VGA Controller

Leif Andersen, Daniel Blakemore, Jon Parker
University of Utah

December 19, 2012

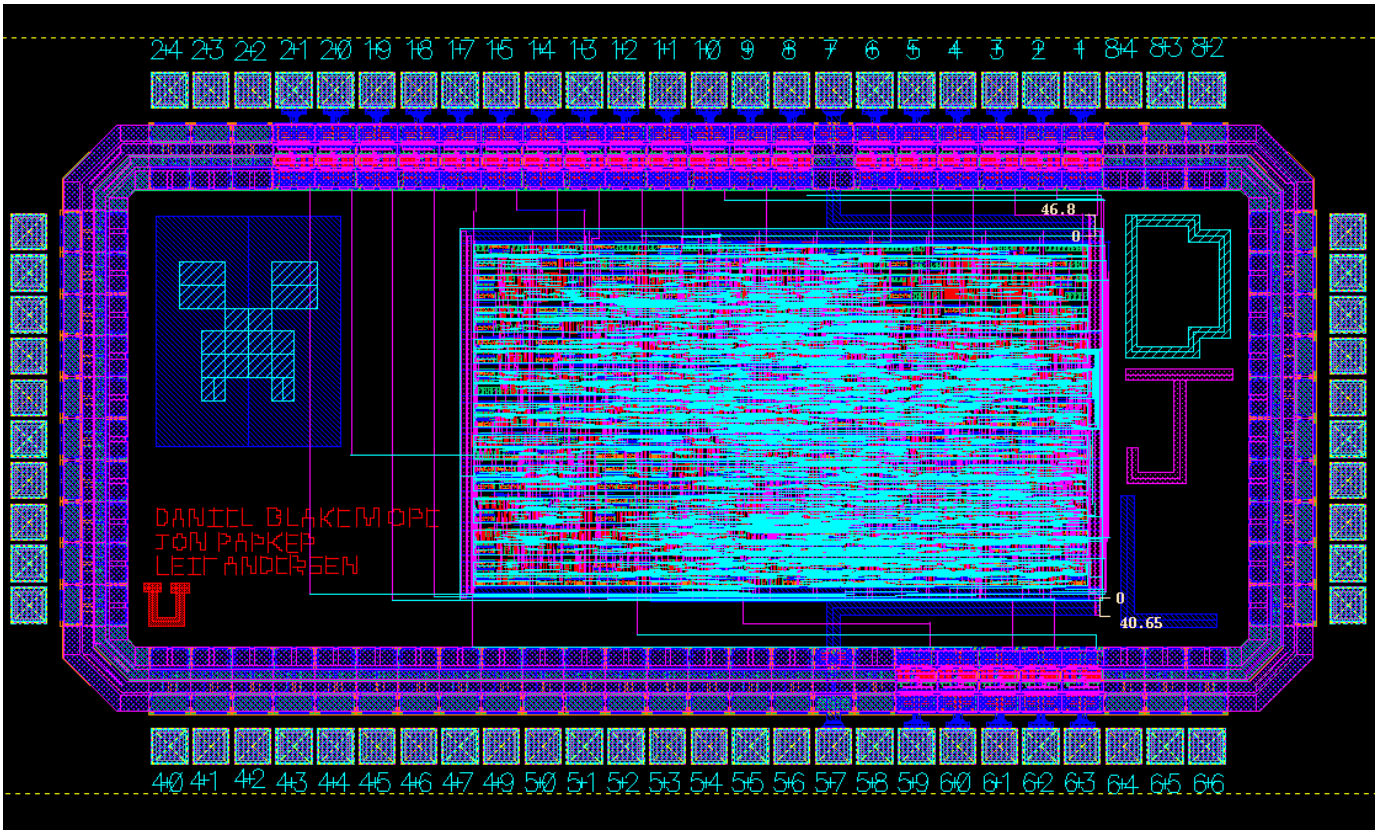

Fig. 1. VGA Controller Components

VGA Controller

Leif Andersen, Daniel Blakemore, Jon Parker
University of Utah

December 19, 2012

***Abstract*—This is a VLSI implementation of a VGA Controller. It uses a custom cell library developed in Cadence for a half micron process. The VGA controller is designed in verilog. It is a coprocessor for an external CPU.**

I. Introduction

The goal of this project is to design a VGA coprocessor. Many embedded systems require visual outputs (phones, cameras, etc), yet many processors do not support these outputs. This project augments a normal processor to add VGA control.

