Proyecto de Ingeniería de Computadores (PEC)

Objetivo del curso

El objetivo de esta asignatura es que el alumno aprenda a desarrollar un prototipo de un computador o un SoC (System on Chip) en un chip programable sobre una placa base para crear un mini-ordenador. Se pondrán en práctica algunos de los conocimientos adquiridos en asignaturas anteriores sobre el diseño de la microarquitectura de un procesador, sobre el diseño e implementación de software de sistema, y sobre el diseño de sistemas digitales.

Fases principales del proyecto.

1) Aprendizaje de las herramientas de desarrollo para los chips programables (FPGA) y práctica del lenguaje de descripción del hardware VHDL.
2) Implementar pequeños componentes o dispositivos en el chip programable de la placa base.
3) Implementar una primera versión simplificada del procesador en una FPGA (sin memoria externa, ni soporte para el sistema operativo o dispositivos externos)
4) Implementar una versión completa del procesador.
5) Programar un sistema de arranque (BIOS) para el Sistema Operativo en el procesador.
6) Evaluar el rendimiento de varias aplicaciones sobre la plataforma que se ha diseñado.

Objetivo de las sesiones

En estas sesiones vamos a implementar físicamente un procesador sencillo en la FPGA. Lo haremos por etapas. En cada etapa cogeremos el trabajo realizado en la anterior y le añadiremos o modificaremos algún componente.

Etapa 6: Controladores PS2 y VGA.

En esta etapa añadiremos soporte para dos de los dispositivos más habituales: un teclado y una pantalla. Los controladores de estos dispositivos ya están implementados y simplemente hay que conectarlos adecuadamente a nuestro procesador. También se aprovechará esta etapa para crear juegos de prueba más complejos y se mostrará cómo se compilan y enlazan programas escritos en C para que se puedan ejecutar en nuestro procesador. De momento, estos dispositivos no funcionarán con interrupciones ya que el procesador aún no las soporta, pero se le añadirá más adelante. Primero añadiremos el controlador para un teclado PS2 y seguidamente el controlador para una pantalla VGA en modo texto.

Subetapa 6.1: Controlador de teclado PS2

En esta sección se realiza una breve descripción de los módulos que componen el controlador de teclado y su interfaz para poder ser utilizado por el SISA. En ningún momento se entrará en los detalles del funcionamiento del protocolo PS2. Para poder comprobar el controlador de teclado en la placa de desarrollo necesitareis un teclado PS2 o un teclado USB con un adaptador a PS2.

Normalmente un controlador de teclado funciona por interrupción y retorna un código de rastreo que indica la tecla que ha causado la interrupción y si ha sido pulsada o soltada. Por simplicidad, de momento este controlador funciona por encuesta y no funciona por interrupción. Además, retorna directamente la codificación en ASCII de la tecla pulsada y algunos 'ASCII' extras no estándar para teclas de control como los cursores o las teclas de función.

Módulos

El controlador de teclado está compuesto por dos módulos:

- **keyboard_controller**: Es el modulo que debe ser instanciado en el SISA y hace las veces de interfaz con el otro módulo, simplificando y adaptando el controlador en sí a nuestras necesidades actuales. Este módulo se encuentra en el fichero *keyboard_controller.vhd*.
- **ps2_keyboard_interface**: Este módulo es el controlador propiamente dicho. Otorga la posibilidad de devolver el código de rastreo, además de la codificación ASCII, para futuras ampliaciones. Este módulo se halla en el fichero *ps2_keyboard.vhd*.

Interfaz

La interfaz con el SISA es bastante sencilla y directa. A continuación se realiza una descripción de esta.

```
entity keyboard_controller is
    port (clk        : in    STD_LOGIC;
          reset      : in    STD_LOGIC;
          ps2_clk    : inout STD_LOGIC;
          ps2_data   : inout STD_LOGIC;
          read_char  : out   STD_LOGIC_VECTOR (7 downto 0);
          clear_char : in    STD_LOGIC;
          data_ready : out   STD_LOGIC);
end keyboard_controller;
```

Una a una, las señales mostradas son:

- **clk**: Señal de reloj del controlador. Para un correcto funcionamiento del controlador debe tener una frecuencia de 50 MHz.
- **reset**: Señal de *reset* del controlador. Debe ponerse a uno al inicio durante al menos un ciclo.
- **ps2_clk**: Señal de reloj del bus de comunicación *ps2* con el teclado. Esta señal debe ir conectada a los pines que conectan la señal de reloj del puerto ps2 (el pin PS2_CLK en el módulo sisa).
- **ps2_data**: Bus de comunicación serie de datos con el teclado. Utiliza la señal de reloj ps2_clk para mantener la sincronización. Se debe conectar a los pines que conectan la señal de datos del puerto ps2 (el pin PS2_DAT en el módulo sisa).
- **read_char**: Indica el código ASCII de la última tecla pulsada en el teclado.
- **clear_char**: *Acknoledge* de la señal que informa de que una tecla nueva ha sido pulsada. Si se desea conectar el teclado para trabajar por encuesta, se debe hacer una instrucción *out* sobre el puerto al que esté conectada esta señal para indicar que la tecla ya ha sido leída. ~~Si se conecta con interrupciones, la señal *interrupt acknoledge* debe ser conectada aquí~~.
- **data_ready**: Señal que indica la presencia de una nueva tecla pulsada. Cuando vale 1 indica que se ha pulsado una tecla desde la última vez. El controlador de teclado no tiene implementado ningún tipo de *buffer*, así que si la tecla no se lee y se pulsa otra la primera se pierde. Para usar el controlador por encuesta, se debe hacer una instrucción *in* sobre el puerto al que esté conectada esta señal para saber si hay una tecla nueva disponible. ~~Por interrupción, esta señal hace las veces de *interrupt*~~.

