Proyecto de Tecnología y Organización de Computadores

Curso 2013/2014

Máquina Recreativa: Space Invaders

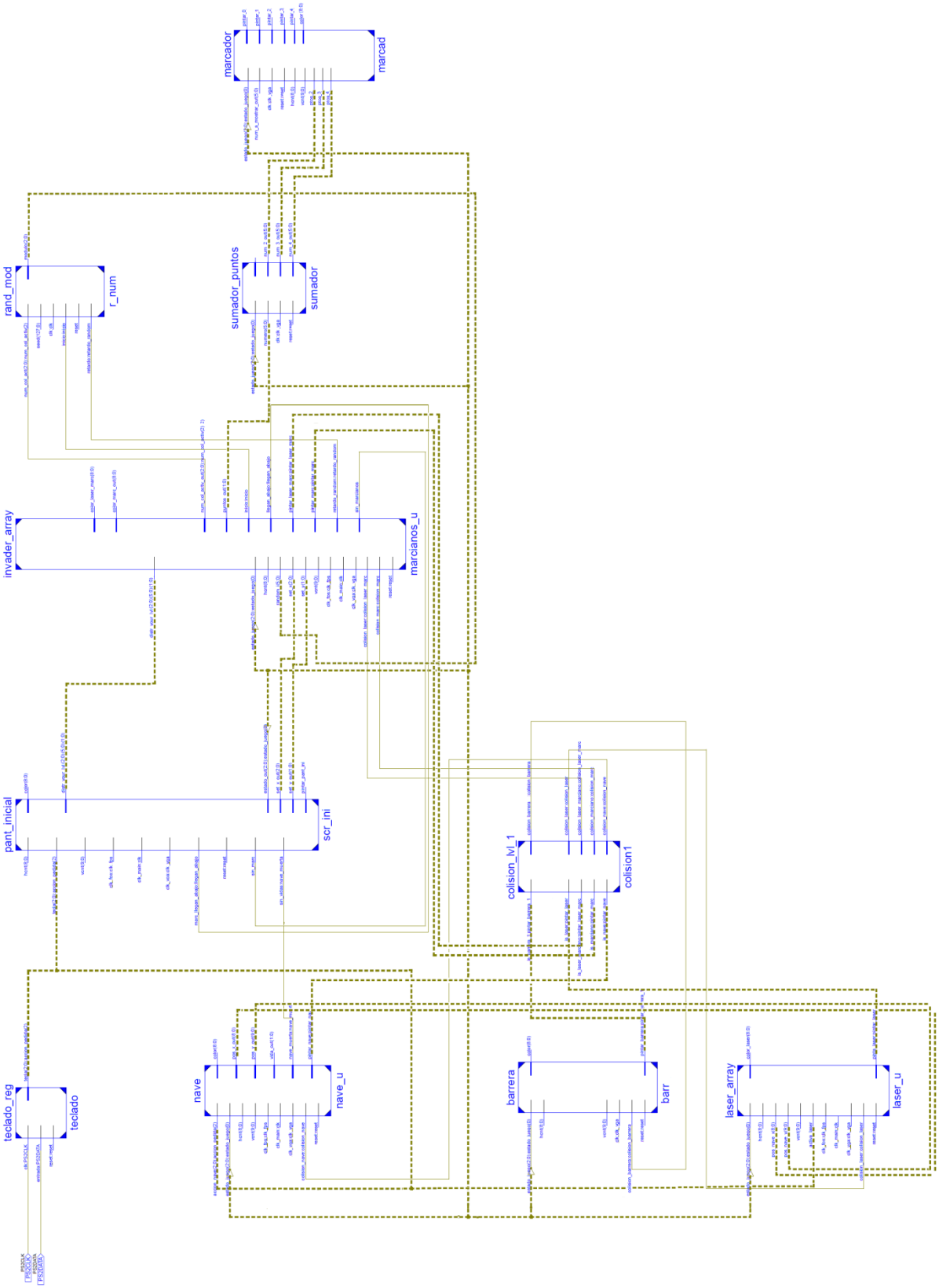
Descripción del Proyecto

Fernando Domínguez Estévez

Mario Lezcano Casado

David Martínez Rubio

1. Ruta de datos



En las siguientes líneas exponemos una explicación detallada por componentes del funcionamiento de cada uno de los módulos del proyecto:

VGA

Interconecta todos los componentes.

Recibe todas las señales de pintar de los diferentes componentes y los pinta en el caso de que se le pida.

