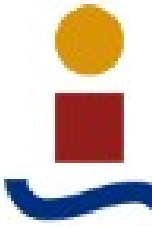
PROYECTO DE SISTEMAS ELECTRÓNICOS

(diseño con VHDL)



Escuela Técnica Superior de
INGENIERÍA DE SEVILLA

Víctor Borrero López

Guillermo Cartes Domínguez

David Romero Pastor

ÍNDICE

1) Presentación:

El proyecto elegido consiste en el diseño, mediante lenguaje VHDL, de un programa que simule un juego similar al conocido "FlappyBird", juego en dos dimensiones en el que un pájaro (jugador) debe conseguir esquivar el máximo número de columnas evitando chocar con ellas, el suelo o el techo; para ello sólo se dispone de un botón, que hará que el pájaro se eleve cierta altura por cada pulsación; el resto del tiempo el pájaro estará cayendo. Este juego servirá como guía y base del proyecto a realizar, introduciéndose varias modificaciones y variaciones que se explicarán a continuación.

Como herramienta se utilizará el software "Xilinx ISE", que proveerá, además de un entorno adaptado a la creación del programa en VHDL, la posibilidad de sintetizar dicho programa para poder ser ejecutado por una FPGA junto con los dispositivos periféricos necesarios.

2) Descripción:

Como ya se ha comentado, tanto el entorno como el modo de juego serán bastante similares al "FlappyBird" original. A continuación, se describen detalladamente sus características en términos del resultado final observable por el usuario.

- ***Pantalla de inicio:*** la pantalla de inicio será lo primero que aparezca al cargar el programa en la FPGA y conectar ésta a un monitor externo. En ella aparecen en una posición estática:
    - Un pequeño alien dentro de un ovni en la parte izquierda de la pantalla, que será el avatar que controlará el jugador.
    - Tres columnas dobles (sale un tramo desde arriba y otro desde abajo dejando un espacio en medio) a lo largo del resto de la pantalla hacia la derecha, que serán los obstáculos a esquivar por el jugador.

  El programa se mantendrá en este estado hasta que se pulse el botón que controla la ascensión del avatar y que será la única entrada externa al programa, además del reset.

- ***Pantalla de juego:*** una vez pulsado por primera vez el botón pasaremos a la pantalla de juego, en la que se desarrolla, como su propio nombre indica, el juego en sí.

  En este momento las columnas empezarán a desplazarse hacia la izquierda con velocidad constante hasta que desaparezcan completamente por el extremo izquierdo de la pantalla, momento en el cuál serán sustituidas por otra columna que aparecerá progresivamente por el borde derecho de la pantalla. El hueco existente entre el tramo de columna superior y el tramo inferior es constante y suficiente para que el avatar pase por él sin chocar con el borde de los tramos. Cada columna nueva tendrá este hueco a una altura diferente, añadiendo así más dificultad al juego.

  Por otra parte, el avatar se moverá de forma parecida a como lo haría en presencia de gravedad. Cada vez que pulsemos el botón ascenderá ligeramente (apareciendo una llama de propulsión bajo la nave), y conforme ascienda, irá perdiendo velocidad hasta alcanzar velocidad cero, momento en el que comenzará a descender (desaparece la llama de propulsión). De la misma forma, al bajar, irá ganando velocidad hasta un valor límite que se le ha impuesto para facilitar la jugabilidad. Para hacer efectiva la pulsación del botón

debemos pulsarlo y liberarlo antes de la siguiente pulsación, además sólo será válida si el avatar está descendiendo en ese momento.

El juego se mantendrá en esta pantalla, generando columnas y sintetizando el movimiento del avatar, hasta que éste choque con una columna, el suelo o el techo.

- ***Pantalla de fin de juego:*** cuando el avatar choque contra un obstáculo, el programa se dirigirá automáticamente hacia la pantalla de fin de juego o partida.

  En este instante la posición de las columnas se congelará y el avatar comenzará a caer hasta desaparecer de la pantalla.

