Implementación de una controladora VGA en un CPLD XC9500

Memòria del projecte
d'Enginyeria Tècnica en
Informàtica de Sistemes
presentada per
Mario Macías Lloret
i dirigida per
D. Isabel Rexachs

Escola Universitària d'Informàtica
Sabadell, septembre de 2002

Agradecimientos

En primer lugar, me gustaría mostrar mi agradecimiento a la tutora de este proyecto, Dolores Isabel Rexachs del Rosario, por guiarme con paciencia durante la realización del mismo.

También me gustaría agradecer al Dr. Domingo Benítez, de la Universidad de las Palmas de Gran Canaria, que nos prestara para este proyecto una placa de prototipado, sin la cual no habríamos podido realizarlo.

Resumen

El objetivo principal de este proyecto es el estudio, diseño e implementación de aplicaciones para las placas de prototipado de XESS, orientándose principalmente hacia la visualización de imágenes, en pantallas compatibles VGA, así como proporcionar la documentación necesaria para tal fin.

Para ello, primero se describen al lector los fundamentos teóricos relacionados con el proyecto, haciendo una pequeña introducción y proporcionando la bibliografía para obtener una información amplia.

Se han elaborado dos tutoriales: uno para aprender a utilizar la placa XS95 y otro en el que se explican los métodos y herramientas necesarios para crear aplicaciones, desarrollando una aplicación sencilla que ejemplifique los conceptos que se introducen. Se busca que inmediatamente después de leerlos, el lector pueda utilizar la placa sin problemas.

Una vez introducido el funcionamiento de la placa y los conocimientos necesarios para hacer aplicaciones, se describe una sencilla controladora VGA y se crea una aplicación que la utiliza, a modo de ejemplo.

Índice

1. Introducción

Desde los últimos años cada vez es más frecuente ver cómo tecnologías anteriormente reservadas a computadoras hoy son incorporadas en otras máquinas más tradicionales, como electrodomésticos, coches, máquinas expendedoras, etc. Esto se debe básicamente a la incorporación de microcontroladores y procesadores digitales del señal (DSP) en todo tipo de aparatos.

Estos avances han llevado, entre otras cosas, a la incorporación de dispositivos de visualización gráfica en todo tipo de aparatos: cada vez es menos extraño ver electrodomésticos con una interfície gráfica de usuario o, como caso más representativo, teléfonos móviles con pantallas gráficas que acompañan la información suministrada al usuario con dibujos e iconos.

Esta evolución viene facilitada por varios factores: la disminución en los costes de la tecnología, la mayor capacidad de integración y, en el caso que más directamente nos afecta, la aparición de nuevas herramientas de diseño y desarrollo. Por un lado están los microcontroladores y sus utilidades de simulación y creación, que permiten controlar dispositivos mediante software; por otro lado están los dispositivos hardware programables, tales como CPLDs o FPGAs, que permiten la creación de circuitos integrados a pequeña escala con unos costes muy bajos de producción. En un mismo diseño, a menudo se combinan microcontroladores y dispositivos programables. Los primeros se utilizan para crear aplicaciones de propósito general mediante software; los segundos, para diseñar controladores específicos vía hardware.

Entre estas herramientas de diseño y creación se encuentran las placas de prototipado de la marca XESS (www.xess.com). Que incluyen elementos tales como microcontroladores, puertos de comunicación, memorias, FPGAs, etc... con la finalidad de facilitar la creación de diseños mixtos hardware/software.

Dicho esto, pasamos a describir los objetivos del proyecto.

1.1 Objetivos

Los objetivos de este proyecto se pueden dividir en dos partes:

1. El estudio de las placas de prototipado XESS, y de la metodología y herramientas software disponibles para la creación de aplicaciones software/hardware, con el fin de proporcionar la documentación necesaria para iniciarse de manera inmediata en dicha tarea. Para tal fin, en el capítulo 3 se explica cómo utilizar la placa y en el capítulo 4 cómo crear aplicaciones para ésta, mediante un ejemplo sencillo pero suficiente para aprender a manejar todas las herramientas necesarias en el proceso de diseño e implementación de aplicaciones.

2. El estudio y aplicación de los elementos necesarios para controlar y visualizar imágenes en un monitor VGA. Los capítulos 5 y 6 corresponden al diseño y desarrollo de un dispositivo VGA controlado íntegramente vía hardware y en el capítulo 7 se han establecido guías para la interconexión del dispositivo con un microcontrolador y la memoria de vídeo.

La herramienta disponible para realizar este proyecto es una placa de prototipado XESS XS95-108, en principio una placa de gama baja, pero que bastará para la iniciación y aprendizaje en el campo propuesto.

2. Fundamentos teóricos

En este capítulo se proporcionan los diferentes conceptos teóricos utilizados en el proyecto que nos ayudarán a comprender el funcionamiento de los componentes hardware básicos de la placa (CPLD y microcontrolador), así como los fundamentos teóricos utilizados en la realización el proyecto: una breve introducción al lenguaje VHDL y una explicación del modo de funcionamiento de la VGA.

2.1 Microcontroladores

Un microcontrolador *[Gonzalez 92]* es similar a un procesador de propósito general, sólo que está diseñado para ser incluido en un sistema empotrado, es decir, están diseñados para ser incluidos en sistemas que utilizan una combinación de hardware y software para desarrollar una función dedicada. Para tal finalidad, los microcontroladores suelen incluir una CPU, una cantidad no muy grande de memoria RAM o ROM, y otros dispositivos como conversores analógico/digital o digital/analógico, temporizadores, puertos de comunicación, etc...

Las principales ventajas de los microcontroladores son sus prestaciones, la flexibilidad, la programabilidad y el reducido tamaño.

2.1.1 Introducción al microcontrolador 8031

Es un microcontrolador compatible con la familia de Intel 8051 *[Yeralan Ahluwalia 95]*. En concreto, la placa de prototipado a utilizar (XS95-108) está equipada con un microcontrolador Winbond W78C32C *[Winbond 99]*, cuyas principales características son las siguientes:

- Controlador CMOS de 8 bits
- Velocidad máxima de 40 MHz
- 256 bytes de RAM interna *scratchpad*.
- Espacio de direccionamiento de 64 KB para memoria de código.
- Espacio de direccionamiento de 64 KB para memoria de datos.
- 4 puertos bidireccionales de 8 bits.

- Tres temporizadores/contadores de 16 bits
- Un puerto serie para comunicación full duplex.
- 6 fuentes de interrupción, con dos niveles.

En la RAM interna, el 8031 tiene varios registros especiales de función (SFR), que son los registros de control y de datos. También pertenecen a los SFR el acumulador, el registro B y la palabra de estado (PSW), que contiene los *flags* de la CPU.

El 8031 tiene arquitectura *Harvard*, es decir, la memoria de datos está separada de la memoria de programa. Así pues, hay dos señales diferentes de lectura en memoria externa: RD# (P3.7) y PSEN#. La primera se activa cuando se va a leer un byte de la memoria de datos externa; la segunda, cuando se va a leer un byte de la memoria de programa externa. Cuando se programa, los bytes de la memoria externa de programa han de ser leídos mediante un conjunto especial de instrucciones de lectura, como MOVC. También hay otro conjunto especial de instrucciones para leer de la memoria de datos externa, como MOVX.

2.1.2 Repertorio de instrucciones del 8031

El conjunto de instrucciones de la familia 8051 lo componen 111 instrucciones, las cuales se describen a continuación según el tipo al que pertenecen. Para más información consultar la referencia bibliográfica *[Yeralan Ahluwalia 95].*

- **Transferencia de datos**
  - Generales

    MOV: realizan una transferencia de un bit o byte desde el operando origen al destino.

    PUSH: incrementa el registro SP y realiza una transferencia de un byte desde el operando origen a la posición de pila direccionada por SP.

    POP: realiza la transferencia de un byte desde la posición de pila direccionada por SP al operando destino y después decrementa el SP.
  - Específicas al acumulador

    XCH: intercambia el contenido del operando fuente con el acumulador.

    XCHD: intercambia el nibble bajo del operando fuente con el nibble bajo

del ACC.

MOVX: realiza la transferencia de un byte entre la memoria de datos externa y el acumulador.

MOVC: realiza la transferencia de un byte desde la memoria de programa al acumulador

- **Aritméticas**
  - Suma

    INC: incrementa el uno el operando fuente y pone el resultado en el operando.

    ADD: suma el contenido del acumulador con el operando fuente y el resultado lo guarda en el acumulador.

    ADDC: suma el acumulador con el operando fuente y guarda el resultado en el acumulador.

    DA: ajuste decimal para las operaciones en BCD.
  - Resta

    DEC: decrementa en 1 el operando fuente y carga el resultado en el operando.

    SUBB: resta el segundo operando fuente del primer operando (el acumulador), resta 1 si el f*lag* de acarreo está a 1 y devuelve el resultado en el acumulador.
  - Multiplicación

    MUL: realiza una multiplicación sin signo del contenido del registro A con el contenido del registro B, devolviendo un resultado de doble byte. En el acumulador se pone la parta baja del resultado y en B la parte alta.
  - División

    DIV: realiza una división sin signo del contenido del registro A con el contenido del registro B, devolviendo en el acumulador el cociente entero y el resto en el registro B.
- **Instrucciones lógicas**
  - Un operando

    CLR: pone a 0 el acumulador o cualquier bit direccionable.

    SETB: pone a 1 cualquier bit direccionable

CPL: complementa el contenido del acumulador o de cualquier bit direccionable.

Instrucciones de rotación: RL, RLC, RR, RRC y SWAP.

- Dos operandos

  ANL, ORL, XRL: realizan la operación lógica AND de los dos operandos fuente y devuelve el resultado en la posición del primer operando.

- **Transferencia de control**
  - Llamadas incondicionales, retornos y saltos: ACALL, LCALL, AJMP, LJMP, SJMP.
  - Saltos condicionales: JZ, JNZ, JC, JB, JNB, JBC, CJNE, DJNZ.
  - Retorno de interrupción: RETI

2.1.3 Estructura del microcontrolador 8031

En la figura 2.1 se muestra la estructura básica del 8031 incorporado en la placa XESS.

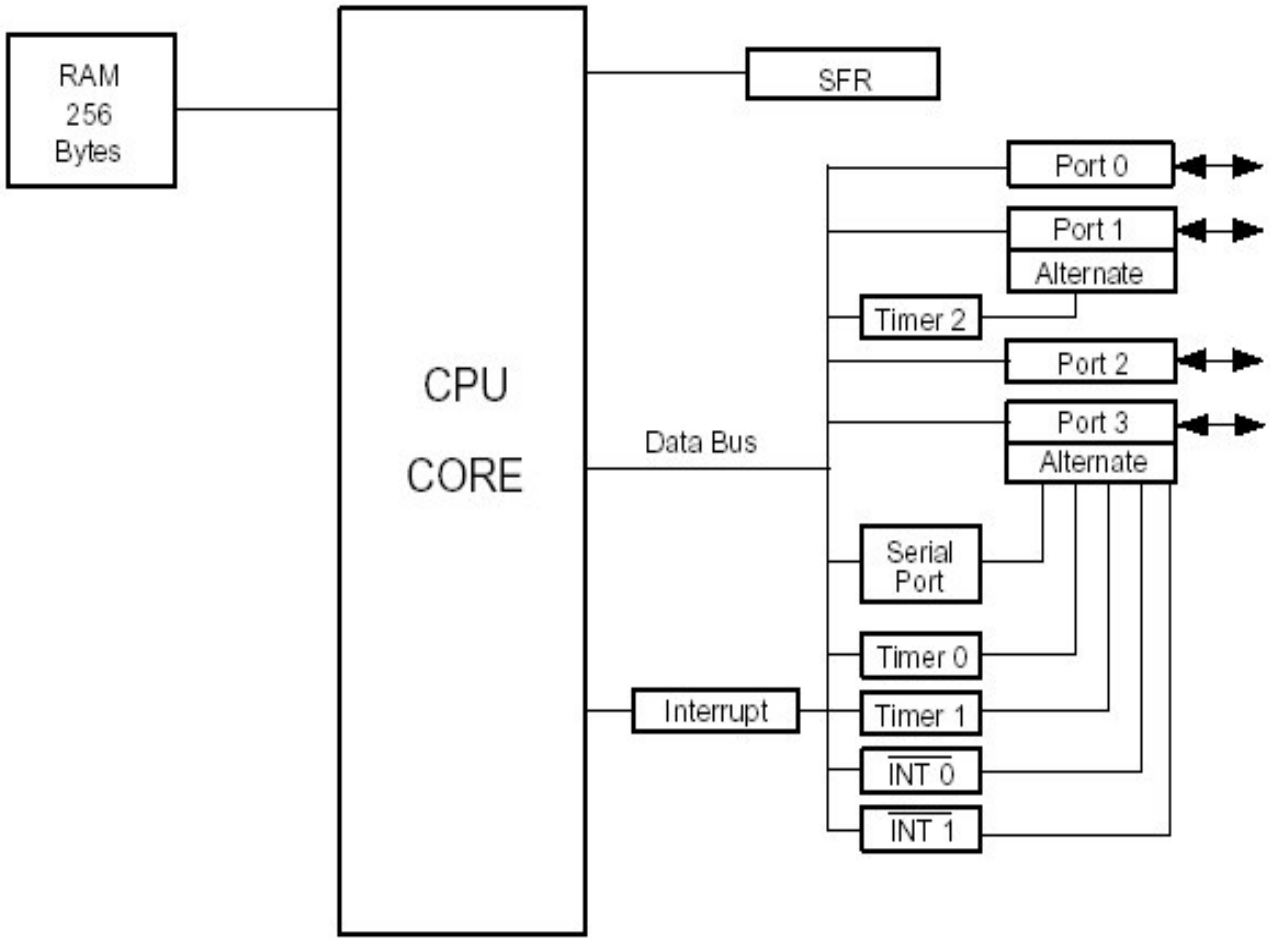


Figura 2.1. Estructura básica del 8031 (Fuente: Winbond)

Aunque el microcontrolador presente las características descritas anteriormente, a la hora de diseñar aplicaciones nos vemos limitados al conexionado de la placa. Mirar la figura 2.2 o leer el manual de la placa *[Xess 01]* para más información.

