THHI

Touhou Highly Responsive to Prayers Hardware Implementation

Memoria del Proyecto

Autores: Angel Alonso

Xi Chen

Hussein Hassan

Daniel Gamo

Curso 2014/2015

Tabla de contenido

1 Que hemos hecho

El proyecto ha consistido en hacer un juego estilo breakout basado en el juego japonés Touhou Highly Responsive to Prayers. Hemos programado el juego en VHDL, usando la FPGA 3s1000 ft256 de Xilinx como hardware objetivo.

2 Funcionalidad – Especificaciones

El juego tiene cinco elementos "físicos": el personaje, la bola, las cartas, los puntos, las vidas.

2.1 Personaje

- Se puede mover horizontalmente en la base de la pantalla.
- Al comienzo de la partida y de cada nivel, el personaje aparece en el punto medio de la base de la pantalla.
- Puede golpear la bola cuando está cerca, aumentando la velocidad en un vector de 45º a hacia el noreste o noroeste según la dirección desde la que viniese.
- Si entra en contacto con la bola y no la golpea, pierde una vida.
- Al perder una vida se teletransporta al punto medio de la base de la pantalla, volviéndose invulnerable durante alrededor de medio segundo.
- Al ser invulnerable no puede perder vidas, pero si golpear la bola.
- Tiene un tamaño de alrededor de 33x39 píxeles.
- Los movimientos hacia derecha e izquierda causan una animación formada por dos sprites, que simulan la imagen del personaje corriendo.
- El ataque que golpea la bola inicia una animación formada por cuatro sprites.
- Al volverse invulnerable todos los gráficos del personaje cambian de color a un patrón intermitente de blanco y rojo.

2.2 Bola

- Se mueve parabólicamente en un área encerrada por los bordes inferior y laterales de la pantalla, y el borde inferior del panel de puntos situado en la parte superior de la pantalla.
- Comienza en próxima a la mitad de la pared derecha de la pantalla, con una ligera velocidad inicial hacia abajo a la izquierda.
- Pierde velocidad con cada colisión con los bordes del área, reduciendo su velocidad en cada eje un 25% y rebotando en la dirección reflexiva.
- Interacciona con el personaje como se ha indicado anteriormente, y con las cartas, volteando una carta cuando pasa por una vecindad suya (al menos un cuarto de la bola superpuesta a la carta).
- Tiene un tamaño de 25x25 píxeles.
- Al moverse realiza una animación de giro en la dirección natural a la dirección del movimiento.

2.3 Cartas

- Se encuentran estáticas en una banda horizontal centrada en la pantalla.
- Hay hasta 200 cartas a la vez en un nivel, organizadas en una cuadricula de 20 (ancho) por 10 (alto).
- Para cada nivel, la cantidad de cartas y su posición es aleatoria.
- Las cartas interactúan con la bola, volteando durante medio segundo.
- Tras voltear desaparecen, obteniendo una puntuación que depende del nivel.

- Tras voltear todas las cartas de un nivel, el nivel acaba y se avanza al siguiente nivel.
- Tienen un dibujo diferente para cada nivel.
- Tienen un tamaño de 32x32 píxeles.
- Tienen cuatro sprites, uno de ellos que representa el estado inicial de la carta; y otros tres que se muestran con la animación del volteo.

2.4 Puntos

- La puntuación inicial por carta volteada es de 500 puntos, y se multiplica por el nivel.
- Los puntos tienen un límite de 7 cifras decimales. Es decir, la puntuación máxima es de 9999999 puntos. Obviamente esta cifra no se puede superar debido al límite de tres niveles y 200 cartas por nivel.
- Aparecen en base decimal en un panel en la parte superior de la pantalla.

2.5 Vidas

- El jugador comienza con seis vidas.
- Las vidas se pierden cuando el personaje colisiona con la bola sin realizar la acción de golpear.
- No se pueden obtener vidas durante la partida.
- Las vidas a aparecen en forma de conjunto de sprites al lado izquierdo de los puntos.

2.6 Otros elementos

A parte de estos elementos, tenemos una pantalla de inicio, pantalla intermedia entre niveles, pantalla final y pantalla de fondo; y una melodía de fondo.
La melodía de fondo reproduce cinco canales de audio simultáneos a través del altavoz de los entrenadores del laboratorio.

3 Diagrama de componentes

El diagrama de macrocomponentes se encuentra en el archivo adjunto *.png

3.1 Chip

Módulo principal. Conecta los componentes entre si, controla el inicio y fin de niveles y la carga de cartas.

Puertos de entrada:

- main_clk: Señal de reloj principal de 50 MHz.
- rst: Señal de reset con activo en baja.
- ps2_clk: Señal de reloj del teclado PS2.
- ps2_data: Señal de datos del teclado PS2.

Puertos de salida:

- hsync: Señal de sincronización horizontal de VGA.
- vsync: Señal de sincronización vertical de VGA.
- rgb (9 bits): Vector de color RGB.
- tec (7 bits): Vector de pulsación de teclas de teclado con activo en alta.
- transmiting: Señal de transmisión de datos de teclado PS2.

- data_re: Señal de dato recibido desde el teclado PS2.
- error: Señal de dato erróneo de teclado PS2.
- pwm: Señal de onda melódica.

3.2 Personaje

Módulo del personaje, realiza los movimientos y ataque, la comprobación de colisión con la bola y el dibujo del personaje. Informa de su muerte a la FSM principal. Informa de colisión y ataque a la bola. Informa de dibujo y color al multiplexor de dibujo principal. Recibe señales de movimiento del teclado

Módulos relacionados: Chip, vga_sync, bola, PS2Interface.

Puertos de entrada:

- reset: Señal de reset con activo en alta.
- clk: Señal de reloj sincronizado con vsync
- start_b: Señal de botón de inicio con activo en alta
- left_b: Señal de botón de movimiento a izquierda con activo en alta
- right_b: Señal de botón de movimiento a derecha con activo en alta
- attack_b: Señal de botón de ataque con activo en alta
- vga_x (10 bits): Vector de posición del escaneo VGA (coordenada horizontal)
- vga_y (10 bits): Vector de posición del escaneo VGA (coordenada vertical)
- ball_x (10 bits): Vector de posición de la bola (coordenada horizontal)
- ball_y (9 bits): Vector de posición de la bola (coordenada vertical)

Puertos de salida:

- pinta_personaje: Señal de pintado de sprite de personaje
- rgb_personaje (9 bits): Vector de color de sprite de personaje
- collision: Señal de colisión de personaje con bola (ataque o no)
- ataque: Señal de acción de atacar en curso
- dead: Señal de personaje muerto

3.3 ball_player_collision

Módulo de colisión entre personaje y bola. Se encarga de comprobar la colisión utilizando las posiciones del personaje y de la bola, variando la zona de colisión si según el personaje esté atacando o no.

Puertos de entrada:

- attack: Señal de realización de ataque de personaje con activo en alta.
- player_x (10 bits): Vector de posición del personaje (coordenada horizontal).
- player_y (10 bits): Vector de posición del personaje (coordenada vertical).
- ball_x (10 bits): Vector de posición de la bola (coordenada horizontal).
- ball_y (9 bits): Vector de posición de la bola (coordenada vertical).

Puertos de salida:

- collision: Señal de colisión entre personaje y bola.

3.4 player_sprite_standing

Módulo de sprite de personaje quieto. Guarda la imagen que se muestra cuando el personaje no se mueve ni ataca.

Puertos de entrada:

- x (10 bits): Vector de direccionamiento de la ROM del sprite.

Puertos de salida:

- colour (3 bits): Vector de color en la ROM con dirección x, codificado en la paleta de colores del personaje.

3.5 player_sprite_moving_1

Módulo de primer sprite de personaje en movimiento. Guarda la primera imagen que se muestra en la animación del personaje cuando se mueve.

Puertos de entrada:

- x (10 bits): Vector de direccionamiento de la ROM del sprite.

Puertos de salida:

- colour (3 bits): Vector de color en la ROM con dirección x, codificado en la paleta de colores del personaje.

3.6 player_sprite_moving_2

Módulo de segundo sprite de personaje en movimiento. Guarda la segunda imagen que se muestra en la animación del personaje cuando se mueve.

Puertos de entrada:

- x (10 bits): Vector de direccionamiento de la ROM del sprite.

Puertos de salida:

- colour (3 bits): Vector de color en la ROM con dirección x, codificado en la paleta de colores del personaje.

3.7 player_sprite_attack_1

Módulo de primer sprite de personaje atacando. Guarda la imagen que se muestra en la primera y séptima parte de la animación del personaje cuando ataca.

Puertos de entrada:

- x (10 bits): Vector de direccionamiento de la ROM del sprite.

Puertos de salida:

- colour (3 bits): Vector de color en la ROM con dirección x, codificado en la paleta de colores del personaje.

3.8 player_sprite_attack_2

Módulo de segundo sprite de personaje atacando. Guarda la imagen que se muestra en la segunda y sexta parte de la animación del personaje cuando ataca.

Puertos de entrada:

- x (10 bits): Vector de direccionamiento de la ROM del sprite.

Puertos de salida:

- colour (3 bits): Vector de color en la ROM con dirección x, codificado en la paleta de colores del personaje.

3.9 player_sprite_attack_3

Módulo de tercer sprite de personaje atacando. Guarda la imagen que se muestra en la segunda y quinta parte de la animación del personaje cuando ataca.

Puertos de entrada:

- x (10 bits): Vector de direccionamiento de la ROM del sprite.

Puertos de salida:

- colour (3 bits): Vector de color en la ROM con dirección x, codificado en la paleta de colores del personaje.

3.10 player_sprite_attack_4

Módulo de cuarto sprite de personaje atacando. Guarda la imagen que se muestra en la cuarta parte de la animación del personaje cuando ataca.

Puertos de entrada:

- x (10 bits): Vector de direccionamiento de la ROM del sprite.

Puertos de salida:

- colour (3 bits): Vector de color en la ROM con dirección x, codificado en la paleta de colores del personaje.

3.11 PS2Interface

Módulo de interfaz de teclado PS2. Recibe las señales del teclado conectado por PS2, traduce los códigos del teclado de las teclas Esc, Z, X, Flecha Izquierda, Flecha Derecha, Flecha Arriba, Flecha Abajo; detectando si se mantiene presionado o se suelta la tecla. Informa de la transmisión, recepción y error en la comunicación con el teclado.

Módulos relacionados: Chip, personaje.

Puertos de entrada:

- clk: Señal de reloj de al menos 1KHz
- reset: Señal de reset con activo en alta
- ps2_clk: Señal de reloj del teclado PS2 (usada para sincronizar la comunicación)

- ps2_data: Señal de datos del teclado PS2

Puertos de salida:

- keyboard: Vector de presión de tecla con activo en alta
- transmiting: Señal de comunicación con el teclado en curso
- data_re: Señal de recepción valida de dato del teclado
- error: Señal de error en la comunicación con el teclado

3.12 PS2Controller

Módulo de controlador de teclado PS2. Se encarga de obtener las señales desde el cable PS2, traducir la información sincronizando la comunicación, empaquetar los datos completos y comprobar su correctitud mediante las medidas especificadas por el protocolo PS2.

Señales de entrada:

- clk: Señal de reloj igual a la de PS2Interface
- reset: Señal de reset con activo en alta
- ps2_clk: Señal de reloj del teclado PS2 (usada para sincronizar la comunicación)
- ps2_data: Señal de datos del teclado PS2

Señales de salida:

- scancode (8 bits): Vector de código de tecla recibido desde el teclado PS2.
- transmiting: Señal de control indicando que se está recibiendo un dato, con activo en alta
- data_received: Señal de control indicando que se acaba de recibir un dato completo, con activo en alta
- error: Señal de control indicando que el dato recibido es erróneo, con activo en alta

3.13 vga_sync

Módulo generador del reloj de sincronización de la pantalla VGA. Genera las señales horizontal y vertical de timing para una pantalla VGA de resolución 640x480. Proporciona un sistema de coordenadas horizontal y vertical para posicionar los sprites en la pantalla.

Módulos relacionados: Chip, personaje, bola, cards, background, overlay.

Puertos de entrada:

- clk: Señal de reloj de 25 MHz
- reset: Señal de reset con activo en alta

Puertos de salida

- hsync: Señal de sincronización horizontal con la pantalla VGA (~31KHz)
- vsync: Señal de sincronización vertical con la pantalla VGA (~60Hz)
- video_on: Señal de posición en limites visibles (pixel actual en el plano 640x480) de activo en alta
- pixel_x (10 bits): Vector de posición horizontal del pixel actual
- pixel_y (10 bits): Vector de posición vertical del pixel actual

3.14 cards

Módulo del panel de cartas. Se encarga de comprobar la colisión de la bola con las cartas, girando las cartas cuando es necesario. Se comunica con la FSM principal para cargar cartas, cuando se acaban las cartas, para saber el nivel actual y para indicar que se ha eliminado una carta. Informa del dibujo y del color de dibujo al multiplexor de dibujo principal.

Módulos relacionados: Chip, bola.

Puertos de entrada:

- clk: Señal de reloj de 25 MHz
- reset: Señal de reset de activo en alta
- load_cards: Señal de comienzo del algoritmo de carga de cartas de activo en alta
- level (3 bits): Vector de nivel actual
- ball_x (10 bits): Vector de posición de la bola (coordenada horizontal)
- ball_y (9 bits): Vector de posición de la bola (coordenada vertical)
- vga_x (10 bits): Vector de posición del escaneo VGA (coordenada horizontal)
- vga_y (10 bits): Vector de posición del escaneo VGA (coordenada vertical)

Puertos de salida:

- no_cards_left: Señal de vaciado de cartas
- loaded: Señal de carga de cartas finalizada
- draw_sprite: Señal de pintado de sprite de carta

- card_out: Señal de desaparición de una carta
- rgb (9 bits): Vector de color de sprite de carta

3.15 cards_sprite_1

Módulo de sprites de primer nivel de cartas. Guarda las imágenes que se muestra en las cartas durante el primer nivel.

Puertos de entrada:

- sprite_num (3 bits): Vector indicando el nivel actual
- x (10 bits): Vector de direccionamiento de las ROM de los sprite.

Puertos de salida:

- draw_sprite: Señal indicando que el color indicado por la dirección x no es transparente.
- rgb (9 bits): Vector de color en la ROM con dirección x.

3.16 cards_sprite_2

Módulo de sprites de segundo nivel de cartas. Guarda las imágenes que se muestra en las cartas durante el segundo nivel.

Puertos de entrada:

- sprite_num (3 bits): Vector indicando el nivel actual
- x (10 bits): Vector de direccionamiento de las ROM de los sprite.

Puertos de salida:

- draw_sprite: Señal indicando que el color indicado por la dirección x no es transparente.
- rgb (9 bits): Vector de color en la ROM con dirección x.

3.17 cards_sprite_3

Módulo de sprites de tercer nivel de cartas. Guarda las imágenes que se muestra en las cartas durante el tercer nivel.

Puertos de entrada:

- sprite_num (3 bits): Vector indicando el nivel actual
- x (10 bits): Vector de direccionamiento de las ROM de los sprite.

Puertos de salida:

- draw_sprite: Señal indicando que el color indicado por la dirección x no es transparente.

- rgb (9 bits): Vector de color en la ROM con dirección x.

3.18 RAM

Módulo de memoria RAM de cartas. Almacena las cartas, tanto existencia como estado, para poder ser modificadas mediante el algoritmo del módulo de cartas.

