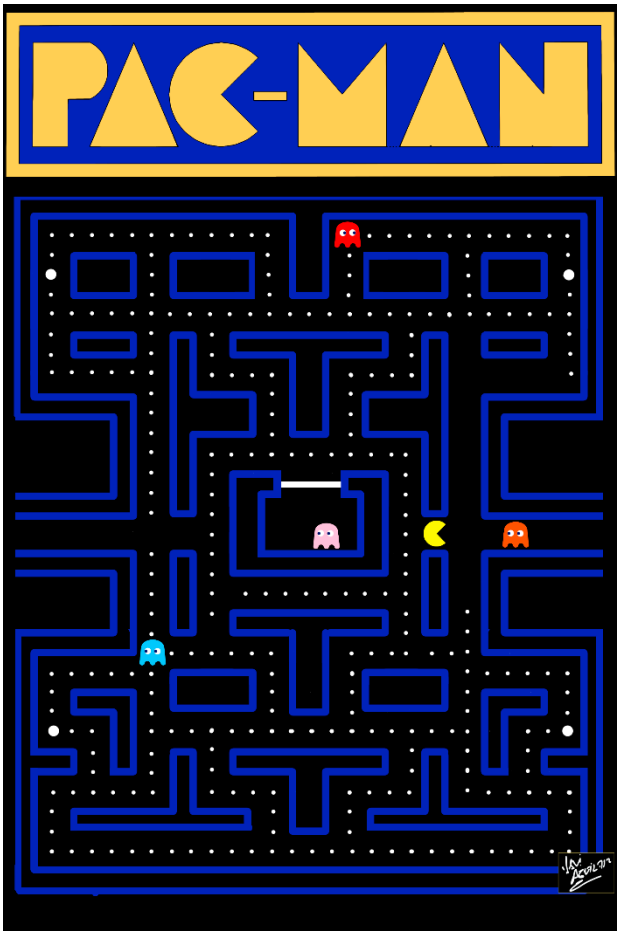


PROYECTO DE CURSO

COMECOCOS

Jaime Garrido Gonzalez | Sistemas Electrónicos | 30/01/2025
Alejandro Roche Aniento
Pedro Rodríguez Mateo

Contenido

1. Introducción 2
2. Design Sources 2
   - 2.1. SUPERIOR 2
     - 2.1.1. Divisor_mov 3
     - 2.1.2. VGA_driver 3
     - 2.1.3. Dibuja 4
     - 2.1.4. Maquina_cmc 5
     - 2.1.5. Maquina_fantasma 7
     - 2.1.6. Maquina_fantasma2 10
     - 2.1.7. Gestion_botones 10
   - 2.2. Acceso_mem 11
3. Coefficient Files 11
   - 3.1. Mapa.coe 11
   - 3.2. Comebolas.coe 12
4. Warnings 12
5. Diagrama de Bloques 14
6. Conclusión 15

1. Introducción

La memoria que se describe a continuación constituye la representación estructural del tablero de un juego inspirado en el clásico Pac-Man. En este diseño, cada celda del tablero tiene una función específica que contribuye al desarrollo del juego. Algunas celdas corresponden a los caminos por los que el personaje principal y los fantasmas pueden desplazarse, otras almacenan los objetos que deben ser recolectados, y otras delimitan las paredes y las áreas de interacción entre los fantasmas y el jugador.

2. Design Sources

2.1. SUPERIOR

- **Descripción General:**

El bloque principal, llamado SUPERIOR, implementa un sistema para controlar la interacción de un videojuego tipo "Comecocos" en un diseño basado en FPGA. Este sistema combina varios módulos, como memoria, drivers VGA, controladores de botones y máquinas de estados, para generar gráficos en pantalla, gestionar el movimiento de los sprites, y actualizar los datos del juego en tiempo real.

El diseño utiliza entradas del usuario (botones de dirección) para mover el personaje principal y coordina las señales de sincronización para renderizar los gráficos en una pantalla mediante una salida VGA.

Este bloque usa las siguientes interconexiones internas que coordinan el funcionamiento de los distintos subbloques del sistema, permitiendo el flujo de datos y la sincronización entre ellos:

- **ENTRADAS:**
  1. clk (STD_LOGIC) - Señal de reloj principal.
  2. up (STD_LOGIC) - Botón de dirección hacia arriba.
  3. down (STD_LOGIC) - Botón de dirección hacia abajo.
  4. left (STD_LOGIC) - Botón de dirección hacia la izquierda.
  5. right (STD_LOGIC) - Botón de dirección hacia la derecha.

