SISTEMAS INFORMÁTICOS 2001

ARQUITECTURA BASADA EN MICROPROCESADOR

DESCRIPCIÓN DE LA ARQUITECTURA Y MANUAL DE PROGRAMACIÓN

JULIO ESTEBAN FERNÁNDEZ JUÁREZ
ANTONIO ORTIZ PINEDA
MIGUEL ANGEL ROJAS GÓMEZ

0 – BREVES NOTAS DEL AUTOR:

El presente manual no pretende explicar de manera detallada la arquitectura diseñada y desarrollada por nuestro grupo de Sistemas Informáticos ni tampoco el desarrollo de la aplicación software propuesta para probarla: el juego Galaxy Invader.

Nuestros objetivos con este manual son más sencillos pero a la vez más prácticos: es presentar de manera muy sencilla y siempre orientado al programador dicha arquitectura mediante sus componentes básicas y unas breves explicaciones sobre cada una de ellas, así como los problemas encontrados para conseguir los objetivos propuestos. Por otra parte se darán pautas al potencial programador de nuestra arquitectura para poder realizar aplicaciones de manera rápida y avisando sobre las restricciones que existen para que dichos desarrollos queden exentos de errores.

El manual se divide en las siguientes secciones:

1 – Introducción : introduciremos la arquitectura desarrollada, los principales objetivos perseguidos de la misma y comentaremos brevemente os principales problemas encontrados para su implementación en el soporte elegido: placa FPGA extendida basada en la XC4010XL.

2 – Entorno experimental : aunque no es un objetivo prioritario de este manual, daremos unas pequeñas referencias sobre las herramientas software y hardware utilizadas para el desarrollo de la arquitectura y de la aplicación.

3 – Galaxy Invader : breve descripción de la aplicación desarrollada para probar la arquitectura. Será una descripción simplemente funcional sin entrar en los detalles de su implementación. De todos modos, algunas rutinas utilizadas en el desarrollo serán mostradas en posteriores secciones como referencia de cómo utilizar la arquitectura.

4 – Descripción de la arquitectura : esta sección explicará la arquitectura de manera más detallada, aunque sin entrar a un nivel lógico. Para cada componente que componen la arquitectura se

mostrará un esquema general, su modelo de programación (por encima), razones por las que se ha elegido ese diseño y no otros, problemas encontrados, etc ...

5 – Modelo de programación: esta es la sección más útil para los futuros programadores que quieran usar el trabajo desarrollado para la creación de nuevas aplicaciones. Se propondrán rutinas básicas como ejemplo de utilización de la arquitectura y se darán las restricciones que el programador ha de tener en cuenta para evitar posibles problemas.

1– INTRODUCCIÓN:

Los objetivos perseguidos con este desarrollo eran principalmente la creación de una arquitectura basada en el microcontrolador 8031 de la que dispone la placa FPGA extendida basada en la XC4010XL. A partir de este microcontrolador y un banco de memoria SRAM de 32 Kb también disponible en la placa, había que crear un controlador así como distintos periféricos como una VGA alfanumérica, un módulo PS/2 para teclado y un generador de sonido para el CODEC suministrado. Además, se desarrollo un módulo generador de números pseudo-aleatorios y un controlador de interrupciones.

En un principio, todo tenía que poder caber en la placa suministrada, pero aparte de los claros problemas por falta de espacio, eso replanteaba nuevos problemas debido a la interconexión física y los componentes de los que se contaba. El problema más importante, que estuvo a punto de replantear la separación de la arquitectura en dos placas distintas, la presentaba el hecho de que sólo existía un único módulo de memoria para almacenar el programa que debía ejecutar el microcontrolador y para almacenar la pantalla a refrescar por la VGA. Además, las líneas de acceso a la memoria estaban compartidas tanto por el micro como por la propia FPGA (VGA).

Había que buscar un mecanismo tanto para solapar la ejecución de instrucciones del 8031 con el continuo refresco de la pantalla, además de resolver el problema de las líneas compartidas (ya impuesto por la propia interconexión física de la placa). Mientras

el primer problema tenía fácil solución aprovechando los tiempos de "blanking" del refresco de la pantalla, el segundo dependía únicamente de las posibilidades que el microcontrolador disponía para poder aislar sus líneas de acceso a la memoria. Por fortuna, el micro suministrado en la versión 1.2. disponía de una capacidad para pararse y poner en alta impedancia sus líneas, con la capacidad de salir de ese estado mediante una interrupción. Más adelante, comentaremos, las consecuencias de estar obligado en disponer de ese apoyo software (dormir/despertar al microcontrolador).

Otros problemas encontrados fueron a falta de espacio en la FPGA, lo que obligó a realizar ciertos componentes del diseño utilizando capacidades "especiales" de la FPGA, como pudieran ser la memoria de LUTs (más adelante, explicaremos en que consiste eso). Además, la incapacidad de cambiar la frecuencia de reloj en la placa, nos obligó a replantear el diseño para poder reducir el tiempo de los caminos críticos (sobretodo en el acceso a la memoria por parte de la VGA). Por último, como un problema menor, la no interconexión de las líneas de interrupción al micro. Tuvimos que utilizar un cableado manual (por supuesto, obligatorio para que las aplicaciones funcionen sobre nuestra arquitectura) que más adelante detallaremos.

2– ENTORNO EXPERIMENTAL:

En esta sección comentaremos, de manera muy breve, el equipo utilizado, tanto H/W como S/W, utilizadas para el desarrollo tanto del la arquitectura como del juego Galaxy Invader.

2.1. – Placa extendida Xstend Board XS40 v1.2 basada en el chip FPGA XC4010XL. La placa sobre la que se ha realizado la arquitectura dispone de un microcontrolador 8031 y un banco de memoria SRAM de 32Kb, además, de salida VGA, CODEC de Sonido, puerto PS/2 para teclado y ratón, puerto paralelo, leds de barras y de 7 segmentos, Switches DIP, etc ...

2.2 – Xilinx Foundation F3.1i : paquete de herramientas que permiten a partir de un código en lenguaje de descripción hardware (código HDL), o esquemático donde se especifica un sistema H/W

de alto nivel (nivel puertas y biestables) realizar una síntesis e implementación física en una estructura de celdas que se elija (en nuestro caso XC4010XL). El paquete incorpora múltiples herramientas como editores de código HDL, editores esquemáticos, simuladores funcionales de tiempo, etc ...

Para nuestro desarrollo, hemos especificado la arquitectura en código HDL y hemos utilizado la herramienta para síntesis e implementación física para obtener el bit-stream final que es cargado en la FPGA, mediante el puerto paralelo.

2.3. – ModelSIM SS-EE 5.4 : esta herramienta dispone de un editor HDL y simulador funcional para los diseños realizados. Hemos utilizado esta herramienta para las simulaciones, en detrimento del simulador incorporado en el paquete de herramientas Xilinx Foundation, debido a sus mayores prestaciones sobretodo de velocidad.

2.4. – Utilidades XSTOOLS : pequeñas utilidades para cargar los diseños en la FPGA mediante el puerto paralelo (así como los programas realizados para nuestra arquitectura) utilizando Gxsload, o para estimular la entrada del puerto paralelo una vez cargado (en nuestro diseño, para resetear el sistema mediante el bit 0 de dicho puerto) utilizando Gxsport.

2.5. - Ensamblador cruzado ASM51 : ensamblador en el cual hemos realizado el juego Galaxy Invader. Dicho ensamblador soporta múltiples plataformas de la familia 8051.

2.6. – Emulador 8051 1.0. de TS-Controls : emulador para depurar aplicaciones desarrolladas en ensamblador 8051. Muy utilizado para el desarrollo del juego Galaxy Invader.

3– GALAXY INVADER:

En esta sección, comentaremos de manera muy breve el juego desarrollado que sirvió tanto para terminar de depurar la arquitectura como para demostrar sus posibilidades.

El juego desarrollado se basa en el género de juegos de marcianitos. El objetivo es obtener el máximo de puntos matando naves alienígenas, antes de que ellas acaben con todas tus vidas. Para ello, dispones de un número limitado de disparos con el que tendrás que acertar a dichas naves que irán apareciendo de manera aleatoria (tu situación es abajo y sólo se te permite moverte izquierda y derecha). Las naves enemigas son de dos tipos: ovnis, que no te disparan y que aparecerán de vez en la parte superior de la pantalla. Esta nave no te podrá matar, pero es la que más puntos te puede proporcionar. Por otra parte, las naves "caza" que pueden matarte de dos maneras: mediante sus disparos si te alcanzan o si te alcanzan ellos mismos. Dispones de 3 vidas, y 250 disparos, aunque puedes conseguir bonus. Existen 3 niveles de dificultad y capacidad de parar el juego.