Recibe la señal procesada por el teclado y se la envía a la nave y a la SCR_INI y a *laser_array.*

Devuelve un '1' a *activa_laser* del *laser_array* en el caso de que la señal sea DISPARAR.

En el caso de que no esté en los niveles Level1, Level2 o Your_level entonces se cogen los últimos 12 bits del contador del divisor de *clk_fps* y se envían como semilla al generador de números aleatorios, con tal de que tenga una nueva semilla cada vez que se entra a un nivel.

Pantalla inicial

En el estado pantalla inicial se muestran las imágenes de flecha, “nivel”, “uno”, “dos” y “crea tu”.

En los estados LEVEL1, LEVEL2, o YOUR_LEVEL se desarrolla el juego.

En el estado SET_LEVEL se selecciona el marciano que se quiera y se cambia de tipo.

En los estado LOSE y WIN se muestran respectivamente una imagen con el texto “GAME OVER” y “VICTORY” a la vez que se muestran los puntos conseguidos en el nivel.

Barrera

Recibe unas coordenadas las cuales serán su posición permanente.

En el caso de que se le notifique que ha recibido una colisión a través de *colision_barrera*, la parte de la misma que se está pintando se desactivará en el siguiente ciclo de *clk_vga*.

Colisiones

Recibe los elementos que se están pintando en ese instante. En el caso de que dos o más se estén pintando en un cierto momento a la vez, asíncronamente se ponen a '1' las señales de colisión correspondientes a los mismos

Marciano

Solo puede estar activo en los estados del juego: Level1, Level2, Your_level y Set_Level

Recibe el tipo de marciano que es, L, M o H, y según este se inicializa con la cantidad de vidas adecuada, 1, 1, o 3 respectivamente.

Recibe del *invader_array* su posición y en relación a esta pone a '1' o '0' la señal de pintar.

Recibe la señal de colisión del *invader_array* y en el caso de que esté a '1' en el siguiente ciclo disminuirá su vida en 1.

Si su vida llega a '0' indica que se ha desactivado y esto se devuelve al *invader_array*.

Sumador

Está activo si no se está en el estado INIT_SCR. En el caso de que esté en este estado se reinicia.

Recibe un número de puntos a sumar desde el *invader_array* que estará activa durante un ciclo. En el caso de que este número sea diferente de cero, lo retiene en una variable y lo sumará a la cantidad que ya tenía. La suma la realiza mediante sumas de una unidad, pudiendo obtener así un marcador en decimal.

Devuelve en tres señales diferentes los valores en decimal de la puntuación y las cifras significativas a pintar.

Marcador

Recibe cinco señales con valores del 0 al 9. Dentro se tienen unos dibujos de un display de 8 segmentos y según el número que se tenga que pintar, se toma una de las configuraciones de segmentos.

Devuelve una señal que está a '1' en el caso de que se tenga que pintar.

Devuelve también el color con el que se ha de pintar en ese momento.

Números Aleatorios

Tiene dos componentes:

Un generador de números aleatorios que recibe una semilla que utiliza en *reset* = ‘1’ y que devuelve un nuevo número aleatorio con cada de la señal retardo, la cual viene del *invader array*. Este ciclo ocurre justo después de cada disparo del marciano.

Un divisor secuencial parecido al realizado en una de las prácticas de clase pero que calcula el resto de la división del número aleatorio que provee el generador de números aleatorios entre otro número que viene como entrada (este número se recibe del *invader_array* y se corresponde con el número de columnas que hay activas en la matriz de marcianos).

El componente recibe una señal de *reset* de la vga (que no es el *reset* global) cuando empieza cualquier nivel de juego que hace que el generador de números aleatorios se resetee con la semilla que viene como entrada. El divisor realiza el módulo con el reloj rápido *clk_main* y en unos 120-140 ciclos devuelve el resultado que coloca en la señal de salida modulo. Este

componente también recibe la señal *inicio* que se pasa al divisor secuencial e indica cuando tiene que empezar las operaciones del cálculo del módulo.

Nave

Está activa en los estados del juego: LEVEL1, LEVEL2 y YOUR_LEVEL.

Recibe una acción que puede ser DERECHA, IZQUIERDA, ARRIBA, ABAJO O NADA. En función de ello la nave se moverá con cada ciclo de *clk_fps.* Si llega al límite de la pantalla no avanzará.

Recibe *colision_nave*, que si es '1', la nave pierde una vida con el *clk_vga*. Si la nave pierde todas sus vidas la señal de salida *nave_muerta* cambia a '1'.

