Department of Electric & Electronic Engineering,
Boğaziçi University

To give an explanation about FPGA implementation of a VGA screen saver

EE240 FINAL PROJECT REPORT

Aras Güngöre 2018401117
Aybars Manav 2018401099

Project Advisor: Şenol Mutlu

30.06.2021

Table of Contents

1 INTRODUCTION

A screensaver (or screen saver) is a computer program that blanks the screen or fills it with moving images or patterns when the computer has been idle for a long time. The original purpose of screensavers was to prevent phosphor burn-in on CRT, plasma and OLED computer monitors (hence the name). Though modern monitors are not susceptible to this issue, screensavers are still used for other purposes.[1]
In this project, we program an FPGA to implement a screen saver on a VGA monitor. A simple implementation which covers all the basics of a real screen saver process.

It is a thrilling adventure for the viewers who patiently wait for the icon to ultimately hit the corners!

2 PROBLEM STATEMENT

In this project, our goal was to develop an old school microsoft style screen saver in VHDL and implement it on Spartan6 FPGA.

3 RELATED BACKGROUND

The screensaver consists of a square 41x41 icon which shifts through different RGB colors in every movement. The icon bounces off the screen walls and viewers enthusiastically expect it to hit the corners!

4 DESIGN

In overall, this project contains 4 VHDL modules:

4.1 Clock Division

These modules are simply clock dividers for getting 60Hz and 25MHz clock frequencies. We implemented the clock dividers by keeping a counter that increments with the board_clock positive edge which is 100MHz. We have divided the clock by 416666 and 4 for 60Hz and 25MHz clock outputs respectively.

4.2 VGA Driver

This module is for designing a standard 640 x 480 screen with 60Hz refresh rate. We have kept the front porch, back porch, display time and pulse width (retrace) line widths in constant integers. Our alignment of intervals is as shown in Figure 1. Starting from display time we increment hpos with every 25MHz positive clock edge until we reach the end of back porch, after that we reset the hpos. We increment vpos with every 25MHz clock positive edge when hpos reaches the end of its back porch and we reset the vpos when it also reaches its back porch's end.
We create a video_on signal during the interval where both vpos and hpos are in their display time to handle blanking intervals.
Module outputs video_enable signal to color generator module and hsync, vsync signals for main module's output.

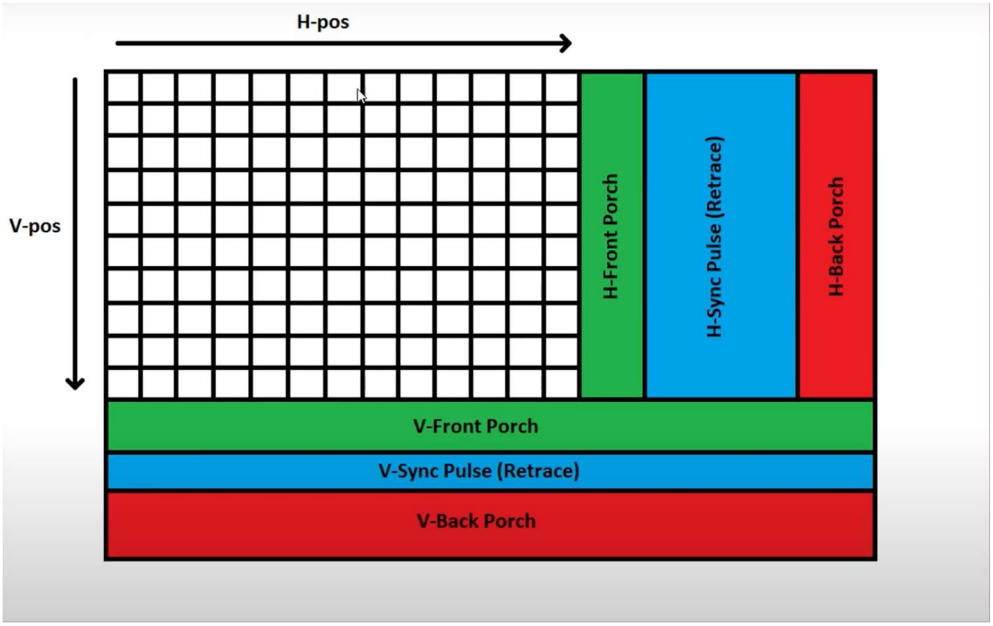

Figure 1: Hpos and Vpos template.

4.3 Square Position Updating Module

This module takes the 60 Hz clock signal as the Enable input signal and outputs the x and y positions of the 41x41 icon. The direction of the icon is stored as a 2-bit logic vector and the position of the ball is incremented along the given direction of the ball by 1. The position of the ball is updated when the internal 3-bit count signal has reached "111". So the position updating takes place once in every 8 frames. Meaning that, the ball moves 60 / 8 = 7.5 pixels in both horizontal and vertical axes so the moving angle is 45 degrees.

4.4 Color Generator (The Shape)

There are two counters included in this module. The first counter tracks the index of the last pixel which is displayed on video. The second counter tracks the index of the last pixel whether it is displayed on video or not. When the second counter hits the max limit, both counters reset to 0. There are 480x640 displayed pixels and 721x800 total pixels in one frame. About a quarter of the total pixels are invisible to the viewer, so we have to keep a separate counter for the latter.

In an indexing scale where the pixels are enumerated from 1 to 480x640, the index of the position of the left corner of the square can be expressed as "x + y * 640". We can determine whether the incoming pixel is in the square range or not by checking if the current pixel index is in the range "square_pos + i*screen_width < curr_pixel_index < square_pos + square_width + i*screen_width" for any i = {0, 1, 2, …, square_height}. If the current pixel index is not in any of these ranges, we can determine that the pixel doesn't belong to the icon so it must be black.

5 RESULTS

After many trials and errors, syntax errors and research we have finally got a square bouncing off the edges of the screen edges. We also made the square to change colors.

6 CONCLUSION

It was a frustrating but nevertheless exciting journey full of research as we did not get to learn fully coding VGA monitors in our labs. Even though we have chosen a simple project in the usual sense, we still think it covers most of the fundamentals of creating animations with VGA programming. We had to learn a lot of things before starting the project.

At first displaying a stable square has proven to be a challenging enough task in itself. Once we successfully displayed a stable square in VGA, displaying an animated square then changing its color while it was moving was not hard as it was to us before. Since the overall implementation is similar to Pong in the sense of a square bouncing off walls, we researched the Pong codes to give us a better idea. We combined what we have learned on Lab 7 with what we have learned about game design.

Coding these modules and fixing several bugs during the process, which was extremely frustrating, we found out that we had come a long way. After having completed the design, we concluded our work in an overall module and generated the programming file.

We tested our design with passion and enthusiasm; however, it was not very successful at the first several attempts. Despite that fact, we didn't give in and debug, searched online where we come along with problems, get ideas and in the end, apply to our design. Hopefully, we solved our problems and we felt very happy when we saw our design actually functions!

We got addicted to our screensaver already, let's see if you will be, too!

7 REFERENCES

1. https://en.wikipedia.org/wiki/Screensaver
2. FPGA BASED VGA DRIVER AND ARCADE GAME, ARMANDAS JARUSAUSKAS, 2009 (For research purposes)
3. FPGA PROTOTYPING BY VHDL EXAMPLES, Xilinx Spartan-3 Version, Pong P. Chu, Cleveland State University, 2008 (For research purposes)