While the VGA controller outputs a full VGA signal, the data only contains at most 160x120 pixels, and eight different colors (the fabricated design is a 4x3 proof of concept due to process and area limitations). The colors that the controller outputs can be set by the designer but are hard coded into the fabricated chip. Colors available on the fabricated chip are:

- Black
- Red
- Green
- Blue
- Yellow
- Purple
- Light Blue
- White

Picture data is sent to the controller as a set of horizontal lines which the processor uses to progressively construct an image from top to bottom. Painter's algorithm is used to draw the data. As such, any data that was previously stored at the position of the new line is overwritten. Data is sent in two sixteen bit word instructions. The first word is split into two eight-bit sections: byte one contains the starting y position of the line, and byte two contains the color. The second word stores the left index of the line in byte one, and the right index in byte two. Depending on the resolution of the chip, some upper bits may be unused. The fabricated design only uses 2 bits for line, left, and right and only 3 bits for color.

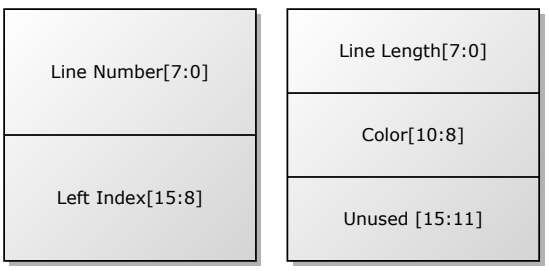

Fig. 2. Data Instructions

Data being sent to the buffer cannot be queued up by the controller, so this queuing must be handled by the chip sending the draw commands. The controller uses a standard double buffering system to prevent artifacts from being drawn to the screen while the picture data is being read in by the processor. Once the processor is satisfied with the resulting picture, a signal is sent to the controller to swap the front and back buffer. The new front buffer is then sent to the VGA output, and any new writes from the processor are directed to the new back buffer.

Given the dimensions of the chip, if registers are used to store all of the data, only a resolution of 4x3 is possible. As such, SRAM chips were developed for this project[1]. However, for testing the design of the chip independent of the ram, a register version was created with the 4x3 resolution.

II. Prior Work

A. Memory

Different memory architectures were investigated related to designing the storage of data for the design. Along with the well-known 6T SRAM cell which was created in our library (but ultimately not used), a special 1T SRAM^{TM} was investigated in a paper from MoSys Inc.[2]. This architecture was interesting in that it used DRAM-style capacitor based memory in a logic-friendly process. See the related paper review for more information.

B. Coprocessors

This design is a coprocessor for a microcontroller. Different coprocessor designs were analyzed while doing research for the project. One of which is the Garp coprocessor[3]. The garp coprocessor was a unique method of combining a CPU and an FPGA.

III. Fabrication

This design will be fabricated. The prototype is the version based on registers rather than SRAM, hence the 4x3 resolution.

IV. Design

The VGA coprocessor takes input from the actual processor and outputs the data as a VGA signal to the display device. The controller assumes that the clock the CPU gives is 25 MHz. The clock of the VGA signal depends on this, and if the clock is a different rate, then the output will not be correct. The $command_in$ interface is where the actual instructions are sent

to the command register to be processed, and $command_we$ is what tells the coprocessor which part of the command it is receiving. Commands are 16 bits. The output is nothing more than VGA output, which includes a VGA signal for RGB (3 pins), as well as an hsync and a vsync pin.

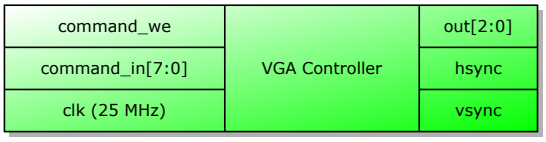

Fig. 3. VGA Controller Interface

Three color pins are output: red, green, and blue. VGA RGB data is an analogue data format. If there is no signal on the red line then there is no red, and if red is high there is bright red. However, unlike normal digital data, the voltage that the signal is at in between low and high determines how much color there is. As such, it becomes extremely difficult for a system using only digital logic to output a VGA signal without a DAC, which this chip does not have. However, given a long enough wire, a PWM signal will average to a desired voltage based on duty cycle and can be used to simulate different colors. This allows for multiple colors that can be composed of not just max and no red/green/blue. As the instruction only has three bits of color data, there is still a limited amount of colors, however this feature allows the chip to be fabricated with any eight colors that the designer needs the controller to produce.

