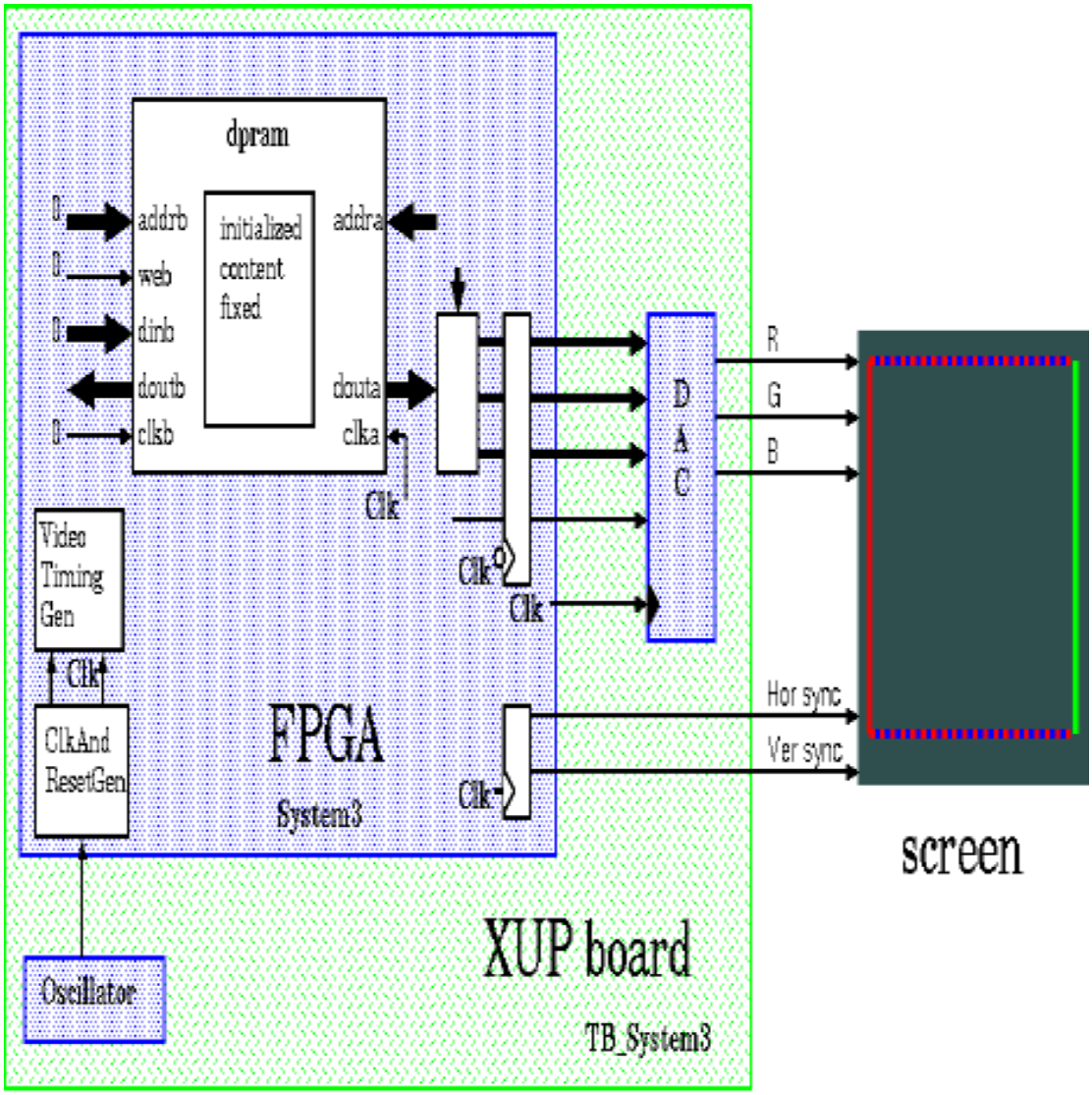
Lab-exercise

Lab6:
Driving a screen from
dual-port RAM

Introduction

In this lab we are going to drive a screen from a video RAM implemented in a dual-port RAM



Objectives

- Describing a full system in VHDL starting from existing code and the specifications of control logic in the form of timing views and hardware hints.
- Verifying this system by means of simulation

- Implementing the system on an FPGA.