- **SALIDAS:**

  1. RED (STD_LOGIC_VECTOR (3 downto 0)) - Canal rojo para la salida VGA.

2. GRN (STD_LOGIC_VECTOR (3 downto 0)) - Canal verde para la salida VGA.
3. BLU (STD_LOGIC_VECTOR (3 downto 0)) - Canal azul para la salida VGA.
4. VS (STD_LOGIC) - Señal de sincronización vertical VGA.
5. HS (STD_LOGIC) - Señal de sincronización horizontal VGA.

2.1.1. Divisor_mov

- **Descripción General**

El módulo divisor_mov es un divisor de frecuencia diseñado para generar una señal de reloj más lenta (clk_pixel) a partir de un reloj de entrada (clk). Este tipo de componente se utiliza para sincronizar movimientos o actualizaciones en sistemas digitales, como la velocidad de refresco o movimiento en un juego (por ejemplo, Comecocos).

- **Entradas y Salidas:**

- **ENTRADAS:**
  1. clk (STD_LOGIC) - Señal de reloj principal utilizada para la sincronización.
  2. reset (STD_LOGIC) - Señal para reiniciar el contador y restablecer el sistema.
  3. refresh (STD_LOGIC) - Controla la actualización del movimiento en el sistema.

- **SALIDAS:**
  1. clk_pixel (STD_LOGIC) - Señal de reloj dividida generada por el contador.

El componente **divisor_mov** divide la frecuencia del reloj de entrada mediante un contador de 16 bits (cont) que incrementa hasta un valor máximo (1111111). Cuando alcanza este valor, genera un pulso (clk_pixel = '1') y lo reinicia.

2.1.2. VGA_driver

- **Descripción General:**

El módulo VGA_driver implementa un controlador para señales VGA (Video Graphics Array), utilizado para manejar la sincronización horizontal y vertical, además de generar las señales de color (RGB) necesarias para la visualización en un monitor VGA.

Este controlador divide el reloj principal para generar un reloj de píxeles (clk_pixel), coordina los contadores horizontales (eje x) y verticales (eje y) para recorrer la pantalla, y genera las señales de sincronización (HS y VS) y las señales de color (RED_out, GRN_out, BLU_out) durante el barrido.

- **Entradas y Salidas:**

- **ENTRADAS:**
  1. clk (STD_LOGIC) - Señal de reloj principal utilizada para sincronizar el controlador VGA.
  2. reset (STD_LOGIC) - Señal para reiniciar el sistema y restablecer todos los contadores.
  3. RGB_in (STD_LOGIC_VECTOR (11 downto 0)) - Señal de entrada con los valores de color (rojo, verde y azul) para generar la señal de salida VGA.

- **SALIDAS:**
  1. **VS** (STD_LOGIC) - Señal de sincronización vertical para la pantalla VGA.
  2. **HS** (STD_LOGIC) - Señal de sincronización horizontal para la pantalla VGA.
  3. **RED_out** (STD_LOGIC_VECTOR (3 downto 0)) - Salida del canal de color rojo, basada en los datos de entrada y las zonas visibles de la pantalla.
  4. **GRN_out** (STD_LOGIC_VECTOR (3 downto 0)) - Salida del canal de color verde, procesada de forma similar a la del color rojo.
  5. **BLU_out** (STD_LOGIC_VECTOR (3 downto 0)) - Salida del canal de color azul, ajustada para las zonas visibles de la pantalla.
  6. **ejex_out** (STD_LOGIC_VECTOR (9 downto 0)) - Coordenada horizontal del píxel actual en la pantalla.
  7. **ejey_out** (STD_LOGIC_VECTOR (9 downto 0)) - Coordenada vertical del píxel actual en la pantalla.
  8. refresh (STD_LOGIC) - Señal que indica el inicio de un nuevo fotograma (refresco de pantalla).

2.1.3. Dibuja

- **Descripción General:**

El módulo dibuja es responsable de generar las señales RGB correspondientes a un área específica de una pantalla de 512x256 píxeles. Este módulo utiliza las coordenadas horizontales y verticales (eje_x y eje_y) junto con datos externos (dout_b y dout) para determinar el color de los píxeles en la pantalla y también calcula direcciones (ADDR y adde_b) para acceder a una memoria. Es importante diferenciar entre los accesos a memoria del tablero y los de la memoria Sprite:

- addb_b y dout_b: Corresponden al puerto B de la memoria del tablero, encargándose de gestionar los datos relacionados con el mapa o layout del juego.

- ADDER y dout: Provienen de la memoria Sprite y están relacionados con los elementos gráficos como los personajes (comecocos y fantasmas) y otros objetos del juego.