Puertos de entrada:

- clk: Señal de reloj igual a la de cards.
- addr (8 bits): Vector de direccionamiento de la memoria RAM.
- din (5 bits): Vector de entrada de estado de carta.
- we: Señal de escritura en memoria.

Puertos de salida:

- dout (5 bits): Vector de salida de estado de carta.

3.19 random_bit_gen

Módulo de generador de bits aleatorios. Usado en la generación de mapas de cartas.

Puertos de entrada:

- clk: Señal de reloj lo de al menos 25MHz

Puertos de salida:

- rng: Señal de bit aleatorio

3.20 bola

Módulo de bola. Controla el movimiento, la gravedad y el rebote de la bola, además de la posición de esta. Informa de la posición de la bola al personaje y a las cartas. Informa de dibujo y color al multiplexor de dibujo principal. Recibe avisos de ataque de personaje y de comienzo de nivel.

Módulos relacionados: Chip, personaje, vga_sync.

Puertos de entrada:

- clk : Señal de reloj sincronizado con vsync
- rst: Señal de reset con activo en alta.

- start_b: Señal de inicio de máquina de estado con activo en alta
- ataca: Señal de ataque de personaje en curso
- col_per: Señal de colisión con personaje
- vga_x (10 bits): Vector de posición del escaneo VGA (coordenada horizontal)
- vga_y (10 bits): Vector de posición del escaneo VGA (coordenada vertical)

Puertos de salida:

- pos_x (10 bits): Vector de posición de la bola (coordenada horizontal)
- pos_y (9 bits): Vector de posición de la bola (coordenada vertical)
- pinta_bola: Señal de pintado de sprite de bola
- rgb_bola (9 bits): Vector de color de sprite de bola

3.21 ball_sprite

Módulo de sprite de bola. Guarda la imagen de la bola.

Puertos de entrada:

- x (10 bits): Vector de direccionamiento de la ROM del sprite.

Puertos de salida:

- colour (2 bits): Vector de color de sprite de bola en la dirección x, codificado con 2 bits en la paleta de la bola.

3.22 Sound

Módulo de melodía de fondo. Se encarga de producir los canales de audio y mezclarlos en uno solo para poder reproducirlo en un altavoz simple. Emite la señal de sonido y una señal de amplificación de audio opcional.

Módulos relacionados: Chip

Puertos de entrada:

- clk: Señal de reloj de 50 MHz
- reset: Señal de reset con activo en alta

Puertos de salida:

- ampPWM: Señal de oscilador de audio

- ampPWM: Señal de amplificación de audio con activo en alta

3.23 PFM

Módulo de modulador de pulso por frecuencia. Se encarga de generar ondas según la media frecuencia introducida.

Parámetros genéricos:

- n (natural): Tamaño en bits del vector limit.

Puertos de entrada:

- clk: Señal de reloj de 50MHz.
- reset: Señal de reset con activo en alta.
- limit(n bits): Vector de límite numérico de media frecuencia de onda.

Puertos de salida:

- clk_out: Señal de reloj a la frecuencia determinada por el doble de limit.

3.24 bass_1_rom

Módulo de primer instrumento bass. Guarda la melodía del primer instrumento bass codificada con el mínimo numero de bits necesarios, traduciéndolos después a la frecuencia deseada.

Puertos de entrada:

- clk: Señal de reloj de 50MHz.
- en: Señal de habilitador de reloj, determinado por la velocidad de reproducción del instrumento.
- addr (4 bits): Vector de direccionamiento de la ROM de la partitura del instrumento.

Puertos de salida:

- d_out (18 bits): Vector de media frecuencia de la nota seleccionada por addr de la partitura.

3.25 bass_2_rom

Módulo de segundo instrumento bass. Guarda la melodía del segundo instrumento bass codificada con el mínimo número de bits necesarios, traduciéndolos después a la frecuencia deseada.

Puertos de entrada:

- clk: Señal de reloj de 50MHz.
- en: Señal de habilitador de reloj, determinado por la velocidad de reproducción del instrumento.
- addr (4 bits): Vector de direccionamiento de la ROM de la partitura del instrumento.

Puertos de salida:

- d_out (18 bits): Vector de media frecuencia de la nota seleccionada por addr de la partitura.

3.26 bass_3_rom

Módulo de tercer instrumento bass. Guarda la melodía del tercer instrumento bass codificada con el mínimo numero de bits necesarios, traduciéndolos después a la frecuencia deseada.

Puertos de entrada:

- clk: Señal de reloj de 50MHz.
- en: Señal de habilitador de reloj, determinado por la velocidad de reproducción del instrumento.
- addr (4 bits): Vector de direccionamiento de la ROM de la partitura del instrumento.

Puertos de salida:

- d_out (16 bits): Vector de media frecuencia de la nota seleccionada por addr de la partitura.

3.27 bass_4_rom

Módulo de cuarto instrumento bass. Guarda la melodía del cuarto instrumento bass codificada con el mínimo número de bits necesarios, traduciéndolos después a la frecuencia deseada.

Puertos de entrada:

- clk: Señal de reloj de 50MHz.
- en: Señal de habilitador de reloj, determinado por la velocidad de reproducción del instrumento.
- addr (4 bits): Vector de direccionamiento de la ROM de la partitura del instrumento.

Puertos de salida:

- d_out (17 bits): Vector de media frecuencia de la nota seleccionada por addr de la partitura.

3.28 vibra_1_rom

Módulo de primer instrumento vibraphone. Guarda la melodía del primer instrumento vibraphone codificada con el mínimo número de bits necesarios, traduciéndolos después a la frecuencia deseada.

Puertos de entrada:

- clk: Señal de reloj de 50MHz.
- en: Señal de habilitador de reloj, determinado por la velocidad de reproducción del instrumento.
- addr (4 bits): Vector de direccionamiento de la ROM de la partitura del instrumento.

Puertos de salida:

- d_out (16 bits): Vector de media frecuencia de la nota seleccionada por addr de la partitura.

3.29 vibra_2_rom

Módulo de segundo instrumento vibraphone. Guarda la melodía del segundo instrumento vibraphone codificada con el mínimo número de bits necesarios, traduciéndolos después a la frecuencia deseada.

Puertos de entrada:

- clk: Señal de reloj de 50MHz.
- en: Señal de habilitador de reloj, determinado por la velocidad de reproducción del instrumento.
- addr (6 bits): Vector de direccionamiento de la ROM de la partitura del instrumento.

Puertos de salida:

- d_out (15 bits): Vector de media frecuencia de la nota seleccionada por addr de la partitura.

3.30 lead_1_rom

Módulo de primer instrumento lead. Guarda la melodía del primer instrumento lead codificada con el mínimo número de bits necesarios, traduciéndolos después a la frecuencia deseada.

Puertos de entrada:

- clk: Señal de reloj de 50MHz.
- en: Señal de habilitador de reloj, determinado por la velocidad de reproducción del instrumento.
- addr (5 bits): Vector de direccionamiento de la ROM de la partitura del instrumento.

Puertos de salida:

- d_out (16 bits): Vector de media frecuencia de la nota seleccionada por addr de la partitura.

3.31 lead_2_rom

Módulo de segundo instrumento lead. Guarda la melodía del segundo instrumento lead codificada con el mínimo número de bits necesarios, traduciéndolos después a la frecuencia deseada.

Puertos de entrada:

- clk: Señal de reloj de 50MHz.
- en: Señal de habilitador de reloj, determinado por la velocidad de reproducción del instrumento.
- addr (4 bits): Vector de direccionamiento de la ROM de la partitura del instrumento.

Puertos de salida:

- d_out (16 bits): Vector de media frecuencia de la nota seleccionada por addr de la partitura.

3.32 overlay

Módulo de gráficos superpuestos. Se encarga de generar el panel superior de puntuación, de traducir la puntuación a decimal y mostrarlo en pantalla, de mostrar las vidas restantes y el nivel actual.

Módulos relacionados: Chip, vga_sync

Puertos de entrada:

- clk : Señal de reloj de 25 MHz
- reset: Señal de reset con activo en alta
- interleave: Señal de pantalla de cambio de nivel
- level (3 bits): Vector de nivel actual
- points (24 bits): Vector de puntuación en base 2
- lives (3 bits): Vector de cantidad de vidas
- vga_x (10 bits): Vector de posición del escaneo VGA (coordenada horizontal)
- vga_y (10 bits): Vector de posición del escaneo VGA (coordenada vertical)

Puertos de salida:

- draw_sprite: Señal de pintado de overlay
- rgb (9 bits): Vector de color de overlay

3.33 lives_sprite

Módulo de sprite de vida. Guarda la imagen que se muestra en el panel de overlay para representar las vidas actuales.

Puertos de entrada:

- x (8 bits): Vector de direccionamiento de la ROM del sprite.

Puertos de salida:

- draw_sprite: Señal indicando que el color indicado por la dirección x no es transparente.
- rgb (9 bits): Vector de color en la ROM con dirección x.

3.34 digits

Módulo de traducción de BCD a sprite del dígito decimal.

Puertos de entrada:

- x (8 bits): Vector de direccionamiento de la ROM del sprite.
- number (4 bits): Vector de dígito BCD.

Puertos de salida:

- draw_digit: Señal de dibujo del pixel actual.

3.35 overlay_sprite

Módulo de sprite complejo de overlay. Guarda el mapa de bits de la barra de overlay.

Puertos de entrada:

- x (16 bits): Vector de direccionamiento de la ROM del sprite.

Puertos de salida:

- draw_overlay: Señal de dibujo del pixel actual.

3.36 BCD

Módulo de transformación de números en binario a números en BCD.

Puertos de entrada:

- clk: Señal de reloj de al menos 25 MHz.
- reset: Señal de reset con activo en alta.
- start: Señal de inicio de algoritmo con activo en alta.
- binary (24 bits): Vector de número binario a convertir.

Puertos de salida:

- finish: Señal de finalización de algoritmo con activo en alta.
- output (28 bits): Vector de salida de número decimal de 7 cifras codificado en BCD.

3.37 startscreen

Módulo de pantalla de inicio.

Puertos de entrada:

- vga_x (10 bits): Vector de posición del escaneo VGA (coordenada horizontal)
- vga_y (10 bits): Vector de posición del escaneo VGA (coordenada vertical)

Puertos de salida:

- draw_sprite: Señal de pintado de pantalla de inicio.

3.38 endscreen

Módulo de pantallas finales de victoria y derrota.

Puertos de entrada:

- win: Señal de selección de pantalla de victoria con activo en alta.
- vga_x (10 bits): Vector de posición del escaneo VGA (coordenada horizontal)
- vga_y (10 bits): Vector de posición del escaneo VGA (coordenada vertical)

Puertos de salida:

- draw_sprite: Señal de pintado de pantallas finales.

3.39 background_1

Módulo de imagen de fondo durante el juego.

Puertos de entrada:

- clk: Señal de reloj a 25MHz.

- reset: Señal de reset con activo en alta.
- vga_x (10 bits): Vector de posición del escaneo VGA (coordenada horizontal)
- vga_y (10 bits): Vector de posición del escaneo VGA (coordenada vertical)

Puertos de salida:

- rgb(9 bits): Vector de color de imagen de fondo.

3.40 divisor

Módulo de divisor genérico. Divide el reloj de entrada por un factor mayor que dos.

Parámetros genéricos:

- input (natural): Frecuencia de entrada en Hz
- output (natural: Frecuencia de salida deseada en Hz

Puertos de entrada:

- reset: Señal de reset con activo en alta.
- clk_entrada: Señal de reloj de entrada a la frecuencia indicada en input.
- clk_salida: Señal de reloj de salida a la frecuencia indicada en output.

4 Descripción detallada de cada componente

4.1 Chip

4.1.1 Hardware (Elementos)

- Señales internas:
  - 1 máquina de estados con 8 estados
    - INICIAL
    - CARGAR_CARTAS
    - CARGANDO
    - JUGANDO
    - INTERMEDIO_CARGANDO
    - INTERMEDIO_CARGAR_CARTAS
    - FIN_JUEGO
    - JUEGO_PERDIDO
  - 1 vector de 6 bits de teclas pulsadas: keyboard.
  - 6 señales de pintado en pantalla:
    - Pintado de personaje: pinta_personaje.
    - Pintado de bola: pinta_bola.
    - Pintado de carta: pinta_carta.
    - Pintado de overlay: draw_overlay.
    - Pintado de pantalla inicial: draw_start.
    - Pintado de pantallas finales: draw_end.
  - 1 señal de carga de cartas: load_cards.
  - 1 señal de cartas cargadas: loaded.
  - 1 señal de ataque de personaje: ataque.
  - 1 señal de escaneo VGA en zona visible: v_on.
  - 1 señal de colision personaje-bola: col_bola.
  - 1 señal de sincronización vertical de VGA: vsynb.
  - 1 señal de vaciado de cartas: no_cards.
  - 1 señal de renicio por cambio de nivel con activo en alta: reset_cambio_nivel.
  - 1 señal de inversión de reset a activo en alta: reset.
  - 1 señal de periodo intermedio entre niveles: interleave.
  - 1 señal de reloj interno de 25 MHz: clock_rel.
  - 1 contador ascendente de 3 bits de nivel: level_c.
  - 1 vector de 10 bits de posición de escaneo horizontal VGA: vga_x.
  - 1 vector de 10 bits de posición de escaneo vertical VGA: vga_y.
  - 1 vector de 10 bits de posición horizontal de bola: bx.
  - 1 vector de 9 bits de posición vertical de bola: by.
  - 5 vectores de 9 bits de color:
    - Color de personaje: rgb_personaje.
    - Color de bola: rgb_bola.
    - Color de cartas: rgb_cartas.
    - Color de overlay: rgb_overlay.
    - Color de fondos: rgb_bg.
  - 1 señal de muerte de personaje: dead.
  - 1 señal de eliminación de carta: carta_eliminada.
  - 1 señal de selección de pantalla final: end_select.

    - 1 contador ascendente de 24 bits de puntuación: puntos.
    - 1 contador descendente de 3 bits de vidas: vidas.
    - 1 multiplexor de 10 a 1 de 9 bits de colores.
- Componentes internos:
    - 1 módulo de personaje.
    - 1 módulo de interfaz PS2.
    - 1 módulo de divisor.
    - 1 módulo de sincronización VGA.
    - 1 módulo de cartas.
    - 1 módulo de bola
    - 1 módulo de sonido.
    - 1 módulo de overlay.
- Descripción de estados:
    - INICIAL
      Estado inicial. No se hace nada. Se pasa directamente al estado CARGAR_CARTAS.
    - CARGAR_CARTAS
      Estado de comienzo de carga de cartas. Se da la orden de inicio del algoritmo de cargar cartas. Se pasa directamente al estado CARGANDO.
    - CARGANDO
      Estado de espera de terminación del algoritmo de cargado de cartas. Se mantiene en este estado hasta que las cartas hayan sido cargadas (loaded activo) y el jugador pulse el botón Esc; en ese caso se pasa al estado JUGANDO. Durante este estado se muestra la pantalla inicial.
    - JUGANDO
      Estado de juego, se mantiene a menos que pasen los siguientes sucesos:
        - El jugador pierde todas las vidas: se pasa al estado JUEGO_PERDIDO.
        - No quedan cartas y no se esta en el tercer nivel: se pasa al estado INTERMEDIO_CARGAR_CARTAS.
        - No quedan cartas y el nivel actual es el tercero: se pasa a FIN_JUEGO.
    - INTERMEDIO_CARGAR_CARTAS
      Estado equivalente a CARGAR_CARTAS. Se da la orden de inicio de cargar cartas. Se pasa al estado INTERMEDIO_CARGANDO.
    - INTERMEDIO_CARGANDO
      Estado equivalente a CARGANDO. Se espera a la finalización de carga de cartas. Se pasa al estado JUGANDO cuando se han cargado las cartas y se pulsa el botón Esc. En este estado se muestra la pantalla intermedia, ocultando la bola y las cartas al mismo tiempo.
    - JUEGO_PERDIDO
      Estado de juego perdido, se mantiene en este estado siempre. Se muestra la pantalla final de derrota.
    - FIN_JUEGO
      Estado de juego ganado, se mantiene en este estado siempre. Se muestra la pantalla final de victoria.