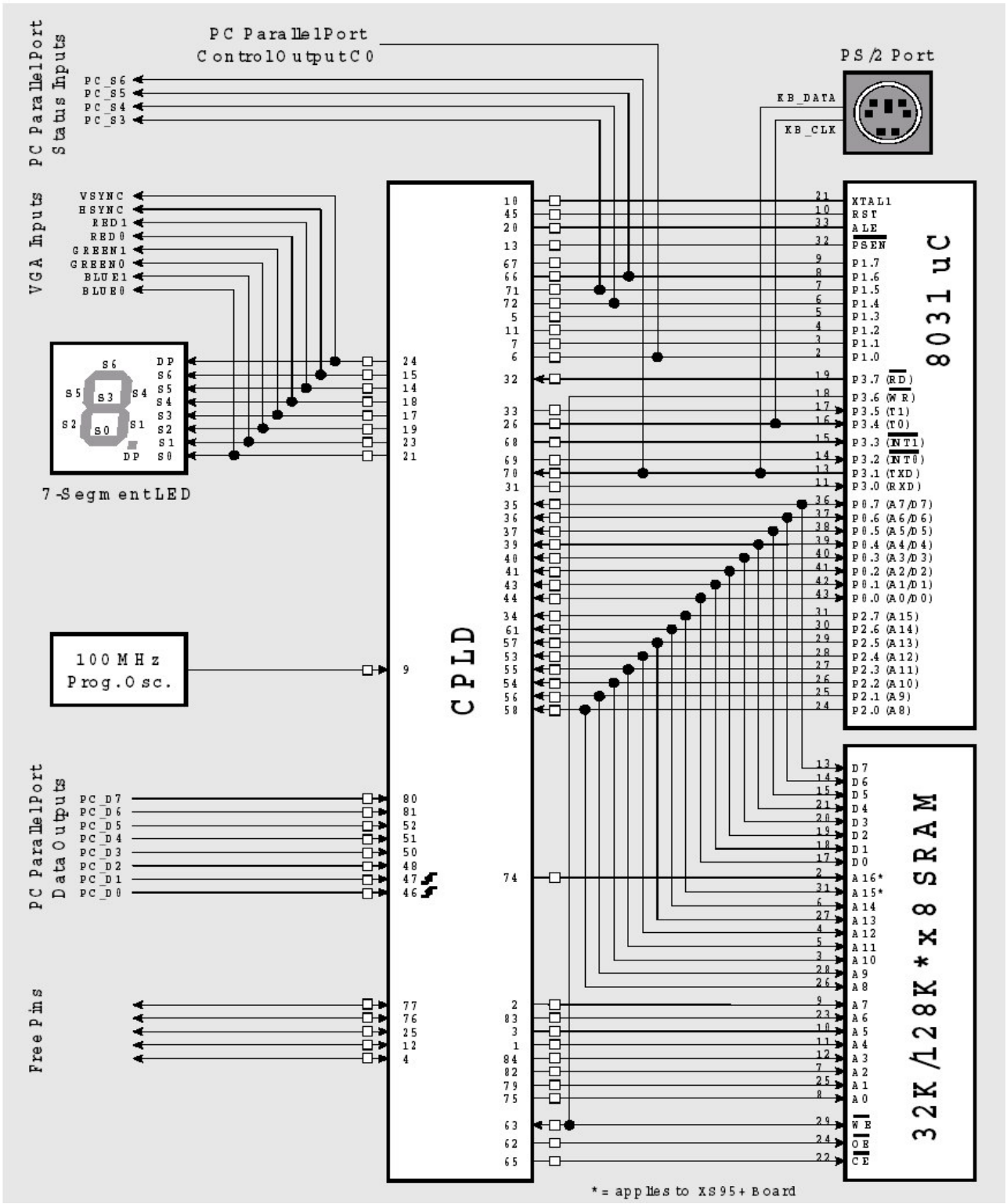


Figura 2.2 Esquema del conexionado de la placa XESS XS95

2.2 Dispositivos Lógicos Programables (PLD)

Son circuitos definibles por el usuario, ya sea mediante algún tipo de lenguaje de descripción (Abel, VHDL) o mediante esquemáticos. De entre todos los circuitos programables veremos los que nos interesan: CPLDs y FPGA, pero antes es necesario introducir otros dispositivos anteriores. Los siguientes apartados son un extracto del documento **Dispositivos Lógicos Programables** *[Redeya 01]*.

2.2.1 PAL

Las PAL son dispositivos de matriz programable. La arquitectura interna consiste en términos AND programables que alimentan términos OR fijos. Todas las entradas a la matriz pueden ser combinadas mediante AND entre si, pero los términos AND específicos se dedican a términos OR específicos.

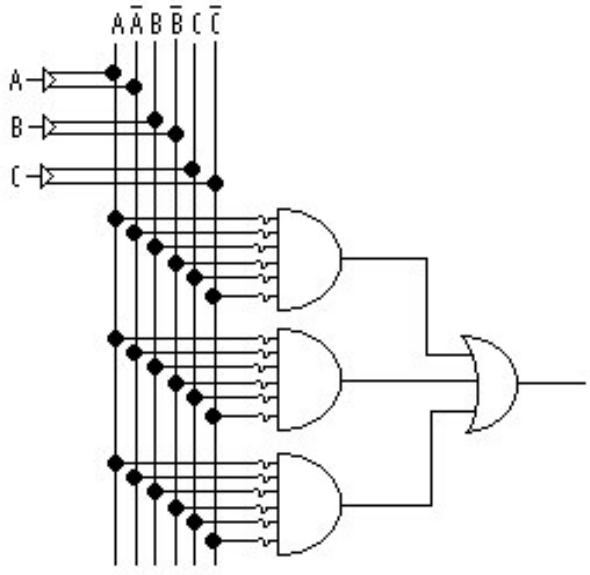


Figura 2.3 Estructura de una PAL

2.2.2 PLA

Las PLA son matrices lógicas programables. Estos dispositivos contienen ambos términos AND y OR programables lo que permite a cualquier término AND alimentar cualquier término OR. Normalmente poseen realimentación desde la matriz OR hacia la matriz AND que puede usarse para implementar máquinas de estado asíncronas.

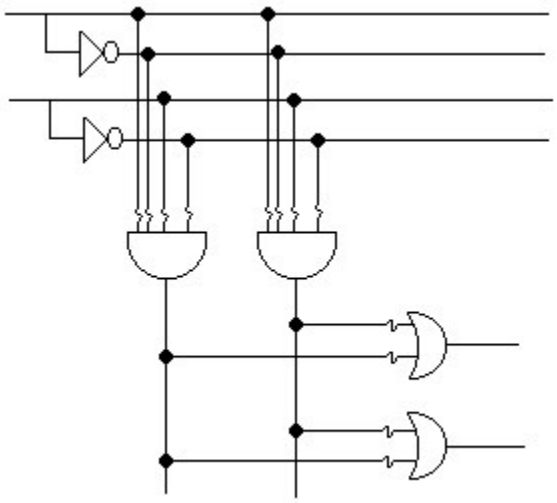


Figura 2.4 Estructura de una PLA

2.2.3 CPLD

La placa XS95 incorpora este tipo de dispositivo programable. Los PLDs complejos son lo que el nombre implica, Dispositivos Complejos de Lógica Programable. Se consideran PAL muy grandes que tienen algunas características de las PLA. La arquitectura básica es muy parecida a la PAL con la capacidad para aumentar la cantidad de términos AND para cualquier término OR fijo. Esto se puede realizar quitando términos AND adyacentes o empleando términos AND desde una matriz expandida. Esto permite que cualquier diseño pueda ser implementado dentro de estos dispositivos.

2.2.4 FPGA

Las FPGA son Campos de Matrices de Puertas programables. Simplemente son matrices de puertas eléctricamente programables que requieren múltiples niveles de lógica. Las FPGA se caracterizan por altas densidades de puerta, alto rendimiento, un número grande de entradas y salidas definibles por el usuario, un esquema de interconexión flexible y un entorno de diseño similar al de la matriz de puertas. No están limitadas a la típica matriz AND-OR.

Los diseñadores que usan FPGAs pueden definir funciones lógicas en un circuito y revisar estas funciones como sea necesario. Así, las FPGAs pueden diseñarse y verificarse en unos días, a diferencia de las varias semanas necesarias para las matrices de puerta programables.

2.2.5 El CPLD XC9500

La placa de prototipado utilizada en este proyecto incluye un CPLD XC95108. Un CPLD de alto rendimiento, el cual proporciona 2.400 puertas y 108 pins de entrada/salida utilizables. En la figura 2.5, se muestra un diagrama de bloques describiendo la arquitectura de dicho CPLD.

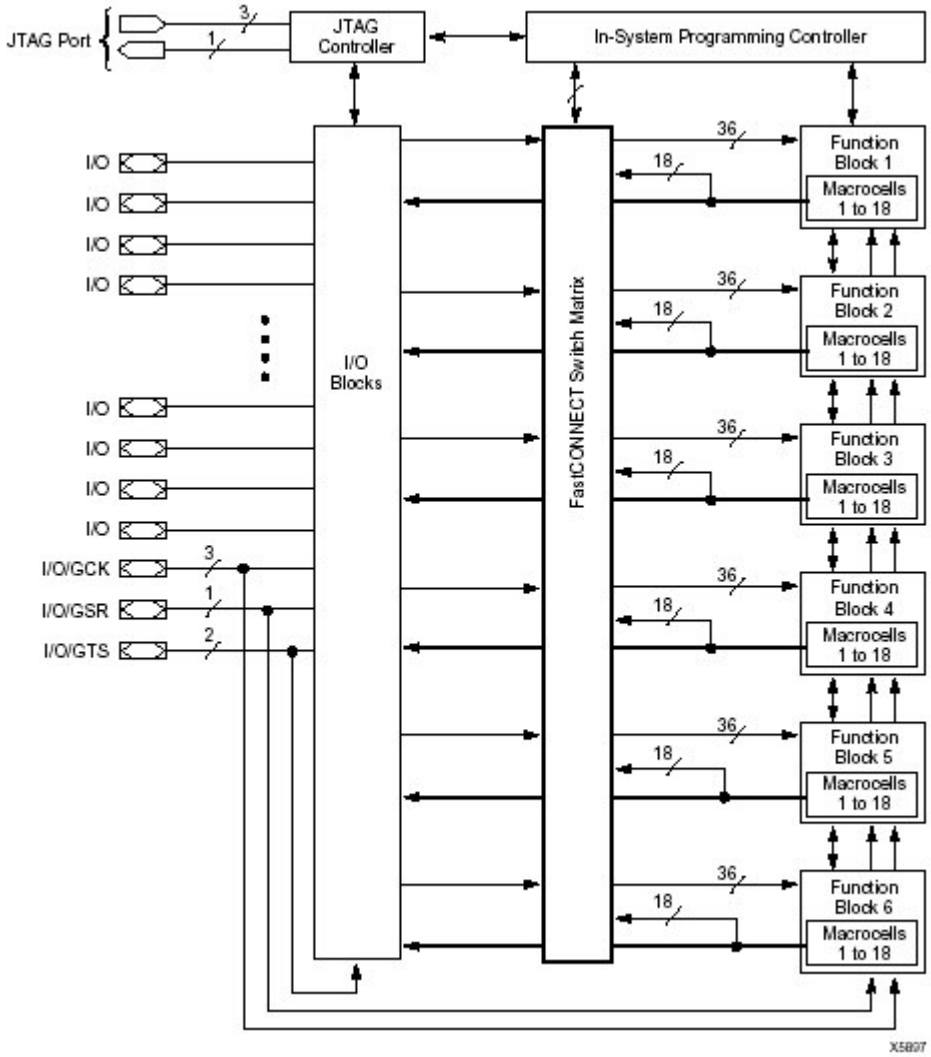


Figura 2.5 Arquitectura del CPLD XC9500

2.3 Los lenguajes de descripción de hardware

Los lenguajes de descripción de hardware (Abel, Verilog, VHDL...) son al diseño lógico lo que los lenguajes de programación de alto nivel (C, Pascal...) son a la programación. Nacen básicamente por el aumento en la complejidad de los circuitos integrados, que lleva al aumento en la complejidad del proceso de depuración y

mantenimiento de éstos. La principal finalidad de estos lenguajes fue eliminar el desfase existente entre tecnología y diseño.

2.3.1 El lenguaje VHDL

Creado en 1983 por IBM y Texas Instruments. Fue revisandose con la colaboración de empresas y universidad. La independencia en la metodología de diseño, su capacidad descriptiva en múltiples dominios y niveles de abstracción, su versatilidad para la descripción de sistemas complejos, su posibilidad de reutilización y en definitiva la independencia de que goza con respecto de los fabricantes, han hecho que VHDL se convierta con el paso del tiempo en el *lenguaje de descripción de hardware* por excelencia *[UPV 02]*.

No es el objetivo de este proyecto la enseñanza de VHDL, pero en las reseñas bibliográficas *[Villar 98], [UPV 02]* y *[Bolton 01]* se encuentran tutoriales electrónicos con los que aprender a programar este lenguaje de descripción.

2.4 VGA

IBM desarrolló VGA en 1987 como uno de los primeros estándares de vídeo en usar señales analógicas. A continuación se describe su funcionamiento.

2.4.1 Señales de color VGA

Hay tres señales de color (rojo, azul y verde), que envían información a un monitor VGA. Sus valores analógicos van desde 0V (totalmente oscuro) hasta 0.7 V (brillo máximo), e indican al monitor el color del punto en la pantalla.

En el caso concreto de la placa XS95, cada valor RGB puede tener 4 niveles de intensidad, ya que cada color primario puede ser representado digitalmente por dos bits. Esto quiere decir que se pueden representar un máximo de 4x4x4=64 colores diferentes.