- **Entradas y Salidas:**

- **ENTRADAS:**
  1. eje_x (STD_LOGIC_VECTOR (9 downto 0)) - Coordenada horizontal de la pantalla, usada para calcular direcciones y determinar la región activa.
  2. eje_y (STD_LOGIC_VECTOR (9 downto 0)) - Coordenada vertical de la pantalla, también utilizada para cálculos de direcciones y regiones activas.
  3. dout_b (STD_LOGIC_VECTOR (2 downto 0)) - Dato de entrada que se utiliza para decidir cómo manipular la señal RGB y otras direcciones.
  4. dout (STD_LOGIC_VECTOR (11 downto 0)) - Datos de entrada que contienen los valores RGB para un píxel.
  5. posicion (STD_LOGIC_VECTOR (3 downto 0)) - Permite determinar la orientación del dibujo del comecocos. Dependiendo de su valor, el comecocos podrá girar en diferentes direcciones, lo que afecta cómo se despliega visualmente en la pantalla.

- **SALIDAS:**
  1. adde_b (STD_LOGIC_VECTOR (8 downto 0)) - Dirección calculada basada en las posiciones horizontal y vertical (eje_x y eje_y).
  2. ADDR (STD_LOGIC_VECTOR (10 downto 0)) - Dirección completa formada combinando valores de bits, las partes menos significativas de eje_x y eje_y.
  3. RGB (STD_LOGIC_VECTOR (11 downto 0)) - Salida de color en formato RGB (rojo, verde y azul), que depende de la entrada dout_b y las regiones visibles.

- **Explicación del funcionamiento de la memoria Sprite**:

- La memoria Sprite almacena los datos gráficos asociados a los elementos dinámicos del juego, como el comecocos y los fantasmas.
- La señal posición también se usa para seleccionar qué parte de la memoria Sprite se accede, ajustando el dibujo que se genera según la dirección o acción que deba realizar el comecocos. Esto asegura que se muestren correctamente los sprites según el estado actual del juego.

2.1.4. Maquina_cmc

- **Descripción del Código VHDL:**

Este código VHDL implementa una máquina de estados para controlar el movimiento de un "Pac-Man" en un juego, interactuando con su entorno a través de una memoria que contiene diferentes tipos de objetos (como muros, bolas, fantasmas, etc.). La máquina de estados maneja varios estados en los que Pac-Man puede estar (reposo, movimiento, muerte, etc.), y toma decisiones basadas en las señales de entrada y el estado del entorno.

- **Entradas y Salidas:**

- **ENTRADAS:**
  1. clk (STD_LOGIC): Señal de reloj para sincronizar los procesos.
  2. reset (STD_LOGIC): Señal de reinicio para regresar la máquina a su estado inicial (Reposo).
  3. movec (STD_LOGIC): Indica si debe iniciar un movimiento.
  4. udlr (STD_LOGIC_VECTOR (3 DOWNTO 0)): Indica la dirección de movimiento (arriba, abajo, izquierda, derecha).
  5. doutc (STD_LOGIC_VECTOR (2 DOWNTO 0)): Datos que indican el tipo de casilla en la posición actual (vacía, muro, elemento, fantasma, etc.).

- **SALIDAS:**
  1. ADDR (STD_LOGIC_VECTOR (8 DOWNTO 0)): Dirección calculada para acceder a memoria.
  2. donec (STD_LOGIC): Indica si el sistema está en reposo y listo para recibir un nuevo comando.
  3. come_bola (STD_LOGIC): Indica que el personaje ha recogido un elemento.
  4. din (STD_LOGIC_VECTOR (2 DOWNTO 0)): Valor a escribir en memoria.
  5. wec (STD_LOGIC_VECTOR (0 DOWNTO 0)): Señal para habilitar escritura en memoria.
  6. enable_mem (STD_LOGIC): Habilita el acceso a la memoria.
  7. posición (STD_LOGIC_VECTOR (3 DOWNTO 0)): Controla la rotación del sprite del comecocos en el bloque **dibuja**

- **Máquina de Estados:**

La máquina de estados se define a través del tipo SERIE, que tiene los siguientes estados:

1. Reposo: Es el estado inicial. Aquí Pac-Man está esperando una señal para moverse. En este estado, se deshabilitan las escrituras en la memoria, y la señal done se pone en bajo.

2. Comprobar_direccion: Pac-Man evalúa la dirección en la que quiere moverse, a partir de la señal udlr. Se determina cuál será la dirección de la próxima acción, y se consulta la memoria para saber qué hay en esa dirección (por ejemplo, si hay un muro, una bola, etc.).

3. Comprobar_dato: En este estado, el sistema comprueba los datos obtenidos de la memoria (doutc). Dependiendo del valor de doutc, Pac-Man puede encontrar:

- Muro: Si hay un muro en la dirección seleccionada, Pac-Man no puede moverse en esa dirección.
- Bola: Si encuentra una bola, se vacía la casilla y se actualiza la posición de Pac-Man.
- Fantasma: Si Pac-Man encuentra un fantasma, pasa al estado de "muerte".

