Laboratorio 8:

Diseño con Block RAM

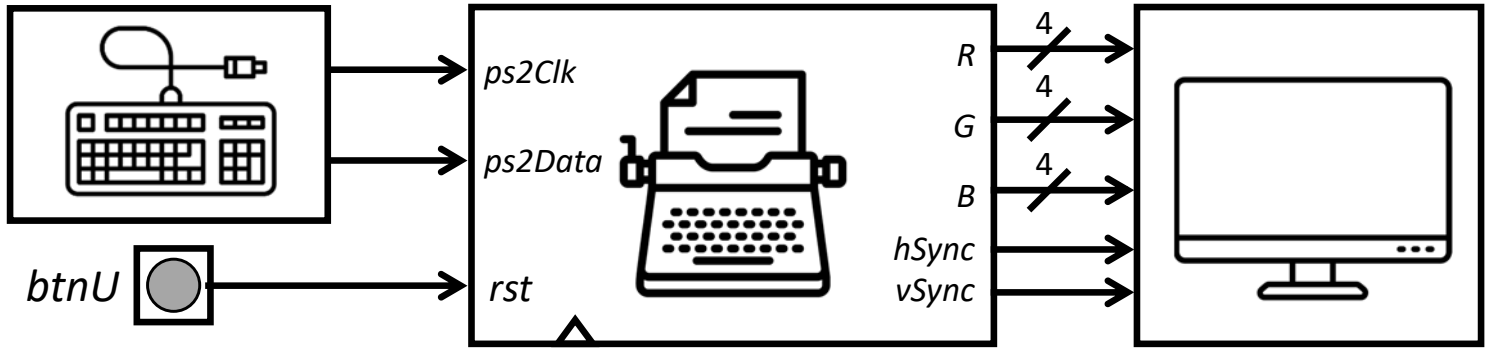
interfaces alfanuméricos de vídeo

Diseño automático de sistemas

José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Presentación

- Diseñar un terminal alfanumérico que visualice en una pantalla VGA los caracteres recibidos desde un teclado con interfaz PS/2.
  - Los caracteres se dispondrán comenzando de arriba a abajo, de izquierda a derecha.
    - Convirtiendo los scancodes de presión al correspondiente código ASCII y activando durante un ciclo la señal de dataRdy del vgaTextInterface.
    - Ignorando el código y el scancode de depresión.
  - Realizará algunas acciones especiales tras pulsar las siguientes teclas:
    - ESC: activará durante 1 ciclo la señal clear del vgaTextInterface
    - ENTER: Los siguientes caracteres que reciba se visualizarán al comienzo de la línea siguiente.
    - MAYUSCULAS: Mientras esté pulsada y convertirá los scancodes de presión recibidos al correspondiente código ASCII en mayúscula
    - BLOQ-MAYUS: Tras su pulsación, convertirá los scancodes de presión recibidos al correspondiente código ASCII en mayúscula. Tras una nueva pulsación a minúscula.



J.M. Mendías
2024

SRAM on-chip

Block RAM (i)

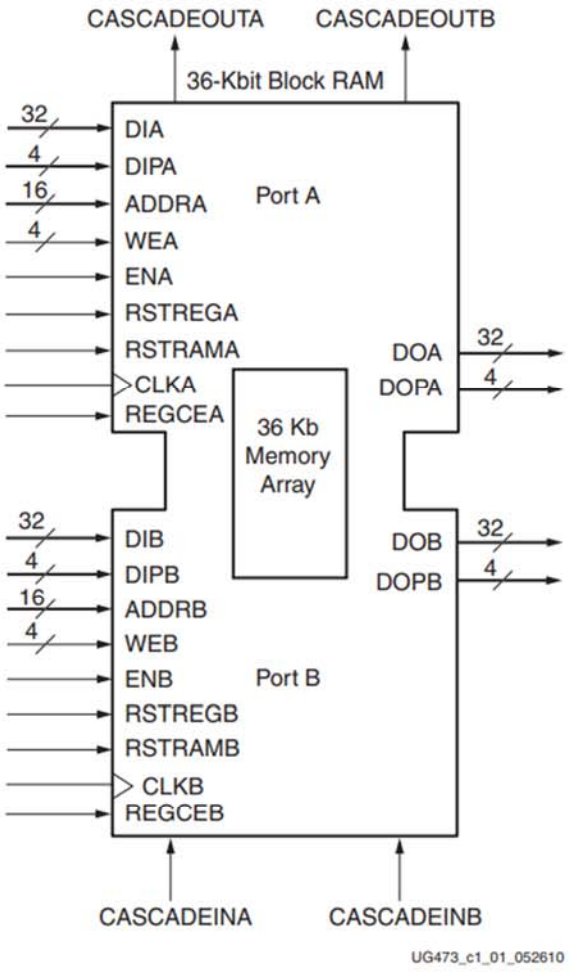
- AMD incorpora en sus FPGA bloques prefabricados de SRAM:
  - Lectura y escritura síncrona.
    - Latencia de escritura: 1 ciclo.
    - Latencia de lectura: 1 o 2 ciclos.
  - Inicializables durante start-up.
  - Buses de datos de entrada y salida separados.
  - Capaces de funcionar en 3 modos:
    - *Single port*: bus de dirección único (simultáneamente lee y escribe una misma celda).
    - *Simple dual port*: bus de dirección doble (simultáneamente lee una celda y escribe otra).
    - *True dual port*: bus de dirección doble (simultáneamente lee y escribe dos celdas).
  - Cada puerto puede presentar organizaciones lógicas diferentes con distinta relación de anchura/profundidad.
  - Pueden incorporarse al diseño mediante:
    - Inferencia a partir de construcciones VHDL.
    - Instanciación como componente generado usando un Core Generator.

