California State University Northridge

ECE 524 - Spring 2023

VGA

Saba Janamian

Agenda

VGA overview

Assignment overview

Video Graphics Array (VGA)

VGA – Video Graphics Array

Video display standard introduced in the late 1980's

VGA – Video Graphics Array

Widely supported by PC graphics hardware and monitors

Used initially with the CRT (cathode ray tube) monitors

Later adopted for LCD (liquid-crystal display) monitors as well

VGA – Characteristic Features

Resolution: 640x480

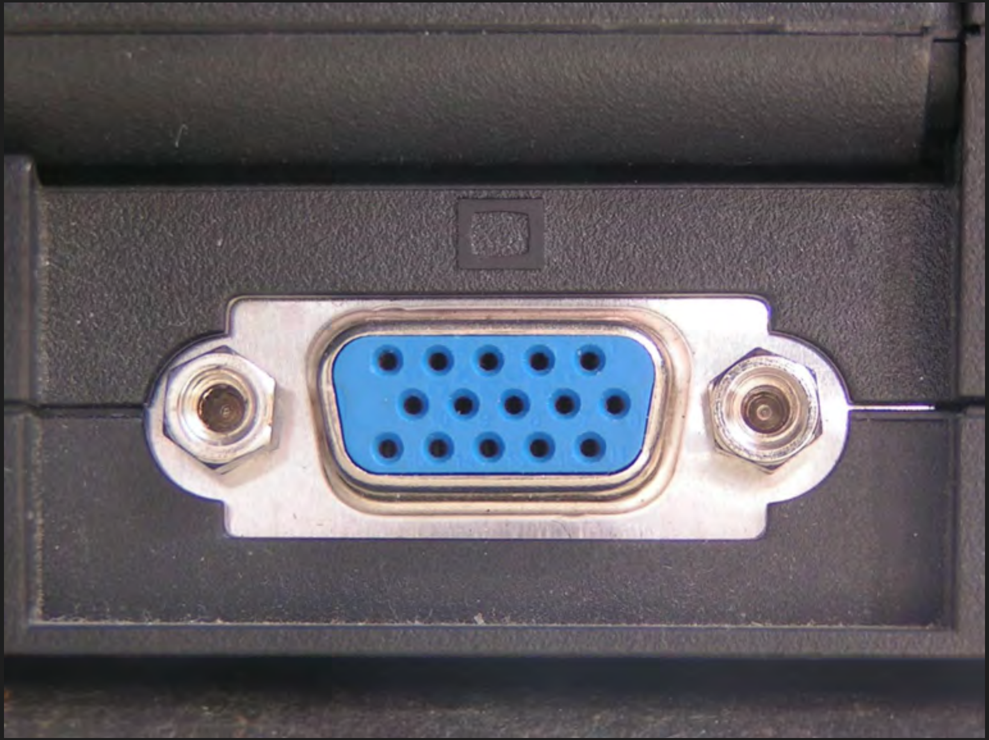
Refresh Rate: 25Hz, 30Hz, 60Hz (frames / second)

Display: up to 256 colors (8 bits)

RGB: Red, Green, Blue analog signals

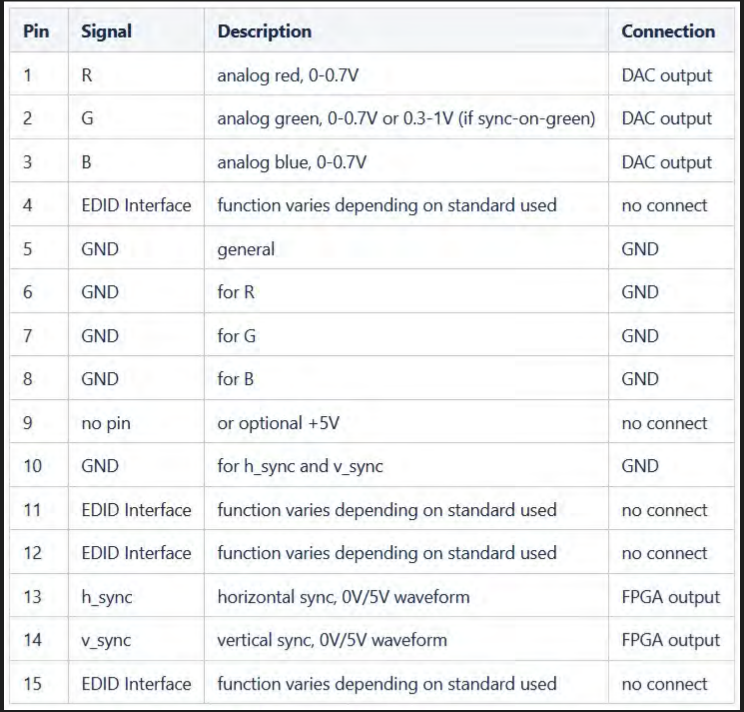
VGA connector

A female DE-15 output in a laptop computer

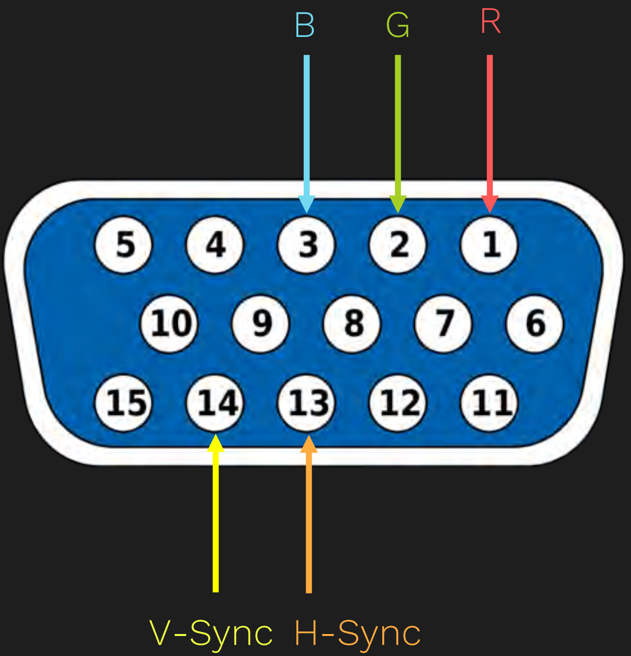


https://en.wikipedia.org/wiki/VGA_connector#/media/File:SVGA_port.jpg

VGA Pinout



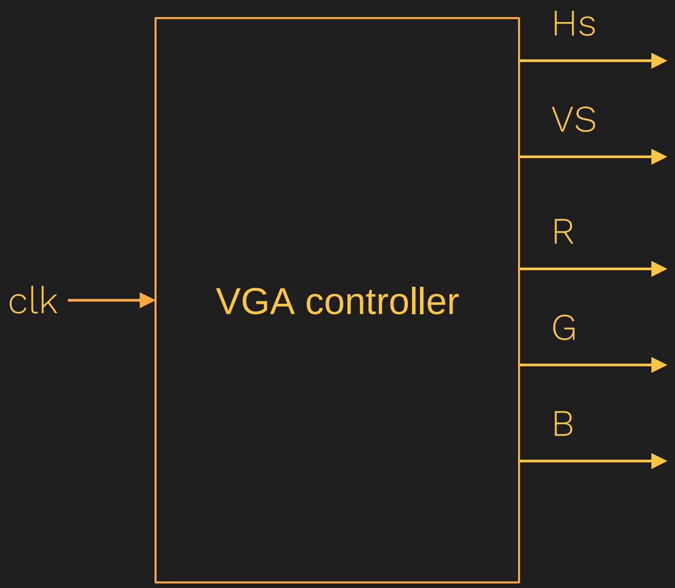
Pin	Signal	Description	Connection
1	R	analog red, 0-0.7V	DAC output
2	G	analog green, 0-0.7V or 0.3-1V (if sync-on-green)	DAC output
3	B	analog blue, 0-0.7V	DAC output
4	EDID Interface	function varies depending on standard used	no connect
5	GND	general	GND
6	GND	for R	GND
7	GND	for G	GND
8	GND	for B	GND
9	no pin	or optional +5V	no connect
10	GND	for h_sync and v_sync	GND
11	EDID Interface	function varies depending on standard used	no connect
12	EDID Interface	function varies depending on standard used	no connect
13	h_sync	horizontal sync, 0V/5V waveform	FPGA output
14	v_sync	vertical sync, 0V/5V waveform	FPGA output
15	EDID Interface	function varies depending on standard used	no connect



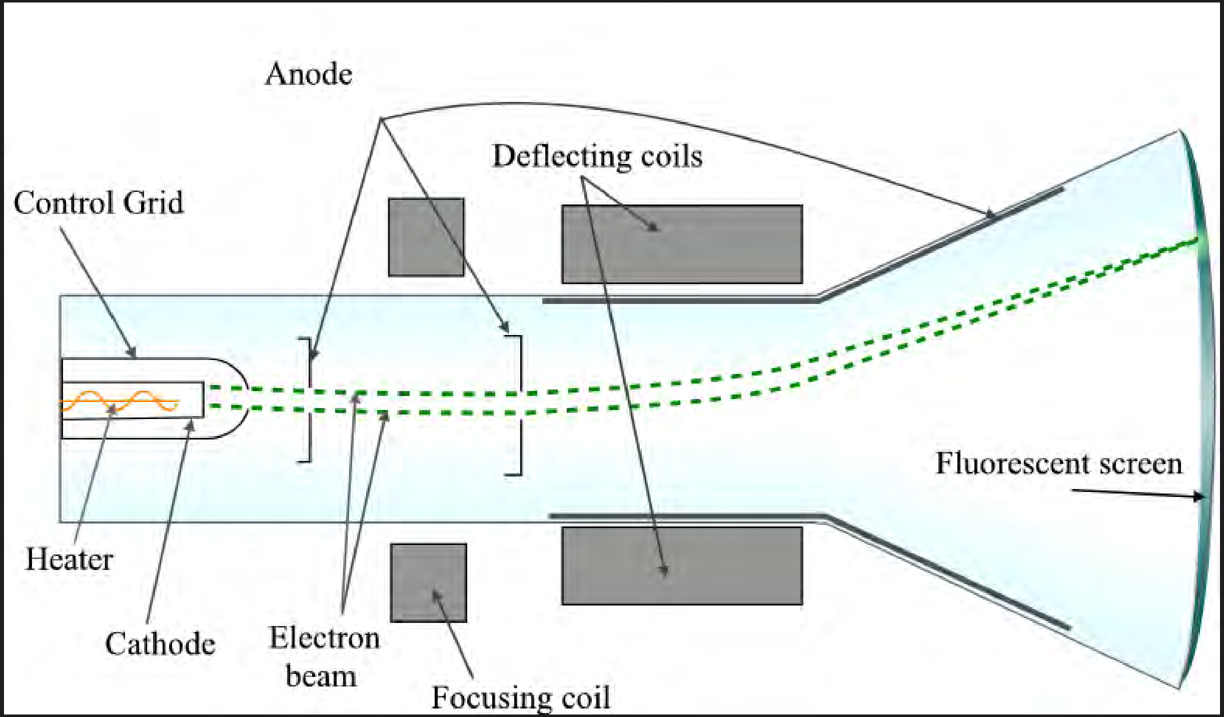
https://en.wikipedia.org/wiki/VGA_connector#/media/File:VGA_Port.svg
https://forum.digikey.com/t/vga-controller-vhdl/12794

VGA Controller Ports

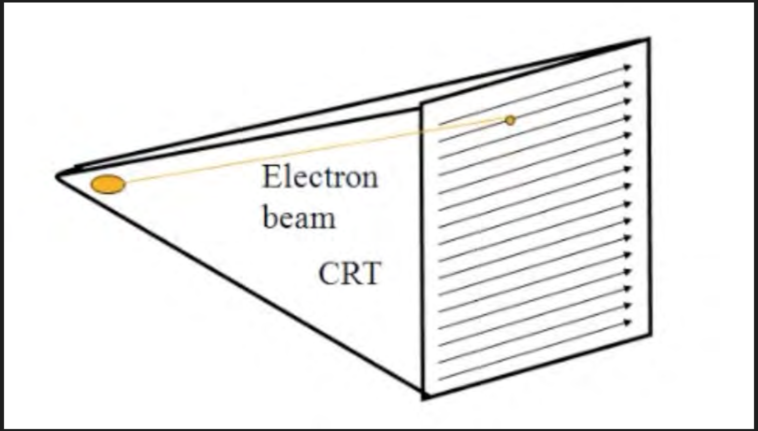
```
entity top is
   Port ( CLK_I : in  STD_LOGIC;
          VGA_HS_O : out  STD_LOGIC;
          VGA_VS_O : out  STD_LOGIC;
          VGA_R : out  STD_LOGIC_VECTOR (3 downto 0);
          VGA_B : out  STD_LOGIC_VECTOR (3 downto 0);
          VGA_G : out  STD_LOGIC_VECTOR (3 downto 0));
end top;
```

