Lab-exercise

Lab6:

Video timing

Introduction

The basis of a hardware system driving the screen is the controller generating the horizontal and vertical synchronization pulses. Errors in the implementation of this unit best case give you a blank or a scrolling or flickering image. Worst case you damage the screen beyond repair. That's why it is important to simulate and verify this unit before implementing it. The simulation itself will be done in module1c.

Goals

The goal of this module is to acquire insight in writing good synthesizable and readable VHDL code for digital systems. After having done this lab a student should be able to write good, synthesizable, easily readable VHDL code for digital systems. The result of doing this lab should be a student that can write a well-documented VHDL architecture, re-useable by e.g. another student.

Knowledge background

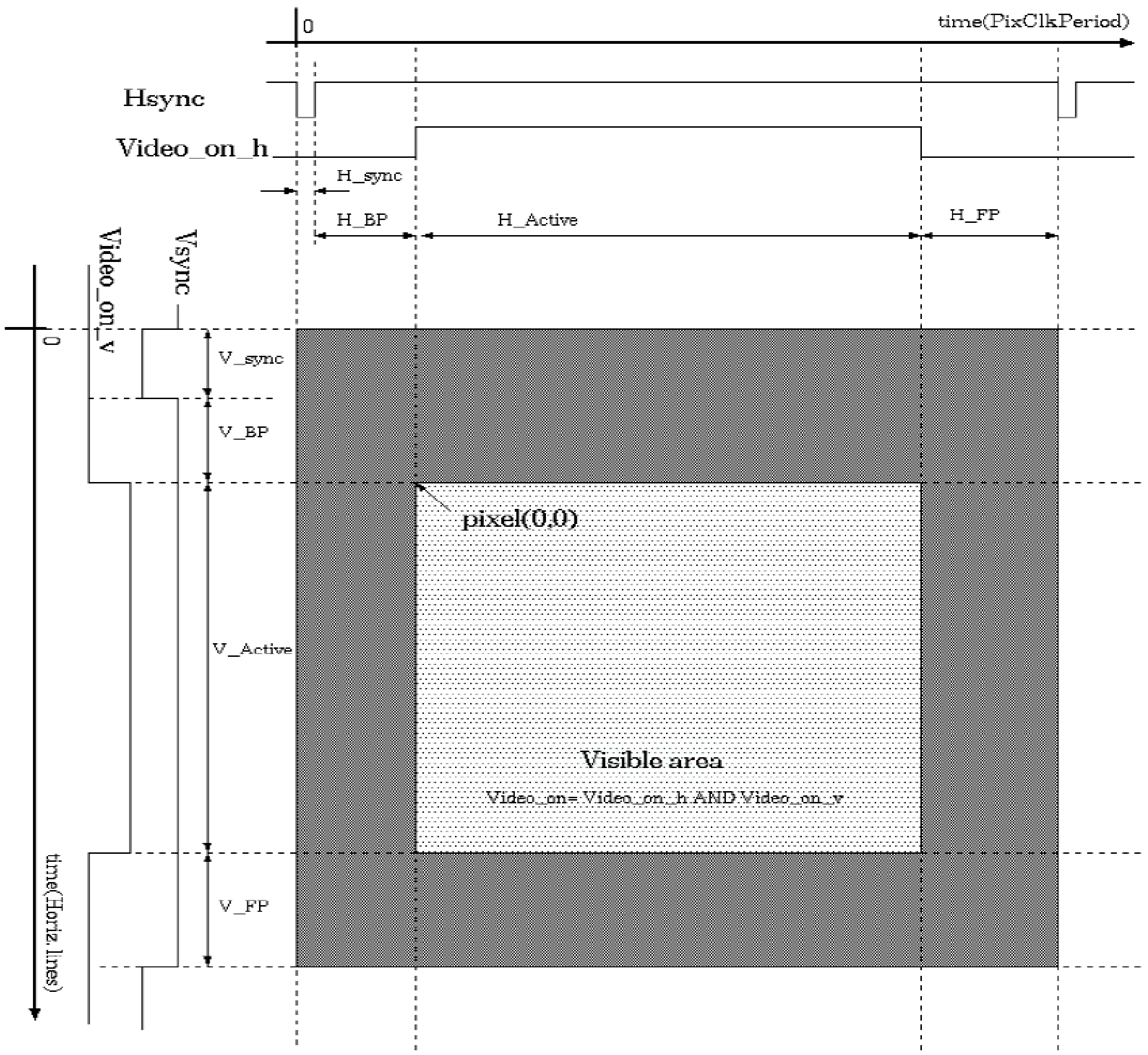
A basic knowledge of VHDL and synchronous digital design is desirable.