J.M. Mendias
2024

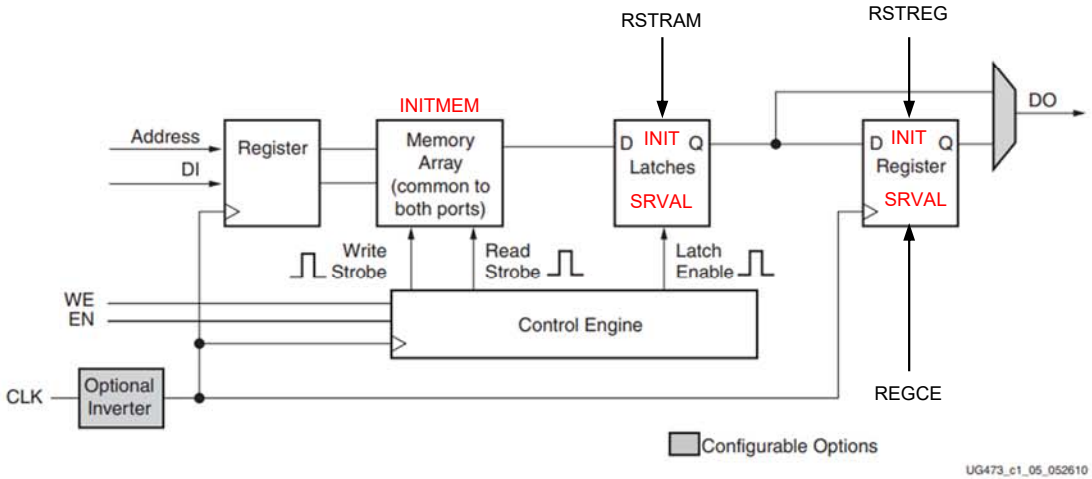
SRAM on-chip

Block RAM (ii)

- En las FPGA 7 Series se organizan físicamente en bloques de 36K×1b



Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices[2]	Block RAM Blocks[3]		
		Slices[1]	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)
XC7A12T	12,800	2,000	171	40	40	20	720
XC7A15T	16,640	2,600	200	45	50	25	900
XC7A25T	23,360	3,650	313	80	90	45	1,620
XC7A35T	33,280	5,200	400	90	100	50	1,800
XC7A50T	52,160	8,150	600	120	150	75	2,700
XC7A75T	75,520	11,800	892	180	210	105	3,780
XC7A100T	101,440	15,850	1,188	240	270	135	4,860
XC7A200T	215,360	33,650	2,888	740	730	365	13,140



J.M. Mendías
2024

Block RAM

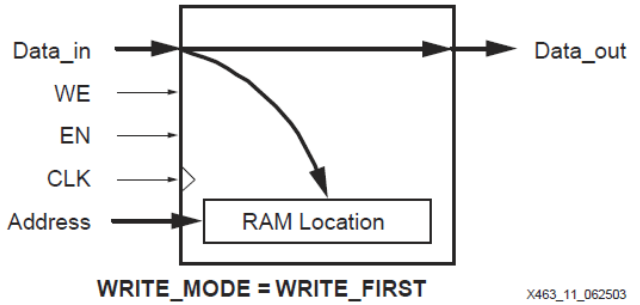
modos de funcionamiento (i)

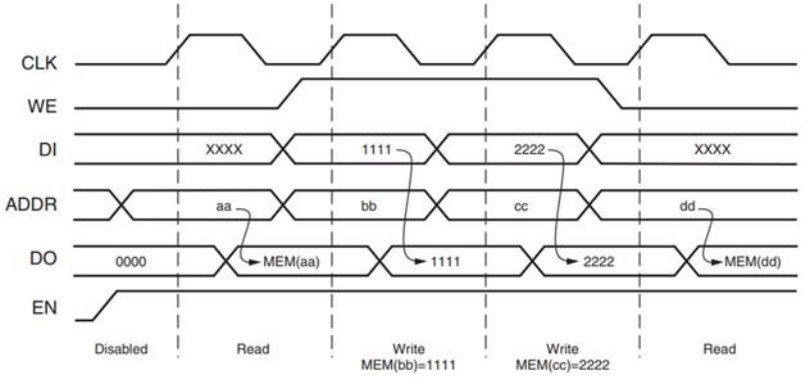
	EN	RST	WE	CLK	DO	Memoria
Tras configuración	X	X	X	X	INIT	INITMEM
RAM desabilitada	0	X	X	X	No cambia	No cambia
Reset síncrono durante lectura	1	1	0	↑	SRVAL	No cambia
Reset síncrono durante escritura	1	1	1	↑	SRVAL	RAM(ADDR) ← DI
Lectura sin escritura	1	0	0	↑	RAM(ADDR)	No cambia
Escritura sin lectura modo de escritura: NO_CHANGE	1	0	1	↑	No cambia	RAM (ADDR) ← DI
Escritura con lectura simultánea modo de escritura: READ_FIRST					RAM(ADDR)	
Escritura con lectura simultánea modo de escritura: WRITE_FIRST					DI	

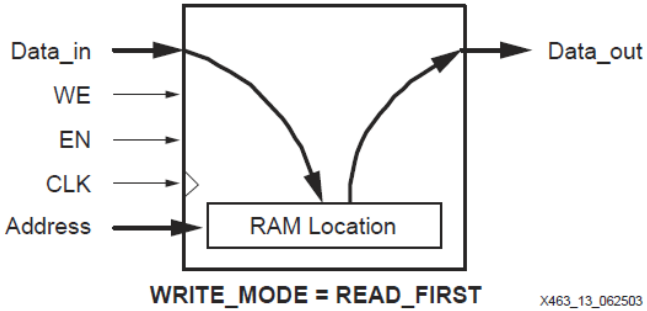
- Las memorias con doble puerto, tienen conflicto cuando:
  - Se escribe por cada puerto un valor distinto en la misma posición.

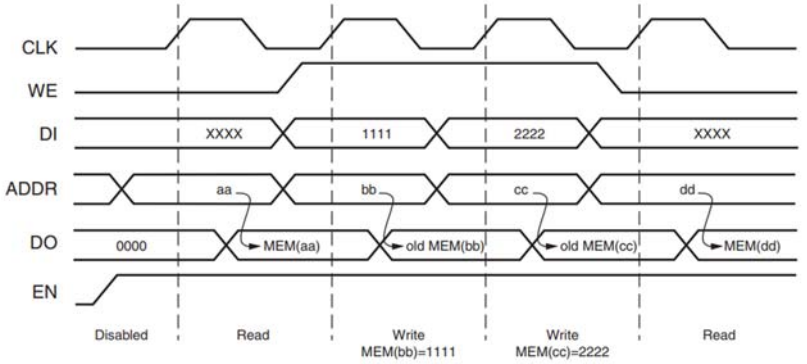
Block RAM

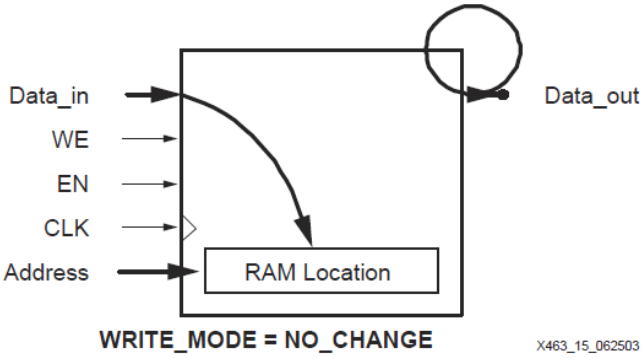
modos de funcionamiento (ii)