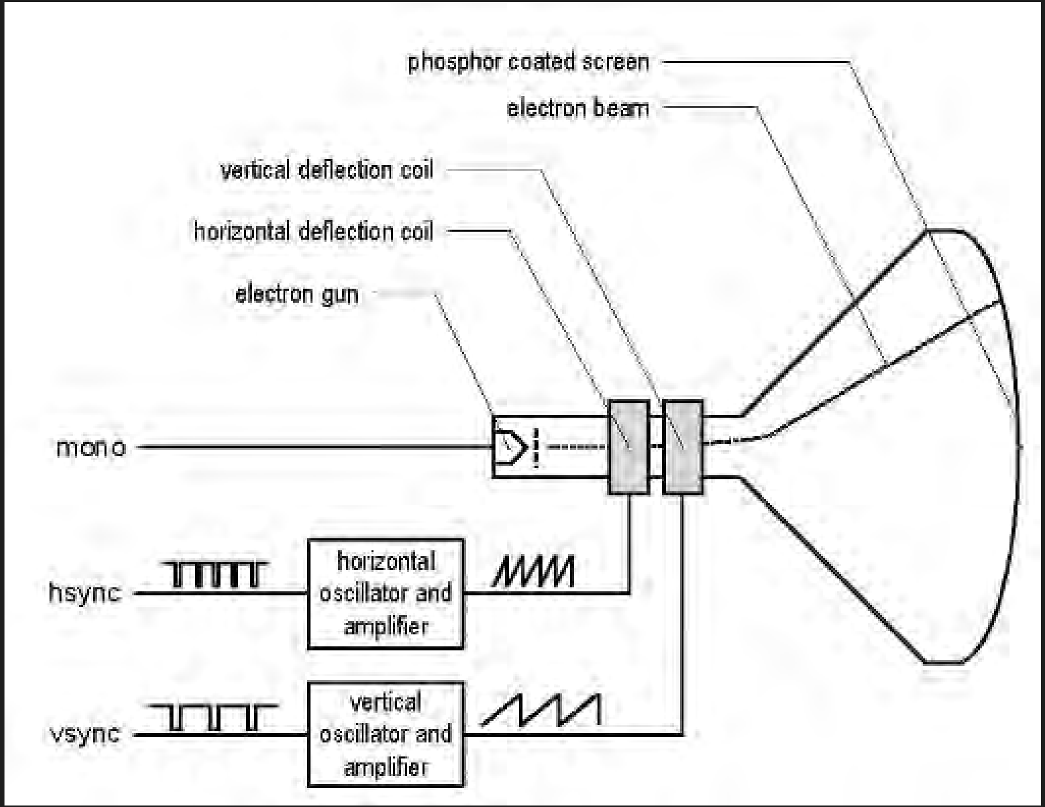


CRT Monitor – Conceptual Diagram

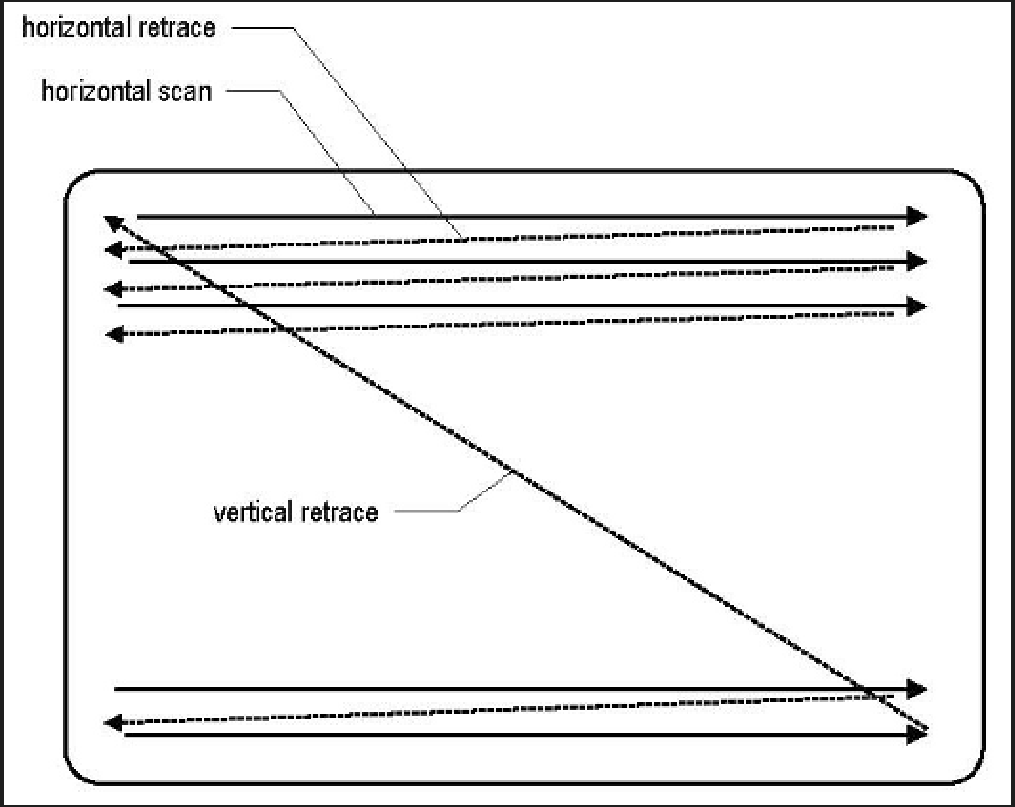


CRT Monitor – Conceptual Diagram



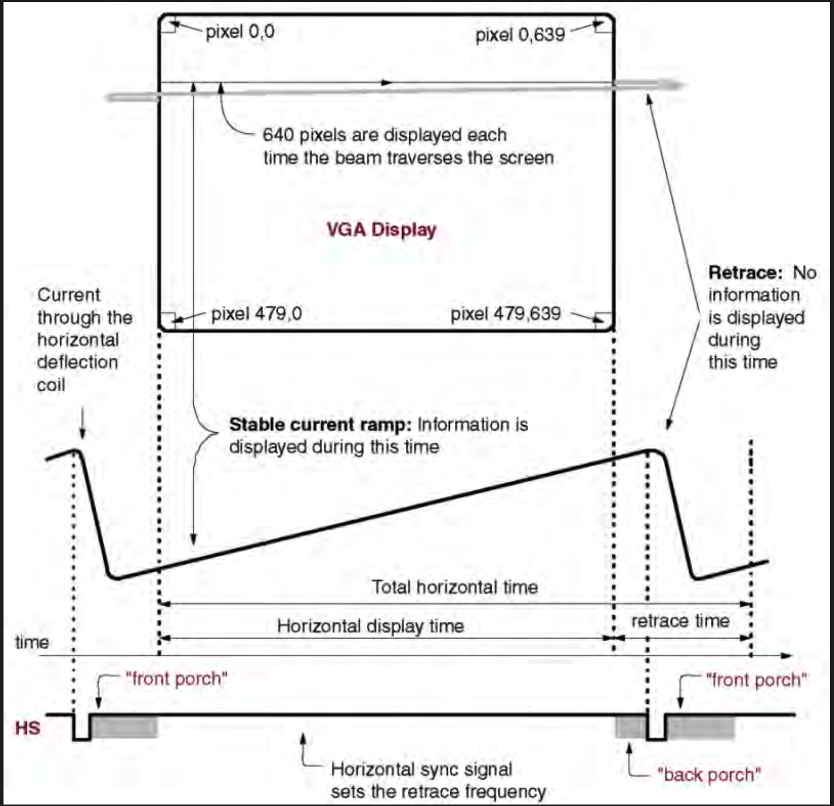

phosphor coated screen
electron beam
vertical deflection coil
horizontal deflection coil
electron gun
mono
hsync
horizontal oscillator and amplifier
vsync
vertical oscillator and amplifier

CRT Monitor – Scanning Pattern



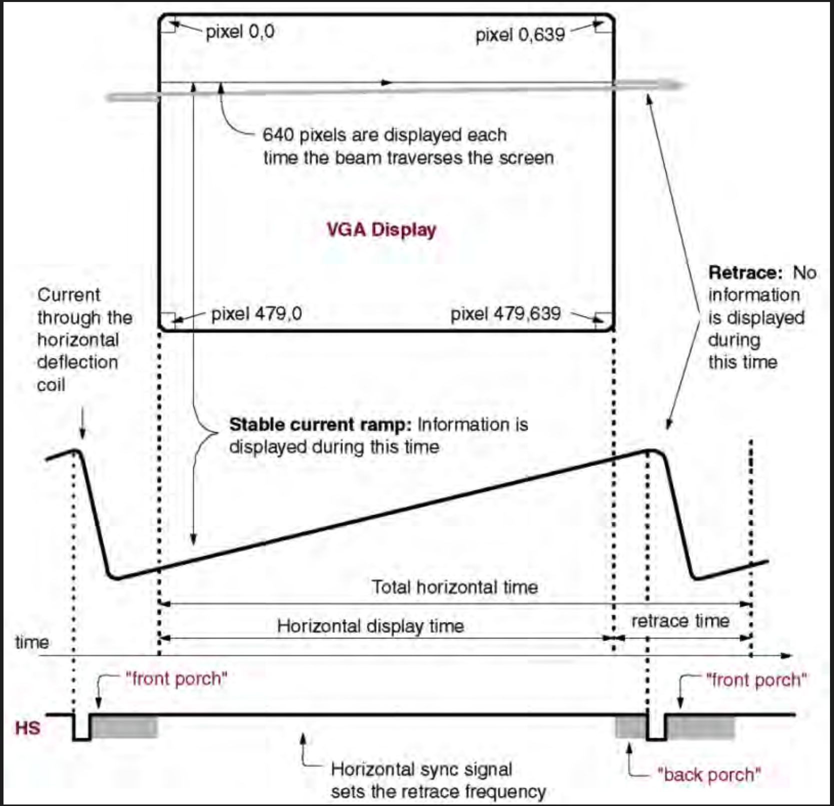
CRT Display Timing Example

The protocol involves "scanning" the screen and using two signals called hsync and vsync to synchronize the exact location on the screen where cursor is ready to draw.


pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
pixel 479,0
pixel 479,639
Current through the horizontal deflection coil
Retrace: No information is displayed during this time
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
"front porch"
HS
"front porch"
Horizontal sync signal sets the retrace frequency
"back porch"

CRT Display Timing Example

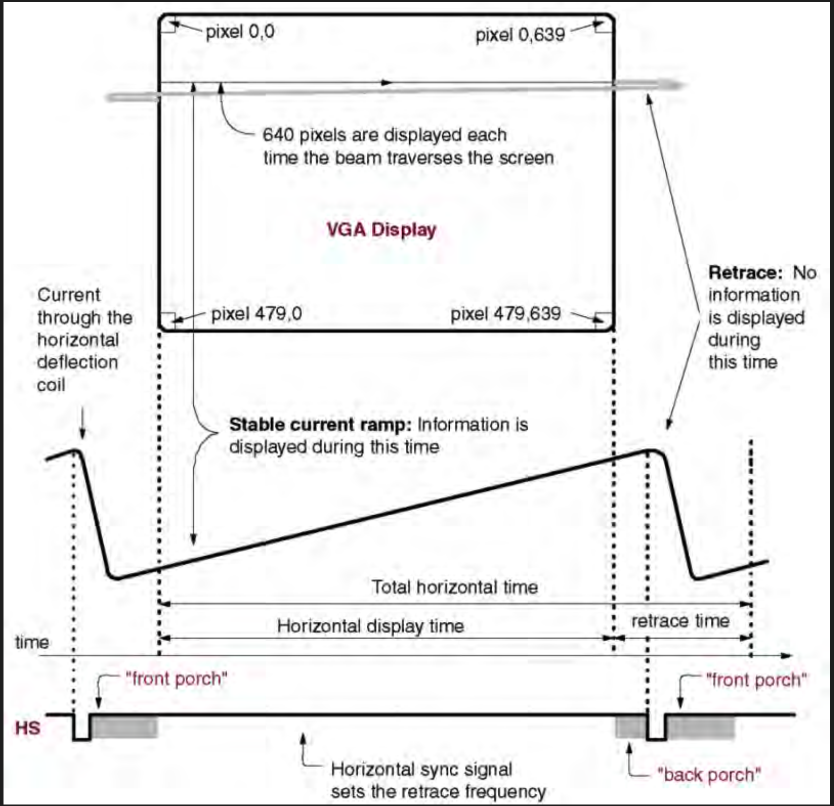
A VGA Driver is typically nothing more than some logic that drives horizontal and vertical sync signals on and off so the scan gun behind your monitor knows when to start shooting the next row or column.



pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
Retrace: No information is displayed during this time
Current through the horizontal deflection coil
pixel 479,0
pixel 479,639
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
"front porch"
"front porch"
HS
Horizontal sync signal sets the retrace frequency
"back porch"

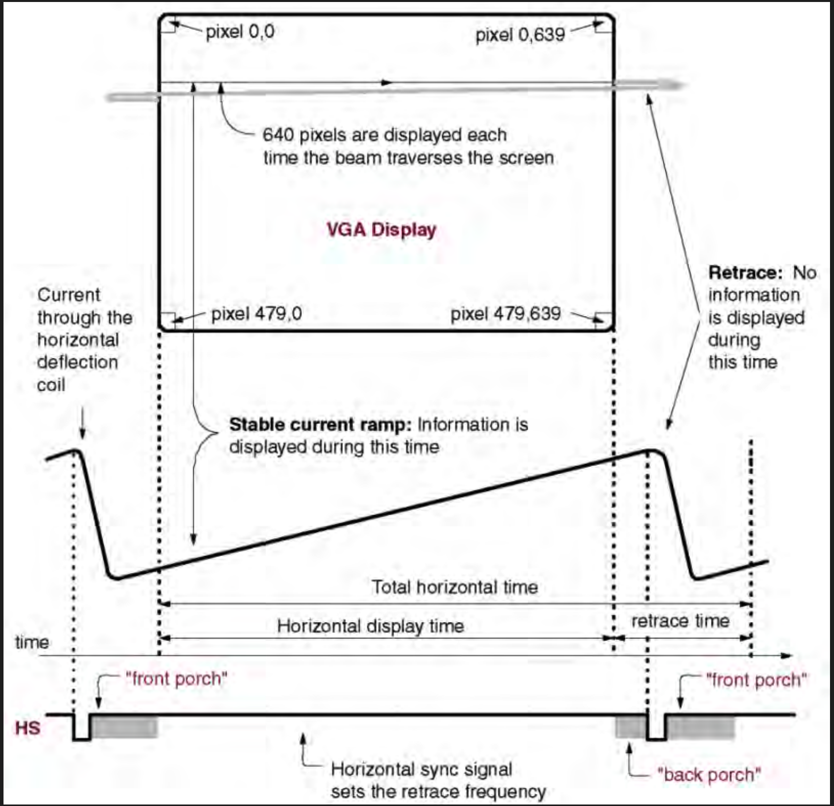
VGA Timing

At the start of a line, the horizontal sync signal drops to a logical low for a certain amount of time.


pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
Current through the horizontal deflection coil
pixel 479,0
pixel 479,639
Retrace: No information is displayed during this time
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
"front porch"
"front porch"
HS
Horizontal sync signal sets the retrace frequency
"back porch"

VGA Timing

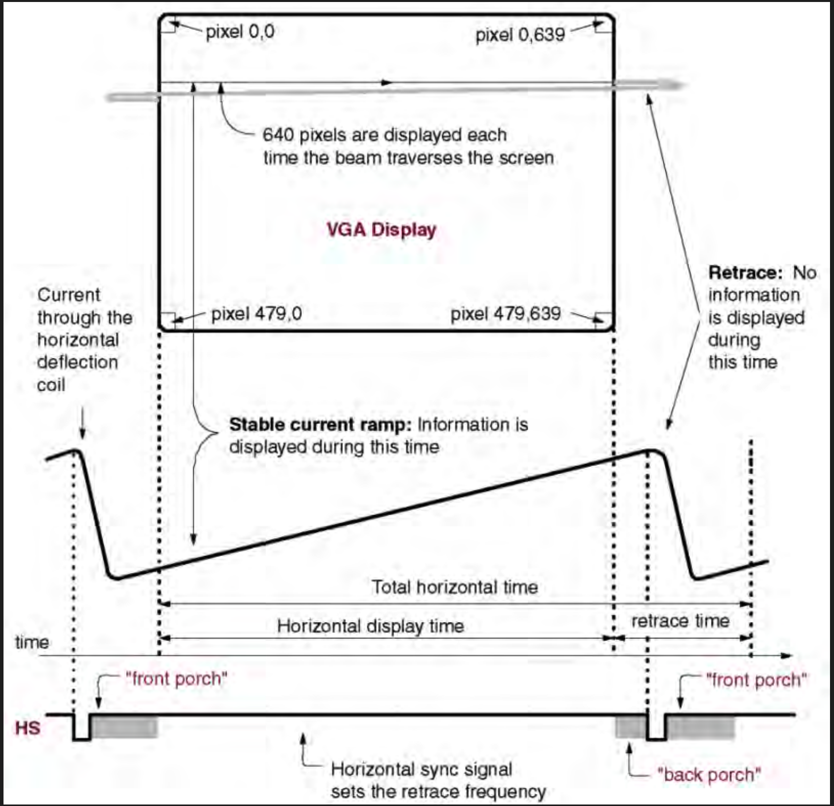
The horizontal sync signal then gets pulsed to a logical high, but at this point, the VGA scan gun is not ready to draw onto the screen; this is called the front porch.



pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
Retrace: No information is displayed during this time
Current through the horizontal deflection coil
pixel 479,0
pixel 479,639
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
"front porch"
"front porch"
HS
Horizontal sync signal sets the retrace frequency
"back porch"

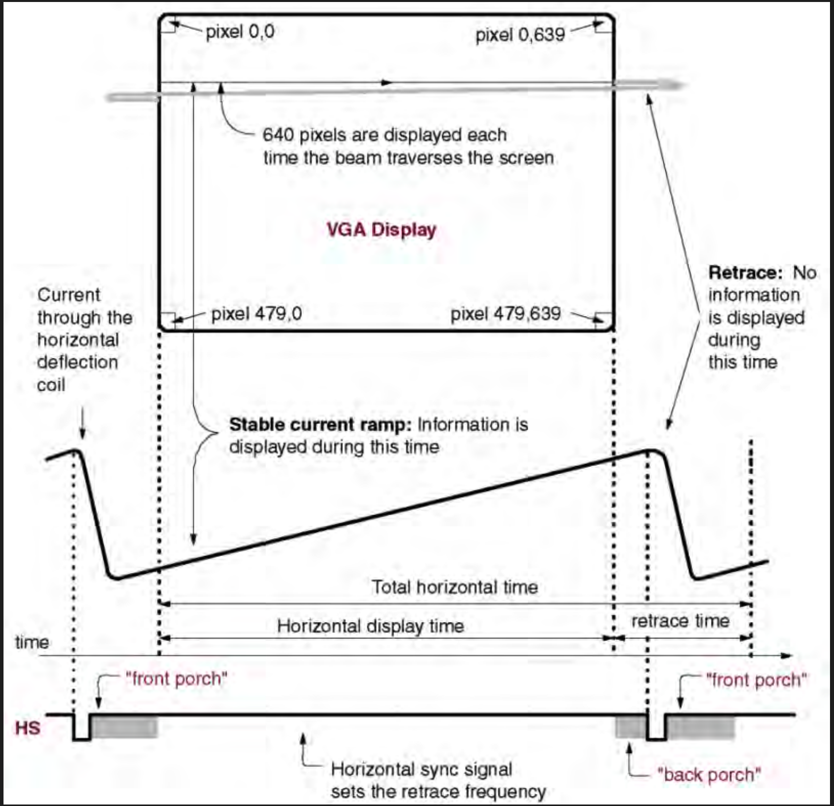
VGA Timing

After a specified front porch period, the screen is ready to display onto the screen.


pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
Retrace: No information is displayed during this time
Current through the horizontal deflection coil
pixel 479,0
pixel 479,639
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
"front porch"
"front porch"
HS
Horizontal sync signal sets the retrace frequency
"back porch"