Programar el controlador por encuesta (polling)

Para utilizar el controlador de teclado por encuesta es imprescindible que las interrupciones estén inhibidas o no implementadas (como es el caso) pues, de otro modo, pulsar una tecla generaría una interrupción a la que el SISA contestaría con un *acknoledge*, limpiando la señal que informa de una nueva tecla pulsada.

Cuando una nueva tecla es pulsada, un registro se activa. Para acceder a este registro se debe realizar una operación de entrada (*in*) sobre el puerto 16. Si la instrucción retorna 0, no hay una tecla nueva pulsada. Si retorna 1, se ha pulsado una tecla desde la última vez que se limpió el registro.

Una vez sabemos que hay una tecla nueva pulsada, debemos realizar otra operación de entrada sobre el puerto 15 para saber que tecla concreta es. A continuación, es importante limpiar el registro que indica que hay una tecla nueva. Para ello basta con hacer una instrucción *out* sobre el puerto 16, sin importar el valor del registro con el que hagamos la operación.

A continuación se muestra un ejemplo de código por encuesta:

```
polling: in  r1, 16       ;leemos estado teclado
         bnz r1, polling
         ;si se ha pulsado una nueva tecla…
         in  r1, 15       ;leemos la tecla
         out 16, r0       ;hacemos clear del teclado. Cualquier registro vale, ya que
                          ;el controlador no tiene en cuenta el valor recibido
         ;el registro r1 contiene el código ASCII de la tecla pulsada
```

Acceso al teclado a través de puertos de Entrada/salida:

El controlador de teclado solo es accesible a través de puertos. Los puertos asignados en el espacio de entrada/salida son el 15 para leer el carácter ASCII pulsado y el 16 como registro de status para saber si se ha pulsado una tecla y para indicarle al controlador de teclado que ya se ha consultado que tecla es la pulsada.

Puertos de entrada:

- El puerto 16 contiene la señal de aviso de tecla pulsada. Solo para encuesta. (ej: in r0, 16)
- El puerto 15 contiene el código ASCII de la tecla pulsada. (ej: in r1, 15)

Puertos de salida:

- El puerto 16 se utiliza para limpiar la señal de aviso de tecla pulsada. Solo para encuesta. (ej: out 16, r0)

Conexión de los módulos:

Para conectar el controlador de teclado es necesario realizar algunas modificaciones en nuestro SoC (System On Chip).

Deberemos incluir los módulos *keyboard_controller.vhd* y *ps2_keyboard.vhd* en nuestro proyecto. Como el controlador de teclado sólo es usado a través de puertos haremos que el componente *keyboard_controller* sea importado e instanciado por el controlador de los puertos de entrada/salida (*controladores_IO*). Además, debemos modificar la gestión de los puertos para permitir que al leer el puerto 15 devuelva el valor del bus *read_char* del controlador de teclado en los 8 bits de menor peso. También debemos permitir que al leer el puerto 16 devuelva el valor de la señal *data_ready* del controlador de teclado en el bit de menor peso y que si se escribe en este mismo puerto (sin importar el valor que se escriba) ponga la señal *clear_char* a 1 durante al menos un ciclo para indicar al controlador que ya se ha leído la tecla pulsada.

Finalmente deberemos modificar la cabecera de la entidad *controladores_IO* para que ahora incluya las dos señales adicionales de comunicación necesarias para el control del teclado PS2.

```
ps2_clk  : inout std_logic;
ps2_data : inout std_logic;
```

El valor de estas dos señales simplemente pasa a través del controlador de entrada/salida para llegar al controlador de teclado que es el módulo que realmente las necesita.

Por último, hay que modificar la entidad sisa para que reciba las nuevas señales externas de la placa provenientes del conector del teclado (PS2_CLK y PS2_DAT) y las conecte con las nuevas señales del controlador de entrada/salida (*ps2_clk* y *ps2_data*).

Juegos de prueba

En el directorio de juegos de prueba hay un directorio llamado "0-ASCII-visores" que contiene un test para que podáis probar el funcionamiento de vuestra implementación. Este test simplemente muestra por los dos visores 7 segmentos de más a la derecha (HEX1-HEX0) el carácter *pseudoASCII* que devuelve el controlador de teclado PS2 correspondiente a la tecla pulsada. Por los dos visores 7 segmentos de la izquierda (HEX3-HEX2) se muestra el número de veces consecutivas que se ha pulsado esa tecla.

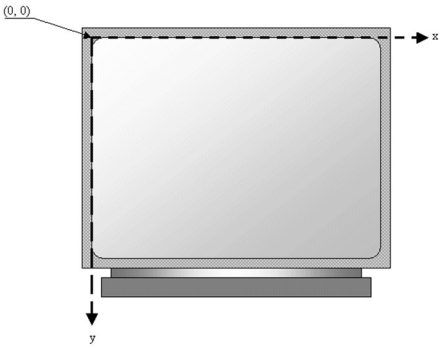
En el directorio hay un Shell Script para Linux (*generarTodo.sh*) que a partir del fichero fuente en ensamblador (*test_keyboard.s*) genera todos los ficheros necesarios para poder ejecutar el programa en el ModelSim, en la placa de desarrollo DE1 y en la placa DE2-115; y un fichero con el código desensamblado con alguna información extra (*test_keyboard.dis*). Para ello usa los compiladores/ensambladores de las herramientas SISA y un script el perl (*limpiar.pl*) que os pueden ser útiles para facilitaros la faena cuando creéis vuestros juegos de prueba.

Subetapa 6.2: Controlador VGA para el SISA

En este documento se muestra una breve descripción sobre la interfaz y el uso del controlador de VGA utilizado para permitir la conexión de la FPGA a un monitor. En ningún momento se entrará en los detalles del funcionamiento del protocolo de comunicaciones de un monitor VGA. Para poder comprobar el controlador de VGA en la placa de desarrollo necesitareis un monitor con entrada VGA o un monitor con entrada DVI y un adaptador VGA-DVI.