El programa se mantendrá en este estado hasta que se pulse de nuevo el botón, momento en el que volverá a la pantalla de inicio y se ejecutará de nuevo todo el proceso.

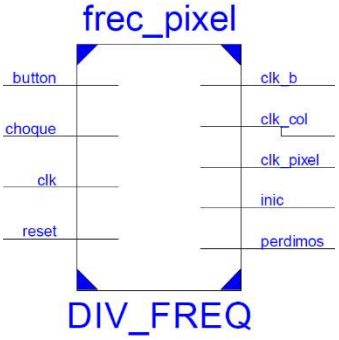
3) Desarrollo:

Para el desarrollo del juego se ha decidido dividir el código que lo genera en distintos bloques que veremos detenidamente más adelante. El bloque 'VGA_driver.vhd' será el que cumple la función de "main" o bloque principal, y que interconecta los demás entre sí y con las entadas y salidas de la FPGA.

A continuación, detallaremos cada uno de ellos explicando las distintas partes o funciones que se llevan a cabo en cada bloque.

❖ Nota: el proceso síncrono 'sinc' sólo se explicará una vez debido a que aparece prácticamente en todos los procesos y siempre cumple la misma función.

3.1 'frec_pixel.vhd'



La función de este bloque será proporcionarnos señales de habilitación a las columnas, al marciano y a los píxeles a una frecuencia determinada. Además, crearemos una máquina de estados para controlar las distintas fases del juego (inicio, en juego y derrota)

Este bloque consta de 4 entradas y 5 salidas:

- *Clk:* señal de entrada que hará la función de reloj.
- *Reset:* señal de entrada que hará la función de reset asíncrono. Activo a nivel alto.
- *Button:* señal de entrada que nos indica cuando se ha pulsado el botón (entrada externa de nuestro sistema)
- *Choque:* señal de entrada que nos indica cuando nuestro alien se ha chocado con alguna columna o los límites de la pantalla.
- *clk_pixel:* señal de salida activa cada 25MHz.
- *clk_b:* señal de salida activa cada 77Hz.
- *clk_col:* señal de salida activa cada 250Hz.

- *perdimos:* señal de salida activa a nivel alto cuando nos encontramos en el tercero de los estados que indicaremos en el próximo apartado.
- *inic:* señal de salida que se activa a nivel alto cuando nos encontramos en la posición de salida, es decir, el primero de los 3 estados que ahora explicaremos.

La señal clk_pixel se hará dividiendo la frecuencia de reloj entre dos. Esto simplemente sería cambiar el valor de clk_pixel cada flanco de subida de la señal de reloj. Las señales clk_b y clk_col se controlarán con un contador interno, que cuando llegue a un valor determinado active su señal correspondiente y luego se resetee. Para conseguir las frecuencias mencionadas antes, el valor de cada contador será de 650.000 para clk_bird y de 200.000 para clk_col.

El resto de señales se controlarán mediante una máquina de estados. Esta consta de 3 estados:

1. Parado: nos encontraremos en este estado cuando aún no hemos empezado a jugar. Por tanto, tendremos que encontrarnos en la posición inicial. La señal inic se pondrá a 1 y la señal perdimos a 0. Cuando se pulse el botón (entrada externa), pasaremos al estado jugamos e iniciaremos el juego.
2. Jugamos: nos encontraremos en este estado mientras estemos jugando. Pondremos a 0 tanto inic como perdimos. Pasaremos al siguiente estado cuando se active la señal choque, esto es, cuando hayamos chocado con alguna columna o los límites de la pantalla.
3. Chocamos: nos encontraremos en este estado cuando hayamos perdido. Por tanto, tendremos la señal perdimos a 1 y la señal inic a 0. Cuando pulsemos dos veces el botón, pasaremos al estado parado e iniciaremos otro ciclo.

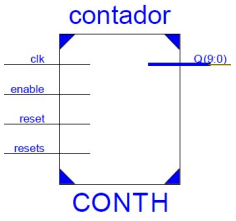
3.2 'Contador.vhd'

La función de este bloque será indicarnos en qué pixel nos encontramos. Como su nombre indica, es un contador, por lo que cada vez que se active el enable incrementará el valor de la salida. Por el contrario, si se activa el reset, todos los bits de nuestra salida se pondrán a 0.