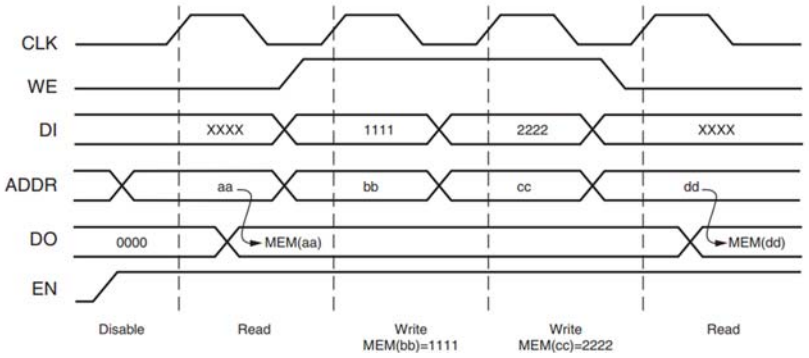












J.M. Mendías
2024

Inferencia de Block RAM

single port RAM: modo WRITE_FIRST

Especificación de una memoria 1K×8b
de tipo block RAM en modo **WRITE_FIRST**

```
architecture ...;
  signal en, we : std_logic;
  signal addr: std_logic_vector(9 downto 0);
  signal di, do: std_logic_vector(7 downto 0);
  type ramType is array (0 to 2**10-1) of std_logic_vector(7 downto 0);
  signal ram : ramType;
  ...
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if en='1' then
        if we='1' then
          ram( to_integer( unsigned( addr ) ) ) <= di;
          do <= di;
        else
          do <= ram( to_integer( unsigned( addr ) ) );
        end if;
      end if;
    end if;
  end process;
  ...
end;
```

profundidad: 1K

anchura: 8b

capacitación opcional

el registro **do** *se actualiza con el dato que entra (el mismo que se escribe)*

Latencia de lectura 1 ciclo:
El dato está disponible en **do** *1 ciclo después del cambio de* **en/addr**

Latencia de escritura 1 ciclo:
El dato se escribe en memoria 1 ciclo después del cambio de **en/we/address**

J.M. Mendias
2024

Inferencia de Block RAM

single port RAM: modo READ_FIRST

Especificación de una memoria 16K×8b de tipo block RAM en modo **READ_FIRST**

```
architecture ...;
  signal en, we : std_logic;
  signal addr: std_logic_vector(13 downto 0);
  signal di, do: std_logic_vector(7 downto 0);
  type ramType is array (0 to 2**14-1) of std_logic_vector(7 downto 0);
  signal ram : ramType;
  ...
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if en='1' then
        if we='1' then
          ram( to_integer( unsigned( addr ) ) ) <= di;
        end if;
        do <= ram( to_integer( unsigned( addr ) ) );
      end if;
    end if;
  end process;
  ...
end;
```

profundidad: 16K

anchura: 8b

capacitación opcional

el registro **do** *se actualiza siempre con el dato leído de memoria (diferente del escrito)*

Latencia de lectura/escritura 1 ciclo

Inferencia de Block RAM

single port RAM: modo NO_CHANGE

Especificación de una memoria 1K×16b de tipo block RAM en modo **NO_CHANGE**

```
architecture ...;
  signal en, we : std_logic;
  signal addr: std_logic_vector(9 downto 0);
  signal di, do: std_logic_vector(15 downto 0);
  type ramType is array (0 to 2**10-1) of std_logic_vector(15 downto 0);
  signal ram : ramType;
  ...
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if en='1' then
        if we='1' then
          ram( to_integer( unsigned( addr ) ) ) <= di;
        else
          do <= ram( to_integer( unsigned( addr ) ) );
        end if;
      end if;
    end if;
  end process;
  ...
end;
```

profundidad: 1K

anchura: 16b

capacitación opcional

el registro **do** *no se actualiza en escrituras (solo en lecturas)*

Latencia de lectura/escritura 1 ciclo

J.M. Mendias
2024

Inferencia de Block RAM

single port RAM: inicialización (i)

Especificación de una memoria 16K×8b de tipo block RAM en modo READ_FIRST

```
architecture ...;
  signal en, we : std_logic;
  signal addr: std_logic_vector(13 downto 0);
  signal di, do: std_logic_vector(7 downto 0);
  type ramType is array (0 to 2**14-1) of std_logic_vector(7 downto 0);
  signal ram : ramType := { X"0f", X"23", ... };
  ...
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if en='1' then
        if we='1' then
          ram( to_integer( unsigned( addr ) ) ) <= di;
        end if;
        do <= ram( to_integer( unsigned( addr ) ) );
      end if;
    end if;
  end process;
  ...
end;
```

opcionalmente puede indicarse el valor inicial de la memoria, la inicialización se realizará solo durante start-up.

