A Video Controller

Prof. Stephen A. Edwards
sedwards@cs.columbia.edu

NCTU, Summer 2005
Project specification:

- Flexible video controller
- Driven by C from the Microblaze
  Not interesting otherwise
- Only uses on-chip RAM
  Specifically, we won’t use the SRAM or SDRAM. Makes it easier to use it in projects that use other peripherals.
Assume we have an XC2S300E-6PQ208C FPGA (slightly different than what is on the Digilent board).

From the Xilinx “Spartan-IIE 1.8V FPGA Family” data sheet, we find

<table>
<thead>
<tr>
<th>Device</th>
<th>CLB array</th>
<th>CLBs</th>
<th>Distributed RAM Bits</th>
<th>Block RAM Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC2S300E</td>
<td>32 × 48</td>
<td>1536</td>
<td>98 304</td>
<td>64K</td>
</tr>
</tbody>
</table>

Distributed RAM uses the LUTs and is awkward.
Block RAM is dual-ported.
We have at most 8K bytes.
From the Xilinx “Spartan-II E 1.8V FPGA Family: Functional Description” datasheet, Block RAMs are 4096 bits each.
The XC2S300E has 16 such blocks.
Many different configurations. We like the byte-wide one: 512 × 8.
The dual-ported nature is very convenient.

Two truly independent input/output ports that do not have to be synchronized.

We will use one port for the video controller, the other for the processor.
VGA is $640 \times 480 = 307200$ pixels

Even at one bit per pixel, this requires
$307200 \div 1024 = 300K\text{ bits} = 37.5K\text{ bytes}$

Too much for on-chip RAM; we only have 8K max.
Memory Usage

How about a text-only display?
80 × 24 is typical.
640 ÷ 80 = 8 pixels horizontally/character
480 ÷ 24 = 20 pixels vertically/character
A bit too vertical. 8 × 16 is more typical.
480 ÷ 16 = 30 lines
80 × 30 = 2400 characters
Over 2K (2048). 2400 = 2048 + 352
Will fit in 2.5K = 2560
How about the font?

For 8 × 16 characters, the 96 basic ASCII characters take

\[ 16 \times 96 = 1536 = 1.5K \]
Memory is our most valuable resource. We’ve decided to allocate it as follows:

2.5K for a 80 × 30 character array

1.5K for a 96-character 8 × 16 font

That is half (4K) of the 8K on-chip memory. Reasonable.

Also nice because we can use standard IBM console fonts.
Aside

That 640 × 480 exactly fits an 80 × 30 8 × 16 font is no accident. Historically,

<table>
<thead>
<tr>
<th>Standard</th>
<th>Resolution</th>
<th>Font</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGA (1981)</td>
<td>640 × 200</td>
<td>8 × 8</td>
<td>80 × 25</td>
</tr>
<tr>
<td>EGA (1984)</td>
<td>640 × 350</td>
<td>8 × 14</td>
<td>80 × 25</td>
</tr>
<tr>
<td>EGA (1984)</td>
<td>640 × 350</td>
<td>8 × 8</td>
<td>80 × 43</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>640 × 400</td>
<td>8 × 16</td>
<td>80 × 25</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>640 × 400</td>
<td>8 × 14</td>
<td>80 × 28</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>640 × 400</td>
<td>8 × 8</td>
<td>80 × 50</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>640 × 480</td>
<td>8 × 16</td>
<td>80 × 30</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>640 × 480</td>
<td>8 × 14</td>
<td>80 × 34</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>640 × 480</td>
<td>8 × 8</td>
<td>80 × 60</td>
</tr>
</tbody>
</table>
Microarchitecture

```
Dout  Char. RAM 2.5K  Controller  VSYNC
Din   Font RAM 1.5K
Addr  Dout/Shift

Shift Register

BLANK

HSYNC

Load/Shift

Video
```

A Video Controller – p. 11/5
It is reasonable to assume the shift register, AND gate, and controller will operate at full speed.

But how fast is the RAM?

