



Laboratorio 7:

Diseño con SRAM on-chip

uso de Block RAM

Diseño automático de sistemas

José Manuel Mendías Cuadros

*Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid*

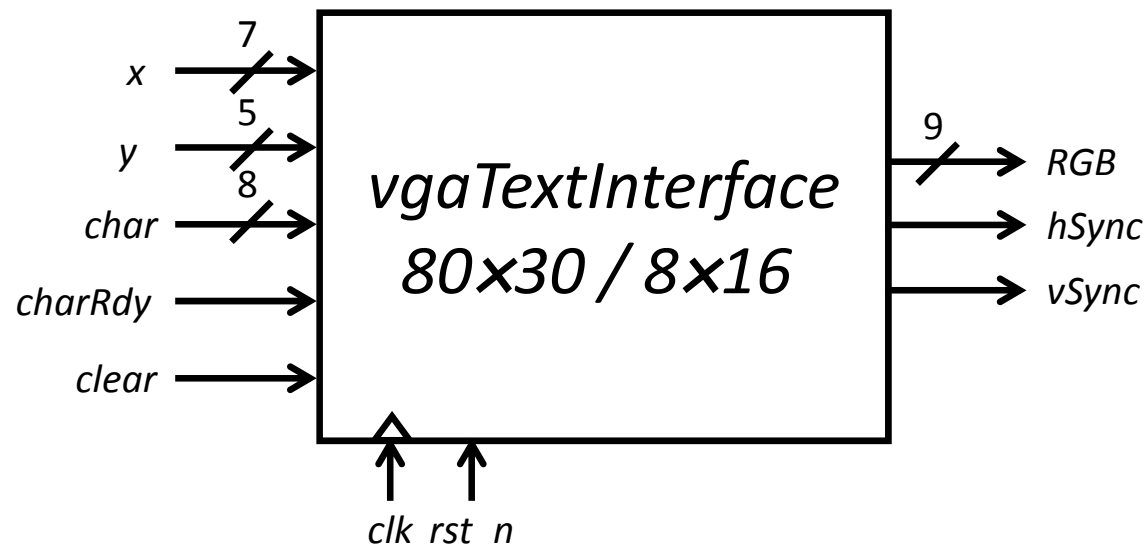


Presentación



■ Diseñar un interfaz alfanumérico VGA:

- Podrá visualizar hasta **256 caracteres** diferentes en una matriz de **80×30 caracteres** de **8×16 pixeles** cada uno. Para ello dispondrá:
 - Una **memoria de refresco (RAM)** que almacena el **código ASCII** de los 80×30 caracteres que están siendo visualizados.
 - Una **memoria de carácter (ROM)** que almacena los 256 **mapas de bits** de 8×16 de cada uno de los caracteres visualizables.
- Estará controlado por 2 señales de tipo strobe que no se activarán simultáneamente
 - **CharRdy**: visualizará en la posición (x,y) del monitor el carácter de código ASCII **char**.
 - **Clear**: borrará la pantalla completa.

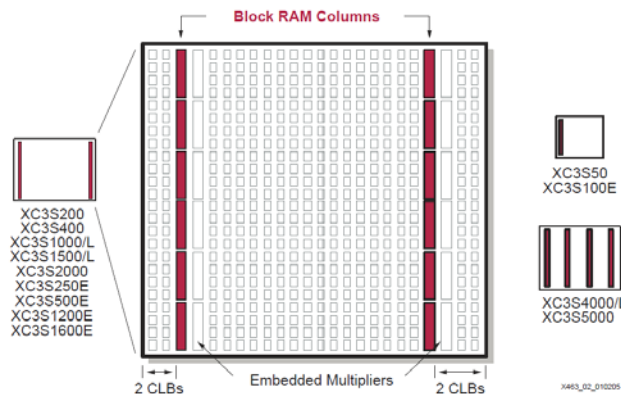


SRAM on-chip

Block RAM



- Xilinx incorpora en sus FPGA bloques prefabricados de **SRAM**:
 - Lectura y escritura síncrona.
 - Inicializables durante start-up.
 - **Doble puerto** (configurable como simple) con entrada y salida de datos separados.
 - Organizados físicamente en bloques de 1 bit
 - Pueden presentar hasta **2 organizaciones lógicas distintas** (una por puerto) con distinta relación de anchura/profundidad de datos.
 - Pueden **incorporarse al diseño** mediante:
 - **Instanciación como componente** (puede usarse un Wizard del Core Generator)
 - **Inferencia** a partir de construcciones VHDL.



