Universidad Carlos III de Madrid

Diseño de Circuitos Digitales

Implementación de Space Invaders en FPGA

Autores:
David Estévez Fernández
Sergio Vilches Expósito

Profesora:
Anna Vaskova

16 de diciembre de 2013

Índice

1. Introducción

El objetivo de esta práctica de laboratorio era realizar un diseño e implementación de un circuito digital con un juego inspirado en el arcade clásico "Space Invaders", en el lenguaje de descripción de hardware VHDL.

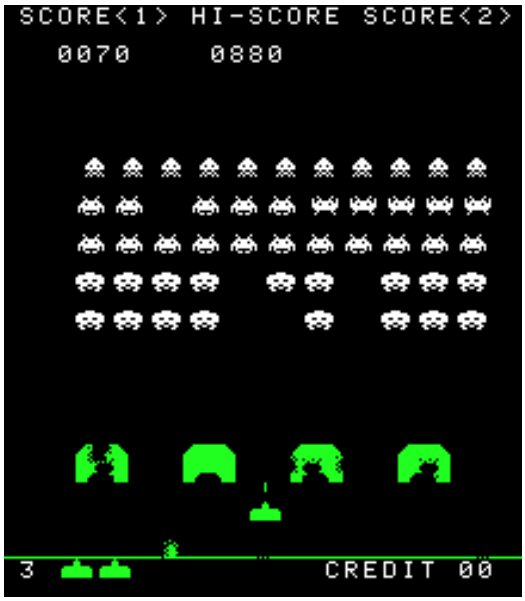

Figura 1: Pantalla del juego "Space Invaders" original

Este juego, que ha sido simplificado para la consecución de este laboratorio, consiste en una nave, controlada por el jugador y localizada en la parte inferior de la pantalla, que se enfrenta a una serie de "Invasores del espacio", que van descendiendo por la pantalla. El jugador gana si elimina a todos los invasores, y pierde si los invasores llegan a la parte inferior de la pantalla, invadiendo la Tierra.

Para su implementación hardware se dispone de una placa entrenadora Spartar 3E Starter Board, la cual posee una FPGA Spartan 3E (modelo xc3s500E-4 FG320) así como distintos periféricos, como LEDs, pulsadores, interruptores, un encoder, conector VGA, etc. Para nuestra implementación hemos utilizado algunos de estos periféricos, como los LEDs, el conector VGA y 2 interruptores, así como un pin de salida, al que hemos conectado un jack de audio con un filtro paso bajo, para los sonidos y dos de los conectores de expansión de la placa, a los que hemos conectado unos mandos diseñados y construidos por nuestros compañeros del año pasado, Ruy García y Nieves Cubo, que nos los han prestado amablemente.

El desarrollo software se ha llevado a cabo usando un gestor de versiones (git) para facilitar la colaboración entre los integrantes del grupo, y se encuentra disponible de forma libre y abierta en la siguiente dirección web:

`https://github.com/David-Estevez/spaceinvaders`

Se dispone también en el repositorio de una serie de etiquetas (tags) que referencian al código de cada una de las sesiones de laboratorio, de forma que es posible acceder al estado del proyecto en cada una de esas sesiones.

2. Funcionamiento general

El sistema final se muestra en la figura 2. El sistema cuenta con los bloques básicos requeridos por el enunciado de la práctica, así como una serie de bloques extra añadidos como mejora del diseño básico. Estos bloques se explicarán de forma detallada en la sección 3, junto a las razones tras su implementación y cómo se llevaron a cabo las pruebas de los distintos bloques.

