UNSAM, ECYT

Electrónica Digital I: Trabajo práctico Final

Voltímetro con salida VGA

Alumno: Martin Penizzotto

24/01/2023

Profesores:

- Miguel Ángel Sagreras
- Nicolás Álvarez

1. Objetivo

El objetivo del presente trabajo consiste en construir un voltímetro con salida VGA embebido en una FPGA. El lenguaje descriptor de hardware a utilizar es VHDL, y la metodología del trabajo se basa en la descripción total de hardware por parte del usuario, sin utilizar código secuencial tal que permita al compilador diseñar hardware de manera automática y sin conocimiento del usuario, salvo excepciones.

2. Introducción

El voltímetro a diseñar comienza con un conversor analógico digital del tipo sigma-delta, el cual está formado por un flip flop tipo D, resistencias y un capacitor. El mismo se conecta al bloque de procesamiento de datos y control, que es el que diseñará el alumno. La figura 1 ilustra estos bloques:

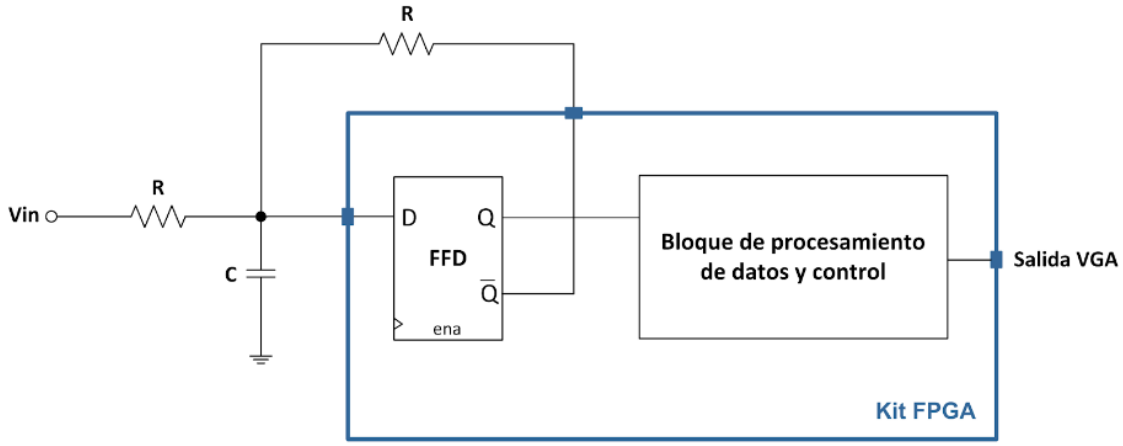

Figura 1: Diagrama en bloques del sistema a implementar.

El resultado de este conversor es que obtendremos a la salida una secuencia de 1's y 0s, donde si tomamos una ventana de tiempo fija, la proporción de 1's obtenidos dentro de esa ventana será proporcional a la tensión real medida.

El diagrama en bloques del voltímetro (bloque de procesamiento de datos y control) propuesto por la cátedra es el siguiente:

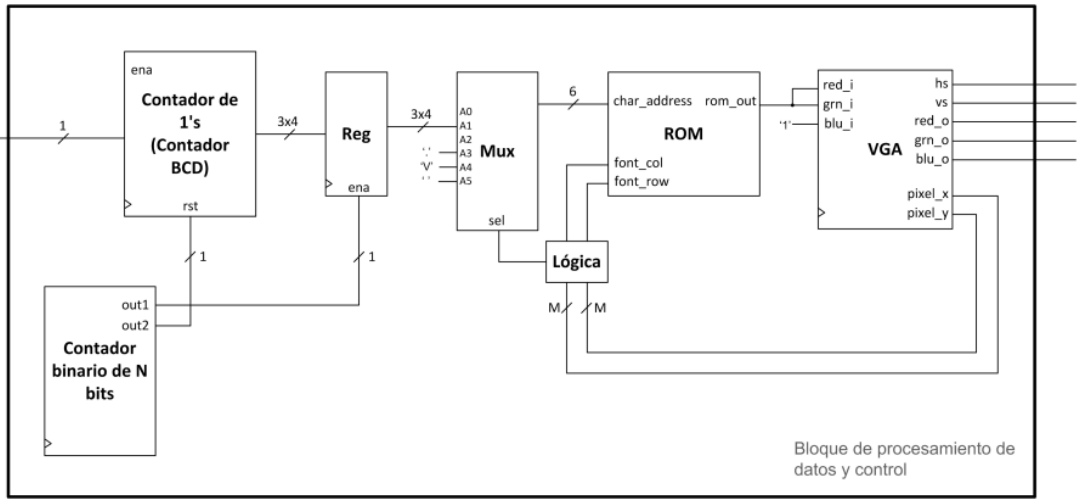

Figura 2: Diagrama en bloques propuesto.

El funcionamiento de este diagrama de bloques comienza con el contador binario de N bits, el cual cuenta una "ventana" de tiempo, en el cual el contaodor de 1's BCD cuenta la cantidad de 1's presentes en la ventana temporal. Cuando la ventana de tiempo termina, el registro "Reg" captura el valor BCD contado por el contador BCD, y, por las razones que se analizan más adelante, ese valor BCD ya es la medición real de tensión.
Un multiplexor tendrá en sus entradas los valores BCD de la medición de tensión capturada por el registro "Reg", así como también caracteres generales como el símbolo ".", "V" y carácter vacío. Según la posición que se esté barriendo en pantalla, el bloque "Lógica" decide qué salida debe presentar el multiplexor, a través de la entrada selectora del mismo multiplexor.
Una memoria ROM almacena la información de los caracteres y números a mostrar en pantalla.
Finalmente, un controlador VGA se encarga de generar las señales de video correspondientes para que la medición pueda verse en un monitor VGA.

En este trabajo particular se respetó en gran medida este diagrama general, añadiendo algunos cambios a lo largo del diseño por conveniencia o practicidad.

El diagrama en bloques utilizado para este diseño en particular es el siguiente:

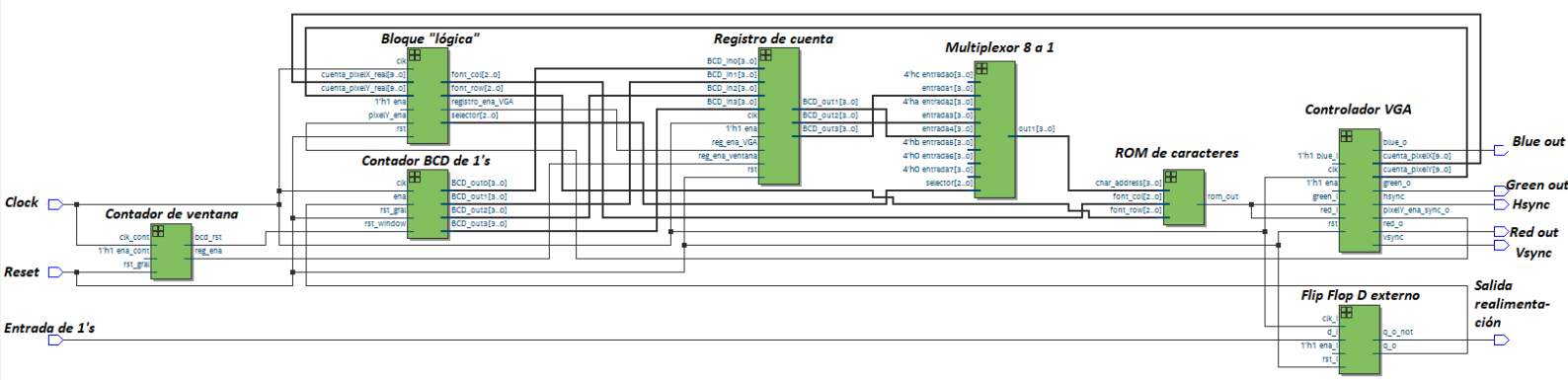

Figura 3: Diagrama en bloques del diseño del alumno.

En este informe se analizará cada uno de los bloques, con su propósito y sus entradas y salidas.

3. Desarrollo del alumno

3.1. El contador binario de N bits (contador de ventana)

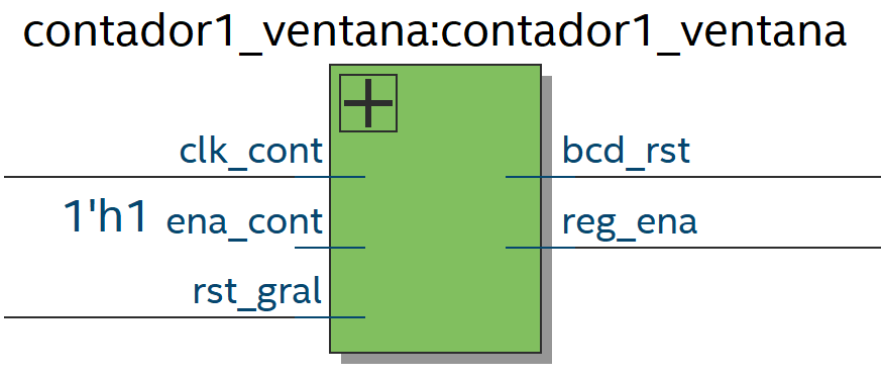

Figura 4: El contador binario de N bits (contador de ventana)

El contador de ventana de tiempo es un contador binario de 12 bits, el cual está diseñado para contar hasta el número decimal 3300. Como el rango de medición del voltímetro es de 0 a 3,3 V, el elegir un número de máxima cuenta que sea múltiplo de 3,3 es muy conveniente, ya que con sólo contar cuántos 1's se reciben en una ventana de tiempo desde el conversor analógico digital, alcanza para saber el valor de tensión medido, sin hacer más cuentas.