Precediendo al juego, desarrollamos una pequeña presentación donde se toca una banda sonora (de la película Misión Imposible) y se hace gala de efectos especiales como el desplazamiento del marco de ventana de visualización y rotación de colores (y así poder demostrar las posibilidades de la arquitectura). Durante el juego, hay muchos detalles que el usuario debe tener en cuenta y que demuestran as enormes prestaciones de la arquitectura : sonidos en ciertos eventos (disparo, movimiento), rápida respuesta del teclado (lo que demuestra su fiabilidad), excelente calidad de imagen junto con exquisitos frames realizados por nuestro grafista, y además la aleatoriedad en la salida de los marcianos (gracias a un componente H/W creado específicamente : generador de número pseudo-aleatorios ALEA).

Las teclas del juego son las siguientes:

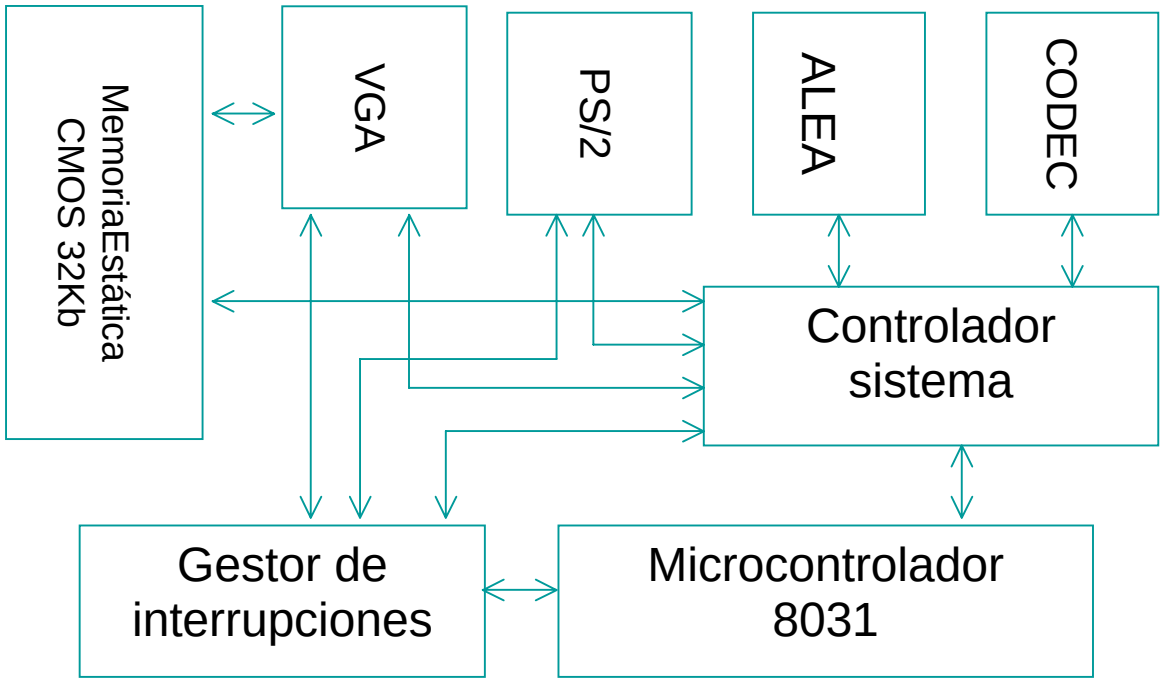
ESC → pausa
+ → incrementar nivel
- → decrementar nivel
O → moverse izquierda
P → moverse derecha
ESPACIO → disparar

4– DESCRIPCIÓN DE LA ARQUITECTURA:

El trabajo desarrollado por nuestro grupo de Sistemas Informáticos contempla toda la circuitería propia que rodea a un microprocesador (en nuestro caso, microcontrolador), desde un controlador del sistema, que haga de interfaz entre el micro, la memoria y los distintos periféricos, pasando por un controlador de interrupciones que se encargue de su gestión, hasta los distintos periféricos de entrada/salida : terminal VGA alfanumérica, generador de sonido, módulo de teclado, y generador de números pseudo-aleatorios.

A continuación, daremos un esquema de la arquitectura completa para después ir comentando los distintos módulos, su estructura básica y ciertas cuestiones de diseño.

4.1 – VISIÓN GENERAL:



El funcionamiento del sistema, a grandes rasgos, es el siguiente:

Tanto los programas como la memoria de vídeo se almacenan en el banco de memoria disponible. Los primeros 16 Kb (0000h-3FFFFh) están disponibles para el código del programa y los últimos 16 Kb (4000h-7FFFh) para la memoria de vídeo, que se desglosará en dos tipos: la VGA, al ser de carácter alfanumérica, debe tener un mapa de caracteres, que mediante sus códigos se trazará la pantalla, pero por otra parte, debe haber una descripción de esos caracteres en pixels (códigos RGB de 6 bits).

Como el esquema de direccionamiento del microcontrolador es de 16 bits para formar las direcciones, nos queda un espacio de 32 Kb (8000h-FFFFh) que utilizamos para mapear los distintos registros de los periféricos de entrada/salida así como de otros.

El controlador del sistema se encarga realizar peticiones a la memoria o a los distintos periféricos, en función de la dirección suministrada por el microcontrolador (NOTA: todas estas direcciones hacen referencia al espacio de direcciones de la memoria externa del micro – no confundir con el espacio de memoria interna, que el micro utiliza para la pila y para sus registros internos – así , debemos utilizar la instrucción movx @DPTR,A para las escrituras ó movx A,@DPTR para las lecturas, donde DPTR es el puntero que debe almacenar la dirección a la que se desea acceder).

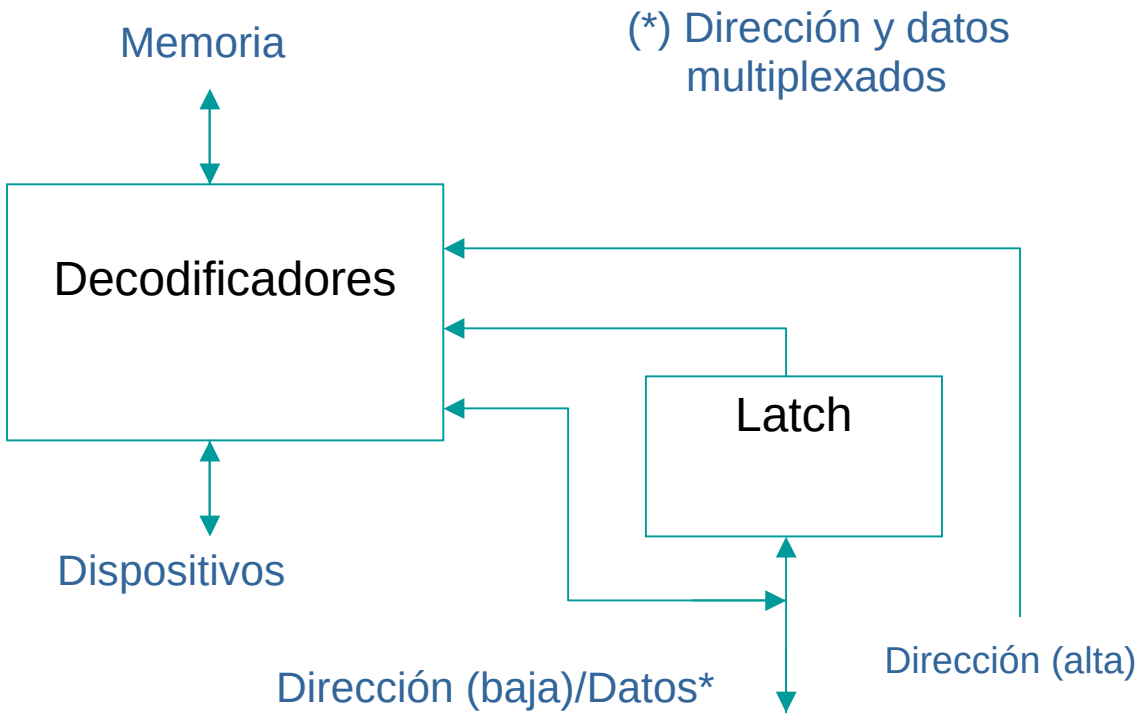
Por otra parte, el controlador de interrupciones es el encargado de controlar y priorizar las dos únicas fuentes de interrupción que existen: dormir/despertar al micro – fuente VGA – y tecla disponible – fuente PS/2, y transformarlas en las dos únicas líneas de interrupción externas del micro (INT0 e INT1). Más adelante, comentaremos más.