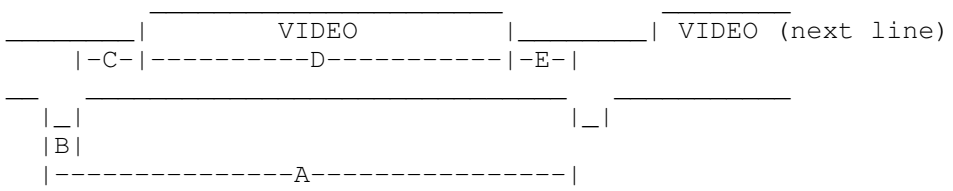
2.4.2 Tiempos de los señales VGA

Cuando se visualiza una imagen en un monitor VGA, se va dibujando línea a línea de arriba a abajo, y cada línea se visualiza punto a punto de izquierda a derecha. Para saber cuándo empiezan y acaban las líneas e imágenes completas son necesarias dos señales de sincronismo, una para el sincronismo vertical y otra para el horizontal. Un pulso negativo en cada una de estas señales indica el fín de una línea (en el caso del sincronismo horizontal) o el fín de una imagen (en el caso del sincronismo vertical).

La duración y la frecuencia de los pulsos no son arbitrarias; dependiendo de la resolución horizontal y vertical que se desee obtener habrá que establecer determinados valores, tanto para duración como para frecuencia. A continuación se muestran los tiempos necesarios para obtener algunas resoluciones *[Engdahl 00]*.

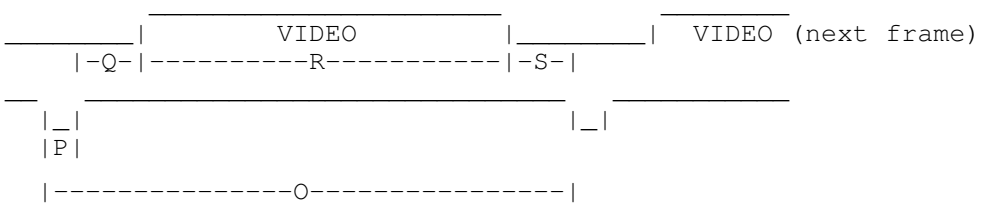
Temporización horizontal

Puntos horizontales	640	640	640	
Líneas de barrido verticales	350	400	480	
Polaridad del sincronismo horizontal	POS	NEG	NEG	
A (us)	31.77	31.77	31.77	Tiempo de línea de barrido
B (us)	3.77	3.77	3.77	Duración pulso de sincronismo
C (us)	1.89	1.89	1.89	Pórtico trasero
D (us)	25.17	25.17	25.17	Tiempo de vídeo activo
E (us)	0.94	0.94	0.94	Pórtico frontal



Temporización vertical

Puntos horizontales	640	640	640	
Líneas de barrido verticales	350	400	480	
Polaridad del sincronismo horizontal	POS	NEG	NEG	
Frecuencia vertical	70Hz	70Hz	60Hz	
O (ms)	14.27	14.27	16.68	Tiempo total de frame
P (ms)	0.06	0.06	0.06	Longitud del sincronismo
Q (ms)	1.88	1.08	1.02	Pórtico trasero
R (ms)	11.13	12.72	15.25	Tiempo de vídeo activo
S (ms)	1.2	0.41	0.35	Pórtico frontal



Para una información más detallada sobre el funcionamiento de los monitores VGA remitirse a la bibliografía *[Engdahl 00]*.

3. Manual de usuario de la placa XS95

Este capítulo es un manual de usuario en el que se explica cómo instalar y configurar la placa XS95, y cómo utilizar las XSTOOLS, los programas proporcionados por XESS para el testeo, configuración, carga de diseños y depuración.

Pese a que este proyecto ha sido realizado sobre una placa XS95 v1.2, es de vital importancia extender este manual a las características de la versión 1.3, la más reciente, con tal de evitar problemas en caso de adquirir en un futuro placas de este tipo. Si la placa a utilizar no pertenece a ninguna de estas versiones, será necesario consultar también los manuales de dicha versión para evitar problemas.

Parte del contenido de este capítulo está extraído y traducido del manual de la placa Xess XS95 *[Xess 01]*.

3.1 Características técnicas de las placas XS95

XS95 v1.2

CPLD XC9518 con 108 macroceldas
Microcontrolador 8031
32KB SRAM
Oscilador de 12 MHz
Puerto paralelo
Puerto VGA
Led de 7 segmentos
Interfície de prototipado de 84 pines
Conector de alimentación de 9V DC

XS95 v1.3

CPLD XC9518 con 108 macroceldas
Microcontrolador 8031
128KB SRAM
Oscilador programable de 100MHz max
Puerto paralelo
Puerto VGA
Conector PS/2 para ratón o teclado
Led de 7 segmentos
Interfície de prototipado de 84 pines
Conector de alimentación de 9V DC

3.2 Instalación de la placa XS95

3.2.1 Componentes esenciales

Para poder empezar a usar la placa XS95 son necesarios los siguientes elementos:

1. Una placa de prototipado XS95.
2. Un cable paralelo con un conector DB-25 macho en cada extremo.
3. Una colección de software que incluye:
    - XSTOOLS: Utilidades para testar, configurar y cargar programas en la placa. Son descargables desde www.xess.com
    - MODELSIM XE: Compilador/Simulador de VHDL.
    - WEBPACK: Necesario para crear los datos a cargar en la placa. Estos dos últimos se pueden descargar en www.xilinx.com
    - JTAG: Necesario para transformar los archivos creados por Webpack en un archivo “svf”.

3.2.2 Instalando la placa

XS95 permite dos modos de operación:

- Libre: Se puede usar la placa XS95 para experimentar con diseños para el CPLD XC9500 y el microcontrolador 8031. Tan sólo hay que colocar la placa en una superficie no conductora y aplicar la alimentación al jack desde un transformador 9V DC con un conector hembra de 2.1 mm, con el polo positivo en el centro. El regulador interno de la placa se encargará de generar las tensiones requeridas en el resto de la placa.
- En una tabla de prototipado: la placa se puede instalar en una tabla de prototipado, utilizando sus patillas como entradas o salidas. Este modo de operación se sale de los objetivos del proyecto, por lo que no será explicado.

3.2.3 Conexión al PC

Conectar un extremo del cable al puerto de impresora del PC y el otro al conector J1 de la placa, el cual podemos ver en la figura 3.1.

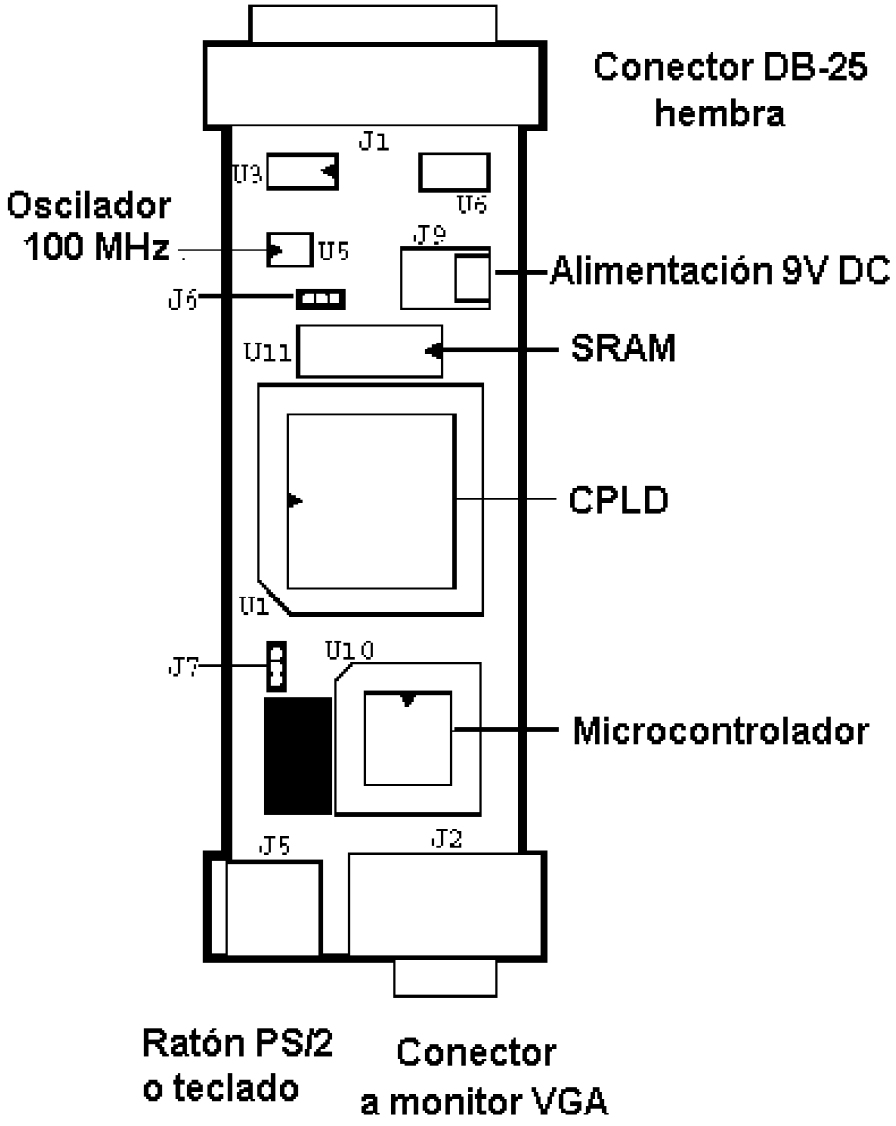


Figura 3.1 Estructura básica de la placa XS95

3.2.4 Configuración de los *jumpers*

La placa dispone de unos *jumpers* que permiten configurar por hardware el modo de funcionamiento del microcontrolador (normal o expandido) y la programación del reloj.

Jumper	Configuración	Propósito
J6 (ausente en la versión1.2 de la placa)	2-3 (osc) (por defecto)	El oscilador programable genera las señales de reloj.
	1-2 (set)	Se está fijando la frecuencia del oscilador programable
J7	1-2 (ext) (por defecto)	El programa del microcontrolador 8031 está almacenado en la RAM externa (U11), de 32KB.
	2-3 (int)	El programa está almacenado internamente en el microcontrolador 8031.

3.2.5 Testeo de la placa

Una vez instaladas las XSTOOLS, ejecutar GXSTEST y aparecerá la ventana mostrada en la figura 3.2.

Figura 3.2 Utilidad GXSTEST

Hay que seleccionar el modelo de la placa (en nuestro caso XS95-108 o XS95-108+) y el puerto al que está conectada, generalmente LPT1. Una vez pulsado el botón “TEST” se ejecutarán unos procedimientos de testeo en la placa. Al acabar estos procedimientos, un “0” en el 7 segmentos indicará que todo ha ido bien. Una “E” indicará que ha sucedido algún error y aparecerá en pantalla una ventana indicando los fallos sucedidos.

3.2.6 Oscilador de la placa

A la hora de diseñar e implementar los proyectos, es muy importante tener en cuenta que la versión 1.2 de la placa XS95 lleva incorporado un oscilador de 12 MHz y la

versión 1.3 incorpora un oscilador programable de 100 MHz, el cual se puede programar para obtener frecuencias que son divisiones de estos 100 MHz.

Hay que contar que algunos diseños para la versión 1.2 de la placa deben ser ligeramente modificados si se van a implementar en la versión 1.3, ya que la frecuencia más cercana a los 12 MHz que podamos conseguir con esta última serán los 12,5 Mhz.

Para programar el oscilador de la XS95 v1.3 está el programa GXSSETCLK de las XSTOOLS (figura 3.3). El modo de funcionamiento es sencillo:

- Seleccionar el modelo de placa y el puerto de conexión al PC.
- En el campo *Divisor*, escribir un número entre 1 y 2052 por el cual dividir 100 y obtener la frecuencia.
- Si se marca la casilla *External Clock*, indicamos a la placa que nosotros le suministraremos un reloj externo a partir de la patilla 64 de la placa.

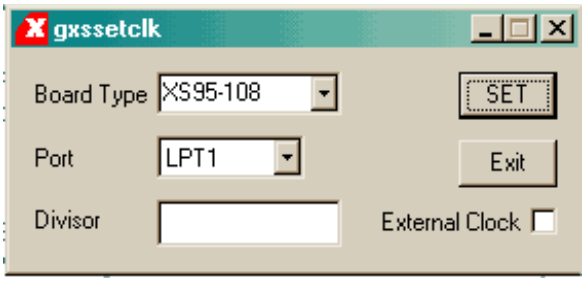


Figura 3.3 Utilidad GXSSETCLK

3.3 Carga de diseños y programas

3.3.1 Carga de diseños

Para tal tarea existe el programa GXSLOAD, la interfície del cual se puede ver en la figura 3.4.

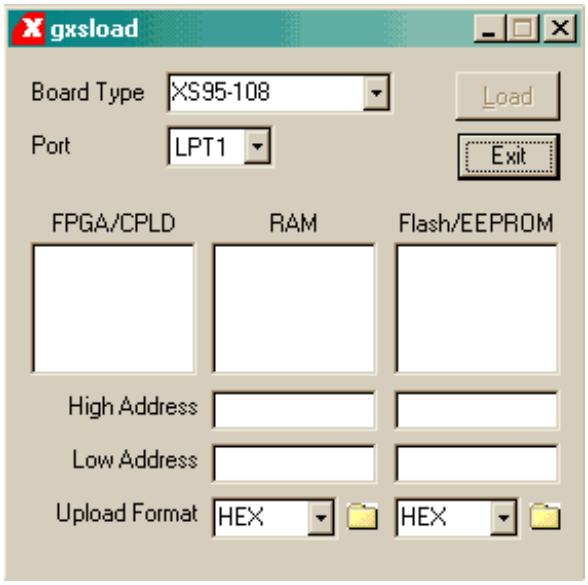


Figura 3.4 Utilidad GSXLOAD

Como en los otros programas, primero hay que seleccionar el tipo de placa y el puerto al que está conectada. Después hay que arrastrar los ficheros .SVF (los que generados por el programa *Xilinx Webpack* y *JTAG*) hacia el recuadro FPGA/CPLD. Pulsando el botón *LOAD* el diseño será cargado en la placa. Aunque sean arrastrados varios archivos al recuadro FPGA/CLPD, sólo se podrá cargar uno a la vez. Para deseleccionar los archivos hay que hacer doble clic sobre ellos.