Device	RAM Columns	RAM Blocks Per Column	Total RAM Blocks	Total RAM Bits	Total RAM Kbits
XC3S50	1	4	4	73,728	72K
XC3S200	2	6	12	221,184	216K
XC3S400	2	8	16	294,912	288K
XC3S1000/L	2	12	24	442,368	432K
XC3S1500/L	2	16	32	589,824	576K
XC3S2000	2	20	40	737,280	720K
XC3S4000/L	4	24	96	1,769,472	1,728K
XC3S5000	4	26	104	1,916,928	1,872K

SRAM on-chip

Distributed RAM



- Adicionalmente, **Xilinx ISE** puede configurar parte de las LUT 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";
```

Block RAM

Pineado



- **Entrada de datos:**
 - $DI[\# : 0]$ / ($DIA[\# : 0]$, $DIB[\# : 0]$) – Bus/es de datos de entrada.
 - $DIP[\# : 0]$ / ($DIPA[\# : 0]$, $DIPB[\# : 0]$) – Bus/es de paridad de datos de entrada.
- **Salida de datos:**
 - $DO[\# : 0]$ / ($DOA[\# : 0]$, $DOB[\# : 0]$) – Bus/es de datos de salida.
 - $DOP[\# : 0]$ / ($DOPA[\# : 0]$, $DOPB[\# : 0]$) – Bus/es de paridad de datos de salida.
- **Entrada de direcciones:**
 - $ADDR[\# : 0]$ / ($ADDRA[\# : 0]$, $ADDRB[\# : 0]$) – Bus/es de direcciones.
- **Entradas de control (activas en alta):**
 - EN / (ENA , ENB) – Capacitación del módulo.
 - SSR / ($SSRA$, $SSRB$) – Inicialización síncrona de los latches de salida.
 - CLK / ($CLKA$, $CLKB$) – Reloj: las operaciones se hacen a flanco de subida.
 - WE / (WEA , WEB) – Capacitación de escritura.
- Los buses de paridad aparecen en configuraciones de más de 8 bits
 - Físicamente no tienen funcionalidad propia, por lo que puede ser usados libremente.

Block RAM

Parametrización



- **Organización de la memoria:**
 - Según el nombre de la instancia usada puede configurarse:
 - El **número de puertos** (simple/doble) de la RAM
 - La **organización lógica** de la RAM (relación anchura/profundidad) para cada puerto.
- **Contenidos iniciales** (cargados durante configuración, por defecto 0):
 - **INIT_xx** – Contenido inicial de la memoria de datos.
 - **INITP_xx** – Contenido inicial de la memoria de paridad.
 - **INIT / (INIT_A, INIT_B)** – Contenido inicial de los biestables de salida de datos.
- **Otros:**
 - **SRVAL / (SRVAL_A, SRVAL_B)** – Valor inicialización síncrona (tras activación de la línea SSR) de los biestables de salida de datos.
 - Por defecto 0
 - **WRITE_MODE** – Comportamiento de los biestables de salida de datos durante la escritura.