The hsync and vsync signals are also part of the VGA specifications. The hsync signal pulses when a row has finished, and the vsync pulses whenever the frame finishes. This allows a monitor to properly display the image without any tearing.

This controller can be created with two different types of memory: SRAM memory, and register memory. As register memory is significantly larger than SRAM, the resolution is adversely affected. The SRAM version is capable of producing a resolution of 160x120 pixels (predicted based on area), while the register based one is only capable of a resolution of 20x15 (predicted, 4x3 implemented). Unlike color data, swapping out the register based version and the sram based version is significantly more difficult. Currently, only the register based version is functional.

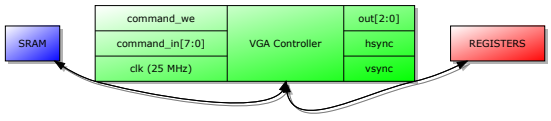

Fig. 4. SRAM

The register was laid out by hand, and was composed entirely as a single cell, rather than as a collection of smaller cells.

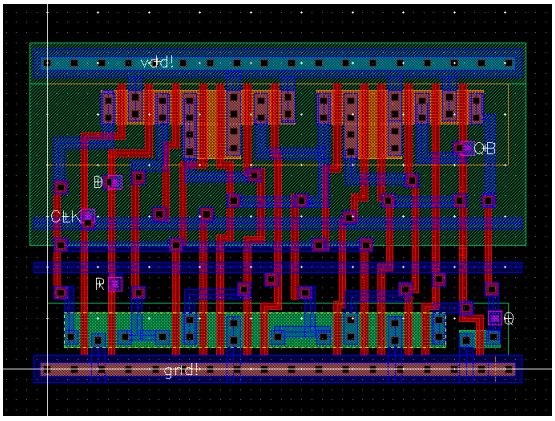

Fig. 5. Register

The SRAM cell design was based off of the design by Lyons[1]. SRAM memory banks are laid out in a grid as wide as the word, and as long as the addressable space, or in sub-banks which are muxed together to form the full memory space.

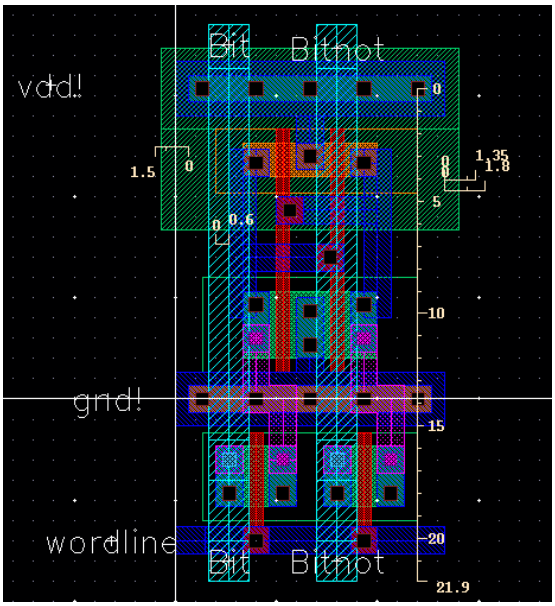

Fig. 6. SRAM

When data is sent to the controller, it is loaded into a command register to process it. From there, the command register will store the output of the line into one of two buffers (whichever one is currently designated at the back buffer). The VGA signal sent out will come from the other buffer (the front buffer). When the processor receives 0xffff, this is a special command that tells the controller to switch the buffers; that is, the buffer being written to switches to being the display buffer (front), and the previously displayed buffer stops being displayed and can now be written to. This method allows for pictures to only be sent out that are completed. While the data is stored at a lower resolution, the actual resolution that is sent to the output is a full VGA signal (640x480 at 60Hz), with all of the pixels just happening to be blocks (in the case of the fabricated chip, blocks of 160x160). Figure 7 depicts the