Teclado

Con cada tecla que se pulsa el teclado recibe una o más veces, 11 bits con el código de la tecla pulsada y al soltar la tecla, el código de depresión F0 y luego otra vez el código de la tecla que se acaba de soltar.

El teclado tiene dos estados: ENVIAR y DESECHAR.

Inicialmente empieza en ENVIAR. Cuando recibe una tecla, si no es F0 la guarda en otro registro auxiliar. Si el código es F0 pasa al estado DESECHAR que desechará el siguiente código que reciba logrando así que el código de la tecla que se acaba de soltar no llegue al registro auxiliar y después automáticamente pasará de nuevo al estado ENVIAR. Asíncronamente, si el registro auxiliar es una de las teclas que reconoce, envía una acción por la salida. Así se logra que sólo se envíe una acción en caso de que el estado sea ENVIAR y no sea F0.

Láser

Este componente sólo está activo en los estados del juego: LEVEL1, LEVEL2 y YOUR_LEVEL.

Tiene dos estados: ACTIVADO o DESACTIVADO.

Si el laser está activo, con cada ciclo de *clk_fps* la posición del mismo aumentará en las dos dimensiones tantos pixeles como indican las señales de velocidad horizontal y velocidad vertical que vienen como entradas.

Si recibe un '1' por la señal *desactiva_laser* después de un ciclo de *clk_vga*, el estado del laser pasará a ser DESACTIVADO. Si el estado es DESACTIVADO y la señal de entrada *activa_laser* es '1', el siguiente estado será ACTIVADO. Si está activo y colisiona o llega a los límites de la pantalla el estado siguiente será DESACTIVADO.

Laser_array

Este componente sólo está activo en los estados del juego: LEVEL1, LEVEL2 y YOUR_LEVEL. La implementación consiste en un array circular de láseres y una señal que

apunta al primero que puede ser activado. Cuando el *laser_array* recibe la señal que indica que tiene que disparar, al láser asociado a la posición apuntada por la señal anterior se le envía por la señal *activa_laser* un '1' durante un ciclo de *clk_vga*. En ese momento, la señal *puede_activar* cambia a '0' y no vuelve a estar a '1' hasta que un contador que empieza a contar en este momento, llega a un valor prefijado. Si la señal *puede_activar* está a '0', aunque el *laser_array* reciba la acción de disparar, no disparará. También, en este momento, la señal que apunta al primer laser que puede ser activado aumenta una unidad. Nunca se va a intentar activar un láser ya activo pues el array circular es de un tamaño tal que antes de dar la vuelta completa al array circular, el láser ya ha desaparecido contra la pantalla.

Cuando un láser se activa, el *laser_array* envía a dicho láser su posición inicial. Ésta, es la posición actual de la nave que el *laser_array* recibe como entrada.

Recibe de cada uno de los láseres si se pintan o no, y en caso de que sí, de qué color. Usamos un codificador de prioridad para saber si alguna de las señales de pintar está a '1' y en ese caso cuál. Si alguno se pinta, la salida de *pinta_laser* del *laser_array* será '1' y el color, será el color del laser en la posición que indique la salida del codificador de prioridad.

Si el laser recibe un '1' por la señal de colisión, envía al láser que se está pintando (qué laser es viene dado por el codificador de prioridad anterior) un '1' por la señal *desactiva_laser*.

Invader_array

Este componente está activo en los estados Set_level, Level1, Level2 y Your_level

Inicialización:

Recibe una matriz con una configuración de marcianos que se utilizará en el caso de estar en el estado SET_LEVEL

Tiene una señal que al estar en SCR_INI estará a '1'. Esta se pondrá a 0 tras estar un ciclo de *clk_vga* en cualquiera de los niveles Level1, Level2 o Your_Level, dejando así tiempo suficiente como para que se inicialice ese nivel. De otro modo, si se está en el nivel SET_LEVEL esta señal permanecerá a '0'.

Observación: Es totalmente necesario que esta señal sea síncrona con el reloj adecuado, dado que en el caso de estar demasiado poco tiempo a '1' la inicialización podría no ser la correcta.

Control de los láseres:

1. Recibe un número aleatorio entre el 0 y la cantidad de columnas con algún marciano activo - 1.
2. Envía este número a un codificador de prioridad, el cual devuelve qué columna ha de disparar y cuál es el primer marciano activo de la misma.
3. Envía una señal de un ciclo de *clk_fps* a un laser o a los cinco de que se activen, dependiendo de si el marciano que ha de disparar es de tipo H o no (fuerte o no).

