ahorrar consumo, y la salida de datos de la FPGA queda activada, para evitar tener entradas no excitadas. En la figura se han rayado las zonas de los buses de dirección y datos sin interés: valores en estado de reposo y dato de escritura cuando se está haciendo una lectura. Nótese que nuestro puerto *MemDataFrom* se corresponde con lo que hay en los pines del bus bidireccional de datos de la memoria (pines "*I/O*").

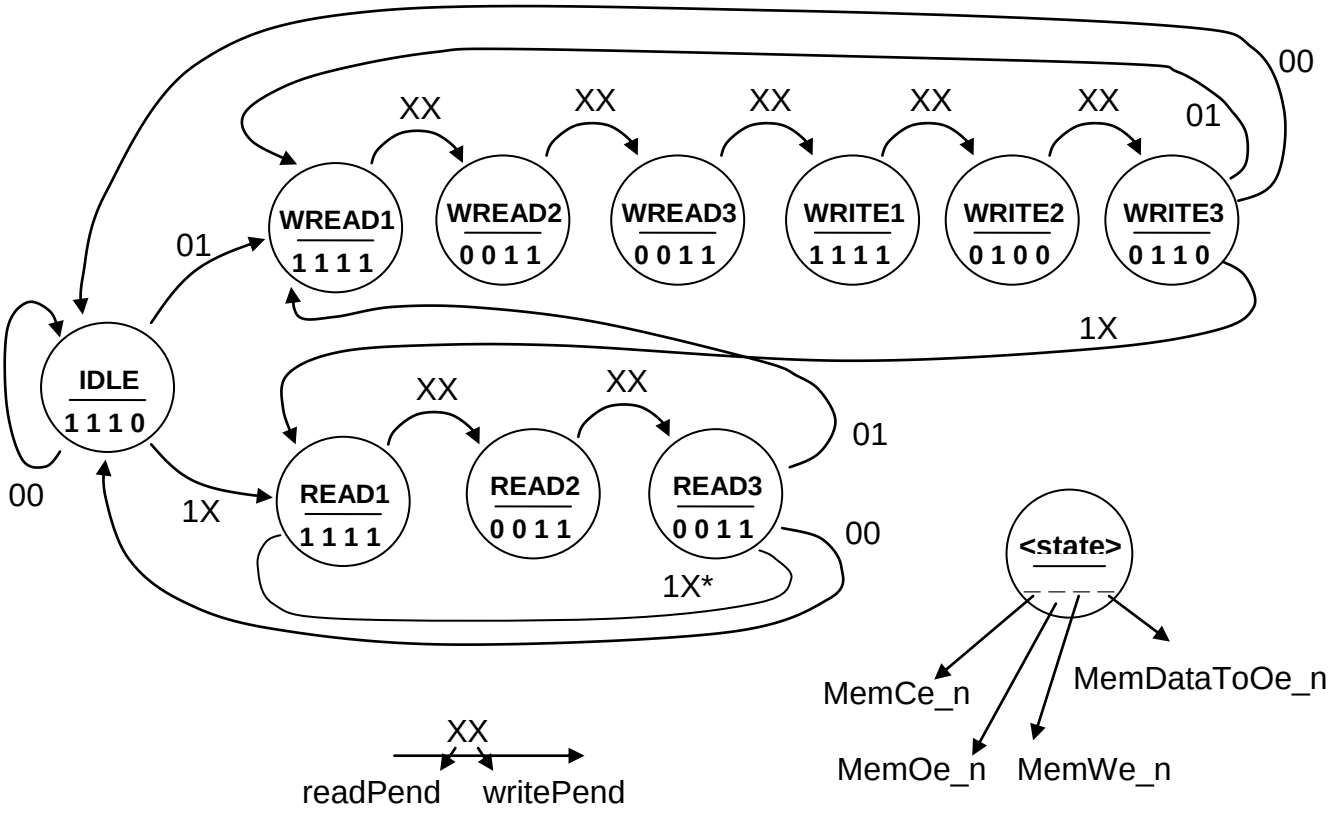
Las señales de control las podemos generar con una máquina de estados que tenga tres estados consecutivos para la escritura y otros tres para la lectura. Además esta máquina tendrá en cuenta que:

- Cada escritura que queramos ejecutar será en realidad una operación Read-Modify-Write, con lo que se pasará por 6 estados consecutivos: 3 de lectura y 3 de escritura.

- La entrada en las secuencias de lectura o escritura se producirá por respectivas peticiones del hardware (por ejemplo podemos tener unas señales internas de lectura pendiente y escritura pendiente, *readPend* y *writePend*).
- Las peticiones de lectura han de tener prioridad sobre las de escritura, para garantizar que siempre se dibuja la pantalla correctamente. La contrapartida será que el software del microprocesador deberá separar sus escrituras un mínimo de ciclos.

Como veremos más abajo, para dibujar la pantalla necesitaremos una nueva palabra de la memoria cada 16 ciclos, así que habría margen hasta para hacer dos escrituras por parte del micro por cada lectura del controlador (6 + 6 + 3 = 15 < 16). En la práctica lo que está claro es que el sistema es factible siempre que se obligue al software a guardar un espaciamiento de al menos 9 ciclos entre las escrituras al registro PIXDATA (6 ciclos para ejecutar la escritura en memoria de PIXDATA y 3 para dejar paso a una posible lectura del controlador).

La siguiente figura muestra un diagrama para la máquina de estados:



* Durante la operativa normal este caso no se puede dar pues las lecturas se hacen separadas 16 ciclos, pero, dependiendo de la implementación, podría ocurrir en una fase de pre-carga de los primeros píxeles de una línea.

Figura 8 : Máquina de estados de accesos a la memoria.