Disponemos de dos controladores VGA distintos pero parecidos en su funcionamiento y completamente intercambiables sin realizar cambios en el código que los utilice. Ambos controladores tienen la misma interface, pero alguno de los dos se adapta mejor que el otro a determinados monitores más sensibles a las frecuencias de rastreo y sincronización (la imagen es más estable y no tiembla).

Este primer controlador de VGA (VGA1) solo imprime caracteres basándose en su posición en la memoria de 8 KB que contiene, mapeándola en una cuadrícula de 80 columnas por 30 filas. El vértice superior izquierdo de esta cuadrícula corresponde a la fila 0 y columna 0. De estos 8 KB (8192 bytes) de memoria solo 4800 bytes son utilizados, 2400 para indicar un carácter y 2400 para indicar el color del carácter. El resto de memoria puede ser utilizada a conveniencia como memoria adicional. El segundo controlador de VGA (VGA2) se diferencia que la cuadrícula de caracteres es de 80 columnas por 40 filas de modo que utiliza los primeros 6400 bytes disponibles. Además, este segundo controlador dispone de la posibilidad de mostrar un cursor por pantalla para facilitar la escritura. Este está pensado para ser controlado mediante una operación de salida (ver apartado "Programar la VGA").



Ambos controladores utilizan la resolución estándar de la VGA (640x480 pixeles) de modo que cada carácter está definido por una matriz de 8x16 pixeles en el controlador VGA1 y por una matriz de 8x12 pixeles en el controlador VGA2. Ambos controladores también difieren un poco en los caracteres que muestran por pantalla cuando el valor ASCII corresponde a algún símbolo especial.

Para empezar se recomienda usar el controlador VGA1 que es más simple. Ya que el controlador VGA2 está todavía un poco más "inmaduro" en el momento de escribir esta documentación.

Módulos

El controlador de VGA1 se compone de los siguientes módulos:

- **vga_controller**: Encapsula el resto de módulos así como adapta sus interfaces para ser compatibles con el SISA. Se halla en el fichero *vga_controller.vhd*.
- **vga_ram_dual**: Contiene una memoria RAM síncrona de 8 KB, direccionados a tamaño word (16 bits), un puerto de lectura y otro de escritura con los que puede acceder de forma simultanea para leer y escribir un dato. Esta memoria es escaneada continuamente por el controlador e imprimiendo por pantalla su contenido. Se encuentra en el fichero *vga_ram_dual.vhd*.
- **vga_font_rom**: Instancia una memoria ROM síncrona de 4 KB, direccionados a tamaño byte. Esta memoria se inicializa con las fuentes de texto utilizadas para escribir por pantalla. Se halla en el fichero *vga_font_rom.vhd*.
- **vga_sync**: Este módulo, incluido en el fichero *vga_sync.vhd*, contiene el controlador de VGA propiamente dicho.

El controlador de VGA2 se compone de los siguientes módulos:

- **vga_controller**: Encapsula el resto de módulos así como adapta sus interfaces para ser compatibles con el SISA. Se halla en el fichero *vga_controller.vhd*.
- **vga_ram**: Contiene una memoria RAM síncrona de 8 KB, direccionados a tamaño word (16 bits) y un puerto de escritura. Esta memoria es escaneada continuamente por el controlador e imprimiendo por pantalla su contenido. Se encuentra en el fichero *vga_ram.vhd*.
- **vga_font_rom**: Instancia una memoria ROM síncrona de 4 KB, direccionados a tamaño byte. Esta memoria se inicializa con las fuentes de texto utilizadas para escribir por pantalla. Se halla en el fichero *vga_font_rom.vhd*.
- **vga_80x40**: Este módulo, incluido en el fichero *vga_80x40.vhd*, contiene el controlador de VGA propiamente dicho. En él se hallan varios módulos más que lo componen y que no serán descritos, pero todos contenidos en el mismo fichero.

Interfaz

Se ofrece a continuación una breve descripción de la interfaz correspondiente a ambos controladores de VGA, a modo de indicación sobre como instanciarlo. La interfaz es la siguiente:

```
entity vga_controller is
    port(clk_50mhz      : in  std_logic;
         reset          : in  std_logic;
         blank_out      : out std_logic;
         csync_out      : out std_logic;
         horiz_sync_out : out std_logic;
         vert_sync_out  : out std_logic;
         red_out        : out std_logic_vector(7 downto 0);
         green_out      : out std_logic_vector(7 downto 0);
         blue_out       : out std_logic_vector(7 downto 0);
         --
         addr_vga          : in  std_logic_vector(12 downto 0);
         we                : in  std_logic;
         wr_data           : in  std_logic_vector(15 downto 0);
         rd_data           : out std_logic_vector(15 downto 0);
         byte_m            : in  std_logic;
         vga_cursor        : in  std_logic_vector(15 downto 0);
         vga_cursor_enable : in  std_logic);
 end vga_controller;
```

En primer lugar, las señales *blank_out*, *csync_out*, *horiz_sync_out*, *vert_sync_out*, *red_out*, *green_out* y *blue_out* son las que se comunican directamente con el monitor. Son las señales que vienen y salen del conector VGA y se deben conectar directamente a los pines de la FPGA que las gobiernan.