3.3.2 Carga de datos

Es posible también cargar datos en la RAM externa de la placa arrastrando archivos del tipo .EXO, .MCS, .HEX y/o .XES en el recuadro RAM del programa GXSLOAD y pulsando el botón *LOAD*.

Esto provoca la siguiente secuencia de eventos:

1. El CPLD de la placa se reprograma para crear una interfície entre el chip RAM y el puerto paralelo del PC.
2. El contenido de los archivos .EXO, .MCS, .HEX o .XES son cargados en la RAM a través del puerto paralelo. **ADVERTENCIA:** Los datos sobreescribirán cualquier otro dato que pueda estar en su mismo rango de direcciones.
3. Una vez cargados los datos en la RAM, el archivo de flujo de bits que tengamos seleccionado (recuadro FPGA/CPLD) se cargará en el CPLD.

También es posible examinar el contenido de la RAM transfiriéndolos al PC. Para cargar datos desde un rango de direcciones en la RAM, escribir los rangos superior e inferior en los campos *High Address* y *Low Address* bajo el recuadro RAM, y seleccionar el formato en el cual se van a guardar los datos en la lista *Upload Format*. Lo único que queda por hacer es arrastrar el icono de carpeta hacia alguna otra carpeta y se iniciará la siguiente secuencia:

1. El CPLD de la placa XS95 se reprograma para crear una interfície entre la RAM y el puerto paralelo del PC.
2. Los datos entre las direcciones alta y baja (ambas inclusive) son transferidas a través del puerto paralelo.
3. Los datos descargados son guardados en la carpeta donde hemos arrastrado antes, en un archivo llamado RAMUPLD con la extensión seleccionada.

4. Cómo crear diseños para la placa XS95

En este capítulo se describirá detalladamente el proceso de creación necesario para elaborar y utilizar diseños en la placa XS95. Al no ser un proceso complicado, la mejor manera de explicarlo será mediante un sencillo ejemplo: un contador que irá visualizando en el 7 segmentos, un numero de la secuencia hexadecimal 0 a F cada segundo.

Para ello se explicará cómo utilizar las herramientas de desarrollo disponibles: el simulador *ModelSim*, el compilador/sintetizador *WebPack* y el programa para crear ficheros .svf *najxp*.

4.1 Estructura del ejemplo

Realmente algo tan sencillo se podría hacer sólo con el CPLD, sin necesidad de utilizar la memoria y el microcontrolador, pero como el objetivo no es optimizar recursos sino aprender, serán utilizados todos los componentes de la placa según la disposición de la figura 4.1.

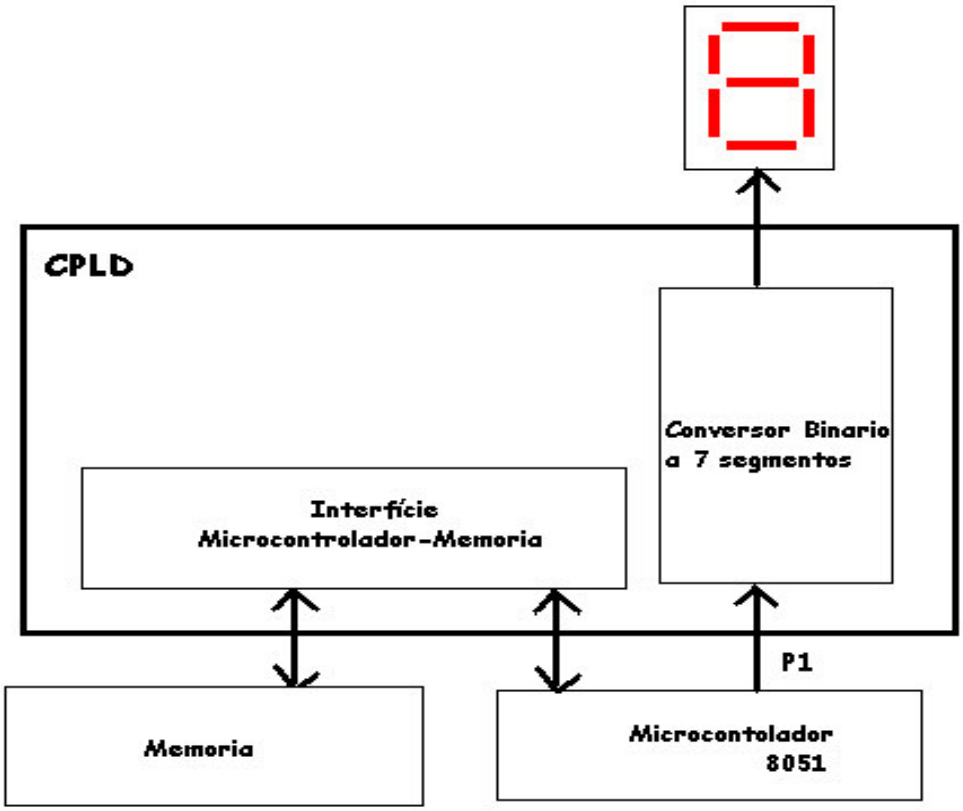


Figura 4.1 Estructura de la aplicación de ejemplo

Como se puede observar en la figura, el CPLD será usado como conversor del número binario que el microcontrolador transmite por el puerto 1 al 7 segmentos. Otro uso será el de interfície entre el microcontrolador y la memoria.

El timer microcontrolador irá controlando el tiempo y cada segundo incrementará el número entre 0 y 15 que pasará al CPLD en los 4 bits más bajos del puerto 1.

4.2 Software necesario

Para la implementación de este proyecto han sido usados los siguientes programas:

- Compilador/Simulador VHDL (ModelSim XE)
- Sintetizador VHDL (Xilinx WebPack)
- JTAG: para crear archivos SVF, descargables a la placa. Se ha utilizado en concreto la versión gratuita *naxjp*.
- ASM51: ensamblador para el microcontrolador 8031/8051
- Simulador 8051
- XSTOOLS

4.3 Proceso de diseño del Microcontrolador + CPLD

En la figura 4.2 se muestra el proceso básico de diseño a seguir para crear una aplicación para el microcontrolador y el CPLD. Como se puede ver, ya se han realizado los tres primeros pasos, así que podemos empezar a diseñar, simular y compilar/sintetizar las funciones del CPLD y el microcontrolador.

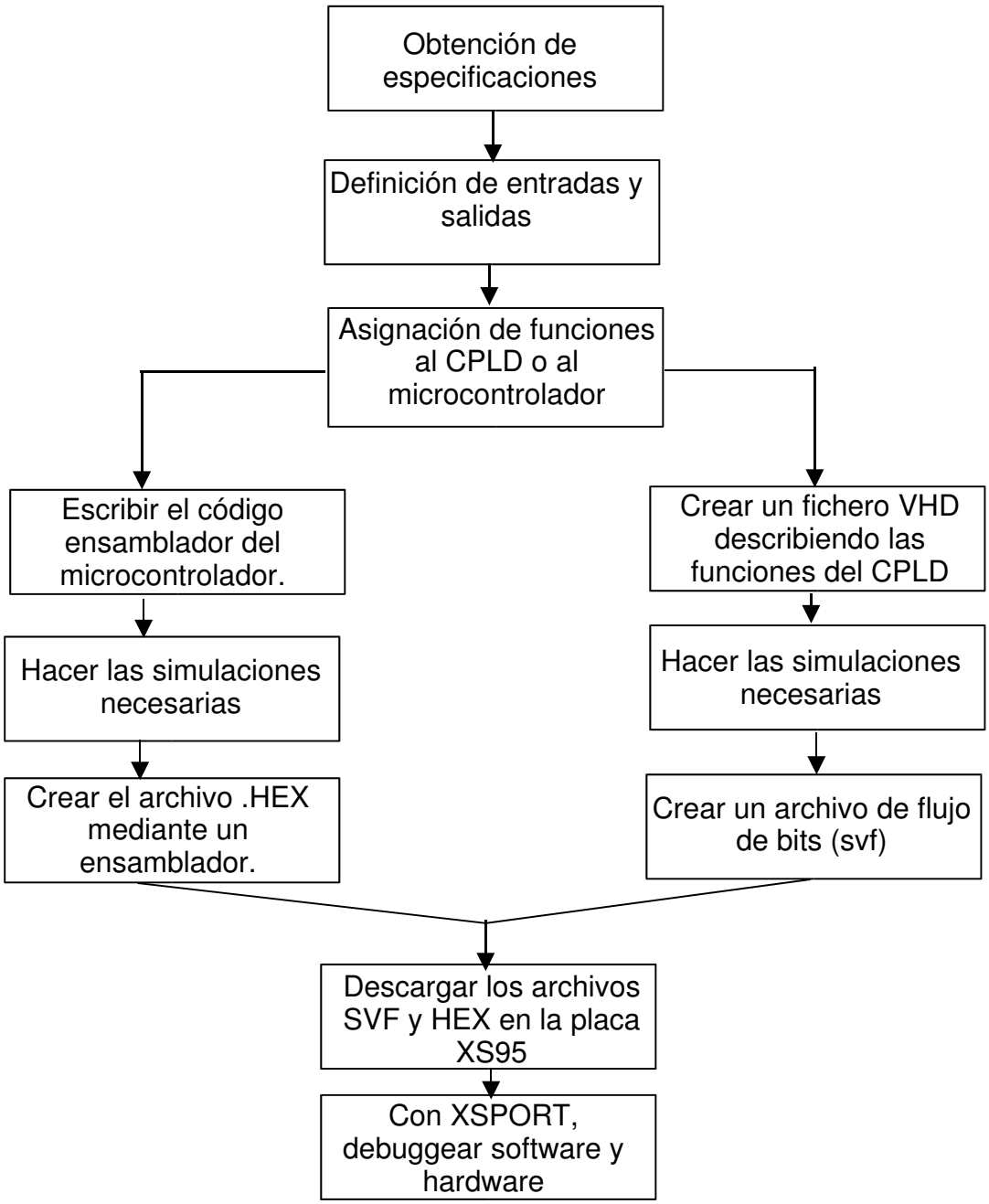


Figura 4.2 Proceso de creación de aplicaciones [Xess 01]

4.3.1 Conversor de binario a 7 segmentos

Empezaremos con la implementación y simulación del conversor. Para ello, seguiremos los siguientes pasos:

- Abrir el programa ModelSim y escoger *Create a New project* en el menú de bienvenida.
- En el diálogo "Create Project" ponemos como nombre del proyecto "ejemplo" y pulsamos el botón OK.
- En el menú *File*, escoger *New-->Source-->VHDL*.
- Escribir el código del conversor en el diálogo de edición:

```
entity conversor is
      port (
            entrada:in bit_vector(3 downto 0);
            salida:out bit_vector(7 downto 0)
      );
end entity conversor;

architecture arch of conversor is
      signal s:bit_vector(7 downto 0);
begin
      process(entrada)
      begin
            case entrada is
                  when "0000" => s<="01110111"; --0
                  when "0001" => s<="00010010"; --1
                  when "0010" => s<="01011101"; --2
                  when "0011" => s<="01011011"; --3
                  when "0100" => s<="00111010"; --4
                  when "0101" => s<="01101011"; --5
                  when "0110" => s<="01101111"; --6
                  when "0111" => s<="01010010"; --7
                  when "1000" => s<="01111111"; --8
                  when "1001" => s<="01111010"; --9
                  when "1010" => s<="01111110"; --A
                  when "1011" => s<="00101111"; --B
                  when "1100" => s<="01100101"; --C
                  when "1101" => s<="00011111"; --D
                  when "1110" => s<="01101101"; --E
                  when "1111" => s<="01101100"; --F
            end case;
      end process;

      salida <= s;
end architecture arch;
```

- Compilación del proyecto: en la ventana principal, clickar en el menú *Options-->Compiler* y marcar la opción *Use 1993 language syntax*. En la ventana de edición, pulsar sobre el primer botón de la barra de herramientas y en el diálogo

pulsar el botón *Compile*. Comprobar y corregir posibles errores.

- Una vez compilado, escribir en la línea de comandos del diálogo principal la siguiente instrucción para comenzar a simular (mirar ayuda del programa para más información sobre la instrucción *vsim*):

```
vsim -t ns work.conversor(arch)
```

- Asignar los valores que queramos probar en la entrada mediante la siguiente instrucción en la línea de caracteres:

```
force entrada 1001 0, 0101 {50 ns} , 1100 {100 ns} , 0000  {150 ns}
```

Hemos asignado a la entrada cuatro valores aleatorios, que van sucediéndose cada 50 nanosegundos.

- En el menú principal pulsar en el menú *View* las opciones *signals y wave* para que aparezcan dichos diálogos.

- Arrastrar las señales que queramos ver desde la ventana *signals* hasta la ventana *wave*.

- En la ventana *wave*, ir pulsando el botón *Run* de la barra de herramientas para ir simulando paso a paso e ir viendo los resultados de la simulación.