4. Vaciar_casilla: Si Pac-Man encuentra una bola, en este estado se vacía la casilla en la que Pac-Man está actualmente. Para hacer esto, se activa la señal de escritura we y se escribe un valor vacío (000) en la dirección de la memoria correspondiente.
5. Actualiza_posicion: Una vez que se ha vaciado la casilla, se actualiza la posición de Pac-Man en la memoria. Dependiendo de la dirección a la que se movió, se incrementa o decrementa la coordenada de la posición de Pac-Man (posx y posy).
6. Mover: Pac-Man se mueve a la nueva posición, actualizando las señales de memoria para reflejar su nueva ubicación.
7. Muerte: Si Pac-Man se encuentra con un fantasma, se reinicia su posición a la inicial (en la parte superior izquierda del mapa), y vuelve al estado Reposo.
8. Hay_muro: En este estado, se comprueba si Pac-Man ha encontrado un muro. Si es así, comprueba la siguiente posición a la que se quiere ir.
9. Comprobar_muro: Verifica si en la posición actualizada existe un muro. Esto permite decidir si el movimiento del Comecocos debe mantenerse en línea recta hasta encontrar un espacio libre, momento en el cual puede cambiar de dirección.

2.1.5. Maquina_fantasma

- **Descripción del Código VHDL:**

Este código VHDL implementa una máquina de estados para controlar el movimiento y comportamiento de los **fantasmas** en el juego Pac-Man. La máquina de estados gestiona los diferentes estados en los que un fantasma puede estar (reposo, movimiento, comprobación de dirección, etc.), y toma decisiones basadas en la posición de Pac-Man, la dirección de movimiento, y los datos del entorno.

- **Entradas y Salidas:**

- **ENTRADAS:**

1. clk (STD_LOGIC): Señal de reloj para sincronizar los procesos.
2. reset (STD_LOGIC): Señal de reinicio para regresar la máquina a su estado inicial (Reposo).
3. movef (STD_LOGIC): Indica si el fantasma debe empezar a moverse.
4. doutf (STD_LOGIC_VECTOR (2 DOWNTO 0)): Datos de la memoria que indican el tipo de casilla en la dirección a la que se moverá el fantasma (vacía, muro, Pac-Man, etc.).
5. donef (STD_LOGIC): Señal que indica si el fantasma ha terminado un ciclo de movimiento.

- **SALIDAS:**

1. ADDRf (STD_LOGIC_VECTOR (8 DOWNTO 0)): Dirección calculada para acceder a memoria.
2. dinf (STD_LOGIC_VECTOR (2 DOWNTO 0)): Valor a escribir en la memoria.
3. wef (STD_LOGIC_VECTOR (0 DOWNTO 0)): Señal para habilitar la escritura en memoria.
4. enable_memf (STD_LOGIC): Habilita el acceso a la memoria.
5. donef (STD_LOGIC): Indica que el fantasma ha terminado su movimiento.
    - **Máquina de Estados:**

La máquina de estados se define a través del tipo SERIE, que tiene los siguientes estados:

1. Reposo:
    - Estado inicial en el que el fantasma no se mueve.
    - Espera la señal movef para comenzar su acción.
2. Comprobar_direccion:
    - El fantasma evalúa su dirección de movimiento.
    - Según el valor de direccion_usar_s, el fantasma calcula la dirección hacia la que debe moverse (arriba, abajo, izquierda, derecha).
    - Se calcula la dirección de memoria (ADDRf) basada en la dirección actual y la dirección de movimiento seleccionada.
    - Si el contador (cont) ha alcanzado el valor máximo (Max_cont), el sistema pasa al estado Comprobar_dato.
3. Comprobar_dato:

- El fantasma evalúa la casilla en la que va a moverse, con base en el valor de doutf:
    - Si se encuentra con un muro, el fantasma cambia de dirección.
    - Si se encuentra con Pac-Man, pasa al estado de Reposo.
    - Si la casilla está vacía o contiene otros objetos, el fantasma continúa su movimiento.
    - Si se llega al valor máximo del contador, el sistema avanza al estado Vaciar_casilla.

4. Vaciar_casilla:
    - Si el fantasma se encuentra en una casilla válida (vacía o con un objeto que debe recoger), se vacía la casilla correspondiente de la memoria (wef = '1').
    - El valor de la casilla se guarda en dinf y la dirección de la casilla en ADDRf.
    - Si el contador ha alcanzado el valor máximo (Max_cont), el sistema pasa al estado Actualiza_posicion.
5. Actualiza_posicion:
    - El fantasma actualiza su posición en la memoria según la dirección de movimiento.
    - Dependiendo del valor de last_udlr (dirección), el fantasma ajusta las coordenadas de su posición (posx y posy).
    - Después de actualizar la posición, el sistema pasa al estado Mover.
6. Mover:
    - En este estado, el fantasma mueve su sprite en la nueva dirección.
    - Se actualizan los valores de memoria correspondientes a la nueva posición.
    - La señal donef se activa para indicar que el movimiento se ha completado.
    - Si el contador ha alcanzado el valor máximo (Max_cont), el sistema vuelve al estado Reposo.