Knowledge background

All previous modules of this lab.

Classification

Degree of difficulty (from 1 to 5): 4
Time needed (without support): 5 hours

Input

The following folders/files are available for module4b:
Hardware

- Constants_pack.vhd
- ClkAndResetGen.vhd
- VideoTimingGen.vhd
- dpram.vhd: result of module 4a
- System3.vhd: file to be adapted

Simulation

- dpram.mif : result of module4a
- TipleDac.vhd
- TB_System3.vhd

Implementation

- System3.ucf

The lab

In this lab, the RGB values have to be read from a dual-port RAM (dpram). Give all signal names at the A side of the dpram postfix Hard and all those connected to the B side postfix Soft. This is done to indicate that in modules following this one, the B side is going to be connected with a software system that will write values in the RAM. Per 32 bit word of the RAM we get color information for 10 pixels (cf. module 4a). At the hardware side, we hence have to read a new word every 10 pixels. This means we have to implement a 0 to 9 counter to select which bits of the RAM output (doutHard) have to be driving the Red, Green and Blue 8-bit busses. In our case we have only 3 color bits –

one for each color- which means a color is either fully on or fully off. In System3.vhd you should write a VHDL construct that drives signals Red, Green and Blue with the corresponding bits of dout_Hard as a function of the pixel counter (PixCntr) as shown in Figure 2

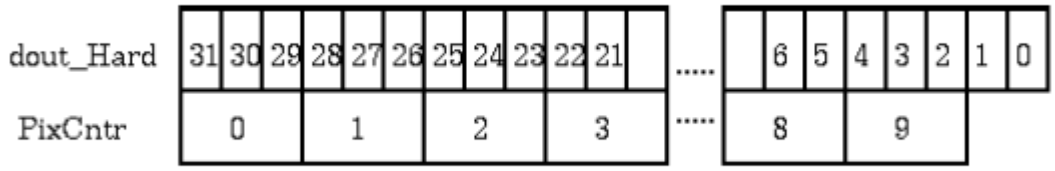


Figure 2

If e.g. PixCntr is zero and dout_Hard(31:29)=”101” then Red=”11111111”, Green=”00000000” en Blue=”11111111”. To avoid an extra one clock period delay, you should do this using combinatorial logic. The difficulty of the full system lies in reading the RAM correctly. Because a synchronous RAM has a one period clock delay, the address has to be available to the RAM during the previous clock period. In Figure 3 you can see that e.g. the address addr_Hard has to increment at the point where the pixel counter is equal to 8 to get the correct new output of the RAM (D1) on the right moment.