4.3.2 Interfície de memoria

Para que el microcontrolador pueda leer el programa que le mandamos a memoria, será necesario implementar en el CPLD una interfície entre ambos componentes. El proceso de creación y simulación es exactamente el mismo que para el conversor. A continuación se adjunta el código fuente de la interfície de memoria:

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

entity interficie_memoria is
      port(
            --Microprocesador
            AL:in std_logic_vector(7 downto 0);
            AH_7:in std_logic;
```

```
            nRD:in std_logic;
            nPSEN:in std_logic; --Program Store ENable
            ALE:in std_logic; --Address Latch Enable

            XTAL1:out std_logic;
            outRST:out std_logic;


            --Memoria
            A:out std_logic_vector(7 downto 0);
            nOE:out std_logic;
            nCE:out std_logic;

            --Otros
            CLK:in std_logic;
            inRST:in std_logic
      );
end entity interficie_memoria;

architecture arch of interficie_memoria is
      signal sigA,sigAH:std_logic_vector(7 downto 0);
begin
      outRST<=inRST;
      XTAL1<=CLK;
      nOE<=not (not nRD or not nPSEN);
      nCE<=AH_7;--AH(7);
      process(ALE,CLK)
      begin
            if ALE='0' and ALE'event then
                  sigA<=AL;
            end if;
      end process;

      A<=sigA;
end architecture arch;
```

4.3.3 Conversor + Interfície de memoria

Ahora es el momento de implementar en el CPLD tanto el conversor para el 7 segmentos como la interfície de memoria. Primero hay que crear un nuevo componente que aúne a los dos.

- Abrir *Modelsim* y crear un nuevo proyecto llamado intyconv.

- En la ventana principal, pulsar el segundo botón del mouse y escoger opción *Add file to project*.

- Escoger los dos ficheros .vhd creados anteriormente para el conversor y la interficie.

- Crear un nuevo fichero vhdl llamado *intyconv.vhd* y escribir el siguiente código.

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

entity intyconv is
port (
      --conversor
      entrada:in bit_vector(3 downto 0);
      salida:out bit_vector(7 downto 0);

      --interficie de memoria
            --Microprocesador
            AL:in std_logic_vector(7 downto 0);
            AH_7:in std_logic;
            nRD:in std_logic;
            nPSEN:in std_logic; --Program Store ENable
            ALE:in std_logic; --Address Latch Enable

            XTAL1:out std_logic;
            outRST:out std_logic;


            --Memoria
            A:out std_logic_vector(7 downto 0);
            nOE:out std_logic;
            nCE:out std_logic;

            --Otros
            CLK:in std_logic;
            inRST:in std_logic
);
end entity intyconv;

architecture arch of intyconv is
component conversor is
port (
      --conversor
      entrada:in bit_vector(3 downto 0);
      salida:out bit_vector(7 downto 0)
);
end component conversor;

component interficie_memoria is
port(
      --Microprocesador
      AL:in std_logic_vector(7 downto 0);
      AH_7:in std_logic;
      nRD:in std_logic;
      nPSEN:in std_logic; --Program Store ENable
      ALE:in std_logic; --Address Latch Enable

      XTAL1:out std_logic;
      outRST:out std_logic;


      --Memoria
      A:out std_logic_vector(7 downto 0);
```

```
        nOE:out std_logic;
        nCE:out std_logic;

        --Otros
        CLK:in std_logic;
        inRST:in std_logic
    );
    end component interficie_memoria;

    begin
        CONV: conversor port map (entrada,salida);
        INTF: interficie_memoria port map ( AL, AH_7,
                        nRD,nPSEN,ALE,XTAL1,outRST,A,nOE,nCE,
                        CLK,inRST );
    end architecture arch;
```

- Abrir la ventana de edición de cada componente y compilar para comprobar errores.

- Hacer las simulaciones necesarias.

Ahora es momento de cargar en el CPLD el diseño hecho. Para ello utilizaremos el programa *Xilinx Webpack* siguiendo los siguientes pasos:

- Abrir el programa *Webpack Project Navigator*.

- Crear un nuevo proyecto mediante el menú *File-->New Project*. Darle al proyecto el nombre de *intyconv.*

- Escoger el modelo de programable a usar. En el caso de la placa XS95, escoger las siguientes opciones:

  Device Family: XS9500 CPLD

  Device: XC95108 PC84

- En *Design flow*, escoger XST VHDL y pulsar OK.

- Clickar con el segundo botón del mouse en el cuadro *Sources in project* del diálogo principal, y escoger la opción *Add sources* para añadir los archivos .vhd creados anteriormente con *Modelsim*.

- Ahora es el momento de asignar a qué pines irá cada entrada o salida del diseño.

Para ello hay dos opciones:

1. Mediante una interfície gráfica: En el cuadro *Process View,* expandir por *Dessign Entry Utilities* y *User constraints*, y hacer doble click en la opción *Assign Pins*. En la ventana que aparece, habrá que clicar en la entrada o salida a asignar y luego en la patilla del chip en la cual lo vamos a hacer.

2. Manualmente, mediante un archivo .ucf: En el cuadro *Process View,* expandir por *Dessign Entry Utilities* y *User constraints*, y hacer doble click en la opción *Edit implementation constraints file*. Cada pin a asignar se indicará de la siguiente manera:

   NET <nombre_pin> LOC=P<numero_patilla>;

La asignación de patillas no se hace arbitrariamente, hemos de mirar los manuales de la placa para saber cómo están conectados sus componentes y mirar a qué patillas debemos conectar las entradas y salidas. En el anexo A se adjunta un esquema de la placa, con las conexiones y los números de patillas del CPLD.

Para este ejemplo, el archivo .ucf es el siguiente:

```
// Template UCF file created by the Project Navigator
NET inrst LOC=P80;
NET clk LOC=P9;
NET npsen LOC=P13;
NET ale LOC=P20;
NET nrd LOC=P32;
NET noe LOC=P62;
NET nce LOC=P65;
NET xtal1 LOC=P10;
NET outrst LOC=P45;
NET a<0> LOC=P75;
NET a<1> LOC=P79;
NET a<2> LOC=P82;
NET a<3> LOC=P84;
NET a<4> LOC=P1;
NET a<5> LOC=P3;
NET a<6> LOC=P83;
NET a<7> LOC=P2;
```

```
NET al<0> LOC=P44;
NET al<1> LOC=P43;
NET al<2> LOC=P41;
NET al<3> LOC=P40;
NET al<4> LOC=P39;
NET al<5> LOC=P37;
NET al<6> LOC=P36;
NET al<7> LOC=P35;
NET ah_7 LOC=P34;

NET entrada<0> LOC=P6;
NET entrada<1> LOC=P7;
NET entrada<2> LOC=P11;
NET entrada<3> LOC=P5;

NET salida<0> LOC=P21;
NET salida<1> LOC=P23;
NET salida<2> LOC=P19;
NET salida<3> LOC=P17;
NET salida<4> LOC=P18;
NET salida<5> LOC=P14;
NET salida<6> LOC=P15;
NET salida<7> LOC=P24;
```

Observar que todos los nombres de las entradas y salidas han de ser en minúsculas.

- En *Process View*, clickar dos veces en *Generate Programming File* para generar el archivo del programable.

- El siguiente paso es crear el archivo .svf para cargar en el programable. Esto se puede conseguir mediante el programa *Naxjp* ejecutando el siguiente comando en una ventana MS-DOS:

  naxjp.exe -write intyconv.jed -svf intyconv.svf

Si ahora cargásemos el archivo intyconv.svf mediante el programa *gxsload* veríamos cómo se muestra una F en el 7 segmentos, ya que el microcontrolador al inicializarse saca un 255 por el puerto 1.

4.3.4 Programación del microcontrolador

Como último paso queda crear un programa que mediante el temporizador mande los datos necesarios al puerto 1. Como este es un programa sencillo, no tendremos necesidad de depurarlo. Se puede usar cualquier ensamblador para compilarlo, las directivas del siguiente código en cuestión son para el ensamblador ASM51. Para más información sobre la programación del microcontrolador 8031, mirar bibliografía *[Yeralan Ahluwalia 95].*

```
       $MOD51

contador equ 128

       CSEG

       mov tmod, #80h      ; configura el contador 0
       setb tr0            ; activa el contador/temporizador 0

       mov r0,contador     ;
       mov p1,#0

loop:  jnb tf0,loop        ; mira si el timer activa tf0
       clr tf0             ; cuando lo activa, lo borra
       djnz r0,loop        ; y decrementa r0, si es >0, va a loop
       mov r0,#contador    ; si no, reinicia r0

       inc p1              ;  e incrementa el valor del puerto 1
       sjmp loop

       end
```

Ahora sólo queda cargar los respectivos archivos .svf y .hex mediante gxload, tal y como se explicó en el capítulo anterior, apartado 3.3.1. Notar que para que el microprocesador salte a la dirección de memoria 0 hay que pulsar el reset, que está colocado en la línea 7 del port paralelo (modificable desde *gxsport*).

5. Diseño del módulo VGA

En este capítulo se procederá al diseño del módulo VGA utilizado en el proyecto. Para más información, consultar el anexo B y la bibliografía *[Engdahl 01]*.

5.1 Estructura básica de una tarjeta VGA

Básicamente consta de un módulo VGA, una memoria de vídeo, un conversor analógico-digital y un conector hacia el monitor por el cual se envían las señales de color y sincronismo, entre otras que no nos hace falta conocer para crear la aplicación VGA. En la figura 5.1 se muestra la estructura.

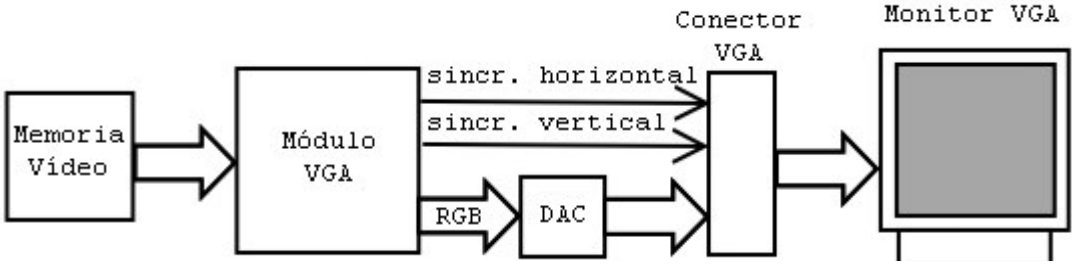


Figura 5.1 Estructura básica de una tarjeta VGA

5.2 Diagramas de tiempo del módulo VGA

Los diagramas de tiempo de la VGA usados para las señales HSYNC y VSYNC son los siguientes:

HSYNC

VSYNC

Figura 5.2 Diagramas de tiempo para el módulo VGA

En el módulo VGA a implementar utilizaremos un ciclo de reloj para procesar cada pixel. Nuestro reloj funciona a 12 MHz. A esta velocidad podremos procesar 25.17*12=302 pixeles por fila. Para visualizar sólo 256 pixeles utilizaremos un contador que se incremente a cada ciclo de reloj y cuando pase de 256 deshabilite la salida RGB.

Si HSYNC es en función del reloj del sistema, VSYNC utilizará como reloj la salida HSYNC. Si cada ciclo HSYNC ocupa 31,77us, en 15.25ms VSYNC podrá procesar 15.250/31,77=480 líneas.

Si realizamos los siguientes cálculos, las señales HSYNC y VSYNC, en función de los ciclos de reloj, quedarán tal y como se muestra en la figura 5.3.

Para el sincronismo horizontal:

25.17 us x12 MHz = 302 ciclos
0.94 us x 12 MHz = 11 ciclos
3.77 us x 12 Mhz = 46 ciclos
1.89 us x 12 MHz = 22 ciclos

Total: 381 ciclos de reloj por línea HSYNC. Esto nos proporcionará a su vez un reloj de 12M / 381 = 31.496 Hz, que conectaremos al contador de VSYNC, obteniendo los siguientes resultados:

15.25 ms x 31.496 KHz = 480 ciclos
0.45 ms x 31.496 KHz = 14 ciclos
64 us x 31.496 KHz = 3 ciclos
1.02 ms x 31.496 KHz = 32 ciclos

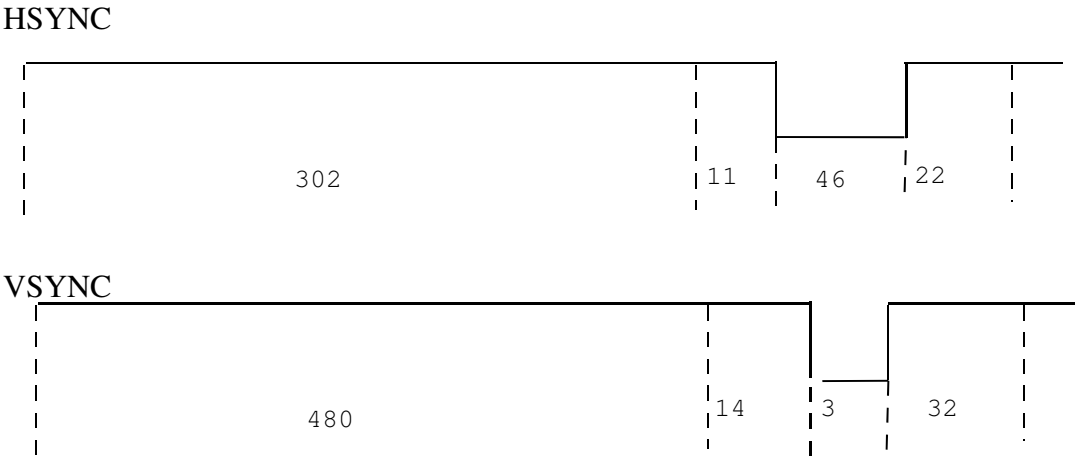


Figura 5.3 Diagramas de tiempo en función de los ciclos de reloj

Para calcular el refresco de pantalla hemos de hacer el siguiente cálculo:

1000 ms / (15.25 + 0.45 + 0.064 + 1.02) ms = 59.58 Hz

Lo que aproximadamente son 60 imágenes por segundo.

5.2 Características del módulo VGA a implementar

El módulo VGA en cuestión muestra el contenido de la memoria en una imagen monocroma, pudiendo visualizar desde la dirección de inicio que se quiera.

