A Video Controller

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

NCTU, Summer 2005

A Video Controller for an FPGA

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.

On-Chip RAM

Assume we have an XC2S300E-6PQ208C FPGA (slightly different than what is on the Digilent board).

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

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

Distributed RAM uses the LUTs and is awkward. Block RAM is dual-ported.
We have at most 8K bytes.

Block RAM Configuration

From the Xilinx “Spartan-IIIE 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 \times 8.

Dual-Ported RAM

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.

Memory Usage

VGA is 640 \times 480 = 307200 pixels
Even at one bit per pixel, this requires 307200 \div 1024 = 300K bits = 37.5K bytes
Too much for on-chip RAM; we only have 8K max.

Memory Map

Memory is our most valuable resource. We’ve decided to allocate it as follows:
2.5K for a 80 \times 30 character array
1.5K for a 96-character 8 \times 16 font
That is half (4K) of the 8K on-chip memory Reasonable.
Also nice because we can use standard IBM console fonts.

Font

How about the font?
For 8 \times 16 characters, the 96 basic ASCII characters take
16 \times 96 = 1536 = 1.5K

Memory Usage

How about a text-only display?
80 \times 24 is typical.
640 \div 80 = 8 pixels horizontally/character
480 \div 24 = 20 pixels vertically/character
A bit too vertical. 8 \times 16 is more typical.
480 \div 16 = 30 lines
80 \times 30 = 2400 characters
Over 2K (2048). 2400 = 2048 + 352
Will fit in 2.5K = 2560
That 640 × 480 exactly fits an 80 × 30
8 × 16 font is no accident. Historically,

**Standard Resolution Font Display**

- CGA (1981) 640 × 200 8 × 8 80 × 25
- EGA (1984) 640 × 350 8 × 14 80 × 25
- EGA (1984) 640 × 350 8 × 8 80 × 43
- VGA (1987) 640 × 400 8 × 16 80 × 25
- VGA (1987) 640 × 400 8 × 14 80 × 26
- VGA (1987) 640 × 400 8 × 8 80 × 50
- VGA (1987) 640 × 480 8 × 16 80 × 30
- VGA (1987) 640 × 480 8 × 14 80 × 34
- VGA (1987) 640 × 480 8 × 8 80 × 60

**Microarchitecture**

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-IIIE 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: 1 / 3ns = 333 MHz
Highest data rate: 1 / 3.5ns = 285 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**

For a 25.175 MHz pixel clock,
HSYNC 96 pixels
BACK_PORCH 48
HACTIVE 640
FRONT_PORCH 16
HTOTAL 800

**Horizontal Timing**

For a 25.175 MHz pixel clock,
HSYNC 96 pixels
BACK_PORCH 48
HACTIVE 640
FRONT_PORCH 16
HTOTAL 800

**Vertical Timing**

For a 25.175 MHz pixel clock,
HSYNC 96 pixels
BACK_PORCH 48
HACTIVE 640
FRONT_PORCH 16
HTOTAL 800
**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>...</td>
<td>... ... ... ... ...</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>...</td>
<td>... ... ... ... ...</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>33</td>
<td>... ... ... ... ...</td>
</tr>
<tr>
<td>464</td>
<td>2320.0 2321.0 ... 2398.0 2399.0</td>
</tr>
<tr>
<td>...</td>
<td>... ... ... ... ...</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 \times 80</td>
</tr>
<tr>
<td>Column[6:0]</td>
<td>Column (0–79) = (Hcount – 141) \div 8</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>

**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) \div 16</td>
</tr>
<tr>
<td>BBLANK_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_S6_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></td>
<td>0 Reset DO</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>addr</td>
<td>data</td>
<td>0</td>
<td></td>
<td>0 Write</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>addr</td>
<td></td>
<td></td>
<td></td>
<td>[addr] Read</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>addr</td>
<td>data</td>
<td>data</td>
<td></td>
<td>Write-through</td>
</tr>
</tbody>
</table>

**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.

**Inputs**

| OPB_Clk | \rightarrow | \leftarrow | sl_DBus |
| OPB_Rst | \rightarrow | \leftarrow | sl_xferAck |
| OPB_ABus | \rightarrow | \leftarrow | sl_retry |
| OPB_BE | \rightarrow | \leftarrow | sl_toutSup |
| OPB_DBus | \rightarrow | \leftarrow | sl_errAck |
| OPB_RNW | \rightarrow | \leftarrow | OPB_select |

**Outputs**

<table>
<thead>
<tr>
<th>OPB_Clk</th>
<th>OPB_select</th>
<th>OPB_ABus</th>
<th>OPB_BE</th>
<th>OPB_DBus</th>
<th>OPB_RNW</th>
</tr>
</thead>
</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 \(\neq 0\) |
| sl_toutSup | Suppress slave time-out \(\neq 0\) |
| sl_errAck | Signal a transfer error occurred \(\neq 0\) |