VGA Timing

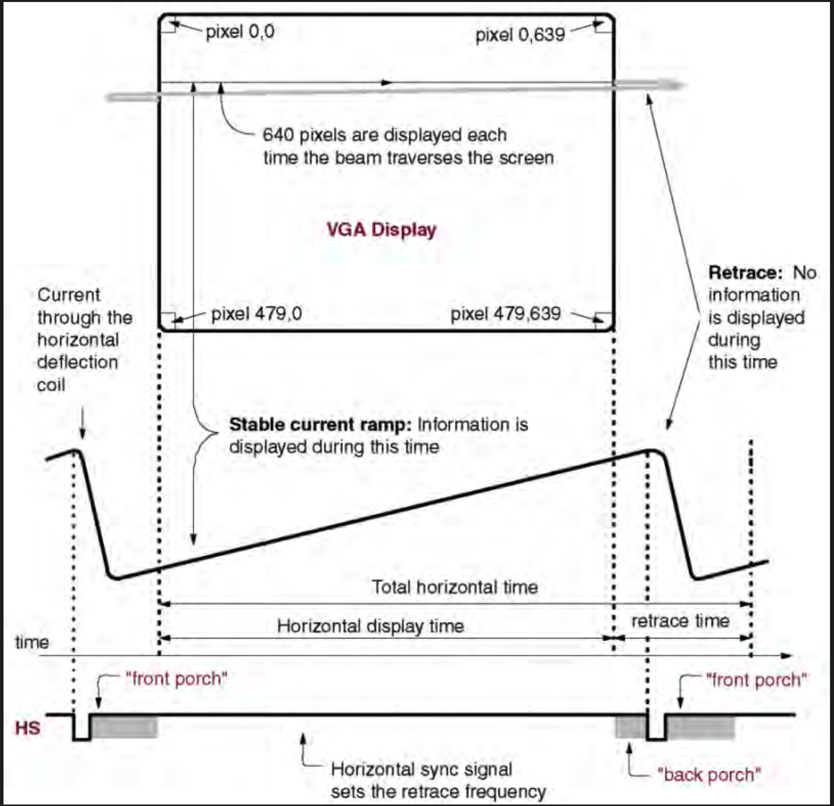
The horizontal sync allows for 640 distinct pixels to be displayed before going into the back porch stage.



pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
pixel 479,0
pixel 479,639
Current through the horizontal deflection coil
Retrace: No information is displayed during this time
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
"front porch"
"front porch"
HS
Horizontal sync signal sets the retrace frequency
"back porch"

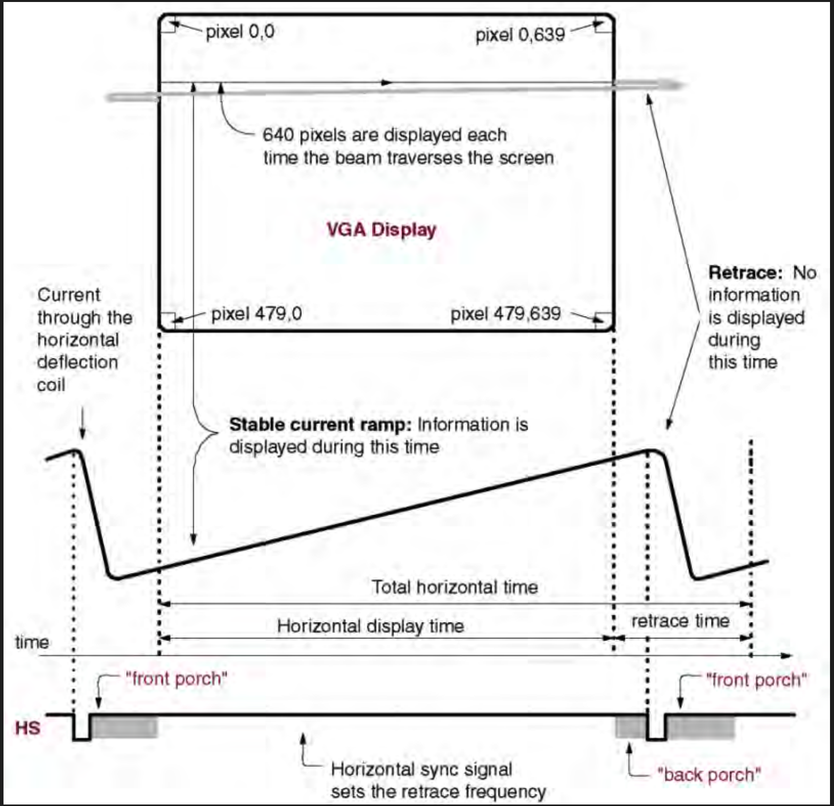
VGA Timing

In the back porch stage, the VGA scan gun has actually passed the viewable portion of the screen, and cannot paint to the screen.


pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
Retrace: No information is displayed during this time
Current through the horizontal deflection coil
pixel 479,0
pixel 479,639
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
"front porch"
"front porch"
HS
Horizontal sync signal sets the retrace frequency
"back porch"

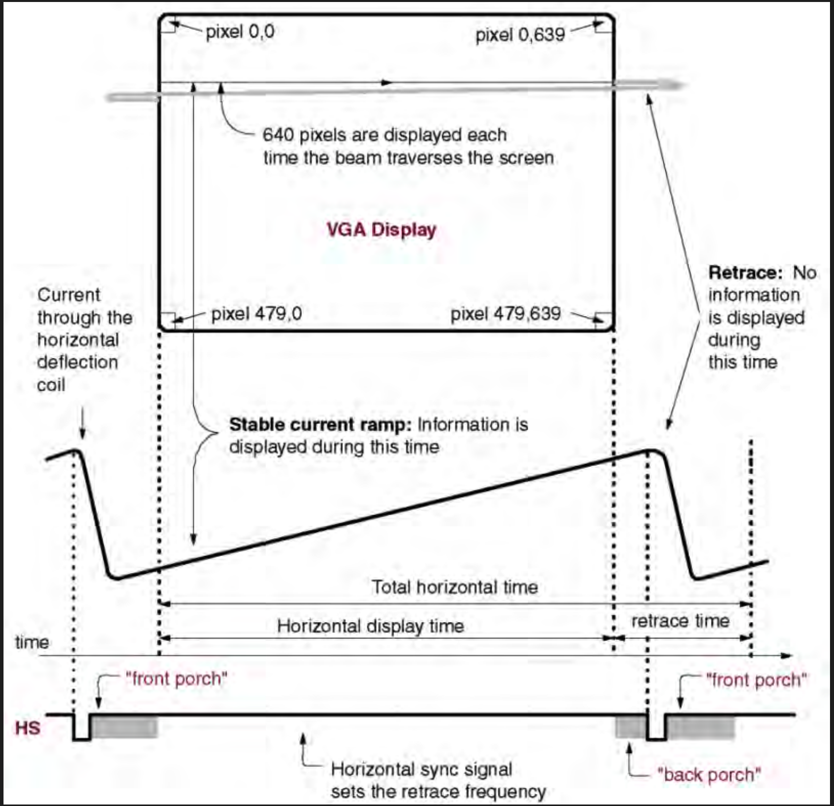
VGA Timing

Finally, the pulse is dropped again, and the process is repeated.


pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
Retrace: No information is displayed during this time
Current through the horizontal deflection coil
pixel 479,0
pixel 479,639
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
"front porch"
"front porch"
HS
Horizontal sync signal sets the retrace frequency
"back porch"

VGA Timing

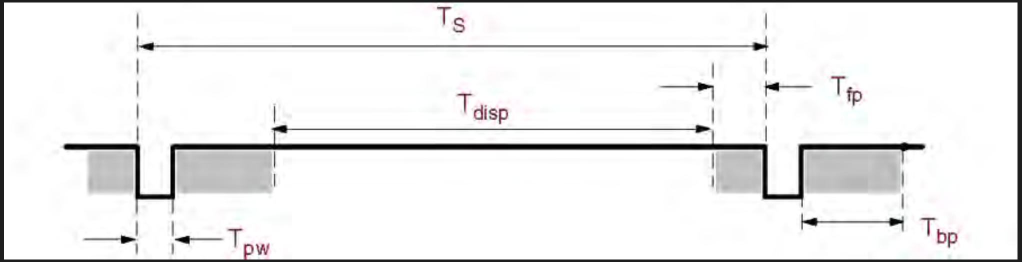
Your first (and most difficult) task will be to capture the VGA Driver behavior as sequential logic.

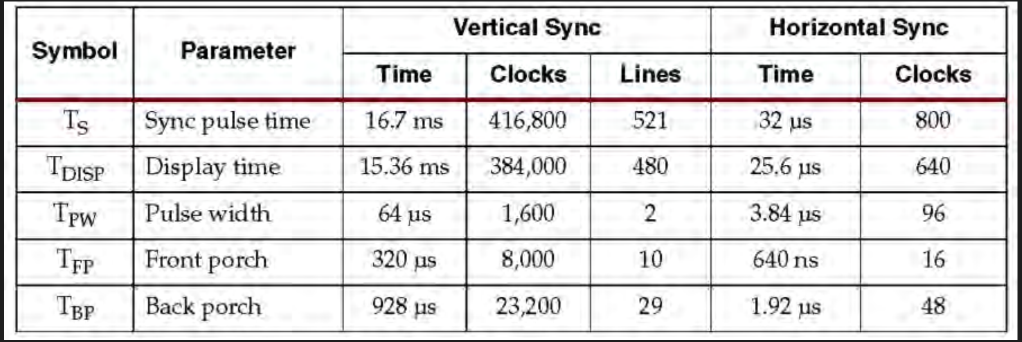


pixel 0,0
pixel 0,639
640 pixels are displayed each time the beam traverses the screen
VGA Display
pixel 479,0
pixel 479,639
Retrace: No information is displayed during this time
Current through the horizontal deflection coil
Stable current ramp: Information is displayed during this time
Total horizontal time
Horizontal display time
retrace time
time
"front porch"
"front porch"
HS
Horizontal sync signal sets the retrace frequency
"back porch"

VGA Timing

Both hsync and vsync, you need to be able to implement timing diagram in figure below.