Al ser una imagen monocroma, cada byte podrá albergar 8 pixeles, si queremos una definición de 256 columnas x 192 filas, cada imagen ocupará (256/8) * 192 = 6144 bytes. Si disponemos de un chip de 32 KB, podremos tener una pantalla virtual de 256x1024 líneas.

Aparte de la pantalla virtual está la parte visible en memoria, que será de 256x192 pixeles por usar una resolución proporcional. Para hacer el scroll de la pantalla se utiliza el registro de scroll. El scroll es cíclico, es decir, cuando se visualiza la linea 1023 se continúa visualizando por la línea 0.

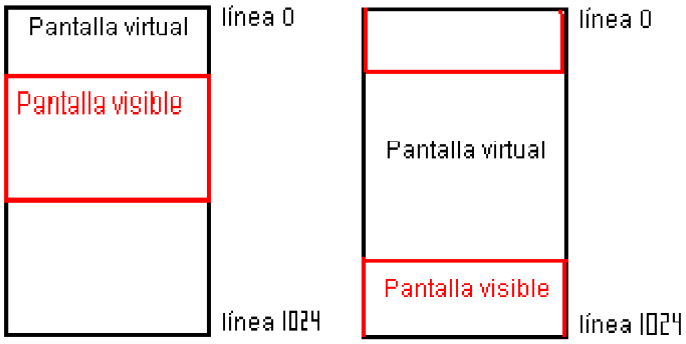


Figura 5.4 Ejemplo de scroll de pantalla cíclico

5.2.1 Modos de visualización

Como el diseño sólo utiliza 192 líneas de las 480 disponibles, el aspecto de la pantalla estará deformado tal y como se muestra en la figura 5.5. Para solucionar esto, se podrá configurar la pantalla mediante un latch llamado STATUS:

1 - Si STATUS=0 --> visualizar las 192 líneas seguidas. Es un método adecuado para dispositivos de visualización con resoluciones verticales bajas. De esta forma, en un monitor convencional la imagen seria representada de la siguiente manera:

Figura 5.5 Modo de pantalla normal

2 - Si STATUS=1 --> visualizar dos veces la misma línea. Es un método conveniente para monitores convencionales, en los cuales la imagen saldría de otra manera deformada.

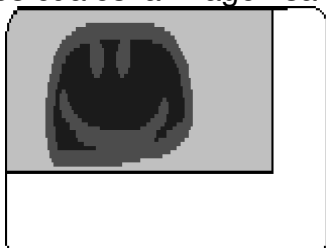


Figura 5.6 Modo de pantalla ampliado

5.2.2 Entradas-salidas del módulo VGA básico

Las entradas-salidas del módulo están detalladas a continuación:

- **Entradas**
  - **clk** : entrada de reloj.
  - **r** : reset, cuando es 1 deja de visualizarse la imagen y se inicializan todos los registros internos.
  - **d<7..0>** : bus de datos, por donde se reciben los datos de la memoria.

- **Salidas**
  - **nvsync, nhsync** : señales de sincronismo horizontal y vertical del monitor. Activas a 0.
  - **RGB<5..0>** : datos sobre el color. Están conectados a un conversor digital-analógico que manda al monitor el señal referente al color.
  - **a<14..0>** : bus de direcciones. 15 bits para direccionar los 32 KB de memoria. Atención: si se usa el modelo XS95-108+ (128 KB) hay que aumentar el tamaño de este bus a 17 bits.
  - **nOE, nCE, nWE** : *output enable, chip enable y write enable*, señales para la configuración de la memoria. Activas a 0.

A parte de estas entradas y salidas básicas, en la aplicación de ejemplo se han añadido dos entradas más.

- **Status** : indica el valor del registro STATUS (0: se visualiza una línea tras otra, 1: se visualiza dos veces la misma línea).
- **Scroll<9..0>** : valor del registro de scroll. De la pantalla virtual de 256x1024, indica el número de fila por el que se empieza a visualizar en el monitor.

6. Creación de la aplicación VGA de muestra

Para mostrar las posibilidades del módulo VGA diseñado en el capítulo anterior, haremos una aplicación que visualice por pantalla una animación, aprovechando el registro de scroll para pasar de un frame a otro.

6.1 Estructura de la aplicación

Esta aplicación se divide en dos partes diferenciadas: el Módulo VGA y el Módulo de control. El proceso de creación es exactamente igual al visto en el capítulo 4. En el anexo C se incluyen los códigos fuente de la aplicación.

6.1.1 Módulo VGA

Se encarga de visualizar en pantalla el contenido de la memoria a partir de donde indica el registro de scroll. En la figura 6.1 se muestra un esquema simplificado, que ayudará a comprender el funcionamiento.

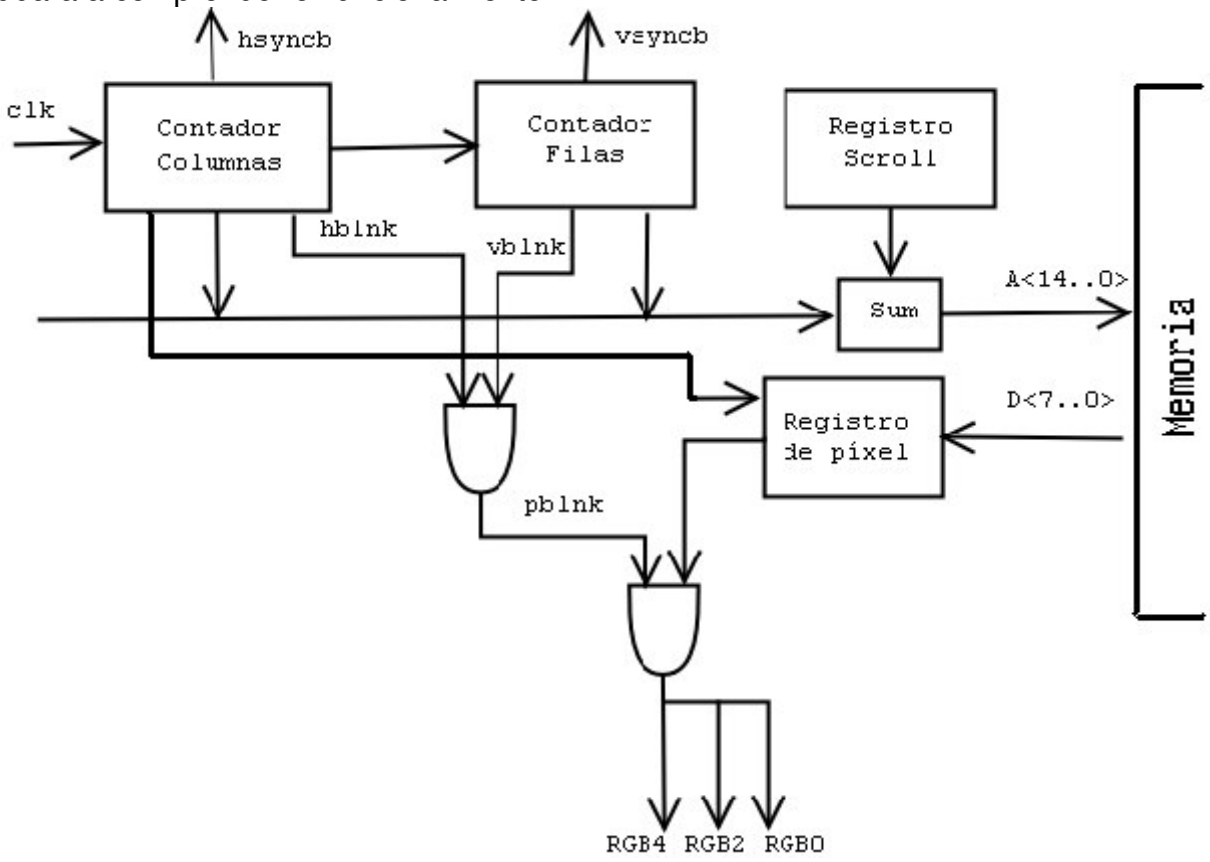


Figura 6.1 Esquema básico del módulo VGA

El funcionamiento es sencillo: a cada ciclo de reloj el contador de columnas se va incrementando. Tiene conectada una salida al contador de filas, cuyo valor está descrito en los diagramas de tiempo del capítulo anterior (figura 5.2). Ambos van generando las señales de blanqueo y sincronismo y, junto con el registro de scroll, se genera la dirección de memoria donde están los pixeles a visualizar.

El registro de pixel va desplazándose a cada ciclo de reloj con tal de visualizar el pixel correspondiente al bit más significativo. Cuando el contador de columnas es un número múltiplo de 8, carga el byte de memoria correspondiente.

La salida RGB envía el bit más significativo del registro de pixel. Cuando la señal de blanqueo es 0, la salida se desactiva.

6.1.2 Módulo de control

Se encarga de ir cambiando el registro de scroll cada determinado número de frames, de tal manera que de la sensación de que la imagen se mueve. Es una máquina de estados: cuando se visualiza el ultimo frame, cambia el estado y el registro de scroll se va decrementando en 192, y cuando se visualiza el primer frame, se pasa al estado en el cual el registro de scroll se va incrementando en 192.

6.2 Carga en memoria del gráfico

Para meter en memoria, se ha creado un bitmap monocromo de tamaño 256x960, en el que caben cinco imágenes de 256x192. Con una aplicación creada especialmente para este proyecto, llamada *bmp2hex*, se crea un fichero .hex que carga los datos a partir de la memoria 0. El gráfico en cuestión son las diferentes posiciones de una pelota que va botando (figura 6.2).

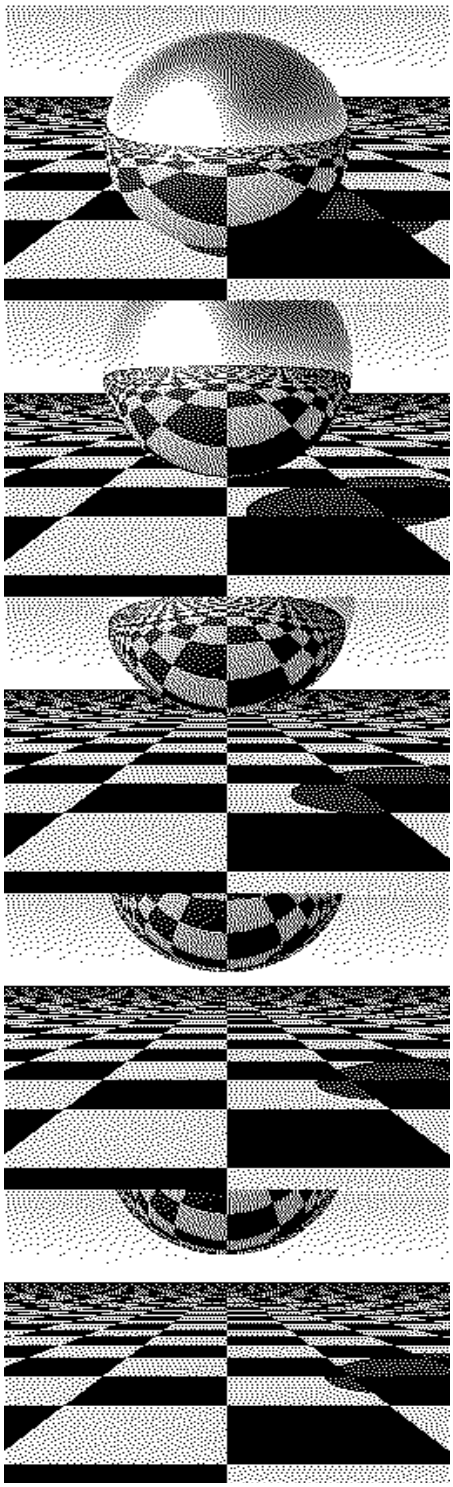


Figura 6.2 Gráfico residente en la memoria de la placa

7. Conclusiones

7.1 Conclusiones

Más que la creación de una sistema que utilice una controladora VGA, creo que el trabajo de mayor importancia ha sido el estudio y utilización de las placas de prototipado XESS, así como de la metodología de diseño y desarrollo y de las herramientas disponibles para tal fin. Espero que este proyecto sirva para que cualquiera que quiera crear por primera vez aplicaciones sobre las placas Xess, tenga a mano la documentación necesaria para iniciarse con rapidez y de una manera sencilla, sin tener que recurrir a información extensa, desperdigada y a veces poco clara, como ha sucedido durante este proyecto.

En lo referente a la controladora VGA, se ha conseguido crear una controladora sencilla, con algunas características como scroll de pantalla o una configuración de la imagen que permita la correcta visualización tanto en monitores convencionales como en pantallas integradas.

Por razones de tiempo, no ha sido posible crear un sistema en el que se utilizara el CPLD como controladora VGA y el microcontrolador, aunque se ha puesto un ejemplo en el que se usa un CPLD y el microcontrolador.

7.2 Líneas Abiertas

Creo que si se continúa el trabajo realizado hasta ahora, el siguiente paso debería ser la realización de una controladora VGA que utilizara también el microcontrolador. El problema que hay es que hay un conflicto a la hora de compartir el chip de memoria, ya que ambos intentan acceder a la vez.

Hay que dar prioridad al módulo VGA a la hora de acceder a memoria, por lo que el microcontrolador no podrá ejecutar ni siquiera un programa, ya que la memoria de vídeo es también la memoria de programa para el microcontrolador. Mirando las características del modelo de microcontrolador incluido en la placa *[Winbond 99]* y de cómo éste está

interconectado, creo que la mejor manera de arbitrar los accesos a memoria es parar el microcontrolador mientras se esté visualizando en pantalla. Para parar el microcontrolador tan sólo habría que parar la señal de reloj, ya que este modelo lo permite, y reiniciarla cuando se quiera volver a activar.

A parte de esto, cada vez que se quisiera parar el microcontrolador habría que guardar el estado de las entradas y salidas de éste, y restaurarlo cuando se quisiera activar.