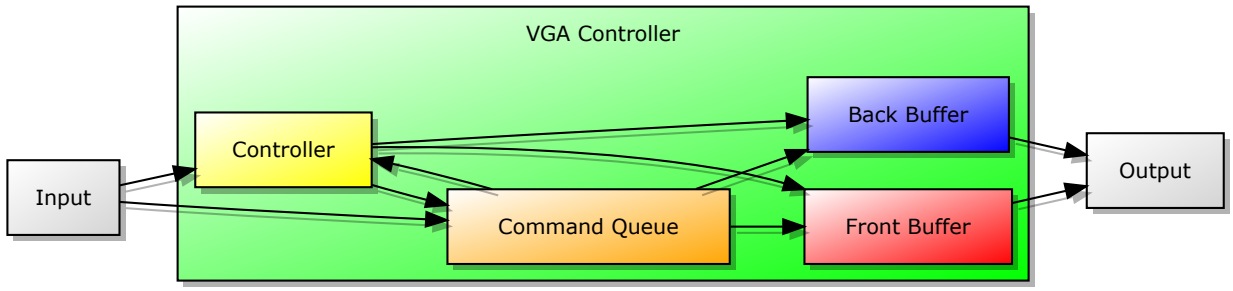

Fig. 7. VGA Controller Components

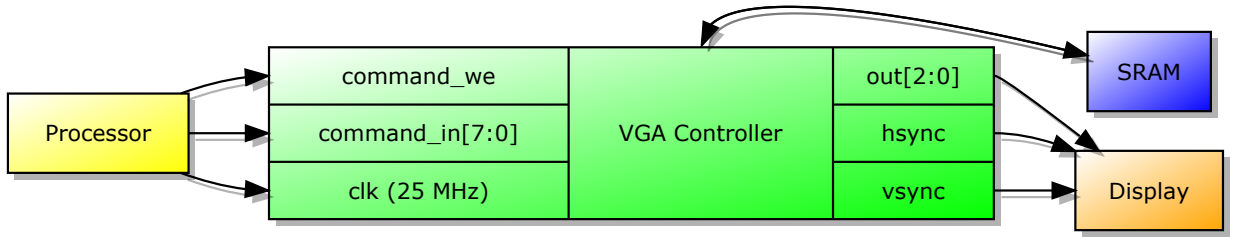

Fig. 8. Top level diagram

actions that are transpiring.

Figure 8 depicts the way the controller is designed to interact with the rest of the system. The processor and controller are designed to be on different dies (but remain on the same PCB). The processor supplies the controller with a clock at the correct rate (25 MHz), as well as the enable and instructions.

Unlike the processor, the SRAM is designed to be placed onto the same die as the controller. This is to provide fast enough access to the control queue and drawing buffers. Still, a future version of the chip could potentially use dynamic RAM to store a larger picture, and that could use dynamic RAM on a different chip.

Finally, the controller outputs to a display using the VGA protocol. If any colors other than the default colors are used, because of the digital PWM to analog hack mentioned earlier, the cable that connects the display to the controller must be long enough, otherwise the display will simply get a PWM signal and may not interpret it properly. Empirical tests show that a standard VGA cable has sufficient length to cause most displays to work properly. It has not been tested on a system where the display is connected more directly to the controller (i.e. possibly a phone like device). If the default eight colors are used, there is no concern of wire length.

V. Simulated Results

There were three stages of simulation: pre-synthesis, post-synthesis, and post-layout. All three stages of the simulation output identical results, meaning there were no inconsistencies between the verilog and the actual chip. The below images of synthesis show a demonstration of painting the bottom row of pixels green, except for the first one. Then the buffer is switched so that row of green is displayed. In all the simulations, vsync takes approximately 16.6ms between pulses (60 Hz), and 480 hsyncs happen during the drawing time of the vsync. Furthermore, these simulations demonstrate that the swap buffers command does indeed work.

VI. Testing Results

The VGA controller functioned under all testing parameters. Figure 9 demonstrates the results of the VGA controller running. Additionally, Figure 10 is a photo of the coprocessor inside of the testing harness.

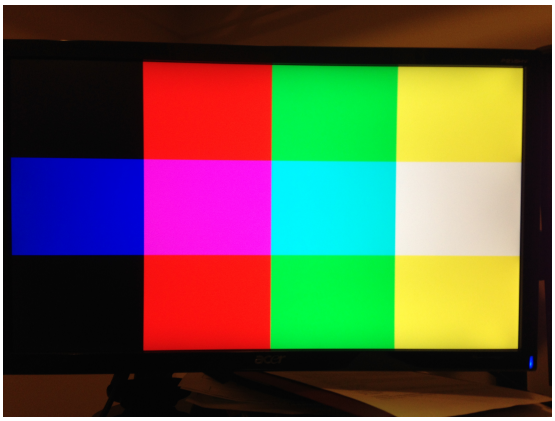

Fig. 10. The VGA coprocessor rendering squares.