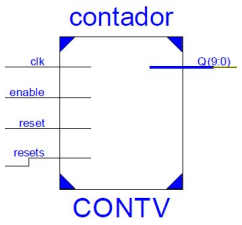
Este bloque consta de 4 entradas y una salida:

- *Clk:* señal de entrada que hará la función de reloj.
- *Reset:* señal de entrada que hará la función de reset asíncrono. Activo a nivel alto.
- *Resets:* señal de entrada que hará la función de reset síncrono. Activo a nivel alto.
- *Enable:* señal de entrada que hará la función de enable del contador. Activo a nivel alto.
- *Q:* señal de salida que será la que indique el valor de la cuenta y, por tanto, en qué pixel estamos. Tendrá N bits, siendo N un generic.

CONTH: En este caso tenemos un contador horizontal, en el que al llegar al borde derecho de la pantalla, nuestra cuenta se reseteará (reset síncrono). Las entradas clk y reset serán las que tenemos para todos los bloques. El enable será nuestra frecuencia de pixel, mientras que resets será la salida O3 del Comph, es decir, la señal que indica que hemos llegado al final horizontal de nuestra pantalla.



CONTV: En este caso tenemos un contador vertical, en el que al llegar al final de la pantalla, nuestra cuenta se reseteará (reset síncrono). Las entradas clk y reset serán las que tenemos para todos los bloques. El enable será una puerta and con nuestra frecuencia de pixel y la salida O3 del Comph, mientras que resets será la salida O3 del Compv, es decir, la señal que indica que hemos llegado al final de nuestra pantalla.

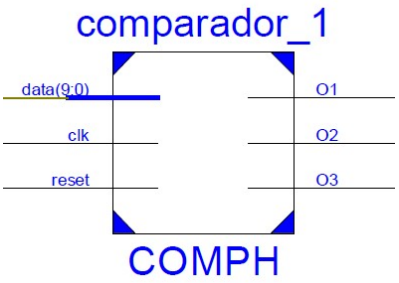


3.3 'Comparador.vhd'

Usaremos un bloque genérico llamado 'comparador', mediante el cual, cambiando los valores genéricos de fin de pantalla, etc., tendremos el comparador del eje horizontal y el del vertical.

- **Comparador horizontal** (COMPH)

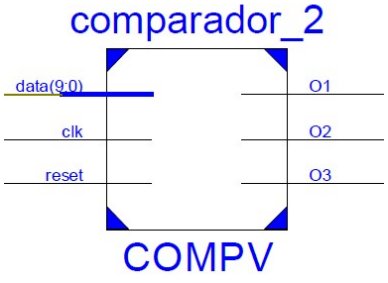
En primer lugar, las entradas clk y reset serán las que tenemos para todos los bloques. Al comparador horizontal le llega un 'data' proveniente del contador del eje horizontal, por lo que ese data sería la lectura del eje x. En función del valor que tome dicha 'data', las salidas irán variando y afectando en diferentes bloques. A continuación, observamos qué función cumple cada salida:



- **O1**: esta salida implicará el final de pantalla horizontal. Por lo tanto, irá dirigida a la entrada 'Blank_H' del generador de color.
- **O2**: esta señal será directamente la salida 'HS' del sistema completo ('VGA_driver'), que es la señal de sincronismo horizontal.
- **O3**: el final de línea horizontal iría directamente a activar el reset síncrono del contador horizontal, para que se reinicie una vez se haya completado la línea horizontal al completo. Además, iría dirigido también al enable del contador horizontal, de manera que una vez leída una línea horizontal, se aumente el contador vertical en una unidad.