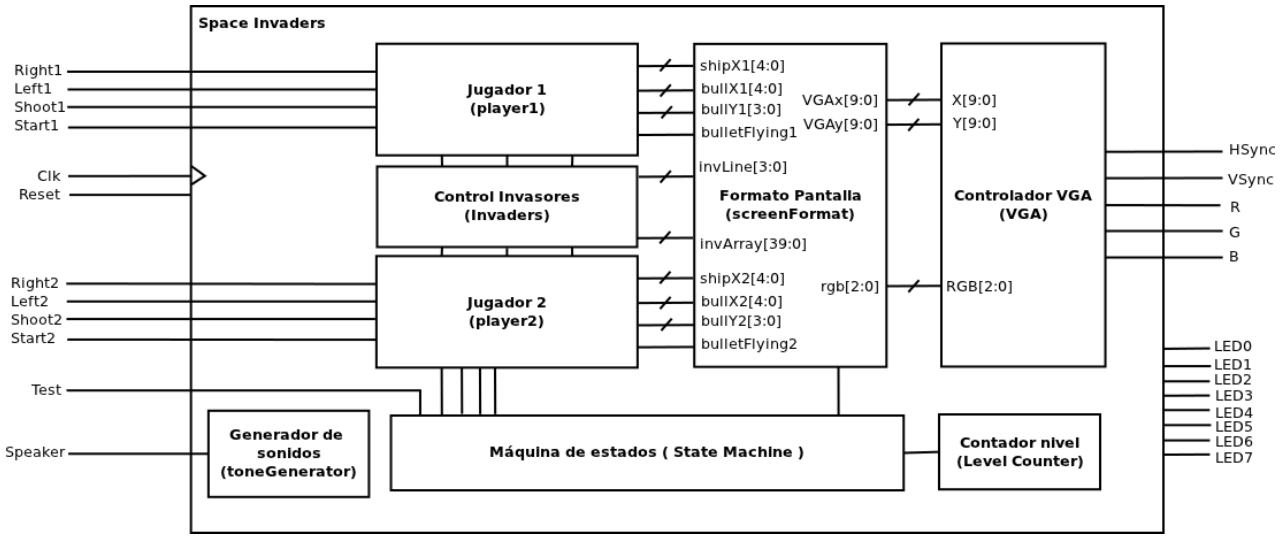

Figura 2: Diagrama de bloques del sistema completo. Algunas conexiones no se han indicado por claridad.

Los bloques básicos incluyen:

- **Controlador VGA**: Encargado de generar la señal de control de la pantalla.
- **Formato Pantalla**: Su función es dibujar los distintos elementos del juego en su correspondiente posición de la pantalla.
- **Control Invasores**: Controla el movimiento de los marcianos y su desaparición tras un disparo.
- **Control nave**: Controla el movimiento de la nave del jugador.
- **Control disparo**: Controla el movimiento del disparo.
- **Detector de flancos**: Convierte la señal de entrada de los pulsadores en un pulso apto para el circuito.
- **Máquina de estados**: Controla el flujo del juego.

Además de esos bloques, se han implementado como mejora los siguientes:

- **Jugador**: Agrupa varios elementos del jugador, como la nave, el disparo y los detectores de flancos, además de contar la puntuación de ese jugador. Es de gran utilidad para incluir dos jugadores en el juego.
- **Sintetizador de sonidos**: Genera sonidos de estilo *arcade* que acompañan diferentes acciones del juego.

- **Contador de nivel**: sirve para seleccionar la velocidad de juego y dificultad de los marcianos dependiendo de las pantallas ganadas.
- **Otros**: Además de los bloques anteriores, se han modificado algunos de los bloques básicos para extender su funcionalidad.

El mecanismo del juego es el siguiente. Al accionar el interruptor de test en cualquiera de los estados se mostrará en la pantalla un patrón de prueba (tablero de ajedrez) y la partida se reiniciará. El juego se inicia en el estado inicial, en el que se muestra la pantalla de juego estática. En este estado inicial un segundo jugador puede activar la señal de inicio para sumarse a la partida, de forma que pueden jugar 1 ó 2 jugadores. Al pulsar el botón de inicio el jugador 1 el juego se inicia, los invasores comienzan su movimiento y los jugadores pueden moverse y disparar una bala cada vez.

Si los invasores alcanzan el final de la pantalla, la partida se termina y los jugadores pierden. Si, por el contrario, los jugadores consiguen alcanzar a todos los invasores, superan la pantalla y el nivel se incrementa en 1, pudiendo jugar el siguiente nivel hasta un máximo de 8 niveles, en los que la dificultad de los marcianos y la velocidad de los mismos aumenta. Existen 3 tipos de invasores, verdes, amarillos y blancos, que requieren 1, 2 y 3 disparos respectivamente para morir. Cada disparo acertado aumenta la puntuación de ese jugador en 1 unidad. Si se pierde la partida en cualquiera de los niveles los jugadores deben empezar de nuevo desde el primer nivel (no se dispone de vidas extra).

3. Explicación de los bloques

3.1. Controlador de pantalla VGA

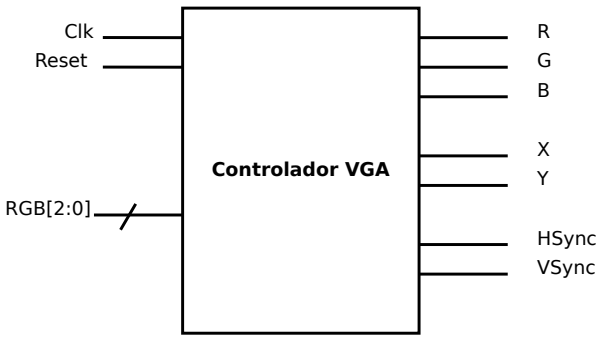

Figura 3: Interfaz del controlador de pantalla VGA

El controlador de pantalla VGA se encarga de generar el patrón de ondas necesario para mostrar los gráficos del juego en una pantalla externa.

