EE 314 DIGITAL CIRCUITS LABORATORY 2022-2023 SPRING TERM PROJECT REPORT

FPGA IMPLEMANTATION OF A 2D STRATEGY GAME

Ahmet Caner Akar
Electrical and Electronics Engineering Department
Middle East Technical University
Ankara, Turkey
e244228@metu.edu.tr

Osama Awad
Electrical and Electronics Engineering Department
Middle East Technical University
Ankara, Turkey
e248849@metu.edu.tr

İsmail Enes Bülbül
Electrical and Electronics Engineering Department
Middle East Technical University
Ankara, Turkey
e244263@metu.edu.tr

***Abstract*—This document is about the end-term project of EE314 Digital Circuits Laboratory, implementation of a 2D strategy game by using FPGA.**

***Index Terms*—FPGA, Verilog HDL, VGA driver, button debouncing, state-machine**

I. Introduction

II. Project Overview

A. VGA Module

In this project, we are expected to use the VGA interface of the FPGA to display the game board and some information related to the game. To do this, we decided to display the game with 640x480 resolution at 60 Hz. [1] The timing diagram is shown in Figure 1, below.

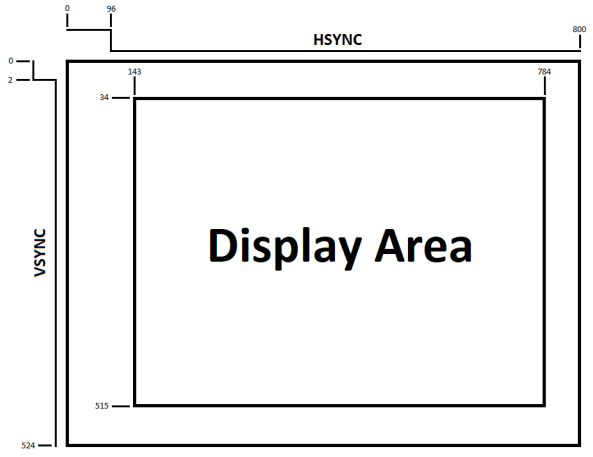


Fig. 1. Timing diagram for the VGA interface

From the timing diagram, we can see that we need two counters namely a horizontal counter and a vertical counter. To drive counters we need a clock with 25 MHz frequency. To obtain a clock signal with that frequency from the internal clock of the FPGA which has a 50 MHz frequency we implemented a clock divider circuit. After we obtained the clock signal with a 25 MHz frequency, we connected it to the horizontal counter. Horizontal counter counts at every clock cycle, and it counts from 0 to 800. Vertical counter counts 1 when the horizontal counter hits 800. We have two synchronization outputs namely HYSNC and VSYNC to tell the monitor beginning and ending of a row and frame respectively. HSYNC is high when the horizontal counter is between 0 and 96, and VSYNC is high when the vertical counter is between 0 and 2. Altera FPGA uses a digital-to-analog converter that takes 8-bit input for each color channel and converts it to an analog signal that has a voltage between 0 and 5 Volts. [2]

When the horizontal counter is between 143 and 784, and the vertical counter is between 34 and 515, RGB values of the pixel at the point (hcounter - 143, ycounter - 34) are returned to the DAC of the FPGA. We implemented a *rgbSelector* module to get the RGB values for a given coordinate on the display. If the given coordinate is in the board region, the module finds the corresponding block coordinate and reads which piece is placed to that coordinate from the memory. Then it gets the RGB value from the memory module that corresponds to the piece. If the given coordinate is not in the board region but in a region that involves a text or picture, it reads the RGB value from the corresponding memory. If it is neither in the board region nor a picture or text region, then it returns a white color.

Since we need only 5 colors (white, black, red, green, and blue), we encoded and stored pictures that we used with 1 bit for each color channel. After the *rgbSelector* module returns RGB values with 1 bit for each channel, another module we called *vgaDecoder* converts it to output with 8 bits for each color channel and returns it to the DAC of the FPGA.

When the counters are not in the display area, 0 is returned to the DAC of the FPGA. An example of the display is shown in Figure 2, below.