4. Tras esto envía una señal al generador de números aleatorios pidiéndole que calcule uno nuevo.
5. Activa un contador, el cual indicará al llegar a cierta cifra el momento de inicio del siguiente disparo.

Observación: Dado que el contador estará bastante más tiempo activo que lo que se tarda en generar el siguiente número aleatorio todo funcionará como se espera.

Control del estado de la matriz:

Se poseen 4 estados: IZQUIERDA, DERECHA, ABAJO y DESACTIVADO.

Inicialmente se inicia en DERECHA. Al llegar al final de la pantalla pasa a ABAJO, estado en el que está un ciclo. Tras esto pasa a izquierda y así sucesivamente.

Control de posición:

Se tienen dos vectores con las posiciones de las coordenadas X e Y respectivamente de los marcianos. Nótese que con dos vectores es suficiente para llevar las coordenadas de todos los marcianos dado que estos se encuentran en una disposición de cuadrícula.

Estos vectores se actualizan coherentemente con el estado de la matriz (IZQUIERDA, DERECHA o ABAJO).

Control de colisiones:

En el caso de que reciba una señal *colisión_marc* = '1' el marciano que se está pintando se desactivará en el siguiente ciclo. Además, si el marciano que se va a desactivar tiene exactamente una vida, se cambia durante 1 ciclo de *clk_fps* la señal puntos al valor correspondiente al tipo de marciano.

Control de la salida pintar

Se tiene una matriz de señales *estado_arriba* que irá alternando entre 0 y 1 la cual indicará que la imagen de cada marciano alterne entre dos posibles: Arriba y Abajo.

En el caso de que se esté en Level1, Level2 o Your_level, todos los marcianos se mostrarán en posiciones alternadas entre un ciclo y otro.

En el caso de que se esté en Set_Level, todos los marcianos se mostrarán hacia abajo salvo el seleccionado en ese momento, que vendrá dado por las entradas *set_x* y *set_y*, que alternará entre los dos estados.

En base a la posición, el tipo, las vidas y el *estado_arriba* de cada marciano, se pintará este de una forma y color.

2. Controlador

Nuestro proyecto no tiene un controlador en sí, ya que cada operación se realiza en paralelo y en un solo ciclo. No obstante hay dos módulos que hacen el papel de controladores globales. El primero de ellos es la VGA que coordina todas las señales y en algunos casos las transforma antes de volverlas a enviar al componente correspondiente. El segundo y más importante, que controla los estados del juego es el siguiente:

SRC_INI (Pantalla Inicial y Controlador principal):

Es la gestora de los estados del juego. Estos estados pueden ser:

- INIT_SCR
- LEVEL1
- LEVEL2
- SET_LEVEL
- YOUR_LEVEL
- WIN
- LOSE.

Inicialmente se comienza en la pantalla inicial, que señala los niveles tiene tres estados:

- FLECHA_LVL_1
- FLECHA_LVL_2
- FLECHA_SET_LVL.

Si está en FLECHA_LVL_1 la flecha apunta al nivel uno, análogo para el dos. Si es FLECHA_SET_LVL la flecha apunta a "crea tu nivel". Si se recibe la acción DERECHA en el estado FLECHA_LVL_1se pasa a FLECHA_LEVEL_2, con DERECHA, en FLECHA_LEVEL_2 se pasa a FLECHA_SET_LVL, con DERECHA, se vuelve a FLECHA_LEVEL_1.Con IZQUIERDA, en FLECHA_SET_LVL se pasa a FLECHA_LEVEL_2. En cualquiera de los estados si se recibe INTRO la pantalla inicial pasa al estado correspondiente al nivel o al estado SET_LEVEL en el caso de que se haya seleccionado en el estado de flecha FLECHA_SET_LVL.En los estados LEVEL1, LEVEL2, o YOUR_LEVEL se pasa al estado WIN si se recibe un '1' por la señal de entrada *sin_vidas*. Análogamente se pasa al estado LOSE si se recibe un '1' o en la señal de entrada *sin_marcianos* o en la señal de entrada *marc_llegan_abajo*.

En los estados LOSE y WIN al pulsar INTRO, se vuelve al estado INIT_SCR.