Classification

Difficulty level (from 1 to 5): 2.5
Length (without support): 2 hours

Driving (S)VGA

Driving a screen is done using three color signals (red, green and blue) in conjunction with a horizontal and a vertical synchronization pulse. The voltage level of the RGB signals - between 0 and 700mV – defines the exact pixel colors. Before each line a horizontal synchronization pulse is sent. A vertical synchronization pulse resets the beamer in the top left corner. The RGB signals are driven while the Video_on signal is active. The screen resolution and refresh rate define the pixel clock frequency and the horizontal and vertical timing parameters. The exact values are to be found in the XUP hardware reference manual. All digital signals like the RGB busses and the horizontal and vertical synchronization pulses are generated in the FPGA. The conversion of the digital RGB busses to the analog signals for the screen is done in a DAC, available on the XUP board.



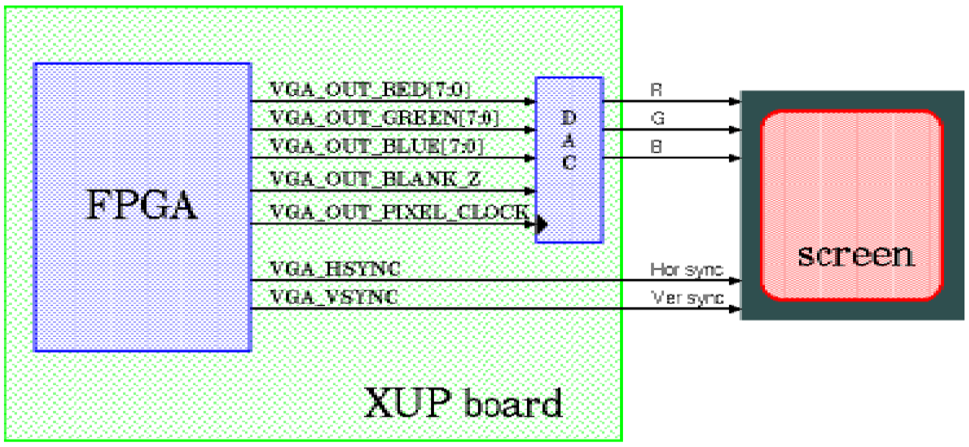


Figure 2

Signal timing

In this lab we are going to drive the screen in its lowest mode. That means a resolution of 640x480 with a refresh rate of 60 Hz and a 25 MHz pixel clock. To allow easy mode changes later, it's best to combine all timing parameters in a VHDL package. Below you can find the relevant part of the Constants_pack package for this module. Only use these constants in your code. Do not use the numbers themselves. The meaning of these constants is apparent from figure 1. Note that the horizontal constants are expressed in pixel clock periods while the vertical ones are expressed in horizontal lines. Writing a full horizontal line hence lasts H_sync+H_bp+H_active+H_fp = H_total pixel klok periods.   V_sync for example takes as long as 2 horizontal lines. I.e. 2*H_total pixel clock periods.

```
package Constants_pack is
constant Del_DAC : integer := 1;
constant H_total : integer := 800; -- pixel clock periods
constant H_active : integer := 640; -- pixel clock periods
constant H_fp : integer := 16; -- pixel clock periods
constant H_sync : integer := 96; -- pixel clock periods
constant H_bp : integer := 48; -- pixel clock periods
constant V_total : integer := 520; -- horizontal lines
constant V_active : integer := 480; -- horizontal lines
constant V_fp : integer := 9; -- horizontal lines
constant V_sync : integer := 2; -- horizontal lines
constant V_bp : integer := 29; -- horizontal lines
end Constants_pack;
package body Constants_pack is
end Constants_pack;
```

Figure 3 shows a timing view of the horizontal and vertical pulses in combination with the video_on signal. We'll explain the start_video signal later.