 - Los valores posibles son: WRITE_FIRST(valor por defecto), READ_FIRST, NO_CHANGE.



Block RAM

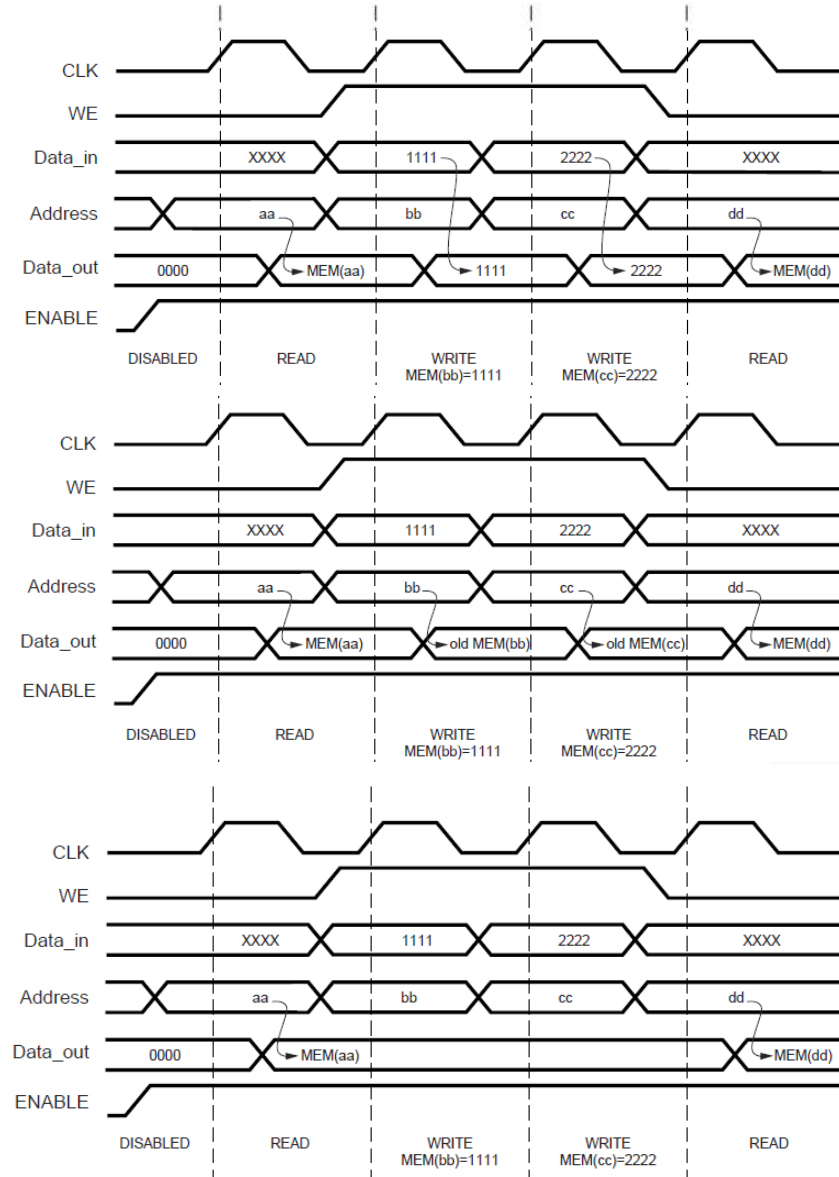
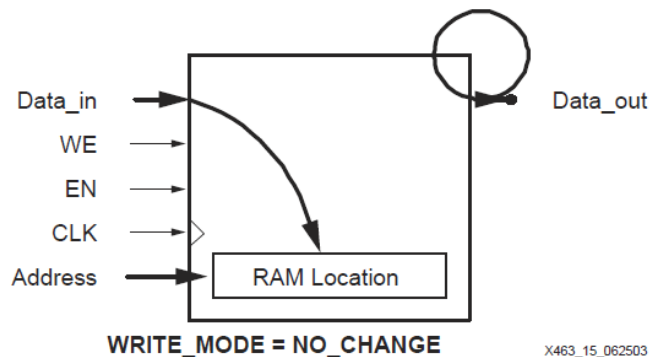
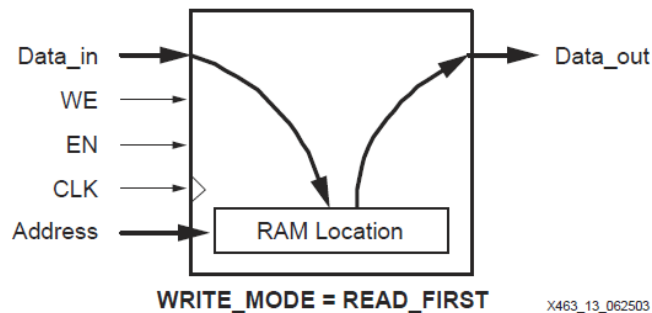
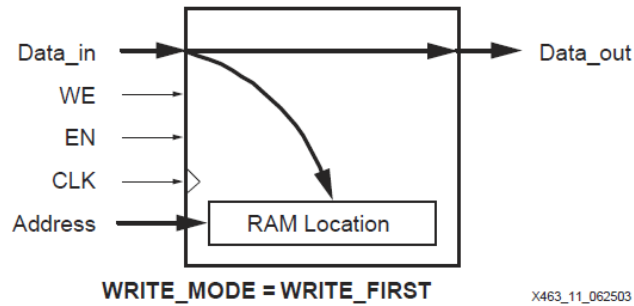
Modos de funcionamiento (i)

	EN	SSR	WE	CLK	DO/DOP	Memoria
Tras configuración	X	X	X	X	INIT	INIT_xx / INTP_xx
RAM deshabilitada	0	X	X	X	No cambia	No cambia
Reset síncrono	1	1	0	↑	SRVAL	No cambia
Reset síncrono durante escritura	1	1	1	↑	SRVAL	RAM(ADDR) ← DI/DIP
Lectura sin escritura	1	0	0	↑	RAM(ADDR)	No cambia
Escritura con lectura simultánea WRITE_MODE = WRITE_FIRST	1	0	1	↑	DI/DIP	RAM (ADDR) ← DI/DIP
Escritura con lectura simultánea WRITE_MODE = READ_FIRST					RAM(ADDR)	
Escritura con lectura simultánea WRITE_MODE = NO_CHANGE					No cambia	

- Las memorias con **doble puerto**, tienen **conflicto** cuando:
 - Se intenta escribir por cada puerto un valor distinto en la misma posición.
 - Se intenta escribir una posición por un puerto configurado en modo WRITE_FIRST o NO_CHANGE y leer la misma posición por el otro.

Block RAM

Modos de funcionamiento (ii)



Block RAM

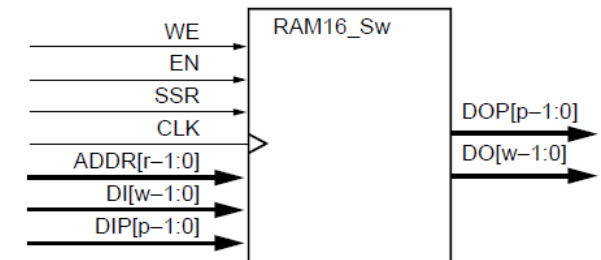
Organizaciones



- El **bloque primitivo** en la familia Spartan 3, es una **RAM de (16+2)K×1b**
 - Las distintas organizaciones lógicas que puede adoptar este bloque se encuentran en la biblioteca **UNISIM**, paquete **VCOMPONENTS**.

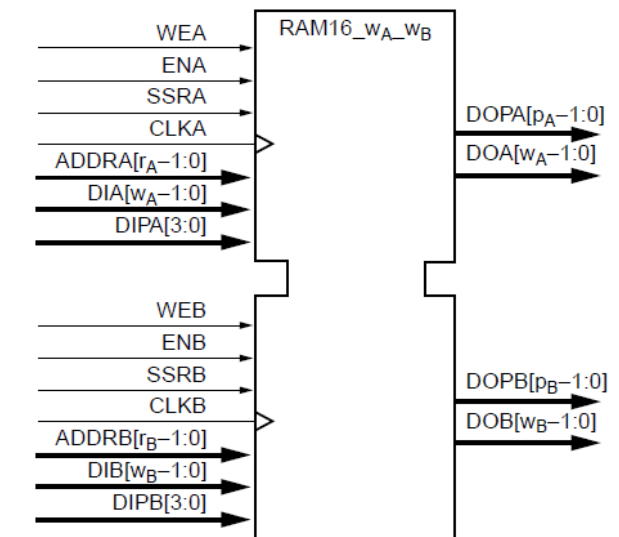
Organizaciones como RAM de simple puerto

Organization	Memory Depth	Data Width	Parity Width	DI/DO	DIP/DOP	ADDR	Single-Port Primitive	Total RAM Kbits
512x36	512	32	4	(31:0)	(3:0)	(8:0)	RAMB16_S36	18K
1Kx18	1024	16	2	(15:0)	(1:0)	(9:0)	RAMB16_S18	18K
2Kx9	2048	8	1	(7:0)	(0:0)	(10:0)	RAMB16_S9	18K
4Kx4	4096	4	-	(3:0)	-	(11:0)	RAMB16_S4	16K
8Kx2	8192	2	-	(1:0)	-	(12:0)	RAMB16_S2	16K
16Kx1	16384	1	-	(0:0)	-	(13:0)	RAMB16_S1	16K



Organizaciones como RAM de doble puerto

		Port A					
		16Kx1	8Kx2	4Kx4	2Kx9	1Kx18	512x36
Port B	16Kx1	_s1_s1					
	8Kx2	_s1_s2	_s2_s2				
	4Kx4	_s1_s4	_s2_s4	_s4_s4			
	2Kx9	_s1_s9	_s2_s9	_s4_s9	_s9_s9		
	1Kx18	_s1_s18	_s2_s18	_s4_s18	_s9_s18	_s18_s18	
	512x36	_s1_s36	_s2_s36	_s4_s36	_s9_s36	_s18_s36	_s36_s36



Block RAM

Primitiva de puerto simple

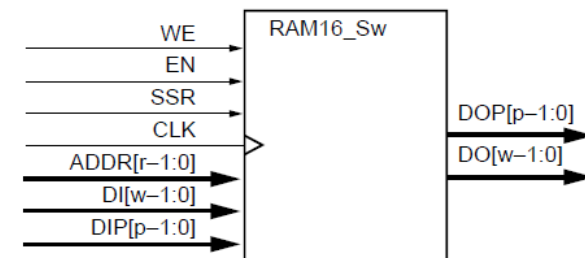


Memoria de puerto simple organizada lógicamente como $2K \times (8D+1P)$