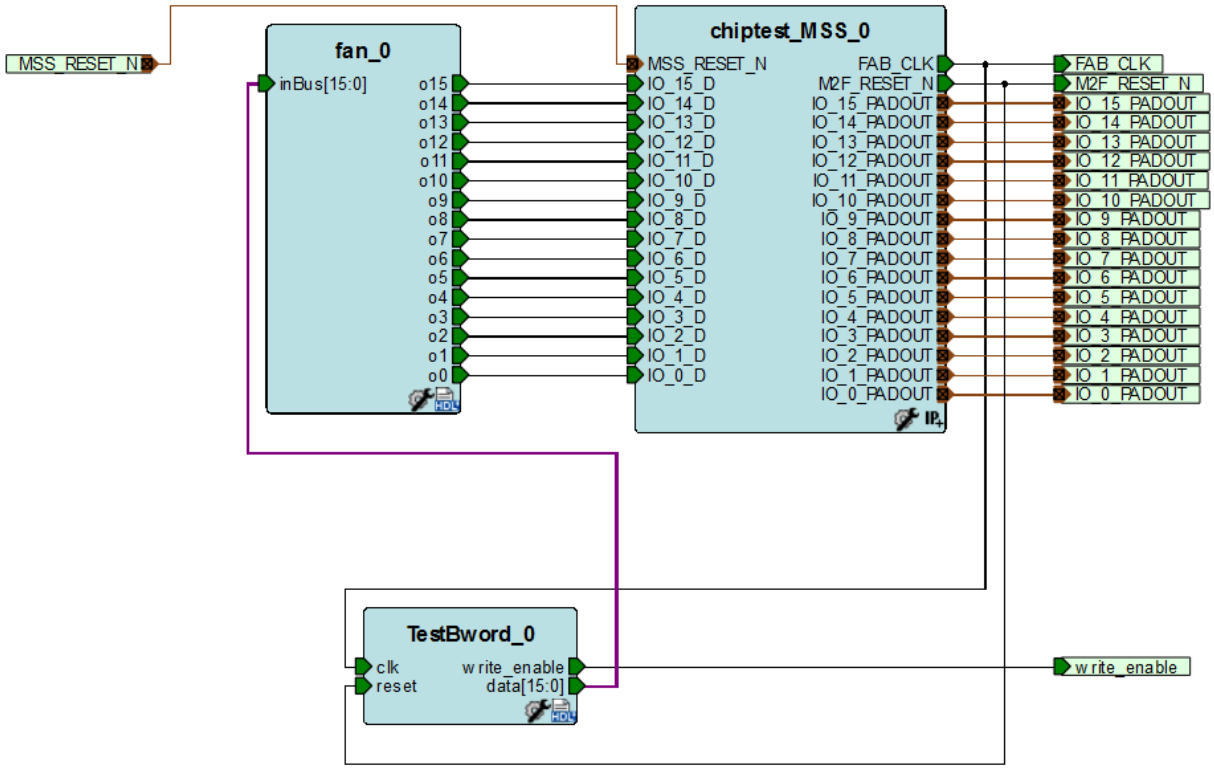

Fig. 9. Testing Diagram

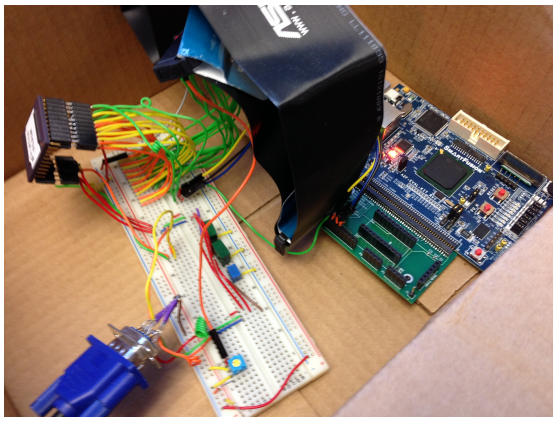

Fig. 11. The VGA coprocessor.

As this is a co-processor, the chip does nothing on its own. As such, we created a testing framework using Actel Libero SoC and an Actel Smartfusion eval kit. Figure 11 depicts the layout of the testing framework.

Five chips were fabricated from MOSIS. Every chip worked as expected (depicted in Figure 9), using the testing harness described above.

A range of clock frequencies were tested on the chips. There was no minimum frequency for operating the chip. The maximum frequency for operating the chip was 73 MHz. At 74 MHz, the latching for the chip fails.

Additionally, we tested to find the minimal operational voltage of the chip, and found it to be 2.49 +/- 0.01 V.

A schmoo was not performed for this chip, instead performance numbers were calculated by manually sweeping two independent variables. Therefore, no msa files were created. There were no defects found in the chip.