In the “Spartan-IIE 1.5V FPGA Family: DC and Switching Characteristics” datasheet:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{BCKO}$</td>
<td>CLK to DOUT</td>
<td>0.6ns</td>
<td>3.5ns</td>
</tr>
<tr>
<td>$T_{BACK}/T_{BCKA}$</td>
<td>Setup/hold on ADDR</td>
<td>1.1ns/0</td>
<td>-</td>
</tr>
<tr>
<td>$T_{BPWH}$</td>
<td>Clock pulse width high</td>
<td>1.5ns</td>
<td>-</td>
</tr>
<tr>
<td>$T_{BPWL}$</td>
<td>Clock pulse width low</td>
<td>1.5ns</td>
<td>-</td>
</tr>
</tbody>
</table>
Timing

Maximum clock frequency: \( \frac{1}{3\text{ns}} = 333 \text{ MHz} \)
Highest data rate: \( \frac{1}{3.5\text{ns}} = 285 \text{ MHz} \)
VGA dot clock is 25 MHz
The RAM is much, much faster than we need.
We can assume data from the RAM appears “quickly” after the clock.
Pixel-Level Timing

Clk
CharAddr
LoadChar
CharData
FontLoad
PixelData
Load/Shift
Bit
For a 25.175 MHz pixel clock,

- HSYNC: 96 pixels
- BACK_PORCH: 48
- HACTIVE: 640
- FRONT_PORCH: 16
- HTOTAL: 800
Start-of-line Detail

Clk
Hcount
Column
LoadChar
CharData
FontLoad
PixelData
Load/Shift
HBLANK
Pixel
Vertical Timing

VTOTAL

Video

VSYNC

HSYNC

BACK_PORCH

FRONT_PORCH

VSYNC  
VACTIVE

2 lines

VSYNC  33

BACK_PORCH  480

VACTIVE  10

FRONT_PORCH  525

VTOTAL  525
### Character Addresses

<table>
<thead>
<tr>
<th>line</th>
<th>character address, row</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,0 1,0 ... 78,0 79,0</td>
</tr>
<tr>
<td>1</td>
<td>0,1 1,1 ... 78,1 79,1</td>
</tr>
<tr>
<td>14</td>
<td>0,14 1,14 ... 78,14 79,14</td>
</tr>
<tr>
<td>15</td>
<td>0,15 1,15 ... 78,15 79,15</td>
</tr>
<tr>
<td>16</td>
<td>80,0 81,0 ... 158,0 159,0</td>
</tr>
<tr>
<td>17</td>
<td>80,1 81,1 ... 158,1 159,1</td>
</tr>
<tr>
<td>31</td>
<td>80,15 81,15 ... 158,15 159,15</td>
</tr>
<tr>
<td>32</td>
<td>160,0 161,0 ... 238,0 239,0</td>
</tr>
<tr>
<td>464</td>
<td>2320,0 2321,0 ... 2398,0 2399,0</td>
</tr>
<tr>
<td>479</td>
<td>2320,15 2321,15 ... 2398,15 2399,15</td>
</tr>
</tbody>
</table>
## Horizontal Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixel_Clock</td>
<td>25 MHz from system clock.</td>
</tr>
<tr>
<td>Hcount[9:0]</td>
<td>Horizontal position (0–799)</td>
</tr>
<tr>
<td>VideoData</td>
<td>Pixel data from shift register</td>
</tr>
<tr>
<td>LoadNShift</td>
<td>Shift register control. 143, 151, … 775</td>
</tr>
<tr>
<td>FontData[7:0]</td>
<td>Byte from font RAM.</td>
</tr>
<tr>
<td>FontLoad</td>
<td>Load byte from font RAM. 142, 150, …, 774</td>
</tr>
<tr>
<td>FontAddr[3:0]</td>
<td>Character row address (0–15)</td>
</tr>
<tr>
<td>FontAddr[10:4]</td>
<td>Character number (0–95) from char. RAM</td>
</tr>
<tr>
<td>LoadChar</td>
<td>Load from char. RAM. 141, 149, …, 773</td>
</tr>
<tr>
<td>CharAddr[11:0]</td>
<td>Character address: column plus row × 80</td>
</tr>
<tr>
<td>Column[6:0]</td>
<td>Column (0–79) = (Hcount – 141) ÷ 8</td>
</tr>
<tr>
<td>HBLANK_N</td>
<td>Horizontal blanking. Off 144–783</td>
</tr>
<tr>
<td>HSYNC_N</td>
<td>Horizontal Synchronization. 0–95</td>
</tr>
</tbody>
</table>
## Vertical Signals