- Descripción del funcionamiento:

  Al haber muchos generadores de colores, estos se priorizan mediante el uso de las señales de dibujo. El orden de prioridad descendente es: 0 cuando se escanea fuera de la pantalla visible, pantallas finales, pantalla intermedia, overlay, personaje, bola, cartas y fondo de pantalla.

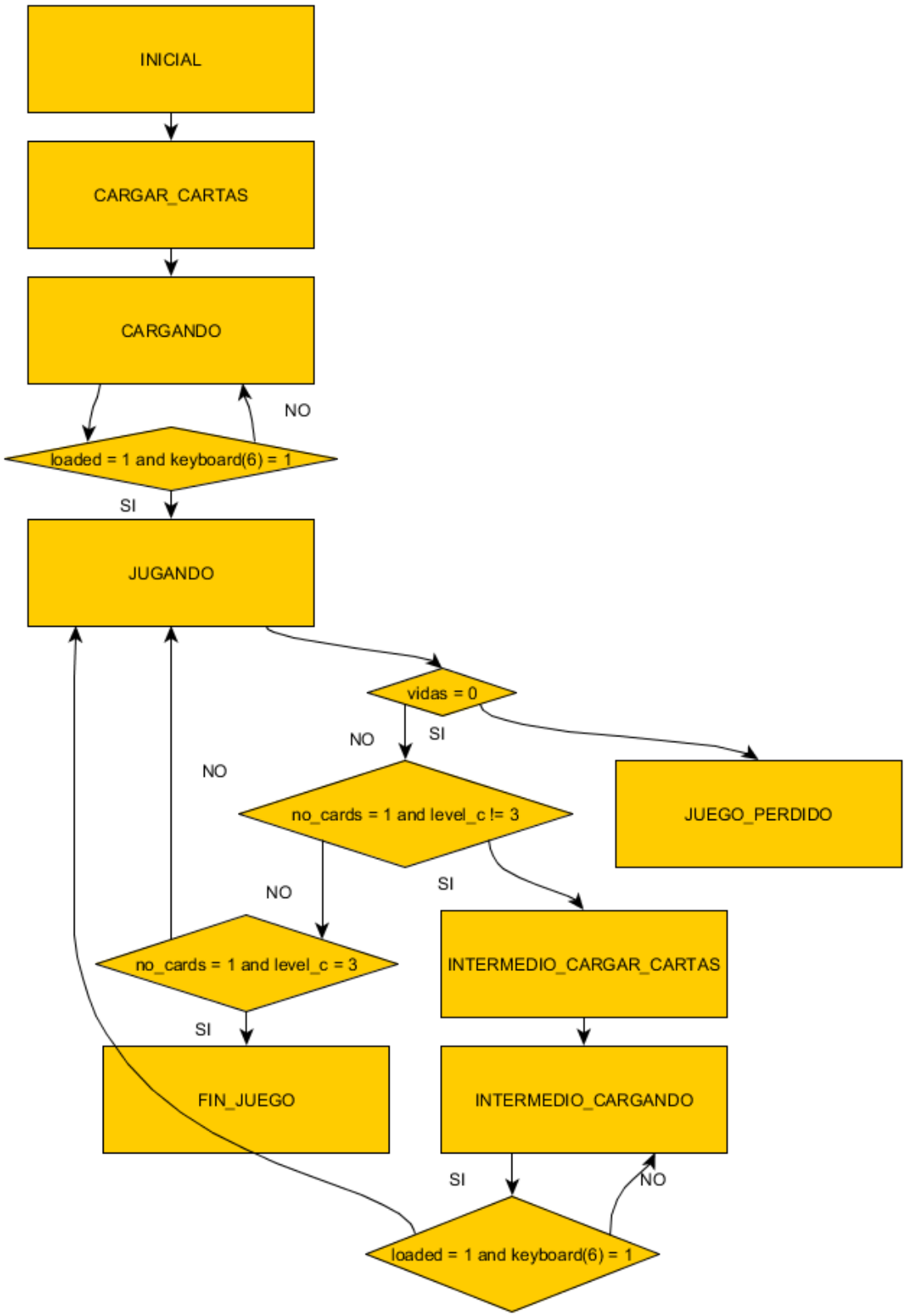
  Los resets de bola, personaje y sonido se activan cuando hay un reset general o cuando se cambia de nivel.

4.1.2 Hardware (FPGA)

Summary:

```
inferred   1 Finite State Machine(s).
inferred   2 Counter(s).
inferred  24 D-type flip-flop(s).
inferred   1 Adder/Subtractor(s).
```

4.1.3 Diagrama ASM



4.1.4 VHDL Original

El código de este módulo se encuentra en el archivo Chip.vhd

4.2 Personaje

4.2.1 Hardware (Elementos)

- Señales internas:
    - 1 máquina de estados con 6 estados.
        - Inicio
        - Quieto
        - Mover_izq
        - Mover_der
        - Atacar
        - Muerto
    - 1 contador ascendente descendente con carga paralela de 10 bits y avance de cuatro en cuatro para guardar la posición en la coordenada x del personaje: player_x.
    - 1 contador ascendente con reset a cero de 3 bits y avance de uno en uno para generar la animación de movimiento a derecha e izquierda: sprite_c
    - 1 contador ascendente con reset a cero de 6 bits y avance de uno en uno para generar el periodo de invulnerabilidad tras muerte del personaje: dead_c
    - 1 contador ascendente con reset a cero de 6 bits y avance de uno en uno para generar el periodo de realización de ataque: attack_c
    - 1 señal de invulnerabilidad de personaje: invulnerable
    - 1 señal de colisión con bola: col_bola
    - 1 señal de alternación de sprite de animación de movimiento: mov_anim
    - 1 señal de alternación de color en periodo de invulnerabilidad: muerto_anim
    - 1 señal de realización de ataque: atq
    - 8 vectores de 9 bits para representar colores RGB
        - 1 vector de multiplexación de 7 a 1 de 9 bits de los colores de los sprites: rgb_dec
        - Vector de sprite de primer frame de animación de movimiento: colour_moving_1
        - Vector de sprite de segundo frame de animación de movimiento: colour_moving_2
        - Vector de sprite de primer y septimo frame de animación de ataque: colour_attack_1
        - Vector de sprite de segundo y sexto frame de animación de ataque: colour_attack_2
        - Vector de sprite de tercer y quinto frame de animación de ataque: colour_attack_3
        - Vector de sprite de cuarto frame de animación de ataque: colour_attack_4
    - 3 vectores de 10 bits de posición de lectura de ROM de sprites:
        - Direccionamiento de sprite de personaje quieto: x_sprite
        - Direccionamiento de sprite de primer frame de animación de movimiento: x_mov_1
        - Direccionamiento de sprite de segundo frame de animación de movimiento: x_mov_2
    - 4 vectores de 11 bits de posición de lectura de ROM de sprites:

        - Direccionamiento de sprite de primer y séptimo frame de animación de ataque: x_attack_1
        - Direccionamiento de sprite de segundo y sexto frame de animación de ataque: x_attack_2
        - Direccionamiento de sprite de tercer y quinto frame de animación de ataque: x_attack_3
        - Direccionamiento de sprite de cuarto frame de animación de ataque: x_attack_4
- Componentes internos:
    - 7 sprites diferentes:
        - Sprite de personaje quieto: player_sprite_standing
        - Sprite de primer frame de animación de movimiento: player_sprite_moving_1
        - Sprite de segundo frame de animación de movimiento: player_sprite_moving_2
        - Sprite de primer y septimo frame de animación de ataque: player_sprite_attack_1
        - Sprite de segundo y sexto frame de animación de ataque: player_sprite_attack_2
        - Sprite de tercer y quinto frame de animación de ataque: player_sprite_attack_3
        - Sprite de cuarto frame de animación de ataque: player_sprite_attack_4
    - 1 submódulo de colision con bola: ball_player_collision
- Descripción de estados:
    - Inicio
      Se usa como estado de pausa al iniciar el juego y tras cada cambio de nivel. A este estado solo se puede acceder tras reiniciar el módulo. Desde este estado se pasa a Quieto al pulsar el botón de comienzo (start_b), que corresponde al botón de ataque Z del teclado.
    - Quieto
      Aquí el módulo de personaje ya ha “arrancado”, por lo que este estado representa que el personaje no está haciendo nada, y se muestra el grafico de estar de pie. Desde este estado se puede pasar a Mover_izq, Mover_der, Atacar o Muerto, según se pulse la tecla izquierda, derecha, el botón de ataque Z o la bola colisione con el personaje. Si se pulsan los botones izquierda y derecha a la vez o si no se da ninguno de los casos anteriores se mantiene en el estado Quieto.
    - Mover_izq y Mover_der
      Son totalmente equivalentes al estado Quieto en cuanto a las transiciones de estado. Sin embargo, en estos estados se produce un movimiento de cuatro pixeles en la dirección respectiva y se muestra el grafico correspondiente a izquierda y derecha en el instante actual.
    - Atacar
      Comienza al haber pulsado el botón de ataque Z en los estados Quieto, Mover_izq, Mover_der, o al estar realizando el ataque en el estado Atacar. Al llegar a este estado desde Quieto, Mover_izq o Mover_der, por construcción, el contador de frames de ataque ha de estar a 0; y por tanto comienza una cuenta hacia arriba desde 0 a 55 frames, durante los cuales el personaje está realizando el ataque. Mientras el contador no alcance el último frame, el estado siempre será Atacar y se informará al

exterior de la realización del ataque mediante la señal de salida ataque. Durante la realización del ataque existe una aplicación matemática que relaciona el número del frame de ataque y el sprite de la animación de ataque. Una vez se llegue al quincuagésimo sexto frame, las transiciones son iguales a las de Quieto, Mover_izq y Mover_der, salvo que no se morir ni realizar otro ataque seguido, ya que queremos balancear las capacidades de defensa del jugador.

- Muerto
  El estado Muerto se alcanza al colisionar la bola con el personaje en los estados Quieto, Mover_izq y Mover_der. Al igual que con el estado Atacar, al llegar a este estado se inicia un contador autónomo desde 0 hasta 63. A diferencia del estado Atacar, una vez finaliza el estado Muerto, el único estado posible al que avanzar es al de Quieto. La función del contador de muerto es contar 64 frames durante los cuales el personaje es invulnerable. Durante el periodo de invulnerabilidad, los colores de los graficos del personaje se sobreescriben con una alternación entre rojo y blanco, que se consigue diferenciando el estado del bit 4 del contador de muerto.

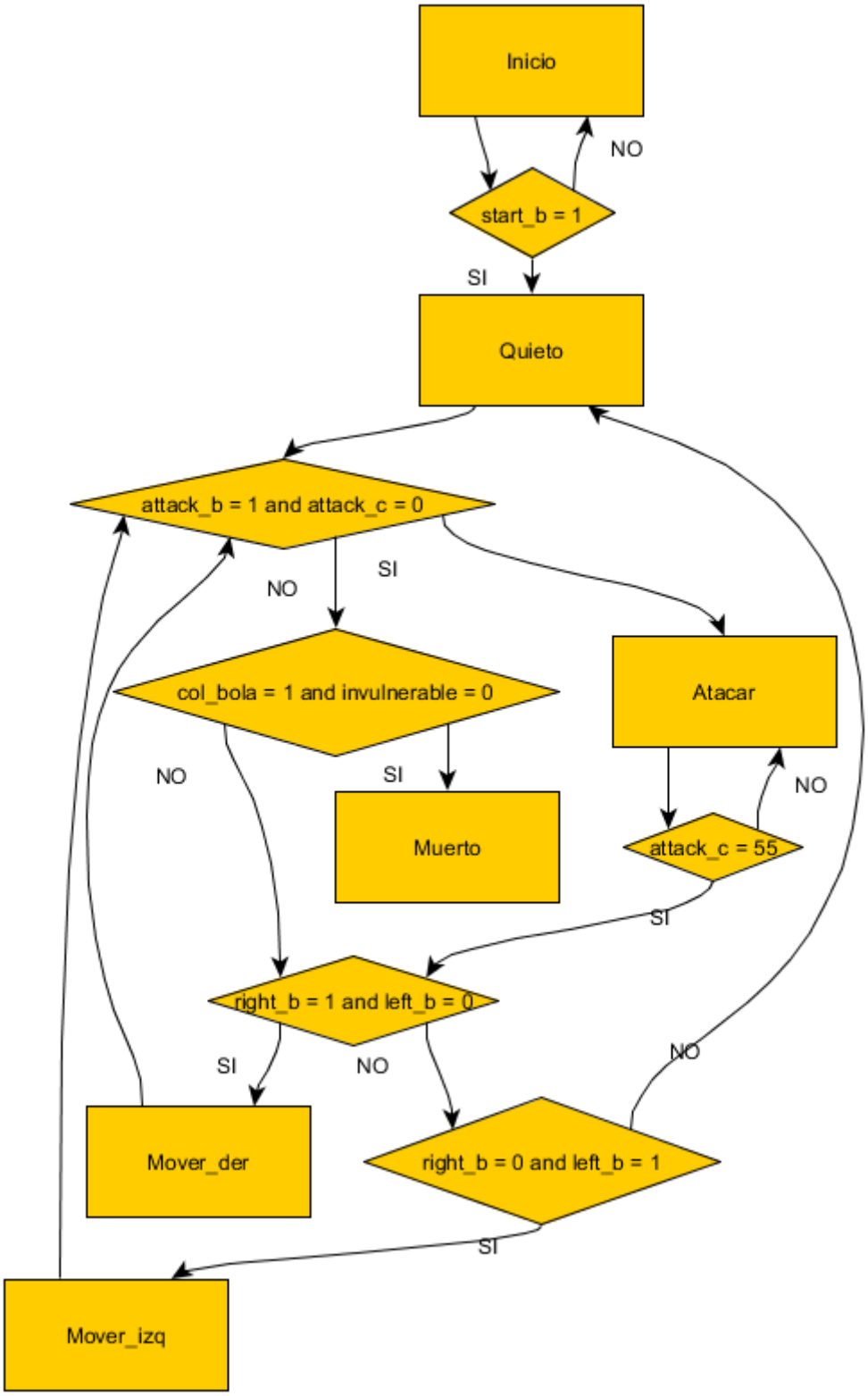
4.2.2 Hardware (FPGA)

Summary:

```
inferred   1 Finite State Machine(s).
inferred   2 Counter(s).
inferred   3 D-type flip-flop(s).
inferred  27 Adder/Subtractor(s).
inferred   6 Multiplier(s).
inferred  27 Comparator(s).
```

Cabe destacar que los multiplicadores se usan para direccionar el acceso a las ROM de sprites. Dado que la FPGA destino tiene integrados varios multiplicadores esto no es un problema.