El propósito del contador de ventana de tiempo es establecer la ventana de tiempo donde se podrán contar los 1's provenientes del conversor analógico digital, para luego poder tomar dicha cuenta de 1's y tratarla como la medición real de tensión. Para ello, el contador de ventana de tiempo, al terminal la ventana, debe dar aviso al registro para que efectivamente saque a su salida el valor final de cuenta de 1's. Este aviso lo brinda con la salida **reg_ena.** También, el contador de ventana debe dar aviso para resetear al contador BCD de 1's, pero este aviso debe ser posterior al aviso al registro, de lo contrario el registro captará una cuenta incorrecta, que sería cero, por el reciente reset. Este aviso lo brinda con la salida **bcd_rst.**

Este contador de ventana fue construido a partir de un "Registro "n" " que cuenta con 12 flip flops tipo D, y de una lógica entre ellos que logra que los mismos cuenten en binario, sin restricción de máxima cuenta. La lógica utilizada para lograr el conteo ascendente en binario es la siguiente (se muestra solamente para 4 flip flops):

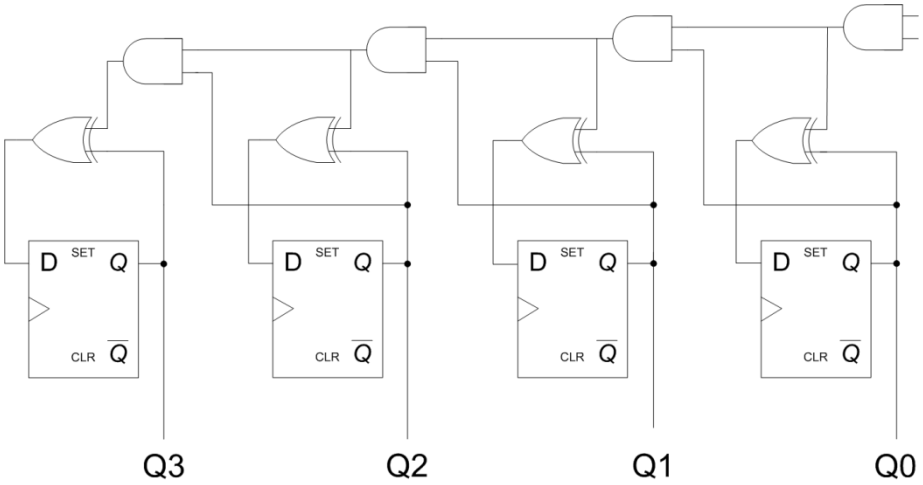

Figura 5: Lógica para un contador binario genérico.

El reseteo de este contador se realiza mediante un comparador que compara continuamente la salida contra la constante binaria 110011100100, que es el decimal 3300. Cuando el contador llega a 3300, el comparador activa la señal “max_cuenta” que dura 1 ciclo de clock.

Dicha señal también sale afuera del módulo, y es la que actuará como habilitador del registro de cuenta (**reg_ena**). También, por lo comentado anteriormente, esta señal “max_cuenta” no puede actuar de reset del contador BCD de 1’s, ni tampoco de sí mismo, porque se resetearía inmediatamente al llegar la cuenta a 3300 (no un ciclo de clk después), y eso no es deseado. Por ello, la señal “max_cuenta” es pasada por un flip flop D, de manera que su salida es la misma señal “max_cuenta” pero retardada un ciclo de clock. Ahora, esta nueva señal sí puede actuar de reset del contador BCD de 1’s (**bcd_rst**), y también de sí mismo, para volver a cero (reset particular).
Se muestra a continuación un diagrama de cómo es el contador por dentro:

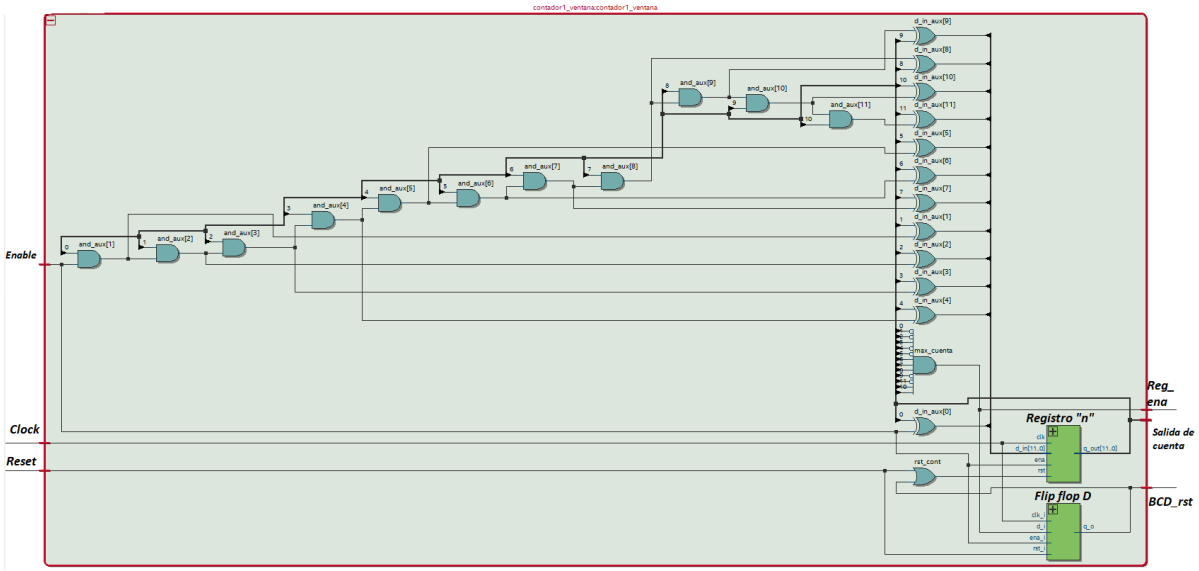

Figura 6: Diagrama en bloques del contador de ventana.

Finalmente, el clock, reset general e enable de este módulo se conectan directamente al del sistema.

3.2. El contador BCD de 1's

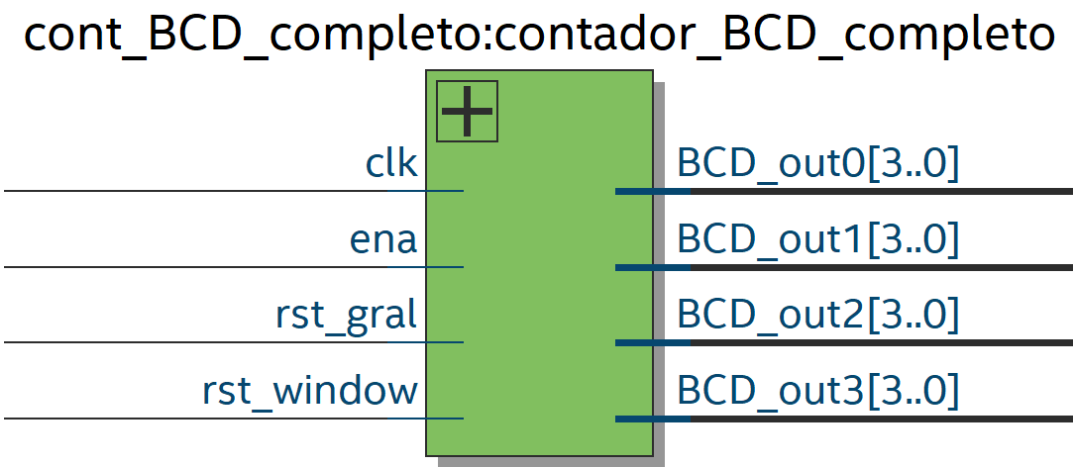

Figura 7: El contador BCD de 1s.

El propósito de este contador es contar en formato BCD la cantidad de 1's presentes en la ventana de tiempo establecida. El mismo fue implementado a partir de 4 contadores BCD singulares con señales de máxima cuenta, con una lógica entre ellos que les permite contar en conjunto.
Este contador completo es reseteado por el contador de ventana de tiempo (a través de la entrada **rst_window)**, cuando este llega a su máxima cuenta. El clock (**clk**), reset general (**rst_gral**) e enable (**ena**) de este módulo se conectan directamente al del sistema.