“Timing” de las señales de vídeo

Necesitamos dibujar en pantalla 640x480 = 307200 píxeles cada cuadro (“frame”), y visualizar los cuadros con una frecuencia suficiente para que la señal parezca estática. En esta práctica para esta frecuencia usaremos un valor de 60 Hz, muy común en los adaptadores / monitores comerciales. Si multiplicamos ambas cifras nos da un total de algo más de 18 millones de píxeles por segundo. Adicionalmente, cada cuadro que se visualiza necesita unos tiempos adicionales, en los que no se “pintan” píxeles pero se sacan señales de sincronismo para que los monitores (analógicos) sepan dónde empiezan y terminan líneas y cuadros.

El reloj de nuestra placa va a 50 MHz, es decir, 50 millones de ciclos por segundo. Aun considerando los tiempos necesarios para las señales de sincronismo, nos da tiempo a dedicar 2 ciclos de reloj por cada píxel que dibujemos. Teniendo en cuenta que cada palabra en memoria alberga la información de color de 8 píxeles, necesitaremos leer una palabra de memoria cada 16 ciclos, como se ha comentado más arriba.

El “timing” de las señales de vídeo se hará como se describe en la siguiente figura, donde los distintos tiempos “fuera de la zona visible” han sido ajustados para que la multiplicación hecha arriba nos dé muy aproximadamente los 60 Hz mencionados, resultando en un tamaño total de cuadro de 800 x 521:

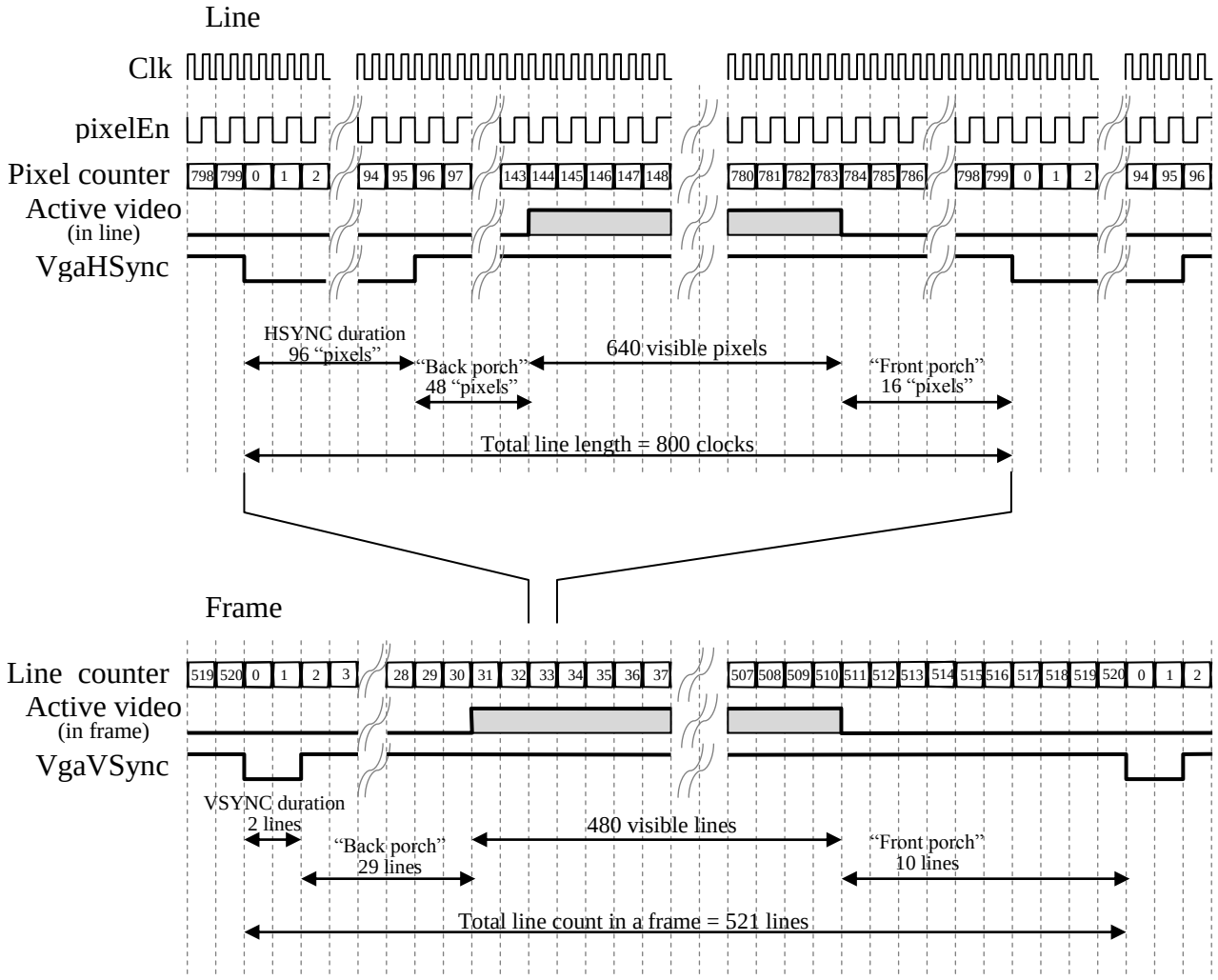


Figura 9 : “Timing” de las señales de vídeo.

En la figura se muestra el reloj del sistema, de 50 MHz, y una posible señal interna *pixelEn* que viene a ser un enable del reloj para que actúe, en los procesos en que sea oportuno, uno de cada dos ciclos, dejando sacar un píxel nuevo a pantalla cada dos ciclos. También se muestran cuentas de número de píxel en la línea y número de línea en el cuadro.

Finalmente se muestran las dos señales de sincronismo que irán al conector VGA (*VgaHSync* y *VgaVSync*), que se ponen a cero al saltar de línea en línea o de cuadro en cuadro, y también las zonas de "vídeo activo" dentro de cada línea y cuadro, que definen el área de 640x480 que realmente se visualiza en pantalla. En esta zona será donde aparezca la señal útil en los pines R,G,B del conector (señales *VgaR*, *VgaG*, *VgaB*).