2.1.6. Maquina_fantasma2

Este es el mismo tipo de máquina que la del fantasma 1, pero con la diferencia de que parte de una dirección distinta en el tablero. En este caso, la máquina comienza en una posición diferente: la dirección inicial es "00001" para posx y "1001" para posy, mientras que el fantasma 1 comienza en "01111" para posx y "0100" para posy. Sin embargo, el comportamiento general y la lógica de la máquina son prácticamente las mismas.

Es importante especificar que, en la lógica del juego, si el Pac-Man se queda quieto, los fantasmas también permanecerán estáticos porque dependen de la señal done que les indica que el Pac-Man ha terminado de moverse. Mientras done no esté en '1', los fantasmas no recibirán la señal de movimiento, lo que hace que permanezcan en su lugar. Esto asegura que los fantasmas solo se desplacen cuando el Pac-Man se haya movido a una nueva casilla.

2.1.7. Gestion_botones

- **Descripción del Código VHDL:**

Este código VHDL implementa un módulo llamado gestion_botones, que controla los movimientos del Pac-Man en un sistema basado en cuatro direcciones: arriba, abajo, izquierda y derecha. Las entradas son los botones de control, y la salida es un vector de 4 bits que representa las direcciones de movimiento.

- **Entradas y Salidas:**

- **ENTRADAS:**
  - up: Señal de entrada que representa el botón de movimiento hacia arriba.
  - down: Señal de entrada que representa el botón de movimiento hacia abajo.
  - left: Señal de entrada que representa el botón de movimiento hacia la izquierda.
  - right: Señal de entrada que representa el botón de movimiento hacia la derecha.
- **SALIDAS:**
  - udlr: Un vector de 4 bits (de 3 a 0) que indica las direcciones de movimiento. Este vector es enviado a otro módulo que probablemente se encargue de mover el objeto en un sistema visual (como el tablero de juego). Los bits corresponden a las señales de dirección:

- udlr(3) = arriba
- udlr(2) = abajo
- udlr(1) = izquierda
- udlr(0) = derecha

2.2. Acceso_mem

- **Funcionalidad del Módulo:**

1. Condición Principal:
   - Si la señal enable_mem está en 1, las señales de la máquina de estados del comecocos (addr, din, we) se redirigen directamente a la memoria (addra, din_a, wea) y la memoria es habilitada (ena = 1).
   - Además, los datos leídos desde la memoria (douta) se asignan como salida (dout), permitiendo a la máquina del comecocos interactuar con la memoria.
2. Estado Predeterminado:
   - Si enable_mem está en 0, las señales predeterminadas (valores por defecto) se asignan a las salidas

- **Relación con la Máquina de Estados del Fantasma:**

1. Cuando enable_mem está en 0, las señales de la máquina de estados del fantasma se conectan a la memoria, reemplazando las asignaciones predeterminadas.

2. Coefficient Files

3.1. Mapa.coe

1. Valores en el Tablero:
   - 0: Representa una casilla vacía.
   - 1: Representa una casilla con un muro o pared.
   - 2: Es una casilla con una bola o un punto.
   - 3: Representa un fantasma
   - 4: Representa el comecocos