```
component RAMB16_S9
  generic (
    INIT : bit_vector (8 downto 0) := X"000";
    INIT_00 : bit_vector (255 downto 0) := X"00...00";
    ...
    INIT_3F : bit_vector (255 downto 0) := X"00...00";
    INITP_00 : bit_vector (255 downto 0) := X"00...00";
    ...
    INITP_07 : bit_vector (255 downto 0) := X"00...00");
    SRVAL : bit_vector (8 downto 0) := X"000";
    WRITE_MODE : string := "WRITE_FIRST"
  );
  port (
    DO : out std_logic_vector (7 downto 0);
    DOP : out std_logic_vector (0 downto 0);
    ADDR : in std_logic_vector (10 downto 0);
    CLK : in std_ulogic;
    DI : in std_logic_vector (7 downto 0);
    DIP : in std_logic_vector (0 downto 0);
    EN : in std_ulogic;
    SSR : in std_ulogic;
    WE : in std_ulogic
  );
end component;
```

Inicialización de la **memoria de datos**:
64 cadenas de 64 dígitos hexadecimales
 $64 \times 64 \times 4 = 16 \text{ Kb}$

Inicialización de la **memoria de paridad**:
8 cadenas de 64 dígitos hexadecimales
 $8 \times 64 \times 4 = 2 \text{ Kb}$



Block RAM

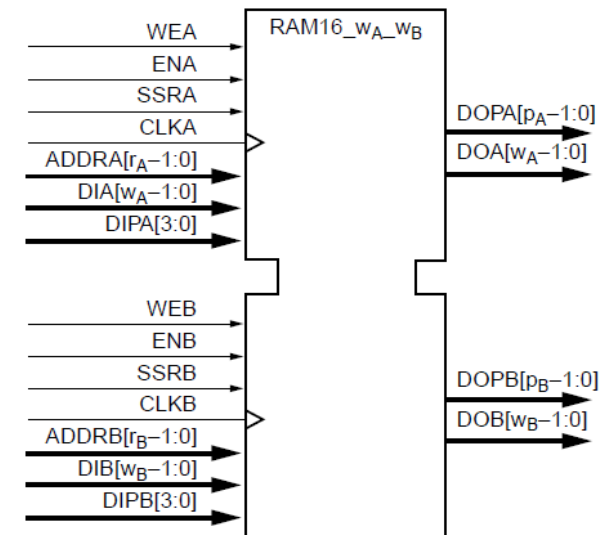
Primitiva de puerto doble