Los distintos periféricos serán estimulados por el controlador de sistema, mediante accesos a sus registros, o avisarán al controlador de interrupciones de eventos que puedan interesar (sólo VGA y PS/2). Cuando comentemos en más detalle los 4 periféricos de E/S del sistema (CODEC – generador de

sonido, PS/2 – módulo de teclado, VGA y ALEA – generador de números aleatorios) daremos su modelo de programación.

En cuanto a cuestiones de diseño, nada que comentar. Y en cuestiones de implementación, lo único el interconexionado manual necesario debido a que las líneas de interrupción del micro (INT0 e INT1) no estaban comunicadas con la FPGA. En el manual del programador, diremos la interconexión necesaria para poder que funcionen correctamente las interrupciones y, por tanto, el sistema.

4.2. – CONTROLADOR DEL SISTEMA:

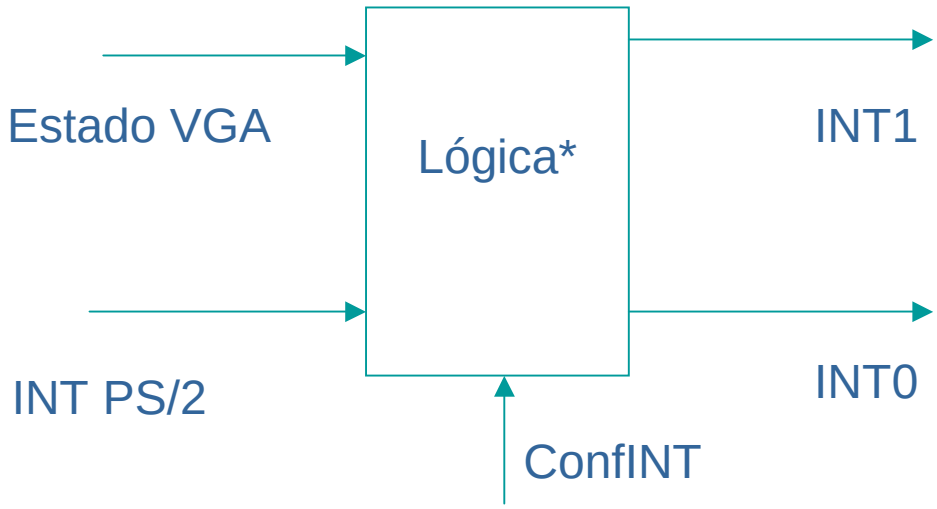


El controlador del sistema hace de interfaz entre el microcontrolador, la memoria y los dispositivos incluidos en la arquitectura. Los descodificadores son los que se encargan de analizar la dirección y transportar la petición a la memoria o a un dispositivo en concreto.

Nada más que comentar sobre el controlador, aparte de citar que el 8031 comparte en las mismas líneas la parte baja de la

dirección y los datos, que a su vez están conectados directamente con la línea de datos de la memoria en el dibujo de la placa. Esto presenta un doble problema: por una parte tener que “latchear” la dirección para que no se corrompa cuando se vuelquen los datos desde memoria o desde el propio micro, y por otro, la necesidad de poner esas líneas en alta impedancia. En la sección de la VGA abordaremos este último punto más detenidamente.

4.3. – CONTROLADOR DE INTERRUPCIONES:



(*) INT1 mayor prioridad

El controlador de interrupciones tiene como objetivos priorizar las dos fuentes de posible interrupción, en las dos líneas de interrupción externas que puede tener el 8031.

Por una parte, la señal EstadoVGA que proviene del módulo VGA, se encarga de informar el estado de la VGA, si se encuentra en disposición de acceder a memoria o de dejar de usarla. De esta manera, cualquier cambio en esta señal provocará una interrupción: INT1 para avisar que la VGA va a usar la memoria, que debe provocar en la rutina de tratamiento la ejecución de la instrucción que “duerma” al micro (pararse y poner sus líneas en alta impedancia) e INT0 para avisar que va a dejar de usarla y que el

micro puede operar. Es importante destacar el hecho de utilizar dos interrupciones distintas para “dormir” y “despertar” al micro: en principio, cualquier interrupción puede sacar al micro de su estado en alta impedancia, pero al parecer no se puede utilizar la misma que lo provocó (debido a que cualquier fuente de interrupción, no solo externas, puede despertar al micro, hay que tener cuidado con la utilización de los contadores/temporizadores del que microcontrolador dispoen y que son causa de interrupción, esto lo veremos más detenidamente al analizar las restricciones de las aplicaciones desarrolladas para la arquitectura).

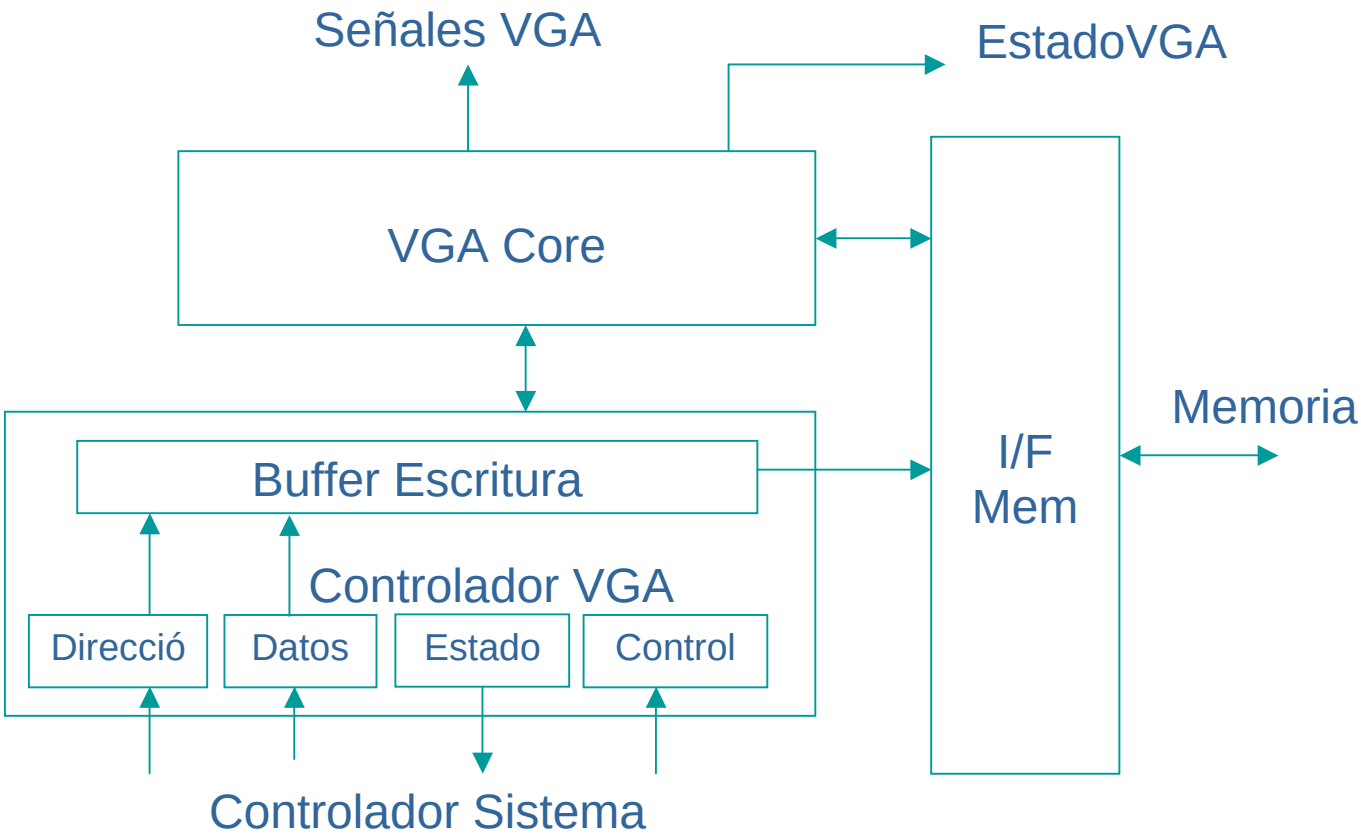
Hay que destacar dos hechos: el primero es que debido a que desde que la VGA avisa que el micro debe “dormirse” hasta que el micro lo hace deben pasar ciertos ciclos (terminación de instrucción en curso, ejecución de instrucciones de confirmación, instrucción para ponerlo en alta impedancia, ...) la VGA debe anticipar este hecho una serie de ciclos mínimos antes de que realmente utilice la memoria, sino puede haber colisión entre microcontrolador y VGA.