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vcount[9:0]</td>
<td>Vertical position (0–524)</td>
</tr>
<tr>
<td>Row[4:0]</td>
<td>Row number (0–29) = (Vcount – 35) ÷ 16</td>
</tr>
<tr>
<td>VBLANK_N</td>
<td>Vertical blanking. Off lines 35–514</td>
</tr>
<tr>
<td>VSYNC_N</td>
<td>Vertical Synchronization. Lines 0 and 1</td>
</tr>
</tbody>
</table>
### Interface to the RAMB4\_S8\_S8

From the Xilinx libraries guide:

<table>
<thead>
<tr>
<th>EN</th>
<th>RST</th>
<th>WE</th>
<th>CLK</th>
<th>ADDR</th>
<th>DI</th>
<th>DO</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>DO No-op</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>↑</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>Reset DO</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>↑</td>
<td>addr</td>
<td>data</td>
<td>0</td>
<td>Write</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>↑</td>
<td>addr</td>
<td>-</td>
<td>[addr]</td>
<td>Read</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>↑</td>
<td>addr</td>
<td>data</td>
<td>data</td>
<td>Write-through</td>
</tr>
</tbody>
</table>

A Video Controller – p. 22/57
The video RAM interface

Both the character and font RAMs are read-only, so we will use

<table>
<thead>
<tr>
<th>Signal</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN</td>
<td>1 for read, 0 for hold</td>
</tr>
<tr>
<td>RST</td>
<td>0</td>
</tr>
<tr>
<td>WE</td>
<td>0</td>
</tr>
<tr>
<td>ADDR</td>
<td>read address</td>
</tr>
<tr>
<td>DI</td>
<td>0 (unused)</td>
</tr>
</tbody>
</table>
The OPB Interface

On-chip Peripheral Bus. Part of IBM’s CoreConnect bus architecture. A lower-speed bus for peripherals such as our video controller.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPB_Clk</td>
<td>sl_DBus</td>
</tr>
<tr>
<td>OPB_Rst</td>
<td>sl_xferAck</td>
</tr>
<tr>
<td>OPB_ABus</td>
<td>sl_retry</td>
</tr>
<tr>
<td>OPB_BE</td>
<td>sl_toutSup</td>
</tr>
<tr>
<td>OPB_DBus</td>
<td>sl_errAck</td>
</tr>
<tr>
<td>OPB_RNW</td>
<td></td>
</tr>
<tr>
<td>OPB_select</td>
<td></td>
</tr>
<tr>
<td>OPB_seqAddr</td>
<td></td>
</tr>
</tbody>
</table>
OPB Signals

**OPB_Clk**  Bus clock: master synchronization
**OPB_Rst**  Global asynchronous reset
**OPB_ABus[31:0]** Address
**OPB_BE[3:0]** Byte enable
**OPB_DBus** Data to slave
**OPB_RNW**  1=read from slave, 0=write to slave
**OPB_select** Transfer in progress
**OPB_seqAddr** Next sequential address pending (unused)

**sl_DBus**  Data from slave. Must be 0 when inactive
**sl_xferAck** Transfer acknowledge. OPB_select→0
**sl_retry**  Request master to retry operation (=0)
**sl_toutSup** Suppress slave time-out (=0)
**sl_errAck**  Signal a transfer error occurred (=0)
Typical OPB Read Cycle Timing