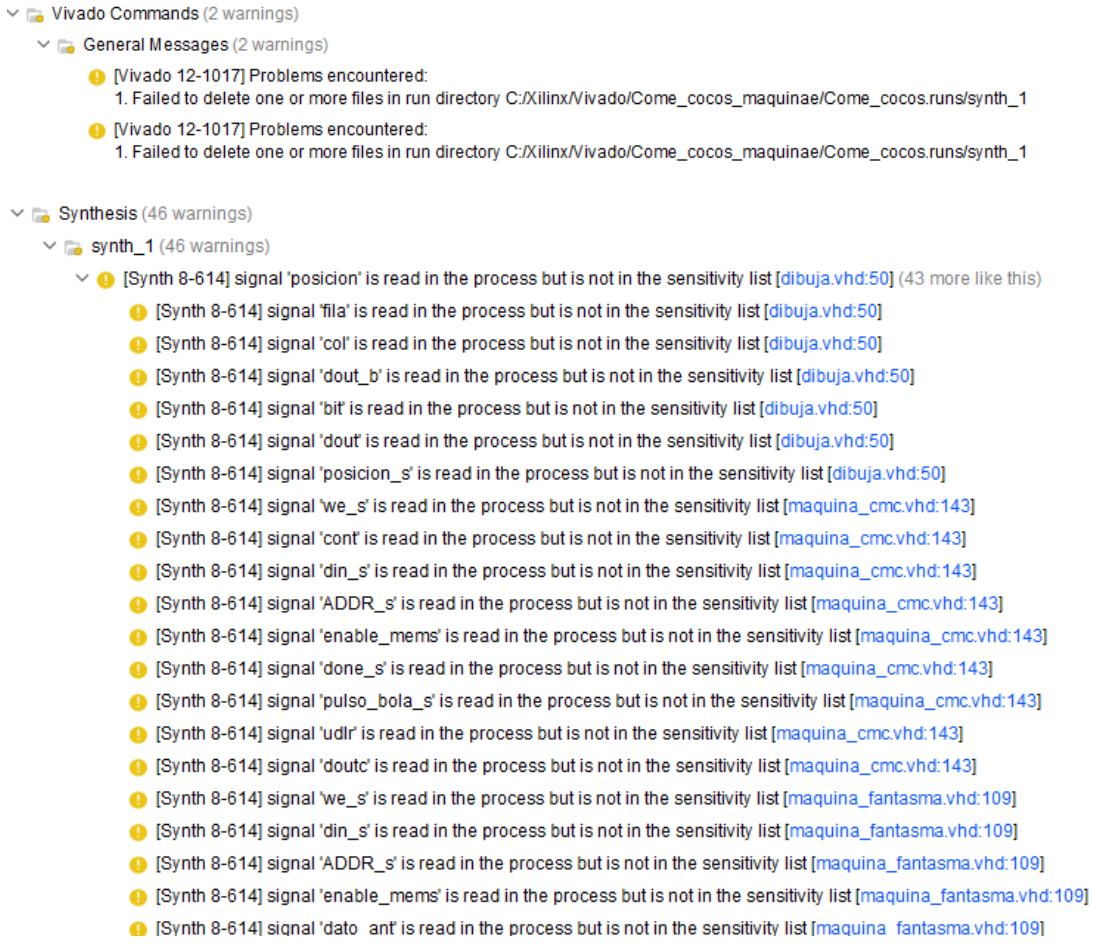
- 5: Representa el fantasma 2

3.2. Comebolas.coe

- **Descripción general del código:**

Este bloque de memoria contiene los “sprites” para representar los elementos del tablero.