La segunda cuestión tiene que ver con la obligación de que cuando el micro se “duerma” entonces la línea INT1 debe estar deshabilitada. Eso nos planteó un posible cálculo de cuántos ciclos debíamos habilitar INT1 en la transición: debía ser lo suficientemente pequeño para que el microcontrolador no la encontrara habilitada cuando se dispusiera a pasar a estado de alta impedancia, pero lo suficientemente largo para que se entrase en la rutina de interrupción. Después de muchas pruebas, ninguna garantizó una estabilidad total, así que decidimos que la propia rutina avisara al controlador de interrupciones de cuando cesar: señal confINT. La rutina de tratamiento INT1 debe avisar al gestor de interrupciones antes de dormir al microcontrolador (lo veremos en más detalle más adelante, cuando analicemos el apoyo software necesario para que el sistema funcione).

La interrupción INT0 es utilizada tanto para despertar como para avisar de que hay tecla disponible en el teclado esperando ser procesada. El gestor de interrupciones establece una prioridad para que mientras el micro este “dormido”, INT0 no se habilite más que para avisar de que se debe “despertar”. Debido a que los eventos

de tecla disponible y “despertar” al micro son compartidos por la misma línea de interrupción, es responsabilidad del usuario saber detectar cada situación.

4.4. – VGA:



El módulo VGA es el más complejo de todos (se lleva casi un 50% de la capacidad de la FPGA en CLB’s). Es el encargado de refrescar la pantalla a partir del mapa de vídeo contenido en la memoria. En una VGA alfanumérica, es decir, el mapa de pantalla no se forma a partir de pixels, sino de caracteres que mediante sus códigos son identificados. Después, cada carácter debe tener su mapa de pixels, también en el banco de memoria. Por lo tanto, tenemos dos tipos de memoria de vídeo: la memoria de refresco y la memoria de pixels. Ambas se pueden modificar.

La VGA está dividida en tres módulos:

- VGA Core : es el encargado de refrescar la pantalla a partir de la información almacenada en memoria. En principio la pantalla es de 30x30 caracteres y cada carácter es de 10x8 pixels (en realidad son caracteres de 10x16, pero las líneas son dobles). Por lo tanto, la pantalla tiene una resolución de 300x240. Existen un total de 128 caracteres distintos, y cada píxel puede tomar 64 valores (código RGB de 6 bits, dos para cada color). Existen muchos atributos (programables), como por ejemplo, VGA encendida o apagada, modo color o modo monocromo, y el tamaño de ventana. Todos esos atributos son gestionados por el controlador de VGA. El módulo VGA Core es el encargado de generar la señal EstadoVGA que avisa al gestor de interrupciones de si la VGA está accediendo a memoria o no.

- Controlador VGA: este módulo es la interfaz de la VGA con el controlador del sistema. Mediante él, se puede programar la VGA, tanto para cambiar su modo mediante el registro de control (color ó monocromo, seleccionar tipo de memoria para escribir, apagar o encender), ver su estado (ver modo de VGA, error en escritura, buffer saturado, estado de blanking vertical) aparte de cambiar la memoria de vídeo (tanto la memoria de refresco como la de pixels) y cambiar el tamaño de ventana. Todos los cambios en el modo se hacen sincronizando con el retrazado vertical (y así evitamos efectos molestos). Este módulo dispone además de un buffer de escritura que contiene todas las peticiones de escritura en memoria de vídeo, para que se hagan cuando la memoria está libre. De esta manera, el acceso a memoria se divide en tres zonas: cuando lo usa el microcontrolador, cuando lo usa la VGA para leer y refrescar, y cuando lo utiliza la VGA para escribir y así poder modificar la memoria de vídeo. Por supuesto, estas escrituras, se realizarán sincronizando con el retrazado vertical, para evitar el efecto “pantallazo”.

- Interfaz de Memoria : como su propio nombre indica sirve para arbitrar entre las lecturas del módulo VGA Core para refrescar y el módulo Controlador VGA para las escrituras del buffer.

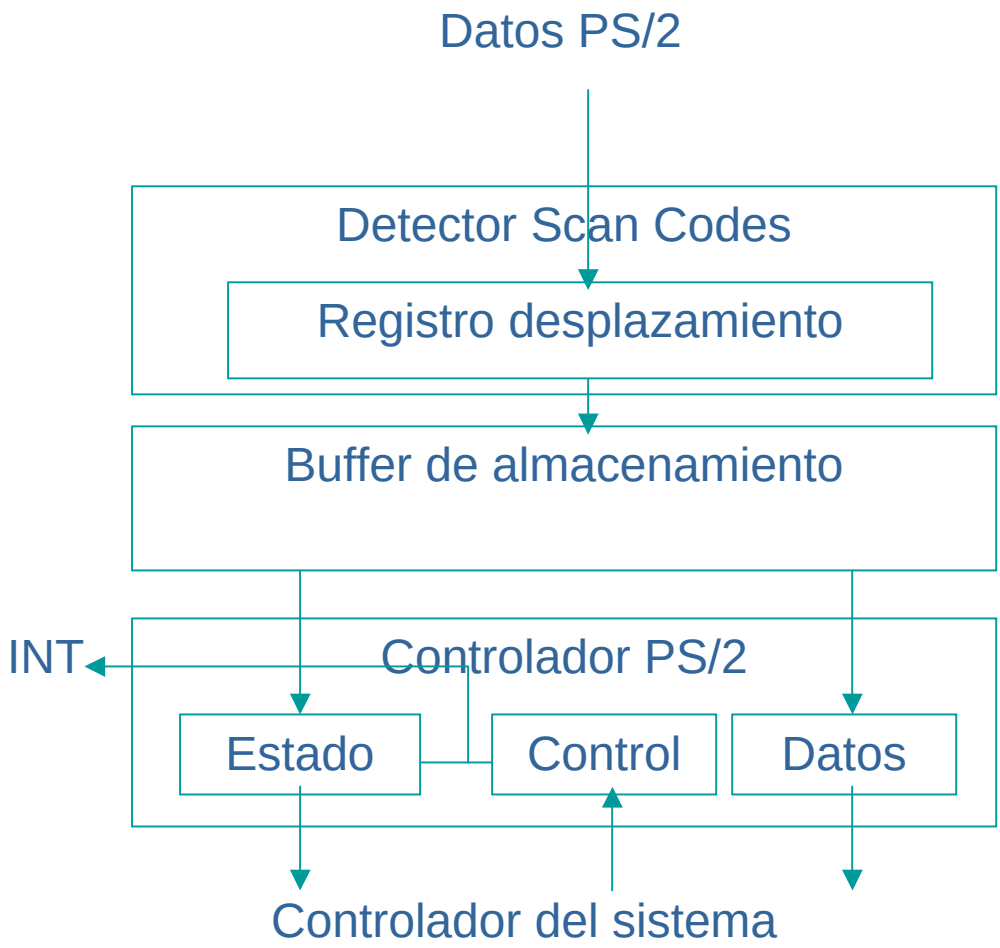
Los puntos más llamativos, en cuanto a diseño, de la VGA alfanumérica son, entre otros, la utilización del buffer de escritura. El tamaño del buffer es de 32x21 bits (hasta 32 peticiones se pueden hacer antes de que se vacíe) y se ha implementado utilizando memoria de LUT's (de esta manera podemos almacenar en 1 CLB 32 bits de memoria), ya que de otra manera no podría haber cabido todo en el diseño (esta misma técnica de usar memoria de LUT's lo hemos aplicado en el buffer de anticipación del VGA Core que almacena los códigos de caracteres de la fila a rellenar en la pantalla y así obtener la dirección del píxel a pintar y para los buffers del PS/2 y CODEC que después comentaremos).

En principio, este buffer no sería necesario, ya que al compartir tanto el microcontrolador como la VGA el mismo banco de memoria, conociendo de que manera se almacena la memoria de vídeo, podríamos modificarla sin pasar por el buffer. El diseño se hizo así debido a la posibilidad de utilizar dos FPGA's para el diseño final, separando así la memoria de programa de la memoria de vídeo. La inclusión de este buffer, nos ofrece dos posibilidades de modificar dicha memoria de vídeo con su pros y contras (hacerlo a través del controlador de VGA ofrece mayor seguridad ya que ésta realiza comprobaciones pero estás limitado al tamaño del buffer que una escritura directa no te impone – piensa que hasta que no le toque el turno al buffer, y es una vez por cada refresco, este no se vacía, por lo tanto, no se pueden hacer más de 32 actualizaciones por refresco).

Por último destacar el hecho de que los tiempos dedicados para el micro y la VGA dependen del tamaño de ventana. Es una cuestión normal, ya que si el tamaño de ventana vertical es menor, hay que rellenar menos líneas y esos tiempos se pueden dedicar al microcontrolador. Un ajuste al tamaño de ventana horizontal no repercutirá en el rendimiento del microcontrolador, debido a que el tiempo que pudiéramos ganar no es rentable ni tampoco el hardware que lo controlase (es mejor disponer de una franja larga

de tiempo para el microcontrolador que se consigue recortando el tamaño de ventana vertical, que múltiples franjas pero pequeñas que se conseguirían modificando el tamaño de ventana horizontal, debido a esos tiempos muertos necesarios que pasan desde que se realiza la petición de dormir al micro hasta que se duerme realmente).