A modo de breve resumen, estas señales son:

- **red_out**, **green_out** y **blue_out**: Las componentes *RED*, *GREEN* y *BLUE* del pixel que actualmente se está imprimiendo en pantalla. Estas señales están compuestas por 8 bits cada una y son convertidas a una señal analógica proporcional al valor binario de estos bits mediante un conversor digital-analógico (DAC) incorporado en la FGPA.
- **horiz_sync_out** y **vert_sync_out**: Sincronización horizontal y vertical de la pantalla. Marca las pautas de cambio de línea y columna al rasterizar la pantalla.
- **blank_out**: Señal de blanqueo del monitor. Mientras esta señal esta activa el monitor no imprime por pantalla. Se activa y desactiva periódicamente.
- **csync_out**: Señal de reloj de la VGA, necesaria para sincronizar correctamente la pantalla con la FPGA. Este reloj debe tener una frecuencia de 25MHz para funcionar correctamente con este controlador.

Estas dos últimas señales de control (*blank_out* y *csync_out*) no siempre son necesarias y dependen del hardware que este implementado la placa de desarrollo. Por ejemplo, no son necesarias para la placa de desarrollo DE1 (y no tienen ningún pin conectado a la FPGA) y si lo son para la placa DE2-115.

El resto de señales se usan para manejar el controlador en sí. A continuación se describen brevemente:

- **clk_50mhz**: Señal de reloj del controlador. Esta debe ser de 50 MHz para el funcionamiento correcto del controlador.
- **reset**: Señal de reset del controlador.
- **addr_vga**: Dirección de la memoria RAM en a la que se desea acceder. A pesar de que la memoria RAM se direcciona a nivel de word, el direccionamiento al controlador de VGA se realiza a tamaño de byte, al igual que el módulo controlador de memoria.
- **we**: Write enable para la memoria RAM.
- **wr_data**: Bus de datos de escritura para la memoria RAM.
- **rd_data**: Bus de datos de lectura de la memoria RAM.
- **byte_m**: Indica si el acceso a la memoria RAM es a nivel de byte (*byte_m*=1) o word (*byte_m*=0).
- **vga_cursor**: Indica la posición del cursor en la pantalla. Esta señal solo es útil para el controlador VGA2, el controlador VGA1 simplemente la ignora. Puesto que la pantalla se divide en una cuadrícula de 80x40, para determinar su posición horizontal se utiliza el valor de esta señal en módulo 80. La división entre 80 indica la fila. Por ejemplo, si *vga_cursor* vale 84 el cursor se situará en la columna 4, fila 1.
- **vga_cursor_enable**: Si vale 1 indica que el valor contenido en el vector de bits *vga_cursor* es válido y la posición del cursor es actualizada. Al igual que la señal anterior solo es útil para el controlador VGA2.

Programar la VGA

El controlador de VGA realiza todas las operaciones necesarias para imprimir en pantalla la memoria contenida en su RAM. Esta memoria RAM es direccionada por el controlador de memoria desde las posiciones 0xA000 hasta 0xBFFF. El acceso a esta memoria es exactamente igual que como se hace con la memoria principal, con las instrucciones de load y store.

Para escribir un carácter basta con poner su valor en código ASCII en la posición adecuada de la memoria. El mapeo que el controlador realiza entre la memoria y la pantalla es el siguiente:

- La primera dirección (0xA000) se corresponde con la primera posición en la cuadrícula de caracteres de 80x30 que produce como salida el controlador VGA1.

- Cada carácter está compuesto por sus 8 bits en código ASCII, así como otros 8 bits indicando el color en formato RGB. Por lo tanto, la segunda posición de la cuadrícula de caracteres le corresponde la dirección 0xA002, a continuación 0xA004 y así hasta las 2400 (80*30) direcciones que el controlador VGA1 mapea en pantalla. Las direcciones restantes son ignoradas ~~y dependiendo del controlador de memoria principal implementado podrían ser usadas como memoria adicional~~.

Coordenadas (x,y)
@memoria

(0,0) 0xA000	**(0,1)** 0xA002	**(0,2)** 0xA004	• • •	**(0,79)** 0xA09E
(1,0) 0xA0A0	**(1,1)** 0xA0A2	**(1,2)** 0xA0A4	• • •	**(1,79)** 0xA13E
(2,0) 0xA140	**(2,1)** 0xA142	**(2,2)** 0xA144	• • •	**(2,79)** 0xA1DE
•	•	•		•
•	•	•		•
•	•	•		•
(29,0) 0xB220	**(29,1)** 0xB222	**(29,2)** 0xB224	• • •	**(29,79)** 0xB2BE

- De los 8 bits de color, solo los 6 de menor peso son tenidos en cuenta. Los 2 bits de menor peso indican la graduación de rojo (R), los 2 siguientes la de verde (G) y los otros dos la de azul (B) quedando el siguiente formato para los 8 bits de color "xxBBGGRR". Así, si los 8 bits de color valen "xx111111" tenemos un 100% de rojo, 100% de verde y un 100% de azul dando como resultado que el carácter se mostrará por pantalla en color blanco. Si los 8 bits de color valen "xx000000" corresponde al color negro y si por ejemplo valen "xx000110" corresponde al color resultante de mezclar 66% de color rojo (R=10), un 33% de color verde (G=01) y un 0% de color azul (B=00).

Por último, si usamos el controlador VGA2 podemos controlar el cursor. Para controlar el cursor del monitor se debe realizar una operación de salida (*out*) sobre el puerto 11, con el valor de la posición del cursor que se desea. Esta posición se calcula multiplicando la fila de la cuadrícula donde deseamos poner el cursor por 80 y sumándole la columna. Y es el controlador el que recibe este valor y para obtener la fila divide entre 80 el valor quedándose con la parte entera y la columna se calcula con el módulo 80. Para controlar la visualización del cursor usaremos el puerto 12. El bit de menor peso del puerto indica si el cursor esta visible (bit a 1) o no (bit a 0). Inicialmente el cursor deberá ser visible.

Conexión de los módulos

Para conectar el controlador de VGA1 es necesario realizar algunas modificaciones en nuestro SoC (System On Chip).

Deberemos incluir los módulos *vga_controller.vhd*, *vga_ram_dual.vhd*, *vga_ram_dual.vhd* y *vga_sync.vhd* en nuestro proyecto.