Uno de los objetivos de la práctica es lograr exactamente la temporización mostrada en la figura entre las señales presentes en el conector (*VgaHSync*, *VgaVSync* y "Active video"={*VgaR*, *VgaG*, *VgaB*}). Es decir, nos tenemos que asegurar de que el monitor vea la temporización especificada, que entenderá como correcta. Es muy normal que en una primera escritura del código VHDL generemos estas señales de forma que algunas se retrasen más debido a la participación de algún registro. Si es así, habrá que retrasar las otras para que al final, en bornas del conector, los tiempos sean los deseados.

Generación de los colores oscuros:

Los colores oscuros sólo serán de utilidad en zonas "rellenas" de color, pues lo que se hará para generar estos colores es apagar selectivamente el píxel en función de su posición en pantalla. Es decir, un cuadrado relleno de color rojo se mostrará como una malla con la mitad de los píxeles rojos y la otra mitad negros. El resultado no es de buena calidad, pero nos permite "jugar" un poco más con el diseño.

Guía para la realización del diseño del controlador

A continuación se comentan algunas funcionalidades más o menos independientes que de una u otra manera hay que implementar en el controlador, con el fin de orientar el desarrollo del código. (Nota: Hay que tener en cuenta que el código no es completamente "lineal", sino que los procesos se "retroalimentan entre sí", de forma que al interpretar las siguientes descripciones aparecerán referencias a posibles señales que serían asignadas por procesos escritos más adelante. Por otra parte, el orden de los procesos en el código puede ser diferente). La implementación puede hacerse de otra manera siempre que se respete una correcta escritura de código sintetizable y una organización limpia y comprensible.

Organización jerárquica

Se puede hacer el diseño con uno o más bloques, pero el nivel de complejidad que tiene permite hacerlo en una sola entidad, y puede resultar más cómodo, recomendándose esta opción.

Definición de constantes

Incluso antes de comenzar a escribir procesos conviene irse planteando la declaración en nuestra arquitectura de algunas constantes de interés, por ejemplo: las direcciones relativas de los registros de I/O para el micro, el número de píxeles por línea y de líneas por cuadro (visibles y en total), la posición de finalización de los pulsos a '0' de las señales de sincronismo, las duraciones de los "back porches", los puntos de comienzo y final de video activo… Si estas constantes se añaden desde el principio o mientras se va escribiendo el código es indiferente, pero cuando haya que usar este tipo de valores siempre se usarán constantes, nunca literales directamente en el código.

Registros de I/O para el micro ("interfaz de usuario")

Para implementar la "interfaz de usuario" para el software del micro, esto es, los registros POSX*, POSY* y PIXDATA, deben escribirse dos procesos diferentes: uno para la escritura de estos registros (proceso secuencial) y otro para la lectura (proceso combinacional). Se puede tomar como referencia el código de uno de los periféricos de PicoBlaze entregados, por ejemplo *gen_interrupt*. En este caso adicionalmente habrá que considerar que una escritura en el registro PIXDATA debe disparar una petición de escritura en memoria (señal *writePend* en la máquina de estados mostrada más arriba, que deberá ponerse a '1' con la escritura en PIXDATA y a '0' cuando se haya empezado la escritura en memoria).

Como variante a lo que se hace en el bloque entregado *gen_interrupt*, puede ser más cómodo para guardar la posición del píxel a dibujar el definir unos registros (*signals*) con el ancho total, no separados en dos cachos como los registros en sí, es decir, por ejemplo podemos definir una señal *posX (9 downto 0)*, y escribir y leer los bits que toque según la dirección con que el micro acceda al bloque.

Máquina de estados

Implementar la máquina de estados expuesta más arriba con el esquema habitual de dos procesos: combinacional para asignar el siguiente estado y secuencial para asignar el estado actual con el siguiente.

Asignación de las señales de control de la memoria

Realizar la asignación de las señales *MemCe_n*, *MemOe_n*, *MemWe_n* y *MemDataToOe_n* en función del estado de la máquina de estados, de acuerdo con lo expuesto más arriba.

Asignación de la dirección para la memoria

En función del tipo de acceso que estemos haciendo, habrá que pasar al bus *MemAddr* una dirección que sea función de la posición (X, Y) solicitada por el micro para la escritura de un píxel o de la posición (X,Y) de un píxel que próximamente haya que visualizar. Esta última posición es más difícil de implementar y requerirá contadores dedicados que tengan en cuenta por un lado la zona de vídeo activa y por otro no ya la coordenada X del píxel que se está visualizando sino la de los que se están leyendo de memoria. Podemos inicialmente declarar y usar las señales correspondientes a esta posición pero dejar su asignación para más adelante, cuando el resto del controlador esté más definido.

Asignación de dato de escritura para la memoria

Como se ha explicado más arriba, el proceso de escritura de un píxel en memoria es en realidad una secuencia *Read-Modify-Write*. Así pues, cada vez que se complete la fase de lectura de esta secuencia se deberá mezclar el dato leído de memoria con el dato que nos ha pedido el microprocesador que escribamos en el píxel correspondiente.

Captura del dato leído de la memoria para su representación en pantalla

Cada cierto tiempo el controlador necesitará leer de memoria la información de nuevos píxeles a representar, solicitando una lectura a la máquina de estados. En el momento adecuado deberá capturarse la información del bus *MemDataFrom* en un registro interno de 32 bits (por ejemplo le podemos llamar *pixWordBuf*), que albergará los colores de 8 nuevos píxeles.

Generación del "enable de píxel"

Se implementará en un proceso la señal *pixelEn* que se ha comentado más arriba, que soporta el hecho de que el controlador visualice un píxel cada dos ciclos de reloj. A esta señal se le dará inicialmente el valor '0'.

Contadores de píxeles y de líneas

Deberán implementarse unos registros para llevar la cuenta de por qué número de píxel y línea va el proceso de "barrido" de la pantalla (teniendo en cuenta zonas visibles y no visibles). Asociado a esto puede ser de utilidad implementar unas señales que se activen justo cuando vaya a cambiar la línea o el cuadro, ya que estas señales nos pueden valer en otros procesos.

