A Video Controller

CSEE W4840

Prof. Stephen A. Edwards

Columbia University

A Video Controller for the XESS

Project specification:
- Flexible video controller
- Runs with hardware on XESS board
  - Mostly the video DAC
- 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

The XESS XSB-300E contains an XC2S300E-6PQ208C FPGA.

From the Xilinx “Spartan-IIIE 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</td>
<td>16K</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 × 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 × 480 = 307 200 pixels

Even at one bit per pixel, this requires

307 200 ÷ 1024 = 300K bits = 37.5K bytes

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

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

Font

How about the font?

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

16 × 96 = 1536 = 1.5K

Memory Map

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 $640 \times 480$ exactly fits an $80 \times 30$
$8 \times 16$ 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>$640 \times 200$</td>
<td>$8 \times 8$</td>
<td>$80 \times 25$</td>
</tr>
<tr>
<td>EGA (1984)</td>
<td>$640 \times 350$</td>
<td>$8 \times 14$</td>
<td>$80 \times 25$</td>
</tr>
<tr>
<td>EGA (1984)</td>
<td>$640 \times 350$</td>
<td>$8 \times 8$</td>
<td>$80 \times 43$</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>$640 \times 400$</td>
<td>$8 \times 16$</td>
<td>$80 \times 25$</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>$640 \times 400$</td>
<td>$8 \times 14$</td>
<td>$80 \times 28$</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>$640 \times 400$</td>
<td>$8 \times 8$</td>
<td>$80 \times 50$</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>$640 \times 480$</td>
<td>$8 \times 16$</td>
<td>$80 \times 30$</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>$640 \times 480$</td>
<td>$8 \times 14$</td>
<td>$80 \times 34$</td>
</tr>
<tr>
<td>VGA (1987)</td>
<td>$640 \times 480$</td>
<td>$8 \times 8$</td>
<td>$80 \times 60$</td>
</tr>
</tbody>
</table>

Microarchitecture

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>$T_{BACK}$/</td>
<td>Setup/hold on ADDR</td>
<td>1.1ns</td>
<td>0</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

It is reasonable to assume the shift register, AND
gate, and controller will operate at full speed.

But how fast is the RAM?

Pixel-Level 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.

Start-of-line Detail

Vertical Timing

Horizontal Timing

For a 25.175 MHz pixel clock,
$HSYNC$ 96 pixels
$BACK_{PORCH}$ 48
$HACTIVE$ 640
$FRONT_{PORCH}$ 16
$HTOTAL$ 800
### 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>-</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>Write</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>addr</td>
<td>-</td>
<td>addr</td>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>addr</td>
<td>data</td>
<td>data</td>
<td>Write-through</td>
<td></td>
</tr>
</tbody>
</table>

### The video RAM interface

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

**Signal**  | **Value**
---|---
EN | 1 for read, 0 for hold
RST | 0
WE | 0
ADDR | read address
DI | 0 (unused)

### 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
- OPB_Rst
- OPB_ABus
- OPB_BE
- OPB_DBus
- OPB_seqAddr

**Outputs**
- sl_DBus
- sl_xferAck
- sl_retry
- sl_toutSup
- sl_errAck
- OPB_RNW
- OPB_SELECT
- OPB_seqAddr

### 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**: Clock input
- **OPB_SELECT**: Select input
- **OPB_ABus**: Address input
- **OPB_BE**: Byte enable input
- **OPB_RNW**: Read/write input
- **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 Write Cycle Timing

- **OPB_CLK**: Clock input
- **OPB_SELECT**: Select input
- **OPB_ABus**: Address input
- **OPB_BE**: Byte enable input
- **OPB_RNW**: Read/write input
- **OPB_DBus**: Data to slave
- **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)

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_OPBAlamat : integer := 32;
            C_OPBWD_WIDTH : integer := 32;
            C_BASEADDR : std_logic_vector(31 downto 0) := "X'FFEF0100";
            C_HIGADDR : std_logic_vector(31 downto 0) := "X'FFEF1FFF");
end entity opb_xsb300e_vga;

architecture Behavioral of opb_xsb300e_vga is
  constant BASEADDR : std_logic_vector(31 downto 0) := "X'FFEF0100";
  constant HSYNC : integer := 96;
  constant HBACK_PORCH : integer := 48;
  constant HACTIVE : integer := 640;
  constant VTOTAL : integer := 525;
  constant VBACK_PORCH : integer := 33;
  constant VACTIVE : integer := 480;
  constant VFRONT_PORCH : integer := 10;