```
component RAMB16_S9_S9
  generic(
    INITP_00      : bit_vector (255 downto 0) := X"00...00";
    ...
    INITP_07      : bit_vector (255 downto 0) := X"00...00";
    INIT_00       : bit_vector (255 downto 0) := X"00...00";
    ...
    INIT_3F       : bit_vector (255 downto 0) := X"00...00";
    INIT_A        : bit_vector (8  downto 0) := X"000";
    INIT_B        : bit_vector (8  downto 0) := X"000";
    SRVAL_A       : bit_vector (8  downto 0) := X"000";
    SRVAL_B       : bit_vector (8  downto 0) := X"000";
    WRITE_MODE_A  : string      := "WRITE_FIRST";
    WRITE_MODE_B  : string      := "WRITE_FIRST");
  port(
    DOA  : out STD_LOGIC_VECTOR (7 downto 0);
    DOB  : out STD_LOGIC_VECTOR (7 downto 0);
    DOPA : out STD_LOGIC_VECTOR (0 downto 0);
    DOPB : out STD_LOGIC_VECTOR (0 downto 0);
    ADDRA : in  STD_LOGIC_VECTOR (10 downto 0);
    ADDR_B : in  STD_LOGIC_VECTOR (10 downto 0);
    CLKA  : in  STD_ULOGIC;
    CLKB  : in  STD_ULOGIC;
    DIA  : in  STD_LOGIC_VECTOR (7 downto 0);
    DIB  : in  STD_LOGIC_VECTOR (7 downto 0);
    DIPA : in  STD_LOGIC_VECTOR (0 downto 0);
    DIPB : in  STD_LOGIC_VECTOR (0 downto 0);
    ENA  : in  STD_ULOGIC;
    ENB  : in  STD_ULOGIC;
    SSRA : in  STD_ULOGIC;
    SSRB : in  STD_ULOGIC;
    WEA  : in  STD_ULOGIC;
    WEB  : in  STD_ULOGIC);
end component;
```

Inicialización independiente
de la organización lógica

Memoria de doble puerto organizada
lógicamente desde ambos puertos
como $2K \times (8D+1P)$



Ejemplos

Instanciación de block RAM (i)



Especificación de una memoria 2K×8b
usando 1 Block RAM organizado en 2K×9b

```
library unisim;
use unisim.vcomponents.all;

architecture ... of ... is
    signal addr: std_logic_vector (10 downto 0);
    signal di, do: std_logic_vector (7 downto 0);
    ...
begin
    ram_2Kx8b: RAMB16_S9
        port map (
            do => do, dop => open, addr=> addr, di => di, dip => "0",
            clk => clk, en => '1', ssr => '0', we => we
        );
    ...
end ...;
```



Ejemplos

Instanciación de block RAM (i)

```
library unisim;
use unisim.vcomponents.all;

architecture ... of ... is
    signal addr: std_logic_vector (13 downto 0);
    signal di, do: std_logic_vector (7 downto 0);
    ...
begin
    ram_16Kx8b: for n in 7 downto 0 generate
        slice: RAMB16_S1
            port map (
                do => do(n downto n), addr => addr, di => di(n downto n),
                clk => clk, en => '1', ssr => '0', we => we
            );
        end generate;
    ...
end ...;
```

Especificación de una memoria 16K×8b
usando 8 Block RAM organizados en 16K×1b

Ejemplos

Instanciación de block RAM (ii)



```
architecture ... of ... is
    signal addr: std_logic_vector (13 downto 0);
    signal di, do: std_logic_vector (7 downto 0);
    signal en: std_logic_vector(7 downto 0);
    signal we: std_logic;
    type dataVector is array (7 downto 0) of std_logic_vector (7 downto 0);
    signal doVector: dataVector;
    ...
begin
    addrDecoder: process ( addr )
    begin
        en <= (others => '0');
        en( to_integer( unsigned(addr(13 downto 11)) ) ) <= '1';
    end process;

    ram_16KBx8b: for n in 7 downto 0 generate
        slice: RAMB16_S9
            port map (
                do=>doVector(n), dop=>open, addr=>addr(10 downto 0), di=>di, dip=>"0",
                clk=>clk, en=>en(n), ssr=>'0', we=>we
            );
    end generate;

    doMultiplexer:
        do <= doVector( to_integer(unsigned(addr(13 downto 11)) ) );
    ...
end ...;
```

Especificación de una memoria 16Kx8b
usando 8 Block RAM organizados en 2Kx9b

Ejemplos

Inferencia de block RAM (i)



```
architecture ...;
  signal addr: std_logic_vector(10 downto 0);
  signal di, do: std_logic_vector(7 downto 0);
  type ramType is array (0 to 2**11-1) 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( address ) ) ) <= di;
          do <= di;
        else
          do <= ram( to_integer( unsigned( address ) ) ) ;
        end if;
      end if;
    end if;
  end process;
  ...
end;
```

Especificación de una memoria 2K×8b
de tipo block RAM en modo WRITE_FIRST