4. Warnings

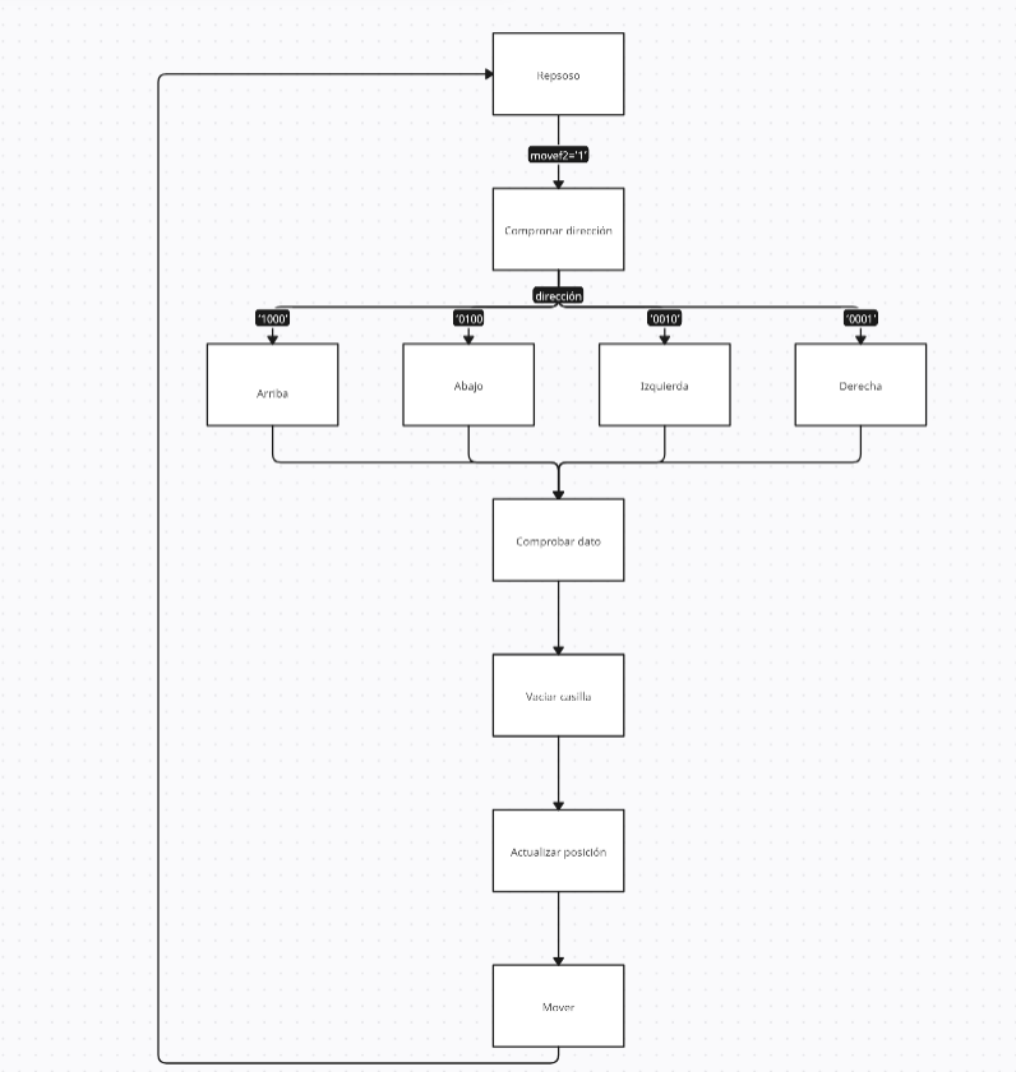


[Synth 8-614] signal 'dato_ant' is read in the process but is not in the sensitivity list [maquina_fantasma.vhd:109]
[Synth 8-614] signal 'dato_prox' is read in the process but is not in the sensitivity list [maquina_fantasma.vhd:109]
[Synth 8-614] signal 'donef_s' is read in the process but is not in the sensitivity list [maquina_fantasma.vhd:109]
[Synth 8-614] signal 'doutf' is read in the process but is not in the sensitivity list [maquina_fantasma.vhd:109]
[Synth 8-614] signal 'we_s' is read in the process but is not in the sensitivity list [maquina_fantasma_2.vhd:109]
[Synth 8-614] signal 'din_s' is read in the process but is not in the sensitivity list [maquina_fantasma_2.vhd:109]
[Synth 8-614] signal 'ADDR_s' is read in the process but is not in the sensitivity list [maquina_fantasma_2.vhd:109]
[Synth 8-614] signal 'enable_mems' is read in the process but is not in the sensitivity list [maquina_fantasma_2.vhd:109]
[Synth 8-614] signal 'dato_ant' is read in the process but is not in the sensitivity list [maquina_fantasma_2.vhd:109]
[Synth 8-614] signal 'dato_prox' is read in the process but is not in the sensitivity list [maquina_fantasma_2.vhd:109]
[Synth 8-614] signal 'donef_s' is read in the process but is not in the sensitivity list [maquina_fantasma_2.vhd:109]
[Synth 8-614] signal 'doutf2' is read in the process but is not in the sensitivity list [maquina_fantasma_2.vhd:109]
[Synth 8-614] signal 'posy' is read in the process but is not in the sensitivity list [gestion_botones.vhd:48]
[Synth 8-614] signal 'posx' is read in the process but is not in the sensitivity list [gestion_botones.vhd:48]
[Synth 8-614] signal 'addrc' is read in the process but is not in the sensitivity list [acceso_mem.vhd:83]
[Synth 8-614] signal 'din_c' is read in the process but is not in the sensitivity list [acceso_mem.vhd:83]
[Synth 8-614] signal 'wec' is read in the process but is not in the sensitivity list [acceso_mem.vhd:83]
[Synth 8-614] signal 'dout_s' is read in the process but is not in the sensitivity list [acceso_mem.vhd:83]
[Synth 8-614] signal 'addrf' is read in the process but is not in the sensitivity list [acceso_mem.vhd:83]
[Synth 8-614] signal 'dinf' is read in the process but is not in the sensitivity list [acceso_mem.vhd:83]
[Synth 8-614] signal 'wef' is read in the process but is not in the sensitivity list [acceso_mem.vhd:83]
[Synth 8-614] signal 'addrf2' is read in the process but is not in the sensitivity list [acceso_mem.vhd:83]
[Synth 8-614] signal 'dinf2' is read in the process but is not in the sensitivity list [acceso_mem.vhd:83]
[Synth 8-614] signal 'wef2' is read in the process but is not in the sensitivity list [acceso_mem.vhd:83]

