Lab-exercise

Lab6:

Implementation of a moving block

Introduction

In the previous module a system was implemented that puts a fixed color on the whole screen. In this lab a second system System2 is going to be implemented, that is going to show a moving block on the screen.

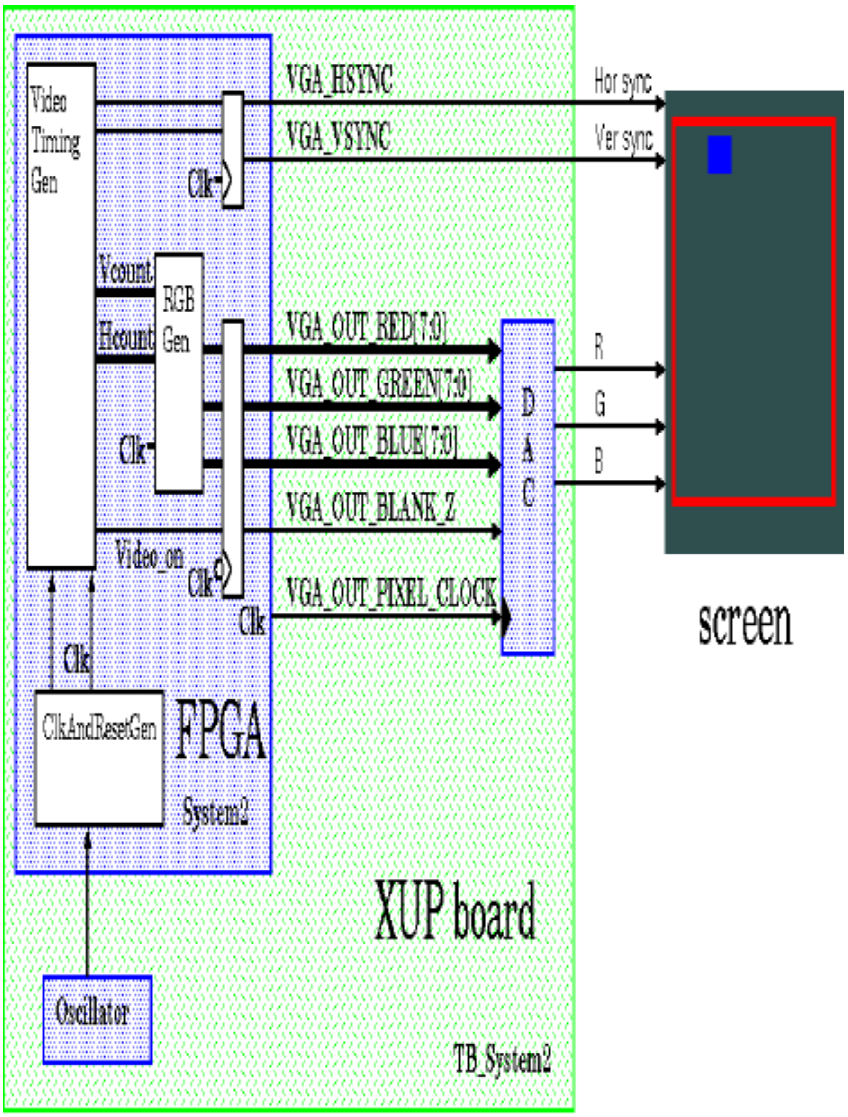


Figure 1

Goals

In the previous module the student had to implement System1 on an FPGA by way of a kind of walk-through. In this module the student has to expand System1 into System2 without much assistance. The student has to write a small hardware unit RGBGen in VHDL and implement the system on the FPGA. This lab module verifies if the student has understood the flow from

specification to implementation on an FPGA.

Knowledge background

Module2a should be done first.

Classification

Difficulty level (from 1 to 5): 4
Time needed (without support): 10 hours

Input

You'll find the following folders/files in the module3a folder.

- Hardware : An empty folder
- Simulation : all files needed to simulate System2
- Implementation : constraints file to implement System2

The lab

Implement a complete hardware system showing a red frame on the screen with in it a blue block of 20 by 10 pixels, starting on position (100,100) and moving under a 45 degree angle. (0,0) is the top-left coordinate. When the block 'hits' the frame, it bounces under a 90 degree angle.