```
...
type ramType is array (0 to 2**14-1) of std_logic_vector(7 downto 0);
signal ram : ramType;
...
```

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



Ejemplos

Inferencia de block RAM (ii)

Especificación de una memoria 2Kx8b
de tipo block RAM en modo READ_FIRST

```
architecture ...;
  signal addr: std_logic_vector(10 downto 0);
  signal di, do: std_logic_vector(7 downto 0);
  type ramType is array (0 to 2**11-1) 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;
```



Ejemplos

Inferencia de block RAM (iii)

Especificación de una memoria 2Kx8b
de tipo block RAM en modo NO_CHANGE

```
architecture ...;
  signal addr: std_logic_vector(10 downto 0);
  signal di, do: std_logic_vector(7 downto 0);
  type ramType is array (0 to 2**11-1) 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;
        else
          do <= ram( to_integer( unsigned( addr ) ) ) ;
        end if;
      end if;
    end if;
  end process;
  ...
end;
```

Ejemplos

Instanciación de block RAM como ROM



```
library unisim;  
use unisim.vcomponents.all;
```

```
architecture ... of ... is
```

```
    signal addr: std_logic_vector (10 downto 0);
```

```
    signal do: std_logic_vector (7 downto 0);
```

```
    ...
```

```
begin
```

```
    rom_2Kx8b: RAMB16_S9
```

```
        generic map (
```

```
            INIT_00 => X"0f235...2c",
```

```
            ...
```

```
            INIT_3f => X"...",
```

```
        );
```

```
        port map (
```

```
            do => do, dop => open, addr => addr, di => X"00", dip => "0",
```

```
            clk => clk, en => '1', ssr => '0', we => '0'
```

```
        );
```

```
    ...
```

```
end ...;
```

Especificación de una ROM 2Kx8b con lectura síncrona
usando 1 Block RAM organizado en 2Kx9b



Ejemplos

Inferencia de block RAM como ROM

Especificación de una ROM 2K×8b con lectura síncrona implementada con block RAM

```
architecture ...;
...
signal addr: std_logic_vector (10 downto 0);
signal do: std_logic_vector (7 downto 0);
type romType is array (0 to 2**11-1) 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;
```

Ejemplos

Inferencia de distributed RAM



Especificación de una RAM
de lectura asíncrona

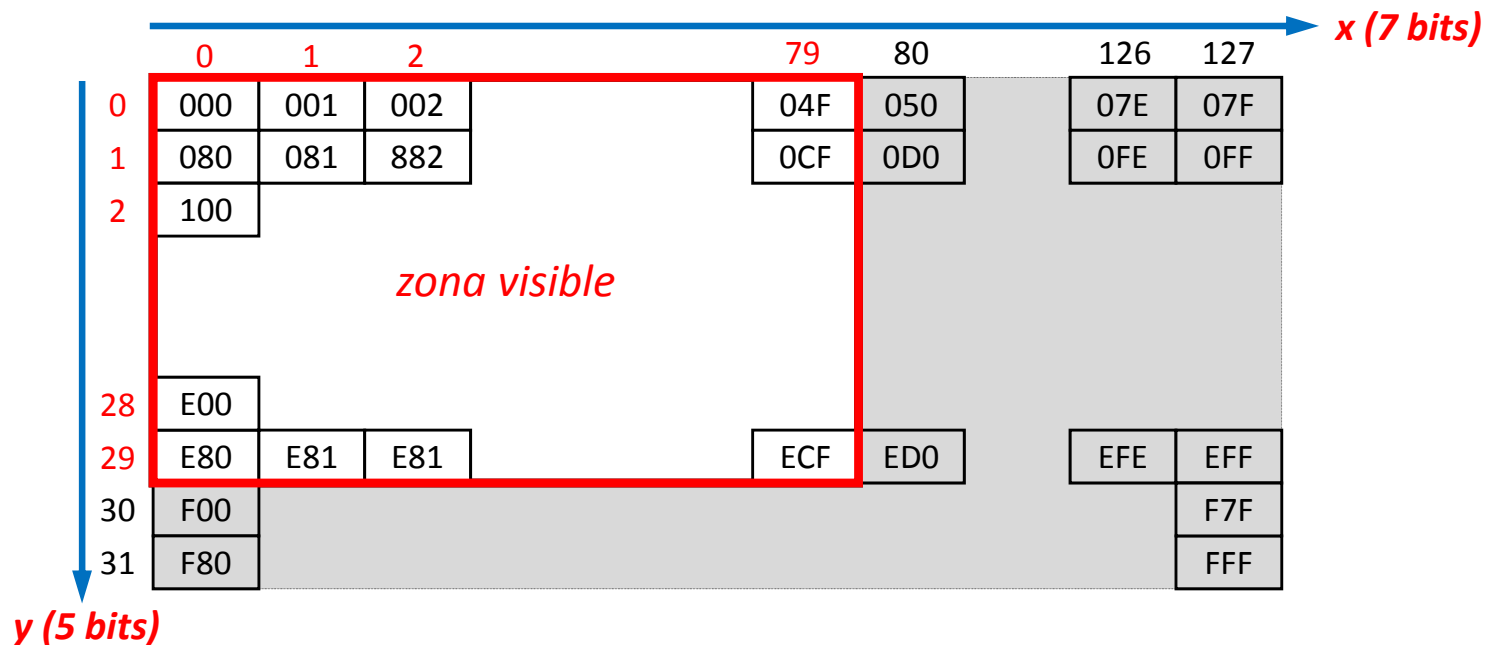
```
architecture ...;
...
signal addr: std_logic_vector(... downto 0);
signal di, do: std_logic_vector(... downto 0);
type ramType is array (0 to ...) of std_logic_vector(... downto 0);
signal ram : ramType;
begin
...
process (clk)
begin
    if rising_edge(clk) then
        if we='1' then
          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