Inferencia de Block RAM

single port RAM: inicialización (i)

Especificación de una memoria 16K×8b de tipo block RAM en modo READ_FIRST

```
architecture ...;
  signal en, we : std_logic;
  signal addr: std_logic_vector(13 downto 0);
  signal di: std_logic_vector(7 downto 0);
  signal do: std_logic_vector(7 downto 0) := (others => '0');
  type ramType is array (0 to 2**14-1) of std_logic_vector(7 downto 0);
  signal ram : ramType;
  ...
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if en='1' then
        if we='1' then
          ram( to_integer( unsigned( addr ) ) ) <= di;
        end if;
        if rst = '1' then
          do <= (others => '0' );
        else
          do <= ram( to_integer( unsigned( addr ) ) );
        end if;
      end if;
    end if;
  end process;
  ...
end;
```

el registro **do** *opcionalmente puede inicializarse durante start-up y síncronamente*

J.M. Mendías
2024

Inferencia de Block RAM

simple dual port RAM

Especificación de una memoria 1K×8b de tipo block RAM **simple dual port**

```
architecture ...;
  signal ena, enb, wea : std_logic;
  signal addra, addrb: std_logic_vector(9 downto 0);
  signal dia, dob: std_logic_vector(7 downto 0);
  type ramType is array (0 to 2**10-1) of std_logic_vector(7 downto 0);
  signal ram : ramType;
  ...
begin
  process (clk)
  begin
    if rising_edge(clk) then
      if ena='1' then
        if wea='1' then
          ram( to_integer( unsigned( addra ) ) ) <= dia;
        end if;
      end if;
      if enb='1' then
        dob <= ram( to_integer( unsigned( addrb ) ) );
      end if;
    end if;
  end process;
...
end;
```

se escribe en una dirección y se lee de otra

J.M. Mendías
2024

Inferencia de Block RAM

true dual port RAM

Especificación de una memoria 1K×8b de tipo block RAM **true dual port**

```
architecture ...;
  signal ena, enb, wea : std_logic;
  signal addra, addrb: std_logic_vector(9 downto 0);
  signal dia, dob: std_logic_vector(7 downto 0);
  type ramType is array (0 to 2**10-1) of std_logic_vector(7 downto 0);
  signal ram : ramType;
  ...
begin
 process (clk)
  begin
    if rising_edge(clk) then
      if ena='1' then
        if wea='1' then
          ram( to_integer( unsigned( addra ) ) ) <= dia;
        end if;
        doa <= ram( to_integer( unsigned( addra ) ) );
      if enb='1' then
        if web='1' then
          ram( to_integer( unsigned( addrb ) ) ) <= dib;
        end if;
        dob <= ram( to_integer( unsigned( addrb ) ) );
      end if;
    end if;
  end process;
  ...
end;
```

se lee y escribe (modo READ_FIRST) en una dirección y se lee y escribe (modo READ_FIRST) de otra

J.M. Mendías
2024

Inferencia de Block RAM

single port ROM

Especificación de una ROM 1K×8b con lectura síncrona implementada con block RAM

```
architecture ...;
  signal en : std_logic;
  signal addr: std_logic_vector (9 downto 0);
  signal do: std_logic_vector (7 downto 0);
  type romType is array (0 to 2**10-1) of std_logic_vector (7 downto 0);
  signal rom : romType := ( X"0f", X"23", ... );
  ...
begin
  ...
  process (clk)
  begin
    if rising_edge(clk) then
      if en='1' then
        do <= rom( to_integer( unsigned( addr ) ) ) ;
      end if;
    end if;
  end process;
  ...
end;
```

debe indicarse el valor inicial de la memoria

capacitación opcional

nunca se escribe: infiere una ROM

Latencia de lectura 1 ciclo

J.M. Mendias
2024

SRAM on-chip

Distributed RAM

- Adicionalmente, Vivado puede configurar las LUTRAM de los CLB para que se comporten como SRAM:
  - Escritura síncrona y lectura síncrona o asíncrona.
  - Inicializables durante start-up.
  - Con organización física variable.
  - Pero su uso consume recursos que pudieran dedicarse a lógica.
  - Se incorporan al diseño solo mediante inferencia a partir de construcciones VHDL.

- Cuando la SRAM se infiere, puede controlarse cómo se implementa:
  - Según la plantilla VHDL usada:
    - Si la lectura es asíncrona, se implementa como distributed RAM.
    - Si la lectura es síncrona, según la plantilla, se implementa como block o distributed RAM.
  - Si la plantilla VHDL usada es aplicable a ambos modelos:
    - Puede forzarse usando atributo ram_style = { block | distributed }

```
type ramType is array (0 to ...) of std_logic_vector(... downto 0);
signal ram : ramType;
attribute ram_style of ram : signal is "block";
```

Inferencia de Distributed RAM

RAM

- Si la lectura de la memoria es asíncrona, se implementa en LUTRAM

Especificación de una RAM de lectura asíncrona

```
architecture ...;
  ...
  signal addr: std_logic_vector(... downto 0);
  signal di, do: std_logic_vector(... downto 0);
  type ramType is array (0 to ...) of std_logic_vector(... downto 0);
  signal ram : ramType := { X"0f", X"23", ... };
begin
  ...
  process (clk)
  begin
    if rising_edge(clk) then
      if we='1' then
        ram( to_integer( unsigned( addr ) ) ) <= di;
      end if;
    end if;
  end process;
  do <= ram( to_integer( unsigned( addr ) ) ) ;
  ...
end;
```

Opcionalmente puede indicarse el valor inicial de la memoria

Latencia de escritura 1 ciclo:
El dato se escribe en memoria 1 ciclo después del cambio de **we/addr**

Sin latencia de lectura:
El dato está disponible en **do** *el mismo ciclo en que cambia* **addr**

J.M. Mendías
2024

Inferencia de Distributed RAM

ROM

- Además, si no se escribe, infiere ROM y se implementa en LUTRAM

Especificación de una ROM de lectura asíncrona

```
architecture ...;
  ...
  signal addr: std_logic_vector(... downto 0);
  signal di, do: std_logic_vector(... downto 0);
  type ramType is array (0 to ...) of std_logic_vector(... downto 0);
  signal ram : romType := { X"0f", X"23", ... };
begin
  ...
  do <= rom( to_integer( unsigned( addr ) ) ) ;
  ...
end;
```