4.2.3 Diagrama ASM



4.2.4 VHDL Original

El código de este módulo se encuentra en el archivo personaje.vhd

4.3 ball_player_collision

4.3.1 Hardware (Elementos)

- Señales internas:
  - Ninguna
- Descripción del funcionamiento:
  - La colisión entre la bola y el personaje se realiza de forma estática mediante la comparación de la posición de la bola y la posición del personaje. De forma más específica, en cuanto la esquina superior izquierda de la bola entra en contacto con un rectángulo que rodea en cierta manera al personaje, se considera que colisionan.
  - Este rectángulo cambia según el personaje este atacando o no, ya que los sprites de movimiento y quieto tiene un volumen menor que los de ataque.
  - Si el personaje no está atacando, el rectángulo es el de borde superior a 30 pixeles, borde izquierdo y derecho a 20 pixeles de la esquina inferior izquierda del personaje medidos en las proyecciones de los ejes.
  - Si el personaje está atacando, el rectángulo es el de borde superior a 67 pixeles, borde izquierdo a 25 y derecho a 42 pixeles de la esquina inferior izquierda del personaje medidos en las proyecciones de los ejes.
  - El módulo compara los valores de la posición de la bola y personaje. Si se produce colisión, esta información se propaga al exterior.

4.3.2 Hardware (FPGA)

Summary:

```
inferred   4 Adder/Subtractor(s).
inferred   6 Comparator(s).
```

4.3.3 Diagrama ASM

El módulo es combinacional.

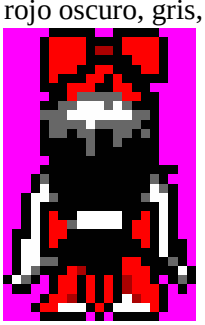
4.3.4 VHDL Original

El código de este módulo se encuentra en el archivo ball_player_collision.vhd

4.4 player_sprite_standing

4.4.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 1024 x 3 bits
- Descripción del funcionamiento:
  - El módulo guarda en una ROM de 1024 palabras de 3 bits la siguiente imagen codificando mediante una codificación mínima de negro, rojo, rojo oscuro, gris, gris oscuro, blanco, beige y transparencia.



4.4.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).

4.4.3 Diagrama ASM

El módulo es combinacional

4.4.4 VHDL Original

El código de este módulo se encuentra en el archivo player_sprite_still.vhd

4.5 player_sprite_moving_1

4.5.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 1024 x 3 bits
- Descripción del funcionamiento:
  - El módulo guarda en una ROM de 1024 palabras de 3 bits la siguiente imagen codificando mediante una codificación mínima de negro, rojo, rojo oscuro, gris, gris oscuro, blanco, beige y transparencia.

4.5.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).

4.5.3 Diagrama ASM

El módulo es combinacional

4.5.4 VHDL Original

El código de este módulo se encuentra en el archivo moving_1.vhd

4.6 player_sprite_moving_2

4.6.1 Hardware(Elementos)

- Señales internas:
  - 1 ROM de 1024 x 3 bits
- Descripción del funcionamiento:
  - El módulo guarda en una ROM de 1024 palabras de 3 bits la siguiente imagen codificando mediante una codificación mínima de negro, rojo, rojo oscuro, gris, gris oscuro, blanco, beige y transparencia.

4.6.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).

4.6.3 Diagrama ASM

El módulo es combinacional

4.6.4 VHDL Original

El código de este módulo se encuentra en el archivo player_sprite_moving_2.vhd

4.7 player_sprite_attack_1

4.7.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 1089 x 3 bits
- Descripción del funcionamiento:
  - El módulo guarda en una ROM de 1089 palabras de 3 bits la siguiente imagen codificando mediante una codificación mínima de negro, rojo, rojo oscuro, gris, gris oscuro, blanco, beige y transparencia.

4.7.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).

4.7.3 Diagrama ASM

El módulo es combinacional

4.7.4 VHDL Original

El código de este módulo se encuentra en el archivo player_sprite_attack_1.vhd

4.8 player_sprite_attack_2

4.8.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 1443 x 3 bits
- Descripción del funcionamiento:
  - El módulo guarda en una ROM de 1443 palabras de 3 bits la siguiente imagen codificando mediante una codificación mínima de negro, rojo, rojo oscuro, gris, gris oscuro, blanco, beige y transparencia.

4.8.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).

4.8.3 Diagrama ASM

El módulo es combinacional

4.8.4 VHDL Original

El código de este módulo se encuentra en el archivo player_sprite_attack_2.vhd

4.9 player_sprite_attack_3

4.9.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 1764 x 3 bits
- Descripción del funcionamiento:
  - El módulo guarda en una ROM de 1764 palabras de 3 bits la siguiente imagen codificando mediante una codificación mínima de negro, rojo, rojo oscuro, gris, gris oscuro, blanco, beige y transparencia.

4.9.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).

4.9.3 Diagrama ASM

El módulo es combinacional

4.9.4 VHDL Original

El código de este módulo se encuentra en el archivo player_sprite_attack_3.vhd

4.10 player_sprite_attack_4

4.10.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 1155 x 3 bits
- Descripción del funcionamiento:
  - El módulo guarda en una ROM de 1155 palabras de 3 bits la siguiente imagen codificando mediante una codificación mínima de negro, rojo, rojo oscuro, gris, gris oscuro, blanco, beige y transparencia.

4.10.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).

4.10.3 Diagrama ASM

El módulo es combinacional

4.10.4 VHDL Original

El código de este módulo se encuentra en el archivo player_sprite_attack_4.vhd

4.11 PS2Interface

4.11.1 Hardware (Elementos)

- Señales internas:
  - 1 máquina de estados con 5 estados.
    - IDLE
    - DATA
    - KEY
    - BREAK
    - BREAKKEY
  - 1 vector de 8 bits de código de tecla: scancode.
  - 1 señal de recepción de tecla: data_received.
  - 7 biestables para guardar el estado de cada tecla:
    - Biestable para la tecla Esc: r_esc
    - Biestable para la tecla Z: r_z
    - Biestable para la tecla X: r_x
    - Biestable para la tecla Arriba: r_up
    - Biestable para la tecla Abajo: r_down
    - Biestable para la tecla Izquierda: r_left
    - Biestable para la tecla Derecha: r_right
  - 1 biestable para guardar información de la llegada de un código de ruptura: r_break.
- Componentes internos:
  - 1 submódulo de controlador de teclado PS2: PS2Controller.
- Descripción de estados:
  - IDLE

    Se encarga de representar los periodos cuando no se ha recibido completamente ninguna tecla. En caso de que el submódulo PS2Controller reciba un código de tecla completo (data_received), este estado cambia a DATA.

  - DATA

    Durante este estado se decodifica el código de tecla recibido, identificando la operación consiste en la pulsación de una tecla(KEY), la llegada del código de ruptura de tecla(BREAK) o la llegada de una tecla tras haber obtenido una ruptura no consumida(BREAKKEY).

  - KEY

    En este estado se sabe que ha llegado la pulsación de una tecla, por lo que se identifica que tecla es y se activa el biestable que representa a

dicha tecla. Tras este estado se vuelve a IDLE en espera de otra operación.

- BREAK

  Se ha recibido un código de ruptura de tecla, esto es, se ha dejado de pulsar una tecla, pero todavía no se sabe cuál se ha dejado de pulsar, por lo que se guarda este hecho(r_break) y se vuelve al estado IDLE. Se supone siempre que el funcionamiento del dispositivo PS2 es correcto y por tanto no genera dos códigos de ruptura seguidos sin indicar la tecla afectada.

- BREAKKEY

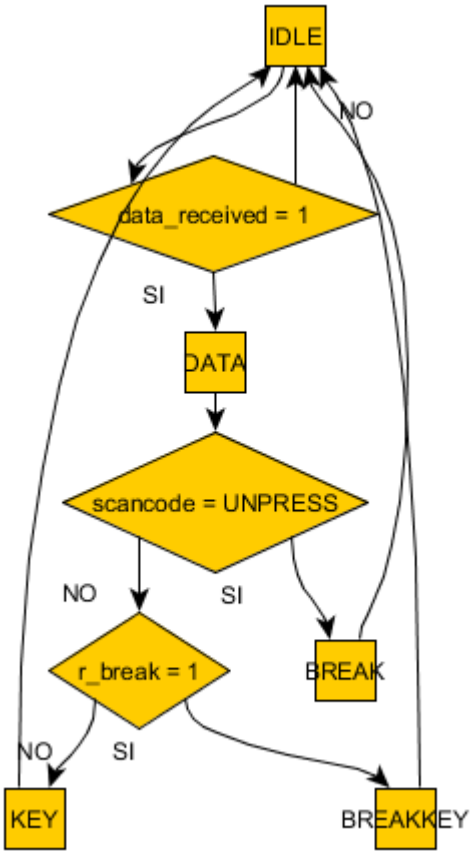
  Se ha recibido el código de una tecla, pero también se tenía guardada la aparición anterior de un código de ruptura no consumido, por lo que se decodifica la tecla en cuestión y se desactiva el biestable que la representa. A su vez se desactiva el biestable de memoria de ruptura, ya que la ruptura acaba de ser consumida.

4.11.2 Hardware (FPGA)

Summary:

inferred 1 Finite State Machine(s).
inferred 8 D-type flip-flop(s).

4.11.3 Diagrama ASM



4.11.4 VHDL Original

El código de este módulo se encuentra en el archivo PS2Interface.vhd

4.12 PS2Controller

4.12.1 Hardware (Elementos)

- Señales internas:
  - 1 máquina de estados con 5 estados.
    - IDLE
    - RECEIVING_DATA
    - WAIT_FOR_END_BIT
    - DATA_GOT
    - ERR
  - 1 registro con desplazamiento a derecha de 8 bits para preservar la recepción de bits del teclado PS2: sr_data.
  - 1 registro con desplazamiento a izquierda de 4 bits para detectar los flancos del reloj del teclado PS2: sr_clk.
  - 1 registro de 8 bits para guardar el código de tecla recibido: r_scancode.
  - 1 biestable para guardar la paridad del dato recibido: r_parity.
  - 1 contador con reset a cero de 4 bits para contar los bits recibidos durante la transmisión de datos: bit_counter.
  - 1 biestable para guardar la ocurrencia de error en el dato recibido: r_error.
- Descripción de estados:
  - IDLE
    En este estado se espera al siguiente flanco de bajada del teclado para comenzar la transmisión. En caso de que se haya producido el flanco se pasa al estado RECEIVING_DATA; si esto no sucede se sigue esperando en IDLE.
    Al recibir el flanco de bajada también aumenta el contador de bits, ya que el bit recibido es el de comienzo.
  - RECEIVING_DATA
    Los cambios en este estado se producen cuando se produce un flanco de bajada en el reloj del teclado. Mientras no se produzca el flanco se mantiene en RECEIVING_DATA. Si el flanco se produce entonces hay dos posibilidades:
    - Se han recibido durante la transmisión entre uno y nueve bits: en este caso se van a recibir los bits 1 a 9 comenzando el conteo desde cero, que corresponden con ocho bits de datos y un bit de paridad. En este caso se aumenta el contador de bits y se actualiza la paridad. Durante la recepción de la paridad se calcula el valor de error.
    - Se han recibido 10 bits: con este nuevo bit se han recibido todos los bits, por lo que pasamos al estado de espera de flanco de subida WAIT_FOR_END_BIT.
  - WAIT_FOR_END_BIT
    Se espera en este estado al flanco de subida del reloj del teclado, marcando el fin de la transacción. En caso de que hubiese sucedido un error se va al estado ERR, si no ocurre esto se va al estado DATA_GOT.
  - DATA_GOT
    El estado indica la recepción correcta del dato, propagando este hecho al exterior mediante el puerto de salida data_received. Durante este

estado el vector de código de teclado r_scancode se mantiene siempre estable. Tras este estado se pasa al estado IDLE.

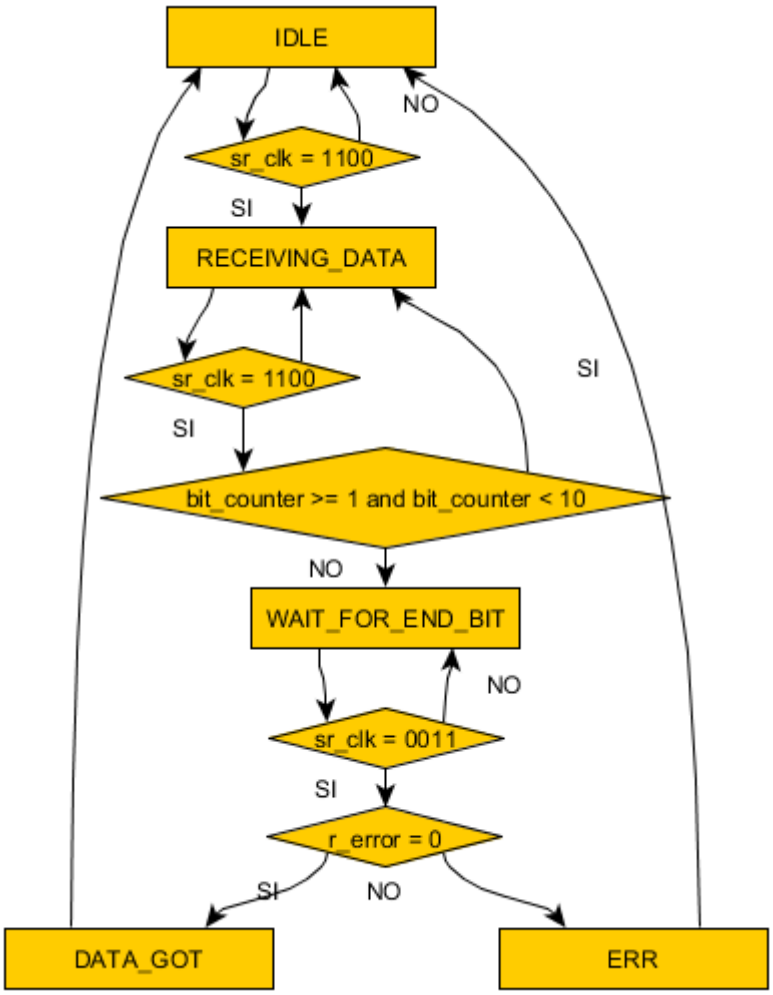
- ERR
  El estado indica el error en la recepción de la tecla, propagando este hecho al exterior mediante el puerto de salida error. Tras este estado se pasa al estado IDLE.

4.12.2 Hardware (FPGA)

Summary:

inferred 1 Finite State Machine(s).
inferred 1 Counter(s).
inferred 22 D-type flip-flop(s).
inferred 3 Comparator(s).