Generación de las señales de sincronismo

Las señales *VgaHSync* y *VgaVSync* se generarán de acuerdo con la posición actual de píxel y línea. Según como se haga la implementación puede ser que inicialmente se genere una versión "previa" de estas señales que más adelante haya que retrasar uno o más ciclos para mantener una correcta distancia con la señal de vídeo activo.

Asignación sucesiva de los píxeles contiguos

Tendremos que hacer una conversión de paralelo a serie con el dato que leamos de la memoria, ya que este tiene 8 píxeles y nosotros queremos ir sacando al conector píxel a píxel. Para ello podemos copiar el contenido del registro en el que capturamos la lectura de memoria sobre otro registro y de este ir seleccionando uno a uno los píxeles (por ejemplo efectuando desplazamientos de 4 en 4 bits y usando siempre los mismos 4 bits del registro para la salida a pantalla). Hay que tener en cuenta que, para no quedarnos sin datos que sacar a la pantalla, cuando se nos acaban los píxeles de este registro lo tendremos que recargar con el dato del registro paralelo (al que más arriba se ha llamado *pixWordBuf*) y en ese momento encargar una nueva lectura de memoria.

Un detalle que nos complica las cosas es que, aparte de la "dinámica" normal de estos registros, habrá que "precargarlos" antes de que toque visualizar el primer píxel de cada línea. Para esto se puede encargar una actualización de los mismos por ejemplo coincidiendo con el final del pulso de sincronismo horizontal.

Asignación de las señales de color e implementación de los colores oscuros

Como se comenta más arriba un píxel color oscuro es simplemente un píxel que en función de su posición recibe su color o se muestra como negro. Para conseguir este efecto en forma de malla basta con hacer que la señal que diga si ponemos el píxel a cero (cuando se trate de un color oscuro) sea el *or exclusivo* del bit menos significativo de las cuentas de píxel y línea.

Teniendo esto en cuenta, podemos asignar las señales *VgaR*, *VgaG* y *VgaB* a partir del dato de color de píxel en curso, según lo indicado anteriormente.

Diseño completo: el "SoC" con PicoBlaze y el controlador

Integraremos el controlador VGA en un diseño de FPGA, en el cual actuará como periférico de un microcontrolador *PicoBlaze*, junto a otros periféricos adicionales.

PicoBlaze es un microcontrolador de 8 bits diseñado especialmente para ser implementado en una FPGA de las familias Virtex/Spartan-II ó Virtex-II/Spartan-3. El diseño ocupa unas 96 *slices* en Spartan 3, tiene un juego de instrucciones reducido, 16 registros, 256 puertos direccionables, 64 posiciones de memoria *scratch-pad*, interrupción enmascarable, y puede obtener un rendimiento en MIPS igual a la mitad de la frecuencia de reloj utilizada (ej.: 25 MIPS a 50 MHz). La memoria de programa que se puede conectar al *PicoBlaze* es de hasta 1024 instrucciones (está pensado para utilizar exactamente una *Block RAM* de la FPGA).

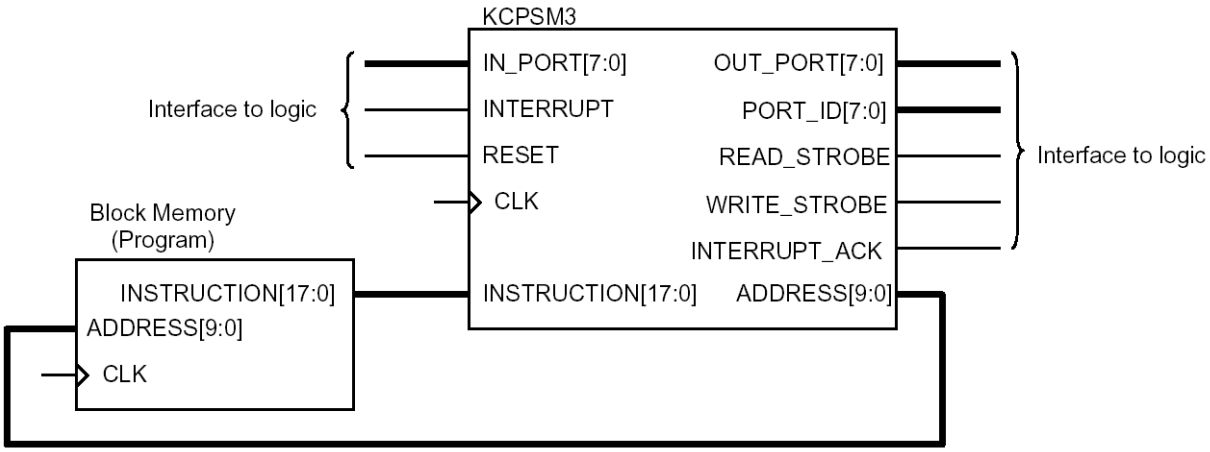


Figura 10 : Diagrama de conexión del PicoBlaze con una memoria externa de 1024 palabras

En la página web del laboratorio está disponible un archivo con información sobre este microprocesador. Se recomienda comenzar consultando las primeras páginas de la presentación "KCPSM3_Manual.pdf", aunque para esta práctica no será necesario un conocimiento amplio del microprocesador ni de su juego de instrucciones.

En la siguiente figura se muestra el diseño de la fpga, *fpga_micro_vga*. En la web del laboratorio se dispone de un archivo con la mayoría de los bloques de este diseño, teniendo los alumnos que diseñar el "top-level" y también el bloque *decoder*.