OPB_CLK

OPB_select

OPB_ABus

OPB_BE

OPB_RNW

sl_DBus 00000000 valid 00000000

sl_xferAck

OPB signals arrive late; DBus and xferAck needed early.
Typical OPB Write Cycle Timing

- OPB_CLK
- OPB_select
- OPB_ABus
- OPB_BE
- OPB_RNW
- OBP_DBus
- sl_xferAck
OPB interface block diagram

- OPB_DBus
- Controller
- OPB_ABus
- OPB_Select
- OPB_RNW
- RAM 0
- RAM 1
- ... (n)
- VGA_DBus
OPB-RAM Read Timing

- **OPB_CLK**
- **OPB_select**
- **OPB_ABus**
- **OPB_RNW**
- **RAM-Data**
- **VGA_DBus**
  - 00000000 valid 00000000
- **RST**
  - (one only)
- **WE**
  - (always low)
- **MemCycle1**
- **MemCycle2**
- **VGA_xferAck**
OPB-RAM Write Timing

- **OPB_CLK**
- **OPB_select**
- **OPB_ABus**
- **OPB_RNW**
- **OPB_DBus**
- **RST** *(always high)*
- **WE** *(one only)*
- **MemCycle1**
- **MemCycle2**
- **VGA_xferAck**
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity opb_xsb300e_vga is
  generic (    
    C_OPB_AWIDTH : integer := 32;
    C_OPB_DWIDTH : integer := 32;
    C_BASEADDR : std_logic_vector(31 downto 0) := X"FEFF1000";
    C_HIGHADDR : std_logic_vector(31 downto 0) := X"FEFF1FFF"
  );
VHDL: OPB Ports

```vhdl
port (  
    OPB_Clk : in std_logic;  
    OPB_Rst : in std_logic;  

    -- OPB signals  
    OPB_ABus : in std_logic_vector(31 downto 0);  
    OPB_BE : in std_logic_vector(3 downto 0);  
    OPB_DBus : in std_logic_vector(31 downto 0);  
    OPB_RNW : in std_logic;  
    OPB_select : in std_logic;  
    OPB_seqAddr : in std_logic;  

    VGA_DBus : out std_logic_vector(31 downto 0);  
    VGA_errAck : out std_logic;  
    VGA_retry : out std_logic;  
    VGA_toutSup : out std_logic;  
    VGA_xferAck : out std_logic;  
);  
```
-- Video signals
Pixel_Clock : in std_logic; -- 25 MHz
VIDOUT_RED : out std_logic;
VIDOUT_GREEN : out std_logic;
VIDOUT_BLUE : out std_logic;
VIDOUT_HSYNC : out std_logic;
VIDOUT_VSYNC : out std_logic
);

end opb_xsb300e_vga;
architecture Behavioral of opb_xsb300e_vga is

constant BASEADDR : std_logic_vector(31 downto 0) := X"FEFF1000";

-- Video parameters

constant HTOTAL : integer := 800;
constant HSYNC : integer := 96;
constant HBACK_PORCH : integer := 48;
constant HACTIVE : integer := 640;
constant HFRONT_PORCH : integer := 16;
constant VTOTAL : integer := 525;
constant VSYNC : integer := 2;
constant VBACK_PORCH : integer := 33;
constant VACTIVE : integer := 480;
constant VFRONT_PORCH : integer := 10;
-- Latched input signals from the OPB
signal ABus : std_logic_vector (31 downto 0);
signal DBus : std_logic_vector (31 downto 0);
signal RNW : std_logic;
signal select_delayed : std_logic;

-- Latched output data for the OPB
signal DBus_out : std_logic_vector (31 downto 0);