Lo primero que debemos decidir es donde vamos ubicar este nuevo componente. Como es un dispositivo que se gestiona como si fuese la memoria principal podría parecernos adecuado instanciarlo dentro del controlador de memoria. Pero este dispositivo también se gestiona a través de puertos de entrada/salida como en el caso del cursor, de modo que también podría parecer interesante instanciarlo desde dentro del controlador de entrada/salida. Para la explicación siguiente sobre las conexiones no se ha tomado ni la primera solución ni la segunda. Se ha optado por instanciar este componente al mismo nivel que el procesador, el controlador de memoria y el controlador de entrada/salida. Por tanto será la entidad sisa quien instanciará a este componente VGA1 y lo conectará adecuadamente con el controlador de memoria, el controlador de entrada/salida y las señales externas de control de la VGA.

Tenemos que modificar ligeramente el controlador de memoria. Este controlador recibe la dirección que genera el procesador, el modo de acceso y el tipo de acceso y se encargará de generar las señales adecuadas para ser enviadas al

controlador de VGA. Por tanto, debemos añadir las siguientes señales a la cabecera de la entidad *MemoryController* para que ahora incluya las señales de comunicación necesarias para el control de la VGA.

```
vga_addr    : out std_logic_vector(12 downto 0);
vga_we      : out std_logic;
vga_wr_data : out std_logic_vector(15 downto 0);
vga_rd_data : in  std_logic_vector(15 downto 0);
vga_byte_m  : out std_logic
```

El valor de la señal **vga_addr** corresponde la posición de la cuadrícula de 8KB de la memoria interna del controlador de VGA. La memoria del controlador VGA empieza en la dirección 0x0000 y por tanto esa es la ubicación del primer carácter de la cuadricula. Puesto que en nuestro sistema hemos decidido que la memoria de video estaría mapeada a partir de la dirección 0xA000 el controlador de memoria debe hacer esta traslación. De modo que a la posición 0xA000 de la memoria del sistema le corresponde la posición 0x0000 de la memoria de video interna del controlador de VGA, a la posición 0xA002 le corresponde la posición 0x0002, y así sucesivamente.

El valor de la señal **vga_we** corresponde al permiso de escritura de la memoria de la VGA. Solo debe valer 1 cuando se esté haciendo una escritura de memoria y esta corresponda a las posiciones de la memoria de video (entre la 0xA000 y la 0xBFFF).

El valor de las señales **vga_wr_data** y **vga_rd_data** corresponden respectivamente al dato que enviamos o recibimos de la memoria interna del controlador de la VGA. Y el valor de la señal **vga_byte_m** corresponde al modo de acceso que hacemos en casos de que se trate de un acceso a nivel de *byte* o a nivel de *word*.

También deberemos modificar la cabecera de la entidad *controladores_IO* para que ahora incluya las dos señales para la gestión del cursor.

```
vga_cursor        : out std_logic_vector(15 downto 0);
vga_cursor_enable : out std_logic;
```

Si usamos el controlador VGA1, simplemente el controlador de entrada/salida puede poner en ellas cualquier valor ya que el controlador de VGA directamente ignora estas señales. Si usamos el controlador VGA2 podemos hacer lo mismo (poniendo un 0 en la señal **vga_cursor_enable** para que no se muestre el cursor) o permitir el control del cursor. Para permitir el control del cursor deberemos permitir que se escriba en el puerto 11 para indica la posición del cursor en la pantalla y en el bit de menor peso del puerto 12 un valor para indicar si queremos activar o desactivar el cursor.

Por último, hay que modificar la entidad sisa para que instancie al componente *vga_controller* y para reciba las nuevas señales externas de la placa de desarrollo provenientes del conector de VGA (VGA_R, VGA_G, VGA_B, VGA_HS y VGA_VS en el caso de la placa DE1) y las conecte con las señales del controlador de VGA.

A las señales **clk_50mhz** y **reset** del componente *vga_controller* le pasaremos el reloj de 50 Mhz del sistema y la señal de *reset* del sistema. A las señales **addr_vga**, **we**, **wr_data**, **rd_data** y **byte_m** le pasaremos los valores que nos envía el controlador de memoria. Y a las señales **vga_cursor** y **vga_cursor_enable** le pasaremos los valores que nos envía el controlador de entrada/salida.

Finalmente hay que conectar las señales externas de la placa provenientes del conector VGA con las señales de salida del controlador de VGA. En el caso de la placa DE1, a las señales VGA_HS y VGA_VS le conectaremos respectivamente las señales **horiz_sync_out** y **vert_sync_out** provenientes del controlador de VGA. Para las señales VGA_R, VGA_G y VGA_B hay que hacer una pequeña conversión. Estas señales codifican el color RGB en 4 bits y los buses de salida del controlador de VGA lo codifican en 8 bits. Simplemente hay que escoger los 4 bits de menor peso para cada señal.

Juegos de prueba

En el directorio de juegos de prueba hay siete juegos de prueba distintos. Los tres primeros están escritos exclusivamente en lenguaje ensamblador y los 4 restantes en C y ensamblador.

El primer juego de prueba, que está en el directorio "**1-VGA_mini**", simplemente muestra algunos caracteres por la pantalla en diversos colores usando las instrucciones STB y ST indistintamente. El segundo juego de prueba, que está en el directorio "**2-VGA-color**", rellena completamente la pantalla de caracteres (80x30). Muestra el repertorio completo de los 256 caracteres ASCII posibles en varios colores (de los 64 disponibles). El resultado de este juego de prueba es similar al que vemos en la siguiente imagen.