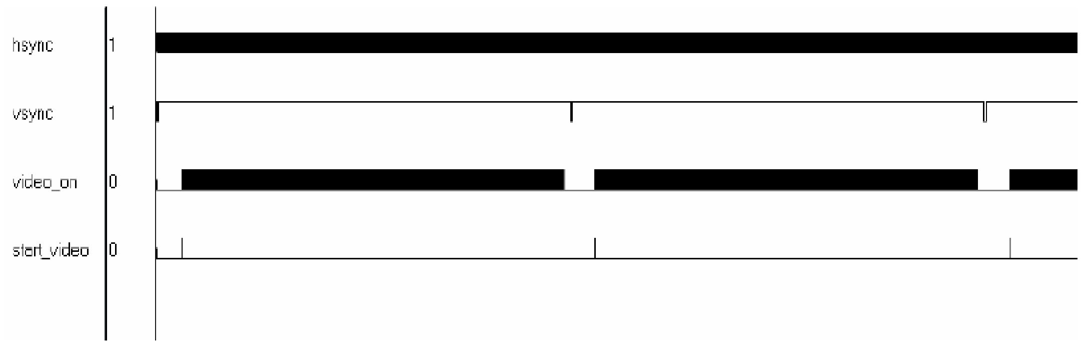


Figure 3

Zooming in (watch figure 4) , you can see that there are two horizontal synchronization (hsync) pulses during the vertical pulse (vsynch) and that there are 29 hsync pulses between the rising edge of the vsync puls and the moment the video_on signal becomes active.

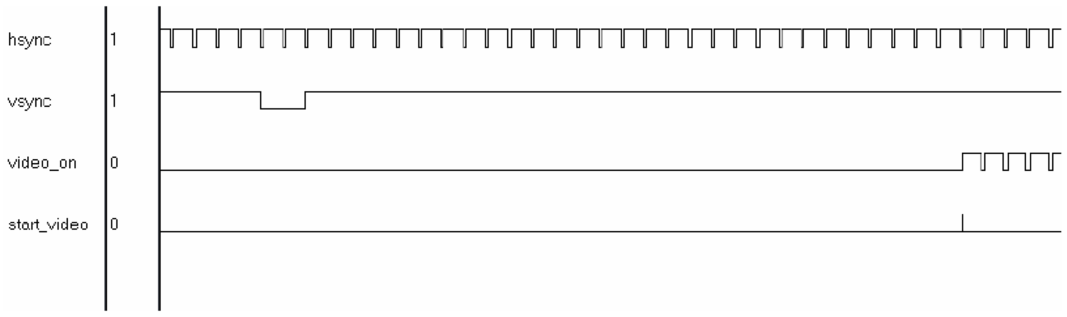


Figure 4

Figure 5 shows signal values after zooming in on pixel_clock level.