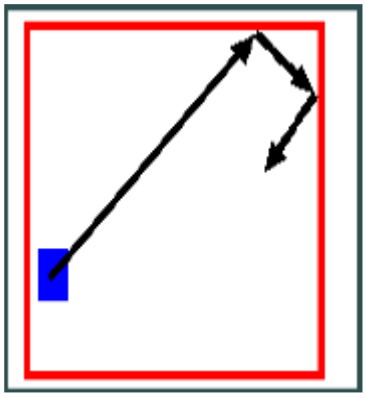
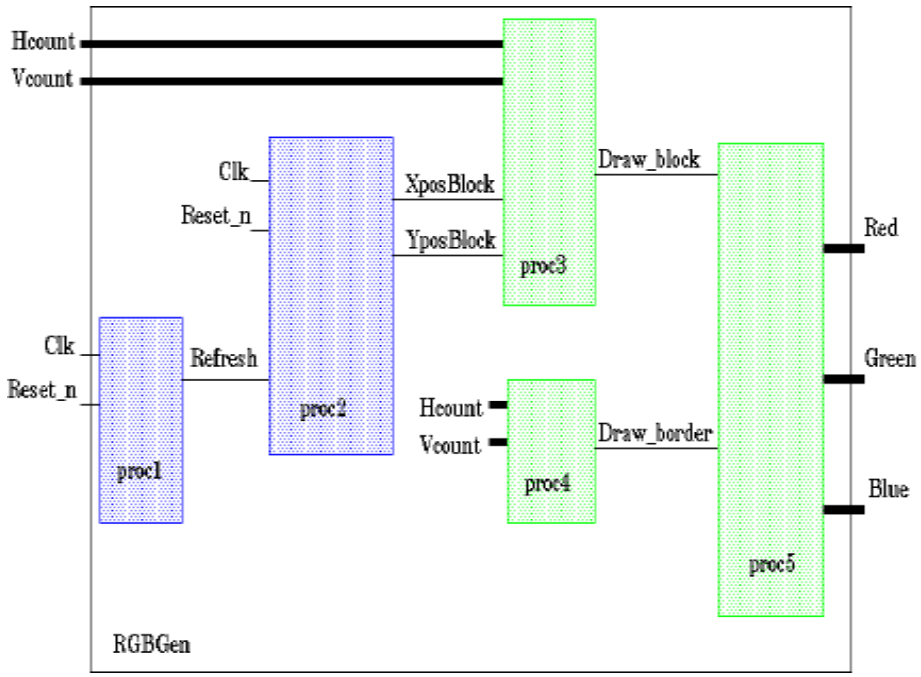


Figure 2

The difference with System1 is that RGB busses now are not getting a constant value, but that their value is defined by a hardware module (RGBGen). This module is going to drive the RGB busses as a function of the position of the blue block and the value of the horizontal and vertical counters in the VideoTimingGen module. These counters define which pixel is currently written. Even designing a system as simple as this one requires a structured approach. Figure 3 below helps you in defining a possible architecture for the RGBGen module.



The different blocks in this figure can be separate instantiated entities. Since their limited functionality however you can better make each of them a different process in the architecture description of System2.

From the code of process proc1 and proc2 flip-flops will be inferred. The other processes are purely combinatorial. In process proc1 a Refresh signal is described generating a puls of 1 Clk clock period with a frequency of around 100 Hz. This Refresh signal is used in the proc2 process to slow down the calculation of the new position of the blue block, so a human eye can see the block moving.

In proc2 the new position of the block is calculated (cf. figure 2). Process proc3 makes Draw_block high when the pixel generated by VideoTimingGen is within the boundaries of the blue block.

Process proc4 makes Draw_border high when the position of that pixel is within the red frame. And the last process, proc5, is defining the color. When Draw_block is high, blue is driven. When Draw_border is high, red is driven.

Hints

Use the following signal declarations in System2.

```
signal Clk : std_logic;
signal Reset_n : std_logic;
signal Hsync : std_logic;
signal Vsync : std_logic;
signal start_video : std_logic;
signal hor_count : std_logic_vector(10 downto 0);
```

```
signal ver_count : std_logic_vector(10 downto 0);
signal Video_on : std_logic;
signal Refresh : std_logic;
signal XposBlock : unsigned(10 downto 0);
signal YposBlock : unsigned(10 downto 0);
signal Up_down : std_logic; -- 1 = up
signal Left_right : std_logic; -- 1 = left
signal Draw_block : std_logic;
signal Draw_border : std_logic;
constant Line0_ver_count: integer := V_sync+V_bp;
constant Col0_hor_count : integer := H_sync+H_bp;
constant BlockHalfHeight: integer :=5;
constant BlockHalfWidth : integer :=10;
constant zero : std_logic := '0';
constant one : std_logic := '1';
```

Signal XposBlock and YposBlock are of type unsigned, allowing easy arithmetic or comparison with integers on them. Assigning an integer value to an unsigned can be done like this.

```
XposBlock <= to_unsigned(100,11);
```

This means that the integer value 100 is assigned to an 11 bits wide unsigned. The follwing is a possible description of proc4.

```
GenDraw_border : process(ver_count,hor_count)
begin
if to_integer(unsigned(ver_count))=Line0_ver_count or
to_integer(unsigned(ver_count))=Line0_ver_count+V_active-
1 or
to_integer(unsigned(hor_count))=Col0_hor_count or
to_integer(unsigned(hor_count))=Col0_hor_count+H_active-1
then
Draw_border <= '1';
else
Draw_border <= '0';
end if;
end process GenDraw_border;
```