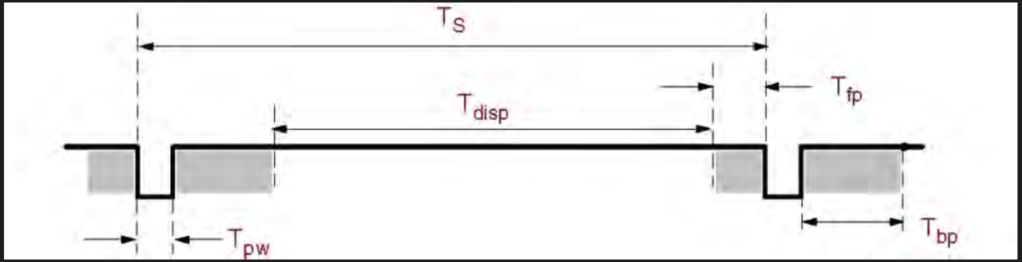


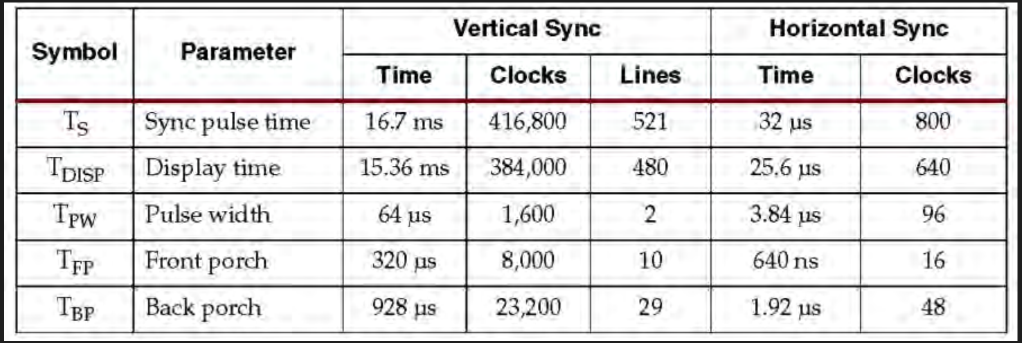


Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48

VGA Timing

Tpw: Time of the pulse width: How long to keep the signal low

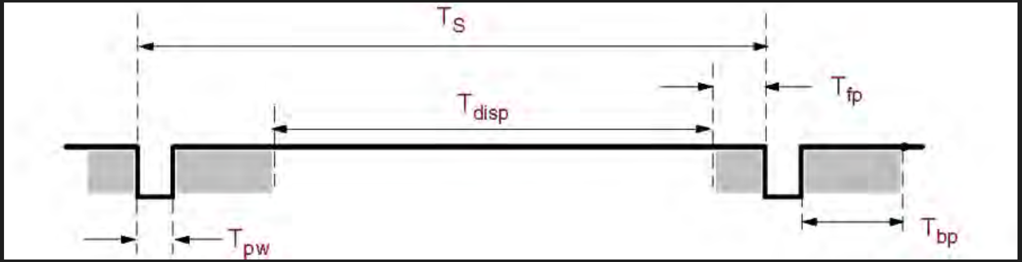


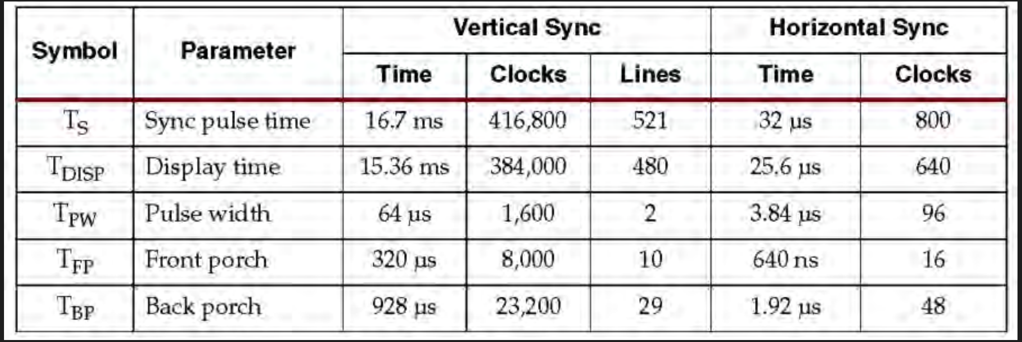


Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48

VGA Timing

Ts : Time period: One complete row/column of information

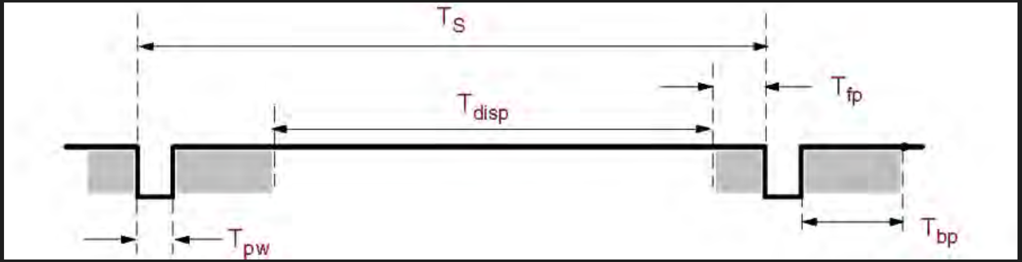


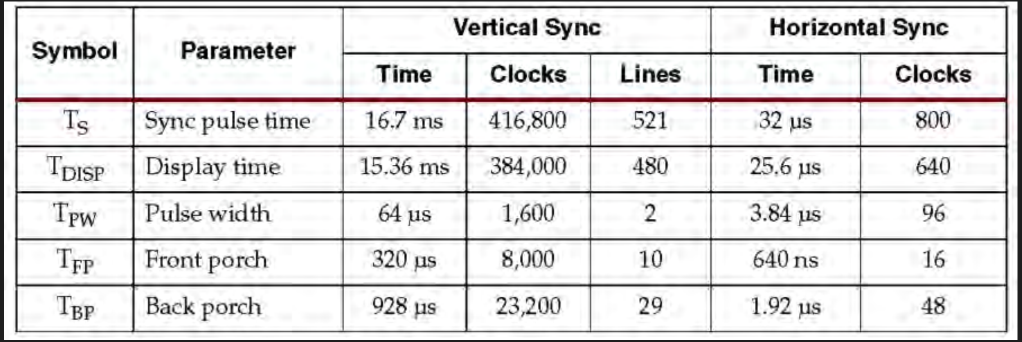


Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48

VGA Timing

Tdisp : Time of Display: This is the time we can actually display information

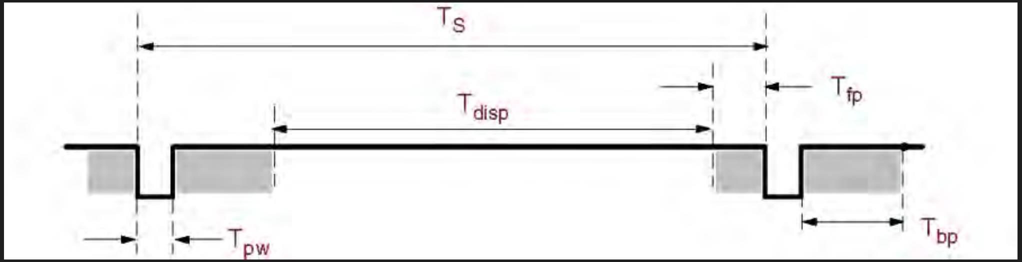


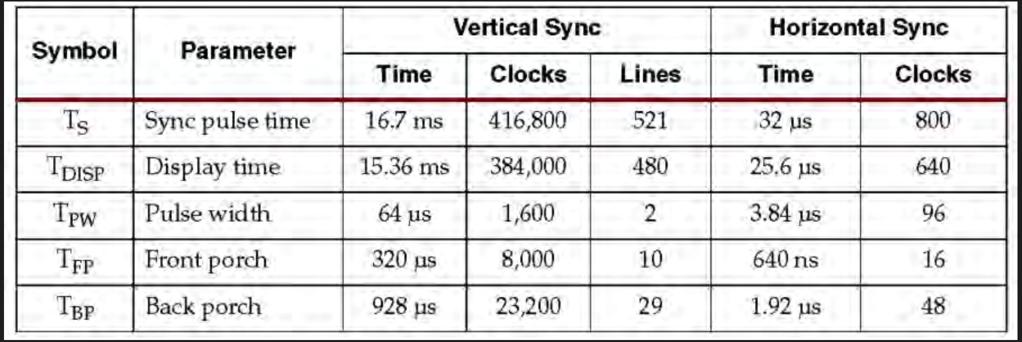


Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48

VGA Timing

Tfp: Time front porch: The amount of time for the front porch

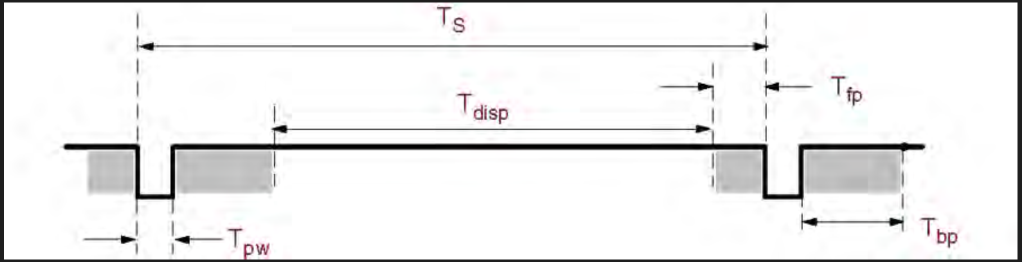


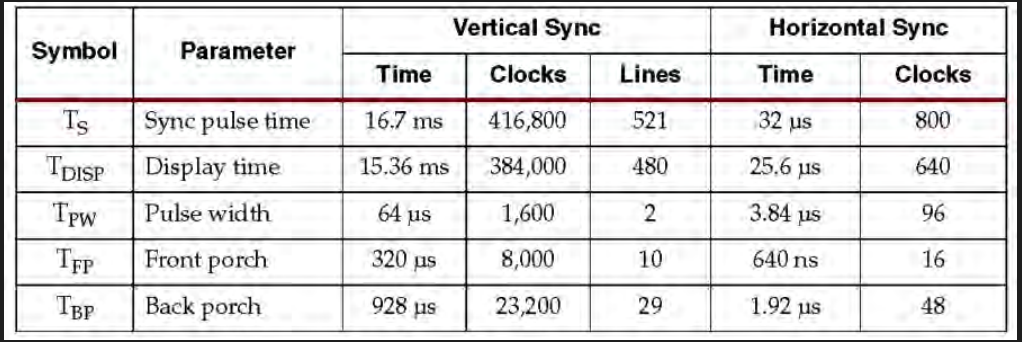


Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48

VGA Timing

Tbp: Time back porch: The amount of time for the back porch





Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7 ms	416,800	521	32 µs	800
T_{DISP}	Display time	15.36 ms	384,000	480	25.6 µs	640
T_{PW}	Pulse width	64 µs	1,600	2	3.84 µs	96
T_{FP}	Front porch	320 µs	8,000	10	640 ns	16
T_{BP}	Back porch	928 µs	23,200	29	1.92 µs	48

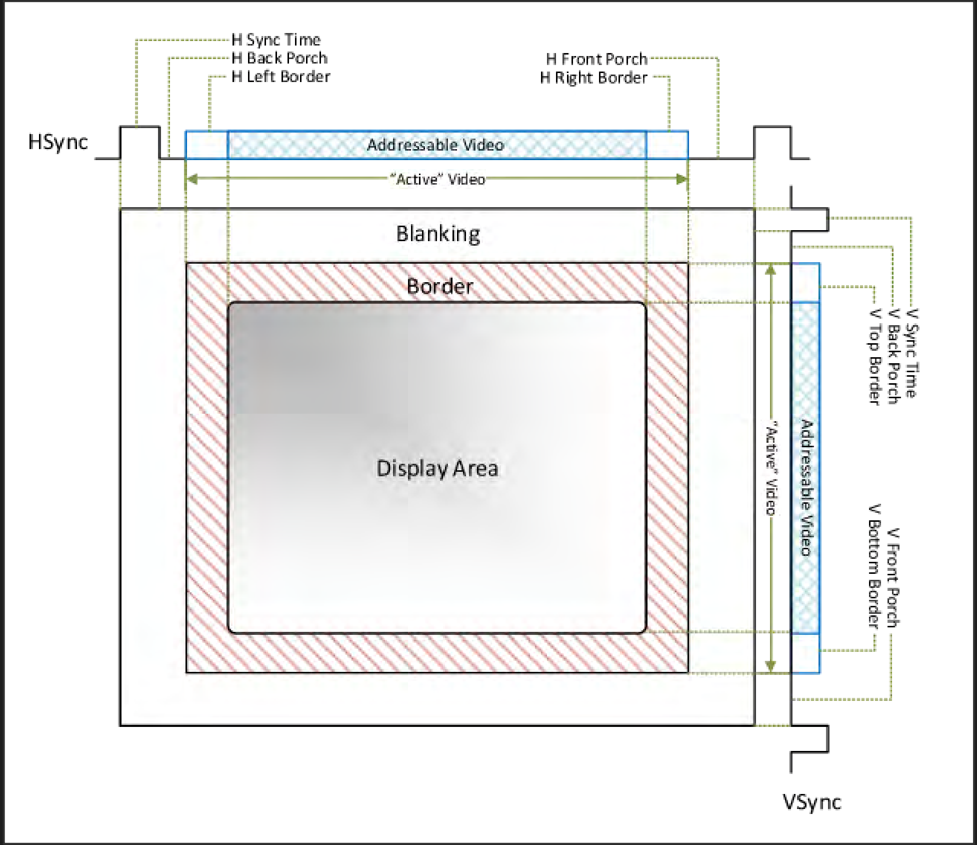
VGA Synchronization

The video synchronization circuit generates the hsync signal, which specifies the required time to traverse (scan) a row, and the vsync signal, which specifies the required time to traverse (scan) the entire screen.

VGA Synchronization

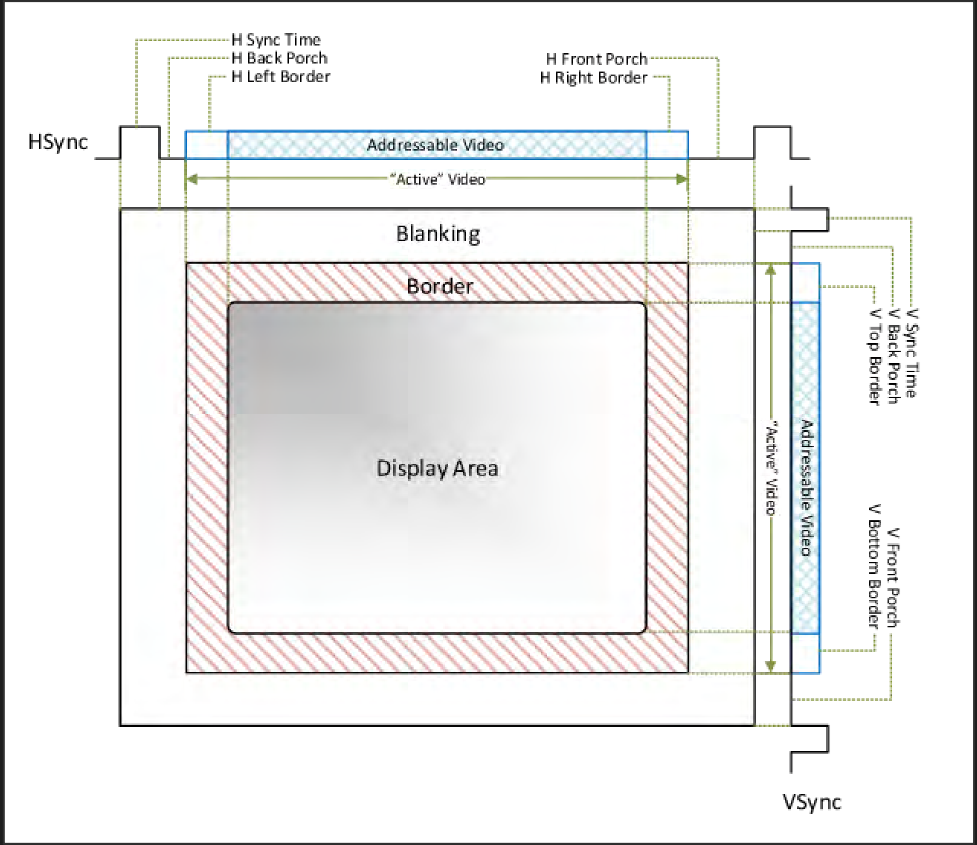
The screen of a CRT monitor usually includes a small black border, as shown at the top of Figure.

The middle rectangle is the visible portion.



VGA Synchronization

Note that the coordinate of the vertical axis increases downward. The coordinates of the top-left and bottom-right corners are (0,0) and (639,479), respectively.



Horizontal synchronization

A period of the hsync signal contains 800 pixels and can be divided into four regions

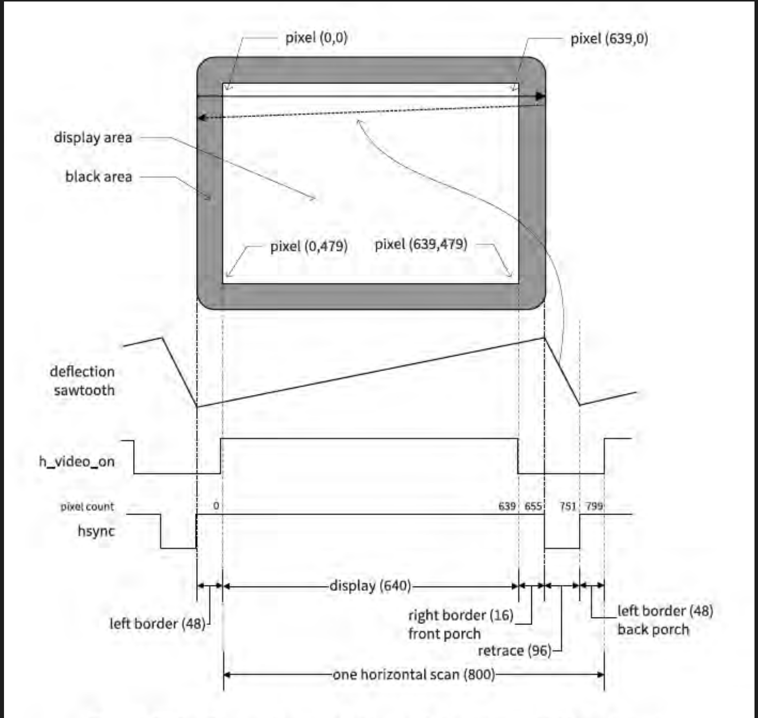


Figure 20.6 Timing diagram of a horizontal scan (with VGA resolution).

Horizontal synchronization

Display: region where the pixels are actually displayed on the screen. The length of this region is 640 pixels

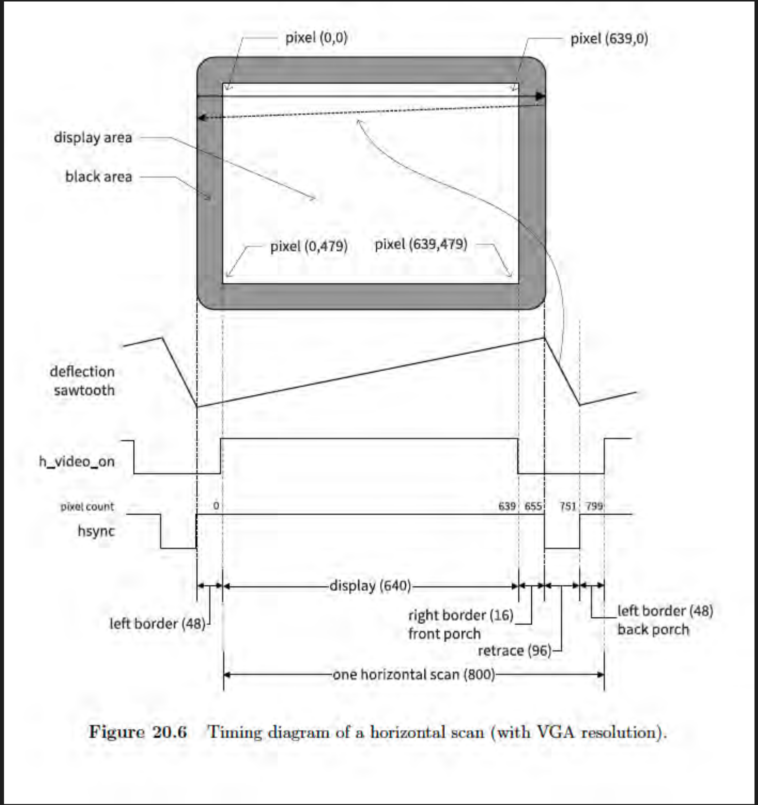


Figure 20.6 Timing diagram of a horizontal scan (with VGA resolution).

Horizontal synchronization

Retrace: region in which the electron beams return to the left edge.

The video signal should be disabled (i.e., black), and the length of this region is 96 pixels

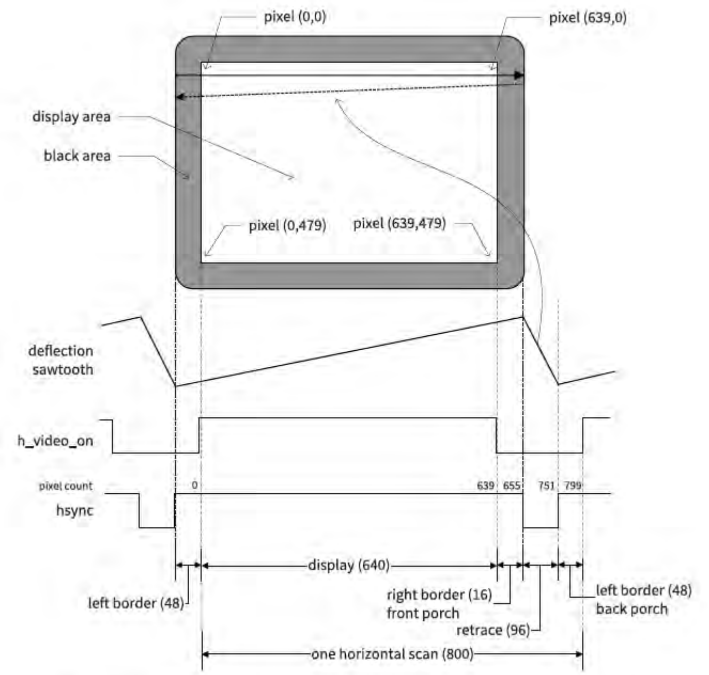


Figure 20.6 Timing diagram of a horizontal scan (with VGA resolution).

Horizontal synchronization

Right border: region that forms the right border of the display region. It is also known as the front porch (i.e., porch before retrace).

The video signal should be disabled, and the length of this region is 16 pixels.