debe indicarse el valor inicial de la memoria

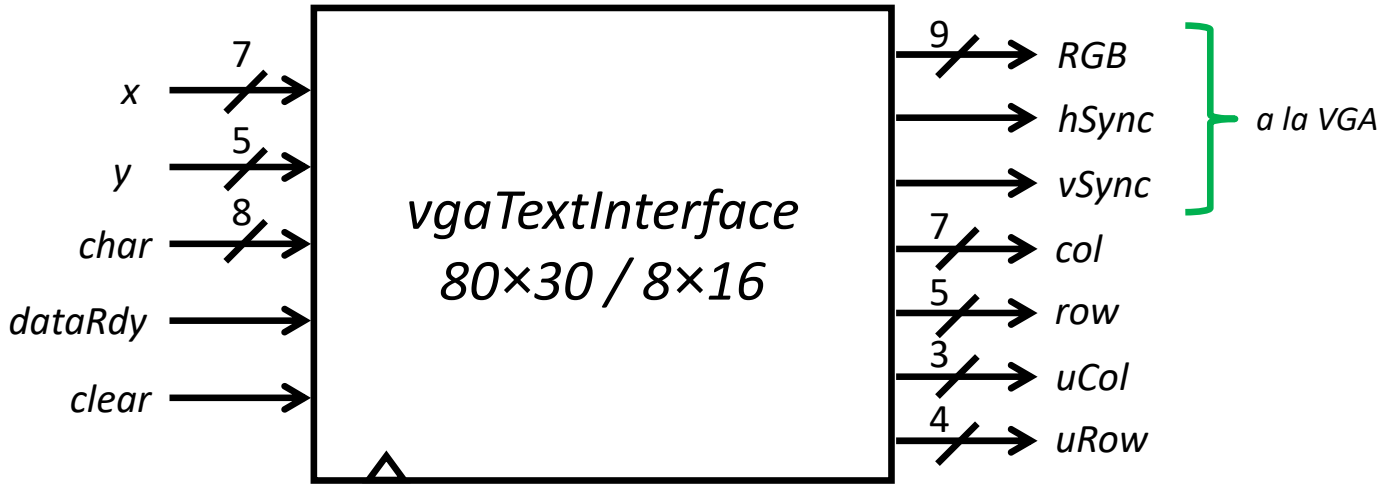
Sin latencia de lectura:
El dato está disponible en **do** *el mismo ciclo en que cambia* **addr**

J.M. Mendías
2024

Interfaz alfanumérico de vídeo

presentación

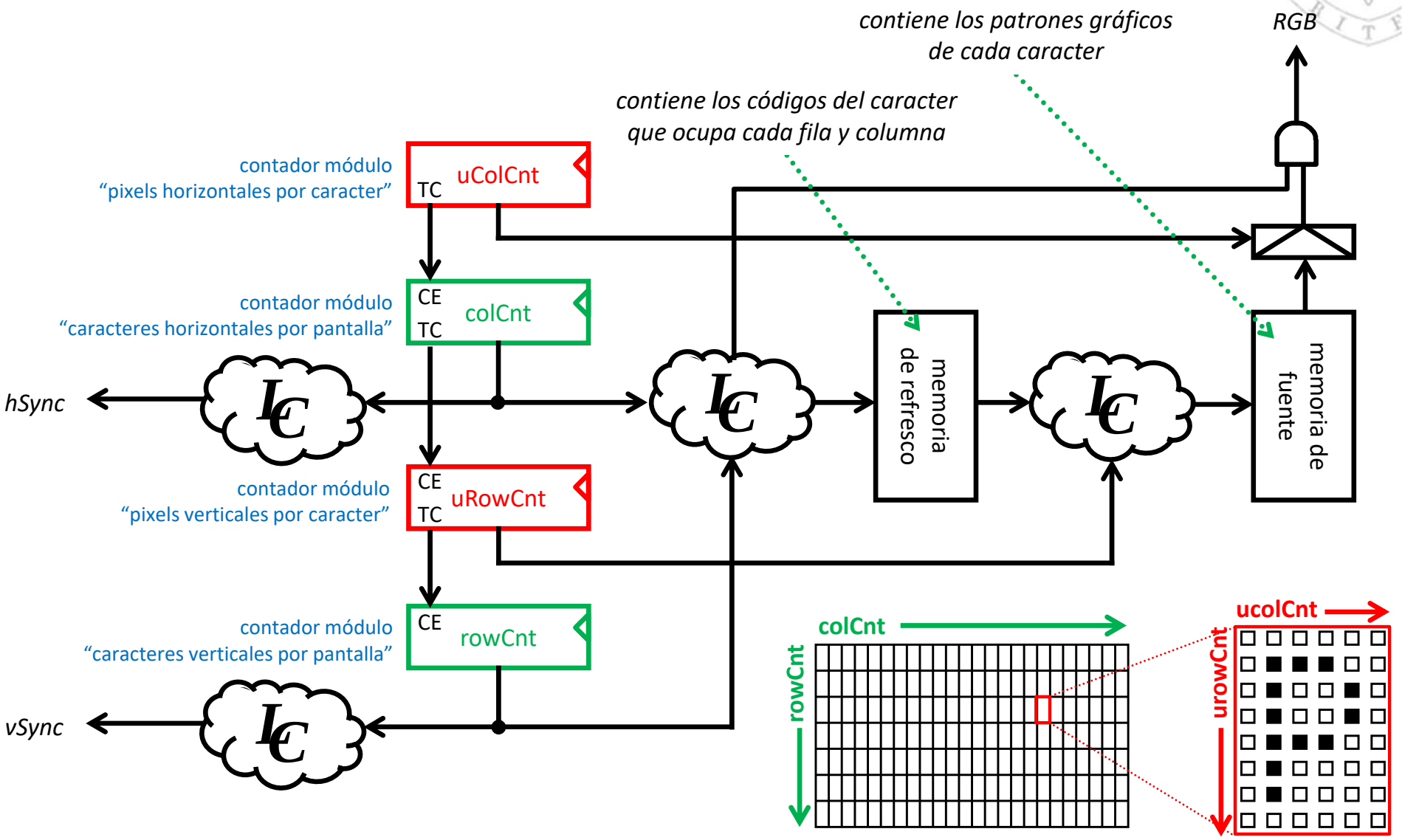
- Diseñar un interfaz monocromo alfanumérico VGA:
  - Capaz de visualizar 80×30 caracteres diferentes de 8×16 pixeles. Para ello dispondrá:
    - Una memoria de refresco (RAM) que almacena el código ASCII de cada caracter.
    - Una memoria de fuente (ROM) que almacena los 256 mapas de bits de 8×16 correspondientes a cada caracter.
  - Estará controlado por 2 señales de tipo strobe:
    - dataRdy: almacenará en la posición (x,y) de la memoria de refresco el código ASCII char.
    - clear: borrará la pantalla, escribiendo 0 en todas las celdas de la memoria de refresco.
  - Por (row,col) y (uRow,uCol) indicará respectivamente las coordenadas del caracter y del pixel del mapa de bits que se está refrescando en cada momento.
    - Por si se desea superponer gráficos "al vuelo".