- **Comparador vertical** (COMPH)

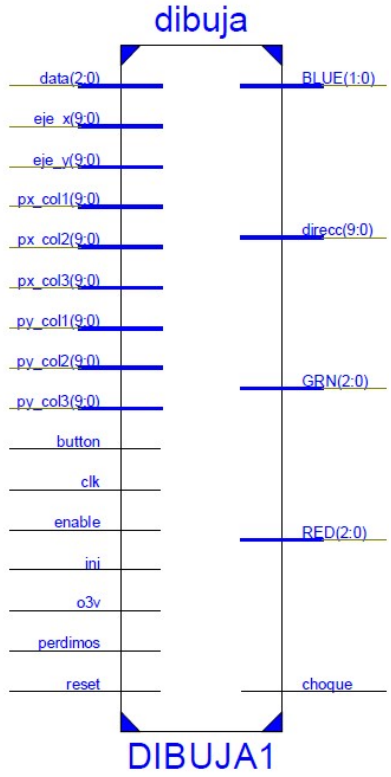
Análogamente al comparador horizontal, este comparador recibe un 'data' que la lectura del eje y, saliente del contador del eje vertical. A continuación, observamos qué función cumple cada salida del comparador:



- **O1**: esta salida implicará el final de pantalla vertical. Por lo tanto, irá dirigida a la entrada 'Blank_V' del generador de color.
- **O2**: esta señal será directamente la salida 'VS' del sistema completo ('VGA_driver'), que es la señal de sincronismo vertical.
- **O3**: el final de línea vertical iría directamente a activar el reset síncrono del contador vertical, para que se reinicie una vez se haya completado la línea vertical al completo.

3.4 ‘Dibuja.vhd’

Antes de describir el funcionamiento de este bloque, debemos conocer sus entradas y salidas. Consta de 16 entradas y 5 salidas:



- *Clk:* señal de entrada que hará la función de reloj.
- *Reset:* señal de entrada que hará la función de reset asíncrono. Activo a nivel alto.
- *eje_x:* entrada que nos indica la coordenada x del píxel.
- *eje_y:* entrada que nos indica la coordenada y del píxel.
- *Data:* señal de entrada correspondiente a la memoria ROM.
- *Px_col1:* entrada que indica la próxima posición horizontal de la columna 1. Tendremos px_col2 y px_col3, que cumplirán la misma función para las columnas 2 y 3, respectivamente.
- *Py_col1:* entrada que indica la próxima posición vertical de la columna 1. Tendremos py_col2 y py_col3, que cumplirán la misma función para las columnas 2 y 3, respectivamente.
- *Button:* señal de entrada que nos indica cuando se ha pulsado el botón (entrada externa de nuestro sistema)
- *Enable:* entrada activa a nivel alto procedente de clk_b. Será la que permita la actualización de la posición del alien.
- *Ini:* señal de entrada que se activa (a nivel alto) cuando nos encontramos en la posición de salida (esperando que empiece el juego). Por tanto, cuando esté activa, todas las variables se encontrarán en la posición inicial y cuando se desactive, podrán actualizarse.
- *O3v:* señal de entrada que se activa cuando se ha terminado de dibujar una pantalla completa.
- *Perdimos:* entrada que indica que el alien ha muerto y por tanto se acaba el juego.
- *BLUE:* salida de 2 bits correspondiente al color azul.
- *GRN:* salida de 3 bits correspondiente al color verde.
- *RED:* salida de 3 bits correspondiente al color rojo.
- *choque:* señal de salida activa a nivel alto cuando el alien ha chocado con alguna columna o los límites de la pantalla.
- *Direcc:* salida de 10 bits que incrementará uno su valor cada vez que leamos un dato de la ROM para así realizar correctamente la transmisión de datos.

Este bloque es sobre el que recaen más funciones dentro del programa global, su código se divide en tres partes: ‘dibuja’, ‘maquina_de_estados’ y ‘pulso_boton’.

- ***Process ‘dibuja’:*** en este proceso se lleva a cabo la jerarquía de imágenes y colores que deben ir apareciendo en la pantalla, mandando al bloque ‘gen_color’ el color que debe aparecer en cada píxel de la pantalla en cada momento como una combinación de 8 bits (3 para el rojo, 3 para el verde y 2 para el azul).