4.5. – Módulo PS/2:



El módulo PS/2 se encarga de recoger los códigos enviados por el puerto PS/2 (conectado a un teclado) y almacenar el SCAN-CODE en un buffer a la espera de ser leído por el microcontrolador. Éste puede avisar mediante interrupción que hay tecla disponible (pulsada o soltada), aunque mediante el registro de control del módulo podemos activar o desactivar dicha interrupción. El registro

de estado avisa del estado del buffer, y así poder realizar comprobaciones antes de la lectura del código.

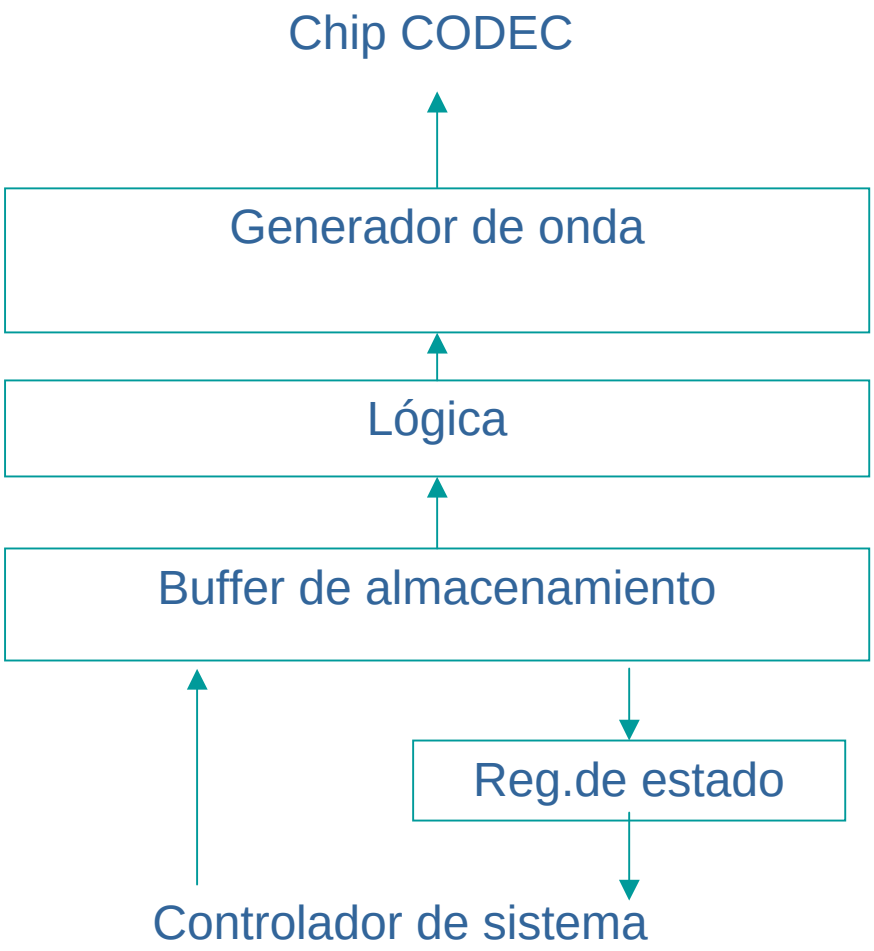
El módulo se divide en tres partes bien diferenciadas:

- Detector SCAN-CODES: es la parte que se encarga de la comunicación con el puerto serie PS/2. Mediante un registro de desplazamiento va recogiendo los datos enviados de manera serie, y una vez terminada la comunicación, extraer el SCAN-CODE y llevarlo al buffer de almacenamiento.

- Buffer de almacenamiento: tiene un tamaño de 16 palabras (de 8 bits) y ha sido implementado mediante memoria de LUT's (como los buffers de la VGA). Este tamaño es suficiente para evitar su desbordamiento siempre y cuando el programa sea correcto. El buffer funciona como una cola FIFO: se llenará por la cola cada vez que una nueva tecla (pulsada o soltada) haya sido reconocida y se vaciará por la cabecera cuando se solicite.

- Controlador PS/2: permite la programación el módulo mediante su registro de control (habilitar/deshabilitar interrupción), ver el estado del buffer mediante el registro de estado (buffer vacío, buffer lleno, buffer desbordado) y leer códigos SCAN-CODES de teclas pulsadas y soltadas mediante su registro de datos (una lectura provocará su eliminación del buffer). La línea de interrupción (si su programación la ha activado) siempre estará activa (en baja) mientras haya teclas disponibles.

4.6. – Generador de Sonido:



El generador de sonido es un periférico de salida para poder tocar notas de cierta frecuencia y duración (existe un banco de 16 frecuencias y 8 duraciones distintas, sólo se puede escoger entre ellas). Es un módulo muy sencillo que se compone de las siguientes componentes básicas:

- Registro de estado/buffer de almacenamiento: es registro de estado representa la parte que se comunica con el controlador del sistema. No es posible la programación del codec de sonido. Lo único que se puede hacer con él es enviar notas con cierta duración y frecuencia (de entre las 16 frecuencias y 7 duraciones disponibles). El buffer va encolando según la política de las colas FIFO (primero en

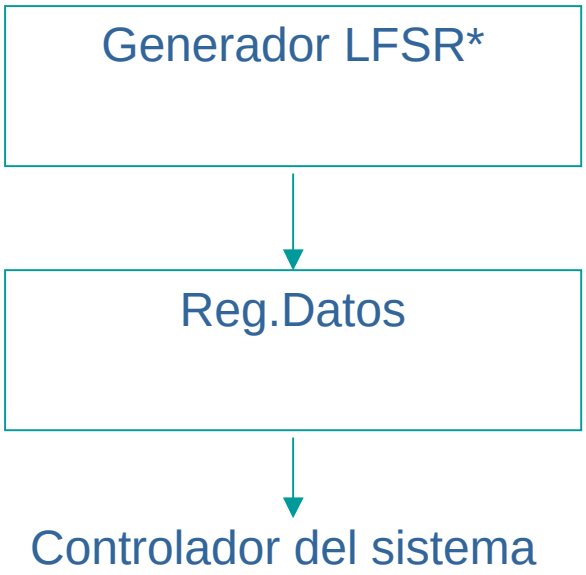
entrar, primero en salir), está implementado con memoria de LUT's y su tamaño es de 16 notas. El generador de onda las va consumiendo hasta vaciarlo. El registro de estado informa de la situación del buffer de almacenamiento (buffer vacío, buffer lleno, buffer desbordado).

- Lógica/Generador de onda : es la parte que se comunica con el CODEC de sonido, y se encarga de estimularlo a partir de la información del buffer de almacenamiento y de las tablas de frecuencias y duraciones. Este módulo está formado básicamente por contadores que generan la onda cuadrada con la frecuencia y duración requeridas y que envían de manera serie al CODEC.

El CODEC de sonido está limitado a las frecuencias y duraciones de las que dispone. Éstas están almacenadas de manera cableada mediante tablas con información para que el generador de onda pueda estimular al CODEC correctamente.

Este módulo no utiliza interrupciones para poder avisar de ciertos eventos, como que el buffer se está vaciando o similares. Por lo tanto, estamos limitando ráfagas de 16 notas distintas. Debemos utilizar recursos software para poder enviar melodías con mayor número de notas. En el manual del programador (próxima sección) veremos como se puede hacer esto, y así superar las limitaciones del hardware.

4.7. – Generador de números pseudo-aleatorios:



(*) Linear Feedback Shift

El generador de números pseudo-aleatorios o ALEA, es un módulo muy sencillo que genera números entre 0 y 254 en secuencias aleatorias. Está basado en el LINEAR FEEDBACK SHIFT, o sea, registro de desplazamiento realimentado que continuamente está generando dichas secuencias de números.

Cuando el controlador del sistema realiza una petición (no hay registro de estado ni de control, sólo se pueden obtener números – registro de datos), éste registro realimentado se detiene y se extrae el número que actualmente se está generando.

En principio este módulo no iba a ser incluido en el diseño final, pero para facilitar la programación de los eventos aleatorios del juego Galaxy Invader, se optó por implementarlo, ya que su bajo coste en espacio y su sencillo diseño lo permitió.

5– MODELO DE PROGRAMACIÓN:

Esta sección permitirá a potenciales programadores escribir aplicaciones libre de errores para la arquitectura diseñada. Debido a las peculiaridades del diseño, el software debe presentar una serie de características y a la vez restricciones para que el sistema pueda funcionar.