-- Signals for the OPB-mapped RAM controller
signal ChipSelect : std_logic;
signal MemCycle1, MemCycle2 : std_logic;
signal RamPageAddress : std_logic_vector(2 downto 0);
signal RamSelect : std_logic_vector (7 downto 0);
signal RST, WE : std_logic_vector (7 downto 0);
signal DOUT0, DOUT1, DOUT2, DOUT3, DOUT4, DOUT5, DOUT6, DOUT7 : std_logic_vector(7 downto 0);
signal ReadData : std_logic_vector(7 downto 0);
-- Master horizontal and vertical video counters
signal Hcount : std_logic_vector(9 downto 0);
signal Vcount : std_logic_vector(9 downto 0);
signal HBLANK_N, VBLANK_N : std_logic;
signal EndOfLine, EndOfField : std_logic;

-- Addresses and control for character RAM
signal LoadChar : std_logic;
signal CharRow, CharColumn
  : std_logic_vector(9 downto 0);
signal Column : std_logic_vector(6 downto 0);
signal Row : std_logic_vector(4 downto 0);
signal CharAddr : std_logic_vector(11 downto 0);
signal CharRamPage : std_logic_vector(2 downto 0);
signal CharRamSelect_N
  : std_logic_vector(4 downto 0);
signal DOUTB0, DOUTB1, DOUTB2, DOUTB3, DOUTB4
  : std_logic_vector(7 downto 0);
-- Addresses and control for font RAM
signal FontLoad : std_logic;
signal FontAddr : std_logic_vector(10 downto 0);
signal FontRamPage : std_logic_vector(1 downto 0);
signal FontRamSelect_N
    : std_logic_vector(2 downto 0);
signal DOUTB5, DOUTB6, DOUTB7
    : std_logic_vector(7 downto 0);

-- Shift register control, inputs, and data
signal LoadNShift : std_logic;
signal FontData : std_logic_vector(7 downto 0);
signal ShiftData : std_logic_vector(7 downto 0);
signal VideoData : std_logic;
-- 512 X 8 dual-ported Xilinx block RAM
component RAMB4_S8_S8
port (  
  DOA : out std_logic_vector (7 downto 0);
  ADDRA : in std_logic_vector (8 downto 0);
  CLKA : in std_logic;
  DIA : in std_logic_vector (7 downto 0);
  ENA : in std_logic;
  RSTA : in std_logic;
  WEA : in std_logic;
  DOB : out std_logic_vector (7 downto 0);
  ADDRB : in std_logic_vector (8 downto 0);
  CLKB : in std_logic;
  DIB : in std_logic_vector (7 downto 0);
  ENB : in std_logic;
  RSTB : in std_logic;
  WEB : in std_logic);
end component;

-- Attributes that control the initial values
-- loaded into block RAMs
attribute INIT_00 : string;
attribute INIT_01 : string;
VHDL: BRAM initialization

attribute INIT_00 of RAMB4_S8_S8_5 : label is
"00000000181801818183c3c3c18000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
begin -- body of architecture

  RAMB4_S8_S8_0 : RAMB4_S8_S8
  port map (DOA => DOUT0,
              ADDRA => ABus(8 downto 0),
              CLKA => OPB_Clk,
              DIA => DBus(7 downto 0),
              ENA => '1',
              RSTA => RST(0),
              WEA => WE(0),
              DOB => DOUTB0,
              ADDRB => CharAddr(8 downto 0),
              CLKB => Pixel_Clock,
              DIB => X"00",
              ENB => '1',
              RSTB => CharRamSelect_N(0),
              WEB => '0');

-- Four more like this
VHDL: Font BRAM instances

```vhdl
-- Two more like this
```

```
RAMB4_S8_S8_5 : RAMB4_S8_S8
port map ( DOA => DOUT5,
            ADDRA => ABus(8 downto 0),
            CLKA => OPB_Clk,
            DIA => DBus(7 downto 0),
            ENA => '1',
            RSTA => RST(5),
            WEA => WE(5),
            DOB => DOUTB5,
            ADDRB => FontAddr(8 downto 0),
            CLKB => Pixel_Clock,
            DIB => X"00",
            ENB => '1',
            RSTB => FontRamSelect_N(0),
            WEB => '0');
```
-- OPB-RAM controller