VII. Conclusions and Lessons Learned

In a word, integrated circuits are complicated. Both the final result and the methods by which a designer arrives at that result are complex and detailed. The design process involves a large number of steps and tools which include the risk of error at each step, but are at the same time extremely powerful and perform much of the design work that would take an untrained human years to complete by hand. Our final design was laid out in an area of around 1.5 TCU (with 70% place-and-route density) and will be fabricated upon a 2 TCU horizontal pad frame. Upon fabrication, future testing will be conducted to verify the operation of the VGA controller.

The project files can be found on the CADE lab at: /home /landerse/ic_cad/cs6710

References

[1] R. F. Lyon and R. R. Schediwy, "Cmos static memory with a new four-transistor memory cell," *Schlumberger Palo Alto Research*.

[2] W. Leung, F.-C. Hsu, and M.-E. Jones, "The ideal soc memory: 1t-sramtm," in *ASIC/SOC Conference, 2000. Proceedings. 13th Annual IEEE International*, pp. 32 –36, 2000.

[3] J. Hauser and J. Wawrzynek, "Garp: A mips processor with a reconfigurable coprocessor," in *FPGAs for Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on*, pp. 12–21, IEEE, 1997.

APPENDIX A
PRE SYNTHESIS

Figures 12 and 13 refer to the waveforms of the pre synthesis tool.

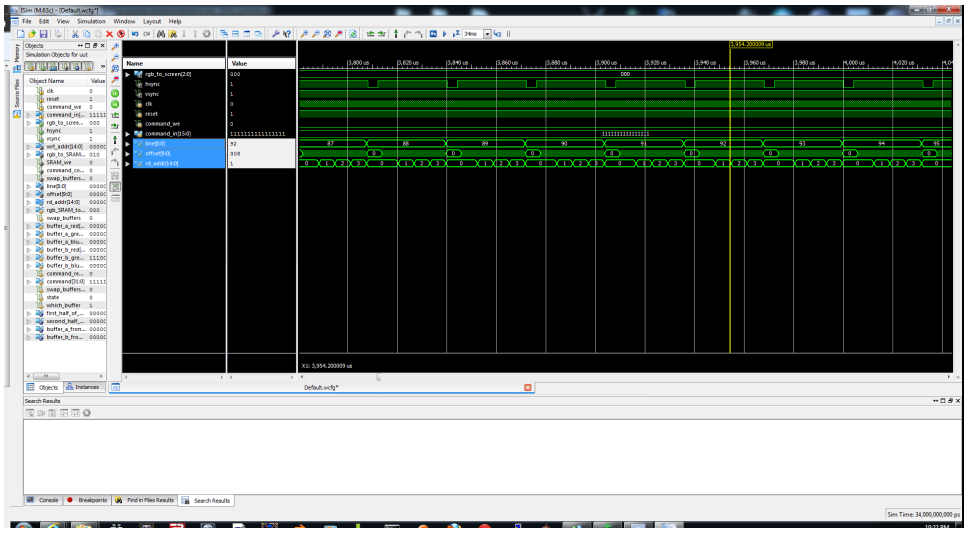

Fig. 12. Pre Synthesis

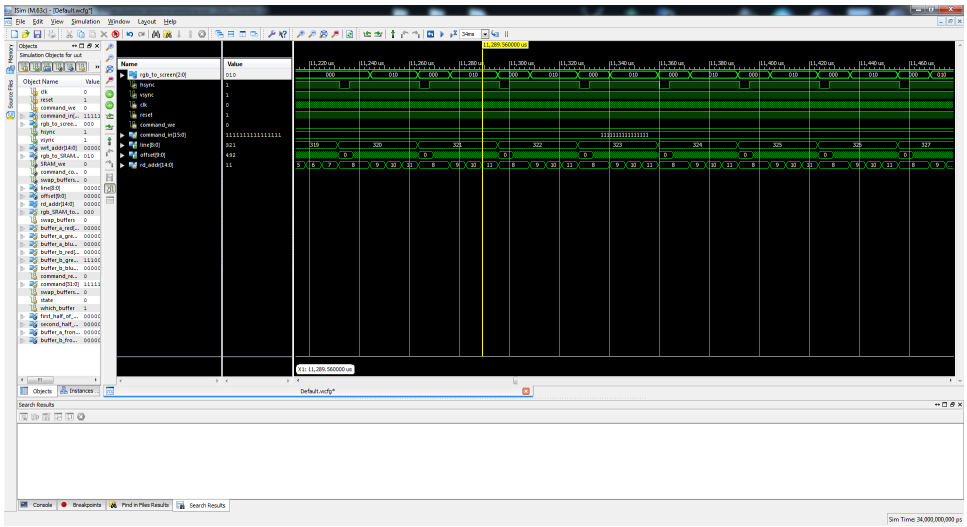

Fig. 13. Pre Synthesis Zoomed

APPENDIX B
POST SYNTHESIS

Figures 14 and 15 refer to the waveforms of the post synthesis tool.