7.3Problemas encontrados

El mayor problema surgido durante la realización de este proyecto ha sido el problema de la logística. No es fácil obtener una placa de prototipado Xess, ya que en Europa no hemos encontrado ningún distribuidor y hemos tenido que pedirlas directamente a Xess, con la que luego hubieron a la hora de escoger la forma de pago. Para hacernos una idea, si el proyecto se inició en Noviembre de 2001, las placas no se han obtenido hasta Agosto de 2002, un mes antes de la finalización del mismo.

Apéndice: familias de placas de prototipado XESS

Aunque el numero de placas es bastante grande, se ha escogido para comparar un modelo representativo de cada familia.

SX95-108

Características técnicas

- CPLD XC95108
  - 2.400 puertas lógicas
- Controlador 8031
- 32K SRAM
- Oscilador programable 100 MHz
- Puerto paralelo
- Puerto PS/2
- Puerto para monitores VGA
- LED de 7 segmentos
- Interfície de prototipado de 84 pins
- Socket serie EEPROM
- Jack de 9V DC
- Regulador de 5V
- Precio: 119$

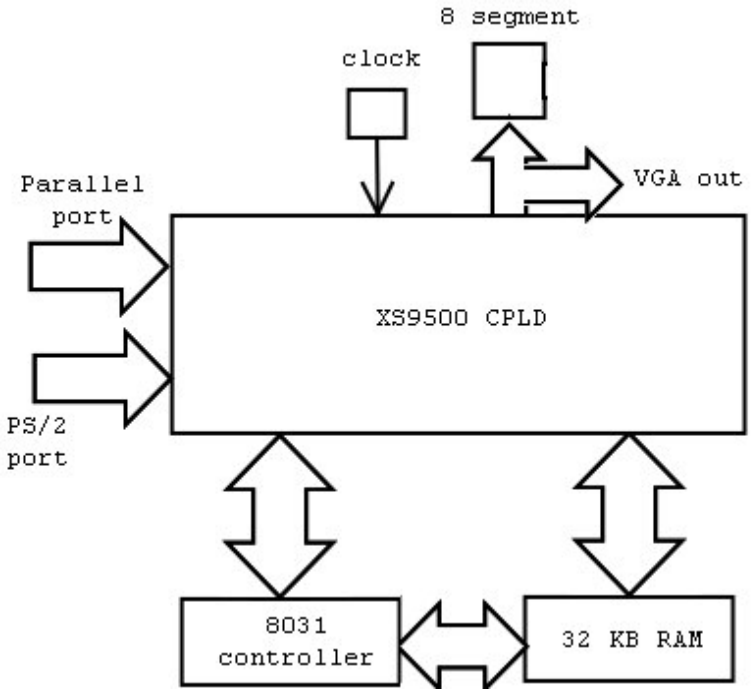


Figura A1. Esquema de la placa XS95

La placa utilizada en este proyecto. Su dispositivo programable es de poca capacidad, pudiéndose quedar pequeño en cuanto se quiere diseñar una aplicación con cierto grado de complejidad. Si se va a utilizar para visualizar gráficos, sobretodo si son en color, es posible que 32 KB de memoria se queden cortos. El microcontrolador es bastante sencillo, pero suficiente, contando que los diseños estarán limitados por el tamaño del CPLD y de la memoria antes que por el microcontrolador.

XS40-005XL

Características técnicas

- FPGA XC4005XL
  - 9000 puertas lógicas
- Controlador 8031
- 32K SRAM
- Oscilador programable 100 MHz
- Puerto paralelo
- Puerto PS/2
- Puerto para monitores VGA
- LED de 7 segmentos
- Interfície de prototipado de 84 pins
- Socket serie EEPROM
- Jack de 9V DC
- Regulador de 5V
- Precio: 129$

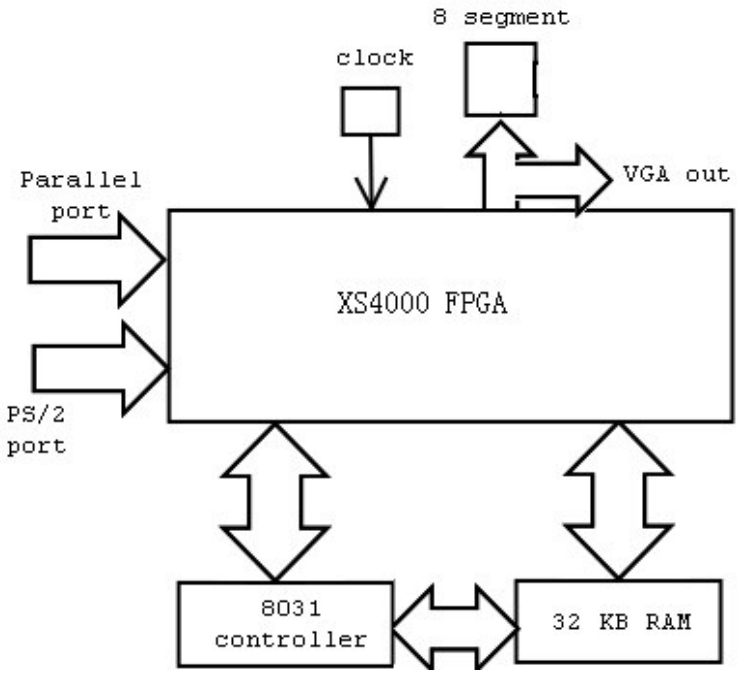


Figura A2. Esquema de la placa XS40

Similar a la otra, pero ofrece casi 4 veces más puertas por tan sólo 10$ más. Es una opción interesante, aunque 32 KB pueden ser un poco justos. Junto con XS95, la opción ideal para la iniciación.

XSA-100

Características técnicas

- FPGA XC2S100
  - 100.000 puertas lógicas
- 16 MB RAM
- 256KB Flash
- Oscilador Programable 100 MHz
- Puerto paralelo
- Puerto PS/2
- Puerto para monitores VGA
- LED de 7 segmentos
- Interfície de prototipado de 84 pins
- Socket serie EEPROM
- Jack de 9V DC
- Reguladores de 5V/3.3V/2.5V
- Precio: 279$

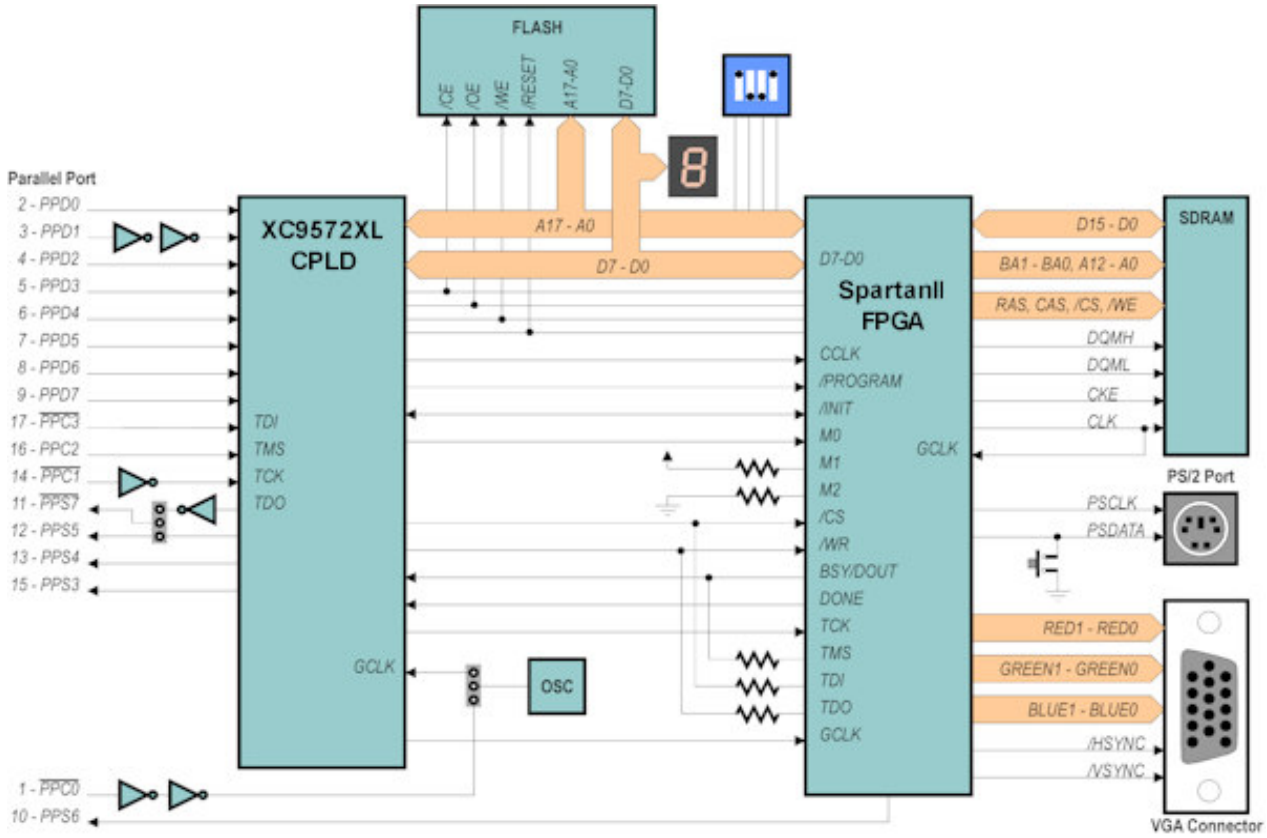


Figura A3. Esquema de la placa XSA-100 (Fuente: Xess Corp)

Hay un incremento enorme de prestaciones respecto a las placas anteriormente comentadas. Para personas ya iniciadas, que exijan a una aplicación características que las anteriores no pueden proporcional. Observar que no incluye microcontrolador, si se necesita uno hay que implementarlo en la FPGA o el CPLD.

XSV-800

Características técnicas

- FPGA XCV800 Virtex
  - 50.000 - 1.000.000 puertas
- XC95108 CPLD
- Dos chips SRAM 512K x 16 SRAM
- Descodificador de vídeo PAL/SECAM/NTSC
- RAMDAC 110 MHz
- Ethernet PHY 10/100
- 16 Mbit Flash RAM
- Oscilador programable 100 MHz
- Dos bancos de prototipado, cada uno con 38 E/S de propósito general
- 4 pulsadores
- *DIP switches*
- Dos 8 segmentos
- *Codec* 20-bit stereo
- *Jacks* E/S stereo
- Salida monitor VGA
- puerto PS/2
- Puerto USB (host o cliente)
- Conexiones puerto série y paralelo
- Entrada Alimentación ATX o *Jack* de 9V DC
- Precio: 1.599$

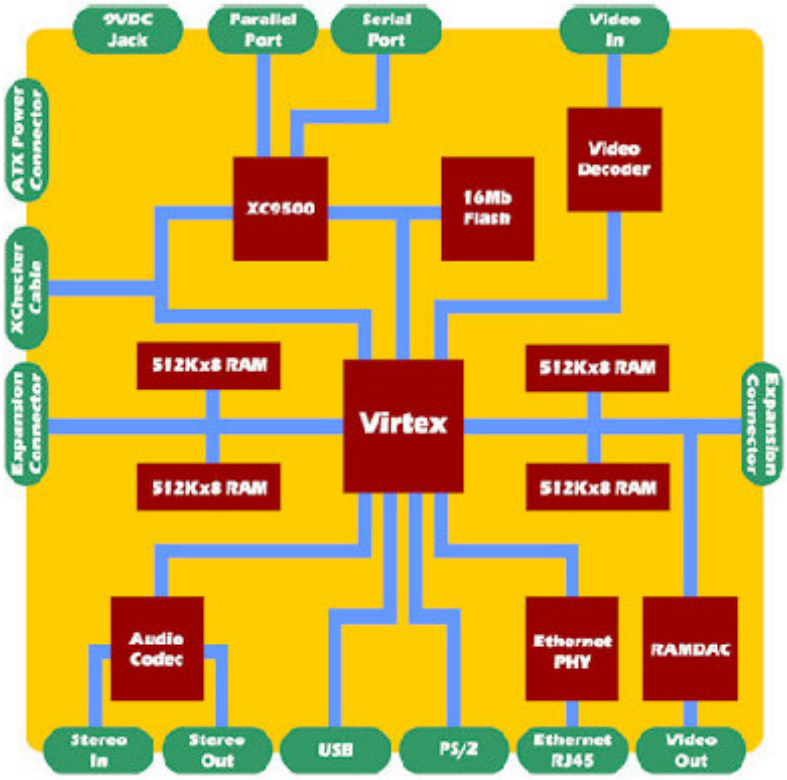


Figura A4. Esquema de la placa XSV-800 (Fuente: Xess Corp)

Sin duda una gran máquina, pero se sale del ámbito académico. Sólo para aplicaciones profesionales que requieran un gran rendimiento.