J.M. Mendías
2024

Interfaz alfanumérico de vídeo

estructura

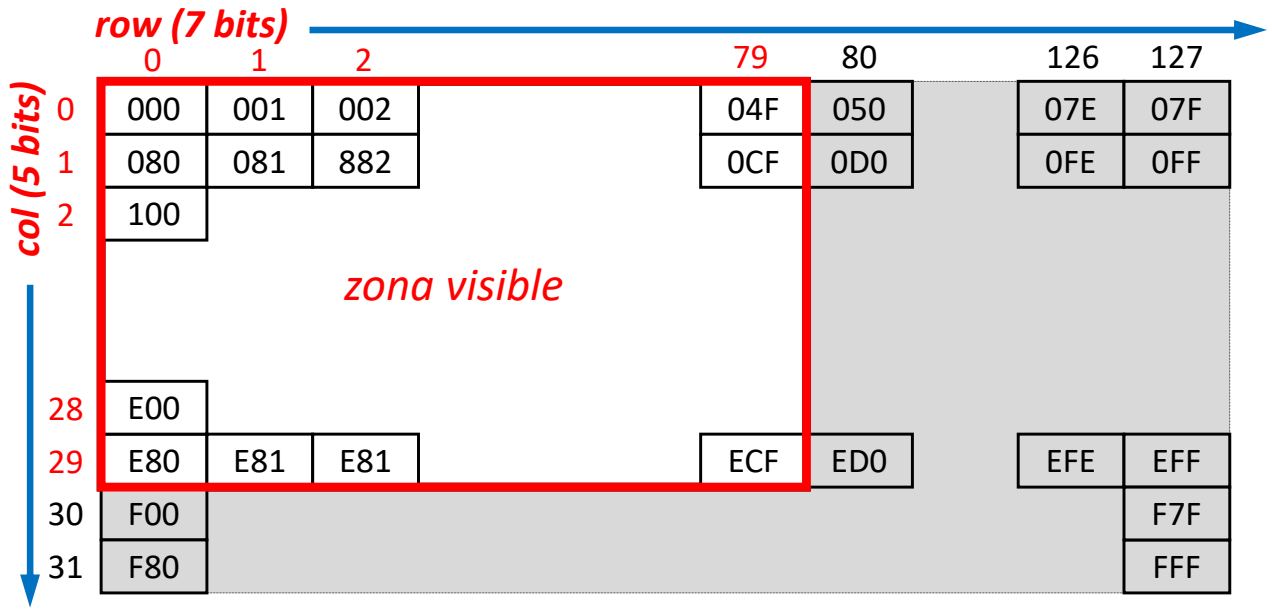


J.M. Mendías
2024

Interfaz alfanumérico de vídeo

memoria de refresco

- En una RAM se almacenan los códigos de cada carácter a visualizar:
  - La dirección de memoria se obtendrá concatenando las coordenadas de la posición.
    - Parte de la memoria queda desaprovechada, a costa de simplificar el acceso.
  - La memoria de refresco completa requiere de 2^{7+5} B = 4 KB = 32 Kb
    - Es proyectable sobre un único Block RAM.

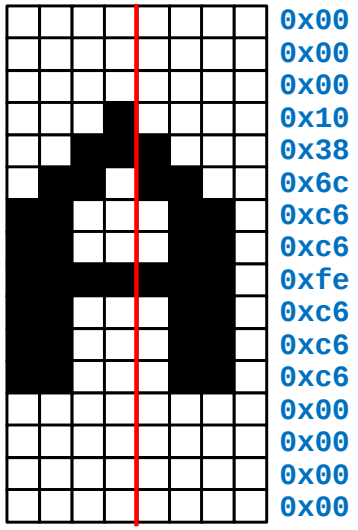


Interfaz alfanumérico de vídeo

memoria de fuente

- En una ROM se almacena una fuente tipo *serif* de 8×16 px/carácter
  - Para cada carácter define su mapa de bits como una matriz de 8×16 bits almacenada en 16 direcciones consecutivas.
    - Los caracteres se almacenan en orden ascendente de su código ASCII.
  - En cada mapa representa con 1 los *foreground pixels* y con 0 los *background pixels*, almacenándolos de izquierda a derecha y de arriba a abajo.
  - La fuente completa requiere de 256·16B = 4 KB = 32 Kb
    - Es proyectable sobre un único Block RAM

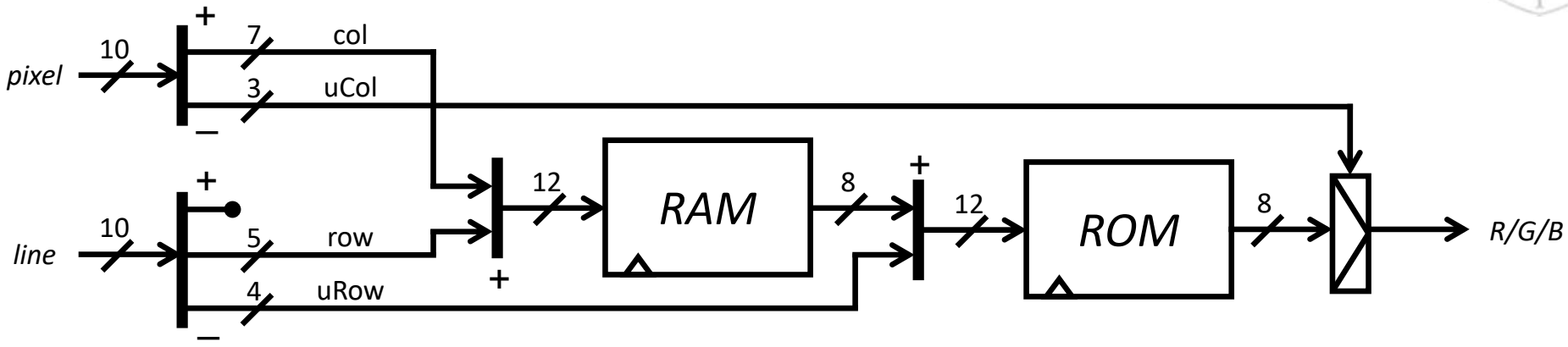
```
type romType is array (0 to 2**12-1) of std_logic_vector (7 downto 0);
signal rom : romType := (
  ...
  X"00", X"00", X"00", X"10", X"38", X"6c", X"c6", X"c6",
  X"fe", X"c6", X"c6", X"c6", X"00", X"00", X"00", X"00",
  ...
);
```