Interfaz VGA de texto

memoria de refresco

- En una **RAM** se almacenan los **códigos de cada carácter a visualizar**:
 - La dirección de memoria se obtiene 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 $2^{7+5} \text{ B} = 4 \text{ KB}$
 - Equivalente a **2 Block RAM** en configuración **2Kx9**
 - Será de **doble puerto** para poder escribir y leer en paralelo
 - Un puerto para escribir (o borrar) y otro para leer el carácter a refrescar.
 - El borrado consistirá en escribir un 0 en todas las posiciones de memoria.



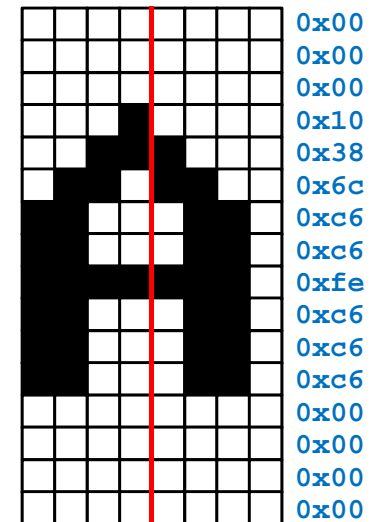


Interfaz VGA de texto

memoria de mapa de bits

- En una **ROM** se almacena una **fuentes 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 $256 \cdot 16B = 4 \text{ KB}$
 - Equivalente a **2 Block RAM** en configuración **2K×9**

```
bitMapRomLow: RAMB16_S9
  generic map (
    ...
    INIT_20 => bitReverse(X"...00000010386cc6c6fec6c6c600000000"),
    ...
  )
  port map ( ... );
```

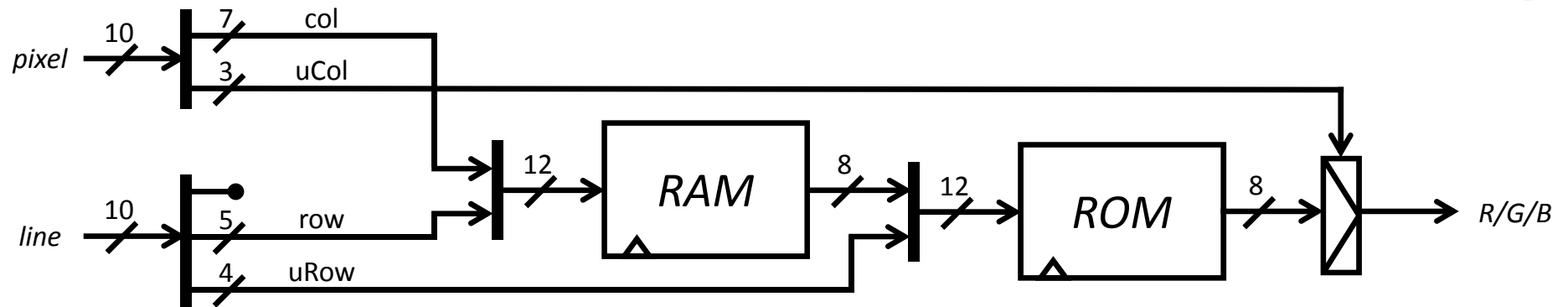


*El tipo de los genéricos es `bit_vector(255 downto 0)`
pero los mapas de bits están ordenados como `bit_vector(0 to 255)`*



Interfaz VGA de texto

procedimiento de refresco (i)



