

A Video Controller CSEE W4840

Prof. Stephen A. Edwards

Columbia University

A Video Controller – p. 16

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.

A Video Controller – p. 26

On-Chip RAM

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

From the Xilinx "Spartan-II E 1.8V FPGA Family" data sheet, we find

Device	CLB array	CLBs	Distributed RAM Bits	Block RAM Bits
XC2S300E	32 × 48	1536	98 304	64K

Distributed RAM uses the LUTs and is awkward.

Block RAM is dual-ported.

We have at most 8K bytes.

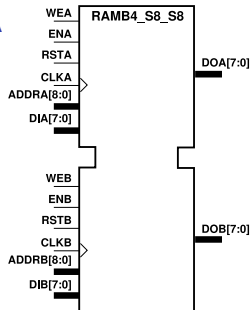
A Video Controller – p. 36

Block RAM Configuration

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 .



A Video Controller – p. 46

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.

A Video Controller – p. 56

Memory Usage

VGA is $640 \times 480 = 307\,200$ pixels

Even at one bit per pixel, this requires $307\,200 \div 1024 = 300\text{K bits} = 37.5\text{K bytes}$

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

A Video Controller – p. 66

Memory Usage

How about a text-only display?

80×24 is typical.

$640 \div 80 = 8$ pixels horizontally/character

$480 \div 24 = 20$ pixels vertically/character

A bit too vertical. 8×16 is more typical.

$480 \div 16 = 30$ lines

$80 \times 30 = 2400$ characters

Over 2K (2048). $2400 = 2048 + 352$

Will fit in $2.5\text{K} = 2560$

Font

How about the font?

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

$16 \times 96 = 1536 = 1.5\text{K}$

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.

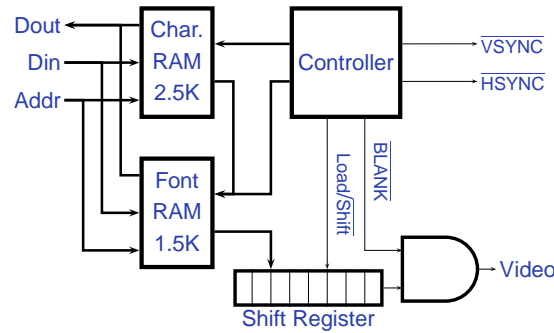
Aside

That 640×480 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	80×28
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

A Video Controller - p. 106

Microarchitecture



A Video Controller - p. 116

Timing

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:

Symbol	Description	Min	Max
T_{BCKO}	CLK to DOUT	0.6ns	3.5ns
T_{BACK}/T_{BCKA}	Setup/hold on ADDR	1.1ns/0	-
T_{BPWH}	Clock pulse width high	1.5ns	-
T_{BPWL}	Clock pulse width low	1.5ns	-

A Video Controller - p. 126

Timing

Maximum clock frequency: $1 / 3\text{ns} = 333\text{ MHz}$

Highest data rate: $1 / 3.5\text{ns} = 285\text{ MHz}$

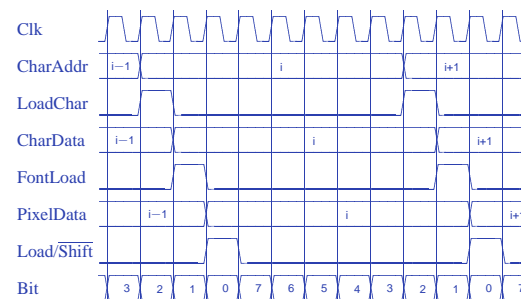
VGA dot clock is 25 MHz

The RAM is much, much faster than we need.

We can assume data from the RAM appears "quickly" after the clock.

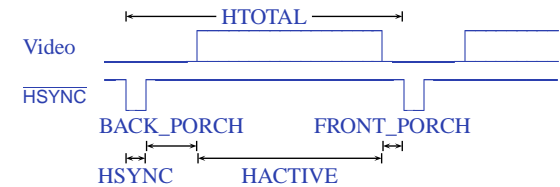
A Video Controller - p. 136

Pixel-Level Timing



A Video Controller - p. 146

Horizontal Timing

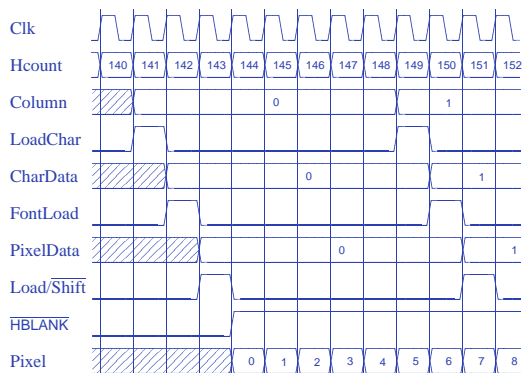


For a 25.175 MHz pixel clock,

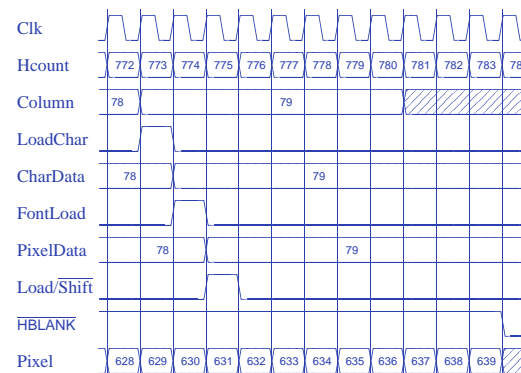
HSYNC	96 pixels
BACK_PORCH	48
HACTIVE	640
FRONT_PORCH	16
HTOTAL	800