3.1.1. Descripción Funcional

El estándar VGA usa tres señales analógicas para codificar el color de cada píxel (R, G y B en la figura 3) y dos señales de sincronización que indican el fin de línea $HSync$ y fin de pantalla $VSync$). Para la implementación del controlador se ha usado una frecuencia de 25 MHz, que genera una resolución de 640 x 480 píxeles a una tasa de refresco de 60 Hz.

Para formar la imagen deseada en la pantalla, el controlador VGA accede a la información de color de cada píxel de manera similar a una memoria: las señales x e y indican la posición del píxel al bloque Formato VGA, que devuelve el valor del color a través del bus RGB.

Dado que la placa de desarrollo *Spartan 3E Starter Kit* contiene un conversor digital a analógico (DAC) de un sólo bit para las señales R,G y B, sólo es posible mostrar 8 colores diferentes en la pantalla.

3.1.2. Implementación

Las señales VGA *HSync* y *VSync* son siempre periódicas y están controladas por dos contadores en cascada: *HCount* (con valores de 0 a 800) y *Vcount* (0 a 521). El contador horizontal *HCount* proporciona un marco temporal para mostrar los 640 píxeles que hay en cada línea de la pantalla, terminando cada ciclo con pulso bajo en *HSync*, indicando que el monitor debe comenzar una nueva linea.

Cada vez que se termina una línea, el contador vertical *VCount* aumenta su valor hasta completar una pantalla (480 líneas). Entonces, un pulso bajo en *VSync* indica al monitor que debe comenzar una nueva imagen.

Estos contadores usan un *prescaler* de 1 bit, ya que la señal *clk* tiene una frecuencia de 50 MHz pero la interfaz con la pantalla está configurada a 25 MHz.

3.2. Formato VGA

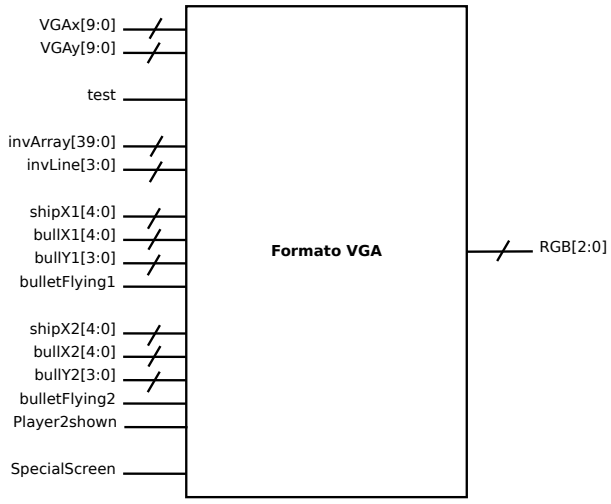

Figura 4: Interfaz del bloque Formato VGA

El bloque *Formato VGA* se encarga de traducir una serie de señales binarias procedentes de los invasores, naves y balas a figuras (*sprites*) que mostrar en la pantalla.

Descripción del proceso

Para facilitar esta tarea, la pantalla se divide en *macropíxels*, celdas de 32 x 32 píxeles que contienen un sólo sprite. Así, el espacio de juego queda reducido a una matriz de 20 x 15 macropíxels.

El proceso de dibujo de la pantalla pude resumirse de la siguiente manera:

1. El controlador VGA solicita la información RGB de un píxel.
2. *Formato VGA* traduce la posición (x,y) a su correspodiente macropíxel.
3. Con la información procedente del resto de entidades, se selecciona qué elemento hay que mostrar en ese macropíxel.
4. Se accede a la matriz del sprite correspondiente, obteniedo el valor del píxel en concreto que se ha de dibujar.

Codificación de la posición de los elementos

La información posicional de los invasores, balas y naves se envía desde los diversos bloques a través de buses, que codifican la información de manera diferente:

- **Bala**: Cada una de las dos balas transmiten su posición en coordenadas de macropíxels, codificadas en dos buses de 5 y 4 bits respectivamente ($bullX_n$, $bullY_n$). La señal $bulletFlying_n$ indica si la bala de cada uno de los jugadores está activa.

- **Nave:** Como la nave sólo puede moverse en la última línea de la pantalla, la señal $shipX_n$ basta para indicar su posición en el juego. Adicionalmente, la señal $player2shown$ indica si hay que mostrar la nave del jugador 2.

- **Invasores:** La posición de cada uno de los invasores, así como su tipo, se codifica en las señales $invLine$ y $invArray$. $invLine$ define en qué linea se mueven los invasores, mientras que $invArray$ codifica el estado de los invasores (activo o abatido) y su nivel (más información en Invasores).