Su estructura interna se muestra a continuación:

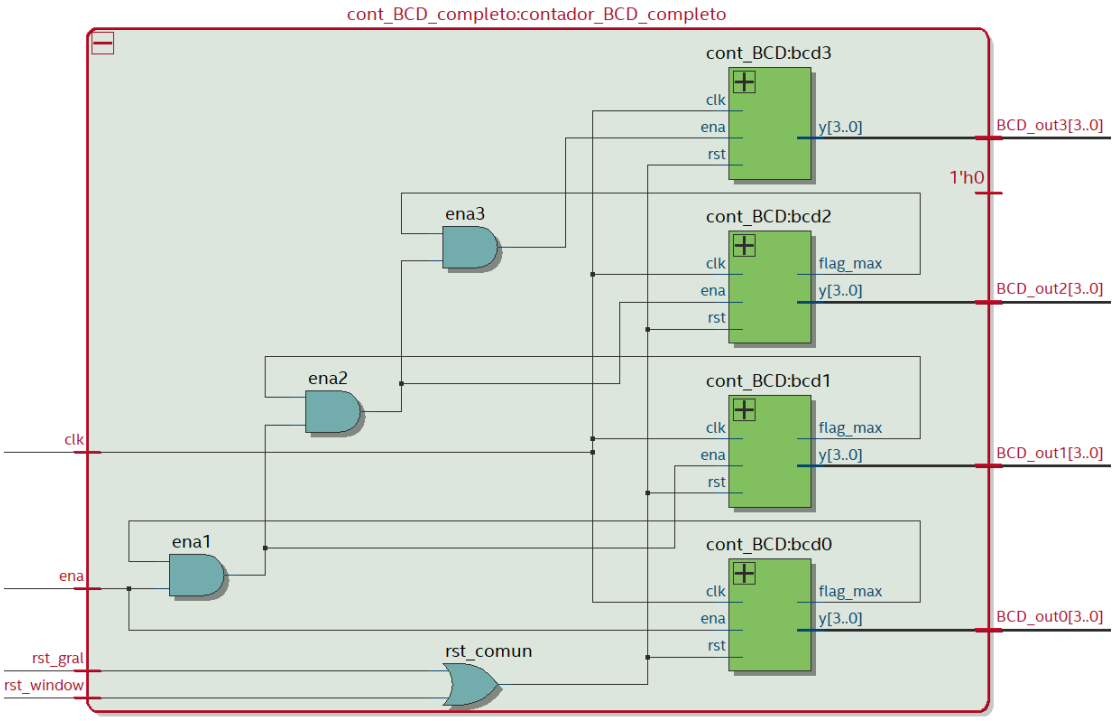

Figura 8: Estructura interna del contador BCD de 1s.

En la figura anterior se puede observar la lógica de interconexión entre contadores BCD singulares. Un contador menos significativo habilita al siguiente, para una sola cuenta en el próximo clock, cuando su flag de máxima cuenta se haya activado y cuando el contador menos significativo de todos haya llegado a su máxima cuenta también.

3.3. El registro de cuenta

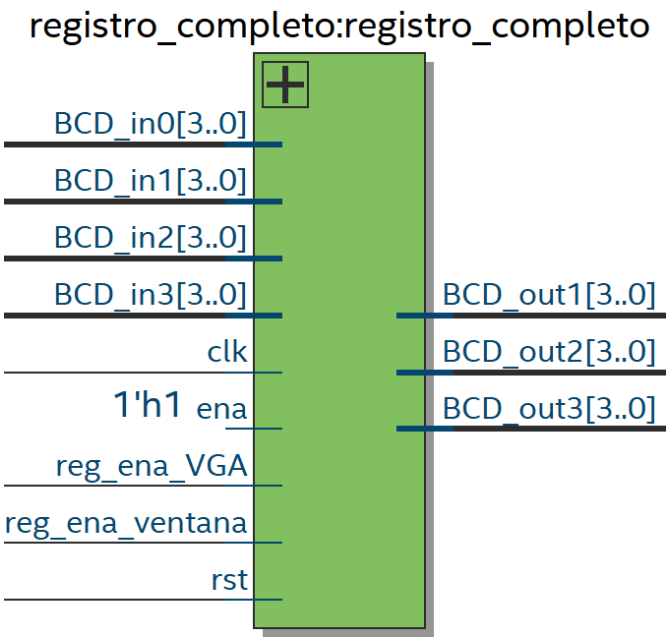

Figura 9: El registro de cuenta.

El registro de cuenta tiene como entradas a los dígitos BCD que vienen del contador BCD de 1's **(BCD_in0 a BCD_in_3)**, y también, dos habilitadores distintos: el de ventana de tiempo (**reg_ena_ventana)** y el habilitador VGA (**reg_ena_VGA).**

El registro de cuenta está formado por 4 grupos de 4 flip flops D cada uno, permitiendo manejar los 4 dígitos BCD que originalmente se disponen (aunque en pantalla se terminarán mostrando sólo 3). Es decir, tiene 4 entradas de 4 bits c/u para los dígitos BCD de entrada, y también 3 salidas de 4 bits cada una para los dígitos BCD de salida **(BCD_out1 a BCD_out3).**

Las entradas de este registro, al estar conectadas a la salida del contador BCD de 1's, están cambiando en cada ciclo de clock. Sin embargo, algún valor de la entrada sólo pasará a la salida cuando se active el enable de este registro de cuenta. Dicho enable tiene dos fuentes de activación, las cuales **ambas** deben activarse para activar al registro. La primera fuente es el contador de ventana (**reg_ena_ventana**), que activa al registro únicamente cuando termina la ventana de tiempo, y la segunda es el bloque "lógica" (**reg_ena_VGA)**, que impide que el registro de cuenta cambie cuando se está dibujando en pantalla la zona donde están los caracteres, evitando dibujos borrosos e ilegibles.

El clock (**clk**) y reset (**rst)** de este bloque se conectan a los del sistema general. El enable general (**ena**) de este bloque se conecta al del sistema, que está seteado a '1'. De todas formas, el registro sólo se habilita cuando los otros dos enables estén también ambos activos.

3.4. El multiplexor

El multiplexor utilizado tiene 8 entradas y 1 salida, de 4 bits de ancho de datos cada una, y tiene como función principal brindarle a la memoria ROM una dirección de memoria correcta para que la misma sepa cuál carácter enviar a pantalla, en base a la posición de barrido de pantalla.

El multiplexor tiene en cada una de sus 8 entradas, en orden, lo siguiente:

Número de entrada	**Descripción**	**Valor en el selector**
0	Dirección de ROM de carácter vacío (es 1100)	000
1	1er dígito BCD más significativo	001
2	Dirección en ROM del carácter punto (es 1010)	010
3	2do dígito BCD más significativo	011
4	3er dígito BCD más significativo	100
5	Dirección en ROM del carácter de Volt	101
6	No usado (es 0000)	110
7	No usado (es 0000)	111

Tabla 1: Entradas del multiplexor.

Dependiendo de la posición de barrido en pantalla (a través de contadores de pixel X y pixel Y), el bloque "lógica" elige el valor particular del selector que posee este multiplexor. Según el valor de selector elegido (de 0 a 7 en binario) se elige, por su número de orden, la entrada a sacar a la salida. En todos los casos son direcciones a mandar hacia la memoria ROM, la cual

tiene guardados los patrones de 0's y 1's específicos para cada carácter.
En el caso particular de los dígitos BCD, estos justamente no son direcciones, son los valores numéricos reales. Sin embargo, la memoria ROM está armada de tal manera que los valores numéricos reales BCD coinciden con la dirección en ROM del comienzo de la descripción de dicho carácter BCD. Para el carácter vacío, carácter punto y carácter Volt, las direcciones son las especificadas arriba.

Con respecto a la construcción del multiplexor, el mismo fue armado a partir de una celda básica, que es un multiplexor de 2 entradas y una salida, con 1 bit de ancho de datos en cada entrada / salida, como se muestra en la siguiente figura:

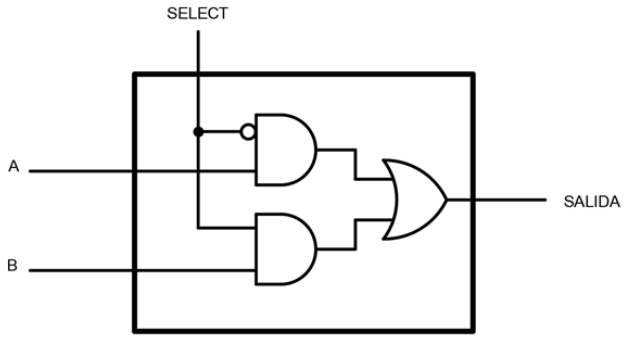

Figura 10: Celda básica de multiplexor de 2 entradas y una salida, con 1 bit de ancho de datos.

Dicha celda puede ser fácilmente ampliada en cuanto a ancho de bits de entrada /salida, simplemente poniendo 4 celdas más en "paralelo": esto es, conectando los terminales de "select" juntos, y luego agrupando los 4 terminales "A", "B" y "salida" como un único terminal de ancho de 4 bits cada uno, como muestra la siguiente figura:

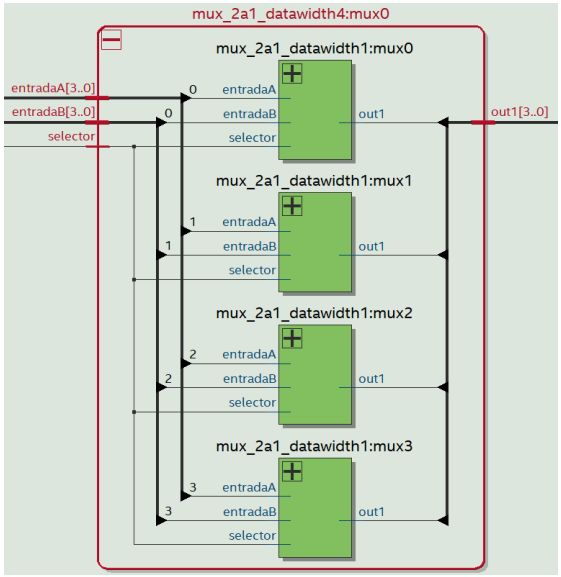

Figura 11: Ampliación del ancho de bits de entrada /salida de la celda básica.