La prioridad más baja la tiene el fondo que, por tanto, se pintará siempre que no haya ningún otro elemento que quiera pintar en el mismo píxel.

Tras el fondo vendría el avatar que, en realidad no solapará con las columnas a menos que choque justo encima de una de ellas, único caso en el que caería por detrás y no se vería.

Y por último vendrían las columnas, que tienen la máxima prioridad.

- ***Process 'maquina_de_estados':*** en este proceso se controla el movimiento del avatar, así como la condición de choque.

  Para el movimiento del avatar se ha creado una máquina de estados que consta de tres estados: 'reposo', 'actualizar_posY' y 'actualizar_Vy'.

  En 'reposo' simplemente se espera hasta que la señal que viene del reloj del avatar se ponga a '1' y se pasa al siguiente estado.

  En 'actualizar_posY' se varía la componente vertical de la posición del avatar dependiendo de la velocidad en ese instante, definiendo la nueva posición como la antigua más la velocidad si el avatar está descendiendo o la antigua menos la velocidad si está ascendiendo. Además, se activa la condición de choque si el avatar supera los límites establecidos de la pantalla por arriba o por abajo.

  En 'actualizar_Vy' se actualiza y se satura la velocidad en función del sentido de movimiento del avatar. Si está descendiendo se le suma cada vez una constante que hace las veces de gravedad, saturándose cuando llega al valor 18. Si está ascendiendo se le va restando esta misma constante.

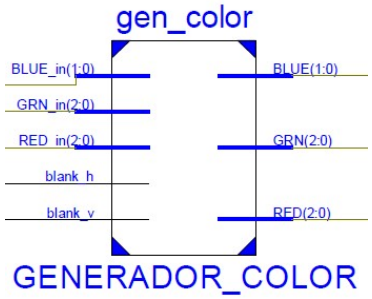
  Dentro del mismo proceso nos encontramos un fragmento de código que se ocupa de poner la variable que controla el sentido del movimiento a '1' cuando recibe el pulso del botón, además de fijar la velocidad de ascensión inicial.

  También nos encontramos otro fragmento que rige la condición de choque con una columna. Cuando la posición del avatar coincide con la de una columna se activa.

- ***Process 'pulso_boton':*** en este proceso se lleva a cabo la máquina de estados que controla el pulso del botón. Los estados son: 'esperopulso', 'pulso' y 'finpulso'.
  En 'esperopulso' se espera a que el botón se active y se pasa al siguiente estado.
  En 'pulso' se pone a '1' una variable que estará activa un ciclo de reloj y que hará la función de pulso del botón.
  En 'finpulso' se vuelve a poner la variable del pulso a '0' (llevándose a cabo la función comentada anteriormente).

3.5 ‘Generador_color.vhd’

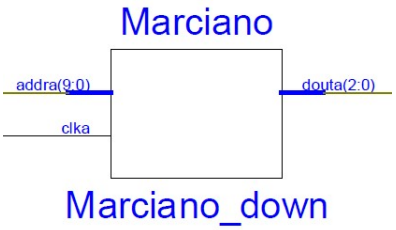
El generador de color toma como entradas las salidas del código ‘Dibuja’, que serían los tres colores RED, GRN y BLUE, y los saca como salidas siempre y cuando no se pongan a uno las señales ‘Blank_H’ o ‘Blank_V’., que indican el fin de pantalla, donde las señales de colores se pondrían todas a cero.



Como podemos comprobar, no existe proceso síncrono en dicho código debido a que funciona realmente como un filtro de las señales de los colores salientes del código ‘Dibuja’.