4.12.3 Diagrama ASM



4.12.4 VHDL Original

El código de este módulo se encuentra en el archivo PS2Controller.vhd

4.13 vga_sync

4.13.1 Hardware (Elementos)

- Señales internas:

    - 2 contadores ascendentes de 10 bits para contar tiempos para el reloj de sincronización horizontal y vertical de la VGA: h_count_reg, v_count_reg.
    - 2 señales de reloj de sincronización horizontal y vertical de la VGA: h_sync_reg, v_sync_reg.
- Descripción de funcionamiento:
  El módulo basicamente genera dos relojes. El reloj horizontal se encarga de indicar cuando se ha consumido una linea horizontal y se ha de avanzar a la siguiente. El reloj vertical se encarga de indicar cuando se han acabado todas las lineas horizontales, es decir, se ha completado un fotograma entero. Aunque esta explicacion es correcta, el funcionamiento en vivo no es intuitivamente igual, el hecho de que se acabe la linea horizontal no implica que en el siguiente ciclo se valla a dibujar el siguiente pixel; existe una franja de reajuste despues y antes de los flancos de sincronización horizontal y vertical. En total la pantalla que usamos tiene 640x480 pixeles visibles, pero para conseguir esto se necesitan en total 800x525 pixels. Por lo tanto, la linea horizontal esta compuesta por 800 pixels y hay 525 lineas horizontales. El reloj horizontal genera los flancos en los pixeles 657 y 751 (bajada y subida respectivamente), y el reloj vertical genera los flancos en los pixeles verticales 490 y 491 (bajada y subida respectivamente). El contador horizontal cuenta los pixeles en una linea horizontal, reiniciando en cero cuando completa los 800 pixeles. El contador vertical cuenta las lineas horizontales, aumentando el valor cada vez que se completa una linea horizontal, y reiniciando cuando se completan las 525 lineas verticales.

4.13.2 Hardware (FPGA)

Summary:

```
inferred   2 Counter(s).
inferred   2 D-type flip-flop(s).
inferred   6 Comparator(s).
```

4.13.3 Diagrama ASM

El módulo no posee máquina de estados.

4.13.4 VHDL Original

El código de este módulo se encuentra en el archivo vga_prototype.vhd

4.14 cards

4.14.1 Hardware (Elementos)

- Señales internas:
    - 1 máquina de estados con cinco estados
        - SET_RAM
        - WAIT_FOR_SCREEN_END
        - CHECK_CARDS
        - MOD_CARD
        - WAIT_FOR_UPDATE
    - 1 contador ascendente con carga paralela de 8 bits para dirigir el acceso a memoria RAM cuando esta se lee o modifica durante los algoritmos del módulo: mod_counter.

    - 1 contador ascendente con carga paralela de 8 bits para dirigir el acceso a memoria RAM cuando se lee para mostrar el color de la carta en el pixel que representa el contador: vga_addr_counter.
    - 1 contador ascendente/descendente para contar las cartas durante los algoritmos de creación y modificación de cartas: card_counter.
    - 1 multiplexor de 3 a 1 de 8 bits para unir los vectores de direccionamiento de la memoria RAM: addr.
    - 1 multiplexor de 2 a 1 de 3 bits para unir los vectores de selección de sprite de carta: sprite_num.
    - 1 vector de 10 bits direccionamiento de memoria ROM de sprites: sprite_addr.
    - 3 señales de dibujo de sprite de carta:
        - Señal para el sprite de nivel 1: draw_sprite_1.
        - Señal para el sprite de nivel 2: draw_sprite_2.
        - Señal para el sprite de nivel 3: draw_sprite_3.
    - 1 señal de control de escritura en memoria RAM: wB_en.
    - 1 señal de colisión de bola y cartas a su alrededor: collision.
    - 1 señal de fin de la región dibujable de la pantalla: screen_end.
    - 3 vectores de 8 bits de color de sprite:
        - Vector de color del sprite de nivel 1: rgb_1.
        - Vector de color del sprite de nivel 2: rgb_2.
        - Vector de color del sprite de nivel 3: rgb_3.
    - 1 vector de 5 bits de entrada de datos de la memoria RAM: dB_in.
    - 1 vector de 5 bits de salida de datos de la memoria RAM: dB_out.
    - 1 señal de bit aleatorio: rng.
    - 1 vector de 9 bits de posición relativa de la bola con respecto de la matriz de cartas: pos_ball_card_array.
- Componentes internos:
    - 3 módulos de sprites de cartas:
    - Sprites de cartas de nivel 1: cards_sprite_1.
        - Sprites de cartas de nivel 2: cards_sprite_2.
        - Sprites de cartas de nivel 3: cards_sprite_3.
        - 1 submódulo de generación de bits aleatorios: random_bit_gen.
- Descripción de los estados:
    - SET_RAM
      Este es el estado inicial, y corresponde al algoritmo de generación de la matriz de cartas. Va recorriendo los valores desde 40 a 239 para rellenar el panel con hasta 200 cartas. Este panel se guarda en una memoria RAM de 256 palabras de 5 bits. Durante este estado se recorre la RAM con el vector de dirección mod_counter, que empieza en 40, insertando 31 o 0 en cada palabra según el bit aleatorio sea 1 o 0 respectivamente, siendo 31 la carta por defecto y 0 la ausencia de carta/carta volteada. Cuando se añade una carta, ademas aumenta el contador de cartas card_counter. Cuando el algoritmo termina, se pasa al estado WAIT_FOR_SCREEN_END. Mientras se realiza el algoritmo se mantiene en SET_RAM.
    - WAIT_FOR_SCREEN_END
      En este estado se espera a que el módulo de vga haya recorrido la parte visible de la pantalla. Tras esto tenemos asegurados 36160 ciclos de 25 MHz para realizar operaciones sin que ocurran fallos de concurrencia

con la visualización de las cartas. Cuando se produce el fin de la pantalla visible se pasa al estado CHECK_CARDS.

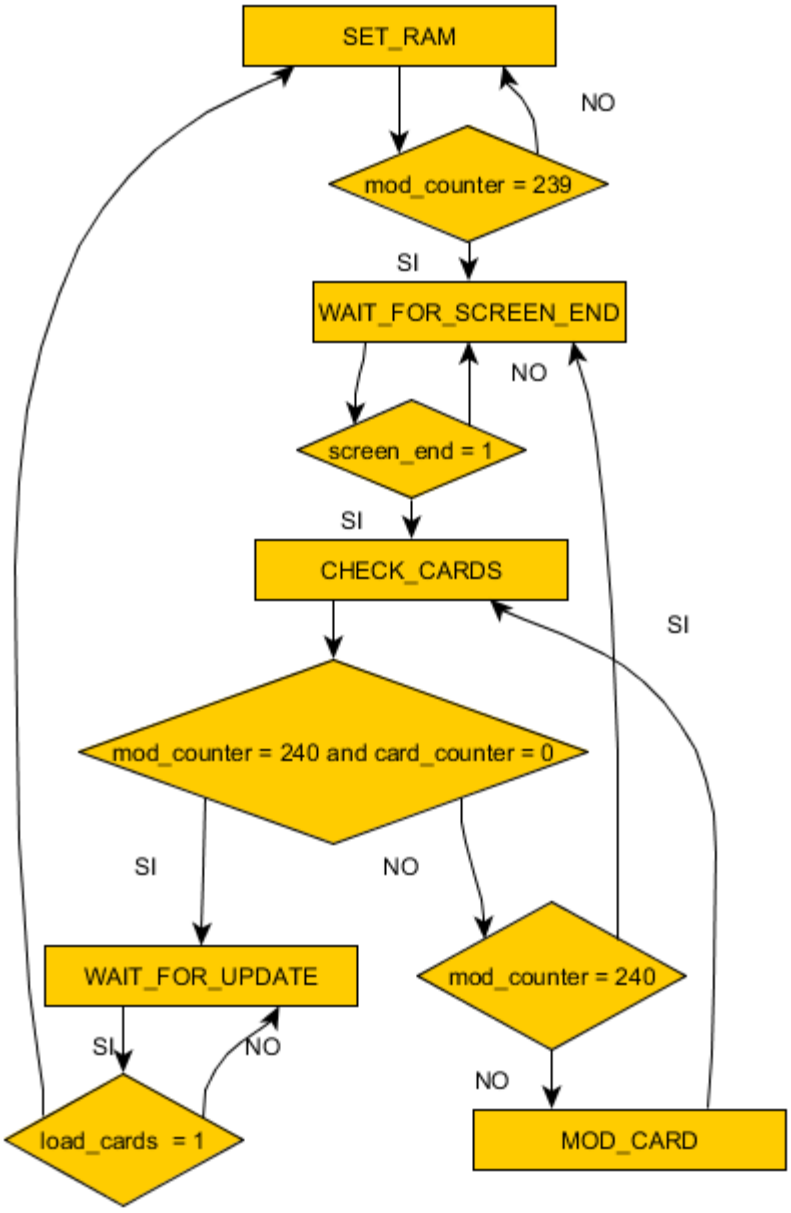
- CHECK_CARDS
  Representa el estado inicial de cada iteración del algoritmo de modificación y comprobación de cartas. Se recorren las cartas de la misma forma que se hizo en SET_RAM. En este estado se manda leer el contenido de la celda indicada por mod_counter. La lectura será efectiva en el siguiente ciclo. Si no se han procesado todas las cartas se procede al estado MOD_CARD. Si se ha finalizado el algoritmo y todavia quedan cartas se pasa al estado WAIT_FOR_SCREEN_END. Si, en cambio, se han volteado todas las cartas entonces se pasa al estado WAIT_FOR_UPDATE.
- MOD_CARD
  En este estado se manda a escribir en la celda anteriormente leida el valor adecuado para la carta. Si la carta estaba en su estado por defecto (31) y no colisiona con la bola, se mantiene igual. Si colisiona con la bola, se le resta uno (pasando a 30). Si la carta no estaba en el valor por defecto diferenciamos dos casos, la carta tenía valor 0 o tenía valor 1 hasta 30. Si tenía valor 0 entonces no habia carta, y por tanto se mantiene a cero. Si se daba el otro caso distinguimos otros dos casos, si tenia valor superior a uno se resta uno y se continua; si tenia valor uno se resta uno y se decrementa el contador de cartas card_counter. El motivo de tener cartas con 5 bits es el de poder animar el giro de las cartas mediante conteo de frames. El algoritmo que acabo de explicar baja los contadores, produciendo la animación de cada carta en cada frame. En este estado tambien se aumenta la dirección mod_counter. Tras este estado se pasa a CHECK_CARDS para continuar el algoritmo.
- WAIT_FOR_UPDATE
  En este estado se realiza la comunicación entre la maquina de estados principal y la actual. Se espera a que la máquina de estados principal le avise a la actual que comience el algoritmo de carga de cartas, pasando al estado SET_RAM. Mientras esto no suceda se mantiene en el estado WAIT_FOR_UPDATE.

4.14.2 Hardware (FPGA)

Summary:

```
inferred   1 Finite State Machine(s).
inferred   2 Counter(s).
inferred   8 D-type flip-flop(s).
inferred   8 Adder/Subtractor(s).
inferred   8 Comparator(s).
```

4.14.3 Diagrama ASM



4.14.4 VHDL Original

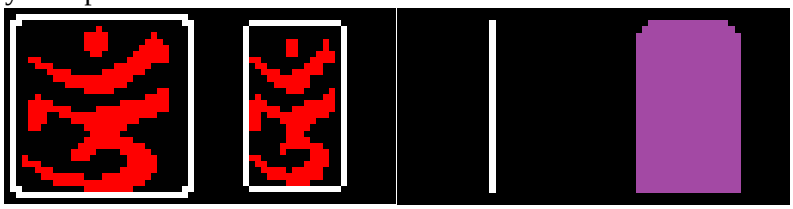
El código de este módulo se encuentra en el archivo cards2.vhd

4.15 cards_sprite_1

4.15.1 Hardware (Elementos)

- Señales internas:
  - 4 ROM de 1024 x 2 bits
    - ROM para el primer sprite de cartas de nivel 1: sprite_c4.
    - ROM para el segundo sprite de cartas de nivel 1: sprite_c3.
    - ROM para el tercer sprite de cartas de nivel 1: sprite_c2.
    - ROM para el cuarto sprite de cartas de nivel 1: sprite_c1.
  - 1 multiplexor de 4 a 1 de 2 bits para unir los colores de las memorias ROM: rgb_dec.
  - 4 vectores de 2 bits con los colores de las memorias ROM:

        - Color para el primer sprite de cartas de nivel 1: rgb_c4.
        - Color para el segundo sprite de cartas de nivel 1: rgb_c3.
        - Color para el tercer sprite de cartas de nivel 1: rgb_c2.
        - Color para el cuarto sprite de cartas de nivel 1: rgb_c1.
- Descripción del funcionamiento:
  El módulo guarda en cuatro ROM de 1024 palabras de 2 bits las siguientes imagenes codificando mediante una codificación mínima de negro, rojo, blanco y transparencia.



4.15.2 Hardware (FPGA)

Summary:

inferred 4 ROM(s).

4.15.3 Diagrama ASM

El módulo es combinacional

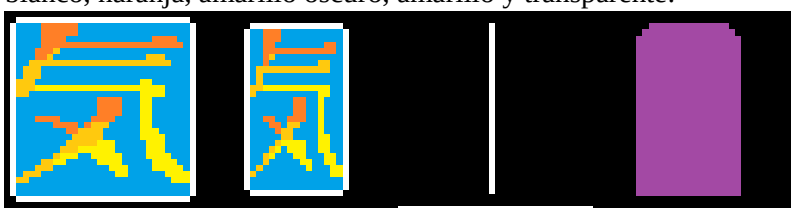
4.15.4 VHDL Original

El código de este módulo se encuentra en el archivo cards_sprite.vhd

4.16 cards_sprite_2

4.16.1 Hardware (Elementos)

- Señales internas:
  - 4 ROM de 1024 x 3 bits
    - ROM para el primer sprite de cartas de nivel 1: sprite_c4.
    - ROM para el segundo sprite de cartas de nivel 1: sprite_c3.
    - ROM para el tercer sprite de cartas de nivel 1: sprite_c2.
    - ROM para el cuarto sprite de cartas de nivel 1: sprite_c1.
  - 1 multiplexor de 4 a 1 de 3 bits para unir los colores de las memorias ROM: rgb_dec.
  - 4 vectores de 3 bits con los colores de las memorias ROM:
    - Color para el primer sprite de cartas de nivel 1: rgb_c4.
    - Color para el segundo sprite de cartas de nivel 1: rgb_c3.
    - Color para el tercer sprite de cartas de nivel 1: rgb_c2.
    - Color para el cuarto sprite de cartas de nivel 1: rgb_c1.
- Descripción del funcionamiento:
  El módulo guarda en cuatro ROM de 1024 palabras de 3 bits las siguientes imagenes codificando mediante una codificación mínima de negro, azul claro, blanco, naranja, amarillo oscuro, amarillo y transparente.



4.16.2 Hardware (FPGA)

Summary:

inferred 4 ROM(s).