-- Unused OPB control signals
VGA_errAck <= '0';
VGA_retry  <= '0';
VGA_toutSup <= '0';

-- Latch late-arriving OPB signals
LatchOPB: process (OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    Abus <= ( others => '0' );
    DBus <= ( others => '0' );
    RNW  <= '1';
    select_delayed <= '0';
  elsif OPB_Clk'event and OPB_Clk = '1' then
    ABus <= OPB_ABus;
    DBus <= OPB_DBus;
    RNW  <= OPB_RNW;
    select_delayed <= OPB_Select;
  end if;
end process LatchOPB;
ChipSelect <=
    '1' when select_delayed = '1' and
        (ABus(31 downto 12) =
            BASEADDR(31 downto 12)) and
        MemCycle1 = '0' and MemCycle2 = '0' else
    '0';

RamPageAddress <= ABus(11 downto 9);
RamSelect <=
    "00000001" when RamPageAddress = "000" else
    "00000010" when RamPageAddress = "001" else
    "00000100" when RamPageAddress = "010" else
    "00001000" when RamPageAddress = "011" else
    "00010000" when RamPageAddress = "100" else
    "00100000" when RamPageAddress = "101" else
    "01000000" when RamPageAddress = "110" else
    "10000000" when RamPageAddress = "111" else
    "00000000";
MemCycleFSM : process(OPB_Clk, OPB_Rst)
begin
  if OPB_Rst = '1' then
    MemCycle1 <= '0';
    MemCycle2 <= '0';
  elsif OPB_Clk'event and OPB_Clk = '1' then
    MemCycle2 <= MemCycle1;
    MemCycle1 <= ChipSelect;
  end if;
end process MemCycleFSM;

VGA_xferAck <= MemCycle2; -- OPB output

WE <= RamSelect when ChipSelect = '1' and RNW = '0' and OPB_Rst = '0'
else "00000000";

RST <= not RamSelect when ChipSelect = '1' and RNW = '1' and OPB_Rst = '0'
else "11111111";
ReadData <=
    DOUT0 or DOUT1 or DOUT2 or DOUT3 or DOUT4 or DOUT5 or DOUT6 or DOUT7
    when MemCycle1 = '1'
else "00000000";

GenDOut: process (OPB_Clk, OPB_Rst)
begin
    if OPB_Rst = '1' then
        DBus_out <= ( others => '0');
    elsif OPB_Clk'event and OPB_Clk = '1' then
        DBus_out <= ReadData & ReadData & ReadData & ReadData;
    end if;
end process GenDOut;

VGA_DBus <= DBus_out;
-- Video controller

HCounter : process (Pixel_Clock, OPB_Rst)
begin
    if OPB_Rst = '1' then
        Hcount <= (others => '0');
    elsif Pixel_Clock'event and Pixel_Clock = '1' then
        if EndOfLine = '1' then
            Hcount <= (others => '0');
        else
            Hcount <= Hcount + 1;
        end if;
    end if;
end process HCounter;