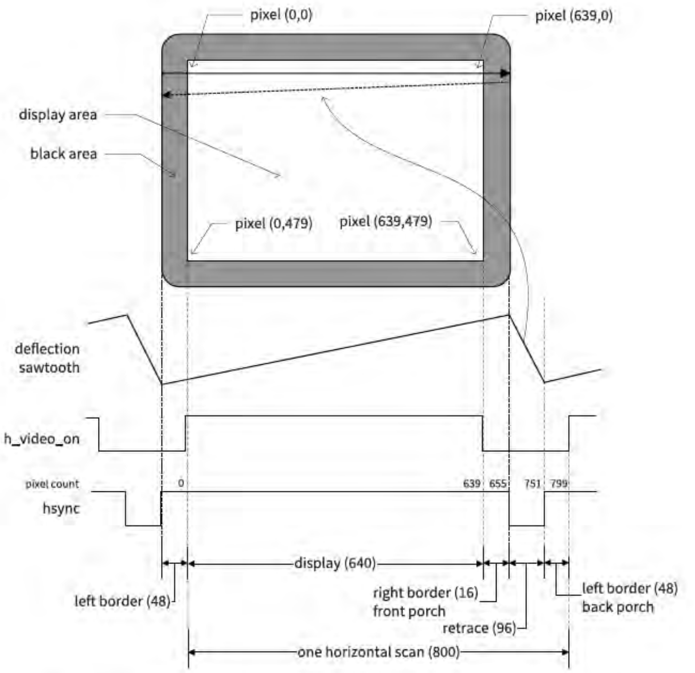


Figure 20.6 Timing diagram of a horizontal scan (with VGA resolution).

Horizontal synchronization

Left border: region that forms the left border of the display region. It is also known as the backporch (i.e., porch after retrace).

The video signal should be disabled, and the length of this region is 48 pixels.

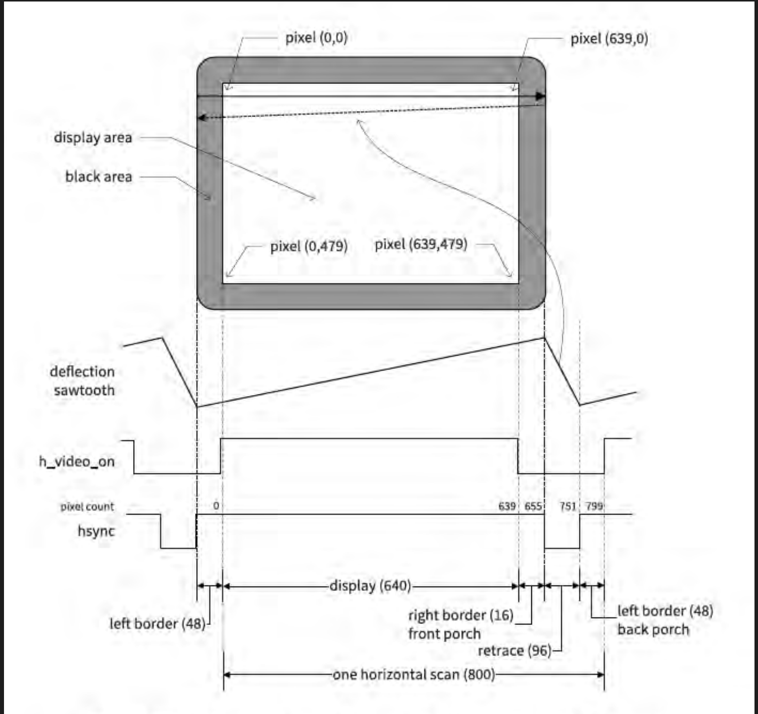
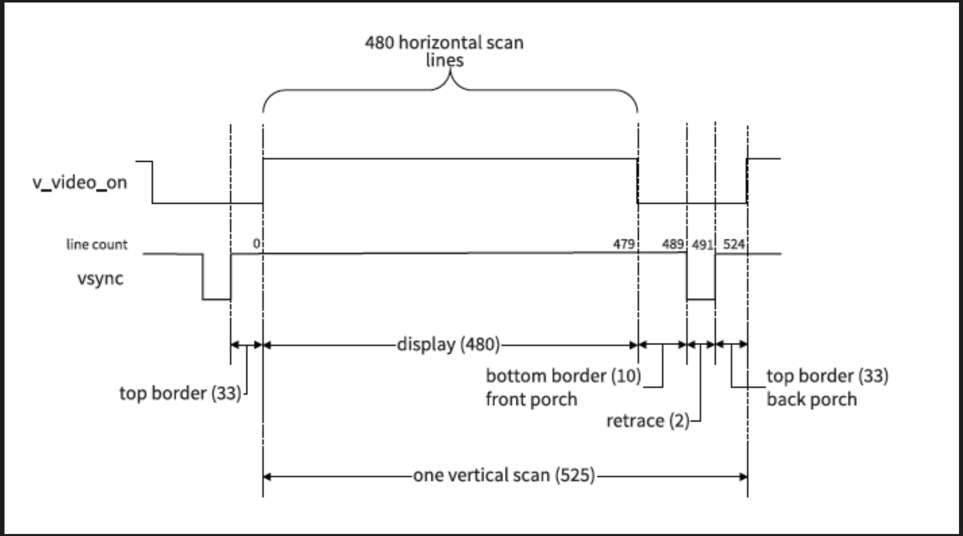


Figure 20.6 Timing diagram of a horizontal scan (with VGA resolution).

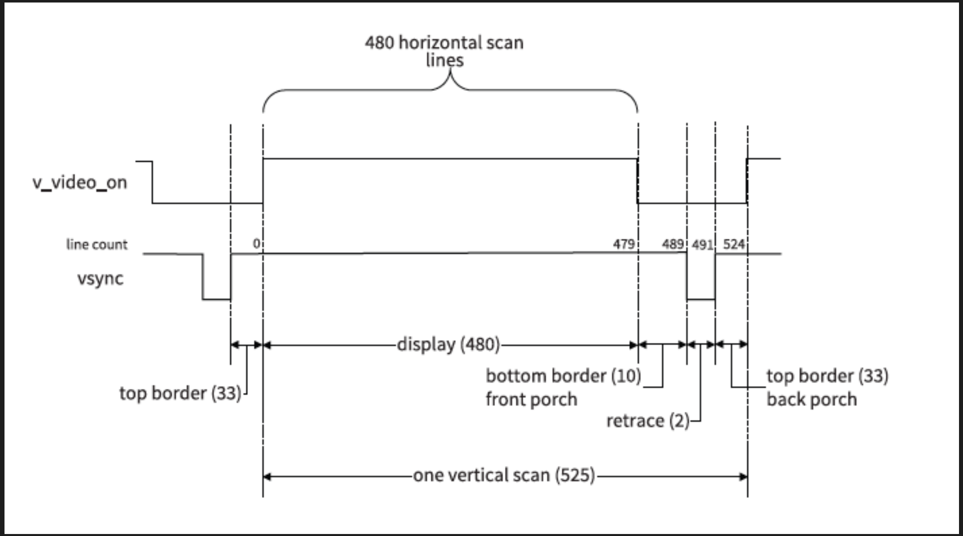
Vertical synchronization

During the vertical scan, the electron beams move gradually from top to bottom and then return to the top.



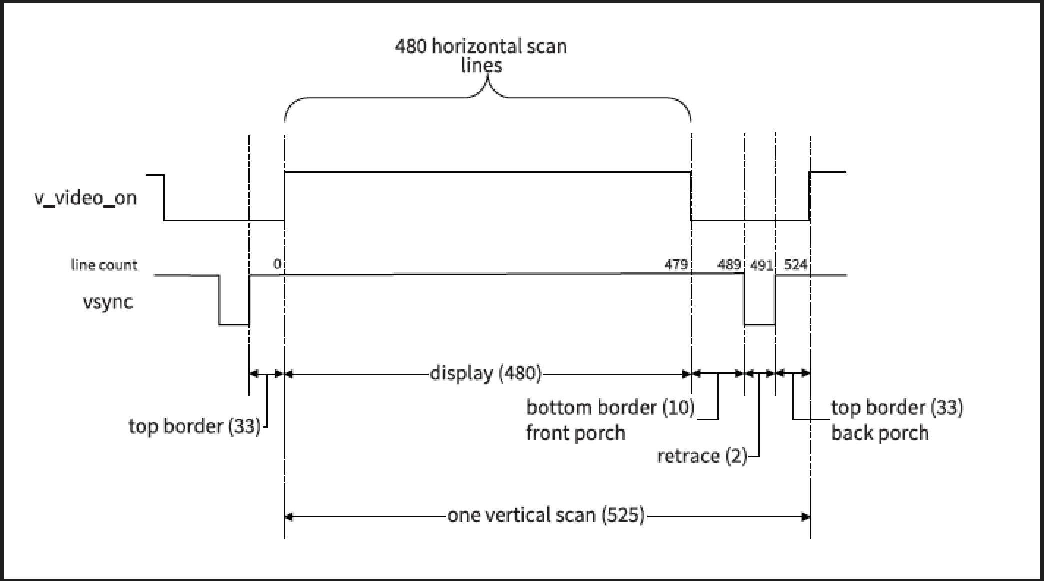
Vertical synchronization

This corresponds to the time required to refresh the entire screen.



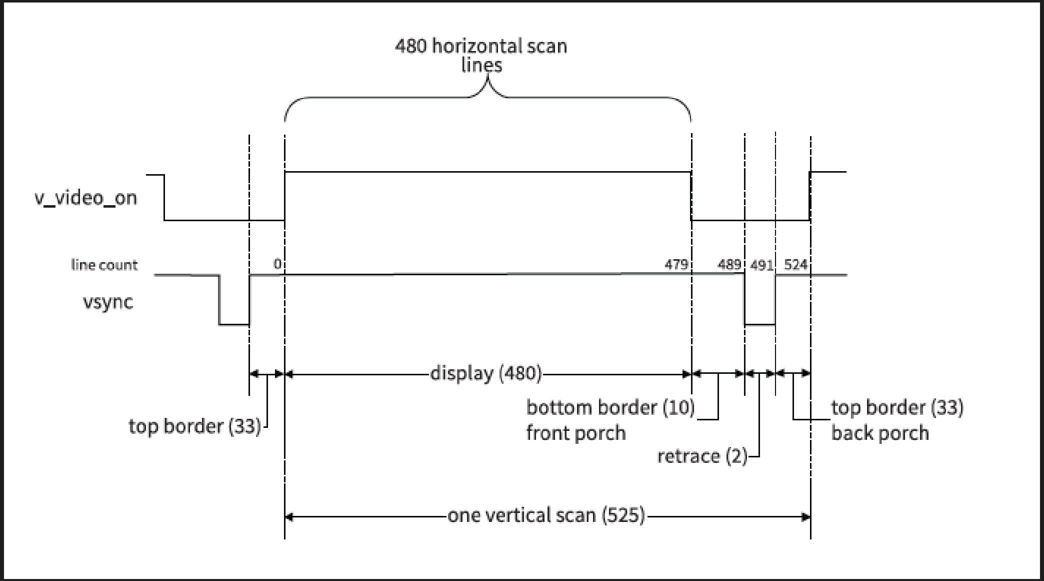
Vertical synchronization

The time unit of the movement is represented in terms of horizontal scan lines. A period of the vsync signal is 525 lines and can be divided into four regions



Vertical synchronization

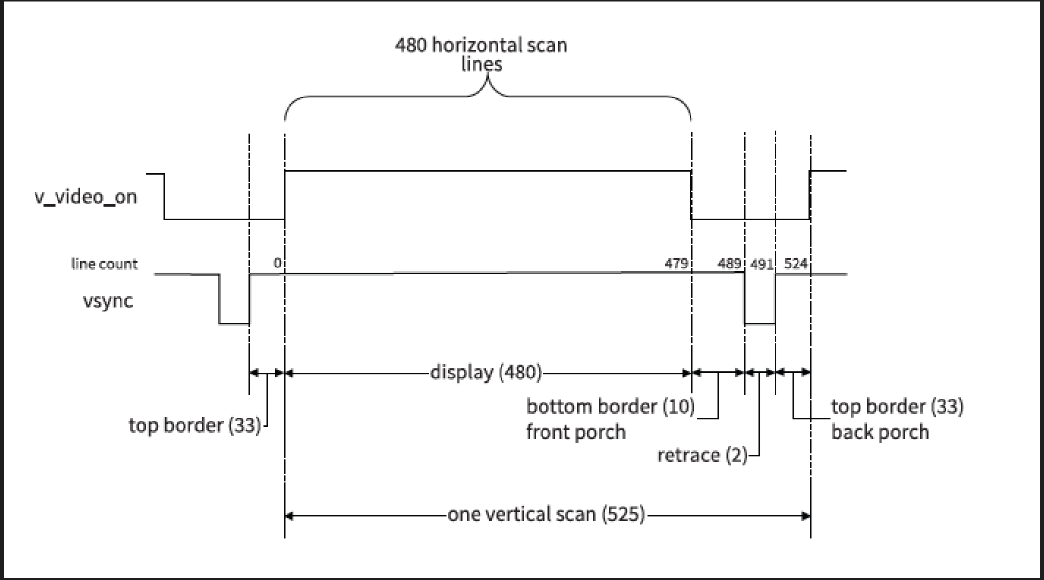
Display: region where the horizontal lines are actually displayed on the screen. The length of this region is 480 lines



Vertical synchronization

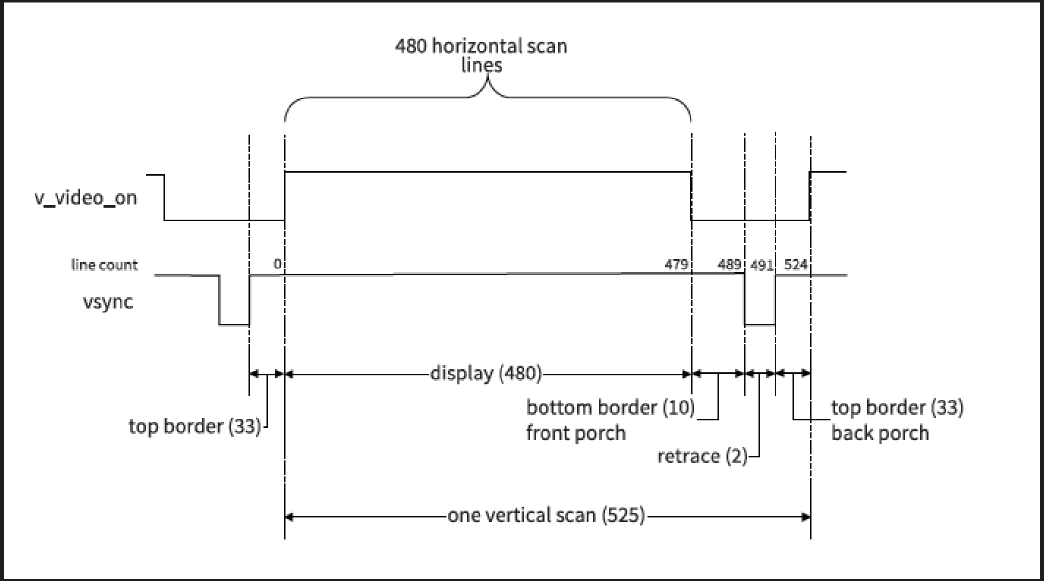
Retrace: region that the electron beams return to the top of the screen. The video

signal should be disabled, and the length of this region is 2 lines.



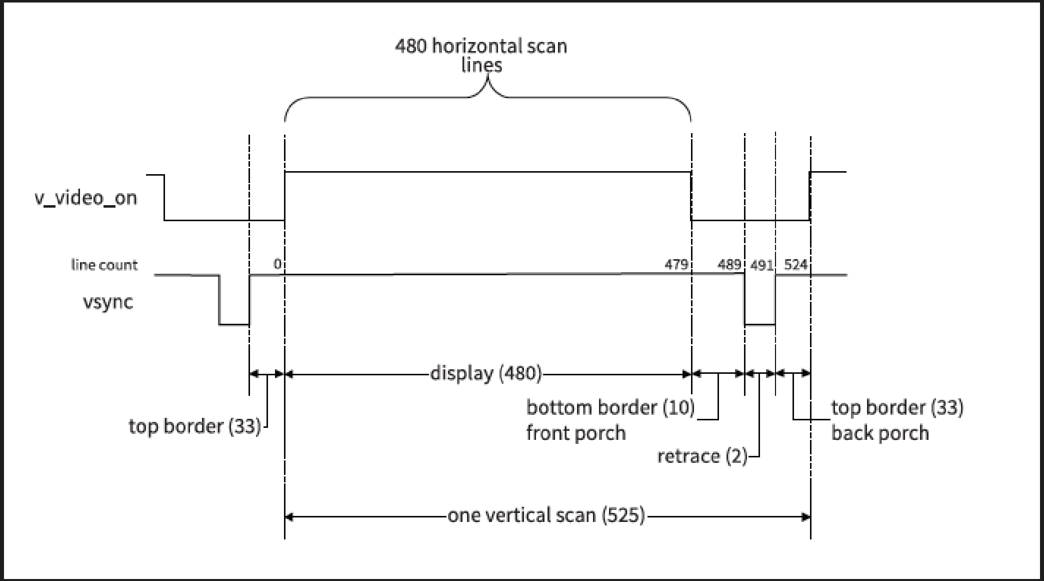
Vertical synchronization

Bottom border: region that forms the bottom border of the display region. It is also known as the frontporch (i.e., porch before retrace). The video signal should be disabled, and the length of this region is 10 lines.