3.3. Detectores de flancos

Para la interfase del diseño con los pulsadores diseñamos dos detectores de flancos, uno estándar y otro antirrebotes, de forma que llegase un sólo pulso a los demás bloques del sistema por cada pulsación del jugador.

En el diseño final sólo hemos añadido el detector de flancos con antirrebote, debido a su mejor respuesta ante la entrada oscilante de los pulsadores reales.

3.3.1. Detector de flancos estándar (edgeDetector)

El detector de flancos estándar es un circuito muy simple. Se tienen dos biestables, uno almacenando el estado actual de la señal de entrada, y el otro almacenando el estado de la señal en el tiempo t-1.

Estas dos señales se comparan, y si hay un nivel alto en el estado actual, pero no en el anterior, entonces ha llegado un flanco, y el detector emite un pulso a la salida.

Banco de pruebas

En el banco de pruebas de este bloque se simuló una entrada de una duración superior a 1 periodo de reloj, con la señal de habilitación ("Enable") activada y desactivada, de forma que se comprobó que la salida al llegar un flanco con el enable activado era de un pulso exacto.

3.3.2. Detector de flancos con antirrebote (edgeDetectorDebounce)

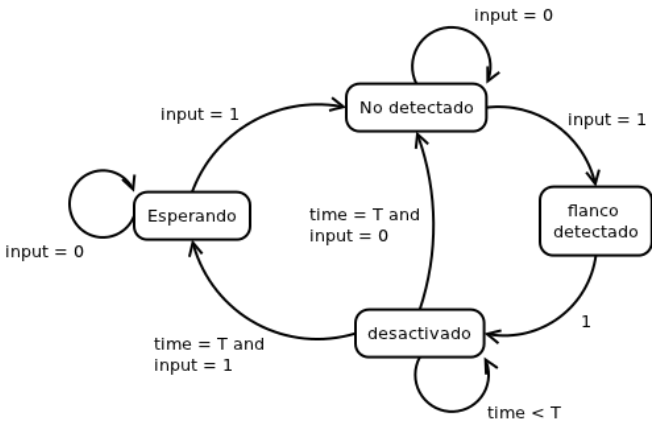

Figura 5: Máquina de estados

El detector de flancos con antirrebote es un detector de flancos estándar al que se le ha añadido un temporizador que lo desactiva durante un cierto periodo de tiempo tras detectar un flanco, para evitar las oscilaciones producidas por los rebotes de los pulsadores.

Su diseño está basado en una máquina de estados simple (figura 5) que consta de cuatro estados: "no detectado", "flanco detectado", "desactivado" y "esperando". La máquina empieza en el estado "no detectado", y permanece en él mientras la entrada valga 0. Cuando recibe un 1, pasa al estado "flanco detectado", y la salida se pone a 1 hasta el siguiente flanco de reloj, en el cual pasa al estado "desactivado", poniendo la salida a 0. Mientras la máquina se encuentra en este estado, un temporizador cuenta el tiempo transcurrido, y bloquea el paso al estado siguiente. Cuando el tiempo deseado, en nuestro caso 100 ms, ha transcurrido, la máquina de estados pasará al estado "no detectado" si la entrada es 0 o a "esperando' si es 1. La máquina pasará entonces al estado de "no detectado" cuando reciba un 0 lógico.

Banco de pruebas

Para probar este bloque se diseñó un banco de pruebas que simulaba una entrada ruidosa, con varias oscilaciones entre nivel alto y bajo, de forma que se comprobase que el detector sólo se activaba una vez tras el flanco, con una duración de un periodo de reloj.

3.4. Máquina de estados principal

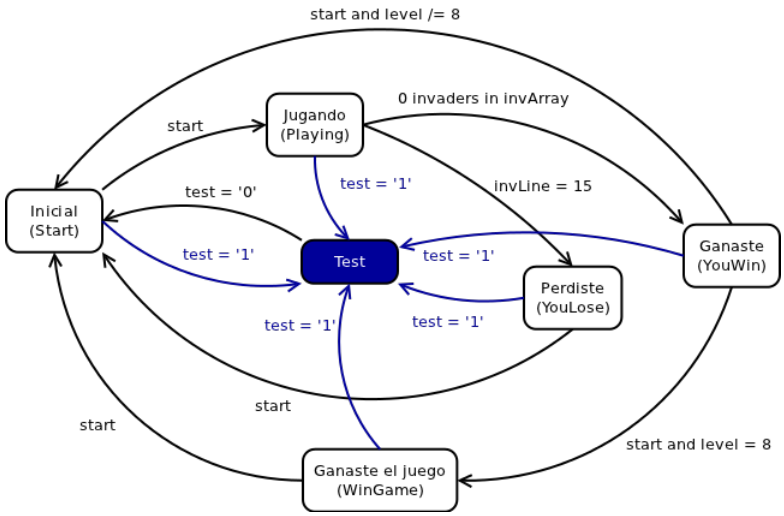

Figura 6: Diagrama de estados

El control del flujo del juego y el funcionamiento de los distintos bloques se lleva a cabo con una máquina de estados. Algunas señales, cuya lógica de activación es más compleja, se activan a partir de las señales de estado en un proceso independiente, que se explicará a continuación. La máquina de estados posee los siguientes estados:

