VGA driven monitor with FPGA

FPGA implementation using a Pmod VGA port and a monitor

1st Di Maggio Alessandro
Università degli Studi di Torino
Dipartimento di Fisica
Turin, Italy
alessandro.dimaggio@edu.unito.it

2nd Monnetti Edoardo
Università degli Studi di Torino
Dipartimento di Fisica
Turin, Italy
edoardo.monnetti@edu.unito.it

***Abstract*—This report describes the method to drive a monitor with a Pmod VGA [1] port, using Verilog as an HDL for an FPGA, in order to show a colored square on a screen. The FPGA board is programmed to drive the VGA port through separate pins by transmitting the three color component signals and two synchronization signals (horizontal and vertical). The physical implementation and a bouncing ball game, as one application among several others, are described in this work.**

I. Introduction

The Pmod VGA [1] (Video Graphic Arrays) provides a VGA port to the FPGA board (Diligent Arty A7-35T) which is equipped with Pmod connectivity too. The VGA port can be used to drive standard displays such as televisions and monitors.
A VGA driven monitor requires five signals to display a picture or a video:

- R (red), G (green) and B (blue) - analog signals,
- HS (horizontal synchronization) and VS (vertical synchronization) - digital signals.

As shown in *Fig. 1* the Pmod VGA [1] port has twenty-four pins, divided in two 12-pin male connectors: since the intensity of red, green and blue is represented by a single hexadecimal digit every color channel has $4\ bits$, so that the total color output has $12\ bits$ going in 12 different pins.

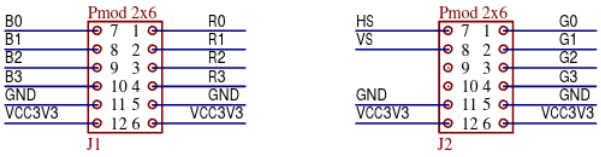

Fig. 1. 12-pin male connectors schematic.

Since the VGA port requires R, G and B to be analog signals and the FPGA controls the colors with bits, a digital-to-analog conversion is performed. Indeed a simple R-2R resistor ladder DAC is implemented among every RGB-dedicated Pmod pin and the actual VGA port.
The monitor adopted for the implementation has a VGA connection with a 640x480 $pixels$ display working with a 60 Hz refresh rate. This is the actual resolution of the pixels that the display is made of. However before every screen refresh there is a waiting time where said pixels are not modified. Since a video is made out of a stream of frames and each frame is made up of a series of horizontal lines (which in turn are composed by pixels), HS and VS are used to define the ends of each line and frame. The screen begins a new line when it receives HS and begins a new frame with VS. Both are active-low signals that are generated in order to make sure that everything is synchronized and ready for the monitor to start a new line (or frame). These sync signals are part of blanking intervals which are made of three parts, as *Fig. 2* shows: front porch (16 $pixels$ for HS and 10 $pixels$ for VS), sync (96 $pixels$ for HS and 2 $pixels$ for VS) and back porch (48 $pixels$ for HS and 33 $pixels$ for VS). Since each positive clock's edge corresponds to a pixel in the active region, these blanking intervals are also considered to be composed of pixels, even if not physical ones. Having a total of 800x525 $pixels$ means that the screen is updating with a pixel frequency of 25.2 MHz in order to refresh with a 60 Hz rate.

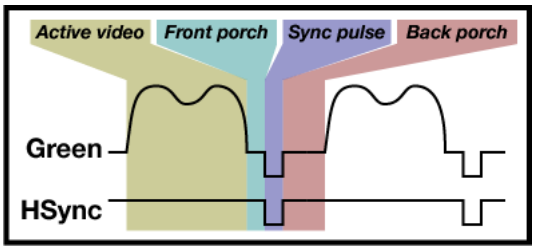

Fig. 2. In the active region RGB values are transmitted to each pixel of a line; in the blanking region only black pixel are transmitted.

In this work, a simple picture of an orange square has been drawn so that the functioning of the implementation could be tested.

II. Code implementation

To implement this system on FPGA using Verilog, three main modules are needed: the first one serves as an HS/VS generator in which all display timings are included, the second deals with the square picture and the last one is the

VGA top module which includes the first two.
A PLL is required to generate the 25.2 MHz pixel clock. As of display timings in the sync signals generator module, two counters are needed to decide on the correct time for synchronization: one for horizontal position (x-axis) and another for vertical position (y-axis). Everything is referred to the pixel clock: as the horizontal counter is incremented with every clock's tick, it eventually reaches the end of a line (800 $pixels$) and then it restarts from zero. The same goes for the vertical counter when the end of the frame is reached (525 $pixels$ along the y-axis): since this counter increases only at the end of every line, VS has lower frequency than HS. Both signals are then generated, considering their active low feature: they are set to zero while the position counters are between the front porch and the back porch. The active region remains in the 640x480 $pixels$ frame.
A 32x32 $pixels$ orange square is then drawn in the top-left corner of the active region: when the screen coordinates are both less than 32, orange is drawn (given the right combination of color intensity among R, G and B); if not, blue is drawn. To do so, a variable that is high in the 32x32 area (otherwise is low) is required. It is also important to make sure that the counters are in the active region of pixels and not in the blanking interval, otherwise the display may be misaligned.