XSTE5 CSoC

Características técnicas

- Triscend TE505 CSoC
- 128KB RAM
- 128KB Flash
- Oscilador Programable 100 MHz
- Puerto paralelo
- Puerto PS/2
- Puerto para monitores VGA
- LED de 7 segmentos
- Interfície de prototipado de 84 pins
- Socket serie EEPROM
- Jack de 9V DC
- Reguladores de 5V/3.3V
- Precio: 169.95$

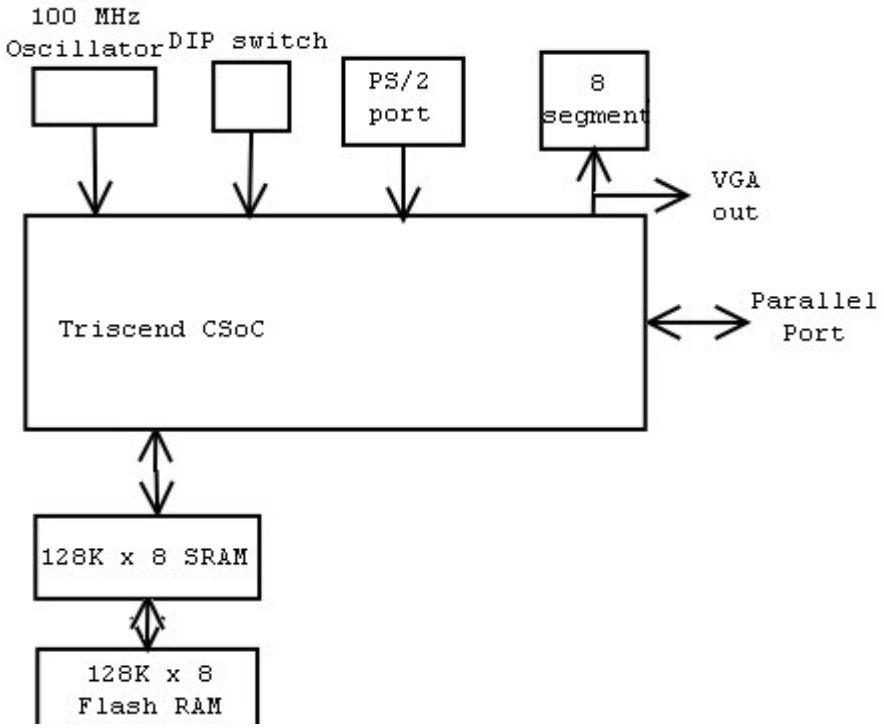


Figura A5. Esquema de la placa XTE5 CSoC

Esta placa tiene similares características a la XS95, pero con un incremento substancial en la memoria. Este dispositivo no usa FPGA, sino un CSoC (Configurable System on Chip), que integra en un solo chip un microcontrolador 8032, dos controladores DMA, 16 KBytes RAM y 512 células programables (para hacernos una idea, el CPLD de la placa XS95 tiene 108 células programables).

Anexo A. Conexionado de la placa XS95

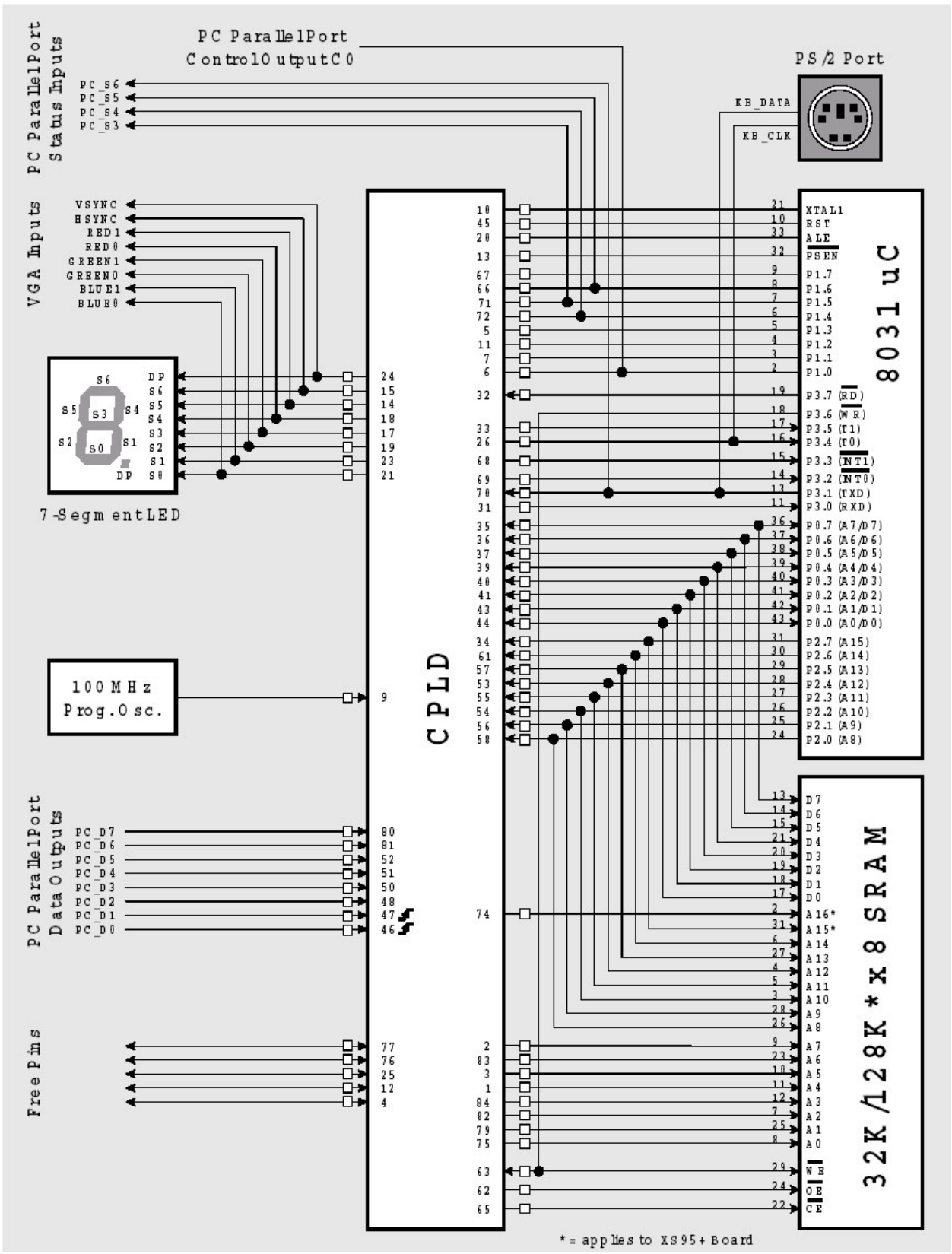


Figura A6. Conexionado de la placa XS95-108

Anexo B. Fuentes de la aplicación de demostración

Fichero demo.vhd

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

entity demo is
   Port (
            CLK:in std_logic;
            Reset:in std_logic;
            D:in std_logic_vector(7 downto 0);              --bus de datos

            A:out std_logic_vector(14 downto 0);            -- bus de direcciones

            nOE,nCE,nWE:out std_logic;
            vsyncb:out std_logic;
            hsyncb:out std_logic;
            RGB:out std_logic_vector(5 downto 0)            --Salida RGB


      );
end entity demo;

architecture arch_demo of demo is
      component vgacore is
      port(
            --entradas
            CLK:in std_logic;
            R:in std_logic;                                 --Reset
            D:in std_logic_vector(7 downto 0);              --bus de datos

            A:out std_logic_vector(14 downto 0);            -- bus de direcciones

            vsyncb:out std_logic;
            hsyncb:out std_logic;
            iscroll:in std_logic_vector(9 downto 0);
            RGB:out std_logic_vector(5 downto 0)            --Salida RGB

      );
      end component vgacore;

      signal scroll:std_logic_vector(9 downto 0);
      signal sigvsyncb:std_logic;
      signal estadoscroll:std_logic;
      signal contframes:std_logic_vector(2 downto 0);

begin
      VGA_core: VGAcore port map (CLK,Reset,D,A,sigvsyncb,hsyncb,scroll,RGB);
      nOE<='0';
      nCE<='0';
      nWE<='1';
      vsyncb<=sigvsyncb;

      animacion:process(sigvsyncb,Reset,scroll)
      begin
            if Reset='1' then
                  scroll<="0011000000";
```

```
                estadoscroll<='0';
                contframes<=(others=>'1');
          else
                if sigvsyncb'event and sigvsyncb='1' then
                      contframes<=contframes+1;
                      if contframes=0 then
                            if estadoscroll='0' then
                                  scroll<=scroll+192;
                            else
                                  scroll<=scroll-192;
                            end if;
                            if estadoscroll='0' and scroll=576 then
                                  estadoscroll<='1';
                            end if;
                            if estadoscroll='1' and scroll=192 then
                                  estadoscroll<='0';
                            end if;
                      end if;
                end if;
          end if;
      end process animacion;
end architecture arch_demo;
```

Fichero Vgacore.vhd

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

entity VGAcore is
port(
      --entradas
      CLK:in std_logic;
      R:in std_logic;                                  --Reset
      D:in std_logic_vector(7 downto 0);               --bus de datos

      A:out std_logic_vector(14 downto 0);             -- bus de direcciones


      vsyncb:out std_logic;
      hsyncb:out std_logic;
      iscroll:in std_logic_vector(9 downto 0);
      RGB:out std_logic_vector(5 downto 0)             --Salida RGB

);
end entity;


architecture arch_VGAcore of VGAcore is

      signal cont_col:std_logic_vector(8 downto 0);
      signal cont_fil:std_logic_vector(9 downto 0);
      signal cont_fil_mode:std_logic_vector(9 downto 0); --Cuenta las filas
cuando esta la pantalla en modo amplio

      signal hblnk:std_logic; --blanqueo horizontal
      signal vblnk:std_logic; --blanqueo vertical
      signal hsync:std_logic; --sincronismo horizontal
      signal vsync:std_logic; --sincronismo vertical
      signal pixreg:std_logic_vector(7 downto 0); --registro de desplazamiento
con los pixels
```

```
      signal spblnk:std_logic;
      signal SCROLL:std_logic_vector(9 downto 0); --registro de scroll
      signal linea:std_logic; --utilizado para el modo de pantalla

      signal status:std_logic;      --para el modo de pantalla
                                    --0 : dibuja las lineas una detras de otra
                                    --1 : dibuja dos veces la misma linea (dobla
el tamaño vertical)


begin
      SCROLL<=iscroll;
      status<='1';

      cuenta_columnas:process(clk,r)
      begin
            if clk='1' and clk'event then
                  if R='1' then
                        cont_col<="100111011";  --315
                  else
                        if cont_col>382 then
                              cont_col<="000000000";
                        else
                              cont_col<=cont_col+1;
                        end if;
                        if cont_col>255 then
                              hblnk<='0';
                        else
                              hblnk<='1';
                        end if;
                        if cont_col>314 and cont_col<361 then
                              hsync<='0';
                        else
                              hsync<='1';
                        end if;
                  end if;
            end if;
      end process cuenta_columnas;

      cuenta_filas:process(clk,hsync,r,STATUS)
      begin
            if R='1' then
                  cont_fil<="0111101110";  --494
                  cont_fil_mode<="0011110111"; --247
            else
                  if hsync='1' and hsync'event then
                        if cont_fil>527 then
                              cont_fil<="0000000000";
                        else
                              cont_fil<=cont_fil+1;
                              if STATUS='0' then
                                    cont_fil_mode<=cont_fil;
                              else
                                    cont_fil_mode<="0" & cont_fil(9 downto 1);
                              end if;

                        end if;
                        if cont_fil_mode>190 then
                              vblnk<='0';
                        else
                              vblnk<='1';
```

```
                        end if;

                        if cont_fil>493 and cont_col<497 then
                              vsync<='0';
                        else
                              vsync<='1';
                        end if;
                  end if;
            end if;
      end process cuenta_filas;

      reg_pixel:process(clk)
      begin
            if clk='1' and clk'event then
                  if cont_col(2 downto 0)=0 then
                        pixreg<=D;
                  else
                        pixreg<=pixreg(6 downto 0) & '0';
                  end if;
            end if;
      end process reg_pixel;


      blanqueo: spblnk<=(hblnk and vblnk);
      direccion_vga: A<=("0" & cont_fil_mode(8 downto 0) & cont_col(7 downto 3))
+ ( SCROLL & "00000");

      --Salida VGA: se pueden cambiar depende del color de salida que se quiera
      RGB(4)<=pixreg(7) and spblnk;
      RGB(5)<='0';
      RGB(2)<=pixreg(7) and spblnk;
      RGB(3)<='0';
      RGB(0)<=pixreg(7) and spblnk;
      RGB(1)<='0';
      hsyncb<=hsync;
      vsyncb<=vsync;
end architecture arch_VGAcore;
```

Bibliografía

Recursos bibliográficos

[Gonzalez 92] José Adolfo González Vázquez. **Introducción a los microcontroladores. Hardware, Software, aplicaciones**. Mc Graw Hill, 1992

[Villar 98] Eugenio Villar. **VHDL. Lenguaje estándar de diseño electrónico**. McGraw-Hill, 1998.

[Yeralan Ahluwalia 95] Sencer Yeralan, Ashutosh Alhluwalia. **Programming and interfacing the 8051 microcontroller**. Addison-Wesley, 1995

Recursos electrónicos

[Bolton 01] Bolton Institute (2001, diciembre). **VHDL quick reference guide.** [en línea]. HTML disponible en: http://www.bolton.ac.uk/campus/elec/ami/courseware/adveda/vhdl/vhdlqr.html (Consultado en Septiembre 2002)

[Engdahl 00] Tomi Engdahl (2000). **VGA to workstation monitor FAQ** [en línea] HTML disponible en: http://documents.epanorama.net/documents/vga2rgb/index.html (consultado en Septiembre 2002)

[Redeya 01] Redeya Electrónica (2001). **Dispositivos lógicos programables**. [en línea]. HTML disponible en: http://eca.redeya.com/tutoriales/pld/pld.htm (Consultado en Septiembre 2002)

[UPV 02] Universidad del País Vasco (2002). **Lenguajes de descripción de hardware. VHDL**. [en línea]. HTML disponible en: http://det.bp.ehu.es/vhdl/pagina/inicio.htm (Consultado en Julio 2002)

[Winbond 99] Winbond Electronics (1999, Julio). **W78C32C 8-bit microcontroller.** [en línea]. PDF disponible en: http://www.winbond.com/PDF/sheet/w78c32c.pdf (Consultado en Septiembre 2002)

[Xess 01] Xess Corp (2001, Septiembre). **XS95 Board v1.3 user manual**. [en línea]. PDF disponible en: http://www.xess.com/manuals/xs95-manual-v1_3.pdf (Consultado en Septiembre 2002)

[Xilinx 98] Xilinx Corp (1998, diciembre). **XC95108 In-System Programmable CPLD.** HTML disponible en: http://direct.xilinx.com/partinfo/95108.pdf (Consultado en Septiembre 2002).