J.M. Mendías
2024

Interfaz alfanumérico de vídeo

procedimiento de refresco (i)



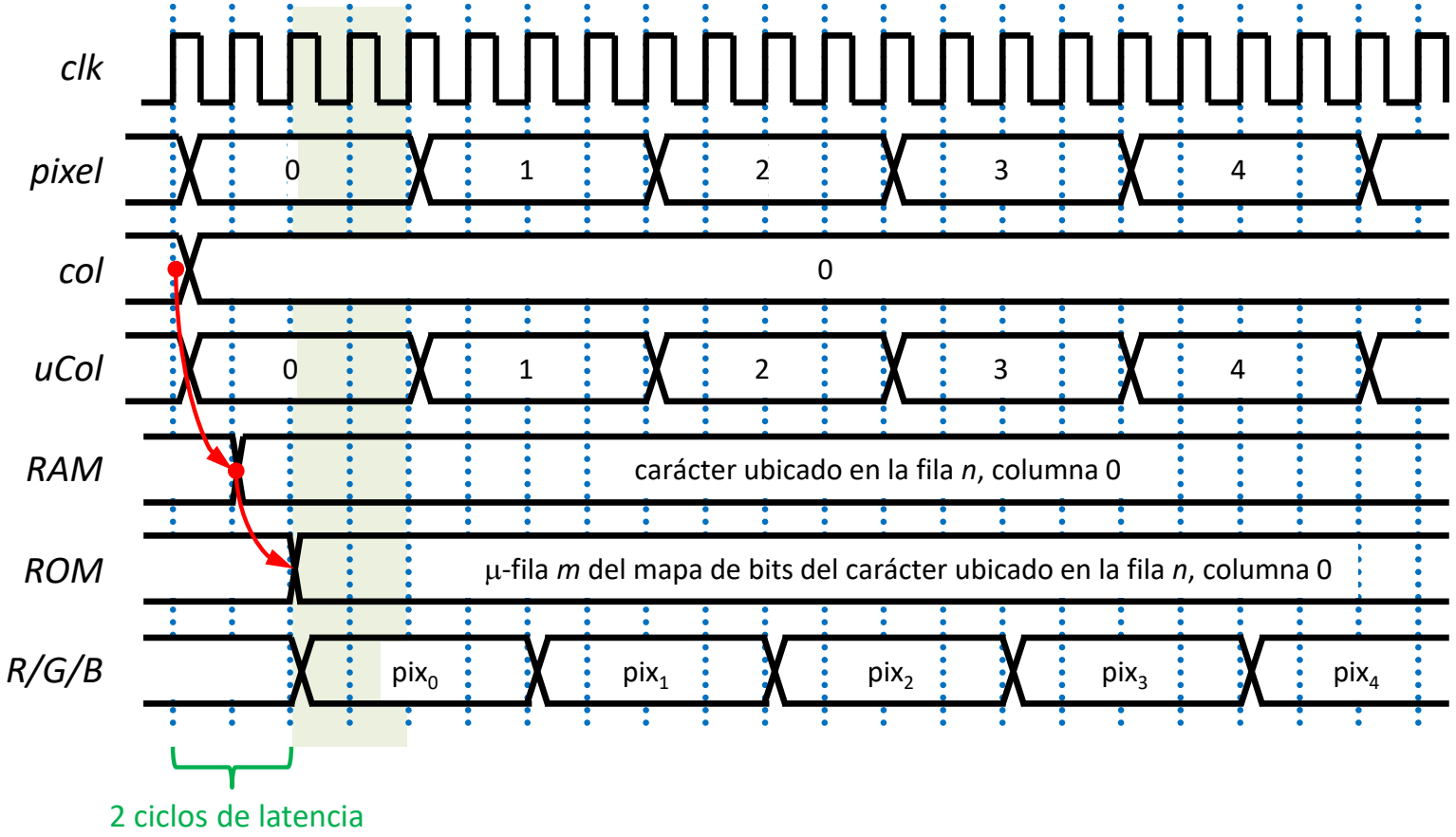
- Usará un módulo vgaInterface para refrescar la pantalla
  - A partir de la salida pixel se obtiene la columna de la pantalla que se está refrescando y la μ-columna del mapa de bits que se debe refrescar.
  - A partir de la salida line se obtiene la fila de la pantalla que se está refrescando la μ-fila dentro del mapa de bits que se debe refrescar.
  - La RAM se direcciona con la columna y fila para obtener el código del carácter.
  - La ROM se direcciona con el código de carácter y la μ-fila para obtener la fila de píxeles del mapa de bits correspondiente.
  - La μ-columna se utiliza para seleccionar uno de esos píxeles y mandarlos al interfaz.

J.M. Mendías
2024

Interfaz alfanumérico de vídeo

procedimiento de refresco (ii)

- Dado que la RAM y la ROM tienen lectura síncrona:
  - Se acumula una latencia de 2 ciclos entre ellos cambios de pixel/line y R/G/B
  - No hay problema porque el vgaRefresher tiene registradas las salidas y la latencia es inferior a 4 (ratio 100 MHz / 25 MHz)