- **Test**: estado de prueba. Se llega a él desde cualquier estado tras activarse la señal "test" y cuando la máquina está en este estado se reinician todos los valores de la partida y se muestra en la pantalla un patrón de tablero de ajedrez.
- **Inicial (Start)**: en este estado la máquina está esperando que el jugador inicie la partida. Los bloques de ambos jugadores se reincian (excepto las puntuaciones) y desactivan, y el bloque de los invasores se reinicia y permanece parado. Al recibir una señal de inicio ("p1StartPulse") del jugador 1, se pasa al estado "jugando (playing)".
- **Jugando (Playing)**: este estado corresponde al juego propiamente dicho. En él se inicia el movimiento de los invasores y se habilitan los controles de los jugadores. Desde este estado se pasará a "ganaste (YouWin)" si se han eliminado todos los invasores y el vector "invArray" sólo contiene '0's, o a "perdiste (YouLose)" si los invasores consiguen llegar al final de la pantalla, es decir, si invLine = 15.
- **Perdiste (YouLose)**: este estado muestra la pantalla de juego perdido. Se permanece en este estado mientras no se reciba una señal de inicio ("pxStartPulse") de cualquiera de los dos jugadores (que lleva la máquina al estado inicial), y en él los invasores y los jugadores permanecen reiniciados y deshabilitados.
- **Ganaste (YouWin)**: este estado se muestra la pantalla de juego ganado. Se permanece en este estado mientras no se reciba una señal de inicio ("pxStartPulse") de cualquiera de los dos jugadores (que lleva la máquina al estado inicial siempre y cuando no se halla superado el último nivel), y en él los invasores y los jugadores permanecen reiniciados y deshabilitados. Al llegar a este estado se habilita una señal que hace que el contador de nivel se incremente una unidad, haciendo que al volver al estado inicial los invasores que aparezcan sean de una dificultad superior.

- **Ganaste el Juego (WinGame)**: se llega a este estado tras haber finalizado el último nivel, y en él se muestra la pantalla de juego completado. Este estado reinicia tanto los invasores como los jugadores, y se queda a la espera de una señal de inicio ("pxStartPulse") de cualquiera de los dos jugadores para iniciar una nueva partida.

Además de las señales que se controlan de forma combinacional dentro de la máquina de estados, se han añadido algunas señales dependientes del estado o de la transición, cuya lógica de activación es ligeramente más compleja, en procesos independientes. Más concretamente, se ha añadido un biestable que contiene la señal de activación del jugador 2, que se acciona cuando nos encontramos en la pantalla de inicio de nivel ("Start") y recibimos una señal de inicio del jugador 2 ("p2StartPulse), y se reinicia al perder la partida o ganar el juego (una vez el jugador 2 se une a la partida, permanece habilitado hasta que la partida acaba). También se ha añadido un proceso que reinicia las puntuaciones y el nivel cuando se produce una transición desde un estado que no sea "Ganaste (YouWin)" hasta el estado "Inicial (Start)".

Banco de pruebas

Al estar la máquina de estados dentro de la entidad "SpaceInv" no se diseñó un banco de pruebas específico para ella, sino que se probó con todo el sistema. Para agilizar la simulación se modificaron todos los temporizadores a unos valores mucho más bajos que en condiciones de juego reales, de forma que se apreciase el funcionamiento en un tiempo más breve.

Para probar los estados de "ganaste" y "perdiste", debido a la imposibilidad de codificar una partida entera en el banco de pruebas, se modificaron las posiciones y estado de los invasores, de forma que al poner el vector "ìnvArray" a 0 se pudo comprobar el estado "ganaste" y colocando inicialmente los marcianos en la línea 14 sólo hubo que esperar 20 movimientos de los mismos para que los marcianos llegasen a la línea 15 y perdiese el juego.