4.16.3 Diagrama ASM

El módulo es combinacional

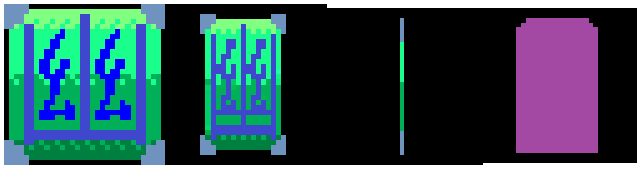
4.16.4 VHDL Original

El código de este módulo se encuentra en el archivo cards_sprite_2.vhd

4.17 cards_sprite_3

4.17.1 Hardware (Elementos)

- Señales internas:
  - 4 ROM de 1024 x 4 bits
    - ROM para el primer sprite de cartas de nivel 1: sprite_c4.
    - ROM para el segundo sprite de cartas de nivel 1: sprite_c3.
    - ROM para el tercer sprite de cartas de nivel 1: sprite_c2.
    - ROM para el cuarto sprite de cartas de nivel 1: sprite_c1.
  - 1 multiplexor de 4 a 1 de 4 bits para unir los colores de las memorias ROM: rgb_dec.
  - 4 vectores de 4 bits con los colores de las memorias ROM:
    - Color para el primer sprite de cartas de nivel 1: rgb_c4.
    - Color para el segundo sprite de cartas de nivel 1: rgb_c3.
    - Color para el tercer sprite de cartas de nivel 1: rgb_c2.
    - Color para el cuarto sprite de cartas de nivel 1: rgb_c1.
- Descripción del funcionamiento:
  El módulo guarda en cuatro ROM de 1024 palabras de 4 bits las siguientes imagenes codificando mediante una codificación mínima de negro, azul gris, verde claro, verde lima, morado, verde oscuro, verde, azul y transparente.



4.17.2 Hardware (FPGA)

Summary:

inferred 4 ROM(s).

4.17.3 Diagrama ASM

El módulo es combinacional

4.17.4 VHDL Original

El código de este módulo se encuentra en el archivo cards_sprite_3.vhd

4.18 RAM

4.18.1 Hardware (Elementos)

- Señales internas:
  - 1 RAM de 256 x 5 bits: memory.
  - 1 registro de 5 bits para hacer una RAM con salida sincrona: dout.
- Descripción del funcionamiento:

El módulo recibe una dirección, un dato de entrada y una señal de habilitación de escritura y realiza la operación requerida, obteniendo el resultado en el ciclo siguiente, guardando el resultado de la lectura en un registro intermendio. La RAM prioriza la lectura, obteniendo el valor antiguo de una posición cuando se solicita una escritura en esa posición.

4.18.2 Hardware(FPGA)

Summary:

inferred 1 RAM(s).
inferred 5 D-type flip-flop(s).

4.18.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.18.4 VHDL Original

El código de este módulo se encuentra en el archivo RAM.vhd

4.19 random_bit_gen

4.19.1 Hardware (Elementos)

- Señales internas:
  - 1 registro con desplazamiento de 12 bitsa izquierda para generar la secuencia aleatoria: pipeline.
  - 1 xor de 4 bits para generar la aleatoriedad.
- Descripción del funcionamiento:
  En este módulo se implementa un linear feedback shift register de Fibonacci de 12 bits con el polinomio x^12 + x^11 + x^10 + x^4 + 1 con 4095 estados posibles, si se comienza con un valor inicial distinto al 0 de 12 bits. Esto nos permite generar bits pseudoaleatorios. Dado que a este módulo no le afecta el reset, en la práctica es impredecible el estado que posee en cada momento.

4.19.2 Hardware (FPGA)

Summary:

inferred 12 D-type flip-flop(s).
inferred 1 Xor(s).

4.19.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.19.4 VHDL Original

El código de este módulo se encuentra en el archivo random_bit_gen.vhd

4.20 bola

4.20.1 Hardware (Elementos)

- Señales internas:
  - 1 máquina de estados con 4 estados.
    - Inicio
    - Mover
    - Atacado
    - Invulnerable
  - 1 registro de 10 bits para guardar la posición horizontal de la bola: px.
  - 1 registro de 9 bits para guardar la posición vertical de la bola: py.

    - 1 registro de 11 bits para guardar la velocidad con signo horizontal de la bola: vx.
    - 1 registro de 11 bits para guardar la velocidad con signo vertical de la bola: vy.
    - 1 multiplexor de 2 a 1 de 2 bits para unir los colores de la bola: rgb_dec.
    - 1 vector de 2 bits de color de bola codificado en el minimo tamaño: colour_ball.
    - 1 vector de direccionamiento de 10 bits de la memoria de sprite: x_sprite.
    - 1 contador ascendente/descendente de 6 bits de selector de orientación de sprite por direccion de movimiento: anim_c.
- Componentes internos:
    - 1 sprite de bola: ball_sprite.
- Descripción de estados:
    - Inicio
      Durante este estado la bola permanece quieta en la posición inicial con componente x 600 y componente y 200. Se mantiene en este estado hasta que se activa el boton de inicio (start_b), entonces pasa al estado Mover.
    - Mover
      En este estado se producen los movimientos marcados por la velocidad x e y, además de cambiar la posición de la bola, se modifican las velocidades. Si la bola impacta con alguna pared, la componente de velocidad del eje ortogonal a la pared se multiplica por 3/4 y cambia de dirección. Esto es, pierde fuerza y rebota. Además de este efecto, la velocidad y tiende hacia abajo, simulando la gravedad. Es preciso indicar que, dadas las propiedades de las coordenadas de la pantalla, una velocidad positiva en x implica movimiento hacia la derecha, y negativa hacia la izquierda; en el caso de la velocidad y se procede en contra de la intuición, por lo que una velocidad negativa implica movimiento hacia arriba y una velocidad positiva implica movimiento hacia abajo. Se habia planteado reducir el limite de velocidades, pero esto no ha sido necesario debido a la perdida de energia al chocar con las paredes. El módulo previene que la pelota se salga de los límites de la pantalla cambiando la posicion de la bola a la del borde (teniendo en cuenta anchura y altura de la bola), y realizando el cambio de dirección en la velocidad como se ha indicado antes. Este estado perdura hasta que la bola colisiona con el personaje y el personaje estaba realizando un ataque.
    - Atacado
      Este estado representa un ataque por parte del personaje que ha afectado a la bola. Durante este estado se aumenta la velocidad en las dos componentes. Si la bola tenía velocidad x hacia la izquierda, se invierte la velocidad actual y se añade una constante de fuerza. Si la velocidad x tenia direccion hacia la derecha, se realiza el mismo procedimiento, pero restando la constante de fuerza. Si la velocidad y tenia direccion hacia arriba se mantiene la dirección y se resta una constante de fuerza. Si la velocidad y tenia direccion hacia abajo se invierte la dirección y se resta una constante de fuerza. Tras este estado

se pasa al estado invulnerable si sigue estando atacado, o al estado Mover si ya no esta siendo atacado.

- Invulnerable
  Este estado comparte transiciones con el estado Atacado, y realiza los movimientos igual que el estado Mover. Este estado existe para que la bola no pueda ser golpeada justo despues de ya haber sido golpeada (sin separarse del personaje).

El dibujo del sprite de la bola se realiza en distintas direcciónes dependiendo de la dirección de la velocidad x, para que se genere una animación de giro.

4.20.2 Hardware(FPGA)

Summary:

```
inferred   1 Finite State Machine(s).
inferred   1 ROM(s).
inferred   1 Counter(s).
inferred  23 Adder/Subtractor(s).
inferred   2 Multiplier(s).
inferred  16 Comparator(s).
```

La ROM que aparece en el Synthesis Report es generada al traducir la codificación de 2 bits de los colores de la bola a los usuales 9 bits de rgb.

4.20.3 Diagrama ASM

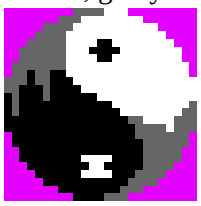
El código de este módulo se encuentra en el archivo bola.vhd

4.20.4 VHDL Original

4.21 ball_sprite

4.21.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 1024 x 2 bits
- Descripción del funcionamiento:
  El módulo guarda el sprite de la bola codificado en 2 bits, representando negro, blanco, gris y transparente.



4.21.2 Hardware (FPGA)

Summary:

```
inferred   1 ROM(s).
```

4.21.3 Diagrama ASM

El módulo es combinacional.

4.21.4 VHDL Original

El código de este módulo se encuentra en el archivo ball_sprite.vhd

4.22 Sound

4.22.1 Hardware (Elementos)

- Señales internas:
    - 1 contador ascendente de 7 bits para contar la posición en las partituras de instrumentos bass y vibra 1: pace_director.
    - 1 contador ascendente de 9 bits para contar la posición en las partituras del instrumento vibra 2: pace director_2.
    - 2 vectores de 18 bits para indicar los limites de frecuencia de modulación para los instrumentos bass 1 y 2:
        - Vector para el instrumento bass 1: clk_bass_1_limit.
        - Vector para el instrumento bass 2: clk_bass_2_limit.
    - 3 vectores de 16 bits para indicar los limites de frecuencia de modulación para los instrumentos vibra 1, lead 1 y lead 2:
        - Vector para el instrumento vibra 1: clk_vibra_limit.
        - Vector para el instrumento lead 1: clk_lead_1_limit.
        - Vector para el instrumento lead 2: clk_lead_2_limit.
    - 1 vector de 15 bits para indicar los limites de frecuencia de modulación para el instrumento vibra 2: clk_vibra_2_limit.
    - 1 vector de 16 bits para indicar los limites de frecuencia de modulación para el instrumento bass 3: clk_bass_3_limit.
    - 1 vector de 17 bits para indicar los limites de frecuencia de modulación para el instrumento bass 4: clk_bass_4_limit.
    - 3 señales de clock enable para marcar el cambio de nota según tres metronomos diferentes:
        - Señal de metrónomo de 0,8 segundos: clk_pace_1.
        - Señal de metrónomo de 0,4 segundos: clk_pace_2.
        - Señal de metrónomo de 0,2 segundos: clk_pace_3.
    - 5 señales de canal de sonido: channel_1, channel_2, channel_3, channel_4, channel_5.
    - 2 contadores ascendentes de 4 bits para direcciónar el acceso a las partituras de los instrumentos bass y vibra 1:
        - Contador para los instrumentos bass 1 y 2: note_bass_addr.
        - Contador para los instrumentos bass 3 y 4: note_bass_3_addr.
        - Contador para el instrumento vibra 1: note_vibra_addr.
    - 1 contador ascendente de 5 bits para direccionar el acceso a las partituras de los instrumentos lead: note_lead_addr.
    - 1 contador ascendente de 6 bits para direccionar el acceso a la partitura del instrumento vibra 2: note_vibra_2_addr.
    - 1 contador ascendente de 13 bits para el mezclado de canales de audio: clk_switch_counter.
    - 2 multiplexores de 2 a 1 de 18 bits para mezclar los limites de frecuencia que usan el canal 1 y 2:
        - Multiplexor para el canal 1: ch_1_limit.
        - Multiplexor para el canal 2: ch_2_limit.
    - 3 multiplexores de 2 a 1 de 16 bits para mezclar los limites de frecuencia que usan el canal 3, 4 y 5:
        - Multiplexor para el canal 3: ch_3_limit.

        - Multiplexor para el canal 4: ch_4_limit.
        - Multiplexor para el canal 5: ch_5_limit.
    - 1 contador ascendente de 26 bits para conteo de ciclos para generar un metronomo de 0,8 segundos: pace_1.
    - 2 contador ascendente de 23 bits para conteo de ciclos para generar un metronomo de 0,4 y un metronomo de 0,2 segundos:
        - Contador de metronomo de 0,4 segundos: pace_2.
        - Contador de metronomo de 0,2 segundos: pace_3.
- Componentes internos:
    - 5 generadores de notas por modulación por frecuencia: PFM.
    - 1 partitura de instrumento bass 1: bass_1_rom.
    - 1 partitura de instrumento bass 2: bass_2_rom.
    - 1 partitura de instrumento bass 3: bass_3_rom.
    - 1 partitura de instrumento bass 4: bass_4_rom.
    - 1 partitura de instrumento vibra 1: vibra_1_rom.
    - 1 partitura de instrumento vibra 2: vibra_2_rom.
    - 1 partitura de instrumento lead 1: lead_1_rom.
    - 1 partitura de instrumento lead 2: lead_2_rom.
- Descripción del funcionamiento:
  El módulo genera cinco canales de audio, los cuales va alternando en partes desiguales, dando prioridad 1/4 de tiempo a cada uno de los tres primeros canales, 1/16 a cada uno de los dos canales restantes, y 1/8 en silencio para aminorar el efecto ruido de alternar los canales rapidamente.
  En el canal 1 se emiten las señales de bass 1 y bass 3, que se multiplexan ya que en la partitura original no se tocan a la vez.
  En el canal 2 se emiten las señales de bass 2 y bass 4, que se multiplexan ya que en la partitura original no se tocan a la vez.
  En el canal 3 se emiten las señales de vibra 1 y vibra 3, que se multiplexan ya que en la partitura original no se tocan a la vez.
  En el canal 4 se emite la señal de lead 1.
  En el canal 5 se emite la señal de lead 2.
  Las ondas en los canales de audio se generan mediante modulos de PFM. El sistema de pulso modulado por frecuencia (Pulse Frequency Modulation) consiste en generar un reloj de la misma frecuencia que la nota que se quiere reproducir, recreando el sonido de forma parecida al real. Para utilizar estos modulos es necesario indicar la mitad de la frecuencia que se quiere usar.
  Las frecuencias que usan los PFM se guardan en memorias ROM en cada instrumento. Estas memorias guardan estas frecuencias (y por tanto las notas de la partitura) codificadas en la menor codificación para las notas usadas en cada instrumento. Como las frecuencias pueden llegar a ocupar alrededor de 18 bits, la codificación en entre 2 y 3 bits permite ahorrar mucha memoria.
  Para leer las frecuencias se usan metronomos, generados por contadores de ciclos. En este caso tenemos tres metronomos. Cada instrumento usa un metronomo diferente con el fin de no tener que repetir muchas notas en las memorias.
  Los instrumentos que cambian de nota más rapido usan un metronomo más rapido, mientras que los más lentos usan el metronomo de 0,8 segundos.
  Para reducir aún más el consumo de memoria, se han reducido las partituras, aprovechando los trozos que se repiten. Para poder usar las repeticiones

correctamente se usan dos directores de ritmo, pace_director y pace_director_2. Estos cuentan la posición en la partitura original, y reinician los contadores de direccionamiento de los diferentes instrumentos de forma coordinada.

4.22.2 Hardware (FPGA)

Summary:

```
inferred  10 Counter(s).
inferred   4 D-type flip-flop(s).
inferred   1 Adder/Subtractor(s).
inferred  12 Comparator(s).
```

4.22.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.22.4 VHDL Original

El código de este módulo se encuentra en el archivo Sound.vhd

4.23 PFM

4.23.1 Hardware (Elementos)

- Señales internas:
  - 1 contador de n bits para generar el reloj dado la media frecuencia: counter.
  - 1 señal de reloj con frecuencia el doble del límite dado: aux.
- Descripción del funcionamiento:

  El módulo genera una señal de reloj con frecuencia el doble del número representado por limit. La estructura de este módulo es análoga a la de un divisor de frecuencia, solo que la frecuencia no esta prefijada al compilar, sino que cambia a petición del usuario del módulo.

4.23.2 Hardware (FPGA)

Summary:

```
inferred   1 Counter(s).
inferred   1 D-type flip-flop(s).
inferred   1 Comparator(s).
```

4.23.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.23.4 VHDL Original

El código de este módulo se encuentra en el archivo PFM.vhd

4.24 bass_1_rom

4.24.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 16 x 2 bits de partitura.
  - 1 ROM de 4 x 18 bits de relación codificación-frequencia.
  - 1 vector de 2 bits de sincronización de salida de ROM: note_bass_1.
- Descripción del funcionamiento:

El módulo guarda la partitura comprimida del instrumento bass 1, con una ROM síncrona para asegurar señales estables, y codificando las notas mediante dos bits para cuatro notas posibles.