En el estado SET_LEVEL las señales set_x y set_y indican el marciano de la matriz que está seleccionado y se modifican con las acciones ARRIBA, ABAJO, IZQUIERDA y DERECHA. La pantalla inicial cuenta con una señal interna de matriz de tipos de marcianos inicializada todos a L. Si se pulsa la barra espaciadora el marciano indicado por la posición set_x, set_y cambiará de tipo. Si es L pasa a M, si es M pasa a H y si es H pasa a L. Estas señales (set_x, set_y y la matriz de tipos de marcianos), saldrán del componente y se enviarán al invader_array. Si se pulsa INTRO la pantalla inicial pasará al estado YOUR_LEVEL.

3. Soy el mejor…

En esta sección destacamos partes que consideramos meritorias de nuestro proyecto, que aunque el mismo se ha detallado en las dos secciones anteriores, creemos que merece la pena destacar alguna parte importante de su diseño e implementación o del aprovechamiento del hecho de que los componentes sean hardware y que todo funcione a la vez en paralelo.

Así mismo, explicamos problemas que hemos tenido durante el desarrollo y depuración de los componentes del proyecto y la solución que hemos dado:

Puntos generales:

- **Coordinación de las interacciones entre diferentes sistemas síncronos:** Todas las decisiones del juego se toman en a lo sumo un ciclo del reloj correspondiente (quitando la excepción del componente que calcula el módulo secuencialmente), haciendo así crucial la perfecta y coherente coordinación de las mismas.

- **Módulos totalmente independientes:** Cada módulo del juego trabaja de manera totalmente independiente. Estos podrían ser usados en otros proyectos con solo asignarles los relojes correspondientes en las entradas. Esto permite además una facilidad tanto de lectura del código como de ampliación del mismo.

- **Total parametrización del código:** No se usan en ninguna parte del código literales, permitiendo así el cambio de parámetros del mismo con solo cambiar las constantes asociadas a él. Quizá el ejemplo más representativo de esto sea la matriz de marcianos, en la cual se puede modificar el número de marcianos y sus posiciones iniciales en los niveles cambiando solo 4 constantes.

Comentarios concretos de componentes que merecen mención.

- **Colisiones:** Dado que las colisiones se efectúan si dos objetos se están pintando a la vez, y dado que tenemos 3 relojes diferentes y otros relojes de retardos, al principio no funcionaban correctamente las colisiones porque un objeto de los dos no desaparecía o ninguno lo hacía. La depuración de estos errores no fue fácil dado que las señales que desactivaban los componentes al chocar permanecían sólo un ciclo de reloj activas, con lo que no teníamos forma de ver qué elemento producía los errores. Al final conseguimos depurar esta parte analizando el código, estructurándolo y modificándolo hasta conseguir una estructuración de relojes que nos permitía realizar correctamente las colisiones de los cinco tipos de componentes.

- **Array circular de láseres:** En el space invaders original la nave sólo disparaba un láser y no podía disparar otro hasta que el primero desaparecía. Nosotros hemos ido más allá y hemos permitido que haya varios láseres a la vez en pantalla. Para ello hemos implementado un array circular de láseres que la nave puede ir activando, tras un retardo entre cada disparo establecido por un contador. El número de láseres del array circular es tal que antes de llenarse, con total seguridad el primero que se creó ya ha desaparecido, es decir, se ha calculado el tiempo que tarda en llegar al final de la pantalla el láser y eso permite que la nave siempre tenga al menos una posición libre en el array de láseres para poder activar un láser. En una primera aproximación y codificación, la señal de activar láser tenía que estar a '1' un tiempo bastante prolongado para que funcionase, lo que provocaba que si algo colisionaba con el láser cerca de su nacimiento, este láser no desaparecía, pues la señal de activa seguía a '1'. Al principio no nos preocupó demasiado pues el único caso en que esto fallaba era cuando un láser de marciano chocaba con uno de la nave en las inmediaciones del nacimiento del láser de la nave. Pero al añadir las barreras que protegen a la nave, los láseres de nave atravesaban todas las capas de la barrera y seguían vivos. Al final conseguimos cambiar la implementación logrando que la señal de activar el láser estuviese a '1' un sólo ciclo del reloj más rápido, corrigiendo el error.

- **Codificadores de prioridad formados a base de celdas:** Han jugado un papel fundamental en el desarrollo del proyecto pues se han utilizado para conseguir toda la información posible del array circular de láseres y de la matriz de marcianos. Además hemos aprovechado el hecho de que disponíamos de toda esta información para implementar muchos cálculos dinámicos.