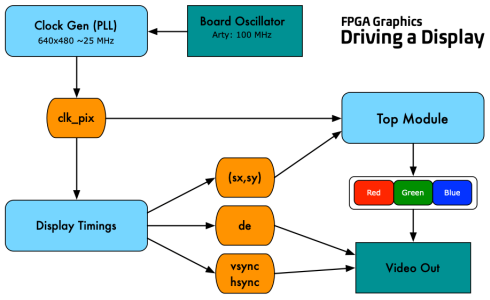

Fig. 3. Block diagram for the whole code.

III. Simulation

A dedicated testbench for sync signals is written in order to simulate the system using Vivado Desing Suite [2], by Xilinx ; in particular, the main stimulus aims to test the display timings after a synchronous reset signal is released.
Fig. 4 shows the waveforms diagram out of a 0.1 μs simulation: a longer simulation is not practical to show as the time to reach the end of the display and see a screen refresh is ~ 16 ms. In the first shot, after the reset is released HS and VS go high as the counters enter the active region. Most importantly it shows that, as long as the horizontal counter is less than 32 (that is 32 clock's edges), red and green are active, giving the orange color to the first 32 pixels. After that, blue (with green) is printed for the remaining pixels. The same would go for the rest of the lines, making up the whole picture.
The second shot is useful to show that, when the horizontal counter reaches the end of the line, HS goes low and all the three colors are turned to zero making up the black refresh frame.

IV. Build

The implementation on FPGA is carried out by a Vivado Synthesis [2] based on the Arty A7-35T xc7a35ticsg324-1L [5] board.
Resource utilization for this implementation is summarised in the following table.

Site Type	*Used*	*Available*	*Util%*
Slice LUTs	28	20800	0.13
Slice Registers	26	41600	0.06
Slice	13	8150	0.16
LUT as Logic	288	20800	0.13
Bonded IOB	16	210	7.62
BUFGCTRL	2	32	6.25
PLLE2_ADV	1	5	20

The primitives that are being used are reported below.

Ref Name	*Used*	*Functional Category*
FDRE	26	Flop & Latch
OBUF	14	IO
LUT6	11	LUT
LUT5	10	LUT
LUT4	6	LUT
LUT3	5	LUT
LUT1	4	LUT
LUT2	3	LUT
IBUF	2	IO
BUFG	2	Clock
PLL2_ADV	1	Clock

This specific board is equipped with an integrated crystal oscillator that generates the 100 MHz clock going into the PLL.
The timing analysis performed by the tool gives the following results.

WNS(ns)	32.684
TNS(ns)	0.000
TNS Failing E.P.	0
TNS Total E.P.	57
WHS(ns)	0.033
THS(ns)	0.000
THS Failing E.P.	0
THS Total E.P.	57
WPWS(ns)	3.000
TPWS(ns)	0.000
TPWS Failing E.P.	0
TPWS Total E.P.	32

These datas show that this implementation should not have timing issues. There are not Failing End Points (i.e. Failing E.P.) and the Total Negative Slack (i.e. TNS) is equal to 0.000 ns, so that every specified timing constraint is met.

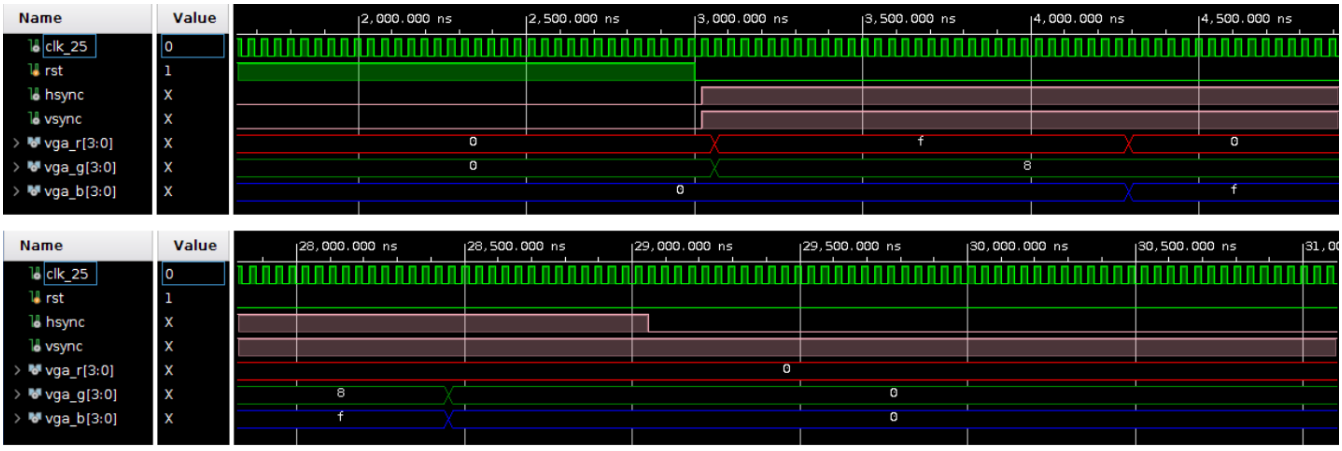

Fig. 4. Waveforms from a Vivado simulation.