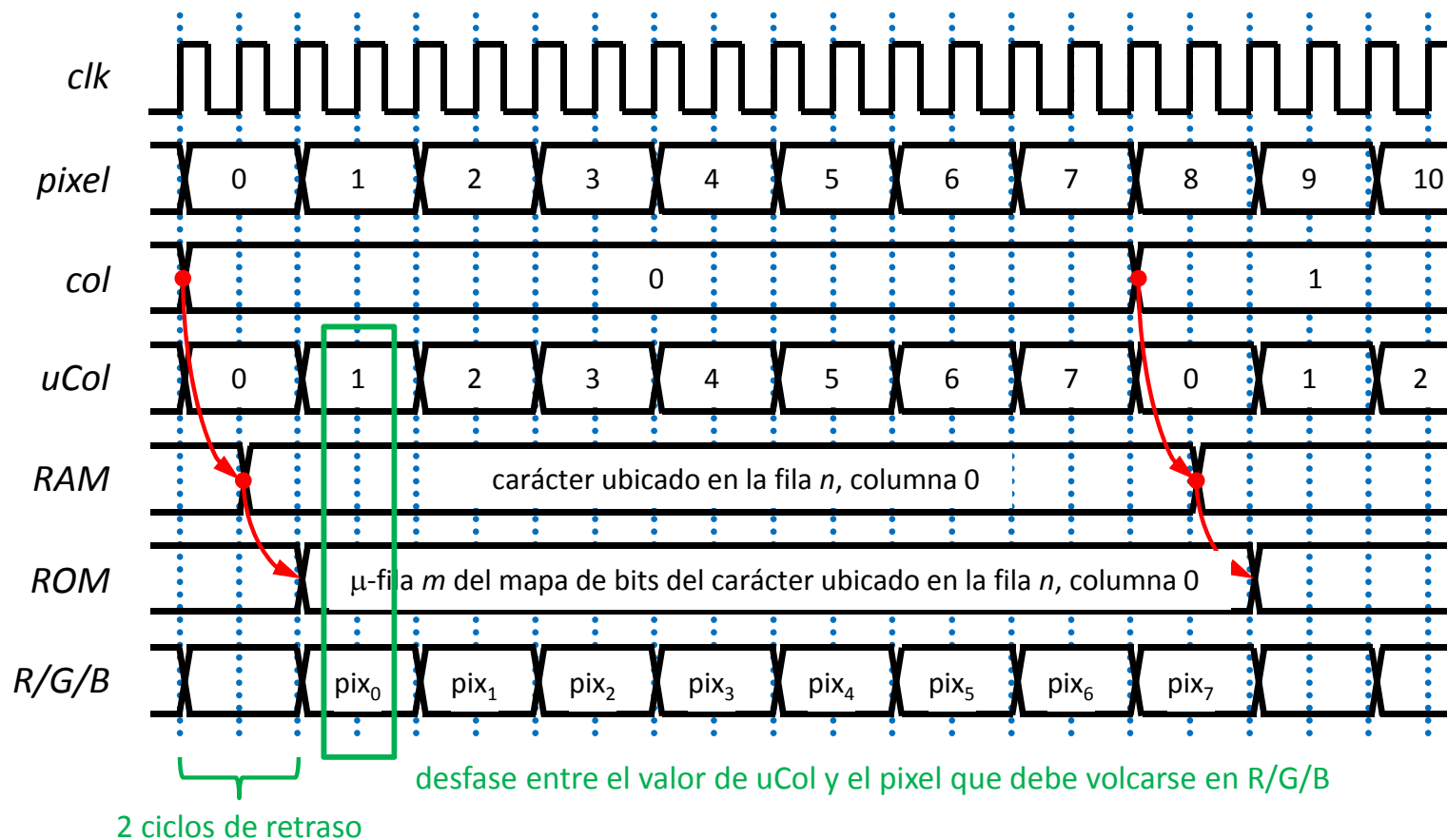
- Usará un módulo **vgaInterface** para refrescar la pantalla
 - A partir de la salida **pixel** se obtiene la **columna** de la pantalla que se está refrescando y la **μ-columna** del mapa de bits que se debe refrescar.
 - A partir de la salida **line** se obtiene la **fila** de la pantalla que se está 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.



Interfaz VGA de texto

procedimiento de refresco (ii)

- Dado que la RAM y la ROM tienen lectura síncrona, es necesario:
 - Compensar el pixel de retraso (2 ciclos) en el envío a la pantalla del píxel correspondiente a una posición dada.
 - Compensar el desfase entre la salida de la ROM y uCol (que selecciona un pixel)



Diseño principal



- Para **testar el interfaz VGA de texto** realizaremos un diseño que visualice los caracteres recibidos desde un teclado PS/2.
 - Lo hará de izquierda a derecha y de arriba a abajo comenzando por la esquina superior izquierda.
 - Convirtiendo los scancodes de presión al correspondiente código ASCII y activando durante un ciclo la señal de **charRdy** del vgaTxtInterface
 - 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 vgaTxtInterface
 - **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.



Tareas



1. Crear el proyecto **lab7** en el directorio **DAS**
2. Descargar de la Web en el directorio **common** el fichero **vgaTxtInterface.vhd**
3. Descargar de la Web en el directorio **lab6** los ficheros:
 - **lab7.vhd** y **lab7.ucf**
4. Completar el fichero **common.vhd** con la declaración del nuevo componente reusable.
5. Completar el código omitido en los ficheros:
 - **vgaTxtInterface.vhd** y **lab7.vhd**
6. Añadir al proyecto los ficheros:
 - **common.vhd**, **common.vhd**, **synchronizer.vhd**, **edgedetector.vhd**, **ps2Receiver.vhd**, **vgaInterface.vhd**, **lab5.vhd** y **lab5.ucf**
7. Sintetizar, implementar y generar el fichero de configuración.
8. Conectar el teclado y el monitor a la placa y encenderla.
9. Descargar el fichero **lab7.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/>