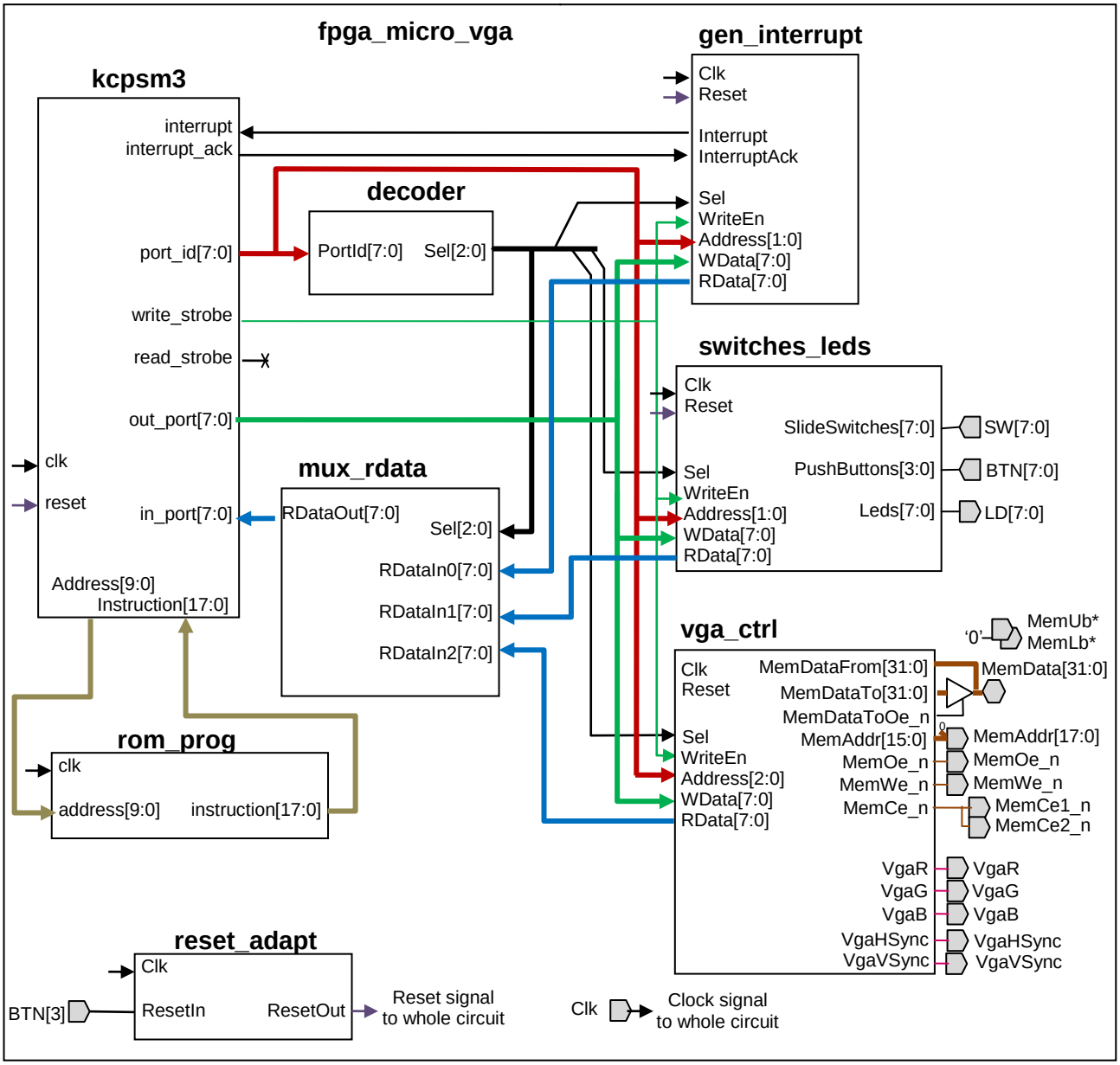


Figura 11 : Esquema de bloques del diseño de FPGA

Puertos del circuito:

Entradas

- *Clk* — Reloj del sistema, generado externamente en la placa.
- *SW[7:0]* – Puerto conectado a los "slide switches" (conmutadores) de la placa S3BOARD (*Spartan-3 Starter Kit Board*). En el diseño básico no tendrán ninguna utilidad real respecto a la salida VGA, simplemente la FPGA los volcará sobre los LEDs de la placa mediante el software del *PicoBlaze*.
- *BTN[3:0]* – Puerto conectado a los "push button switches" (pulsadores) de la placa. Un botón pulsado da un nivel alto en la entrada. Se usará el botón 3 (BTN3) como reset general del circuito. Los demás botones pueden ser leídos por el software pero no se utilizan en el diseño básico salvo para controlar el parpadeo los LEDs.

Salidas

- *LD[7:0]* – Puerto conectado a los LEDs de la placa (líneas activas a nivel alto).
- *MemAddr[17:0]*, *MemCe1_n*, *MemCe2_n*, *MemWe_n*, *MemOe_n*, *MemLb1_n*, *MemUb1_n*, *MemLb2_n*, *MemUb2_n* – Señales de dirección y control hacia la memoria SRAM de la placa.

Bidireccionales

- *MemData[31:0]* – Datos de escritura/lectura para la memoria SRAM de la placa.

Bloques:

- **fpga_micro_vga** : Éste es el top-level del circuito. Los alumnos deberán desarrollar este bloque completamente, instanciando y conectando los demás bloques.
- **kcpsm3** : Este bloque es el microprocesador *PicoBlaze* ("(K)Constant Coded Programmable State Machine 3"). Consultar la documentación del mismo para más detalles.
- **rom_prog** : Memoria de programa. El código VHDL de este bloque lo generan automáticamente las herramientas de *PicoBlaze* (p.ej. *pBlaze IDE*) a partir del código ensamblador. Este fichero VHDL se entrega a los alumnos, conteniendo un programa de prueba del bloque controlador VGA, de forma que no es necesario trabajar con el software.
- **decoder** : Este bloque define el mapa de memoria de los distintos periféricos que va a utilizar el micro. En función del valor de la salida del micro *port_id* activará una sola señal de selección (un bit de su puerto de salida *Sel[2:0]*). Este bloque, muy simple, deberá ser desarrollado enteramente por los alumnos, implementando el siguiente mapa de memoria:

Tabla 3 : Mapa de direcciones base de periféricos

PERIFÉRICO	DIRECCIÓN BASE (en hexadecimal)
gen_interrupt	20
switches_leds	28
vga_ctrl	40

A cada periférico se le asignará un espacio de 8 direcciones. Así por ejemplo, la señal de selección para el bloque *switches_leds* (dirección base = 28h) se activará si *port_id* = 00101xxx , donde 'x' significa que cualquier valor es válido.