3.6 ‘Marciano_down’

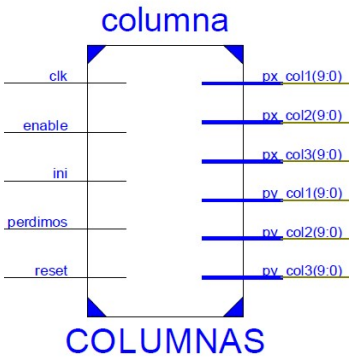
Consiste en la memoria ROM de los colores asignados a nuestro avatar. De esta manera, la ROM lee los valores de un archivo coe previamente creado, donde cada línea será el valor del color de cada píxel. El proceso de lectura de los datos de la memoria se realizará en el código ‘Dibuja’.



Las propiedades de esta memoria serían 4 bits de anchura de datos y 16 bits de longitud de la memoria.

3.7 ‘Columnas.vhd’

En primer lugar, trabajaremos con tres columnas que irán desapareciendo a la izquierda de la pantalla y apareciendo en el extremo derecho, de manera que cree el efecto visual de columnas apareciendo constantemente.



Como entradas tendríamos las ya conocidas clk y reset. Además, tendremos un enable que se activa a nivel alto cada 250Hz (clk_col del bloque freq_pixel) y permite la actualización de las señales a una frecuencia de interés; la variable ‘perdimos’ (que como su nombre indica, significará haber perdido en el juego); y, por último, la variable ‘ini’, que marca el inicio del juego. Como salidas, enviaremos al programa ‘dibuja’ las posiciones, tanto en el eje x como en el eje y, de cada columna.

Definimos las posiciones iniciales de las tres columnas en el eje x, que veremos al iniciar el juego y cada vez que se resetee. Creamos unas señales con las que podremos trabajar y que asignaremos a las salidas correspondientes (posiciones de las columnas).

En el proceso síncrono simplemente actualizaremos las variables cuando se produzca un flanco de subida de la señal de reloj, y en caso de que se resetee el sistema, establecer los valores iniciales para las variables.

A continuación, en el proceso '**movimiento'** se produce la evolución de las posiciones de las columnas en ambos ejes:

*Nota: La señal **enable** es una señal creada en el divisor de frecuencia llamada **enable_columna**, de manera que podemos modificar la velocidad a la que éstas se mueven. Además, la variable **aux** nos permitirá actualizar la posición de las columnas en el flanco de subida de este **enable_columna**.

- En la primera condición, cuando la variable 'ini' se ponga a '1', el juego se iniciará y por lo tanto las variables tomarán sus valores iniciales.
- En caso de que 'enable' esté a '1', 'aux' a '0' y perdimos a '0' (no hayamos perdido aún y sea la primera vez que 'enable' se pone a '1'), entonces actualizamos la posición en el eje x de las tres columnas. Además, el próximo valor de 'aux' será '0'.
- Cuando 'enable' toma valor '0', el próximo valor de 'aux' es '0'.
- Cuando la columna 1 llega a la posición 0 en el eje x, su próxima posición sería el extremo derecho de la pantalla. Para que el hueco por el que el avatar debe de pasar no se encuentre siempre en la misma posición, le sumaremos un valor a la próxima posición, y en caso de que el valor obtenido sea demasiado extremo, 'saturamos' dicha posición y establecemos una que permita al avatar superar el obstáculo. Este proceso se repetiría para las tres columnas.

