Department of Electric & Electronic Engineering,
Boğaziçi University

EE240 SPRING 2018 FINAL PROJECT

PIXEL EDITOR

Umut Eren Usturalı
2016401234
erenust@gmail.com

Abdurrahman Berat Sert
2015401069
beratsert41@gmail.com

31.05.2018

Table of Contents

1 INTRODUCTION

In this document, we report our work on the final project assignment for EE 240. After briefly summarizing the original proposal, we explain the final design and form of the project, as well as touch on a few technical points. The simulations, synthesis report and the final live test results are discussed in the end.

2 ORIGINAL PROPOSAL

The goal in this project was to realize a pixel ed itor that allows the user to work on a 32x32 8-bit color image using a VGA screen and a Nexys-3 FPGA. It was initially planned that the VGA screen would have a 640x480 resolution and the pixels on the image would correspond 10x10 squares on the screen. In this design, the user would select a pixel on the image using the left, right, up and down push buttons, with the central push button enabling the user to color the selected pixel with the color specified by switch buttons on the FPGA board.

3 TOOLS USED

The project was made using Nexys-3 Spartan 6 FPGA. The VHDL code written for the project was synthesized and converted to a bitstream file using Xilinx ISE software package.

4 DESIGN

4.1 User Interface

The main input devices are the switch buttons and the push buttons on the FPGA. The user interface consists of the color sliders, the color bar, the canvas and the clear button on the screen. The part of the interface which includes the color bar, the color selection sliders and the clear button is called the setting interface. The part of the interface which consists of the canvas is called image editing interface. The leftmost switch button is the active-low global reset button which should be turned on to start the pixel editor. The rightmost switch button controls which part of the user interface is active. If this switch is turned on, the setting interface is activated. Otherwise, the image editing interface is active. When the settings interface is active, the up and down push buttons switch between the color sliders and the clear button. When one of the color sliders is selected, it will have a brighter appearance and pressing left and right push buttons in this state will change the amount of the corresponding color present in the brush color. When the clear button is selected, it will have an orange color and pressing the middle push button in this state will clear the canvas, locking the user interface for a very brief time period which is a few microseconds long. When the image editing is activated, the directional push buttons move the brush on the image. The position of the brush is indicated on the screen with white borders around the selected pixel. Pressing the middle button colors the selected pixel with the color selected using the settings interface.

This final design is different from the initial plan in a number of ways. The initial design did not provide a global reset button which could potentially cause the circuit to start in an invalid state. The addition of such a global reset switch made it impossible to reserve the switch buttons for color input, since there were only seven more of them on the FPGA board. This, and the fact that switches were an inconvenient method of input, were the reasons that

the settings interface and the color sliders were added to the design. This addition gave the user a much better way of specifying brush color.

4.2 Circuit Modules

The circuit is divided into several modules. A schematic of the circuit generated by Xilinx ISE can be seen in the figure in the appendix. The names of the modules in the code and their functions are as follows:

- **button_control**: This module takes the raw input from a single button and works as a debouncer. It updates an internal counter every clock cycle if the raw input is high. When the counter reaches a certain threshold, its output is set to high for a single clock cycle. This module is initialized five times; once for each of the push buttons.
- **settings_ui**: This module controls the settings interface. It is activated via an enable input signal when the *img_clear* module is inactive and the rightmost switch is turned on. It takes the outputs of the *button_control* modules as inputs. Its outputs are the brush color, the clear command signal and the selected UI element signal.
- **image_control**: This module controls the canvas. It is activated via an enable input if the *image_clear* module is inactive and the rightmost switch is turned off. It takes the outputs of the *button_control* modules as inputs. Its outputs are horizontal and vertical positions of the brush (i.e. the selected pixel) on the canvas and the signals required to communicate with the image ram when a write operation needs to be done on the image.
- **img_ram**: This module stores the 32x32 image shown on the canvas. It has the inputs for a synchronous write operation and an asynchronous read operation. The synchronous write is used by *image_control* and the *image_clear* modules and the asynchronous read is used by the *pixel_count* module. See 4.2.1 for further details on this module.
- **freq_div**: This module lowers the 100 MHz board clock signal to 25 MHz for the modules which drive the VGA output.
- **hsync_gen**: Generates the horizontal synchronization signal for the VGA output. Also, it generates an enable and a draw signal. The enable signal is connected to the *pixel_count* module to signal when the horizontal position should be updated. The draw signal value indicates whether pixel color data can be sent through the VGA output.
- **vsync_gen**: Generates the vertical synchronization signal for the VGA output. Also, it generates an enable and a draw signal. The enable signal is connected to the *pixel_count* module to signal when the vertical position should be updated. The draw signal value indicates whether pixel color data can be sent through the VGA output.
- **pixel_count**: Takes as input the vertical and horizontal enable signals and all the outputs of the *image_control* and *settings_ui* modules. This module determines the position of the pixel drawn on the screen at any given moment and to which part of the screen it belongs to by updating several counters. Then, based on its current state, it sets its output to the color data that should be sent through the VGA connection. See section 4.2.2 for a more detailed explanation of the operation of this module.
- **img_clear**: This module is activated when a signal is sent from the *settings_ui* module. Once this signal is sent the *img_clear* module sets its state to active. After this, at each clock cycle, it selects the next pixel on the image and sets its value in the *img_ram* to 0, which corresponds to the color black. Once it has looped through all the pixels its state is once again set to inactive. See section 4.2.3 for further point regarding this module.