El tercer juego de prueba, que está en el directorio "**3-echo_keyboard**", muestra por la pantalla de caracteres (80x30) cada uno de los caracteres que se van pulsando en el teclado PS2. También muestra por los visores 7 segmentos el valor de la posición actual del cursor y en caso de que el driver de pantalla tenga implementado el cursor y se gestione en los controladores de entrada/salida lo muestra activo y en su posición correcta.

El resto de juegos de pruebas están escritos en C. Por tanto, primero hay que ver como se compilan y enlazan (*link*) los ficheros. Como se le indica al compilador de C las particularidades de nuestro sistema, como que la dirección de inicio es la 0xC000 y que tiene que reubicar el código a partir de esta dirección. Si vamos a compilar código en C necesitaremos disponer de una pila para poder pasar los parámetros en las llamadas a subrutinas. También deberemos ver como accede el compilador de C a la pila y como devuelve los resultados de las subrutinas.

El compilador de C **sisa-gcc** usa el registro r7 como puntero a la pila y por tanto espera que esté inicializado correctamente antes de llamar a cualquier subrutina. La dirección de retorno una la subrutina debe estar en el registro r5 antes de llamarla y el retorno de los resultados de una subrutina se hace a través del registro r1.

Necesitamos un código en ensamblador que se encargue de la inicialización. Este código se ejecutará a partir de la dirección 0xC000 que es la posición de la primera instrucción que ejecuta el procesador al arrancar. Esta área de memoria corresponde a la memoria Flash/ROM (aunque hemos usado la SRAM en el proyecto) y por tanto no olvidemos que no se puede escribir datos en ella en tiempo de ejecución. Antes de hacer ninguna llamada a subrutina debemos configurar la

pila. Esta debe estar colocada en un área de memoria en la que sí se puedan hacer escrituras. Podemos escoger cualquier dirección, pero nosotros hemos escogido reservar los primeros 512 bytes a partir de la posición 0x8000, es decir de la 0x8000 hasta la 0x81FF. El primer valor de la pila se escribirá en la posición más alta y el puntero a la pila irá decreciendo.

El código ensamblador que se encarga de esta inicialización es el que podéis encontrar en el fichero "*entrada.s*". Este código inicializa el registro r7 con el valor correspondiente a la primera posición de la pila, inicializa el registro r5 con la dirección de una rutina que contiene la tarea a realizar después de ejecutar el programa en C (esta rutina simplemente hace un *halt*) y salta a la rutina principal, en C la rutina *main()*. Este código de inicialización es el que debe ubicarse en la posición 0xC000.

Cuando ensamblamos un código o compilamos un módulo de código en C, el código objeto generado es siempre a partir de la dirección 0x0000. Esto afecta básicamente a los saltos absolutos (las instrucciones JMP y JAL) ya que las direcciones destino de salto dependen de donde esté ubicado finalmente el código en la memoria. Así que debe ser el enlazador (*linker*) el que debe encargarse de reubicar el código a su posición definitiva para generar un único ejecutable y ajustar todas las direcciones de saltos absolutos a su valor correcto. Para ello, se le indica al enlazador de código sisa (**sisa-ld**) como debe hacerlo mediante un fichero de configuración llamado **system.lds** que está en cada juego de pruebas. Este fichero de configuración le indica al enlazador que el código y los datos estáticos (de sólo lectura) deben estar ubicados a partir de la posición 0xC000 y alineados a posiciones pares de memoria y que la zona de datos dinámicos, si existe, estará ubicada a partir de la dirección 0x8000.

Veamos los pasos que hay que realizar para poder generar el fichero a cargar en la memoria de la placa de desarrollo. A modo de ejemplo vamos a escoger el cuarto juego de pruebas "**4-HelloWorld**". Este juego de prueba simplemente muestra por pantalla la frase "Hello World" donde cada carácter está en un color distinto. En la siguiente secuencia de comandos vemos las tareas a realizar.

```
$> sisa-as entrada.s -o entrada.o
$> sisa-gcc -c HelloWorld.c -o HelloWorld.o
$> sisa-ld -s -T system.lds entrada.o HelloWorld.o -o temp_HelloWorld.o
$> sisa-objdump -d temp_HelloWorld.o >HelloWorld.code
$> limpiar.pl codigo HelloWorld.code
```

El primer comando se encarga de ensamblar el código de inicialización. El segundo comando se encarga de compilar el código en C donde hay la rutina principal *main()*. El tercer comando enlaza los dos ficheros anteriores y reubica el código a su posición definitiva. El cuarto comando simplemente desensambla el fichero generado por el enlazador para poder obtener los códigos de las instrucciones sisa. El último comando es una utilidad que a partir del código desensamblado genera todos los ficheros para las placas de desarrollo con los formatos de fichero adecuados. En cada juego de pruebas hay un *script* que hace todos estos pasos automáticamente (*generarTodo.sh*). Para probar este juego de pruebas simplemente hay que grabar el fichero *.hex* adecuado en la posición 0x6000 de la memoria SRAM de la placa, como hemos hecho hasta ahora.

El quinto juego de pruebas, que está en el directorio "**5-Helloworld2**", es muy parecido al anterior. Muestra por pantalla varios mensajes de texto. Estos mensajes de texto están almacenados en variables globales en C y por lo tanto el enlazador deberá ubicarlos en una zona de memoria en la cual puedan ser modificados por si fuese el caso. Así que si ejecutamos el script que genera todos los ficheros veremos que aparece un fichero con la terminación "**.code.DE1.hex*" que corresponde al código máquina que debe ejecutar el procesador y por tanto debe estar ubicado en la posición de memoria 0xC000 (0x6000 en la SRAM de la placa). También se genera otro fichero, con la terminación "**.data.DE1.hex*", que corresponde a la sección de datos del programa y que debe estar ubicado en la posición de memoria 0x8000 (0x4000 en la SRAM de la placa) tal y como se le ha indicado al enlazador mediante el fichero *system.lds*.