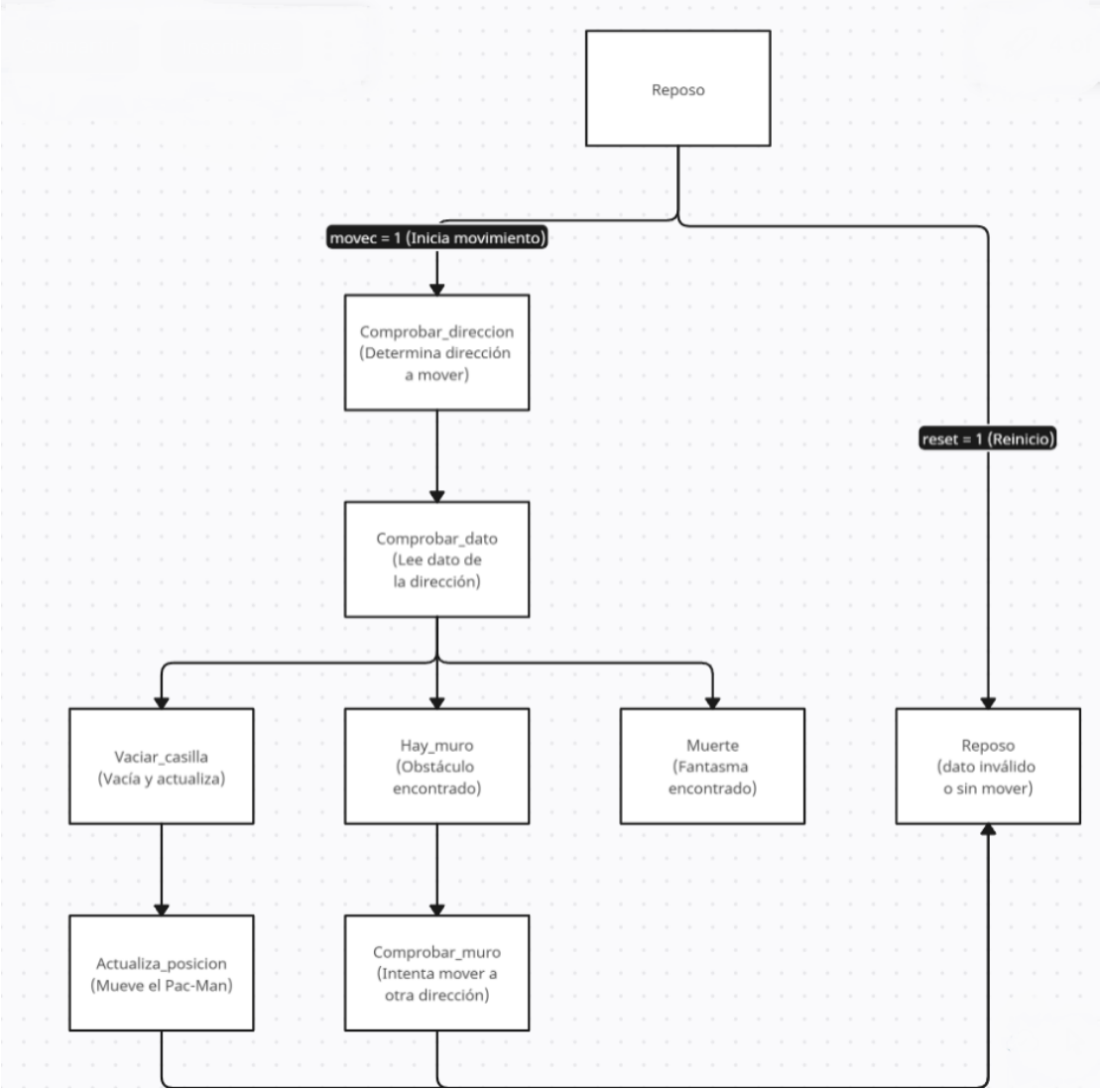
[Timing 38-316] Clock period '20.000' specified during out-of-context synthesis of instance 'mem_sprite' at clock pin 'clka' is different from the actual clock period '10.000', this can lead to different synthesis results. (1 more like this)
[Timing 38-316] Clock period '20.000' specified during out-of-context synthesis of instance 'memoria' at clock pin 'clka' is different from the actual clock period '10.000', this can lead to different synthesis results.

5. Diagrama de Bloques

- Bloque máquina de los fantasmas:



- Bloque máquina del comecocos:



6. Conclusión

El bloque de código en cuestión representa la estructura de la memoria utilizada para crear el tablero del juego del *Comecocos*. A través de la inicialización y distribución de valores específicos, se define el entorno del juego, permitiendo la interacción entre Pac-Man, las bolas, los fantasmas y las paredes. Cada celda del tablero está cuidadosamente diseñada para garantizar que el juego funcione de manera fluida y que el jugador pueda experimentar una experiencia de juego coherente. Este enfoque modular de asignar valores a las diferentes partes del tablero asegura que tanto el jugador como los elementos del juego puedan moverse y actuar de manera lógica.