4.2.1 The *img_ram* module

The VHDL code for the *img_ram* module has three important facets. The first of these is the usage of an array type for actual storage. This array type is a 32-element long array of arrays of 32 8-bit logic vectors; meaning that it stores a total of 1024 bytes. The second important part of the module is the synchronization of the write operation with the rising edge of the clock signal. The last critical point is the asynchronous read operation. This module is composed along these three important points to ensure that it is synthesized as a distributed dual-port ram and not as arrays of flip-flops, since this would strain the capacity of the FPGA. The asynchronous read operation could be changed for a synchronous version in order to synthesize block RAMs, which allow faster clock rates. However, this was found unnecessary as the timing constraints were already met and the asynchronous read provides a much more convenient interface for the *pixel_count* module, which has to access the contents of the image when the canvas is being drawn.

4.2.2 The operation of the *pixel_count* module

The pixel_count module keeps track of which pixel is being drawn on the screen at any given time. It is synchronized with the 25 MHz pixel clock which also drives the *vsync_gen* and *hsync_gen* modules. It is comprised of two processes, one which is sequential and one of which is combinational. The former of these updates the state of the module and the latter generates its outputs depending on the current state.

By keeping track of the position being drawn on the screen, it can be determined which component is being drawn at any given time. The basic mechanism for this works in the following way: For each component being drawn, two bits are stored in registers. The first of these is called the “begin” signal, which is set to high when the first pixel of the component is drawn and is reset when the last pixel of the component is drawn. (The first pixel for a rectangular area on the screen is the upper left corner, whereas the last pixel is the lower right corner.) The second signal is the “enable” signal which is set to high when the first pixel of the component is drawn and reset when the right side is reached. It is set to high once again when the left side of the component is reached if the “begin” signal is high, and reset again when the rightmost pixel in the current row is drawn. From these rules for state updates; it can be seen that the “enable” signal is high when a pixel of the corresponding component is drawn, and is low otherwise. The “enable” signal can thus be used by a combinational circuit to determine which part is being drawn.

Each of the pixels in the 32x32 image are shown as 9-pixel-by-12-pixel rectangles on the screen. The *pixel_count* needs to keep track of which of these 1024 pixels is being drawn, and to do so it must also keep track of which pixel in the corresponding 9x12 area is being drawn. To this end, four integer counters used in the *pixel_count* module; two of which are the horizontal and the vertical “in-pixel” positions in the 9x12 area and two of which are the horizontal and the vertical “in-image” positions on the image. The horizontal in-image position is updated when the horizontal in-pixel positions reaches its maximum value and is reset to zero; the vertical in-pixel position is updated when the the horizontal in-image position reaches its maximum value and is reset to zero; and finally, the vertical in-image position is updated once the vertical in-pixel position reaches its maximum value and is reset to zero. This method keeps track of the various positions without needing comparators and integer multiplication and division circuits.

The position of the slider thumbs (in the VHDL code, referred to as the “tip”) are calculated using multiplication and addition circuits and the values are inferred from the brush color input for the module. The mechanism of begin and enable drawing is not used for these; and when they are to be drawn is determined by doing comparisons.

The clear button is drawn on a certain rectangular area using a bitmask. When the rectangular are for the button is being drawn, the bitmask is checked to see whether the current pixel belongs to the text image or to the background. If the current pixel belongs to the text, it is drawn white when the clear button is unselected and yellow when it is selected. This bitmask is a constant double array in the VHDL code and is synthesized as a ROM in the FPGA. It was generated using a simple Python script and a PNG image of the text.