- **mux_rdata** : Multiplexor para la lectura de datos de puertos de entrada por parte del micro. En función de la señal de selección provista por el *decoder*, deja pasar uno entre varios buses. Este bloque se entrega ya listo para ser integrado.
- **switches_leds** : Periférico que facilita el acceso del programa a los *slide switches*, los pulsadores y los LEDs. Los dos primeros se corresponden con dos puertos de entrada en diferentes direcciones y el último con un puerto de entrada/salida en otra dirección (el programa puede tanto escribir los LEDs como leer qué ha escrito previamente). Nótese que en este diseño el pulsador 3 se usará como reset general, por lo que se conectará un cero a la correspondiente entrada de este bloque en lugar del pulsador. Este bloque se entrega ya listo para ser integrado. Con el software

entregado no afecta a la prueba del controlador VGA pero es empleado para volcar el contenido de los conmutadores sobre los LEDs.

	Bit							
Dirección relativa	7	6	5	4	3	2	1	0
0	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
1	0	0	0	0	0	BTN2	BTN1	BTN0
2	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0

- **gen_interrupt** : Periférico que genera una interrupción cada cierto número de ciclos de reloj. Este número de ciclos es un valor de 32 bits que puede ser configurado por el microprocesador a través de cuatro puertos de 8 bits. El periférico también permite leer el valor configurado. La señal de interrupción se borra automáticamente al activar el *PicoBlaze* su señal de *interrupt acknowledgement*. Consúltese la documentación de *PicoBlaze* para adquirir una mejor comprensión del mecanismo de interrupciones. Este bloque se entrega ya listo para ser integrado. En el diseño propuesto no afecta a la prueba del controlador VGA pero es empleado por el software para hacer parpadear los LEDs.
- **vga_ctrl** : El controlador VGA desarrollado en esta práctica.
- **reset_adapt** : Bloque de adaptación de la señal externa de reset. Este bloque se encarga de sincronizar adecuadamente esta señal con el reloj del sistema. Se entrega ya listo para ser integrado.

Programa:

El programa que se entrega (tanto en código fuente como en forma de fichero VHDL de la ROM), aparte de realizar un volcado del valor de los conmutadores sobre los LEDs, usa el controlador VGA para dibujar en pantalla unos patrones básicos, estáticos, que nos ayuden a verificar si el controlador funciona correctamente en placa y en simulación. En la siguiente figura se esquematiza la pantalla generada:

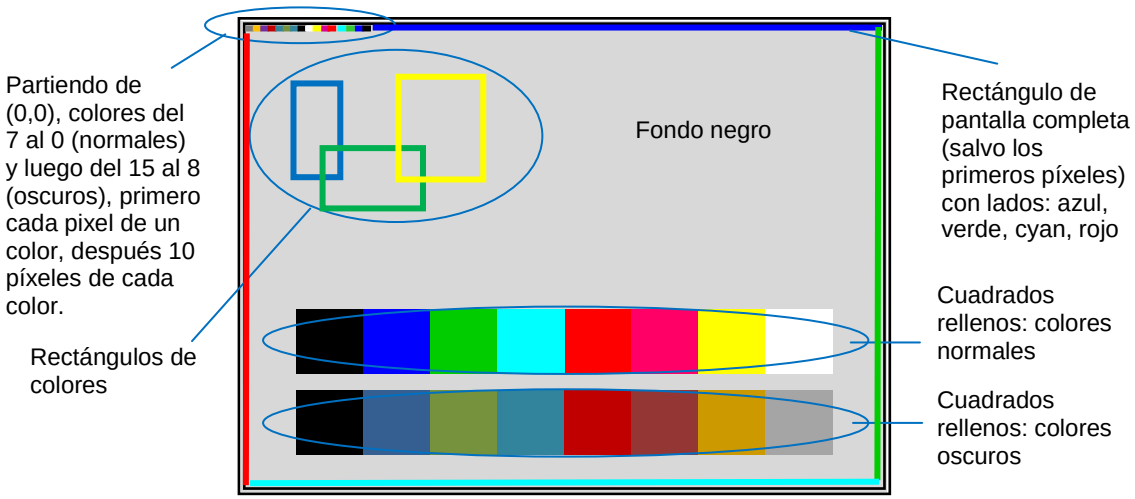


Figura 12 : Imagen mostrada en pantalla con el programa de prueba

Importante: para ver la imagen entera en pantalla puede ser necesario ajustar en el monitor la posición horizontal y vertical de la imagen.

Guía para la integración en el diseño completo y su simulación

Directorios y ficheros

Deberá respetarse la estructura de directorios presente en el archivo que se entrega:

P7_vga/

- *rtl/* - Código fuente VHDL para el diseño, con excepción del fichero VHDL describiendo la ROM de programa. Este directorio se entrega a los alumnos con varios ficheros ya listos para su uso.

- *sim/* - Directorio de simulación, contendrá el/los ficheros de test-bench (top-level y modelo de simulación de la memoria) y ficheros relacionados con *ModelSim* (proyecto, scripts…).

- *soft/* - Directorio de software de *PicoBlaze*, contendrá el código fuente del programa utilizado (*.psm*) y el correspondiente fichero VHDL (*rom_prog.vhd*). También contendrá la plantilla para la generación del VHDL (*ROM_form.vhd*). En principio la única acción necesaria relacionada con este directorio será incorporar al proyecto el fichero *rom_prog.vhd*. Sin embargo se entrega el código fuente del programa por si se desea hacer algún cambio. En ese caso, si se usa el entorno *pBlazeIDE* se deberá copiar el fichero ejecutable *pBlazIDE.exe* en este directorio.

- *ise/* - Directorio para el proyecto ISE. Se entrega a los alumnos con un fichero UCF completo.