La información que obtenemos con el codificador de prioridad del array de láseres es si en un determinado momento hay que pintar un láser y cuál es el primer elemento del array que tiene la señal de pintar a uno y la usamos para evitar tener una or enorme de todas las señales pinta_laser de cada láser del array y para saber inmediatamente cuál es el láser que se está pintando.

Para la matriz de marcianos, se ha implementado de tal manera que cada columna tiene un codificador de prioridad lineal con sus salidas, las cuales son la entrada de otro codificador de prioridad que tiene por salida toda la información. Hay dos codificadores implementados de esta manera, el primero recibe la matriz de si se pinta o no, funciona de manera análoga al láser, tiene por salidas, si se pinta o no algún marciano y en caso de que sí, qué coordenadas tiene. El segundo codificador recibe como entrada la matriz de '0's y '1's que indican si el marciano correspondiente está activo o no y recibe el número aleatorio para disparar. Lo que se obtiene de este codificadores es:

- El número de columnas que están activas. Se utiliza para que el divisor secuencial de los números aleatorios haga el módulo correspondiente.

- Primer marciano activo de cada columna empezando por abajo. Se utiliza para que cuando dispara una columna, dispare el primer marciano activo. También se utiliza

para que al perder debido a que los marcianos lleguen abajo, no se pierda cuando la matriz llegue a una posición prefijada, sino cuando el primer marciano activo llega abajo.

- Primera columna por la izquierda que tiene algún marciano vivo (Ídem por la derecha). Se utiliza para el cálculo dinámico con la pared y el cambio de dirección logrando que los marcianos cambien de dirección no en una dirección prefijada, sino cuando un marciano vivo llega al borde

- Utilizando el número aleatorio N que está entre 0 y columnas_vivas - 1, saca una salida que indica la N-ésima columna viva, saltándose las columnas que están muertas. Se utiliza para el disparo aleatorio.

- **Teclado:** El teclado es un componente muy sencillo que nos ha dado multitud de problemas debido a un mal diseño inicial. En un primer momento, utilizando el teclado realizado durante las clases, cuando se pulsaba una tecla, por ejemplo, la de mover la nave a la derecha, ésta se mantenía en el registro de teclado y la nave seguía moviéndose aunque se dejase de pulsar. Para evitar esto, con la recomendación del profesor, al teclado se le añadió una señal de borrar el registro y cada componente cuando había recibido una tecla que iba a utilizar mandaba una señal al teclado para borrar el registro. Esta implementación, además de entorpecer la jugabilidad, pues había que mover la nave pulsando sucesivas veces la tecla de movimiento, acabó dando múltiples problemas a la hora de introducir nuevas funcionalidades que involucraban al teclado. Al final decidimos utilizar el código de depresión que envía el teclado cuando se deja de pulsar una tecla, de tal manera que la implementación final consta de un registro con desplazamiento, y otro auxiliar, que es la salida del componente teclado (este auxiliar es para evitar que durante el desplazamiento dé casualidad en un estado intermedio de desplazar se envíe una tecla incorrecta). El registro auxiliar se actualiza cuando un contador indica que han pasado 11 ciclos. Cuando detecta que ha recibido el código F0, desecha los siguientes 11 bits, que se corresponderían con la tecla que se ha dejado de pulsar y en su lugar el registro se borra.

- **Disparos aleatorios de los marcianos:** Lo más sencillo para esta parte habría sido utilizar el reloj rápido para obtener un número y haber hecho que la columna de marcianos correspondiente a ese número disparara, pero dado que el número de columnas ni es ni tendría por qué ser potencia de dos, al coger este número pseudoaleatorio del reloj, había muchas posibilidades de que el número no estuviese en el rango adecuado y entonces cuando tocase disparar, no dispararía ningún marciano. Lo mismo ocurriría si se mata a una columna entera de marcianos y el número dice que tendría que disparar esa columna muerta. En ese caso, la opción habría sido, o no disparar, o que dispararse la siguiente columna lo cual implicaría una pérdida de la aleatoriedad pues la columna siguiente tendría más posibilidades de disparar que las demás.

Para evitar todos estos problemas y realizar disparos lo más aproximados a una aleatoriedad real, hemos utilizado un componente que en base a una semilla que obtiene al iniciar el nivel (que se obtiene del contador que a la vez se usa para el divisor del reloj de las fps) y que cada vez que se inicia un nivel es diferente devuelve un número aleatorio de 32 bits.