3.5. Jugador (Player)

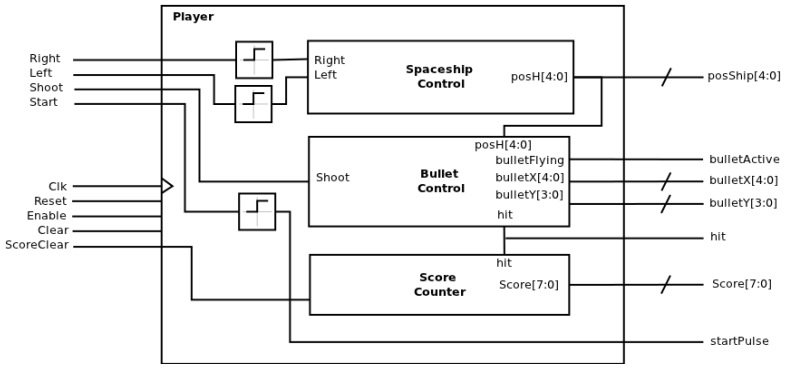

Figura 7: Diagrama de bloques del Jugador (player)

Con el fin de agrupar todas las funciones de un jugador para mayor claridad, hemos creado un bloque Jugador (Player) que incluye toda la funcionalidad del jugador, de forma que además podemos instanciar varios jugadores de forma sencilla. Esto ha facilitado el añadir un segundo jugador a nuestro juego, simplemente instanciando este bloque 2 veces.

El bloque Jugador (figura 7) incluye: tres detectores de flancos con antirrebote para las señales de movimiento a izquierda y derecha y de inicio, un control de la nave, un control del disparo y un contador para la puntuación.

Una serie de salidas comunican las señales más importantes (posición de nave y bala, puntuación, etc) al resto de bloques del sistema.

También posee una serie de señales de control del bloque: Reset asíncrono, Clear síncrono (pone todo a su estado inicial excepto la puntuación), un Clear síncrono para la puntuación y una entrada Enable para su habilitación, que afecta a todos los bloques excepto al detector de flancos de inicio, que siempre está activo.

La implementación en VHDL simplemente instancia los bloques necesarios y realiza las conexiones entre ellos. El contador de la puntación, sin embargo, se ha implementado como un proceso, que incrementa una unidad el contador cada vez que le llega una señal de marciano alcanzado por un disparo (hit).

Banco de pruebas

Para probar este bloque no se dispuso un banco de pruebas específico, sino que se comprobó su funcionalidad dentro del sistema "SpaceInv", al haber probado ya gran parte de los bloques internos de forma independiente. Por tanto, las pruebas de este bloque se limitaron a comprobar que al agrupar todos los bloques individuales se conservaba su funcionalidad intacta.

3.6. Nave

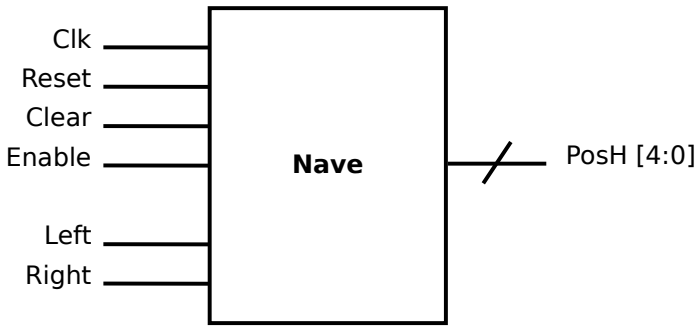

Figura 8: Interfaz de la entidad Nave

La entidad *nave* controla el movimiento de cada usuario sobre la pantalla de juego. Es parte de la entidad *Jugador*.

El movimiento de la nave es controlado por las señales *left* y *right*, siempre y cuando no se haya alcanzado los límites de la pantalla. La señal *enable* (conectada a un temporizador externo) limita la velocidad de movimiento de la nave.

La posición de la nave se codifica en la señal de 5 bits *posH*, que se usa tanto para mostrarla en la pantalla como para indicar a la entidad *Bala* la posición de disparo.

3.6.1. Testbench

El banco de pruebas `spaceship_tb.vhd` comprueba el correcto funcionamiento de la nave en cualquier dirección y que no sobrepasa los límites de la pantalla.

3.7. Bala

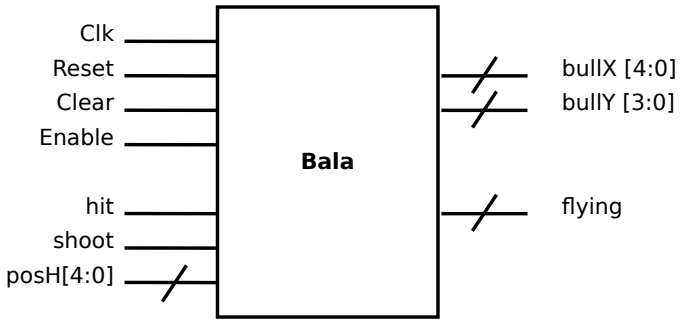

Figura 9: Interfaz de la entidad Bala

Cada entidad *player* incluye una entidad *bala* que, disparada desde la nave correspondiente, puede dañar o destruir invasores.