A Video Controller - p. 156

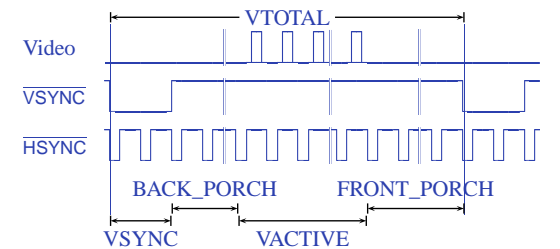
Start-of-line Detail



End-of-line detail



Vertical Timing



VSYNC	2 lines
BACK_PORCH	33
VACTIVE	480
FRONT_PORCH	10
VTOTAL	525

Character Addresses

line	character address, row				
0	0,0	1,0	...	78,0	79,0
1	0,1	1,1	...	78,1	79,1
...
14	0,14	1,14	...	78,14	79,14
15	0,15	1,15	...	78,15	79,15
16	80,0	81,0	...	158,0	159,0
17	80,1	81,1	...	158,1	159,1
...
31	80,15	81,15	...	158,15	159,15
32	160,0	161,0	...	238,0	239,0
...
464	2320,0	2321,0	...	2398,0	2399,0
...
479	2320,15	2321,15	...	2398,15	2399,15

A Video Controller - p. 196

Horizontal Signals

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

A Video Controller - p. 206

Vertical Signals

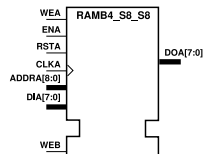
Signal	Description
Vcount[9:0]	Vertical position (0-524)
Row[4:0]	Row number (0-29) = (Vcount - 35) ÷ 16
VBLANK_N	Vertical blanking. Off lines 35-514
VSYNC_N	Vertical Synchronization. Lines 0 and 1

A Video Controller - p. 216

Interface to the RAMB4_S8_S8

From the Xilinx libraries guide:

EN	RST	WE	CLK	ADDR	DI	DO	Operation
0	-	-	-	-	-	DO	No-op
1	1	0	↑	-	-	0	Reset DO
1	1	1	↑	addr	data	0	Write
1	0	0	↑	addr	-	[addr]	Read
1	0	1	↑	addr	data	data	Write-through



A Video Controller - p. 226

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)

A Video Controller - p. 236

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		Outputs	
OPB_Clk	→	←	sl_DBus
OPB_Rst	→	←	sl_xferAck
OPB_ABus	→	←	sl_retry
OPB_BE	→	←	sl_toutSup
OPB_DBus	→	←	sl_errAck
OPB_RNW	→		
OPB_select	→		
OPB_seqAddr	→		

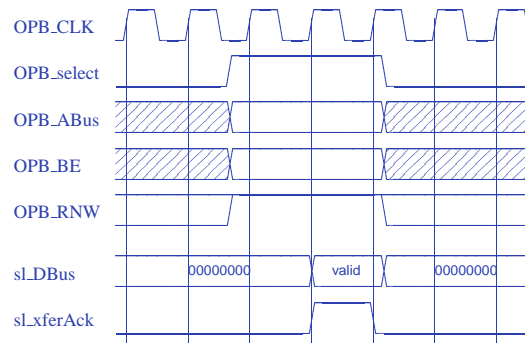
A Video Controller - p. 246

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)

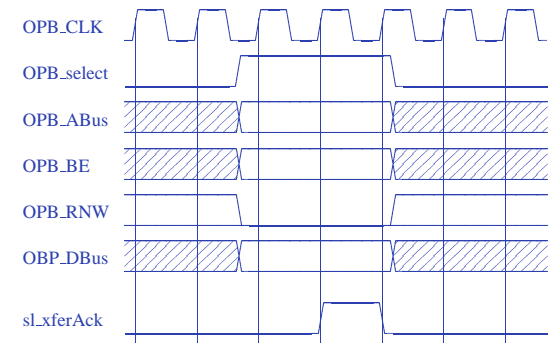
sl_DBus	Data from slave. Must be 0 when inactive
sl_xferAck	Transfer acknowledge. OPB_select→0
sl_retry	Request master to retry operation (=0)
sl_toutSup	Suppress slave time-out (=0)
sl_errAck	Signal a transfer error occurred (=0)

Typical OPB Read Cycle Timing

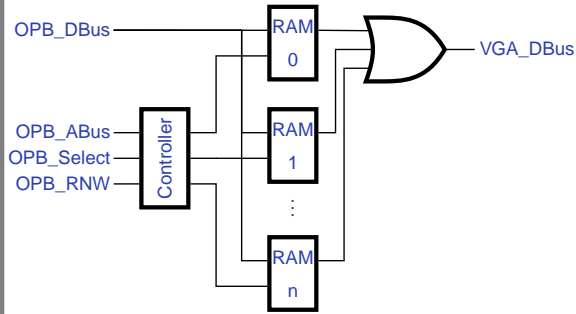


OPB signals arrive late; DBus and xferAck needed early.