Una vez formado este multiplexor de 2 entradas y 1 salida, pero con 4 bits de ancho de datos, necesitaremos tomar 7 de estos multiplexores para combinarlos de tal manera de lograr las 8 entradas, en lugar de 2. La forma de combinar estos multiplexores se muestra a continuación:

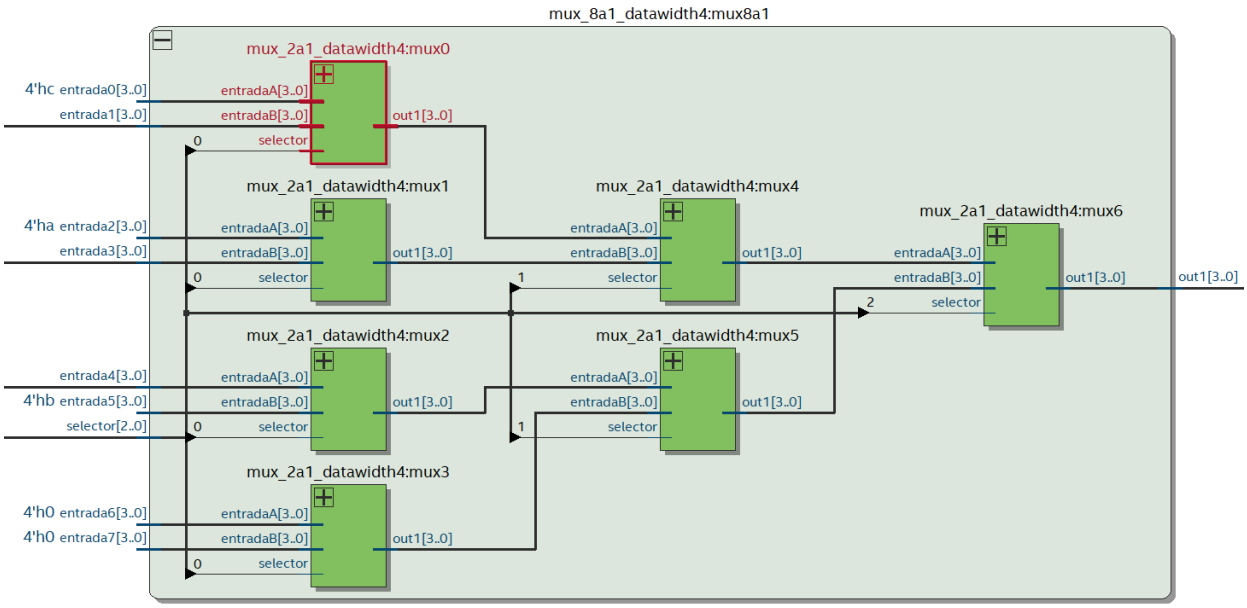

Figura 12: Ampliación de la cantidad de entradas de un multiplexor de 2 entradas, hacia 8 entradas.

3.5. La memoria ROM de caracteres

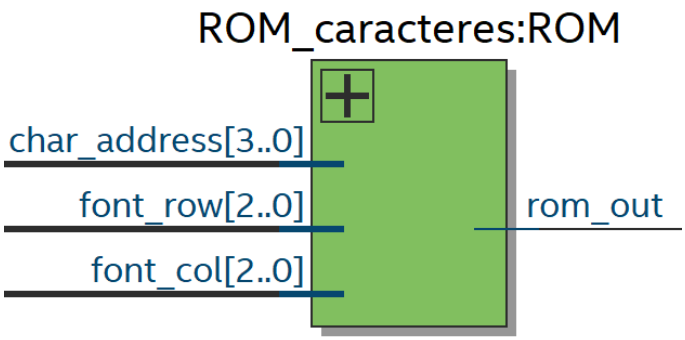

Figura 13: La memoria ROM de caracteres.

La memoria ROM de caracteres contiene todos los patrones de 1's y 0's que forman, en pantalla, a los números y caracteres a utilizar. Está organizada en palabras de 8 bits (vectores de 8 bits), conteniendo 128 de las mismas. La información de cada carácter está contenida en 8 palabras de 8 bits cada uno, y podemos pensarlo como una representación del carácter en un display de 8x8 pixeles. Donde figure '1' tendremos dibujado un pixel blanco de algún carácter, y donde haya un '0' no habrá color blanco sino el color de fondo predeterminado.

Se muestra a continuación un ejemplo de cómo está formado el número "0", con sus 8 vectores de 8 bits cada uno:

```
-- "0"
        "00000000",
        "00011000",
        "00100100",
        "00101100",
        "00110100",
        "00100100",
        "00011000",
        "00000000",
```

Figura 14: Código VHDL para el número "0" en memoria ROM.

La siguiente figura explica muy bien la estructura de la memoria:

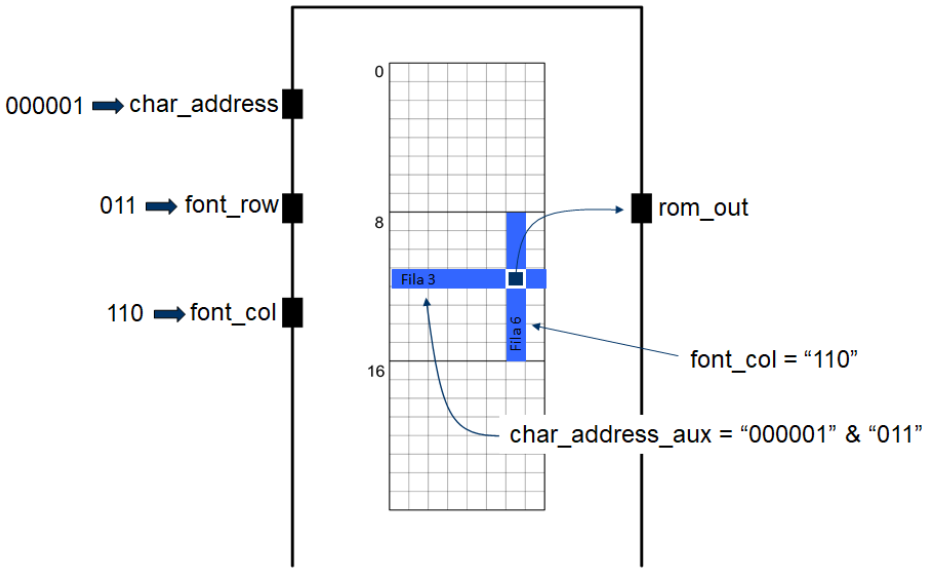

Figura 15: Estructura de la memoria ROM (Fuente: Presentación ROM.ppt, cátedra de Electrónica Digital I UNSAM).

En la figura anterior podemos observar que las direcciones del comienzo de la información de los distintos caracteres están separadas 8 posiciones. Entonces, la dirección para el comienzo del dígito "1" sería 8, la del dígito "2" sería 16, y así sucesivamente. Sin embargo, si miramos el 3er, 4to y 5to bit de la dirección total de cada fila, esta coincide con el dígito BCD a buscar en sí mismo. Haciendo esto, podemos enviar los números BCD directamente a la memoria, a través de su entrada **char_address**, sin transformarlos en direcciones particulares. Luego, concatenamos un valor que de información de la fila en la que se está en pantalla (la entrada **font_row, de 3 bits**), y así formamos la dirección total de la fila.
Luego, la memoria ROM tiene una entrada adicional de 3 bits, que le permiten seleccionar columna (**font_col**) del carácter, lo que permite saber exactamente qué bit sacar a la única salida (**rom_out**) que tiene la ROM.

Finalmente, el hardware específico de este bloque fue realizado automáticamente por el compilador, y no por el usuario, ya que sería muy voluminoso.

3.5.1. Estructura de la pantalla visible

La memoria ROM tiene almacenados caracteres en formato 8x8 pixeles, y esta no es la forma en la que se mostrarán en pantalla. Por ello, es conveniente primero definir la estructura de los caracteres en pantalla que se utilizarán, para luego adaptar los caracteres de 8x8 pixeles a ello.

La estructura en pantalla que se utilizará es la siguiente:

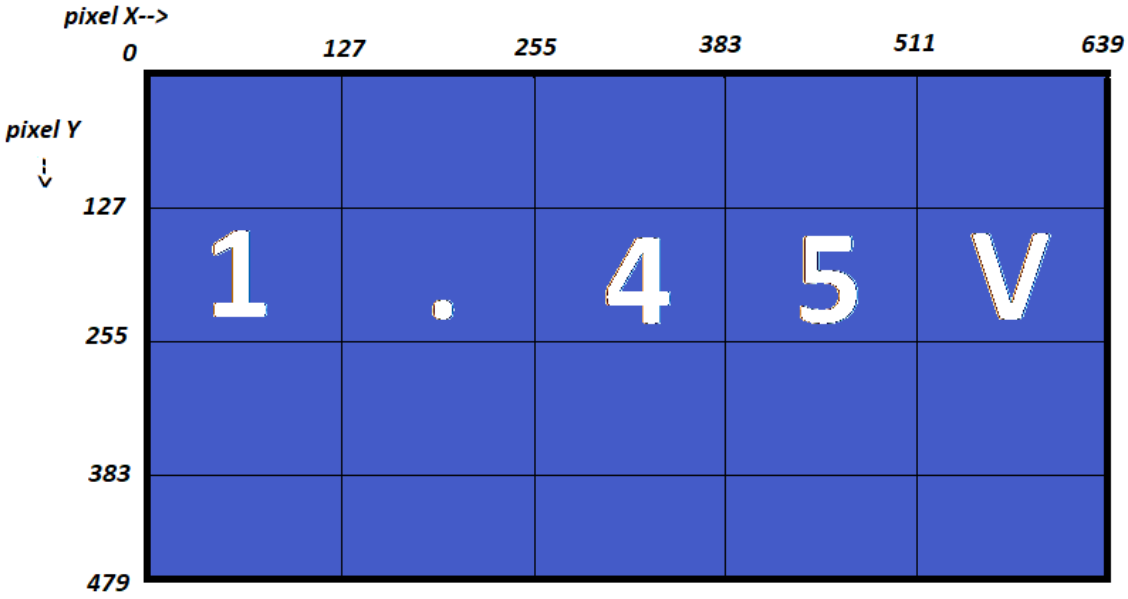

Figura 16: Estructura de la pantalla a utilizar.