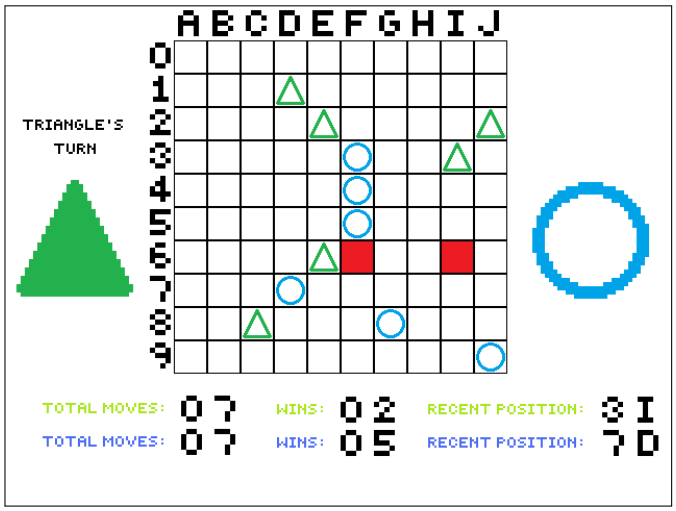


Fig. 2. Example of the display

B. Button Debouncing and Edge Detector

In this project, we need to get three inputs, **logic 1**, **logic 0**, and **activity**, from the user by using push buttons on the FPGA. However, due to mechanical and physical issues, pushbuttons often generate noisy signals called dirty bounces, and these bounces prevent us from properly triggering the program. Thus, to eliminate these undesirable effects, we used the *debouncer_delayed* module that makes a noisy pushbutton input signal to the ideal input case.

The working principle of the *debouncer_delayed* module is quite simple. When a button is pressed, the timer is going to count the elapsed time up to a predefined threshold parameter. If the timer hits the threshold value, the program concludes that the button has reached its steady state and has been pressed. Similarly, when the button is released and the steady state is reached, the program concludes that the button has been released.

After debouncing the button input, we designed another module called *edge_detector* to detect the negative and positive edges of the debounced input so that we will use the edge signals directly as button signals in the game controller.

The *button* module contains both *debouncer_delayed* and *edge_detector* modules so that the hierarchical design principle is followed throughout the project. Also, both of these modules, are written like a state machine to make it easier to implement condition-based and flexible code. The waveform simulation result of the *button* module in Quartus II is given in Figure 3. Besides these, in the *button_top* module each button is defined separately in a hierarchical manner by using the *button* module.

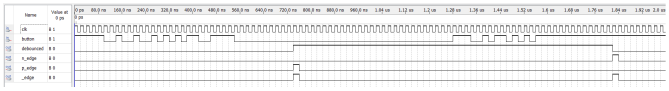
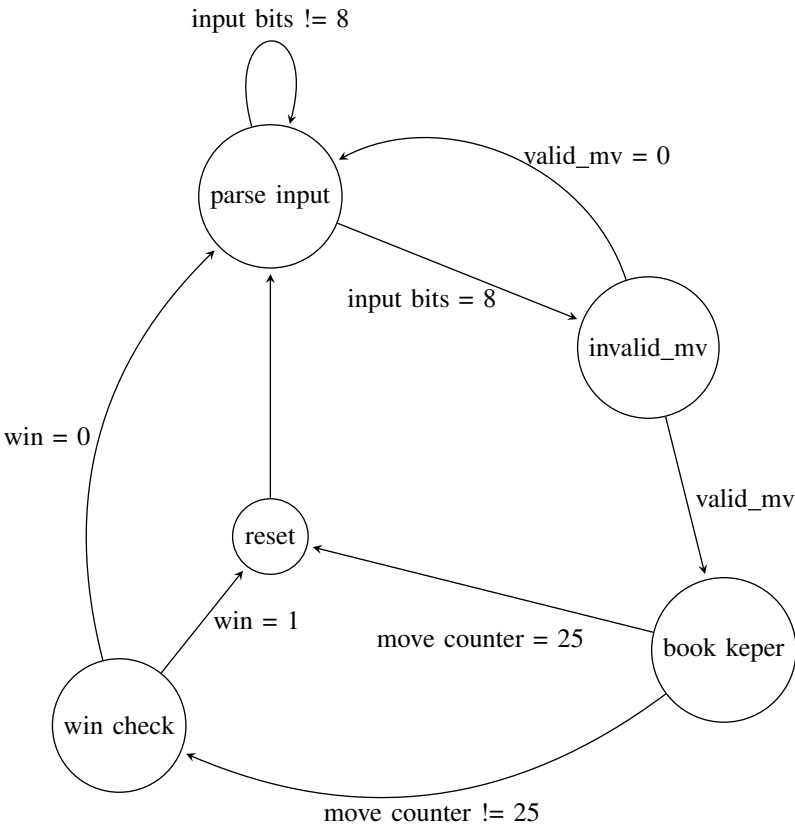


Fig. 3. Button module, waveform simulation result

C. Game Controller



III. Conclusion

References

[1] "Vga signal 640 x 480 @ 60 hz industry standard timing." http://tinyvga.com/vga-timing/640x480@60Hz.

[2] terasIC, *DE1-SoC*, 2014. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836&PartNo=4#contents.