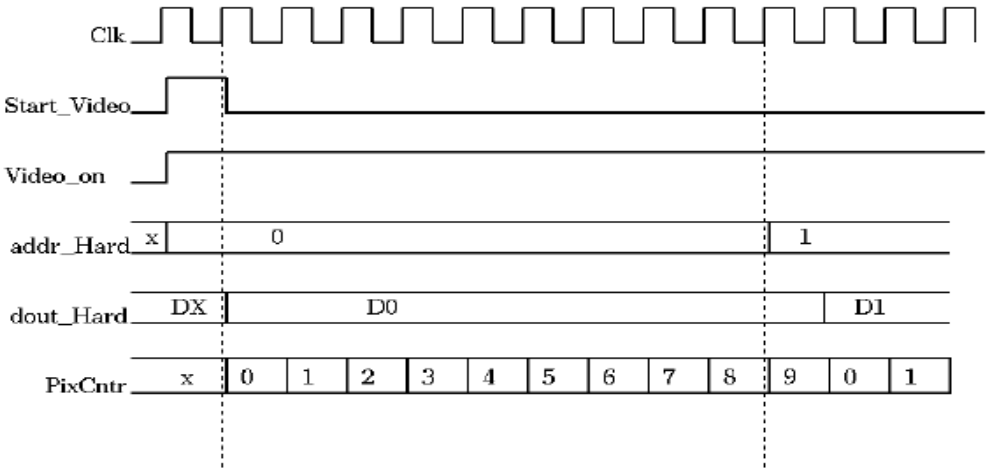


Figure 3

When the Start_Video pulse comes we want to provide address 0 as soon as possible to the RAM. If you want to keep everything purely synchronous (easier for testability if you are targeting ASICS, but also easier for timing analysis in both ASIC and FPGA flows) you have to implement addr_Hard as shown schematically in figure4. Without the _Hard process the address will come 1 period later and the timing of the whole system will become much more complicated.

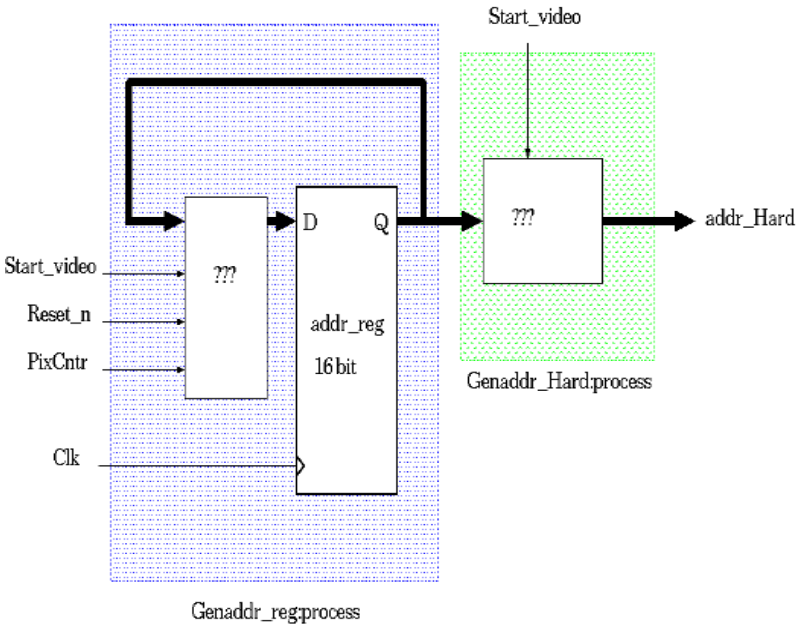


Figure 4

Even if we are using this way to generate addr_hard, you can see in figure 3 that dout_hard as well as the Red, Green and Blue busses get their values 1 clock later than the Video_on signal, To drive the screen correctly we hence also have to delay VGA_OUT_BLANK_Z, VGA_HSYNC and VGA_VSYNC with 1 clock period. Don't forget however, that we already need a delay of 1 clock on VGA_HSYNC and VGA_VSYNC to compensate the delay in the DAC. Eventually, you should not forget to retime the blank and RGB signals to the falling edge of the clock, just like in System2. Summarized, System3 should hold the following VHDL elements:

- Instantiation of ClkAndResetGen
- Instantiation of VideoTimingGen
- Instantiation of the dual port RAM
- Driver for the A (_Soft) side of the RAM with zeroes
- A process inferring flip-flops describing the PixCntr
- A process inferring flip-flops and a combinatorial one describing the addr_Hard functionality
- A VHDL statement to drive the Red, Green and Blue busses
- A process inferring flip-flops for delay compensation delays
- A process inferring flip-flops for the re-timing on the falling edge of the clock

Assignment:
1. Write the VHDL code for System3
2. Simulate System3
3. Implement System3
After implementation, the result should be a frame on the screen where the left boundary is red
and the right boundary is green. The top and bottom boundaries look like this:
rgbzrgbzrg rgbzrgbzrg rgbzrgbzrg rgbzrgbzrg rgbzrgbzrg rgbzrgbzrg rgbzrgbzrg
(r= red pixel g=green pixel,b= blue pixel,z=black pixel)

NOTICE：

必须在建完工程后再添加文件 不用添加 **EDN** 和 **NGC** 文件 如果在建工程时就添加文件 在 **translate** 时将会报错。 暂时还不知道原因。