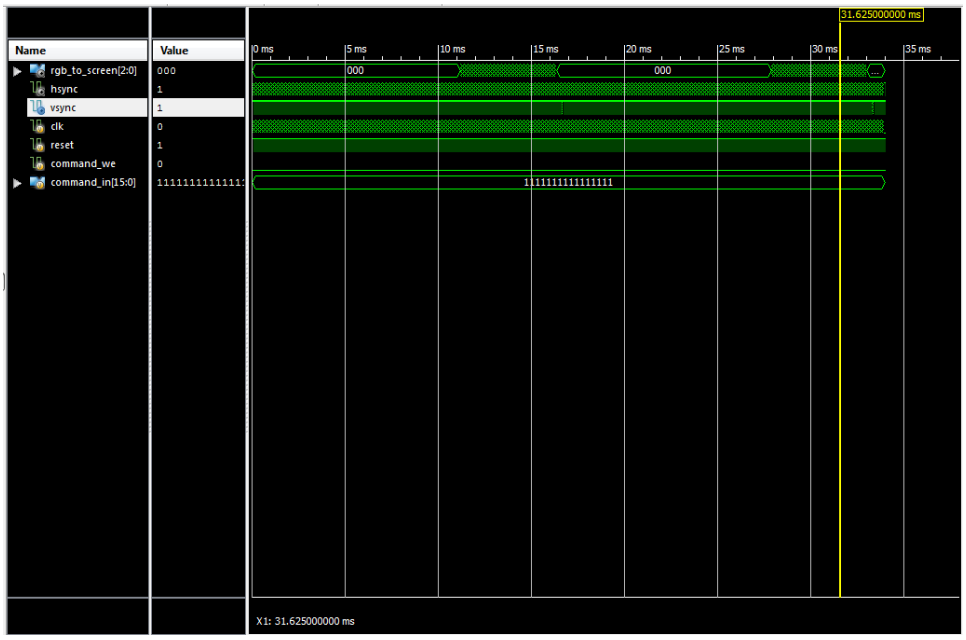

Fig. 14. Post Synthesis

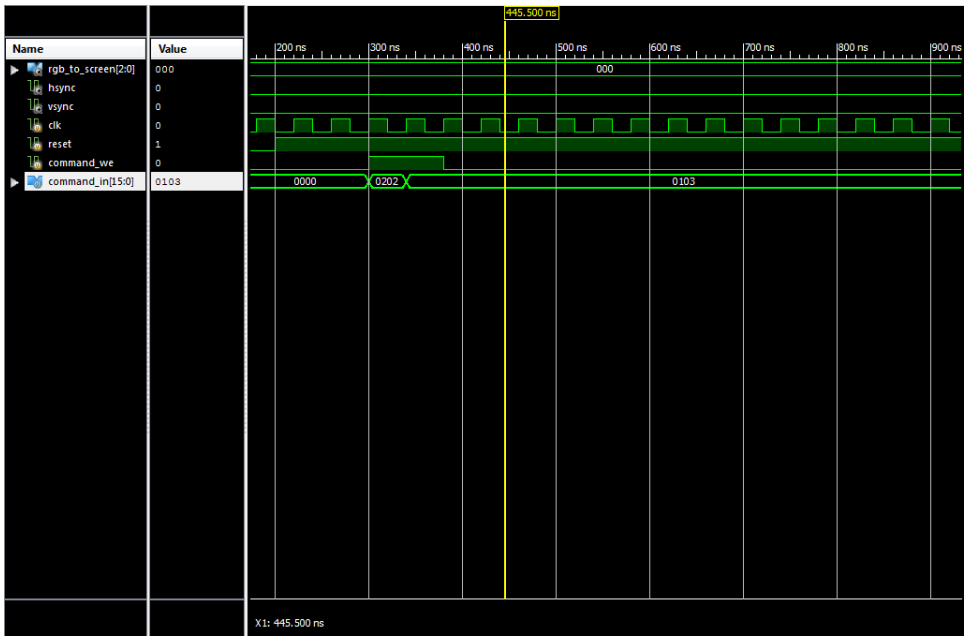

Fig. 15. Post Synthesis Zoomed

APPENDIX C
POST LAYOUT

Figures 16 and 17 refer to the waveforms of the post layout tool.