En el instante en el que se detecta una pulsación de disparo (señal *shoot*), la bala copia la posición horizontal de la nave y se sitúa justo encima de ella. A partir de entonces, la bala asciende verticalmente a una velocidad marcada por la señal *tick*, que procede de un temporizador externo.

La bala continúa moviéndose hasta que llega al final de la pantalla o alcanza a un invasor. Este último caso se detecta gracias a la señal *hit*, que procede de la entidad *invaders*. Tras esto, la señal *flying* pasa a nivel bajo, indicando que la bala ha desaparecido y no debe ser mostrada en pantalla.

Debido a que no se puede disparar otra bala hasta que la anterior no ha desaparecido, esta entidad no requiere detector de flancos.

3.8. Control de los invasores (Invaders)

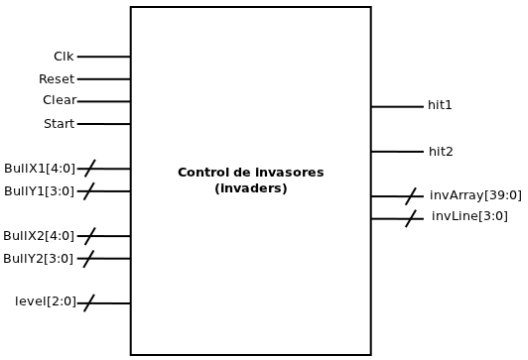

Figura 10: Diagrama de bloques del Control de los invasores (Invaders)

El bloque de control de los invasores se encarga tanto del movimiento de los invasores a lo largo de la pantalla, como de comprobar la posición del disparo y matar a los correspondientes marcianos. Para ello, además de las entradas de control típicas (Clk, Reset, Clear) posee 2 conjuntos de entradas que le proporcionan la posición y estado de cada una de las 2 balas del juego (una por cada jugador).

Para ello hemos definido una matriz constante que almacena la disposición inicial de los marcianos para cada uno de los 8 niveles implementados. Esta disposición se selecciona a través de la entrada "level", que indica el nivel actual.

Existen tres tipos de invasor, dependiendo del número de vidas que tengan (verde, 1 vida; amarillo, 2 vidas y blanco, 3 vidas) que se codifican usando 2 bits, siendo 00 un hueco sin marciano, 01 un marciano verde, 10 un marciano amarillo y 11 un marciano blanco, de 3 vidas. Estos marcianos se guardan en un vector de 40 elementos, 2 elementos por casilla de la pantalla en la que puede haber un marciano, que como se ha mencionado previamente, se selecciona dependiendo del nivel actual.

Este bloque también posee dos temporizadores que controlan la velocidad de avance de los invasores. Dependiendo del nivel se selecciona uno u otro, teniendo dos velocidades posibles (normal y rápida). Se seleccionó este diseño en lugar de un sólo temporizador con 8 velocidades (una por nivel) debido a que al haber aumentado la dificultad añadiendo marcianos más poderosos no tenía mucho sentido incrementarla aún más aumentando la velocidad en sólo 8 niveles, y había recursos suficientes en la FPGA como para poner dos teporizadores, cuyo diseño ya estaba realizado en prácticas anteriores, en lugar de uno nuevo con varias comprobaciones de cuenta.

La implementación en VHDL posee una señal "start", que sirve de señal de disparo, cambiando el estado de un biestable interno "moving" que indica al bloque que los marcianos han comenzado a moverse. Una vez en movimiento, cada pulso de reloj (en el que el temporizador lo permita), el vector de marcianos sufre un desplazamiento de 2 posiciones hacia la derecha. Si una de las 2 últimas posiciones del vector contiene un 1, es decir, si un marciano ha llegado al final de la línea, la posición en el eje Y se incrementa una unidad, y se invierte el sentido del desplazamiento. Este proceso se repite hasta que todos los marcianos hayan muerto, o hayan llegado a la última posición.

Por último se comprueba si el disparo coincide con la posición de alguno de los marcianos, y si es así, decrementa la vida del marciano en 1 unidad (un marciano blanco pasaría a ser amarillo, uno amarillo a ser verde, y uno verde moriría) y envía un pulso a través de la señal "hit1" o "hit2", dependiendo de qué jugador haya alcanzado al marciano.