Además, explicaremos de que manera usar los distintos periféricos de los que disponemos: VGA alfanumérica, PS/2 para teclado, generador de sonido y generador de números pseudo-aleatorios (ALEA). No sólo nos quedaremos en una simple referencia sobre su uso, sino que daremos consejos para obtener mayores rendimientos y técnicas (con ejemplos inclusive) para poder superar las limitaciones del hardware, todas ellas basadas en nuestra experiencia al programar el juego Galaxy Invader.

Por último, mostraremos como configurar la placa FPGA (interconexionado manual, estado de los jumpers, ...) para el correcto funcionamiento y como resetear el sistema una vez cargado mediante la utilidad Gxsport.

5.1. – Restricciones del sistema / apoyo Software:

No todo programa creado para el 8031 funcionará en el sistema. Existen una serie de restricciones y propiedades que todo programa debe respetar para que el sistema funcione.

La más importante de ellas hace referencia a la RUTINA DE INTERRUPCIÓN INT1 que se debe encargar de “dormir” al micro para que no haya colisión en el acceso a la memoria con la VGA. Dicha rutina debe deshabilitar las interrupciones, guardar el estado de la máquina, confirmar al gestor de interrupciones que se ha recibido la petición, ejecutar la instrucción para “dormir” al micro, recuperar el estado de la máquina y, por último, habilitar las interrupciones. Opcionalmente, se puede hacer una comprobación por si el micro se hubiera despertado sin tener que hacerlo, aunque

no debería pasar si se aplican las reglas que se darán (y así evitar que el micro se despierte sin tener que hacerlo).

Por supuesto, la INTERRUPCIÓN INT1 DEBE ESTAR HABILITADA desde el principio y ninguna más exceptuando INT0 ya que es controlada por el gestor de interrupciones. El resto de interrupciones (interrupción por comunicación serie y la de los contadores/temporizadores) NO ES ACONSEJABLE ACTIVARLAS.

Los contadores TIMER0,TIMER1 y TIMER2 NO SE DEBEN UTILIZAR ya que aunque las interrupciones correspondientes estén desactivadas, pueden provocar que el que el micro despierte sin tener que hacerlo.

A continuación, veremos un poco de código de ejemplo y lo comentaremos:

```
; RUTINA PRINCIPAL
MAIN:

; inicializo pila
mov SP,40
```

Aquí por ejemplo sólo activamos INT0 e INT1 (es aconsejable no activar ninguna más, debido a que cualquier interrupción puede despertar al micro):

```
; INICIALIZACION INTERRUPCIONES

; preparo interrupciones

; interrupciones controladas por IE (bit PCT=1)
; PCT --- PT2 PS PT1 PX1 PT0 PX0
;  1     0  0    0    0    0    0    0
mov IP,#10000000b

; habilito interrupciones INT1 siempre,e INT0 solo si interesa (teclado)
; EA --- ET2 ES ET1 EX1 ET0 EX0
;  1   0    0   0   0    1    0    1
mov IE,#10000101b
```

INT1 siempre programado por flanco e INT0 por flanco SI Y SOLO SI la interrupción de teclado está activada (IMPORTANTE):

```
; habilito INT1 por flanco (IT1=1)
; habilito INT0 por flanco solo si esta activado su interrupcion (importante)
; TF1 TR1 TF0 TR0 IE1 IT1 IE0 IT0
;  0    0    0    0    0    1   0   1
```

```
mov TCON,#00000101b


; CODIGO DEL PROGRAMA

……
……
```

Ahora analizaremos la rutina INT1 (dormir al micro) que todo programa debe tener (recordamos que INT1 siempre debe estar activada en todos los programas):

```
; RUTINA DE TRATAMIENTO DE INTERRUPCION PARA EL INT1
; DORMIR AL MICRO
RUT_INT1:

; desactivar interrupciones
mov IE,#00000000b

; guardo estado de la maquina
push PSW
push ACC
push DPH
push DPL
```

Después de entrar en la rutina y antes de “dormir” al micro es OBLIGATORIO confirmar al gestor de interrupciones este hecho : eso se hace realizando una escritura (da igual el valor de dicha escritura) en la dirección 0FFFFh:

```
; avisamos que int atendida
mov DPTR,#0FFFFh
movx @DPTR,A
```

Dormimos el microcontrolador:

```
volverDormir:

; pongo bit ALF=1, para ponerme en estado FLOATING
; --- T32 SERR IZC P3HZ P2HZ P1HZ ALF
;  0   0     0      0     0      0      0     1
; IOCON = 0F8h
mov 0F8h,#00000001b

; duermo al micro colocandolo en modo Power Down (PD=1),
; y permito que cualquier interrupcion le despierte, continuando la ejecucion
; normal (RPD=1)

; SMOD HPD RPD --- GF1 GF0 PD IDL
;   0       0      1     0  0     0    1   0
orl PCON,#00100000b
orl PCON,#00000010b
```

Comprobación de que hemos despertado “correctamente” : se hace mediante una lectura a la dirección 0FFFFh : si devuelve cero entonces no hemos “despertado” correctamente:

```
; hago una lectura -> esta el micro dormido? si es asi volvemos
mov DPTR,#0FFFFh
movx A,@DPTR
jz volverDormir

; recupero estado de la maquina
pop DPL
pop DPH
pop ACC
pop PSW

; activar interrupciones
mov IE,#10000101b

;volvemos
reti
```

REPASO de las restricciones/propiedades de los programas:

- SIEMPRE HABILITAR INT1 (NUNCA DESHABILITARLA EN NINGUNA RUTINA)
- OPCIONALMENTE SE PUEDE HABILITAR INT0 (si se va a utilizar el teclado por interrupción)
- PROGRAMAR INT1 SIEMPRE POR FLANCO E INT0 POR FLANCO SI ESTÁ HABILITADA (SINO, POR NIVEL)
- NO HABILITAR EL RESTO DE INTERRUPCIONES
- NO USAR LOS CONTADORES/TEMPORIZADORES TIMER0,TIMER1 y TIMER2
- INCLUIR RUTINA INT1 PROPUESTA ANTERIORMENTE

5.2. – Programación de la VGA:

La VGA alfanumérica puede ser programada mediante su registro de control cambiando su modo (color, monocromo), apagarse o encenderse, cambiar el tamaño de la ventana y, por supuesto, su contenido. Todos los cambios que se hagan en la VGA se aplican sincronizándose con el retrazado vertical y así evitar el molesto efecto de parpadeo. Por otra parte, es posible ver el estado de la VGA y su programación actual mediante su registro de estado.

Sus dimensiones son de 30x30 caracteres y cada carácter una dimensión de 10x8 pixels (en realidad son 10x16 pixels porque las líneas horizontales se desdoblan). En total, la resolución de la pantalla es de 300x240 pixels (300x480). Existe un total de 128 caracteres distintos para programar y una gama de 64 colores distintos con 4 tonos para rojo, verde y azul (código RGB).

Debido a que es una VGA alfanumérica, se distinguen dos tipos de memoria de vídeo : la memoria correspondiente al mapa de caracteres y otra para describir cada carácter mediante sus pixels. Ambos tipos de memoria se pueden escribir de dos formas distintas: a través de la programación de la VGA y escribiendo en la memoria de vídeo directamente (eso es posible porque en el diseño se utiliza la misma memoria para el micro como para la VGA). Cada forma tiene una ventaja y un inconveniente : escribir a través de la VGA permite una mayor seguridad ya que hay control sobre lo que se escribe pero estás limitado al tamaño de buffer de escritura (capacidad de 32 escrituras) que se vacía una vez por cada refresco de pantalla. Utilizar la escritura a través de la memoria de vídeo no te limita al tamaño del buffer, pero no hay ningún mecanismo que proteja de escrituras erróneas.

Programación del registro de estado/control de la VGA:

Registro de control : dirección 0FF10h

-	-	-	-	-	Encender	Tipo	Modo

					VGA	Mem	Color

Registro de estado : dirección 0FF10h

-	-	Error Escritura	Buffer Esc.Lleno	Blanking Vertical	Encender VGA	Tipo Mem	Modo Color

El registro de control sólo tiene 3 bits (los 3 bits más bajos del registro de estado se corresponden con esos bits para saber su actual programación) para:

- poner modo color/monocromo (bit 0 : 0 -> modo monocromo, 1 -> modo color).
- seleccionar tipo de memoria para escribir (bit 1 : 0 -> seleccionar memoria de pixels, 1 -> seleccionar memoria de refresco/caracteres).
- para encender/apagar la VGA (bit 2 : 0 -> apagar la VGA, 1 -> encender la VGA).

Inicialmente la VGA arranca encendida, modo color y seleccionada la memoria de refresco.