Como podemos observar, cada cuadrante para los caracteres tiene 128x128 pixeles, y en memoria disponemos de los mismos en 8 x 8 pixeles. Sin embargo, si al momento de dibujar en pantalla repetimos cada pixel que vemos en esa memoria de 8x8 pixeles, 16 veces (tanto horizontalmente como verticalmente) , estaríamos convirtiendo esos 8x8 pixeles en 128 x 128 pixeles, manteniendo la proporción del dibujo.

Se dispone de contadores de pixel X y pixel Y que van barriendo la pantalla de izquierda a derecha, siendo el bloque “VGA controller” el que se encarga de ellos. Cuando el contador de pixel X termina de barrer una línea horizontal, se incrementa en 1 el contador de pixel Y. Originalmente, como la memoria es para caracteres de 8x8 pixeles, y la misma necesita 3 bits de fila y de columna, alcanzaría con darle los 3 bits menos significativos de estos contadores de pixel X y pixel Y (bits 0, 1 y 2) como columna y fila, respectivamente. Sin embargo, en lugar de dar estos bits, cuya cuenta en X aumenta en cada clock, podemos dar los bits 3, 4 y 5 de ambos contadores, cuya cuenta en X aumenta cada 16 clocks, como cuenta de fila y columna. Así, durante 16 clocks se da el mismo número de fila y columna a la ROM, y la misma repite el mismo bit 16 veces a su salida, tanto horizontal como verticalmente. Queda entonces en pantalla el caracter ampliado a 128 x 128 pixeles.

3.6. El controlador VGA

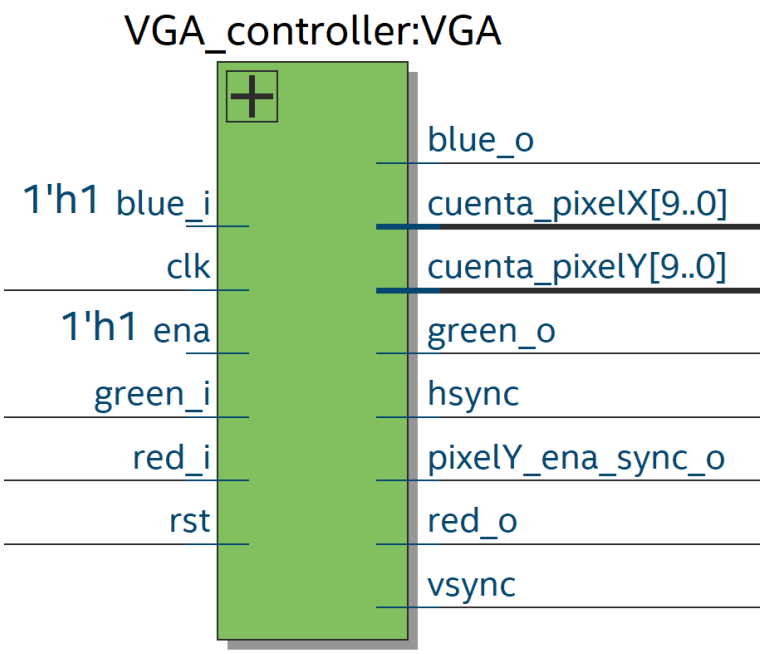

Figura 17: El controlador VGA.

El controlador VGA es el módulo que se encarga de varias tareas relacionadas a la salida de video VGA. En primer lugar, este módulo posee los contadores de pixel X y pixel Y, cuya cuenta saca por su salida **cuenta_pixel X [9..0]** y **cuenta_pixel Y [9..0].** Estos contadores llevan la cuenta de cada pixel que se va barriendo en pantalla, incluso de aquellos que **no son visibles**.

También se encarga de generar las señales necesarias para enviar video al puerto VGA. Estas señales se componen de dos señales de sincronismo horizontal y vertical, llamadas **hsync** y **vsync,** y dos señales de habilitación de video horizontal y vertical, llamadas **h_vidon** y **v_vidon**. Estas últimas pueden quedar de forma interna en el módulo, no es necesario que salgan afuera.

El controlador VGA recibe señales de video como entrada, en forma de tres colores (rojo, verde y azul), y decide cuándo enviarlas realmente hacia la salida VGA según los "timings" de sus señales de video. Es decir, no en cualquier momento se pueden enviar señales de colores hacia la salida VGA. Esto se explica mejor en el siguiente apartado.

3.6.1. Las señales de video VGA

El origen de las señales de video VGA está relacionado con la vieja tecnología de monitores llamada CRT o tubo de rayos catódicos. En esta tecnología, un haz de electrones se hacía incidir sobre una pantalla fluorescente, tal que la misma emita luz por un corto período de tiempo cuando el rayo la alcanzaba en cierta posición. Para dibujar toda una imagen, el haz de electrones tenía que recorrer toda la pantalla. Empezaba por la esquina superior izquierda de la pantalla, recorría toda una línea horizontal hacia la derecha, luego retraía el haz de electrones hacia la izquierda mientras bajaba una línea verticalmente, y repetía el proceso hasta cubrir toda la pantalla. Al llegar al final de la pantalla, el haz también era retraído verticalmente hasta su posición inicial, la esquina superior izquierda.

Se necesitaban señales con forma de “diente de sierra” para mover a los haces de electrones. Sin embargo, bastaba con darle al monitor una señal digital con timings específicos, y luego el mismo monitor se encargaría internamente de generar las señales diente de sierra. Esas señales digitales con timings específicos son las llamadas **hsync** y **vsync.**

En la siguiente figura se puede observar la región visible de la pantalla, junto con un par de señales horizontales necesarias: el h_vidon y el hsync.

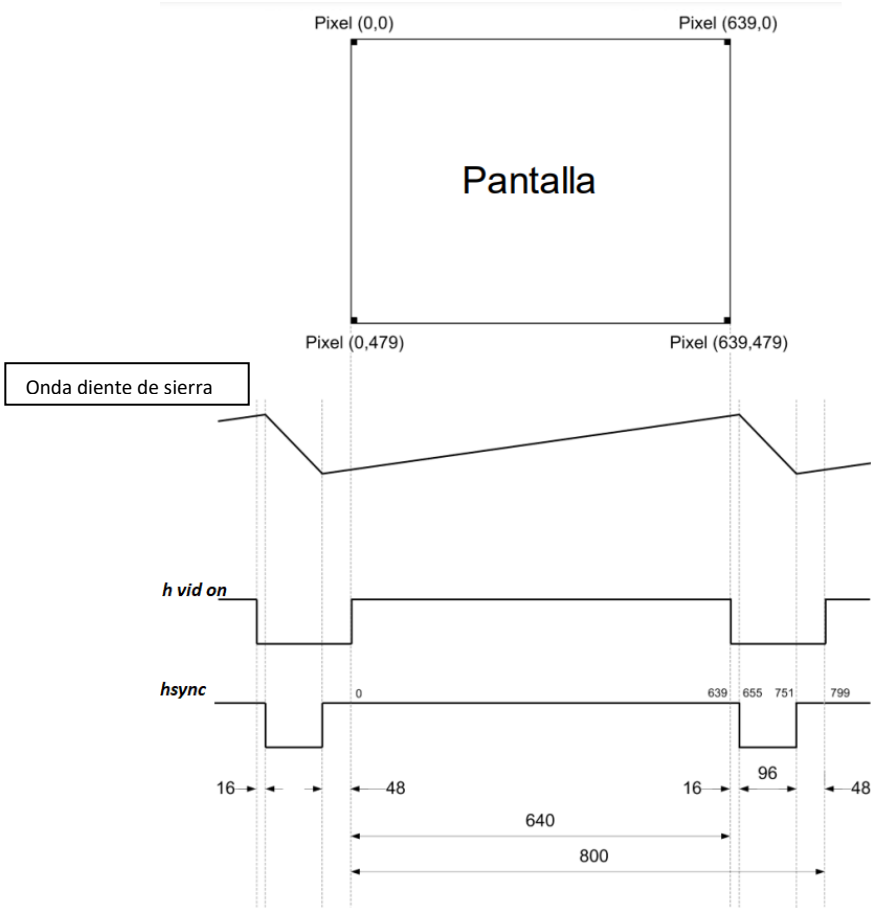

Figura 18: La zona visible de la pantalla y las señales de video horizontales.

La pantalla tiene zonas no visibles, llamadas “back porch” y “front porch”. Estas zonas no visibles le dan tiempo al haz de electrones a retraerse hacia su posición inicial para volver a barrer una línea.

Debido a estas porciones no visibles, tanto horizontales como verticlaes, los contadores de pixel X y pixel Y no contarán hasta 640 y 480 respectivamente (que es la resolución de la pantalla), sino que necesitan hacerlo hasta **800 y 525**. Cuando se recorre toda una línea horizontal (es decir, cuando el contador de pixel X llega a 800) el contador de pixel Y puede avanzar una unidad.