Entorno de trabajo:

La primera tarea será completar el diseño y simularlo comprobando su corrección. Para ello, utilícese exclusivamente la herramienta *ModelSim*, y opcionalmente un editor de texto diferente si así se desea. Es decir, hasta que no se compruebe al menos mínimamente el funcionamiento por simulación, no debería abrirse el entorno *ISE* (salvo si se desea utilizar su editor de texto). En ModelSim puede crearse si se desea un nuevo proyecto, añadiendo (sin copiar) los ficheros fuente VHDL. Sin embargo, utilizar un "proyecto ModelSim" no es estrictamente necesario cuando se trabaja con scripts como vamos a hacer en esta práctica. En cualquier caso, para todo lo relativo a la simulación se deberá trabajar siempre bajo el directorio *sim/*.

[Nota: si se desea trabajar en casa consúltense las notas al respecto en la página web del laboratorio (en esta práctica el simulador necesita utilizar las librerías de primitivas de Xilinx, que deben estar correctamente instaladas y referenciadas)].

Completar el diseño de la FPGA:

Deberá crearse el "top-level" del diseño, la entidad *fpga_micro_vga*, en el fichero *fpga_micro_vga.vhd*. Aquí se instanciarán todos los componentes mostrados en el diagrama de bloques de la Figura 11, incluido el controlador VGA desarrollado en esta práctica, y se harán también las conexiones que adaptan el interfaz de memoria de este controlador a los pines de conexión con los chips de memoria, tal como se ha explicado más arriba. El buffer

tri-estado necesario para los datos de la memoria se hará mediante una asignación concurrente (no mediante un componente).

Testbench:

Los alumnos deberán desarrollar un mínimo testbench, que simplemente arranque el sistema (generación de reloj y pulso de reset inicial) y corra la simulación por un cierto tiempo. No se exigirá que el testbench compruebe nada automáticamente (las comprobaciones se harán sobre las ondas; en el mundo profesional un testbench nunca debe limitarse a esto pero aquí nos conformaremos con la fase inicial de depuración "visual").

En la siguiente figura se muestra la estructura del testbench, que deberá llamarse *fpga_micro_vga_tb*:

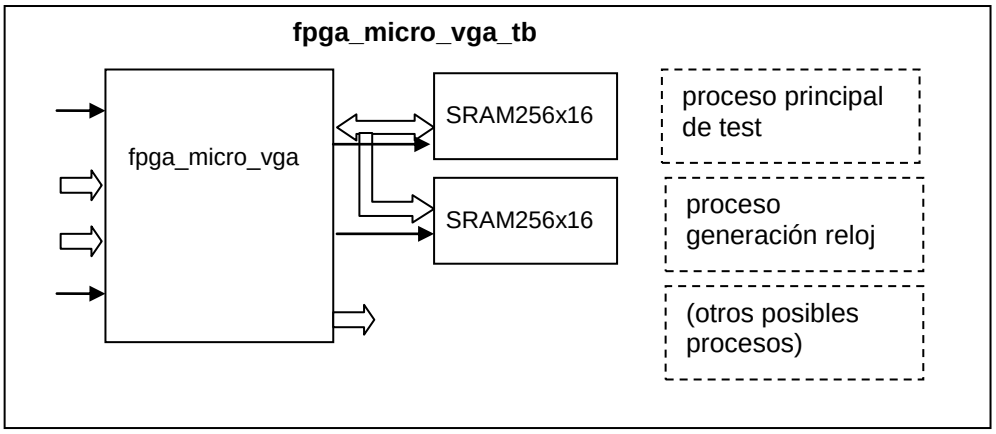


Figura 13 : Diagrama simplificado de la estructura del testbench

El testbench instanciará el top-level del diseño FPGA (*fpga_micro_vga*) y dos instancias de una memoria de 256k x 16 bits, modelando los dos chips de memoria existentes en la placa. Se entrega a los alumnos, en el directorio *sim/*, un modelo de memoria simplificado, *SRAM256kx16.vhd*. Si se echa un vistazo a este fichero se podrá observar que no se trata de una descripción sintetizable, sino de código específicamente creado para su utilización en testbenches, y optimizado para ocupar la menor memoria posible en el ordenador que está ejecutando la simulación.

En el mismo fichero del testbench se implementarán los procesos necesarios para llevar a cabo la simulación: como mínimo un proceso para generar un reloj continuo y otro para adjudicar valores a diversas entradas, generar el pulso de reset y esperar el tiempo de simulación deseado.

Simulación:

Para realizar las sucesivas iteraciones de simulación con *ModelSim*, se hará uso de un fichero *.do*, que permite ejecutar en forma de "script" diversos comandos: de compilación, simulación, selección de ondas a mostrar, etc. Se entrega un fichero *runsim.do* incompleto, que deberá completarse al menos con los comandos necesarios para compilar los ficheros creados por los alumnos. Como parte de este fichero de comandos para el simulador, se llama a otro script, *wave.do*, con la configuración de visualización de ondas. Este fichero puede generarse desde el propio visor de ondas (tras añadir las que queramos); para ello usar el comando de guardado del menú *File* mientras se tiene seleccionada la ventana de ondas. Los ficheros *.do* se ejecutan desde la consola de *ModelSim* con el comando *do*, esto es, por

ejemplo: "*do runsim.do*". Si se desea reducir la ocupación de memoria de la simulación puede sustituirse en el script la sentencia "log –r /*" por una orden de almacenamiento de ondas más limitada (se pueden superponer con "*add log [-r] <path jerárquico>*"). Por otra parte, para acelerar las primeras simulaciones, especialmente si se usa la versión de *ModelSim* para estudiantes, puede ser interesante crear un testbench para tan sólo el bloque controlador VGA, o bien modificar temporalmente el diseño no instanciando el micro y su memoria de programa (o comentando estas instanciaciones).