  signal DBus_out : std_logic_vector (31 downto 0);
  signal MemCycle1, MemCycle2 : std_logic;
  signal VGA_DBus : out std_logic_vector(31 downto 0);
  signal DOUTB0, DOUTB1, DOUTB2, DOUTB3, DOUTB4
  signal VGA_xferAck : out std_logic;

  architecture Behavioral of opb_xsb300e_vga is
    constant BASEADDR : std_logic_vector(31 downto 0) := "X'FFEF0100";
    constant HSYNC : integer := 96;
    constant HBACK_PORCH : integer := 48;
    constant HACTIVE : integer := 640;
    constant VTOTAL : integer := 525;
    constant VBACK_PORCH : integer := 33;
    constant VACTIVE : integer := 480;
    constant VFRONT_PORCH : integer := 10;

    signal DBus_out : std_logic_vector (31 downto 0);
    signal MemCycle1, MemCycle2 : std_logic;
    signal VGA_DBus : out std_logic_vector(31 downto 0);
    signal DOUTB0, DOUTB1, DOUTB2, DOUTB3, DOUTB4
    signal VGA_xferAck : out std_logic;

    -- 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 : std_logic;
    signal 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 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
    signal ReadData : std_logic_vector(7 downto 0);
VHDL: Video signals 2

VHDL: BRAM component

-- 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;

begin -- body of architecture
RAMB4_S8_S8_0 : RAMB4_S8_S8
port map ( DOA => DOUT5,
ADDRA => ABus(8 downto 0),
CLKA => OPB_CLK,
DIA => DBus(7 downto 0),
ENA => '1',
RSTA => '1',
WEA => '1',
DOB => DOUTB5,
ADDRB => FontAddr(8 downto 0),
CLKB => Pixel_Clock,
DIB => X'00',
ENB => '1',
RSTB => CharRamSelect_N(0),
WEB => '0');
end process MemCycleFSM;

MemCycleFSM : process(OPB_CLK, OPB_Rst)
begin
if OPB_Rst = '1' then
  RamSelect <= "00000000" when ChipSelect = '1' and
  MemCycle1 <= '0' and MemCycle2 = '0' else '0';
end if;
end process MemCycleFSM;

VGA_xferAck <= MemCycle2; -- OPB output
WE <= RamSelect when ChipSelect = '1' and
RST <= not RamSelect when ChipSelect = '1' and
  MemCycle2 = '0' else "00000000";
end if;
end process GenDout;

VGA_Dbus <= Dout;
VHDL: Video HCounter

-- 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';

VHDL: Video VCounter

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';

VHDL: Hsync

HSyncGen : process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    VIDOUT_HSYNC_N <= '0';
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    if EndOfLine = '1' then
      VIDOUT_HSYNC_N <= '0';
    elsif Hcount = HSYNC - 1 then
      VIDOUT_HSYNC_N <= '1';
    end if;
    if EndOfField = '1' then
      VIDOUT_VSYNC_N <= '0';
    elsif Vcount = VSYNC - 1 then
      VIDOUT_VSYNC_N <= '1';
    end if;
  end if;
end process HSyncGen;

VHDL: VBlank

VBlankGen : 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 then
      HBLANK_N <= '1';
    elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
      HBLANK_N <= '0';
    end if;
  end if;
end process VBlankGen;

VHDL: Video Timing, addresses

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');

VHDL: Font RAM I/O

FontRamPage <= FontAddr(11 downto 9);
CharRamSelect_N <= "1110" when CharRamPage = "00" else
                    "1101" when CharRamPage = "01" else
                    "1011" when CharRamPage = "10" else
                    "0111" when CharRamPage = "11" else
                    "1111";
FontData <= DOUTB5 or DOUTB6 or DOUTB7;

VHDL: character RAM I/O

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);
**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 LoadNShift = '1' then
      ShiftData <= FontData;
    else
      ShiftData <= ShiftData(6 downto 0) & '0';
    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_BLANK_N <= '0';
    VIDOUT_RED <= "0000000000";
    VIDOUT_GREEN <= "0000000000";
    VIDOUT_BLUE <= "0000000000";
  elsif Pixel_Clock'event and Pixel_Clock = '1' then
    VIDOUT_BLANK_N <= VBLANK_N and HBLANK_N;
    if VideoData = '1' then
      VIDOUT_RED <= "1111111111";
      VIDOUT_GREEN <= "1111111111";
      VIDOUT_BLUE <= "1111111111";
    else
      VIDOUT_RED <= "0000000000";
      VIDOUT_GREEN <= "0000000000";
      VIDOUT_BLUE <= "0000000000";
    end if;
  end if;
end process VideoOut;

VIDOUT_CLK <= Pixel_Clock;
end Behavioral; -- end of architecture