Vertical synchronization

Top border: region that forms the top border of the display region. It is also know as the backporch (i,e,,porch after retrace). The video signal should be disabled, and the length of this region is 33 lines.



Timing calculation of VGA synchronization signals

We mentioned earlier, we assume that the pixel processing rate is 25 MHz.

You might ask how is this number calculated?

Timing calculation of VGA synchronization signals

The pixel rate is determined by three parameters:

p : the number of pixels in a horizontal scan line.

For 640-by-480 resolution p = 800 pixels/line

Timing calculation of VGA synchronization signals

The pixel rate is determined by three parameters:

l: the number of lines in a screen (i.e., a vertical scan).

For 640-by-480 resolution, it is l=525 lines/screen

Timing calculation of VGA synchronization signals

The pixel rate is determined by three parameters:

s: the number of screens per second. For flickering-free operation, we can set it to s=60 screens/second

Timing calculation of VGA synchronization signals

The s parameter specifies how fast the screen should be refreshed. For a human eye, the refresh rate must be at least 30 screens per second to make the motion appear to be continuous. To reduce flickering, the monitor usually has a much higher rate , such as the 60 screens per second specification above.

Pixel rate

The pixel rate can be calculated by the three parameters :

Pixel rate = $p * l * s \sim= 25$ M pixels/second

Pixel rate

The pixel rate for other resolutions and refresh rates can be calculated in a similar fashion.

640x480 resolution timing constants

```vhdl
constant FRAME_WIDTH : natural := 640;
constant FRAME_HEIGHT : natural := 480;

constant H_FP : natural := 16; --H front porch width (pixels)
constant H_PW : natural := 96; --H sync pulse width (pixels)
constant H_MAX : natural := 800; --H total period (pixels)

constant V_FP : natural := 10; --V front porch width (lines)
constant V_PW : natural := 2; --V sync pulse width (lines)
constant V_MAX : natural := 525; --V total period (lines)

constant H_POL : std_logic := '0';
constant V_POL : std_logic := '0';
```

Sync Generation

```
process (pxl_clk)
 begin
   if (rising_edge(pxl_clk)) then
     if (h_cntr_reg = (H_MAX - 1)) then
       h_cntr_reg <= (others =>'0');
     else
       h_cntr_reg <= h_cntr_reg + 1;
     end if;
   end if;
 end process;
```

Sync Generation

```
process (pxl_clk)
begin
  if (rising_edge(pxl_clk)) then
    if ((h_cntr_reg = (H_MAX - 1)) and (v_cntr_reg = (V_MAX - 1))) then
      v_cntr_reg <= (others =>'0');
    elsif (h_cntr_reg = (H_MAX - 1)) then
      v_cntr_reg <= v_cntr_reg + 1;
    end if;
  end if;
end process;
```

Sync Generation

```vhdl
process (pxl_clk)
begin
  if (rising_edge(pxl_clk)) then
    if (h_cntr_reg >= (H_FP + FRAME_WIDTH - 1)) and (h_cntr_reg < (H_FP + FRAME_WIDTH + H_PW - 1)) then
      h_sync_reg <= H_POL;
    else
      h_sync_reg <= not(H_POL);
    end if;
  end if;
end process;
```

Sync Generation

```vhdl
process (pxl_clk)
begin
  if (rising_edge(pxl_clk)) then
    if (v_cntr_reg >= (V_FP + FRAME_HEIGHT - 1)) and (v_cntr_reg < (V_FP + FRAME_HEIGHT + V_PW - 1)) then
      v_sync_reg <= V_POL;
    else
      v_sync_reg <= not(V_POL);
    end if;
  end if;
end process;
```

Sync Generation

```
active <= '1' when ((h_cntr_reg < FRAME_WIDTH) and (v_cntr_reg < FRAME_HEIGHT))else
          '0';
```

Pixel generation with VGA Assignment

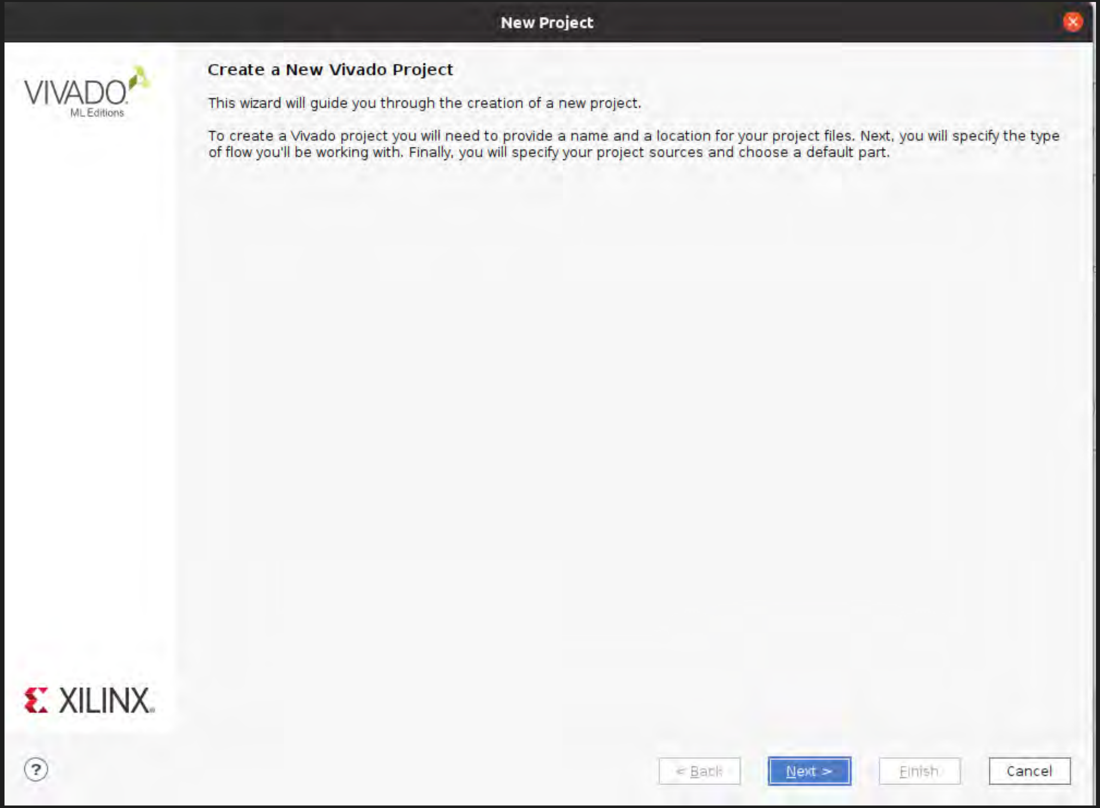
Clone project

```
git clone git@github.com:Digilent/Zybo-Z7.git

cd Zybo-Z7/

git checkout 10/Pmod-VGA/master

git submodule init

git submodule update
```

VGA controller file

Use the Zybo-Z7/hw/src/hdl/top.vhd for your VGA controller

Create a new project

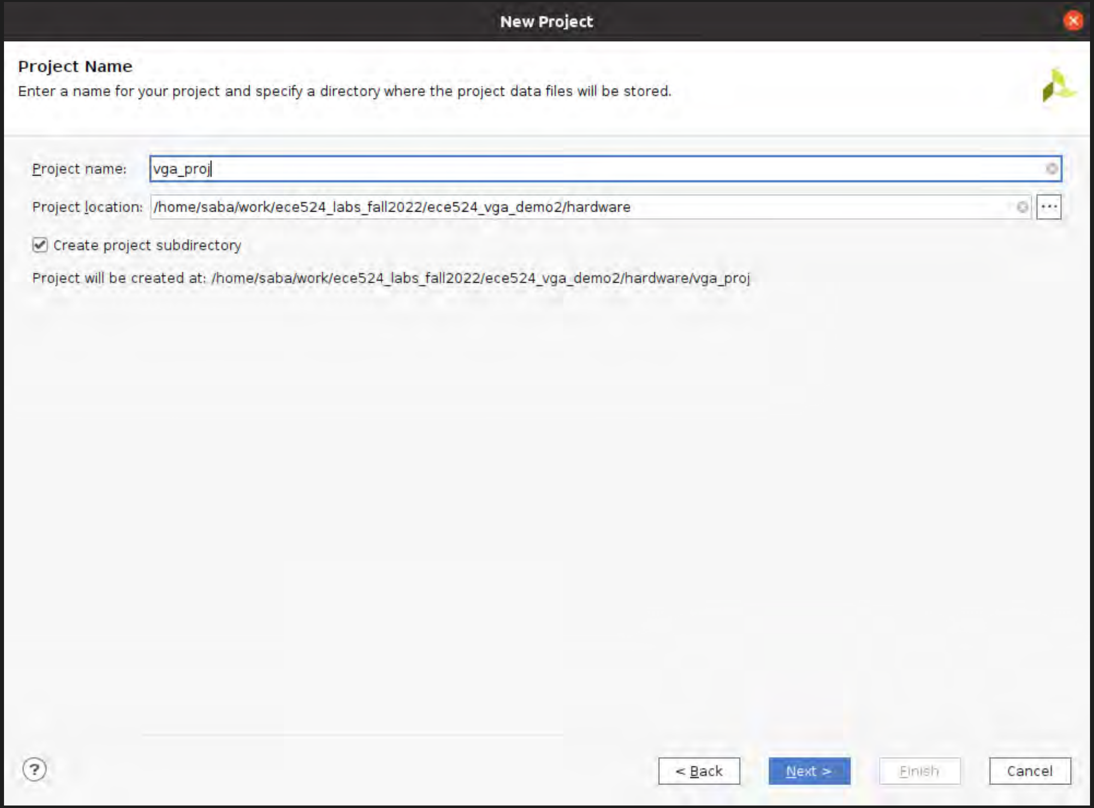


Create a New Vivado Project

This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

Create a new project


New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

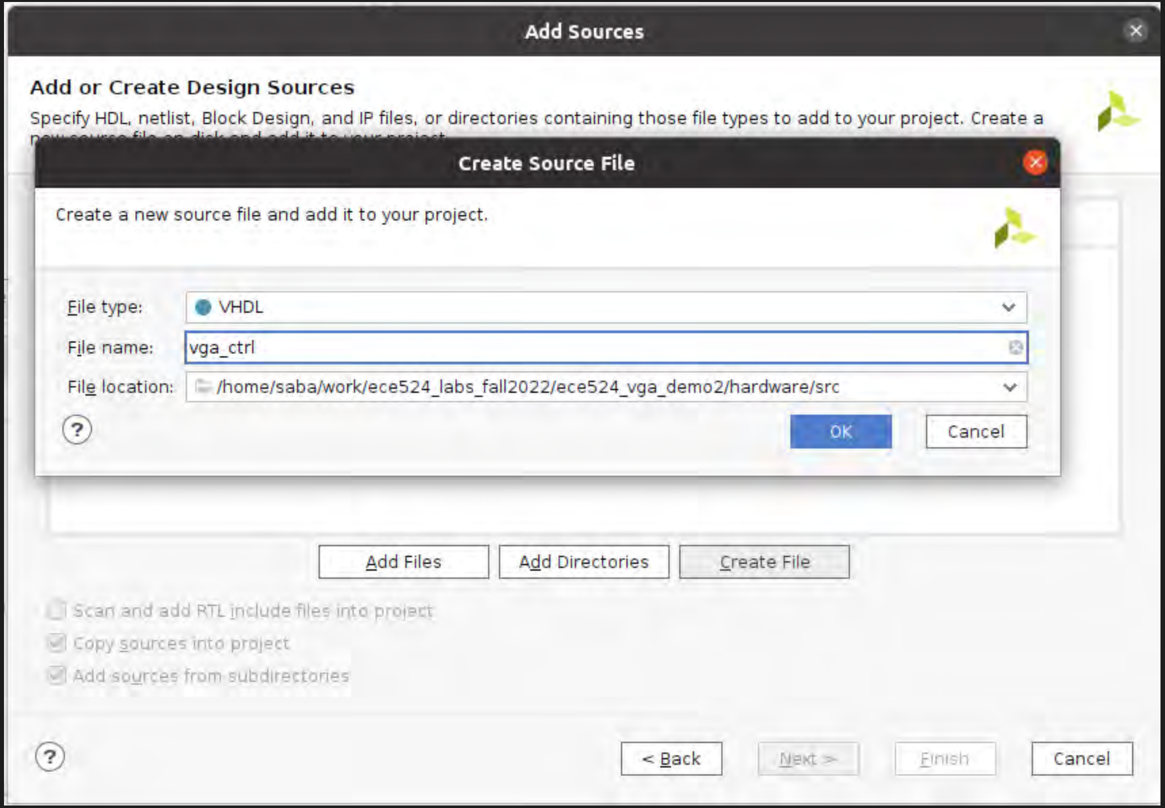
Project name: vga_proj

Project location: /home/saba/work/ece524_labs_fall2022/ece524_vga_demo2/hardware

☑ Create project subdirectory

Project will be created at: /home/saba/work/ece524_labs_fall2022/ece524_vga_demo2/hardware/vga_proj

Create a source file and name it vga_ctrl



Add Sources

Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

Create Source File

Create a new source file and add it to your project.

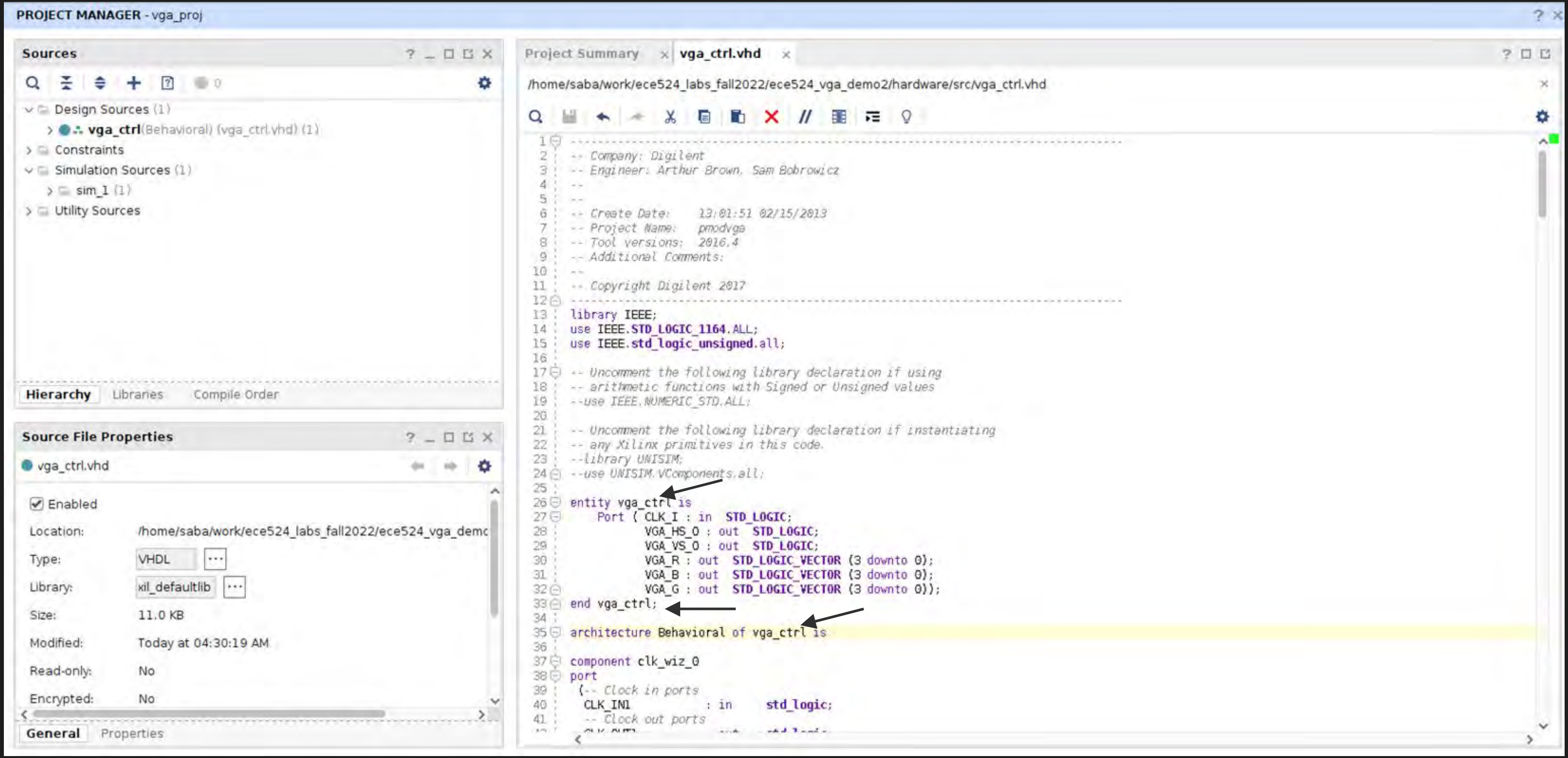
File type: VHDL

File name: vga_ctrl

File location: /home/saba/work/ece524_labs_fall2022/ece524_vga_demo2/hardware/src