Este componente funciona con ciertos desplazamientos y puertas xor (ver bibliografía). Una vez que disponemos del número aleatorio de 32 bits, con un divisor secuencial, muy parecido al realizado durante las prácticas de clase pero modificado para obtener el módulo y no el cociente, obtenemos el módulo del número aleatorio de 32 bits dividiendo entre el número de columnas que quedan vivas, información que nos da el codificador de prioridad. Así tenemos pues, un número aleatorio entre 0 y el número de columnas de marcianos vivas - 1. Con este número N y usando de nuevo los codificadores de prioridad, hacemos disparar la N-ésima columna que está viva, logrando así que de manera dinámica cada vez que hay que disparar obtenemos un número aleatorio para el número de columnas vivas, cada columna tiene las mismas posibilidades de disparar que las demás, y siempre se dispara (no como en el caso sencillo en el que a veces si el numero aleatorio no estaba en el rango, no se disparaba).

- **Diferentes tipos de marcianos:** Inicialmente codificamos un solo tipo de marciano con un estado activo o desactivo y a petición del profesor nos disponíamos a hacer dos tipos los cuales tuviesen imágenes diferentes. Al final fuimos más allá y además de hacer dos tipos de marcianos (los del tipo 1 y 2) con formas diferentes, creamos un tercer tipo de marciano que es más fuerte, en el sentido de que dispone de tres vidas y que cuando le toca disparar, dispara cinco láseres a la vez. Con esto viene unido la consiguiente codificación de tipos y estructuración de los módulos para que de manera dinámica, cuando le toca disparar a un marciano el juego decida si disparar un láser o cinco en función del tipo de marciano que dispara y también la estructuración necesaria para que cuando un láser colisiona con un marciano, en vez de desaparecer, el juego acceda al tipo del marciano y a sus vidas restantes y decida si desaparece o todavía le quedan vidas o en caso de que sea un marciano débil desaparezca simplemente. También esto conlleva que la función de pintar sepa en todo momento qué tipo de marciano está pintando, pues tienen formas diferentes y en caso de ser fuerte, cuántas vidas tiene pues los marcianos se pintan de un color diferente en función de sus vidas.

- **Diseña tu propio nivel:** Creemos que es de especial importancia destacar esta parte, construida a última hora porque demuestra la buena estructuración realizada a lo largo de todo el código y el buen establecimiento de constantes que han hecho nuestro código muy escalable. Esta parte permite tener una matriz de tipos de marcianos que cambia dinámicamente en base a lo que el usuario modifica en la parte de “tu nivel” con lo que se logra que se pueda jugar a cualquier nivel posible, sólo cambiando el tipo de marciano que va a salir en el nivel. Y no sólo se cambia la imagen del marciano, si uno pasa de ser de los dos primeros tipos a ser fuerte, el marciano tendrá sus tres vidas y disparará cinco láseres cuando le toque disparar.

Al igual que esta modificación, se podrían haber realizado otras cambiando sólo pequeñas partes de código como un nivel de supervivencia infinita (que no se han implementado por falta de tiempo) en el que si matas a todos los marcianos, una nueva matriz de marcianos “nace” en la posición inicial de los marcianos y el juego se acaba cuando la nave muere, siendo el objetivo conseguir el máximo número de puntos.

- **Imágenes desde RAM** (ver bibliografía): Las imágenes originales de los marcianos han sido obtenidas de los sitios web indicados en la bibliografía. La mayoría han sido retocadas personalmente por nosotros con Paint para adecuarlas a nuestras necesidades estéticas, y el resto han sido creadas desde cero, también con Paint. Para convertir estas imágenes en archivos .data, se guardan en formato .bmp de 256 colores y se procesan con un programa en C++, que hemos modificado previamente para habilitar la exportación de las imágenes al formato que necesitamos, ya que dicho programa no traía esta característica inicialmente. Los archivos .data generados están formados bien por un valor de '0' o '1' para cada píxel de la imagen, para imágenes que sólo tienen dos colores, o bien por vectores de nueve componentes de '0' y '1', que representan la codificación RGB de nueve bits de cada píxel, para las imágenes que tienen muchos colores. Ya con los .data en nuestro poder, los metemos cada uno en una RAM, mediante una función VHDL, y de ahí vamos a sacar los datos de las imágenes para mostrarlos por pantalla.

4. …pero no tanto

En esta sección enumeramos los bugs y errores conocidos que presenta el proyecto:

- Al moverse las imágenes que son cargadas de memoria, aparece un rastro.

- Si la nave se mueve durante mucho tiempo sin realizar ningún disparo, puede llegar a disparar espontáneamente.