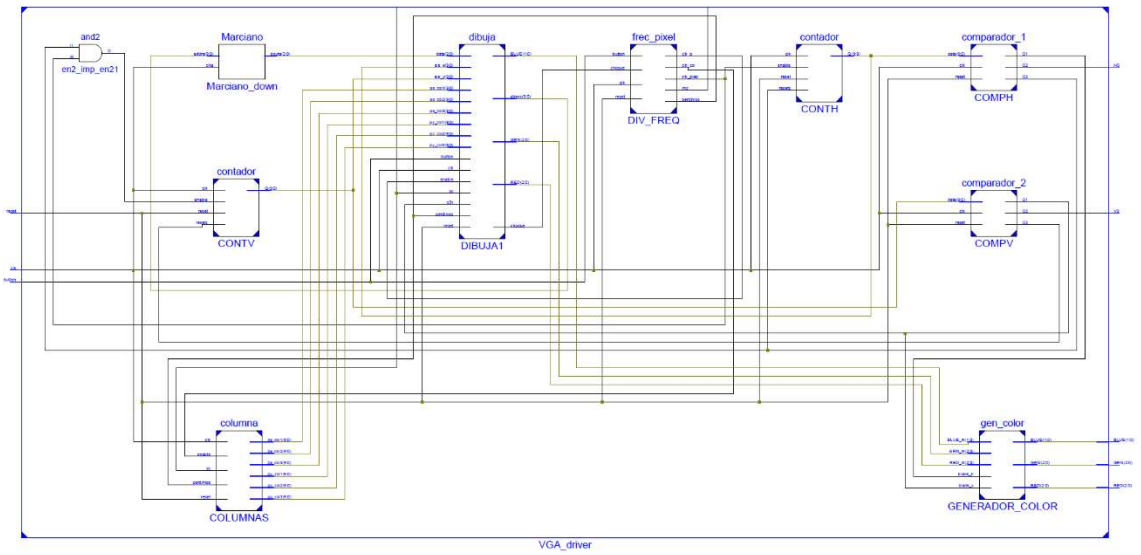
3.8 'VGA_diver.vhd'

En este bloque, el más importante, interconectamos todos nuestros pequeños bloques entre sí. Además, instanciaremos cada uno de ellos con los valores concretos de nuestro programa. Para ello, haremos uso de muchas señales creadas con el fin de facilitar dichas conexiones. Serán las siguientes (también comentadas en el código en vhd):

- **en1**: señal que contendrá la salida 'clk_pixel' del divisor de frecuencia.
- **en2** es la función AND de la salida 'clk_pixel' (en1) y la salida O3 del comparador horizontal.
- **o1h**: señal que toma la salida 1 del comparador horizontal.
- **o3h**: señal que toma la salida 3 del comparador horizontal.
- **o1v**: señal que toma la salida 1 del comparador vertical.
- **o3v**: señal que toma la salida 3 del comparador vertical.
- **choque**: señal creada para cuando se produzca un choque del avatar contra una columna o techo/suelo.
- **para_juego**: señal que indica la congelación de la pantalla y la caída del avatar.
- **Ini:** señal que marca el inicio del juego.
- **q1**: señal que será la salida 'data' del contador horizontal.
- **q2**: señal que será la salida 'data' del contador vertical.
- **red1**: señal que toma los valores a la salida del RED del 'dibuja'.
- **grn1**: señal que toma los valores a la salida del RED del 'dibuja'.
- **data**: señal que toma los valores de la salida 'douta' de la memoria ROM del Marciano, que posteriormente se dirigiría al 'dibuja', para poder pintar correctamente cada píxel.

- **blue1**: señal que toma los valores a la salida del BLUE del 'dibuja'.
- **enable_pajaro**: señal que toma los valores de la salida 'clk_b' del divisor de frecuencia. Será el reloj del avatar.
- **enable_columna**: señal que toma los valores de la salida 'clk_col' del divisor de frecuencia. Será el reloj de la columna.
- **direcc**: señal que toma los valores de la salida 'addra' de la memoria ROM del Marciano, que posteriormente se dirigirá al 'dibuja', para ir leyendo una posición por ciclo de reloj.
- El resto, px_col1, etc., son las posiciones en los diferentes ejes de las columnas, que serán la salida del código 'Columnas'.

El ***diagrama de bloques*** correspondiente será el siguiente:



4) Lista de Warnings:

Sólo tenemos uno, que es el siguiente:

WARNING:Xst:2211 - "C:/Users/guillermo/Desktop/GIERM/Sistemas electronicos/monitorVGA_version12_mejora/VGA_driver.vhd" line 181: Instantiating black box module <Marciano>.

Esto es debido a que, a efectos prácticos, la memoria ROM se comporta como una caja negra, es decir, que el sistema puede modificar su entrada y leer su salida, pero no sabe que está pasando dentro y no tiene control sobre ello. No es importante ya que esto no influye en el correcto funcionamiento de nuestro proyecto.