EndOfLine <=
    '1' when Hcount = HTOTAL - 1 else '0';
VCounter: process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    Vcount <= (others => '0');
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if EndOfLine = '1' then
      if EndOfField = '1' then
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
      end if;
      EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';
    end if;
  end if;
end process VCounter;
HSyncGen : process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    VIDOUT_HSYNC <= '0';
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if EndOfLine = '1' then
      VIDOUT_HSYNC <= '1';
    elsif Hcount = HSYNC - 1 then
      VIDOUT_HSYNC <= '0';
    end if;
  end if;
end process HSyncGen;
HBlankGen : process (Pixel_Clock, OPB_Rst) begin
    if OPB_Rst = '1' then
        HBLANK_N <= '0';
    elsif Pixel_Clock'event and Pixel_Clock = '1' then
        if Hcount = HSYNC + HBACK_PORCH - 1 then
            HBLANK_N <= '1';
        elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE - 1 then
            HBLANK_N <= '0';
        end if;
    end if;
end process HBlanksGen;
VSyncGen : process (Pixel_Clock, OPB_Rst)
begin
    if OPB_Rst = '1' then
        VIDOUT_VSYNC <= '0';
    elsif Pixel_Clock'event and Pixel_Clock = '1' then
        if EndOfLine = '1' then
            if EndOfField = '1' then
                VIDOUT_VSYNC <= '1';
                elsif VCount = VSYNC - 1 then
                    VIDOUT_VSYNC <= '0';
            end if;
        end if;
    end if;
end process VSyncGen;
VBlankGen : process (Pixel_Clock, OPB_Rst) begin
  if OPB_Rst = '1' then
    VBLANK_N <= '0';
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if EndOfLine = '1' then
      if Vcount = VSYNC + VBACK_PORCH - 1 then
        VBLANK_N <= '1';
      elsif VCount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
        VBLANK_N <= '0';
      end if;
    end if;
  end if;
end process VBlankGen;
LoadChar <=
  '1' when Hcount(2 downto 0) = X"5" else '0';
FontLoad  <=
  '1' when Hcount(2 downto 0) = X"6" else '0';
LoadNShift <=
  '1' when Hcount(2 downto 0) = X"7" else '0';

CharColumn <= Hcount - HSYNC - HBACK_PORCH + 4;
Column    <= CharColumn(9 downto 3);
CharRow   <= Vcount - VSYNC - VBACK_PORCH;
Row       <= CharRow(8 downto 4);

-- Character address = Column + Row * 80
CharAddr <= Column +
  ("0" & Row(4 downto 0) & "000000")
  ("000" & Row(4 downto 0) & "0000");
CharRamPage <= CharAddr(11 downto 9);
CharRamSelect_N <=
    "11110" when CharRamPage = "000" else
    "11101" when CharRamPage = "001" else
    "11011" when CharRamPage = "010" else
    "10111" when CharRamPage = "011" else
    "01111" when CharRamPage = "100" else
    "11111";

FontAddr(10 downto 4) <=
    (DOUTB0(6 downto 0) or DOUTB1(6 downto 0) or
    DOUTB2(6 downto 0) or DOUTB3(6 downto 0) or
    DOUTB4(6 downto 0));
FontAddr(3 downto 0) <= CharRow(3 downto 0);
FontRamPage <= FontAddr(10 downto 9);
FontRamSelect_N <=
  "110" when FontRamPage = "00" else
  "110" when FontRamPage = "01" else
  "101" when FontRamPage = "10" else
  "011" when FontRamPage = "11" else
  "111";

FontData <= DOUTB5 or DOUTB6 or DOUTB7;
ShiftRegister: process (Pixel_Clock, OPB_Rst) begin  
  if OPB_Rst = '1' then  
    ShiftData <= X"00";  
  elsif Pixel_Clock'event and Pixel_Clock = '1' then  
    if LoadNShift = '1' then  
      ShiftData <= FontData;  
    else  
      ShiftData <= ShiftData(6 downto 0) & '0';  
    end if;  
  end if;  
end process ShiftRegister;  

VideoData <= ShiftData(7);
VideoOut: process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    VIDOUT_RED  <= '0';
    VIDOUT_BLUE <= '0';
    VIDOUT_GREEN <= '0';
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if VideoData = '1' and HBLANK_N = '1' and VBLANK_N = '1' then
      VIDOUT_RED  <= '1';
      VIDOUT_GREEN <= '1';
      VIDOUT_BLUE <= '1';
    else
      VIDOUT_RED  <= '0';
      VIDOUT_GREEN <= '0';
      VIDOUT_BLUE <= '0';
    end if;
  end if;
end process VideoOut;
end Behavioral; -- end of architecture
This video controller is not quite right for the Spartan-3 chip on the Digilent boards.

Our boards have an XC3S400, which has 32K of on-chip RAM, not 8K. The block RAMs are similar, but are 4K each, not 512 bytes.

What is presented here works, but is wasting memory resources.

A better design would use only 3 of the Spartan-3 block RAMS, not 8.