All of the positions for the components are declared as constants in the VHDL files. Note that the canvas and the pixels inside it are drawn as compressed squares so that they would appear as squares on a 16:9 widescreen monitor.

4.2.3 The *img_clear* module

This module is activated either when the clear command signal, which is an output of the *settings_ui* module, is set to high for a single clock cycle or when the global reset is set to low. Once activated, all other input modules are locked temporarily until this module completes its operation. When it is active, this module writes to a different pixel in the edited image at each clock cycle and resets its contents.

5 RESULTS

5.1 Simulation Results

The first simulation was done on the hsync_gen and vsync_gen modules using ISim. The timings for the front porch, back porch, synchronization pulse and display time were found to be correct.

Since it is another critical part, the pixel_count was also simulated. It was observed that it gave the correct color values for various points on the screen. In the simulation, the pixel_count was enabled for counting for the entirety of the simulation since it was only necessary to see if it gave the correct color values for the points in the screen.

Finally, the main circuit was tested in a 50ms simulation. Two full frames were simulated this way. It was seen that all circuit elements worked in cohesion and outputs were correct.

Some results from the simulations are shown in the appendices.

5.2 Synthesis results

The synthesis report generated by the Xilinx ISE indicated that a total of 270 slice registers out of a maximum of 18224 and a total of 1021 slice LUTs out of a maximum of 9112 were used. Of the 1021 slice LUTs, 765 were used as logic and 256 were used as memory. The report also shows that 270 flip-flops and 96 RAMs were used. Overall utilization ratios were relatively low, which suggests that a smaller FPGA could have been used for this application.

The synthesis report also includes information on the timings. It is indicated on the report that the clock rate for the whole circuit could be at most 208 MHz, which is well above the necessary 100 MHz.

5.3 Live Demo

The pixel editor was used to create a copy of a simple pixel art image. The push buttons were tested to see whether they were sufficiently responsive to inputs. The settings interface could be activated expected and the brush color selection worked as intended. The clear button functioned correctly. The brush movement on the canvas worked properly. The

test image created by the tester and the final version of the project can be seen in the picture below.