4.24.2 Hardware (FPGA)

Summary:

```
inferred   2 ROM(s).
inferred   2 D-type flip-flop(s).
```

4.24.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.24.4 VHDL Original

El código de este módulo se encuentra en el archivo bass_1_rom.vhd

4.25 bass_2_rom

4.25.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 16 x 2 bits de partitura.
  - 1 ROM de 4 x 18 bits de relación codificación-frequencia.
  - 1 vector de 2 bits de sincronización de salida de ROM: note_bass_2.
- Descripción del funcionamiento:

  El módulo guarda la partitura comprimida del instrumento bass 2, con una ROM síncrona para asegurar señales estables, y codificando las notas mediante dos bits para cuatro notas posibles.

4.25.2 Hardware (FPGA)

Summary:

```
inferred   2 ROM(s).
inferred   2 D-type flip-flop(s).
```

4.25.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.25.4 VHDL Original

El código de este módulo se encuentra en el archivo b_2_rom.vhd

4.26 bass_3_rom

4.26.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 16 x 2 bits de partitura.
  - 1 ROM de 4 x 16 bits de relación codificación-frequencia.
  - 1 vector de 2 bits de sincronización de salida de ROM: note_bass_3.
- Descripción del funcionamiento:

  El módulo guarda la partitura comprimida del instrumento bass 3, con una ROM síncrona para asegurar señales estables, y codificando las notas mediante dos bits para cuatro notas posibles.

4.26.2 Hardware (FPGA)

Summary:

inferred 2 ROM(s).
inferred 2 D-type flip-flop(s).

4.26.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.26.4 VHDL Original

El código de este módulo se encuentra en el archivo bass_3_rom.vhd

4.27 bass_4_rom

4.27.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 16 x 2 bits de partitura.
  - 1 ROM de 4 x 17 bits de relación codificación-frequencia.
  - 1 vector de 2 bits de sincronización de salida de ROM: note_bass_4.
- Descripción del funcionamiento:

  El módulo guarda la partitura comprimida del instrumento bass 4, con una ROM síncrona para asegurar señales estables, y codificando las notas mediante dos bits para cuatro notas posibles.

4.27.2 Hardware (FPGA)

Summary:

inferred 2 ROM(s).
inferred 2 D-type flip-flop(s).

4.27.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.27.4 VHDL Original

El código de este módulo se encuentra en el archivo baass_4_rom.vhd

4.28 vibra_1_rom

4.28.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 16 x 3 bits de partitura.
  - 1 ROM de 6 x 16 bits de relación codificación-frequencia.
  - 1 vector de 3 bits de sincronización de salida de ROM: note_vibra.
- Descripción del funcionamiento:

  El módulo guarda la partitura comprimida del instrumento vibra 1, con una ROM síncrona para asegurar señales estables, y codificando las notas mediante tres bits para seis notas posibles.

4.28.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).
inferred 3 D-type flip-flop(s).

Por algún motivo no se sintetiza la ROM de traducción a notas, aunque el código es análogo a los anteriores.

4.28.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.28.4 VHDL Original

El código de este módulo se encuentra en el archivo vibra_1_rom.vhd

4.29 vibra_2_rom

4.29.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 64 x 3 bits de partitura.
  - 1 ROM de 8 x 15 bits de relación codificación-frequencia.
  - 1 vector de 3 bits de sincronización de salida de ROM: note_vibra.
- Descripción del funcionamiento:

  El módulo guarda la partitura comprimida del instrumento vibra 2, con una ROM síncrona para asegurar señales estables, y codificando las notas mediante tres bits para ocho notas posibles.

4.29.2 Hardware (FPGA)

Summary:

inferred 2 ROM(s).
inferred 3 D-type flip-flop(s).

4.29.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.29.4 VHDL Original

El código de este módulo se encuentra en el archivo vibra_2_rom.vhd

4.30 lead_1_rom

4.30.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 32 x 3 bits de partitura.
  - 1 ROM de 7 x 16 bits de relación codificación-frequencia.
  - 1 vector de 3 bits de sincronización de salida de ROM: note_vibra.
- Descripción del funcionamiento:

  El módulo guarda la partitura comprimida del instrumento lead 1, con una ROM síncrona para asegurar señales estables, y codificando las notas mediante tres bits para siete notas posibles.

4.30.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).
inferred 3 D-type flip-flop(s).

4.30.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.30.4 VHDL Original

El código de este módulo se encuentra en el archivo lead_1_rom.vhd

4.31 lead_2_rom

4.31.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 32 x 3 bits de partitura.
  - 1 ROM de 6 x 16 bits de relación codificación-frequencia.
  - 1 vector de 3 bits de sincronización de salida de ROM: note_vibra.
- Descripción del funcionamiento:

  El módulo guarda la partitura comprimida del instrumento lead 2, con una ROM síncrona para asegurar señales estables, y codificando las notas mediante tres bits para siete notas posibles.

4.31.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).
inferred 3 D-type flip-flop(s).

4.31.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.31.4 VHDL Original

El código de este módulo se encuentra en el archivo lead_2_rom.vhd

4.32 overlay

4.32.1 Hardware (Elementos)

- Señales internas:
  - 1 máquina de estados con 3 estados
    - WAIT_FOR_CARD_END
    - START_CALCULATE
    - WAIT_CALCULATE
  - 1 señal de inicio de algoritmo de conversión: start.
  - 1 señal de fin de algoritmo de conversión: finish.
  - 1 contador de 16 bits para direccionar el acceso a la ROM del sprite de overlay: addr_c.
  - 1 señal de dibujo de overlay: draw_overlay.
  - 1 señal de dibujo de de número: draw_number.
  - 1 señal de comienzo de la zona segura de ejecución del algoritmo de conversión.
  - 1 señal de dibujo de sprite de vida: draw_life.
  - 1 señal de activación de sprite de vida: level_dec.

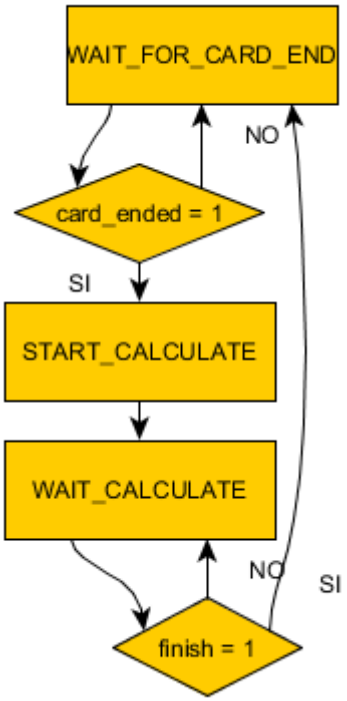
    - 1 multiplexor de 9 a 1 de 4 bits para mezclar los números de BCD en la ROM de números: number.
    - 1 vector de 8 bits de direccionamiento de ROM de sprite de vidas.
    - 1 vector de 8 bits de direccionamiento de ROM de sprite de números.
    - 1 vector de 28 bits de número BCD traducido: output.
    - 1 vector de 9 bits de color a dibujar: rgb_life.
- Componentes internos:
    - ROM de sprite de vidas: lives_sprite.
    - ROMs de traducción de BCD a sprites de dígitos: digits.
    - ROM de sprite de overlay: overlay_sprite.
    - Módulo de algoritmo de traducción binario a BCD: BCD
- Descripción de estados:
    - WAIT_FOR_CARD_END
      Este es el estado inicial, que representa la espera hasta que la lectura de pantalla se encuentre en el pixel (0,500), cuando el algoritmo de modificación de cartas ha terminado y ya se han sumado todos los puntos. Si llega este momento se pasa al estado START_CALCULATE, sino se mantiene en el estado WAIT_FOR_CARD_END.
    - START_CALCULATE
      En este estado se ordena el comienzo del algoritmo de paso de binario a BCD al modulo BCD, mediante la señal de start. Tras este estado se pasa a WAIT_CALCULATE
    - WAIT_CALCULATE
      En este estado se espera a la finalización del algoritmo de conversión. Cuando llegue en activo la señal finish se pasa al estado WAIT_FOR_CARD_END, en caso contrario se mantiene en el estado WAIT_CALCULATE.
- Descripción del funcionamiento:
  Como se ha indicado antes, se realiza la conversión de binario a BCD controlado por la máquina de estados. El resto del módulo se encarga de proporcionar la salida gráfica correcta. Recibiendo la posición de lectura de la VGA determinamos que ha de dibujarse. Las vidas se dibujan en un rectángulo con esquina superior izquierda (128,0) e inferior derecha (223,15), y se dibuja un sprite de 16 x 16 por cada vida que se tenga. Los dígitos de puntos se muestran en un rectángulo con esquina superior izquierda (256,0) e inferior derecha (367,13). El dígito de nivel se muestra en un rectángulo con esquina superior izquierda (608,33) e inferior derecha (623,46). Para direccionar los dígitos se usan multiplicadores junto con subcadenas de vga_x y vga_y. El direccionamiento de las vidas solo consiste en concatenación de vga_y y vga_x dando que están colocados en múltiplos de 16. Es sprite del overlay es muy grande, por lo que en vez de usar multiplicadores se usa un contador indicando el pixel a dibujar. El overlay ocupa la banda superior con vga_y menor que 64 pixeles.
  El módulo de overlay también se encarga de mostrar la pantalla intermedia, ocultando los puntos y el nivel en el panel superior y mostrando los puntos en un rectángulo gris en el centro izquierda de la pantalla, con esquinas (64,128) y (319,173).

4.32.2 Hardware (FPGA)

Summary:

```
inferred   1 Finite State Machine(s).
inferred   1 Counter(s).
inferred   2 Adder/Subtractor(s).
inferred   2 Multiplier(s).
inferred  33 Comparator(s).
```

4.32.3 Diagrama ASM



4.32.4 VHDL Original

El código de este módulo se encuentra en el archivo Overlay.vhd

4.33 lives_sprite

4.33.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 256 x 3 bits de sprite de vidas.
  - 1 ROM de 5 x 9 bits de decodificación a RGB de 9 bits.

El módulo guarda el sprite de vidas de 16 x 16 pixeles, usando una codificación de 3 bits para los colores blanco, negro, rojo, beige, gris y transparente.

4.33.2 Hardware (FPGA)

Summary:

```
inferred   1 ROM(s).
```

4.33.3 Diagrama ASM

El módulo es combinacional.

4.33.4 VHDL Original

El código de este módulo se encuentra en el archivo lives_sprite.vhd

4.34 digits

4.34.1 Hardware (Elementos)

- Señales internas:
  - 10 ROM de 182 x 1 bit de sprites de dígitos.
  - 1 multiplexor de 10 a 1 de selección de ROM.

El módulo tiene 10 ROM, una por cada dígito decimal, y se muestra la ROM seleccionada mediante el número BCD de la entrada.

4.34.2 Hardware (FPGA)

Summary:

inferred 10 ROM(s).

4.34.3 Diagrama ASM

El módulo es combinacional.

4.34.4 VHDL Original

El código de este módulo se encuentra en el archivo digits.vhd

4.35 overlay_sprite

4.35.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 40960 x 1 bit de sprite de overlay.

4.35.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).

4.35.3 Diagrama ASM

El módulo es combinacional.

4.35.4 VHDL Original

El código de este módulo se encuentra en el archivo overlay_sprite.

4.36 BCD

4.36.1 Hardware (Elementos)

- Señales internas:
  - 1 máquina de estado con 4 estados:
    - IDLE
    - SET_UP
    - SHIFT
    - ADD
  - 1 registro con desplazamiento de 52 bits para computar los desplazamientos del algoritmo: work_reg.
  - 1 contador ascendente de 5 bits para marcar el ritmo del algoritmo: counter.

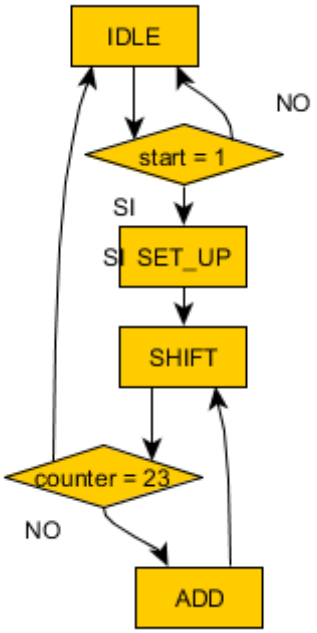
- Descripcion de estados:
    - IDLE
      El estado inicial, donde se espera a que se inicie el algoritmo activando la señal de inicio (start). En este estado se reinicia el contador del algoritmo a cero. Si se activa la señal se pasa al estado SET_UP, en otro caso se mantiene en IDLE.
    - SET_UP
      En este estado se carga el número en binario en los 24 bits menos significativos del registro con desplazamiento. Se pasa al estado SHIFT.
    - SHIFT
      En este estado se realiza un desplazamiento hacia la izquierda. Se aumenta en uno el valor del contador de algoritmo. Si se han realizado 23 anteriormente se pasa al estado IDLE. En otro caso se pasa al estado ADD.
    - ADD
      En este estado se suma 3 a cada digito BCD que supere el valor 4. Hay 7 digitos BCD en los 28 bits más significativos del registro de desplazamiento. Se pasa al estado SHIFT.
- Descripción del funcionamiento:
  El algoritmo implementado en este módulo es el llamado algoritmo de Double-dabble. Consiste en pasar un número binario de n bits al equivalente en BCD mediante n desplazamientos hacia la izquierda, sumando entre desplazamiento y desplazamiento un tres a cada cifra BCD cuando superen el valor 4, de forma que se propague el acarreo de cada cifra a la siguiente.

4.36.2 Hardware (FPGA)

Summary:

```
inferred   1 Finite State Machine(s).
inferred   1 Counter(s).
inferred  52 D-type flip-flop(s).
inferred   7 Adder/Subtractor(s).
inferred   7 Comparator(s).
```

4.36.3 Diagrama ASM



4.36.4 VHDL Original

El código de este módulo se encuentra en el archivo BCD.vhd

4.37 startscreen

4.37.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 300 x 1 bits.
- Descripción del funcionamiento:

  El módulo guarda la siguiente imagen de 20 x 15 pixeles, que se amplia a 640 x 480 pixeles multiplicando el ancho y alto del pixel por 32. La imagen se guarda con blancos y negros. Para el direccionamiento se suma la coordenada x a la coordenada y multiplicada por 20.

4.37.2 Hardware (FPGA)

Summary:

inferred 1 ROM(s).
inferred 2 Adder/Subtractor(s).

4.37.3 Diagrama ASM

El módulo es combinacional

4.37.4 VHDL Original

El código de este módulo se encuentra en el archivo startscreen.vhd

4.38 endscreen

4.38.1 Hardware (Elementos)

- Señales internas:
  - 2 ROM de 204 x 1 bits de pantalla de victoria y pantalla de derrota.
  - 1 multiplexor de 3 a 1 de 1 bit para juntar la salida de dibujo de las dos ROMs.
- Descripción del funcionamiento:

  El módulo guarda las siguientes imagenes de 34 x 6 pixeles, que se amplia a 640 x 480 pixeles. La imagen se guarda con blancos y negros.