Las señales **h_vidon** y **v_vidon** son señales digitales que indican cuándo es correcto enviar señal de video al puerto VGA propiamente dicho. Ambas tienen que estar activas a la vez para

poder enviar señal de video correctamente.
Los timings de todas las señales de video se pueden observar en la siguiente figura:

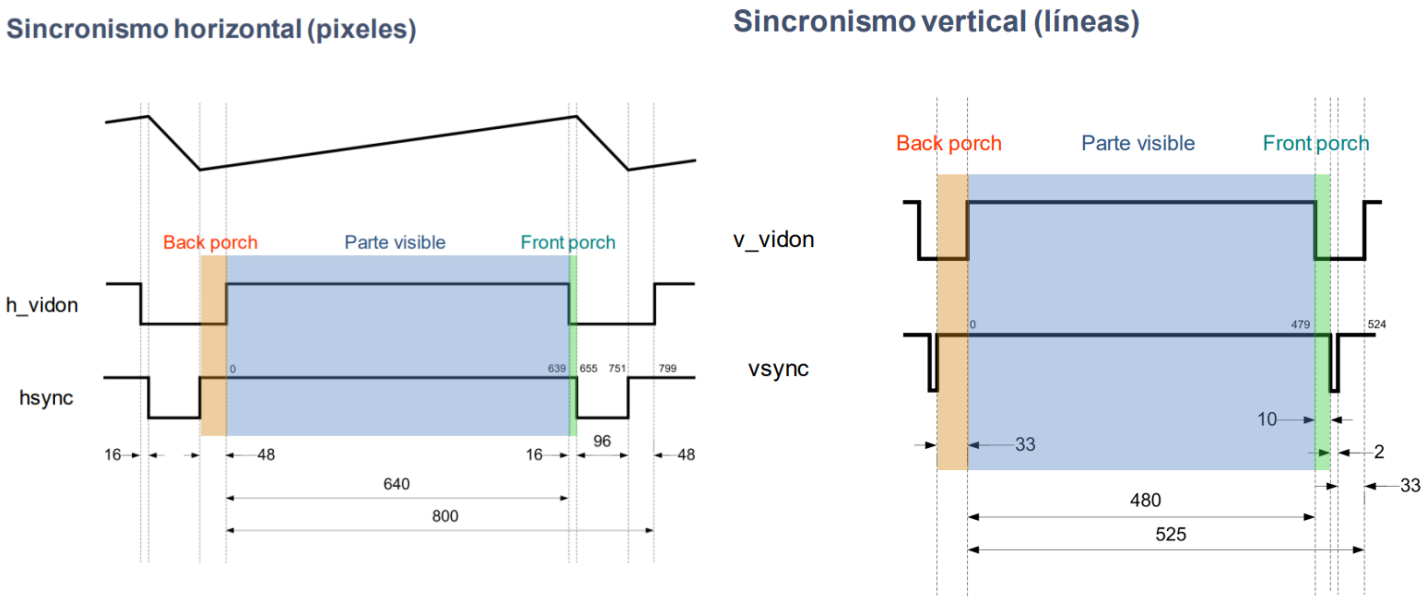

Figura 19: Timings para las distintas señales de video horizontales y verticales.

En todos los casos, los números indicados representan cuentas del contador de pixel correspondiente. El clock que maneja estas señales de video debe ser de 25Mhz, para lograr una frecuencia de actualización de pantalla de 60 Hz, un valor típico.

3.6.2. Implementación del controlador VGA

En la siguiente figura podemos observar el interior del controlador VGA implementado:

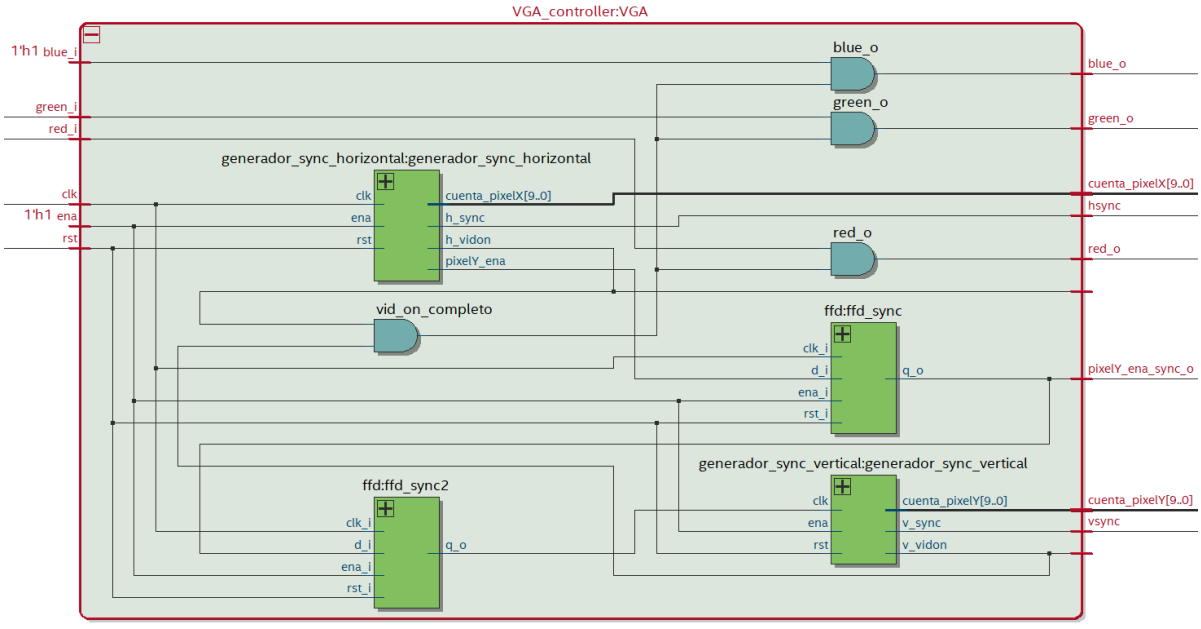

Figura 20: Estructura interna del controlador VGA

El primer bloque interno a destacar es el **generador de sincronismo horizontal (generador_sync_horizontal).** El mismo cuenta con un contador de pixel X, de 0 a 799, el cual se auto resetea en el clock siguiente de llegar a la cuenta máxima.
También, el bloque cuenta con dos flip flops tipo T, uno para generar la señal **hsync**, y otro para generar la señal **h_vidon**. Existen comparadores, que están comparando continuamente el valor de la cuenta del contador de pixel X, con valores específicos de los timings de la figura 19 y sus salidas asertan las entradas de los flip flops T para cambiar su salida y poder generar las señales digitales correctamente.

Una salida muy importante del generador de sincronismo horizontal es la señal "**pixelY_ena**". La misma se activa cuando el contador horizontal llega a su máxima cuenta, indicando que terminó de barrer una línea horizontal. Entonces, esta señal "**pixelY_ena"** actúa como "clock" del **generador de sincronismo vertical** (que contiene al contador de pixel Y), para que este avance únicamente cuando el contador de pixel X haya terminado de barrer una línea completa horizontal.
El enviar una señal combinacional como clock de un módulo secuencial no es una buena práctica, y es una de las razones por las que inicialmente el diseño actual no funcionaba en hardware, pero sí en simulación. El motivo es la desincronización que puede existir entre la señal combinacional y el clock general del sistema. Por ello, primero la señal "**pixelY_ena**" es sincronizada a través de un flip flop D con el clock del sistema, y luego es utilizada como clock del contador de pixel Y. En realidad, es sincronizada con dos flip flops tipo D. El motivo de esto es para dejar bien en fase a las señales hsync y vsync, que demostraron estar correctas en timings pero dos clocks desfasadas en simulación. De esta forma, el **generador de sincronismo vertical**, que también contiene dos flip flops T en su interior, genera correctamente las señales **vsync** y **v_vidon** de forma análoga al generador de sincronismo horizontal.

Finalmente, el módulo toma las señales de colores rojo, verde y azul a la entrada, y las saca a la salida únicamente cuando ambas señales de vid on, **h_vidon** y **v_vidon**, estén activas.

3.7. El bloque "lógica".

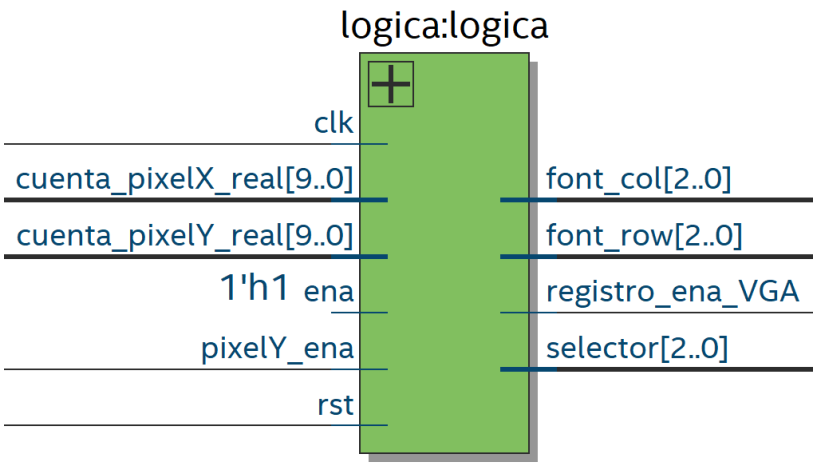

Figura 21: El bloque "lógica"

El bloque "lógica" se encarga de varias tareas respecto a la posición actual de barrido en pantalla. En primer lugar, se encarga de gestionar la entrada selectora del multiplexor, según la posición actual de barrido en pantalla. También, se encarga de brindar los bits correspondientes a la memoria ROM de fila y columna, tal que la misma pueda saber correctamente qué bit exacto sacar a su salida, nuevamente, según la posición en un cuadrante específico de la pantalla.