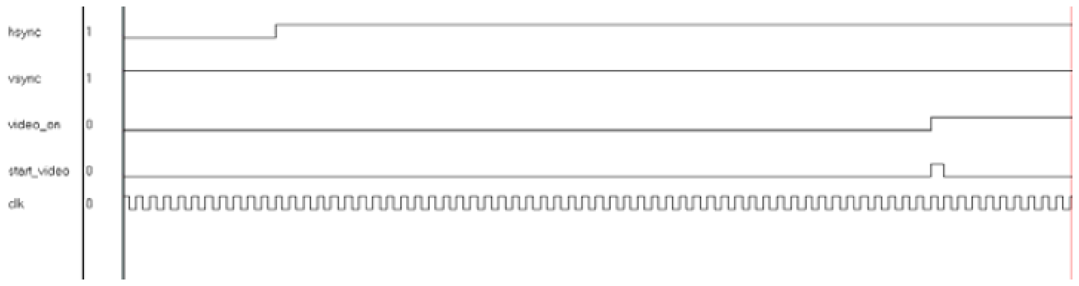


Figure 5

Write a VHDL architecture of the entity VideoTimingGen for the generation of the horizontal and vertical sync pulses. As all outputs will be re-timed in the modules that will follow this one, none of the outputs have to be spike free. In this lab, just try to make your design as small as possible.

```
ENTITY VideoTimingGen IS
PORT(
Clk : in std_logic;
Clk_en : in std_logic;
Reset_n : in std_logic;
Hsync : out std_logic;
Vsync : out std_logic;
Hcount : out std_logic_vector(10 downto 0);
Vcount : out std_logic_vector(10 downto 0);
Start_video : out std_logic;
Video_on : out std_logic
);
end VideoTimingGen;
```

- Clk : 25 MHz clock
- Clk_en : active high, it allows to deactivate the VideoTimingGen
- Reset_n : active low synchronous reset
- Hsync : horizontal synchronization pulse
- Vsync : vertical synchronization pulse

· Hcount : horizontal counter (from 0 tot 799)

· Vcount : vertical counter (from 0 tot 519)

· Start_video : one Clk period wide pulse coinciding with first Video_on of a screen

· Video_on : indicates when the RBG signals are to be driven

Note that the Hcount and Vcount ports in fact have too many bits for what we are trying to do here. Using 11 bits however allows for a smooth transition from VGA to SVGA later.

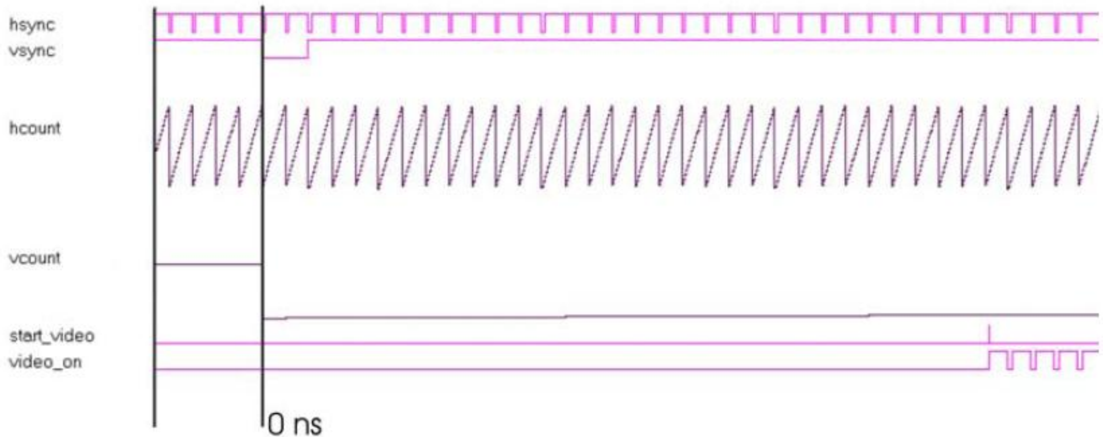


Figure 6

Figure 6 shows a simulation output of the system. The use of Hcount, Vcount and the Start_video signal will become clearer in subsequent modules of this lab.

Hints

· Generate a VHDL library Const_lib and compile the package Constants_pack in it.

The advantage of a constants package is that all your parameters are in a central place so that you only have to adapt one file when parameters change.

Internally in the architecture, use signals of unsigned type for the horizontal and vertical counter. To be able to do this you have to make the numeric_std package from the IEEE library visible. In the source code of this package [Modelsim installation folder under vhdl_src/ieee/mti_ numeric_std.vhd] you can find all functions and procedures that are pre-defined for this type.

 De advantages of the IEEE numeric_std package are:

1. Code portability;
2. You have to use signed and unsigned types for arithmetic operations;
3. Your code is more readable.

· You can “cast” signed or unsigned signals to std_logic_vector like this:

```
Hcount <= std_logic_vector(hcount_sig);
```

- Don’t’ forget to add the synchronous reset.

- Only generate registers for the horizontal and vertical counter. All other output signals should be combinatorial functions of those two counters.
- The architecture hence holds at least 2 processes: one describing the clocked behavior and another that describes the combinatorial functions.