4.38.2 Hardware (FPGA)

Summary:

```
inferred   2 ROM(s).
inferred   4 Adder/Subtractor(s).
inferred   4 Comparator(s).
```

4.38.3 Diagrama ASM

El módulo es combinacional

4.38.4 VHDL Original

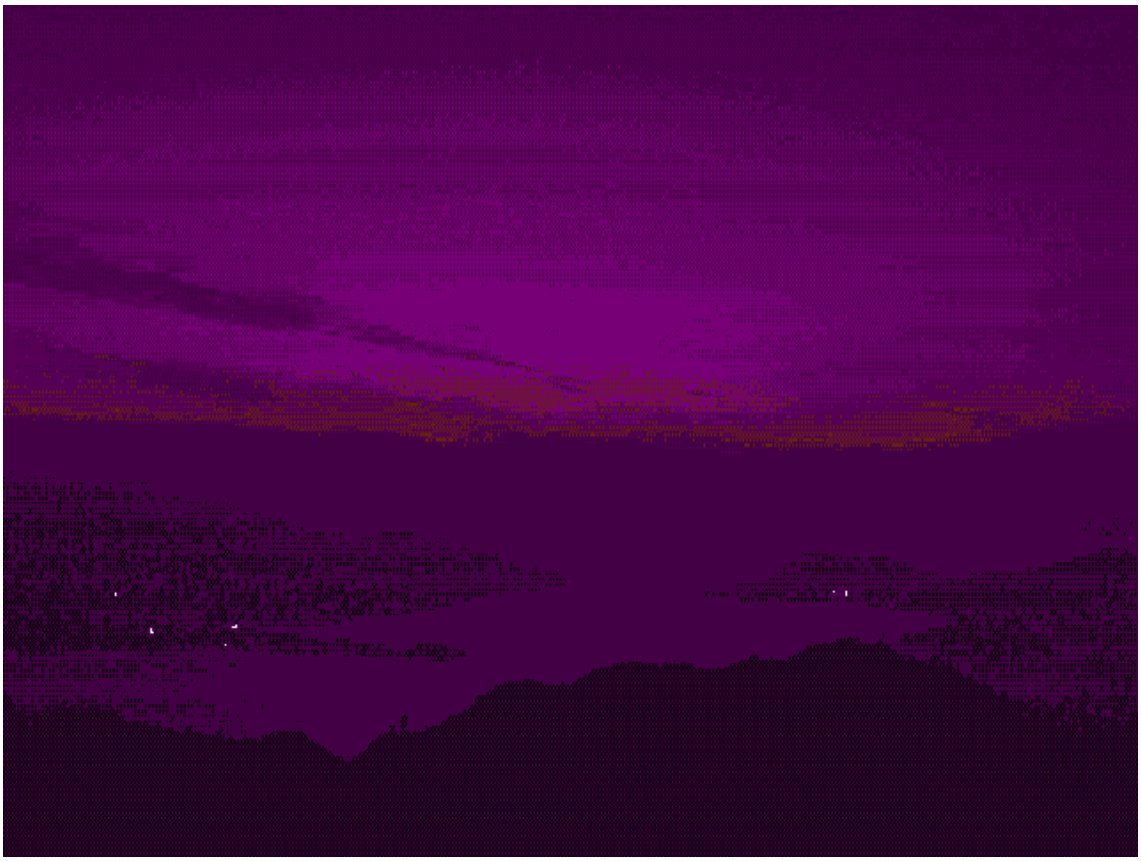
El código de este módulo se encuentra en el archivo endscreen.vhd

4.39 background_1

4.39.1 Hardware (Elementos)

- Señales internas:
  - 1 ROM de 307200 x 2 bits de pantalla de fondo.
  - 1 vector de 2 bits de decodificación de color: colour.
  - 1 contador ascendente de 19 bits para direccionar la ROM: addr_c.
- Descripción del funcionamiento:

  El módulo guarda la siguiente imagen de fondo. Para evitar multiplicadores muy grandes para direccionar la ROM, se usa un contador que marca el pixel a dibujar en la ROM, y que se reinicia en el flanco anterior al comienzo de la pantalla. La imagen se codifica con 2 bits y 4 colores, naranja, violeta, morado y negro.



4.39.2 Hardware (FPGA)

Summary:

```
inferred   2 ROM(s).
inferred   1 Counter(s).
inferred   2 D-type flip-flop(s).
inferred   2 Comparator(s).
```

4.39.3 Diagrama ASM

El módulo no tiene máquina de estados.

4.39.4 VHDL Original

El código de este módulo se encuentra en el archivo background_1.vhd

4.40 divisor

4.40.1 Hardware (Elementos)

- Señales internas:
  - 1 contador ascendente de n bits para contar los ciclos para generar el reloj dividido: cuenta.
  - 1 señal de reloj dividido: clk_aux.
- Descripción del funcionamiento:

  El módulo divide la señal de reloj de entrada para producir la señal de reloj de salida. Para esto se divide la señal de entrada entre la de salida. Esta valor entre 2 menos 1 es hasta el que tiene que contar el contador. Durante los valor/2 ciclos se mantiene una señal de reloj de salida en baja y los otros valor/2 ciclos se mantiene una señal de reloj de salida en alta, produciendo el

reloj de salida deseado. Para que el módulo funcione correctamente el máximo reloj de salida es ¼ del de entrada.

4.40.2 Hardware (FPGA)

Para un divisor de ¼ de frecuencia.

Summary:

```
inferred   1 T-type flip-flop(s).
inferred   1 D-type flip-flop(s).
```

4.40.3 Diagrama ASM

El módulo no tiene maquina de estados.

4.40.4 VHDL Original

El código de este módulo se encuentra en el archivo divisor.vhd

5 Nuestro proyecto es el mejor

A continuación cada uno de los integrantes expresa su opinión por la que considera nuestro proyecto el mejor.

Ángel Alonso:

Nuestro proyecto es el mejor y a continuación explicamos porqué. Ningún otro proyecto esta a la altura técnica ni ha sabido combinar tantos diferentes elementos en un producto que prácticamente desde el primer momento estaba funcionando. Además, no ha sido el primer proyecto en funcionar decentemente por implementar una idea excesivamente sencilla (al contrario, es una de las ideas más complicadas) si no por la gran capacidad de trabajo y esfuerzo de nuestro grupo. También se ha realizado un profundo trabajo de investigación para acercar al público general un juego japonés de culto pero poco conocido en occidente. Por todo esto, no me cabe ninguna duda de que nuestro proyecto es el mejor.

Xi Chen:

Nuestro proyecto es mejor.

1. Nuestro proyecto es una copia de un juego tradicional fabricado por una empresa japonesa. El juego original se vende bastante y tiene una lógica bien diseñada.

2. Se practica el equilibrio y colaboración entre manos y ojos. Tiene 3 niveles para jugadores de distinto nivel. Depende de la habilidad, todos jugadores pueden alcanzar su nivel adecuado. Gana una y pasa el nivel siguiente.

3. El dibujo está muy animado. Se puede ver el movimiento del personaje como una persona corriendo (las piernas se mueven). Cuando elimina las cartas, se ve una acción como da la vuelta. Se muestra la vida y los puntos conseguidos. Así puede ser un juego de compate entre amigos.

Daniel Gamo:

Opino que nuestro proyecto es muy interesante, no sólo por tratarse de un juego divertido, sino porque está inspirado en el clásico juego japonés TouHou y permite darlo a conocer entre el público. De hecho, muchos de los elementos que se han añadido al

programa (como las imágenes de las cartas de los niveles, el aspecto y movimientos del personaje, los fondos de pantalla e incluso la música) se han tomado casi directamente del juego original. Sin embargo, en nuestra versión se han cambiado cosas respecto al original, principalmente a nivel de jugabilidad (por ejemplo se han eliminado las balas, que en el nuestro no aparecen) para centrarnos en el desarrollo de los elementos principales del juego.

Hussein Hassan:

Nuestro proyecto esta basado en un juego ya existente con cierta complejidad. Mediante el uso de casi la totalidad de los componentes que hemos tenido a mano, y usando todos los métodos practicos aprendidos en clase hemos conseguido hacer una replica merecedora de serlo. En el ámbito técnico se han usado ROMs, RAMs, varios algoritmos mediante maquinas de estado y comunicación síncrona con el teclado de forma adecuada. En el ámbito de consumo, el juego es vistoso y permite un nivel de inmersión bastante aceptable, con pulsación de teclas realista, animaciones y elementos típicos en juegos como vidas, puntos, niveles; todo esto acompañado con sus elementos graficos respectivos para no necesitar un conocimiento amplio sobre el proyecto para poder jugar. A diferencia de resto de proyectos, el nuestro siempre ha sido el más avanzado, el que mas progresos ha mostrado, y hemos sido los primeros en conseguir funcionar los graficos y el sonido. Nuestro proyecto tiene unas características graficas de mayor calidad y belleza, usando la capacidad completa de colores y espacio en las pantallas del laboratorio. En nuestro proyecto se ha intentado simular un juego que originalmente se ejecutaba en un procesador, por lo que hemos tenido que hacer cambios ingeniosos para que las operaciones que requieres mucho hardware no lo hiciesen, sin sacrificar realismo en la experiencia de juego. Por estas razones y otras más considero que nuestro proyecto es el mejor.

6 El sistema PLB es lo peor

Al igual que en el apartado anterior, cada integrante expresa a continuación su opinión sobre el método de la asignatura:

Ángel Alonso:

A diferencia del software, que se presta más a elucubraciones teóricas como diferentes algoritmos y estructuras de datos, el hardware es una materia eminentemente práctica. Por esto mismo una asignatura enfocada al hardware sin aspectos prácticos carece de sentido alguno. Y la mejor manera de poner a trabajar todos los aspectos prácticos para efectivamente cohesionar los conocimientos es un proyecto completo. Pero tampoco se pueden olvidar las clases de teoría, pues sirven como base indispensable para desarrollar estos conocimientos prácticos.

Xi Chen:

En mi opinión, me gusta más esa forma nueva de dar clase. Así aprendemos más cosas prácticas.

Es que, en sentido común, muchas asignaturas de informática nos dan clases teórica y prácticas en laboratorio. Y así piensan que vamos a prender la teoría y la práctica. Pero, para mí es el contrario. A mí no me da tanto tiempo hacer las cosas y prepara las teorías

para el examen. Este año, con TOC, el trabajo me quita el examen y me da más tiempo para otras asignaturas que hay que examinar. Me beneficia bastante.

¿Y cuál forma aprendemos más de TOC? Pienso que tiene sus ventajas cada una. Seguro que las clases tradicionales, aprendemos más de teoría y resolver los problemas. Pero, tampoco tiene muchos sentidos de un informático sabe más resolver problemas que hacer un proyecto en realidad.

Como has dicho, un proyecto hecho es el mejor curriculum de un informático. Y además, la experiencia de hacer un proyecto que funciona, trabajar en grupo y aprender cosas en trabajando es una cosa maravillosa. Voto a hacer una práctica.

Daniel Gamo:

Pienso que el sistema de evaluación por proyecto presenta la ventaja de que permite ahondar en los contenidos de la asignatura de una forma práctica. En este caso se aplican los conocimientos de la asignatura al desarrollo de un juego de nuestra elección (además del controlador de pantalla, de teclado, etc.), haciendo la tarea más entretenida.
Sin embargo, habría sido recomendable según mi criterio aunque sin renunciar al sistema de evaluación por proyecto, dar más clases de teoría, ya que algunos conceptos importantes de la asignatura no se aplican ni se pueden relacionar directamente con lo que hacemos en el proyecto. Además éstos pueden ser interesantes de cara a otras asignaturas de la carrera, y deberían quedar bien claros.

Hussein Hassan:

Considero que el proyecto ha sido algo interesante y que hemos aprendido bastantes cosas realizándolo. Aún así, creo que se habrían conseguido mejores resultados en cuanto al aprendizaje general de la clase si se hubiese planteado como la asignatura de Tecnología de Programación de segundo; es decir, un proyecto común a todos los alumnos, guiado parcialmente pero con cierta libertad de implementación, en parejas o individual, a la vez que se dan clases teóricas normales. De esta forma habriamos obtenido lo mejor de los dos métodos.

7 Tambien he utilizado

El generador de reloj de VGA ha sido extraido del libro de ejemplos en Spartan-3 (FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 de Pong P. Chu) y modificado para arreglar unos fallos.

El LFSR y el algoritmo Double Dabble fueron extraidos de sus respectivas páginas en la Wikipedia inglesa:

http://en.wikipedia.org/wiki/Linear_feedback_shift_register

http://en.wikipedia.org/wiki/Double_dabble

La especificación detallada del protocolo de comunicación PS2 se extrajo de la siguiente página web:

http://www.computer-engineering.org/ps2protocol/

Se uso un programa Java propio para transformar los BMP a codificaciones propias para usarlas en las ROM del proyecto. El código se adjunta en un apéndice.

La melodía se obtuvo del siguiente video de Youtube:

https://www.youtube.com/watch?v=Z7FeCEp7vBs

Para manipular la melodía MIDI se uso la aplicación MidiEditor:

http://midieditor.sourceforge.net/

Los diagramas ASM se han hecho usando el programa yEd:

http://www.yworks.com/en/products/yfiles/yed/

En la fase alpha del módulo de sonido se uso un programa Phython para decodificar las notas a frecuencias para las ROM y un programa en C++ para simular la reproducción de sonido de la FPGA. Se incluye más información en un apéndice.

Para simular la conexión a una pantalla desde el simulador de Xilinx se ha usado la siguiente página:

http://ericeastwood.com/lab/vga-simulator/

Durante la fase beta del módulo de cartas se usaron memorias RAM de doble puerto, extraidas de la siguiente fuente:

http://danstrother.com/2010/09/11/inferring-rams-in-fpgas/

Durante la etapa final del proyecto se trato de usar la memoria SDRAM y la memoria Flash externas a la FPGA, pero esto resulto imposible. Se uso la documentación de ambos dispositivos y ejemplos de uso de la memoria SDRAM.

8 Trabajo en grupo

Aquí se indica el reparto de trabajo. Los datos proporcionados aquí son indicativos, ya que no se ha seguido un orden estricto durante el desarrollo del proyecto.

Chip, puntos y overlay: Hussein Hassan.

Personaje: Xi Chen y Hussein Hassan.

Bola: Xi Chen, Daniel Gamo y Hussein Hassan.

Cartas: Hussein Hassan.

Video, animaciones y sprites: Hussein Hassan.

Musica:

- Investigación del método y primer prototipo en funcionamiento con varias melodias: Ángel Alonso y Daniel Gamo.
- Implementación de la melodía definitiva: Hussein Hassan.

Interfaz y controlador PS2: Hussein Hassan.

Investigación sobre el uso de memoria SDRAM: Xi Chen y Hussein Hassan.

Investigación sobre el uso de memoria Flash: Ángel Alonso y Daniel Gamo.

9 Apendices

9.1 Código Java de codificación de colores

El proyecto Java se adjunta en la parte de material adicional como ColourEncoder.

9.2 Proyecto alpha de prototipo de melodía, con código Python y C++

Se adjuntan en el material adicional con nombre audio_buzzer.

9.3 Código de RAM de doble puerto, junto con antigua iteración del módulo de cartas

Esta antigua versión del modulo de cartas hacia uso de cuatro RAMs de doble puerto para conseguir máxima concurrencia en las colisiones de la bola y las cartas. Como el método nos pareció interesante lo incluimos en el material adicional, con nombres AltRAM.vhd y Cartas.vhd