**Typical OPB Read Cycle Timing**

![Typical OPB Read Cycle Timing Diagram]

**Typical OPB Write Cycle Timing**

![Typical OPB Write Cycle Timing Diagram]

OPB signals arrive late; DBus and xferAck needed early.
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_DWIDTH : integer := 32;
        C_OPB_CWIDTH : integer := 32;
        C_BASEADDR : std_logic_vector(31 downto 0) := X"FEFF1000";
        C_HIGHADDR : std_logic_vector(31 downto 0) := X"FEFF1FFF";
    )
    port ( 
        OPB_CLK : in std_logic;
        OPB_ABuse : in std_logic;
        OPB_RAM : in std_logic;
        OPB_seqAddr : in std_logic;
        OPB_select : in std_logic;
        OPB_RAMW : in std_logic;
        VGA_DBuse : out std_logic;
        VGA_RAMSelect : out std_logic;
        VGA_RAMRead : out std_logic;
        VGA_RAMWrite : out std_logic;
        VGA_RAMsramSelect : out std_logic;
        VGA_RAMsramRead : out std_logic;
        VGA_RAMsramWrite : out std_logic;
        VGA_RAMsramRowAddress : out std_logic;
        VGA_RAMsramColumnAddress : out std_logic;
        VGA_RAMsramPageAddress : out std_logic;
        VGA_RAMsramSelect : out std_logic;
        VGA_RAMsramRowSelect : out std_logic;
        VGA_RAMsramColumnSelect : out std_logic;
        VGA_RAMsramPageSelect : out std_logic;
        VGA_RAMsramDataIn : in std_logic;
        VGA_RAMsramDataOut : out std_logic;
    );

architecture Behavioral of opb_xsb300e_vga is
    constant BASEADDR : std_logic_vector(31 downto 0) := X"FEFF1000";
    constant HCOUNT : integer := 800;
    constant HSNSC : integer := 96;
    constant HACTIVE : integer := 640;
    constant HFRONT_PORCH : integer := 16;
    constant VCOUNT : integer := 525;
    constant VSNSC : integer := 2;
    constant VBACk_PORCH : integer := 33;
    constant VACTIVE : integer := 480;
    constant VFRONT_PORCH : integer := 10;

    -- Video signals
    signal Pixel_Clone : in std_logic;
    signal VIDOUT_RED : out std_logic;
    signal VIDOUT_GREEN : out std_logic;
    signal VIDOUT_BLUE : out std_logic;
    signal VIDOUT_HSYNC : out std_logic;
    signal VIDOUT_VSYNC : out std_logic;

    end opb_xsb300e_vga;

-- Latched input signals from the OPB
signal ABuse : std_logic_vector(31 downto 0);
signal DBuse : std_logic_vector(31 downto 0);
signal RAMW : std_logic;
signal select_delayed : std_logic;

-- Latched output data for the OPB
signal DBuse : 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);

-- Addresses and control for character RAM
signal LoadChar, CharColumn : std_logic_vector(9 downto 0);
signal Column : std_logic_vector(9 downto 0);
signal Row : std_logic_vector(9 downto 0);
signal EndOfLine, EndOfField : std_logic;

-- Master horizontal and vertical video counters
signal Hcount : std_logic_harbit(9 downto 0);
signal Vcount : std_logic_harbit(9 downto 0);
signal Hcount : std_logic_harbit(9 downto 0);
signal Vcount : std_logic_harbit(9 downto 0);
signal EndOfLine, EndOfField : std_logic;

-- Addresses and control for character RAM
signal LoadChar, CharColumn : std_logic_vector(9 downto 0);
signal Column : std_logic_vector(9 downto 0);
signal Row : std_logic_vector(9 downto 0);
signal EndOfLine, EndOfField : std_logic;

-- Video signals
signal Pixel_Clone : in std_logic;
 signal VIDOUT_RED : out std_logic;
 signal VIDOUT_GREEN : out std_logic;
 signal VIDOUT_BLUE : out std_logic;
 signal VIDOUT_HSYNC : out std_logic;
 signal VIDOUT_VSYNC : out std_logic;
 end opb_xsb300e_vga;
--- 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';

--- Video controller

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;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

--- Video controller

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';
    else
      end if;
    end if;
end process HSyncGen;

--- Video controller

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 HBlankGen;

--- Video controller

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;

--- Video controller

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 process VBlankGen;

--- Video controller

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" & "000000";

--- Video controller

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
                         "00111";

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";

CharColumn(9 downto 3);
VHDL: Shift Register

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 LoadNSRShift = '1' then
      ShiftData <= FontData;
    end if;
  end if;
end process ShiftRegister;

VideoData <= ShiftData(7);

VHDL: DAC output

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

The Punchline

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.