El registro de estado extiende esos 3 bits a otros 6 para saber el estado actual de la VGA :

- blanking vertical (bit 3 : 1 si la VGA no está rellenando).
- buffer de escritura lleno (bit 4 -> 1 si el buffer está lleno y no permite más escrituras hasta que se vacía). Si el buffer está lleno y se intenta otra escritura, ésta última se descarta.
- error de escritura (bit 5 -> si devuelve 1 la última escritura programada fue errónea debido a que se salió de los límites establecidos, por supuesto, esta última escritura es descartada).

Ejemplos:

Programar VGA para encenderse, color y seleccionar memoria de refresco:

```
mov DPTR,#0FF10h
```

```
mov A,#00000111b
movx @DPTR,A
```

Seleccionar memoria de pixels sin cambiar el resto de su programación actual:

```
; consulto estado actual
mov DPTR,#0FF10h
movx A,@DPTR
; cambio bit 1 a cero
anl A,#11111101b
movx @DPTR,A
```

Verificar que el buffer de escritura no esté lleno:

```
mov DPTR,#0FF10h
movx A,@DPTR
anl A,#00010000b
jz bufferNoLleno
........
........
........
bufferNoLleno:
........
........
```

Programación del tamaño de ventana:

La programación del tamaño de ventana se hace mediante cuatro registros (0FF14h para coordenada izquierda, 0FF15h para coordenada derecha, 0FF16h para coordenada superior y 0FF17h para coordenada inferior). Lo único que hay que hacer es indicar un número entre 0 y 29 para establecer la nueva coordenada (una coordenada mayor que 29 la asume la VGA como 29).

Se pueden hace escrituras en posiciones que quedan fuera del tamaño de ventana. Los cambios se harán pero no se notarán hasta que se vuelva a cambiar el tamaño de ventana y englobe a la posición modificada.

La modificación de las coordenadas inferior y superior de la ventana de la VGA supone una redistribución de los tiempos dedicados a microcontrolador y VGA. Una menor ancho vertical

supone un mayor rendimiento en la aplicación debido a que la VGA no debe refrescar tanto.

Un ejemplo: cambiar tamaño de ventana a 10-20 posiciones horizontales y 5-15 posiciones verticales:

```
mov DPTR,#0FF14h
mov A,#10
movx @DPTR,A
mov DPTR,#0FF15h
mov A,#20
movx @DPTR,A
mov DPTR,#0FF16h
mov A,#5
movx @DPTR,A
mov DPTR,#0FF17h
mov A,#15
movx @DPTR,A
```

Modificación de la memoria de vídeo:

Como hemos dicho anteriormente existen dos tipos de memoria (memoria de refresco/caracteres y memoria de pixels) y dos formas de modificarla:

A través de la VGA:

Pasos a seguir:

1) Seleccionar tipo de memoria (bit 1 del registro de control : 0 -> memoria de pixels, 1 -> memoria de refresco)
2) Escribir la dirección a escribir en los registros 0FF11h (dir.alta) y 0FF12h (dir.baja). Su codificación es la siguiente:

   Seleccionada memoria de refresco:

   0FF11h -> 0 0 0 y4 y3 y2 y1 y0
   0FF12h -> 0 0 0 x4 x3 x2 x1 x0

   Siendo y4y3y2y1y0 coordenada Y del carácter a modificar y x4x3x2x1x0 coordenada X.

Seleccionada memoria de pixels:

0FF11h -> 0 c6 c5 c4 c3 c2 c1 c0
0FF12h –> 0 y2 y1 y0 x3 x2 x1 x0

Siendo c6c5c4c3c2c1c0 código de carácter a modificar y, y2y1y0 y x3x2x1x0 coordenada Y y X del píxel a modificar (del carácter seleccionado).

3) Escribir valor en registro 0FF13h (reg.datos). Cuando se realiza este paso, entonces la VGA, si no ha habido ningún error en los rangos o el buffer no está lleno, encola la petición a dicho buffer (NOTA : si la VGA está apagada, no es aconsejable realizar escrituras mediante este mecanismo porque el buffer nunca se vacía). Su codificación es la siguiente:

Seleccionada memoria de refresco:

0FF13h -> 0 c6 c5 c4 c3 c2 c1 c0

Siendo c6c5c4c3c2c1c0 código nuevo para la posición establecida (en los registros 0FF11h y 0FF12h).

Seleccionada memoria de vídeo:

0FF13h -> 0 0 R R G G B B

Siendo RRGGBB código RGB del nuevo píxel seleccionado.

Ahora mostramos una rutina utilizada en el Galaxy Invader como ejemplo de la modificación de la memoria de refresco a través de la VGA:

```
; Procedimiento para dibujar un caracter en la VGA :
; Parametros utilizados (que se pasan por memoria) por esta funcion;
;     carYDibujarCaracter:    coordenadaY caracter a modificar
```

```
;     carXDibujarCaracter:    coordenadaX caracter a modificar
;     codigoDibujarCaracter:  codigo del nuevo caracter

; Es necesario que la VGA este encendida y seleccionada para
; modificar la memoria de caracteres y no la de pixels

dibujarCaracter:
; seleccionamos memoria de refresco
mov DPTR,#DirRegControlVGA
mov A,#00000111b
movx @DPTR,A
; podemos escribir en la VGA?
mov DPTR,#DirRegEstadoVGA
poderEscrVGA:
movx A,@DPTR
anl A,#BufferLlenoVGA
jnz poderEscrVGA
; ya podemos escribir
mov DPTR,#DirRegDirAltaVGA
mov A,carYDibujarCaracter
movx @DPTR,A
mov DPTR,#DirRegDirBajaVGA
mov A,carXDibujarCaracter
movx @DPTR,A
mov DPTR,#DirRegDatosVGA
mov A,codigoDibujarCaracter
movx @DPTR,A
finDC:
; volvemos
ret
```

Otra rutina equivalente pero para escribir en la memoria de pixels a través de la VGA (también del Galaxy Invader):

```
; Procedimiento para dibujar un pixel de un caracter en la VGA :
; Parametros utiizados (que se pasan por memoria) por esta funcion;

;     codigoDibujarCaracter:  codigo del nuevo caracter
;     carYDibujarCaracter:    coordenadaY caracter a modificar
;     carXDibujarCaracter:    coordenadaX caracter a modificar
;     colorDibujarPixel:      color nuevo


dibujarPixel:
; seleccionamos memoria de pixel
mov DPTR,#DirRegControlVGA
mov A,#00000101b
movx @DPTR,A
; podemos escribir en la VGA?
mov DPTR,#DirRegEstadoVGA
poderEscrVGAPixel:
movx A,@DPTR
anl A,#BufferLlenoVGA
jnz poderEscrVGAPixel
; ya podemos escribir
mov DPTR,#DirRegDirAltaVGA
mov A,codigoDibujarCaracter
```

```
movx @DPTR,A
mov DPTR,#DirRegDirBajaVGA
mov A,carYDibujarCaracter
RL A
RL A
RL A
RL A
anl A,#11110000b
add A,carXDibujarCaracter
movx @DPTR,A
mov DPTR,#DirRegDatosVGA
mov A,colorDibujarPixel
;movx @DPTR,A
finDP:
; volvemos
ret
```

Ahora explicaremos la codificación en la memoria física de la memoria de vídeo para poder modificarla directamente:

Memoria de caracteres:

Dirección : - 1 0 0 y4 y3 y2 y1 y0 x4 x3 x2 1 1 x1 x0
Dato : - c6 c5 c4 c3 c2 c1 c0

Siendo y4y3y2y1y0 y x3x2x1x0 coordenadas Y y X del carácter
Y c6c5c4c3c2c1c0 código del carácter.

Memoria de pixels:

Dirección : - 1 c6 c5 c4 c3 c2 c1 c0 y2 y1 y0 x3 x2 x1 x0
Dato : - - R R G G B B

Siendo c6c5c4c3c2c1c0 código del carácter a modificar, y2y1y0 y x3x2x1x0 coordenada del píxel a modificar, y RRGGBB código RGB del nuevo píxel.