Para estos juegos de pruebas sólo se han reservado 512 bytes para almacenar completamente la sección de datos y la pila. Cantidad de memoria adecuada para estos juegos de prueba pero probablemente insuficiente para programas más

complejos. Esto ya se modificará más adelante cuando implementemos el sistema operativo y tengamos secciones distintas para modo usuario y para modo sistema.

El sexto juego de pruebas, que está en el directorio "**6-MaquinaEscribir**" emula a una especie de máquina de escribir. Muestra por la pantalla los caracteres que se van pulsando por el teclado en el color seleccionado encada momento. Para este juego de prueba ya se ha creado una librería en C con las subrutinas más útiles para poder controlar los dispositivos y poder realizar nuestros propios juegos de prueba. Esta librería se encuentra en los ficheros *lib_sisa.c* y *lib_sisa.h*.

Por último, tenemos como juego de pruebas una adaptación del juego de la serpiente, famoso por estar presente en todos los primeros móviles de Nokia. El juego está en el directorio "**7-Snake**". Este juego es una adaptación de una versión libre disponible en internet. Este juego original está implementado usando la librería "*ncurses*". La librería *Ncurses* es una biblioteca de programación que provee una API que permite al programador escribir interfaces basadas en texto. Hemos creado una versión especial de la librería *lib_sisa* para este juego que emule/simule las llamadas que el juego hacia a las *ncurses* originales. También se ha tenido que adaptar un poco el juego con mayor o menor fortuna (se ha eliminado el acceso a disco para almacenar los records). Por la forma en cómo funcionan las *ncurses* originales y que nuestras rutinas trabajan directamente en la memoria de video en vez de en un *framebuffer* como en la librería original, se produce un pequeño parpadeo en el refresco de la pantalla, aunque no es molesto.

Para que este juego funcione, además hay que hacer alguna ligera modificación a nuestro procesador. La implementación de este juego necesita tener un sistema de referencia de tiempo para poder determinar a la velocidad con la que se mueve la serpiente por la pantalla y necesita un generador de números aleatorios para determinar la posición de pantalla aleatoria donde aparecerá la siguiente "fruta". El programa en C llama a las rutinas *usleep(int usec)* y *rand()*. Para poder implementar estas subrutinas en la librería *lib_sisa* necesitaremos un poco de soporte hardware por parte del procesador.

Primero implementaremos en el módulo de los controladores de entrada/salida un contador de milisegundos en el puerto 21. Cada vez que escribamos un valor en este puerto, el valor se irá decrementando en una unidad cada milisegundo hasta llegar a cero. Así que si queremos dejar pasar 12 milisegundos lo único que necesitamos es escribir el valor 12 en ese puerto e ir consultándolo frecuentemente. Cuando su valor sea cero habrán pasado los 12 milisegundos. Para contar un milisegundo necesitaremos un contador de ciclos. A 50MHz necesitaremos contar hasta 50000 ciclos para que sea un milisegundo. El siguiente fragmento de código VHDL es un posible ejemplo de cómo podría ser este fragmento de código.

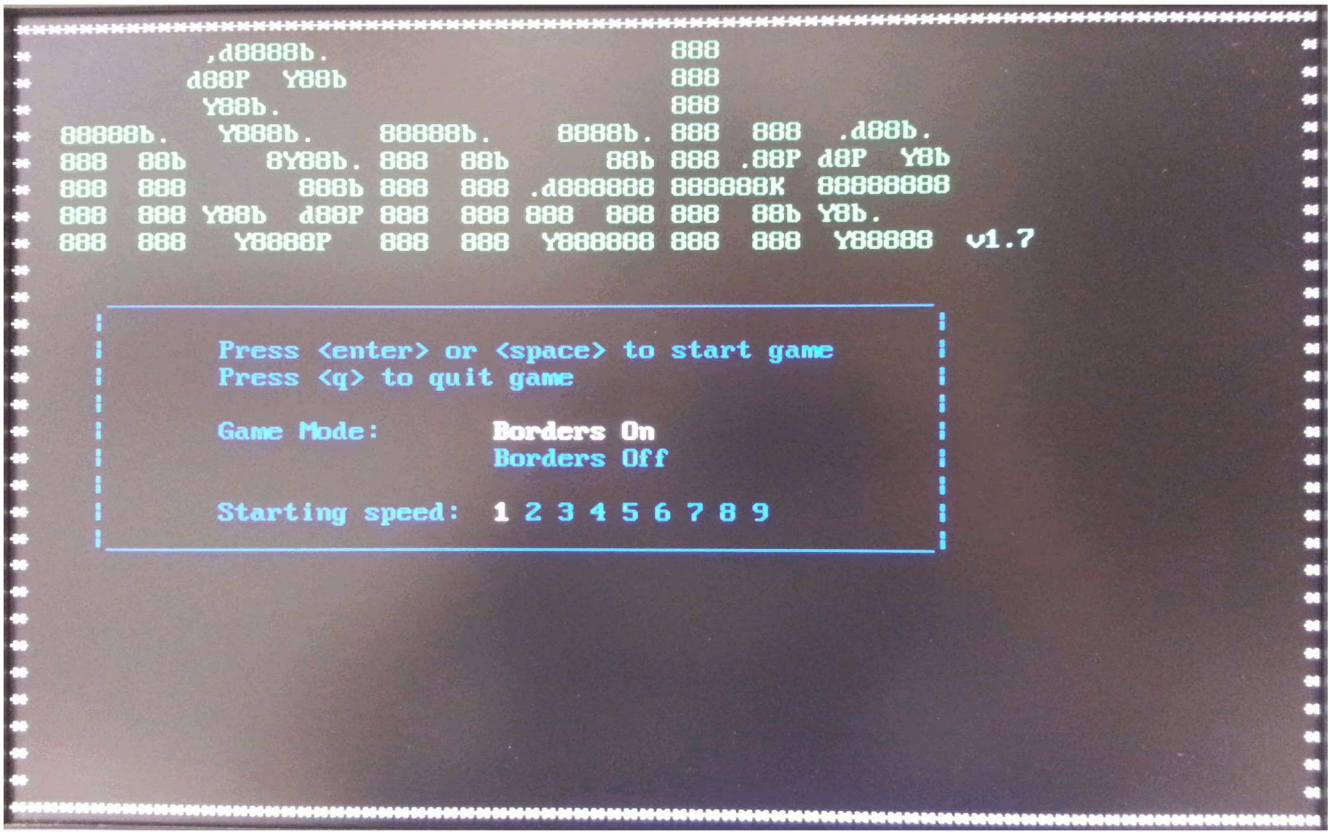
```
signal contador_ciclos       : STD_LOGIC_VECTOR(15 downto 0):=x"0000";
signal contador_milisegundos : STD_LOGIC_VECTOR(15 downto 0):=x"0000";

begin
   process(CLOCK_50)
   begin
      if rising_edge(CLOCK_50) then
         if contador_ciclos=0 then
            contador_ciclos<=x"C350";   -- tiempo de ciclo=20ns(50Mhz) 1ms=50000ciclos
            if contador_milisegundos>0 then
               contador_milisegundos <= contador_milisegundos-1;
            end if;
         else
            contador_ciclos <= contador_ciclos-1;
         end if;
      end if;
   end process;
```