OK
Cancel
Add Files
Add Directories
Create File
Scan and add RTL include files into project
Copy sources into project
Add sources from subdirectories
< Back
Next >
Finish
Cancel

Copy top.vhd to vga_ctrl.vhd



PROJECT MANAGER - vga_proj

Sources

- Design Sources (1)
  - vga_ctrl(Behavioral) (vga_ctrl.vhd) (1)
- Constraints
- Simulation Sources (1)
  - sim_1 (1)
- Utility Sources

Hierarchy | Libraries | Compile Order

Source File Properties

vga_ctrl.vhd

☑ Enabled	
Location:	/home/saba/work/ece524_labs_fall2022/ece524_vga_demc
Type:	VHDL
Library:	xil_defaultlib
Size:	11.0 KB
Modified:	Today at 04:30:19 AM
Read-only:	No
Encrypted:	No

General | Properties

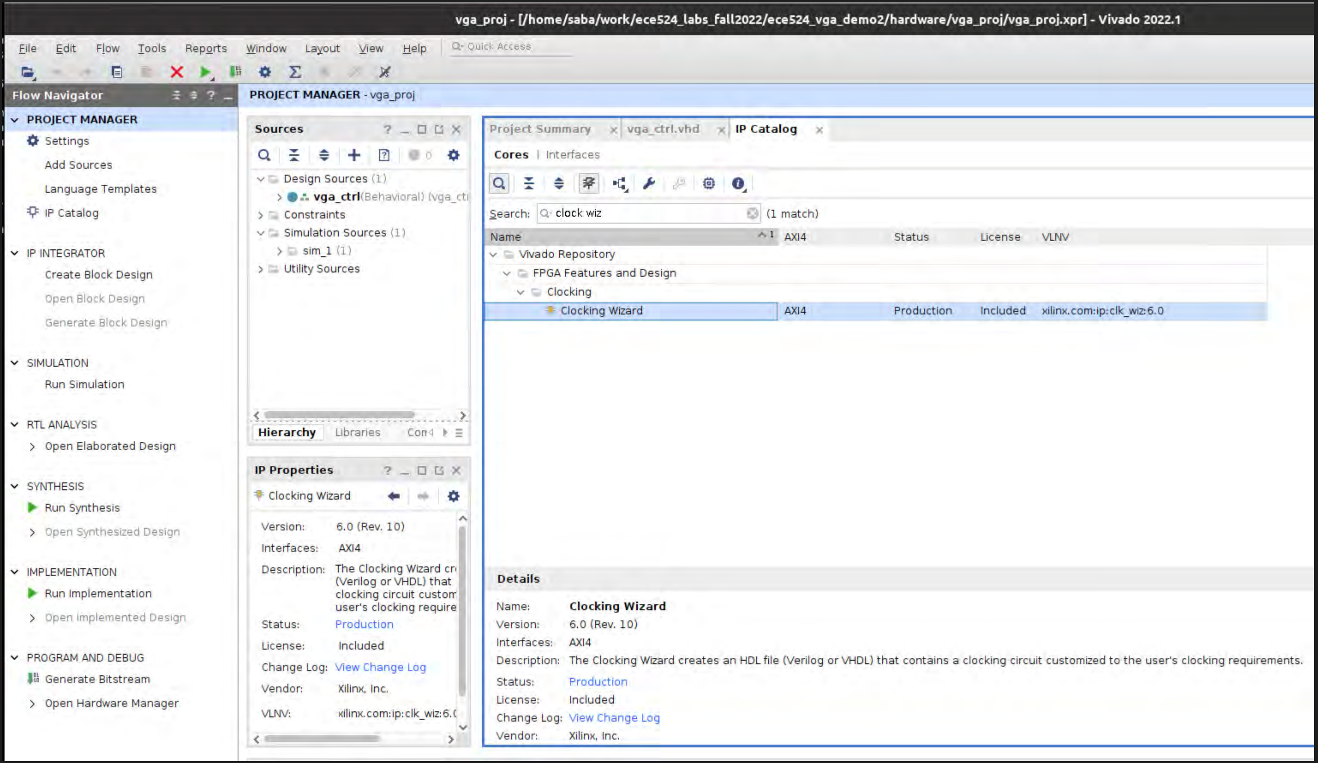
Project Summary | vga_ctrl.vhd

/home/saba/work/ece524_labs_fall2022/ece524_vga_demo2/hardware/src/vga_ctrl.vhd

```vhdl
----------------------------------------------------------------------------------
-- Company: Digilent
-- Engineer: Arthur Brown, Sam Bobrowicz
--
--
-- Create Date:    13:01:51 02/15/2013
-- Project Name:   pmodvga
-- Tool versions:  2016.4
-- Additional Comments:
--
-- Copyright Digilent 2017
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity vga_ctrl is
    Port ( CLK_I : in  STD_LOGIC;
           VGA_HS_O : out  STD_LOGIC;
           VGA_VS_O : out  STD_LOGIC;
           VGA_R : out  STD_LOGIC_VECTOR (3 downto 0);
           VGA_B : out  STD_LOGIC_VECTOR (3 downto 0);
           VGA_G : out  STD_LOGIC_VECTOR (3 downto 0));
end vga_ctrl;

architecture Behavioral of vga_ctrl is

component clk_wiz_0
port
 (-- Clock in ports
  CLK_IN1           : in     std_logic;
  -- Clock out ports
```

Add clk Wiz

From the IP catalog add Clock Wizard IP.


vga_proj - [/home/saba/work/ece524_labs_fall2022/ece524_vga_demo2/hardware/vga_proj/vga_proj.xpr] - Vivado 2022.1

Search: clock wiz (1 match)

Name	AXI4	Status	License	VLNV
Vivado Repository				
FPGA Features and Design				
Clocking				
Clocking Wizard	AXI4	Production	Included	xilinx.com:ip:clk_wiz:6.0

IP Properties

Clocking Wizard

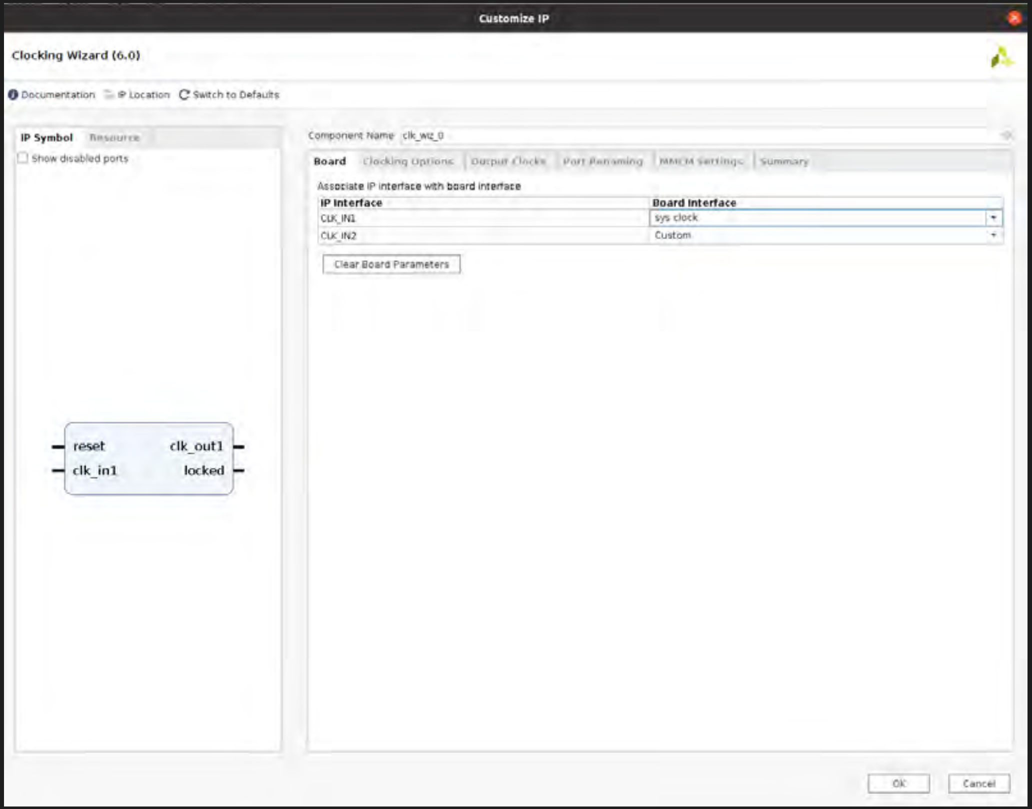
Version:	6.0 (Rev. 10)
Interfaces:	AXI4
Description:	The Clocking Wizard cr (Verilog or VHDL) that clocking circuit custom user's clocking require
Status:	Production
License:	Included
Change Log:	View Change Log
Vendor:	Xilinx, Inc.
VLNV:	xilinx.com:ip:clk_wiz:6.(

Details

Name:	**Clocking Wizard**
Version:	6.0 (Rev. 10)
Interfaces:	AXI4
Description:	The Clocking Wizard creates an HDL file (Verilog or VHDL) that contains a clocking circuit customized to the user's clocking requirements.
Status:	Production
License:	Included
Change Log:	View Change Log
Vendor:	Xilinx, Inc.

Set clk reference

Set the clock reference to sys clock



Component Name clk_wiz_0

Associate IP interface with board interface

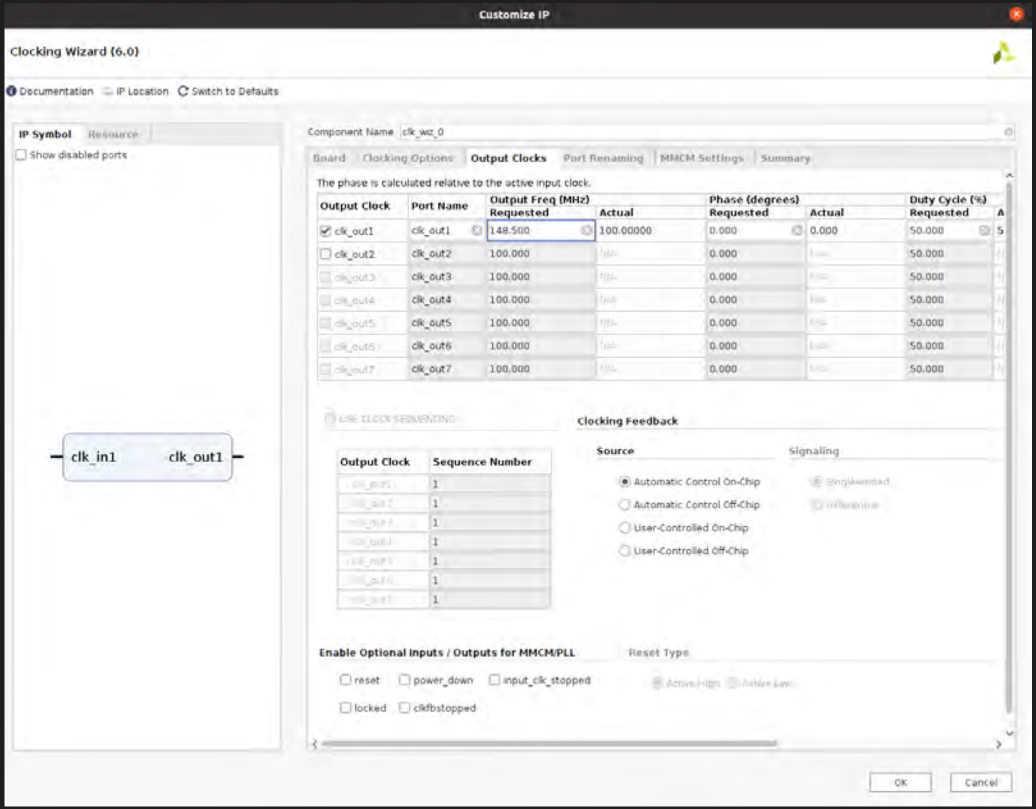
IP Interface	Board Interface
CLK_IN1	sys clock
CLK_IN2	Custom

Set output clk

Set the output clock to 148.5 MHz. This is the required clock for resolution

1920x1080 @ 60Hz

Uncheck reset and locked option.



Component Name clk_wiz_0

The phase is calculated relative to the active input clock.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)	
		Requested	Actual	Requested	Actual	Requested	A
✓ clk_out1	clk_out1	148.500	100.00000	0.000	0.000	50.000	5
☐ clk_out2	clk_out2	100.000	n/a	0.000	n/a	50.000	
☐ clk_out3	clk_out3	100.000	n/a	0.000	n/a	50.000	
☐ clk_out4	clk_out4	100.000	n/a	0.000	n/a	50.000	
☐ clk_out5	clk_out5	100.000	n/a	0.000	n/a	50.000	
☐ clk_out6	clk_out6	100.000	n/a	0.000	n/a	50.000	
☐ clk_out7	clk_out7	100.000	n/a	0.000	n/a	50.000	

☐ USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Clocking Feedback

Source: [✓ Automatic Control On-Chip] Automatic Control Off-Chip User-Controlled On-Chip User-Controlled Off-Chip