Es por ello que este bloque tiene como entradas a los contadores de pixel X (**cuenta_pixelX_real)** y pixel Y (**cuenta_pixelY_real**). También necesita de la señal "**pixelY_ena"**, que, como se mencionó previamente, es la señal que actúa de clock para el contador de pixel Y. El motivo de por qué necesita esta señal es porque internamente este bloque tiene dos nuevos contadores de pixel X y pixel Y, pero con su cero posicionado o alineado con el pixel cero de la zona visible de la pantalla, tanto horizontal como vertical. De esta forma, podemos saber con exactitud en qué pixel horizontal y vertical estamos en pantalla en cierto momento, sin tener que hacer restas adicionales sobre los contadores de pixel X y pixel Y "reales". Podemos llamarlos contadores de pixel X e Y "corregidos".

El bloque tiene también en su interior varios comparadores. Los comparadores monitorean constantemente los bits 7, 8 y 9 de los contadores de X e Y "corregidos" y de esta forma pueden saber qué cuadrante horizontal y vertical (de 128 x 128 pixeles) en pantalla se está barriendo en el momento. La siguiente imagen numera los cuadrantes horizontales, verticales, y los de texto:

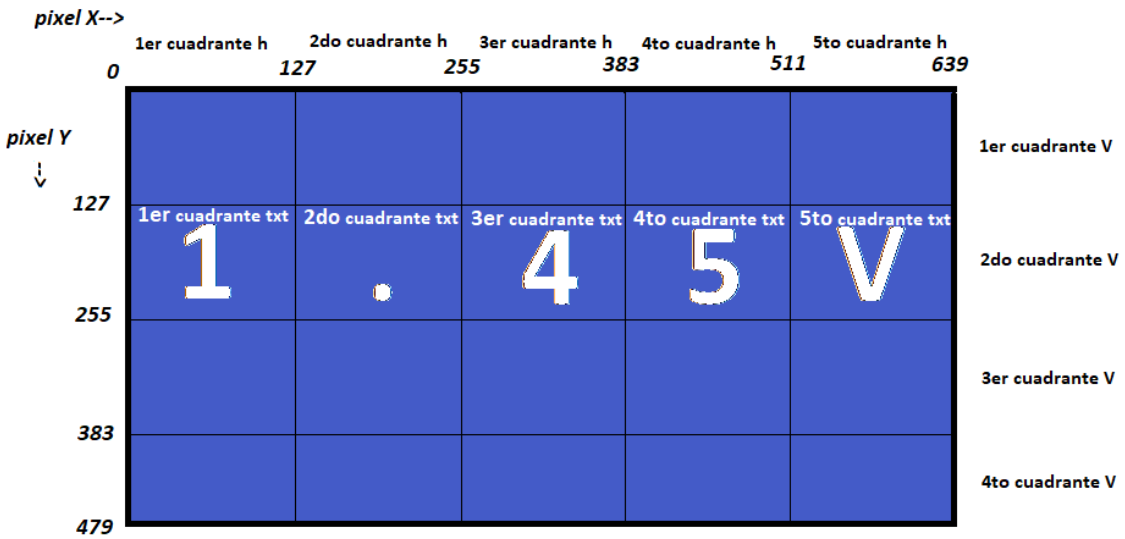

Figura 23: Numeración de cuadrantes en pantalla.

A partir de la información de cuadrantes horizontales y verticales, es muy fácil saber si se está barriendo algún cuadrante donde se deban dibujar caracteres, o si no. La siguiente porción de código lo explicita claramente:

```
flag_estoy_en_1cuadrante<=flag_h_1cuadrante and flag_v_2cuadrante;
flag_estoy_en_2cuadrante<=flag_h_2cuadrante and flag_v_2cuadrante;
flag_estoy_en_3cuadrante<=flag_h_3cuadrante and flag_v_2cuadrante;
flag_estoy_en_4cuadrante<=flag_h_4cuadrante and flag_v_2cuadrante;
flag_estoy_en_5cuadrante<=flag_h_5cuadrante and flag_v_2cuadrante;
```

Figura 22: Código VHDL para discriminar entre cuadrantes de pantalla.

Los valores de los flags *"flag_estoy_en_xcuadrante"* indican en qué cuadrante de texto se encuentra el barrido actual, y son muy importantes, ya que de esto depende la decisión de qué valor seleccionar en la entrada selectora del multiplexor.

La siguiente tabla especifica el valor que tomará el selector del multiplexor, según los valores de los flags mencionados anteriormente.

Flags de cuadrante de texto					Selector			Observación
Flag_c1	Flag_c2	Flag_c3	Flag_c4	Flag_c5	S(2)	S(1)	S(0)	
0	0	0	0	0	0	0	0	Corresponde a carácter vacío
1	0	0	0	0	0	0	1	1er dígito BCD
0	1	0	0	0	0	1	0	Carácter punto
0	0	1	0	0	0	1	1	2do dígito BCD
0	0	0	1	0	1	0	0	3er dígito BCD
0	0	0	0	1	1	0	1	Carácter "V"

Tabla 2: Tabla de verdad para la salida del bloque "Lógica".

Para cada bit del selector, se construyó un circuito combinacional sumando mini términos, tal que cumpla la tabla anterior. Como no puede haber más de un flag de cuadrante activo a la vez, el tamaño de la tabla se reduce bastante.

Cuando la pantalla se está barriendo en una zona donde no hay caracteres, todos los flags de cuadrante están en cero, por lo tanto se elige al carácter vacío para que el multiplexor saque su dirección por su salida. Este carácter está compuesto en memoria por todos "0s", de tal manera que el resultado es mostrar el color azul en todas partes. Por esta razón, el fondo es azul.
También, es importante recordar que el bloque lógica tiene una salida, llamada "**registro_ena_VGA**", que habilita al registro de cuenta únicamente cuando **ningún flag de cuadrante de texto está activado**, o lo que es lo mismo, cuando no se está dibujando ningún carácter en pantalla.

Finalmente, el bloque lógica también entrega a la memoria ROM las salidas "**font_col"** y **"font_row"**, que le indican a la memoria ROM en qué fila y columna de un cuadrante de 128x128 se encuentra el barrido actual, como para que la misma sepa qué bit sacar a su salida. Como se mencionó anteriormente, se envían como "**font_col"** y **"font_row"**, los bits 3, 4 y 5 de los contadores de pixel X y pixel Y "corregidos", porque estos bits sólo cambian cada 16 cuentas, y esto es necesario pues convierte los dígitos hechos en 8x8 pixeles, a 128x128 pixeles.

4. Utilización de recursos

En las siguientes figuras podemos observar información sobre los recursos utilizados en la placa:

Resource	Utilization	Available	Utilization %
LUT	164	20800	0.79
FF	138	41600	0.33
IO	11	210	5.24
MMCM	1	5	20.00

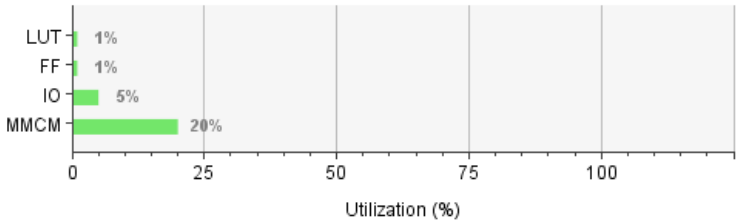

Figura 23: Utilización de LUTs, y flip flops.

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (5)
Voltimetro_toplevel	164	138	2	67	164	90	11	2	1
clk25MHz_gen (clk_wiz...	0	0	0	0	0	0	0	2	1
inst (clk_wiz_0_clk_...	0	0	0	0	0	0	0	2	1
genUnos (dac_pwm)	5	7	0	4	5	3	0	0	0
inst_voltimetro (Voltim...	108	94	2	40	108	52	0	0	0
contador1_ventana ...	13	13	0	5	13	9	0	0	0
ffd_aux (ffd_23)	1	1	0	1	1	0	0	0	0
reg_contador (re...	12	12	0	4	12	9	0	0	0
contador_BCD_co...	12	16	0	4	12	9	0	0	0
bcd0 (cont_BCD)	3	4	0	1	3	2	0	0	0
bcd1__0 (cont_B...	3	4	0	1	3	2	0	0	0
bcd2__0 (cont_B...	3	4	0	1	3	2	0	0	0
bcd3__0 (cont_B...	3	4	0	1	3	3	0	0	0
ffd_externo (ffd_neg...	1	1	0	1	1	0	0	0	0
logica (logica)	30	24	0	15	30	13	0	0	0
contador_pixel_a...	12	10	0	6	12	6	0	0	0
contador_pixel_a...	17	10	0	8	17	7	0	0	0
reg_contador ...	17	10	0	8	17	7	0	0	0
ffd_auxX (ffd)	0	1	0	1	0	0	0	0	0
ffd_auxY (ffd_2)	0	1	0	1	0	0	0	0	0
fft_conteo_pantal...	0	1	0	1	0	0	0	0	0
fft_conteo_pantal...	1	1	0	2	1	0	0	0	0
registro_completo (...	12	12	2	6	12	0	0	0	0
reg_bcd1 (registr...	0	4	0	2	0	0	0	0	0
reg_bcd2 (registr...	0	4	0	2	0	0	0	0	0
reg_bcd3 (registr...	12	4	2	5	12	0	0	0	0
VGA (VGA_controller)	40	28	0	15	40	19	0	0	0
ffd_sync (ffd_24)	0	1	0	1	0	0	0	0	0
ffd_sync2 (ffd_25)	0	1	0	1	0	0	0	0	0
generador_sync...	21	13	0	6	21	9	0	0	0
contador_vent...	20	11	0	6	20	7	0	0	0
fft_h_vidon (fft...	1	1	0	1	1	0	0	0	0
fft_hsync (fft_3...	0	1	0	1	0	0	0	0	0
generador_sync...	19	13	0	8	19	9	0	0	0
contador_vent...	18	11	0	7	18	7	0	0	0
fft_h_vidon (fft...	0	1	0	1	0	0	0	0	0
fft_vsync (fft_n...	1	1	0	2	1	0	0	0	0

Tabla 3: Utilización de slices por módulo.