5. Bibliografía

Componente que dada una semilla en cada ciclo devuelve un número aleatorio de 32 bits. Proporcionado por Daniel Bascones García. Nos proporcionó una implementación de la siguiente función en C++

```
int t, w, y;

t = x ^ (x << 11);

y = w ^ (w >> 19) ^ (t ^ (t >> 8));

x = (x << 32) & (long long int)y;

return y;
```

donde x es una variable entera de 128 bits que es la concatenación de los cuatro últimos números aleatorios obtenidos

Cargar imágenes:

El programa original utilizado para pasar las imágenes de formato .bmp a formato .data ha sido obtenido de la página web: http://www.widget-101.com. Las imágenes de los marcianos se han obtenido de www.spriters-resource.com.

6. Trabajo individual

Para esta sección hemos decidido mostrar en forma de tabla los sucesivos Items/Subitems realizados y su/s ejecutor/es en las fases de diseño, codificación y prueba y depuración. Es la siguiente:

Item / Subitem	**Diseño**	**Codificación**	**Prueba y depuración**
Teclado inicial. Si se deja de pulsar la nave sigue moviéndose	David Mario	David	David
Nave	David Mario	David	Mario David
Láser	Mario David	Mario	Mario
Array de láseres	Mario David	Mario	Mario David
Marciano débil inicial	Fernando	Fernando	Fernando
Teclado intermedio, con señal en la nave y el laser de borra_tecla	David Mario	David	David Mario
Codificador de prioridad lineal	David Mario	Mario	Mario
Matriz de codificador de prioridad con codificadores lineales	Mario David	Mario	Mario David
Matriz de marcianos	Fernando	Fernando	Fernando David Mario
Colisiones	David Mario	David Mario	David Mario
Disparo de marcianos	Mario David	David	David
Programa en C++ y función en VHDL para pasar una imagen BMP a .data y cargarla en una RAM	Fernando	Fernando	Fernando

Cambio en los codificadores para conseguir rebote dinámico, y que dispare el primer marciano vivo de la columna en vez de disparar abajo del todo	Mario David	Mario	Mario
Mejora de la nave con vidas y diferentes colores por cada vida	Mario David	David	David Mario
Marcador de vidas de la nave	Mario David	David	Mario David
Pintado de marciano desde memoria	Mario David	Mario David	Mario David
Teclado definitivo. Con estados ENVIAR y DESECHAR. Sin borra_tecla	David	David	David
Diferentes tipos de marcianos, marcianos fuertes con más vidas diferentes imágenes. Cinco láseres cuando disparan los fuertes	David Mario	Mario	David Mario
Pantalla inicial, gestión de niveles y estados de juego	David Mario	David	Mario David
Implementación de señales de nave_muerta, sin_marcianos y llegan_abajo para decidir cuando se pierde y se gana	Mario David	Mario	Mario David
Inicialización de la matriz de marcianos de forma diferente en cada nivel	Mario David	David	Mario David
Salida por pantalla de imágenes en los estados de pantalla inicial de game over y victory	Mario David	Mario David	Mario David
Barreras	Fernando	Fernando	Mario Fernando David
Divisor secuencial para hacer módulo dinámico y generar números aleatorios para los disparos	Mario David	David	David
Cambio en los codificadores de marcianos para una vez obtenido el número aleatorio, decidir quién dispara	Mario David	Mario David	Mario David
Implementación con lo anterior de disparos aleatorios	Mario David	David	Mario David

“Crea tu nivel” selección del nivel entero que se quiere jugar	David	David	David
En vez de sacar de ram los colores y forma de los marcianos, sacar de RAM una imagen con negro ‘0’ y blanco ‘1’ y poner un color plano para cada marciano. Aprovechando eso, con la misma RAM, diferentes colores del marciano fuerte en función de sus vidas	Mario	Mario	Mario
Al cargar imágenes de RAM rectangulares las colisiones con marcianos y nave se hacían al llegar un láser a dicho rectángulo. Cambio para que, aunque seguimos cargando las imágenes de RAMS rectangulares, la colisión se produzca con el objeto real	Mario David	Mario David	Mario David
Movimiento de brazos de los marcianos con cada movimiento	David Mario	Mario	Mario David
Marcador y sumador de puntos	Fernando	Fernando	Fernando Mario David
Movimiento de brazos del marciano seleccionado en la fase de crear tu nivel	Mario David	Mario David	Mario David
Corrección de visualización y retoques finales	Fernando Mario David	Fernando Mario David	Fernando Mario David