Signaling: [✓ Single-ended] Differential

Enable Optional Inputs / Outputs for MMCM/PLL

☐ reset ☐ power_down ☐ input_clk_stopped

☐ locked ☐ clkfbstopped

Reset Type: [✓ Active High] Active Low

Important signals for image drawing

h_cntr_reg : horizontal counter

v_cntr_reg : vertical counter

active : Indicates VGA signal is in display region

Create a process block to handle image display

```
process(active, h_cntr_reg, v_cntr_reg, sw)
begin
   if active = '1' then
       case sw is
           when "0000" =>
               -- do something
           when "0001" =>
               -- do something
           when "0010" =>
               -- do something
           when "0011" =>
               -- do something
           when "0100" =>
               -- do something
           ...
           when others
           vga_red <= (others=>'0');
           vga_green <= (others=>'0');
           vga_blue <= (others=>'0');
       end case;
   else
       vga_red <= (others=>'0');
       vga_green <= (others=>'0');
       vga_blue <= (others=>'0');
   end if;
end process;
```

when "0000"

```vhdl
-- Monitor off
vga_red <= (others=>'0');
vga_green <= (others=>'0');
vga_blue <= (others=>'0');
```

when "0001"

```vhdl
-- Show solid red
vga_red <= (others=>'1');
vga_green <= (others=>'0');
vga_blue <= (others=>'0');
```

when "0010"

```vhdl
-- Show solid green
vga_red <= (others=>'0');
vga_green <= (others=>'1');
vga_blue <= (others=>'0');
```

when "0100"

```vhdl
-- Divide the screen into 3 regions and show RGB
if  (h_cntr_reg >= 0 and h_cntr_reg < 640) then
   vga_red <= (others=>'1');
   vga_green <= (others=>'0');
   vga_blue <= (others=>'0');
elsif  (h_cntr_reg >= 640 and h_cntr_reg < 2*640) then
   vga_red <= (others=>'0');
   vga_green <= (others=>'1');
   vga_blue <= (others=>'0');
elsif  (h_cntr_reg >= 2*640 and h_cntr_reg < 3*640) then
   vga_red <= (others=>'0');
   vga_green <= (others=>'0');
   vga_blue <= (others=>'1');
else
   vga_red <= (others=>'0');
   vga_green <= (others=>'0');
   vga_blue <= (others=>'0');
end if;
```

when "0101"

```
-- Divide Monitor to 8 regions and show white, yellow, cyan, green,
magenta, red, blue, and black
if  (h_cntr_reg >= 0 and h_cntr_reg < 240) then
   -- white
   vga_red <= (others=>'1');
   vga_green <= (others=>'1');
   vga_blue <= (others=>'1');
elsif  (h_cntr_reg >= 240 and h_cntr_reg < 2*240) then
   -- yellow
   vga_red <= (others=>'1');
   vga_green <= (others=>'1');
   vga_blue <= (others=>'0');
elsif  (h_cntr_reg >= 2*240 and h_cntr_reg < 3*240) then
   -- cyan
   vga_red <= (others=>'0');
   vga_green <= (others=>'1');
   vga_blue <= (others=>'1');
elsif  (h_cntr_reg >= 3*240 and h_cntr_reg < 4*240) then
   -- green
   vga_red <= (others=>'0');
   vga_green <= (others=>'1');
   vga_blue <= (others=>'0');
elsif  (h_cntr_reg >= 4*240 and h_cntr_reg < 5*240) then
   -- magenta
   vga_red <= (others=>'1');
   vga_green <= (others=>'0');
   vga_blue <= (others=>'1');
elsif  (h_cntr_reg >= 5*240 and h_cntr_reg < 6*240) then
   -- red
   vga_red <= (others=>'1');
   vga_green <= (others=>'0');
   vga_blue <= (others=>'0');
elsif  (h_cntr_reg >= 6*240 and h_cntr_reg < 7*240) then
   -- blue
   vga_red <= (others=>'0');
   vga_green <= (others=>'0');
   vga_blue <= (others=>'1');
elsif  (h_cntr_reg >= 7*240 and h_cntr_reg < 8*240) then
   -- black
   vga_red <= (others=>'0');
   vga_green <= (others=>'0');
   vga_blue <= (others=>'0');
else
   vga_red <= (others=>'0');
   vga_green <= (others=>'0');
   vga_blue <= (others=>'0');
end if;
```

when "0110"

```
-- divide the monitor into 8 sections and show 8 shades of
gray
if  (h_cntr_reg >= 0 and h_cntr_reg < 240) then
   -- white
   vga_red <= (others=>'1');
   vga_green <= (others=>'1');
   vga_blue <= (others=>'1');
elsif  (h_cntr_reg >= 240 and h_cntr_reg < 2*240) then
   vga_red <= "1110";
   vga_green <= "1110";
   vga_blue <= "1110";
elsif  (h_cntr_reg >= 2*240 and h_cntr_reg < 3*240) then
   vga_red <= "1100";
   vga_green <= "1100";
   vga_blue <= "1100";
elsif  (h_cntr_reg >= 3*240 and h_cntr_reg < 4*240) then
   vga_red <= "1010";
   vga_green <= "1010";
   vga_blue <= "1010";
elsif  (h_cntr_reg >= 4*240 and h_cntr_reg < 5*240) then
   vga_red <= "1000";
   vga_green <= "1000";
   vga_blue <= "1000";
elsif  (h_cntr_reg >= 5*240 and h_cntr_reg < 6*240)
then
   vga_red <= "0110";
   vga_green <= "0110";
   vga_blue <= "0110";
elsif  (h_cntr_reg >= 6*240 and h_cntr_reg < 7*240)
then
   vga_red <= "0010";
   vga_green <= "0010";
   vga_blue <= "0010";
elsif  (h_cntr_reg >= 7*240 and h_cntr_reg < 8*240)
then
   -- black
   vga_red <= (others=>'0');
   vga_green <= (others=>'0');
   vga_blue <= (others=>'0');
else
   vga_red <= (others=>'0');
   vga_green <= (others=>'0');
   vga_blue <= (others=>'0');
end if;
```

when "0111" - Use btn to create different horizontal stripes

```
case btn is
   when "0000" => vga_red <= h_cntr_reg(3 downto 0);
                  vga_green <= h_cntr_reg(3 downto 0);
                  vga_blue <= h_cntr_reg(3 downto 0);
   when "0001" => vga_red <= h_cntr_reg(4 downto 1);
                  vga_green <= h_cntr_reg(4 downto 1);
                  vga_blue <= h_cntr_reg(4 downto 1);
   when "0010" => vga_red <= h_cntr_reg(5 downto 2);
                  vga_green <= h_cntr_reg(5 downto 2);
                  vga_blue <= h_cntr_reg(5 downto 2);
   when "0100" => vga_red <= h_cntr_reg(6 downto 3);
                  vga_green <= h_cntr_reg(6 downto 3);
                  vga_blue <= h_cntr_reg(6 downto 3);
   when "1000" => vga_red <= h_cntr_reg(11 downto 8);
                  vga_green <= h_cntr_reg(11 downto 8);
                  vga_blue <= h_cntr_reg(11 downto 8);
   when others =>
                   vga_red <= (others=>'0');
                   vga_green <= (others=>'0');
                   vga_blue <= (others=>'0');
end case;
```

when "1000" - Use btn to create different vertical stripes

```
case btn is
   when "0000" => vga_red <= v_cntr_reg(3 downto 0);
                  vga_green <= v_cntr_reg(3 downto 0);
                  vga_blue <= v_cntr_reg(3 downto 0);
   when "0001" => vga_red <= v_cntr_reg(4 downto 1);
                  vga_green <= v_cntr_reg(4 downto 1);
                  vga_blue <= v_cntr_reg(4 downto 1);
   when "0010" => vga_red <= v_cntr_reg(5 downto 2);
                  vga_green <= v_cntr_reg(5 downto 2);
                  vga_blue <= v_cntr_reg(5 downto 2);
   when "0100" => vga_red <= v_cntr_reg(6 downto 3);
                  vga_green <= v_cntr_reg(6 downto 3);
                  vga_blue <= v_cntr_reg(6 downto 3);
   when "1000" => vga_red <= v_cntr_reg(11 downto 8);
                  vga_green <= v_cntr_reg(11 downto 8);
                  vga_blue <= v_cntr_reg(11 downto 8);
   when others =>
                   vga_red <= (others=>'0');
                   vga_green <= (others=>'0');
                   vga_blue <= (others=>'0');
end case;
```

when "1001" - create different size checker board pattern

```
-- create different size checker board pattern
case btn is
when "0000" =>
               vga_red <= (others=>(v_cntr_reg(5) xor h_cntr_reg(5)));
               vga_green <= (others=>(v_cntr_reg(5) xor h_cntr_reg(5)));
               vga_blue <= (others=>(v_cntr_reg(5) xor h_cntr_reg(5)));
when "0001" =>
               vga_red <= (others=>(v_cntr_reg(6) xor h_cntr_reg(6)));
               vga_green <= (others=>(v_cntr_reg(6) xor h_cntr_reg(6)));
               vga_blue <= (others=>(v_cntr_reg(6) xor h_cntr_reg(6)));
when "0010" =>
               vga_red <= (others=>(v_cntr_reg(7) xor h_cntr_reg(7)));
               vga_green <= (others=>(v_cntr_reg(7) xor h_cntr_reg(7)));
               vga_blue <= (others=>(v_cntr_reg(7) xor h_cntr_reg(7)));
when "0100" =>
               vga_red <= (others=>(v_cntr_reg(8) xor h_cntr_reg(8)));
               vga_green <= (others=>(v_cntr_reg(8) xor h_cntr_reg(8)));
               vga_blue <= (others=>(v_cntr_reg(8) xor h_cntr_reg(8)));
when "1000" =>
               vga_red <= (others=>(v_cntr_reg(9) xor h_cntr_reg(9)));
               vga_green <= (others=>(v_cntr_reg(9) xor h_cntr_reg(9)));
               vga_blue <= (others=>(v_cntr_reg(9) xor h_cntr_reg(9)));
when others =>
           vga_red <= (others=>'0');
           vga_green <= (others=>'0');
           vga_blue <= (others=>'0');
end case;
```

when "1010"

```vhdl
when "1010" =>
-- create checkerboard with inner repeat pattern
case btn is
   when "0000" =>
               vga_red <= v_cntr_reg(6 downto 3) and h_cntr_reg(6 downto 3);
               vga_green <= v_cntr_reg(6 downto 3) and h_cntr_reg(6 downto 3);
               vga_blue <= v_cntr_reg(6 downto 3) and h_cntr_reg(6 downto 3);
   when "0001" =>
               vga_red <= v_cntr_reg(6 downto 3) or h_cntr_reg(6 downto 3);
               vga_green <= v_cntr_reg(6 downto 3) or h_cntr_reg(6 downto 3);
               vga_blue <= v_cntr_reg(6 downto 3) or h_cntr_reg(6 downto 3);
   when "0010" =>
               vga_red <= v_cntr_reg(6 downto 3) xor h_cntr_reg(6 downto 3);
               vga_green <= v_cntr_reg(6 downto 3) xor h_cntr_reg(6 downto 3);
               vga_blue <= v_cntr_reg(6 downto 3) xor h_cntr_reg(6 downto 3);
   when "0100" =>
               vga_red <= v_cntr_reg(7 downto 4) xor h_cntr_reg(7 downto 4);
               vga_green <= v_cntr_reg(7 downto 4) xor h_cntr_reg(7 downto 4);
               vga_blue <= v_cntr_reg(7 downto 4) xor h_cntr_reg(7 downto 4);
   when "1000" =>
               vga_red <= v_cntr_reg(8 downto 5) xor h_cntr_reg(8 downto 5);
               vga_green <= v_cntr_reg(8 downto 5) xor h_cntr_reg(8 downto 5);
               vga_blue <= v_cntr_reg(8 downto 5) xor h_cntr_reg(8 downto 5);
   when others =>
           vga_red <= (others=>'0');
           vga_green <= (others=>'0');
           vga_blue <= (others=>'0');
end case;
```

when "1110"

```
when "1110" =>
-- show moving ball
if  pixel_in_box = '1' then
   vga_red <=  (others => '1');
else
   vga_red <=  (others=>'0');
end if;
```

```
when "1110" =>
-- show moving ball
if  pixel_in_box = '1' then
   vga_red <=  (others => '1');
else
   vga_red <=  (others=>'0');
end if;
```

Moving ball logic

```vhdl
------------------------------------------------------
-------         MOVING BOX LOGIC                ------
------------------------------------------------------
process (pxl_clk)
begin
  if (rising_edge(pxl_clk)) then
    if (update_box = '1') then
      if (box_x_dir = '1') then
        box_x_reg <= box_x_reg + 1;
      else
        box_x_reg <= box_x_reg - 1;
      end if;
      if (box_y_dir = '1') then
        box_y_reg <= box_y_reg + 1;
      else
        box_y_reg <= box_y_reg - 1;
      end if;
    end if;
  end if;
end process;
```

Moving ball logic

```vhdl
process (pxl_clk)
begin
  if (rising_edge(pxl_clk)) then
    if (update_box = '1') then
      if ((box_x_dir = '1' and (box_x_reg = BOX_X_MAX - 1)) or (box_x_dir = '0' and (box_x_reg = BOX_X_MIN + 1))) then
        box_x_dir <= not(box_x_dir);
      end if;
      if ((box_y_dir = '1' and (box_y_reg = BOX_Y_MAX - 1)) or (box_y_dir = '0' and (box_y_reg = BOX_Y_MIN + 1))) then
        box_y_dir <= not(box_y_dir);
      end if;
    end if;
  end if;
end process;
```

Moving ball logic

```
process (pxl_clk)
begin
  if (rising_edge(pxl_clk)) then
    if (box_cntr_reg = (BOX_CLK_DIV - 1)) then
      box_cntr_reg <= (others=>'0');
    else
      box_cntr_reg <= box_cntr_reg + 1;
    end if;
  end if;
end process;
```

Moving ball logic

```
update_box <= '1' when box_cntr_reg = (BOX_CLK_DIV - 1) else
              '0';
```

Moving ball logic

```
ball_point <= BALL_ROM(conv_integer(v_cntr_reg(4 downto 0) - box_y_reg))(conv_integer(h_cntr_reg(4 downto 0) - box_x_reg))
when (((h_cntr_reg >= box_x_reg) and (h_cntr_reg < (box_x_reg + BOX_WIDTH))) and
     ((v_cntr_reg >= box_y_reg) and (v_cntr_reg < (box_y_reg + BOX_WIDTH)))) else
     '0';

pixel_in_box <= ball_point when (((h_cntr_reg >= box_x_reg) and (h_cntr_reg < (box_x_reg + BOX_WIDTH))) and
                    ((v_cntr_reg >= box_y_reg) and (v_cntr_reg < (box_y_reg + BOX_WIDTH)))) else
             '0';
```

Displaying a Non-Rectangular Object

Option 1: Using Equation of a Circle

Option 2: Using Pattern ROM

Option 1: Using Equation of a Circle

$(x - x0)^2 + (y - y0)^2 \leq R^2$

Option 2: Using Pattern ROM

First check whether

$x0 - R \leq x \leq x0 + R$

and

$y0 - R \leq y \leq y0 + R$

Then, use

$x - (x0 - R)$ and $y - (y0 - R)$

as an address in the pattern memory

Python code to generate bitmap for a ball

```python
from math import pi, sqrt, ceil

radius = 10

for i in range(-radius, radius+1):
   num = ceil(sqrt(radius**2 - i**2))
   print("{:^{}}".format("".join("1" for _ in range(2*num)), 2*radius))
```

Ball bitmap

```
     1111111111
    111111111111
  1111111111111111
  1111111111111111
 111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
11111111111111111111
 111111111111111111
  1111111111111111
  1111111111111111
    111111111111
     1111111111
```

Ball bitmap ROM

```vhdl
type rom_type is array (0 to 19) of std_logic_vector(19 downto 0);
-- ROM definition
constant BALL_ROM: rom_type :=
(
"00000111111111100000",
"00001111111111110000",
"00111111111111111100",
"00111111111111111100",
"01111111111111111110",
"11111111111111111111",
"11111111111111111111",
"11111111111111111111",
"11111111111111111111",
"11111111111111111111",
"11111111111111111111",
"11111111111111111111",
"11111111111111111111",
"11111111111111111111",
"11111111111111111111",
"01111111111111111110",
"00111111111111111100",
"00111111111111111100",
"00001111111111110000",
"00000111111111100000"
);

signal ball_point : std_logic;
```

when "1111"

Given the following bitmap display it on the middle of the screen and use push buttons to show different size of the character

```vhdl
type rom_type_2 is array (0 to 27) of std_logic_vector(27 downto 0);
-- ROM definition
constant img: rom_type_2 :=
(
"0000000000000000000000000000",
"0000000000000000000000000000",
"0000000000000000000000000000",
"0000000000000000000000000000",
"0000000000000000000000000000",
"0000000000001111111111110000",
"0000000011111111111111110000",
"0000000111111111111111100000",
"0000000111111111110000000000",
"0000000011111101100000000000",
"0000000001111100000000000000",
"0000000000011110000000000000",
"0000000000011110000000000000",
"0000000000001111110000000000",
"0000000000000111111000000000",
"0000000000000011111100000000",
"0000000000000001111100000000",
"0000000000000000011110000000",
"0000000000000011111110000000",
"0000000000001111111100000000",
"0000000000111111111000000000",
"0000000011111111110000000000",
"0000001111111111000000000000",
"0000111111111100000000000000",
"0000111111110000000000000000",
"0000000000000000000000000000",
"0000000000000000000000000000",
"0000000000000000000000000000");
```

Bitmap

```vhdl
type rom_type_2 is array (0 to 27) of std_logic_vector(27 downto 0);
-- ROM definition
constant img: rom_type_2 :=
(
"0000000000000000000000000000",
"0000000000000000000000000000",
"0000000000000000000000000000",
"0000000000000000000000000000",
"0000000000000000000000000000",
"0000000000001111111111110000",
"0000000011111111111111110000",
"0000000111111111111111100000",
"0000000111111111110000000000",
"0000000011111101100000000000",
"0000000001111100000000000000",
"0000000000011110000000000000",
"0000000000011110000000000000",
"0000000000001111110000000000",
"0000000000000111111000000000",
"0000000000000011111000000000",
"0000000000000001111000000000",
"0000000000000000011110000000",
"0000000000000011111110000000",
"0000000000001111111100000000",
"0000000000111111111000000000",
"0000000011111111110000000000",
"0000001111111111000000000000",
"0000111111111100000000000000",
"0000111111110000000000000000",
"0000000000000000000000000000",
"0000000000000000000000000000",
"0000000000000000000000000000");
```

Resources

http://tinyvga.com/vga-timing

https://digilent.com/reference/programmable-logic/cora-z7/demos/pmod-vga

https://github.com/Digilent/Cora-Z7-10-Pmod-VGA

https://digilent.com/shop/pmod-vga-video-graphics-array/

https://github.com/Digilent/Zybo-Z7

THANKS!

Do you have any questions?