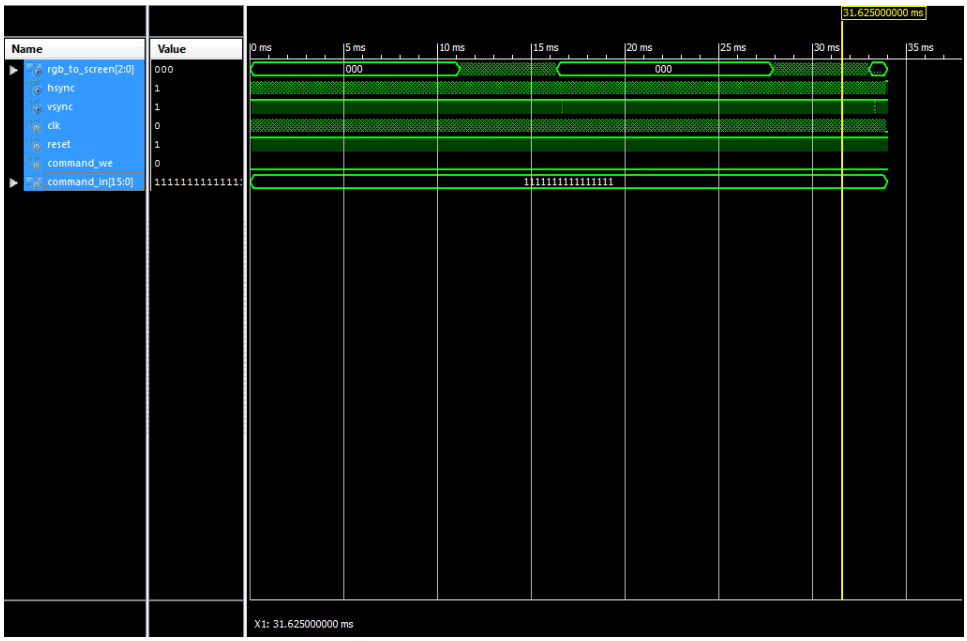

Fig. 16. Post Layout

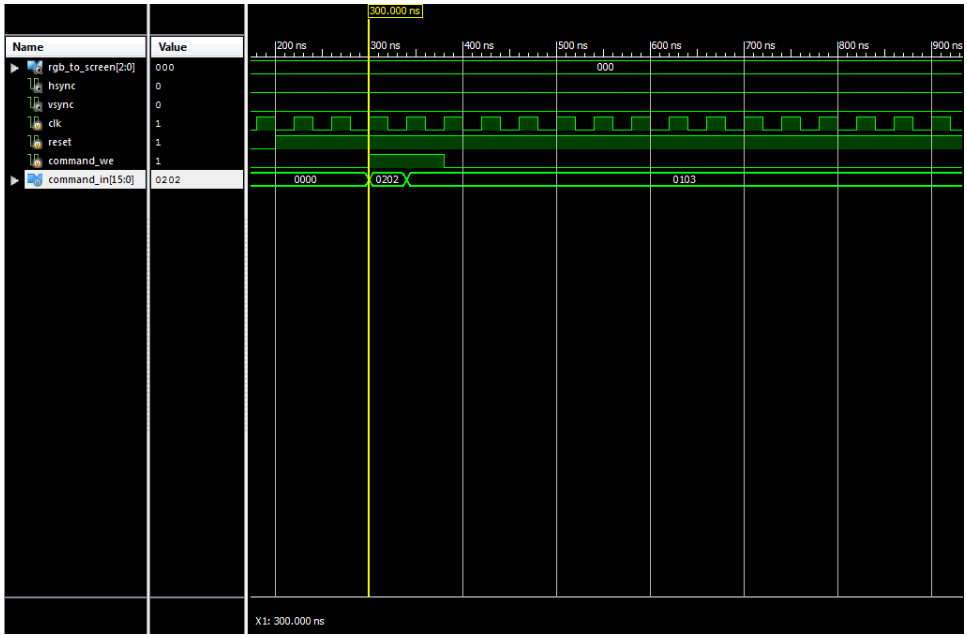

Fig. 17. Post Layout Zoomed