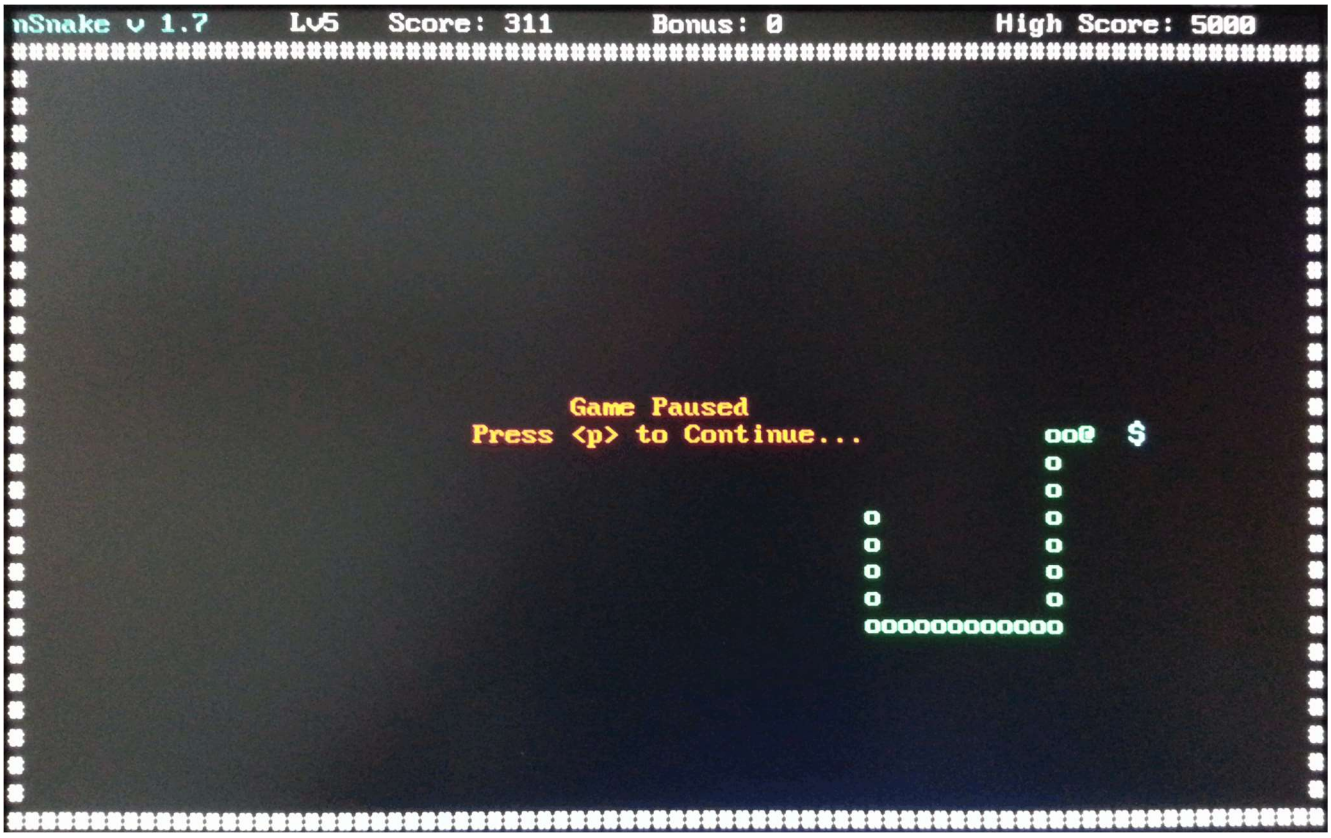
Ahora simplemente hace falta conectar el valor de la señal *contador_milisegundos* al puerto 21 permitiendo el acceso para lectura y escritura.

Para generar el número aleatorio no nos complicaremos demasiado (ni garantizaremos la equiprobabilidad, ni la aleatoriedad ni nada). Simplemente usaremos la señal *contador_ciclos* como valor aleatorio. Para ello conectaremos el valor de esa señal directamente al puerto 20 permitiendo únicamente lecturas sobre este puerto.

Si todo ha ido bien, una vez ejecutemos el juego deberían aparecer las siguientes pantallas:



Pantalla 1: Menú de selección del juego



Pantalla 2: Un momento de una partida

Pantalla 3: Final de una partida