Para aquellos que deseen hacer cambios en el software del microprocesador (no es necesario para la realización básica de esta práctica), téngase en cuenta lo siguiente:

- Para ensamblar el programa y convertirlo a un VHDL de la ROM, cópiese el ejecutable *pBlazIDE.exe* en el directorio *soft/* y ejecútese desde este directorio. (El entorno *pBlaze IDE* permite además simular el comportamiento del micro, haciendo uso de directivas especiales en el ensamblador para crear controles en el interfaz gráfico dedicados a los diferentes puertos de entrada / salida). Importante: configurar este programa seleccionando en el menú *"Settings*" la opción "*PicoBlaze 3*".
- Es posible mostrar en la ventana de ondas del simulador los registros del microprocesador, la instrucción en curso, etc. Para ello, en la ventana *Workspace*, solapa *sim*, buscar bajo la instancia del procesador el proceso *simulation* (hacia el final de los objetos que cuelgan del procesador, con un icono en forma de círculo) y seleccionarlo. A continuación, usar el menú *View | Locals*. Aparecerá una nueva ventana donde podremos seleccionar estas informaciones y mandarlas al visor de ondas.

Implementación:

Una vez satisfecha la simulación funcional, y en particular, **comprobada la temporización de las señales de vídeo**, se abrirá el entorno ISE, creando un proyecto en el directorio *ise/*:

- Usar *New Project*, nombrar el proyecto "fpga_micro_vga" y tras usar el botón "…" de *Location* borrar *fpga_micro_vga* del final del path para que éste termine en ...*ise*).
- Comprobar que se selecciona como "top-level source type" *HDL*, que la FPGA seleccionada es la adecuada (Spartan 3 XC3S200 FT256 -4) y que las herramientas seleccionadas son las correctas.
- Añadir al proyecto todos los ficheros *.vhd* del directorio *rtl/*, el fichero *rom_prog.vhd* del directorio *soft/*, y el fichero *fpga_micro_vga.ucf* del directorio *ise/* (utilizar siempre *Add Source*, nunca *Add Copy of Source*).

Antes de conectar la placa al ordenador deberá comprobarse que el reporte de *pads* se corresponde con las constraints de localización especificadas en el fichero UCF (al menos mirar algunos pines para verificar que efectivamente la herramienta ha considerado dicho fichero de constraints).

Proceder a la síntesis e implementación del circuito tal como se ha aprendido en prácticas anteriores.

De nuevo se recuerda que **antes de descargar el diseño** a la placa hemos de estar seguros de tener una **correcta temporización en las señales de vídeo**. Una vez descargado el diseño, conectar la placa a un monitor para comprobar el patrón de video y comprobar también el volcado de los conmutadores a los LEDs.

Análisis de resultados de la implementación

Una vez completado el proceso de implementación, comprobar el resultado en *timing* y área y **responder a las siguientes preguntas** (anotar en papel o en un fichero las respuestas).

- ⇨ ¿Cuál es el área del circuito, es decir, la ocupación de la FPGA? Elegir los parámetros que se consideren más apropiados para responder esta pregunta.
- ⇨ ¿De dónde a dónde va el peor *path*?
- ⇨ ¿Cuánto tarda?
- ⇨ ¿Cuál es el periodo mínimo de reloj? ¿Y la frecuencia máxima?
- ⇨ ¿Cumple el circuito las restricciones de tiempo especificadas?

Mejoras opcionales

Como mejoras de la práctica se podrá añadir funcionalidad al hardware y/o al software, con temática libre.

Como una posible mejora de hardware y software se propone añadir una funcionalidad de "sprite" al controlador VGA, usando una memoria dentro de la FPGA para albergar un mapa de 16 x 16 píxeles donde el micro pueda dibujar un sprite a través de un conjunto de nuevas direcciones de I/O en *vga_ctrl*. Otro conjunto de puertos de I/O permitiría al software definir la posición en pantalla del sprite. El bloque controlador deberá saber si la posición de un píxel está en la zona cubierta por el sprite para utilizar los datos de la memoria interna (sprite) en vez de los de la memoria externa (buffer de vídeo). El programa de test se modificaría para utilizar el sprite, haciendo que se mueva por la pantalla.

Como una posible mejora exclusivamente software, más simple, se propone modificar el programa para hacer que un rectángulo, de color y tamaño a elegir, se mueva continuamente por la pantalla con una trayectoria oblicua, "rebotando" en los bordes de la pantalla.

Entrega

El **correcto funcionamiento sobre la placa S3BOARD deberá mostrarse al profesor** en clase como muy tarde el día publicado en la web del laboratorio.

Por otra parte, **deberá entregarse por web un archivo zip con el diseño realizado**, **incluido el software**, antes de la fecha límite especificada en la web del laboratorio. Entregar el directorio "P7_vga" completo, eliminando tan sólo:

- Los ficheros de ondas de simulación, que son los que suelen ocupar más espacio ("wlf*", "*.wlf"…), y que estarán localizados en el subdirectorio "sim/" si se ha seguido el flujo de trabajo con *ModelSim* indicado en este enunciado.
- El ejecutable *pBlazIDE.exe*.

Importante: Todos los ficheros fuente que se entreguen (.vhd, .ucf, etc.), que hayan sido creados o modificados por los alumnos, deberán llevar una **cabecera** adecuada, **incluyendo siempre el nombre de los autores**. Además todo el código fuente deberá llevar comentarios apropiados, tanto en contenido como en cantidad.

El archivo zip deberá tener la siguiente nomenclatura para los alumnos de DIE:

```
<{DIE_L | DIE_X | DIE_J}><numero_pareja_2digitos>_P7.zip
```

y la siguiente para los alumnos de DCSE:

```
<{DCSE_L | DCSE_XA | DCSE_XB}><numero_pareja_2digitos>_P7.zip
```

(Ejs.: `DCSE_XB01_P7.zip` para pareja 1 de DCSE-b miércoles, `DIE_J02_P7.zip` para pareja 2 de DIE jueves).

No será necesario presentar ninguna memoria.

Se recuerda que los alumnos deberán comprender el diseño completo y familiarizarse con él.