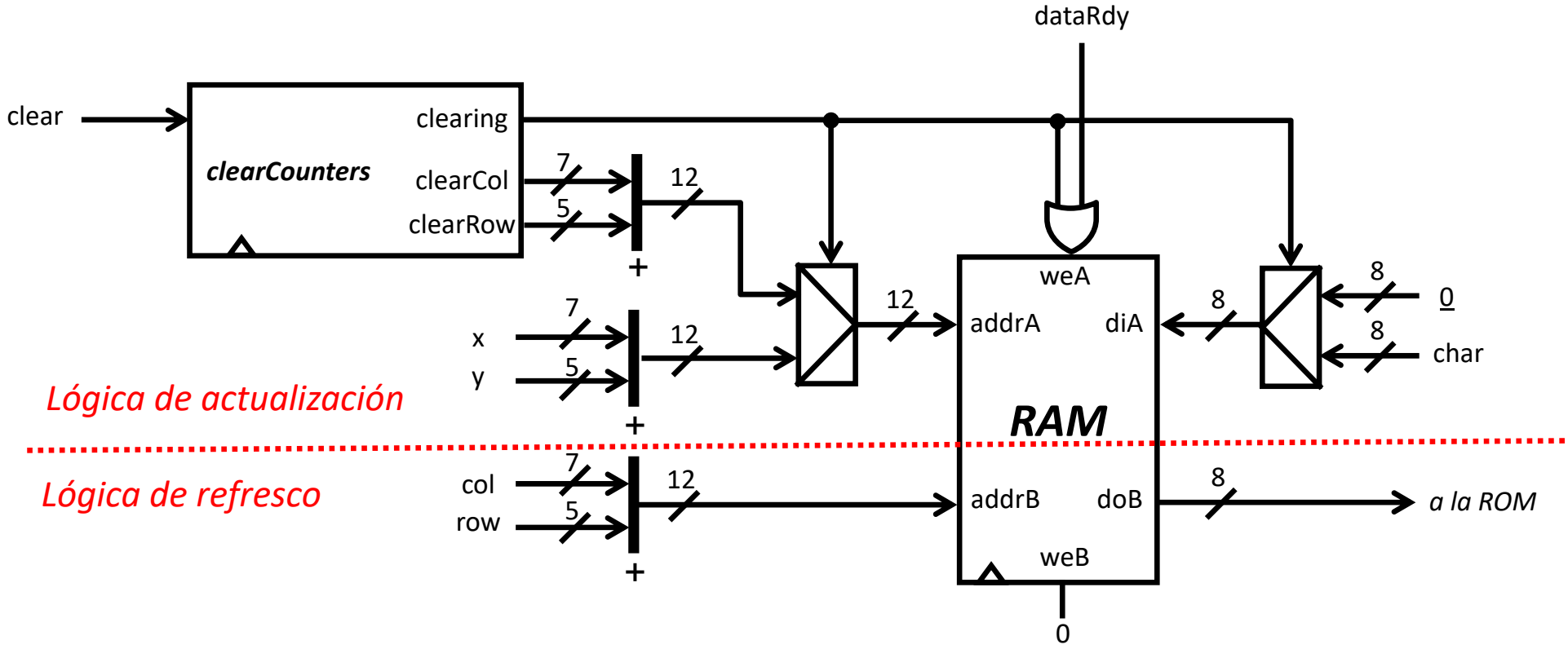


J.M. Mendías 2024

Interfaz alfanumérico de vídeo

procedimiento de actualización

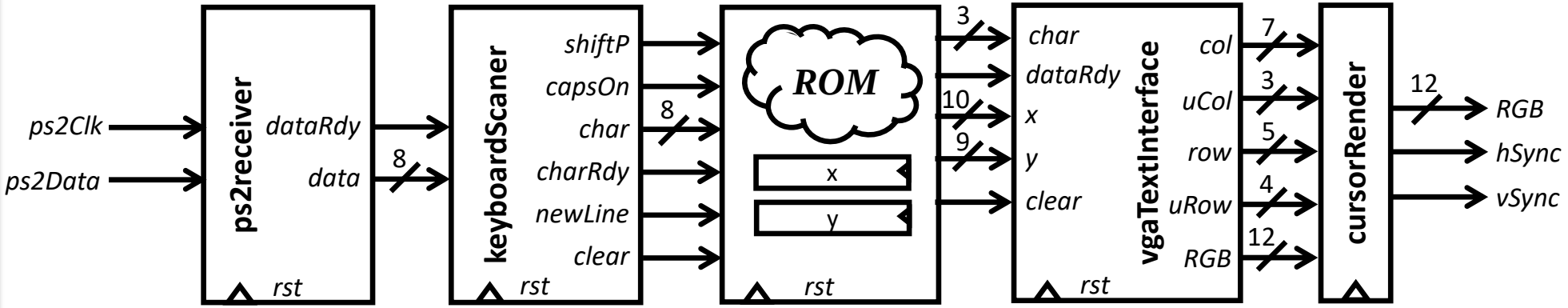
- La RAM será de tipo simple dual port, ya que al aislar la lógica de actualización de la de refresco se simplifica el diseño.
  - Un puerto dedicado a leer el carácter a refrescar y otro a escribir/borrar un caracter.
  - Al ser un interfaz de refresco, el modo de la Block RAM podrá ser cualquiera.
  - Para borrar la pantalla se barrerá la memoria escribiendo 0 en todas sus direcciones.



Diseño principal

esquema RTL

- El diseño constará de:
  - Un ps2receiver conectado a un escáner de teclado (FSM con flags).
  - Una ROM con lectura asíncrona (Distributed RAM)
    - Convierte scancodes en código ASCII,
    - Diferentes según se haya pulsado MAYUSCULAS o BLOQ-MAYUSCULAS.
  - Dos registros que almacenen la posición (x,y) de pantalla en donde escribir el dato:
    - Inicialmente x e y valdrán 0. A cada tecla pulsada, x deberá incrementarse.
    - Cada vez que se complete una línea, x volverá a 0 e y deberá incrementarse.
    - Una vez completada una pantalla, x e y volverán a 0.
    - Las pulsaciones de ESC y ENTER deberán cambiar convenientemente x e y.
  - Un vgaTextInterface.
  - Un módulo combinacional que visualice "al vuelo" un cursor en la posición (x,y).



J.M. Mendías
2024

Tareas

1. En la carpeta **DAS/source** crear las carpeta **lab8**.
2. Descargar de la Web los ficheros en:
   - **common:** **vgaTextInterface.vhd**
   - **lab8:** **lab8.vhd** y **lab8.xdc**
3. Completar **common.vhd** con la declaración del nuevo componente.
4. Completar el código omitido en los ficheros:
   - **vgaTextInterface.vhd** y **lab8.vhd**
5. En la carpeta **DAS/projects** crear el proyecto **lab8**.
6. Incluir en el proyecto (sin copiarlos) los siguientes fuentes:
   - **common.vhd**, **synchronizer.vhd**, **edgedetector.vhd**, **ps2Receiver.vhd**, **vgaRefresher.vhd**, **vgaTextInterface.vhd**, **lab8.vhd** y **lab8.xdc**
7. Sintetizar, implementar y generar el fichero de configuración.
8. Conectar el teclado y el monitor a la placa y encenderla.
9. Volcar el fichero de configuración **lab8.bit**

Acerca de *Creative Commons*

- Licencia CC (Creative Commons)

  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

Reconocimiento (*Attribution*):
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No comercial (*Non commercial*):
La explotación de la obra queda limitada a usos no comerciales.

Compartir igual (*Share alike*):
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/