3.9. Sintetizador de sonidos

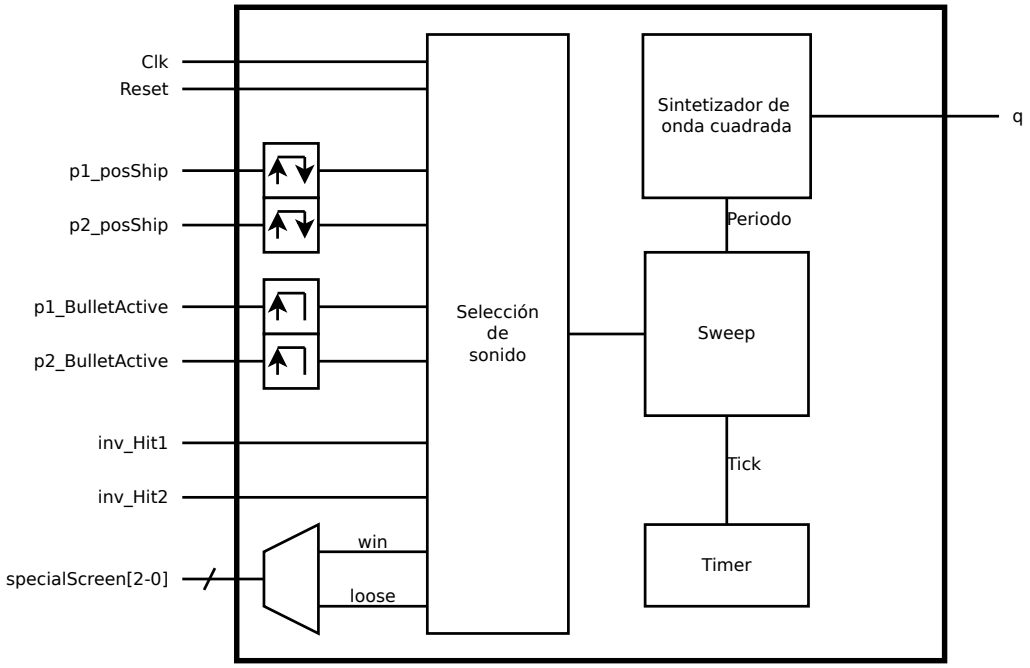

Figura 11: Diagrama de bloques del sintetizador de sonido

Para emular el videojuego original, se ha implementado un sintetizador digital de sonidos que produce melodías que acompañan diferentes acciones del juego: movimiento de la nave, disparo, alcance de invasores y pantallas de victoria y derrota.

3.10. Implementación

Como puede verse en la figura 11, la sintetización del sonido consta de cuatro pasos:

1. **Detección de acción**: Para no requerir señales adicionales dentro de la entidad *spaceInvaders*, la detección de acciones se decodifica a través de las señales intermedias *p1_posShip, p2_posShip, p1_BulletActive, p2_BulletActive, inv_Hit1, inv_Hit2* y *specialScreen*, con la ayuda de detectores de flanco y decodificadores.
2. **Selección de sonido**: Cada sonido del sintetizador se caracteriza por el rango de frecuencias que barre (codificado como *startPeriod* y *endPeriod*). Este bloque selecciona estos valores dependiendo de la acción que da orígen al sonido.
3. **Barrido de frecuencia**: El rango de frecuencias seleccionado se divide en 32 escalones de 30 ms de duración. En cada instante, este bloque controla la frecuencia a la que oscila el sintetizador de onda cuadrada.
4. **Sintetizador de onda cuadrada**: Basado en un temporizador de período regulable, el sintetizador genera una onda cuadrada de ciclo de trabajo 50 %.

Una vez generada la señal cuadrada, un filtro RC analógico externo limita el ancho de banda a 1 KHz, suavizando la señal y haciendo el sonido menos estridente.

3.10.1. Testbench

El testbench comprueba el funcionamiento del sintetizador aislado del resto del videojuego, simulando una de las señales de acción.

4. Posibles mejoras

Incluso con la funcionalidad extra que se ha añadido a esta implementación, sigue habiendo muchas diferencias con el videojuego original. Muchas de ellas se deben a la recomendación del uso de celdas de 32x32 píxeles que se detalla en el manual de prácticas. Esto fuerza a que el movimiento de la nave, bala e invasores sea escalonado en vez de contínuo.

Otras diferencias con el juego original son más fáciles de aplicar y han sido incluidas por diferentes compañeros de prácticas: varias líneas de invasores, disparos desde los invasores hacia la nave o animaciones de los iconos.

5. Conclusión

Se ha conseguido implementar con éxito una versión multijugador de Space Invaders usando lógica programable y programación estructurada en VHDL.

Esta práctica ilustra el propósito de los lenguajes de descripción de hardware: para desarrollar el videojuego se han configurado más de 1000 LUTs (que equivalen a varios miles de puertas lógicas) y se ha conseguido que no haya ningún problema de realimentación (latches) ni funcionalidad no prevista (bugs). Esto sería imposible sin la abstracción que proveen los lenguajes de descripción de harware.

La forma con la que está organizada la práctica empuja a encapsular y probar entidades de diseño, conceptos básicos para poder reusar código y que el diseño crezca a paso seguro.