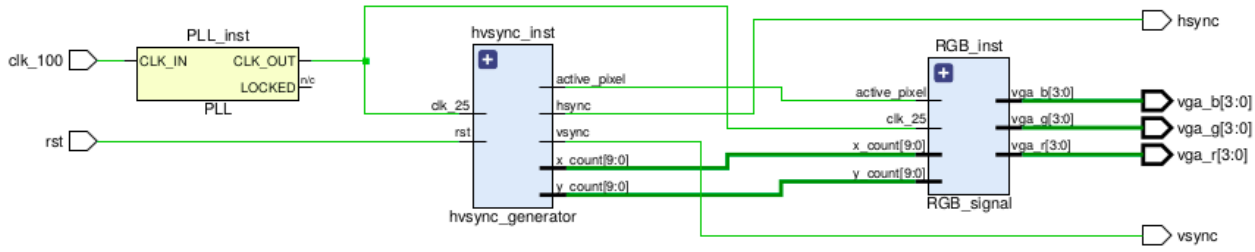

Fig. 5. Block diagram.

V. Constraints

The clock signal supplied by the PLL for this implementation comes with the following parameters:

Clock	*Waveform(ns)*	*Period(ns)*	*Frequency(MHz)*
CLK_OUT_PLL1	{0.000 19.839}	39.677	25.203

As regards the electrical constraints, the Configuration Bank Voltage Select (CFGBVS) is set to 3.3 V. This parameter determines the I/O voltage operating range and voltage tolerance for the dedicated configuration bank.

VI. Hardware implementation

After the Vivado [2] flow is completed, a real hardware implementation is carried out in order to graphically show the results. The Pmod VGA [1] is connected to the monitor's VGA port through an appropriate cable.

VII. Application

Bouncing ball

As a further real-life application a color-changing ball bouncing off all four sides of the screen has been implemented. To generate the ball and its movements another Verilog module is required. Clock is the same as before since the position of the ball is updated once per frame: there is a dedicated variable that goes high at the start of every vertical sync blanking, enabling the animation of the moving ball by updating its location for the next frame. To make this happen there is need for three variables that control the instant position of the ball, its next direction and its constant speed: all these variables come in couples since both horizontal and vertical coordinates are necessary in order to describe the ball's motion. When the x-direction (y-direction) is set to zero, the position variable is increased by the speed value, meaning that the ball is moving right (down); when the x-direction (y-direction) is set to one, the position variable is decreased so that the ball is moving left (up). As soon as one of the two position coordinates reaches a screen side the ball bounces since an appropriate flag becomes true and makes the direction change its former value.
Another feature for this application, is that the ball changes color everytime it bounces off a screen side: to do so, a finite state machine is included in the above Verilog module. As

a first step, a state counter is written, every state being one of the following colors: red, orange, yellow, green, light blue, blue and purple. The combinational part of the machine is then described with a simple Verilog *case* construct that switches among the seven states: every state in turn contains an RGB combination to make up the desired color. The next state logic is then updated at every clock tick.
The hardware implementation shows again the expected results on the screen.

VIII. CONCLUSIONS

The goal of this work is to drive the VGA port of a real monitor using a Verilog programmed FPGA board.
The Vivado Design Suite's implementation [2] flow has been followed: the code that generates the requested signals has been written and compiled. Then a simulation of the system has been carried out and the implementation has been completed through the usual building and placing steps. Finally a timing analysis has been performed to make sure that the system would not meet any timing issue.
The real hardware implementation has been succesfull both with the square picture and the bouncing ball, showing the desired results directly on the screen.

REFERENCES

[1] **Pmod VGA by Diligent**, https://digilent.com/reference/pmod/pmodvga/start?redirect=1
[2] **Vivado Design Suite Documentation**, https://www.xilinx.com/products/design-tools/vivado.html
[3] **Project F - FPGA Development - FPGA Graphic**, https://projectf.io/posts/fpga-graphics/
[4] **Project F - FPGA Development - Pong**, https://projectf.io/posts/fpga-pong/
[5] **Arty A7-35T xc7a35ticsg324-1L**, https://www.xilinx.com/products/boards-and-kits/1-elhaap.html