5. Simulaciones

Captura Vsync

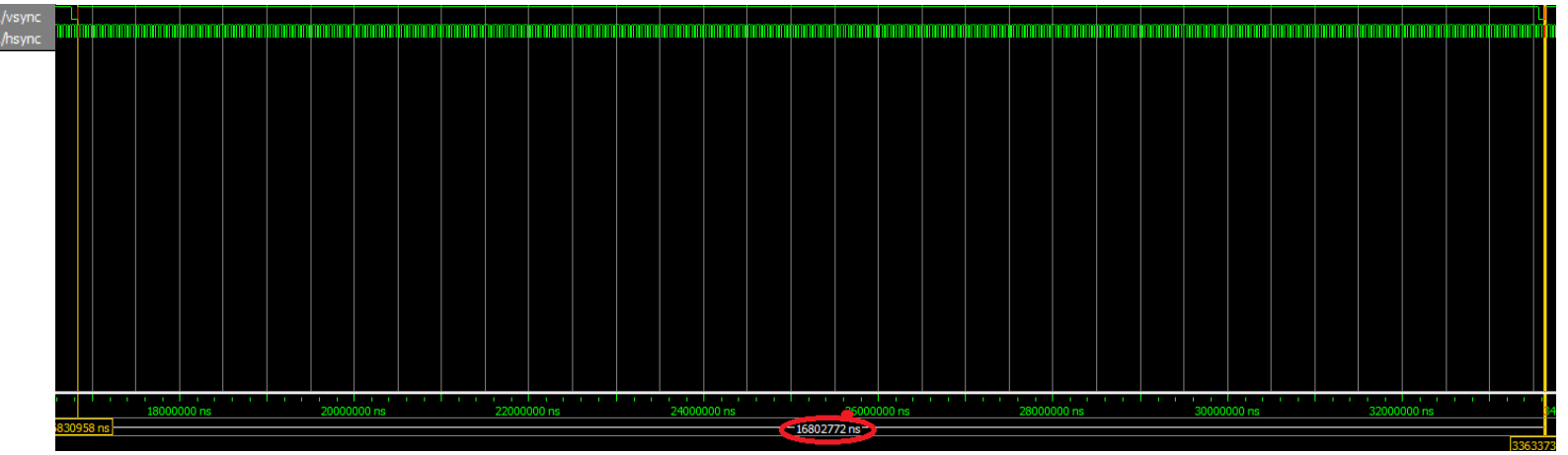

Figura 24: Señal vsync.

En la parte superior de la figura 24 podemos observar la señal **vsync,** y su período, que en la imagen figura como “16.802.772 ns” por error manual de cursores, pero en realidad el período es 16.800.000 ns, que es el resultado de multiplicar (1 ciclo de clock * 800 cuentas horizontales * 525 cuentas verticales) = 40ns *800 * 525.

Captura Hsync

Figura 25: Señal hsync.

La señal inferior de la figura 25 es la llamada **hsync**, en la que se puede observar que el período medido es exactamente 32000 ns, que es el resultado de multiplicar (1 ciclo de clock * 800 cuentas horizontales) = 40ns * 800.

Captura V vid on

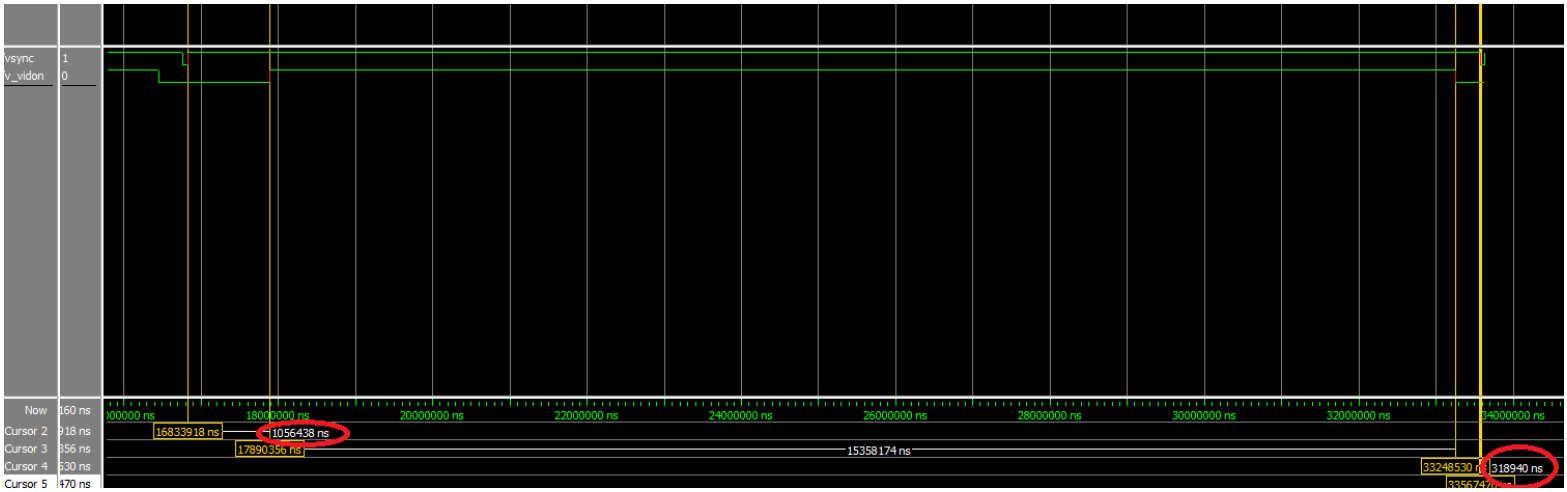

Figura 26: Señal v_vidon

En la figura 26, la señal superior es **vsync**, mientras que la inferior es **v_vidon**. Podemos observar cómo **v_vidon** sube 1.056.438 ns más tarde que **vsync**. En realidad, el timing real, sin errores de cursor manuales, es de 1.056.000 ns, y representa 33 cuentas verticales (1 cuenta vertical= 1 ciclo de clock * 800 cuentas horizontales = 32000ns).

Luego, observamos que el **v_vidon** cae 318.940 ns antes que **vsync**, siendo el número sin errores manuales 320.000 ns, indicando que **v_vidon** baja correctamente 10 cuentas verticales antes que **vsync**.
Se concluye entonces que los timings de **v_vidon** cumplen correctamente lo que establece la figura 19.

Captura H Vid on

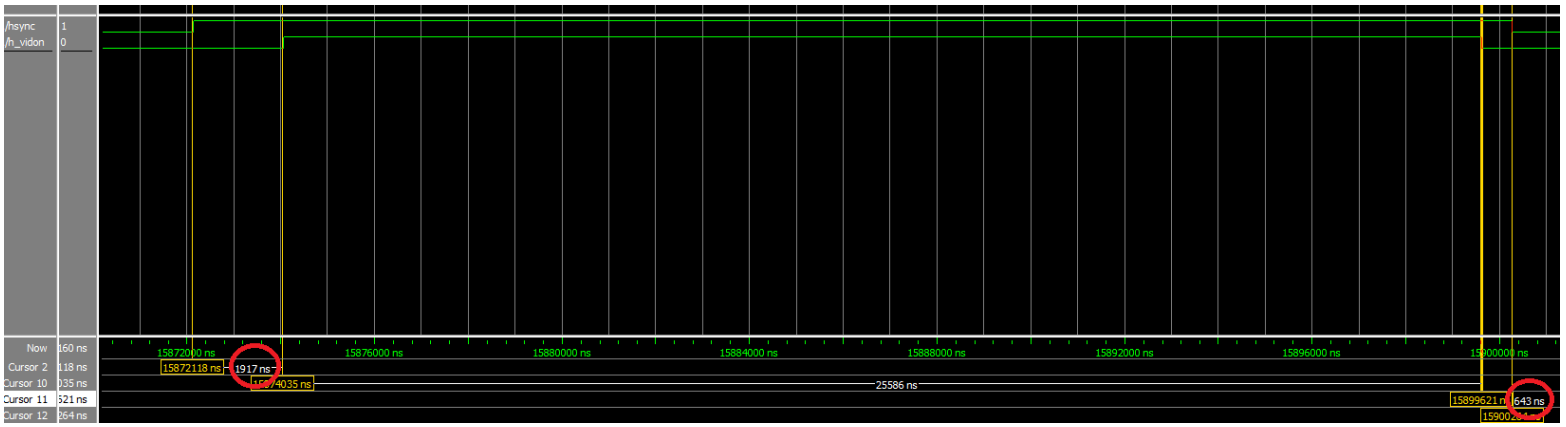

Figura 27: Señal h_vidon

En la figura 27, la señal superior es **hsync**, y la inferior, **h_vidon**.
Podemos observar cómo **h_vidon** sube 1917 ns (siendo el número sin errores manuales 1920 ns) después que **hsync**, indicando que sube 48 clocks después. Luego, observamos cómo **h_vidon** baja 643 ns antes, siendo el número correcto 640ns. Esto equivale a 16 clocks antes.
Concluimos entonces que **h_vidon** también cumple con lo establecido por la figura 19.