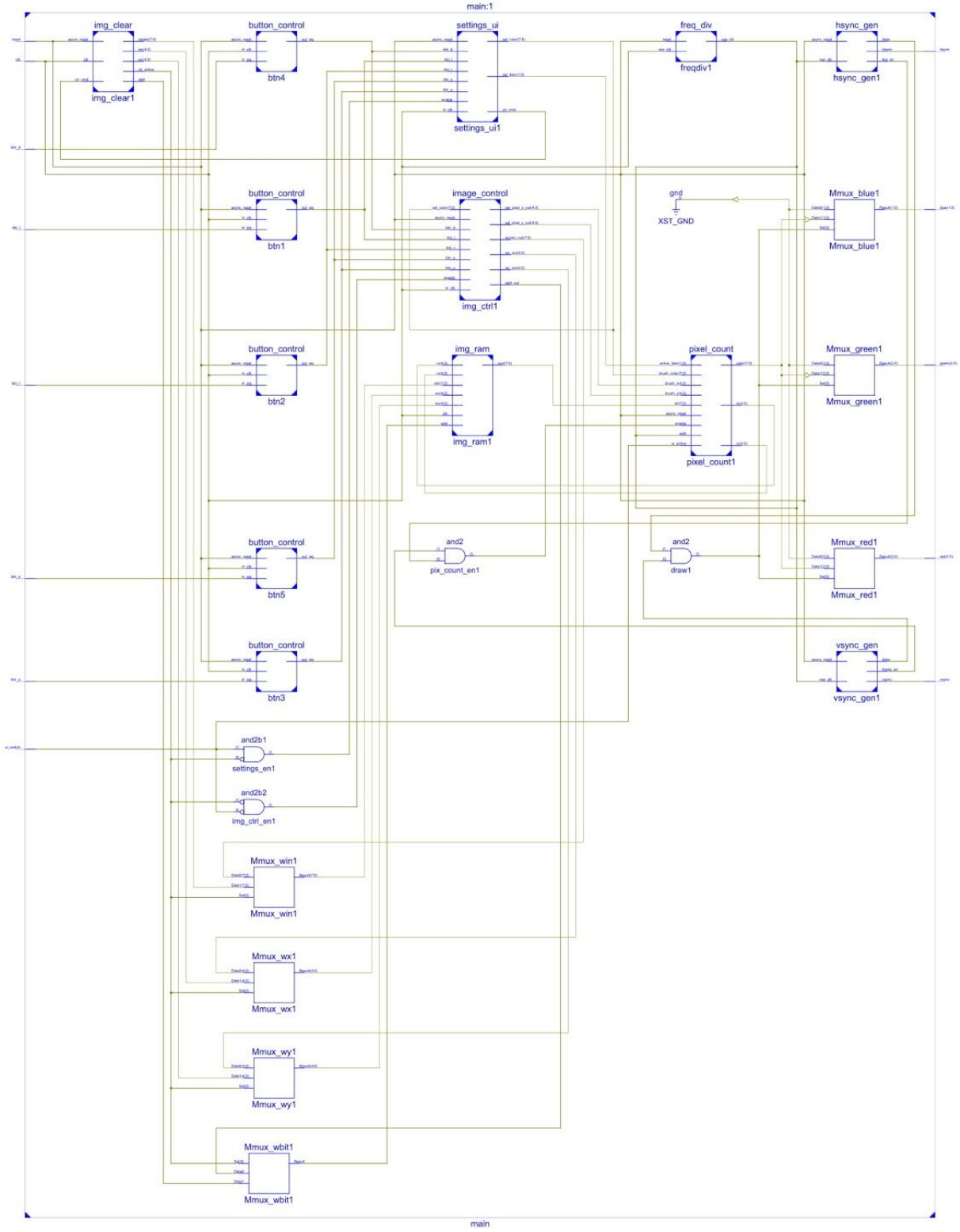
6 CONCLUSION

During the design and building process some minor changes had to be done in the project goals. The final design could be reliably and easily used as a pixel editor. Further improvements could be in the form of add in some other image editing features as well as the ability to save the image drawn.

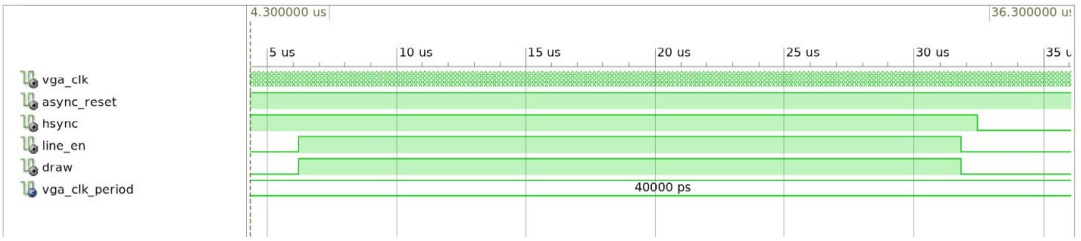
7 REFERENCES

1. XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices, March 20, 2013, https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/xst_v6s6.pdf
2. Nexys 3 FPGA Board Reference Manual, April 11, 2016, https://reference.digilentinc.com/_media/nexys:nexys3:nexys3_rm.pdf

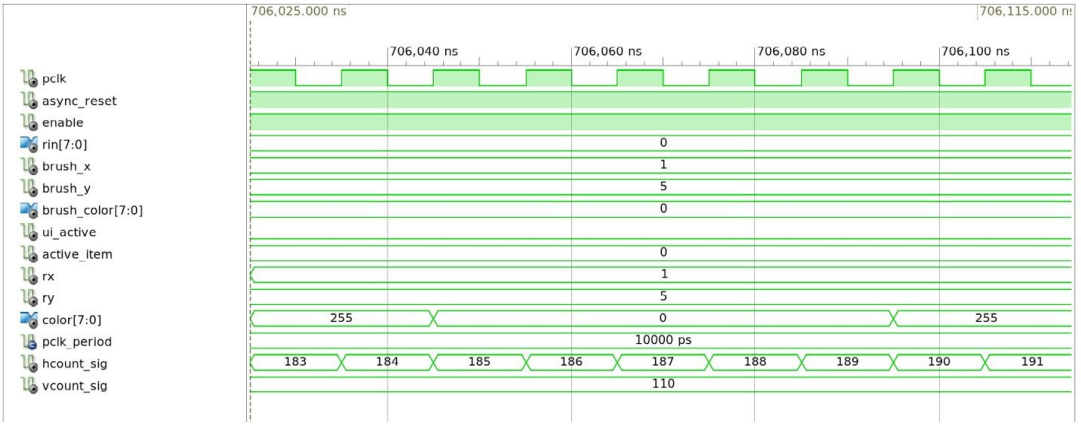
Appendix
A. The schematic of the circuit generated by Xilinx ISE



B. Horizontal synchronization simulation



C. *pixel_count* simulation, *hcount_sig* and *vcount_sig* are the horizontal and vertical positions of the pixel drawn



D. Main circuit simulation, drawing of a single line