Ejemplo de cómo inicializar la memoria de video mediante programa (Galaxy Invader):

```
ORG 100000000000000b

; caracter 0000000 => 0
ORG 100000000000000b
```

```
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO
ORG 100000000010000b
        DB CFO,CFO,CFO,CNU,CNU,CNU,CNU,CFO,CFO,CFO
ORG 100000000100000b
        DB CFO,CFO,CNU,CFO,CFO,CFO,CFO,CNU,CFO,CFO
ORG 100000000110000b
        DB CFO,CFO,CNU,CFO,CFO,CFO,CFO,CNU,CFO,CFO
ORG 100000001000000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO
ORG 100000001010000b
        DB CFO,CFO,CNU,CFO,CFO,CFO,CFO,CNU,CFO,CFO
ORG 100000001100000b
        DB CFO,CFO,CNU,CFO,CFO,CFO,CFO,CNU,CFO,CFO
ORG 100000001110000b
        DB CFO,CFO,CFO,CNU,CNU,CNU,CNU,CFO,CFO,CFO



; caracter 0000001 => 1
ORG 100000010000000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO
ORG 100000010010000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO
ORG 100000010100000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CNU,CFO,CFO
ORG 100000010110000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CNU,CFO,CFO
ORG 100000011000000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO
ORG 100000011010000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CNU,CFO,CFO
ORG 100000011100000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CNU,CFO,CFO
ORG 100000011110000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO

; caracter 0000010 => 2
ORG 100000100000000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO,CFO
ORG 100000100010000b
        DB CFO,CFO,CFO,CNU,CNU,CNU,CNU,CFO,CFO,CFO
ORG 100000100100000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CNU,CFO,CFO
ORG 100000100110000b
        DB CFO,CFO,CFO,CFO,CFO,CFO,CFO,CNU,CFO,CFO
ORG 100000101000000b
        DB CFO,CFO,CFO,CNU,CNU,CNU,CNU,CFO,CFO,CFO
ORG 100000101010000b
        DB CFO,CFO,CNU,CFO,CFO,CFO,CFO,CFO,CFO,CFO
ORG 100000101100000b
        DB CFO,CFO,CNU,CFO,CFO,CFO,CFO,CFO,CFO,CFO
ORG 100000101110000b
        DB CFO,CFO,CFO,CNU,CNU,CNU,CNU,CFO,CFO,CFO
```

5.3. – Programación del teclado PS/2:

La programación del teclado permite establecer modo interrupción o modo polling mediante el registro de control. En modo interrupción (arranca por defecto en este modo), cada vez que haya una tecla disponible en el buffer se avisará mediante la interrupción INT0 (la misma utilizada para "despertar" al micro) que debe recoger tal evento y procesar la tecla. Es importante destacar que este buffer recoge tanto las teclas pulsadas (su scan-code) como teclas soltadas (su scan-code detrás del prefijo 0F0h). Por supuesto, para que el modo interrupción del teclado funcione la interrupción INT0 debe estar activada en el microcontrolador y programada por flanco.

El registro de estado permite saber el estado del buffer de teclas (si está vacío, si está desbordado – tiene un tamaño de 16 – etc ...). Por último, el registro de datos se debe leer para poder obtener dichos códigos del buffer (para las teclas soltadas se deben hacer dos lecturas : leer prefijo 0F0h y el scan-code correspondiente).

Registro de control : dirección 0FF00h

-	-	-	-	-	-	-	Modo INT

Modo INT : 1 -> modo interrupción, 0 -> modo polling

Registro de estado : dirección 0FF00h

-	-	-	-	-	Buffer Desbord	HayDatos Buf	Buffer Lleno

Buffer Lleno : 1 -> buffer lleno
HayDatos Buf : 0 -> buffer vacío
Buffer Desbord : 1 -> buffer desbordado

Cuando el buffer está lleno o desbordado, cualquier tecla nueva reconocida por el puerto PS/2 se ignora y no se encola.

Registro de datos : dirección 0FF01h

Ejemplo de programación del teclado por polling:

```
; programamos teclado por polling
mov DPTR,#0FF00h
mov A,#00000000b
novx @DPTR,A

; bucle hasta que haya tecla que tratar
hayTecla:
mov DPTR,#0FF00h
movx A,@DPTR
anl A,#00000010b
jz hayTecla

; leer tecla y tratarla
mov DPTR,#0FF01h
movx A,@DPTR

; en A está el scan-code de la tecla leída
.....
.....
```

Ejemplo de programación del teclado por interrupción:

```
; programamos teclado por interrupción
mov DPTR,#0FF00h
mov A,#00000001b
novx @DPTR,A

; activamos INT0 por flanco
mov IP,#10000000b
mov IE,#10000101b
mov TCON,#00000101b

.....
.....
.....

; RUTINA DE TRATAMIENTO DE INTERRUPCION PARA EL INT0
; DESPERTAR AL MICRO Y ATENDER PS2
RUT_INT0:

; deshabilito todas las interrupciones menos INT1
mov IE,#10000100b
```

```
; guardo estado de la maquina
push PSW
push ACC
push DPH
push DPL

; primero vemos si hemos pulsado tecla
; (podemos entrar aqui debido a que el micro despierta)
mov DPTR,#0FF00h
movx A,@DPTR
anl A,#00000010b
jz finINTPS2

; tratamiento del teclado
mov DPTR,#0FF01h
movx A,@DPTR


; en A está el scan-code de la tecla leída
.....
.....

finINTPS2:
; recupero estado de la maquina
pop DPL
pop DPH
pop ACC
pop PSW

; habilito INT0,INT1
mov IE,#10000101b

reti
```

5.4. – Programación del generador de sonido:

El generador de sonido permite enviar sonidos al registro de datos dando su frecuencia y duración para que sean tocadas por el CODEC. Existen 15 frecuencias y 7 duraciones distintas y éstas no se pueden cambiar. Las notas se van encolando en un buffer (cuyo tamaño es de 16 notas) que va consumiendo el generador de onda que estimulará al CODEC de sonido para que toque la nota correspondiente. El registro de estado permite informar acerca del estado de este buffer.

Registro de estado : Dirección 0FF20h

-	-	-	-	-	Buffer Desbord	HayDatos Buf	Buffer Lleno

Buffer Lleno : 1 -> buffer lleno
HayDatos Buf : 0 -> buffer vacío
Buffer Desbord : 1 -> buffer desbordado

Registro de datos : Dirección 0FF20h

-	D2	D1	D0	N3	N2	N1	N0

Donde : D2D1D0 es la duración de la nota y
N3N2N1N0 es la frecuencia de la nota

Tabla de frecuencias:

0000 -> DO
0001 -> DO sostenido
0010 -> RE
0011 -> RE sostenido
0100 -> MI
0101 -> FA
0110 -> FA sostenido
0111 -> SOL
1000 -> SOL sostenido
1001 -> LA
1010 -> LA sostenido
1011 -> SI
1100 -> DO mayor
1101 -> RE mayor
1110 -> MI mayor
1111 -> silencio

Tabla de duraciones:

000 -> semifusa
001 -> fusa
010 -> semicorchea
011 -> corchea
100 -> negra
101 -> blanca
110 -> redonda
111 -> redonda

Ejemplo de una rutina para tocar una nota (Galaxy Invader):

```
; Procedimiento para enviar un sonido al CODEC :
; Parametros utilizados (que se pasan por memoria) por esta funcion;
```

```
;       sonido:    nota-> formato 0d2d1d0f3f2f1f0 (d->duracion,f->frec)

enviarSonido:

mov DPTR,#0FF20h

; esperamos a enviar sonido
esperarSonido:
movx A,@DPTR
anl A,#00000101b
jnz esperarSonido

; enviamos sonido
mov DPTR,#0FF20h
mov A,sonido
movx @DPTR,A
```

La limitación del generador de sonido es el tamaño de su buffer. Para enviar melodías compuestas por un mayor número de notas es preciso mecanismos software. En el juego Galaxy Invader la melodía de la presentación del juego presenta ese problema que solucionamos con contadores que van alimentando al CODEC cada cierto tiempo. La dificultad está en saber elegir cada cuanto tiempo y que número de notas se envían para evitar tanto que el desarrollo de la presentación como la melodía se corten. Para mayor información ver el código del juego.

5.5. – Programación del generador de número pseudo-aleatorios:

La programación de este módulo es muy sencilla. Simplemente hay que realizar una lectura a su registro de datos (dirección 0FF30h) y obtener un número entre 0 y 254 que servirá de base para calcular la ocurrencia de un evento aleatorio.

Ejemplo de programación:

```
vov DPTR,#0FF30h
movx A,@DPTR

; en A número entre 0 y 254
.....
.....
.....
```

5.6. – Configuración de la placa y carga de los programas:

Para que la arquitectura funcione bien es preciso configurar los jumpers de la placa para que todos los dispositivos (CODEC, PS/2 ...) estén activos. Además, los switches deben estar en alta para que funcione bien el puerto PS/2.

Por último, realizar un par de conexionados manuales para que funcionen las interrupciones INT0 e INT1 que por defecto están desconectadas. Estas conexiones son:

- Patilla 12 con Patilla 71
- Patilla 21 con Patilla 72

Desde que un programa se realiza en ensamblador hasta que se carga en la arquitectura hay que seguir los siguientes pasos:

1 – Ensamblarlo :

```
asm51 programa.asm
```

2 – Cargarlo junto con el diseño :

```
xsload arquit.bit programa.hex
```

3 – Resetear la arquitectura:

Mediante la aplicación gxsport poner a 0 el bit más bajo del puerto paralelo y después a 1. En ese momento, el programa empieza a correr.