Typical OPB Write Cycle Timing

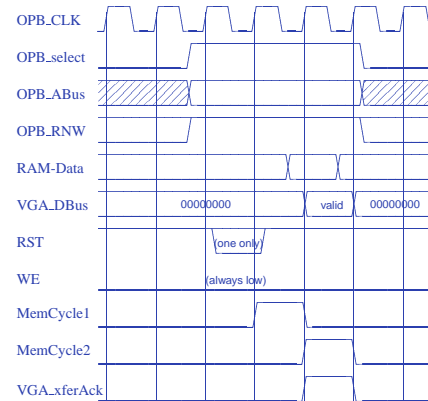


OPB interface block diagram



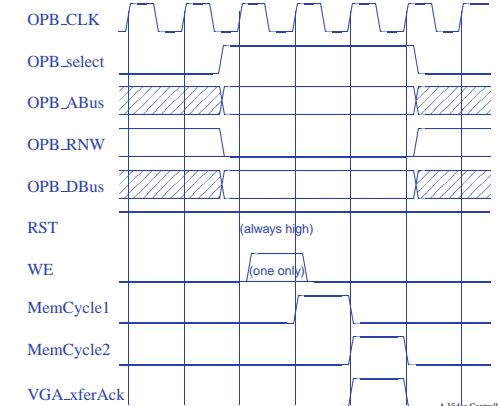
A Video Controller - p. 286

OPB-RAM Read Timing



A Video Controller - p. 286

OPB-RAM Write Timing



A Video Controller - p. 306

VHDL: Entity generics

```
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"
);
```

A Video Controller - p. 316

VHDL: OPB Ports

```
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;
```

A Video Controller - p. 326

VHDL: Video ports

```
-- Video signals
Pixel_Clock : in std_logic; -- 25 MHz
VIDOUT_CLK  : out std_logic;
VIDOUT_RED  : out std_logic_vector(9 downto 0);
VIDOUT_GREEN : out std_logic_vector(9 downto 0);
VIDOUT_BLUE : out std_logic_vector(9 downto 0);
VIDOUT_BLANK_N : out std_logic;
VIDOUT_HSYNC_N : out std_logic;
VIDOUT_VSYNC_N : out std_logic
);

end opb_xsb300e_vga;
```

A Video Controller - p. 336

VHDL: Architecture constants

```
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;
```

VHDL: OPB signals

```
-- 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);
```

VHDL: Video signals 1

```
-- 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);
```


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' t
  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';
```

A Video Controller - p. 466

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' t
  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';
```

A Video Controller - p. 476

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' t
  if EndOfLine = '1' then
    VIDOUT_HSYNC_N <= '0';
  elsif Hcount = HSYNC - 1 then
    VIDOUT_HSYNC_N <= '1';
  end if;
end if;
end process HSyncGen;
```

A Video Controller - p. 486

VHDL: HBlank

```
HBlankGen : process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    HBLANK_N <= '0';
  elsif Pixel_Clock'event and Pixel_Clock = '1' t
  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 HBlankGen;
```

A Video Controller - p. 496

VHDL: VSync

```
VSyncGen : process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    VIDOUT_VSYNC_N <= '0';
  elsif Pixel_Clock'event and Pixel_Clock = '1' t
  if EndOfLine = '1' then
    if EndOfField = '1' then
      VIDOUT_VSYNC_N <= '0';
    elsif Vcount = VSYNC - 1 then
      VIDOUT_VSYNC_N <= '1';
    end if;
  end if;
end if;
end process VSyncGen;
```

A Video Controller - p. 506

VHDL: VBlank

```
VBlankGen : process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    VBLANK_N <= '0';
  elsif Pixel_Clock'event and Pixel_Clock = '1' t
  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;
```

A Video Controller - p. 516

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: 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: Font RAM I/O

```
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;
```

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);
```

A Video Controller - p. 556

VHDL: DAC output

```
VideoOut: process (Pixel_Clock, OPB_Rst)
begin
  if OPB_Rst = '1' then
    VIDOUT_BLANK_N <= '0';
    VIDOUT_RED <= "0000000000";
    VIDOUT_BLUE <= "0000000000";
    VIDOUT_GREEN <= "0000000000";
  elsif Pixel_Clock'event and Pixel_Clock = '1'
    and VBLANK_N = '0' and HBLANK_N = '0' then
    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
```

A Video Controller - p. 566