A Text Editor

List of Authors (McKenzie Walsh, Aaron Boening, Andrew Glenn, Andrea Taylor)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: mtwalsh@oakland.edu, ajboenin@oakland.edu, awglenn@oakland.edu, aataylo3@oakland.edu

***Abstract*— The purpose of *A Text Editor* was to explore different applications of Field Programmable Gate Arrays (FPGA). This project includes various applications covered during ECE 378 lecture such as UART and PS2 serial communication and VGA display. This project accepted a text input from a PS2 keyboard and then would output on the VGA. In addition to displaying the imputed text to the VGA display UART was used to change the color and background color of the display. Changing the color of the text and background color was accomplished by a separate computer using a terminal program such as Putty. Overall this project was very successful and all desired goals were accomplished.**

I. INTRODUCTION

The goal of our project was to emulate a typing experience of a simplified word processor using VHDL coding and an FPGA. To use *A Text Editor* one will input text using a standard PS2 keyboard and change the text color and background color via UART. The motivation for this project was to emulate a device that is used in everyday life. As developers of this device it was important to understand the functionality of the PS2 keyboard, UART functionality, and VGA functionality. Although this project required knowledge of many topics covered in class, some topics required additional research. Learning and incorporating all of these functions allowed us to create *A Text Editor*.

II. METHODOLOGY

A. Choosing "A Text Editor"

Initially our group felt pressure to design a video game as this is a common final project for the ECE 378 course. After some research we concluded that a more practical application would be a better suit. Thus we chose to research serial communication and UART. After reviewing UART we considered some applications that were covered in class; it was then determined that a text editor with varying background color and text color would be appropriate as it also included the VGA monitor and a PS2 keyboard which is also a form of serial communication .

B. PS2 Keyboard

The PS2 port is the most common and widely known interface to communicate with a host through synchronous serial communication. Inside the PS2/USB cable there are four wires: PS2 clock, PS2 data, ground and vcc. A keyboard, like the one we used, has a microprocessor that scans the keys for activity. When a key is pressed down it will transmit a "make code". Then once the key is released it will also transmit a "break code". The make and break codes are generally 1 byte wide but can be up to 4 bytes wide depending on which key you are pressing. We used all 1 byte wide make and break codes for our project.

C. UART Receiver

A Universal Asynchronous Receiver and Transmitter is a circuit that uses a serial line to send parallel data. The receiver portion was only used in this project. The UART receiver shifts in data bit by bit and then reassembles the data, this is similar to a shift register. This circuit contains one finite state machine. A modulo counter and a shift register. The data communication begins with a start bit, '0', followed by the data bits. The stop bit or idle bit occurs when the serial line is at '1'.

For the constant that indicates the number of data bits,

Figure 1: Data Transmission From Chu's Text

D_BIT, and SB_TICK is the constant number that indicates the number of ticks needed for the stop bit. We used 8 data bits and 16 for the ticks for stop bits. We implemented the UART receiver in our project to change the color of text and the background at the same time. Therefore we assigned several keys on a different keyboard than the PS2 to belong to a text color and background color. A table of color combinations for text and background color can be seen below in table 1. A schematic of the entire UART receiving circuit can be seen below in figure 2.

Q	Black Background / Gold Text
W	Black Background / Green Text
E	Red Background / Green Text
R	Blue Background / Red Text
T	Yellow Background / Black Text
Y	Magenta Background / White Text
U	White Background / Black Text
Default	Black Background / Green Text

Table 1: Color Codes

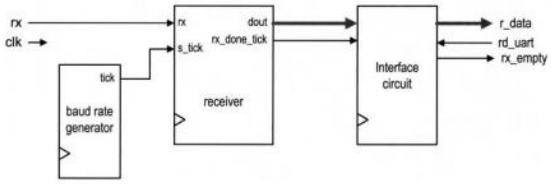

Figure 2: Schematic of UART Receiver System from Chu's Text

D. VGA

VGA (video graphics array) is a video display that is widely accepted by PC graphics hardware and monitors. The VGA port has five signals, three video signals for red, green, and blue, along with the horizontal and vertical synchronization signals. The h-sync and v-sync are linked with the vertical and horizontal scans in the monitor. This scan correlates to the X and Y pixels that give the position of the pixel. It is assumed that the pixel rate is 25MHz . To reduce the memory requirement and complexity of the Bit-Mapped scheme display, we choose to use the Tile-Mapped scheme. In this type of scheme it allows us to group a collection of bits into a tile. Each tile is treated as its own unit so if you had 640-by-480 pixel and the unit was defined to an 8 by 8 square, then the screen becomes 80-by-60 tile-oriented screen. This is the configuration used in our project to display the text on the monitor.

E. General System Overview

As seen in Figure 3, the system is split into two sub-blocks. One handles the UART communication and the other handles the keyboard PS2 communication. The UART portion is intended to translate UART inputs into a background and text color. The PS2 portion actually displays something to a VGA screen. The system requires a clock input, a reset input, the UART input, two PS2 inputs (one for clock and one for data) as well as an PS2 initialization input. It only outputs three bits for RGB, h-sync, and v-sync. These signals are discussed briefly in the *VGA* section.

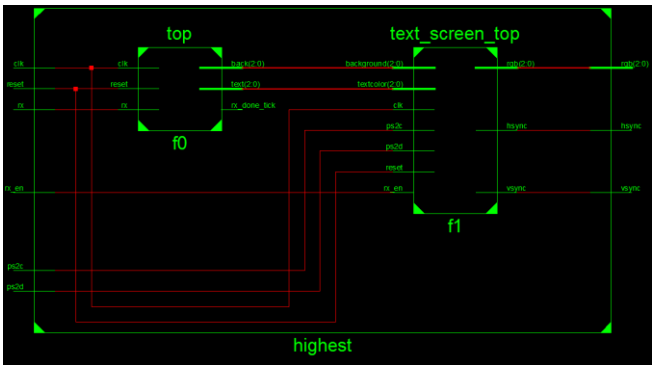

Figure 3: Top-level schematic showing the two sub-blocks of the system.

The block labeled *top* in Figure 3 represents the UART portions of the design. It is divided into three blocks. A modulo-m counter, a UART receiver, and a color decoder (discussed in a later section). The modulo-m counter is used in conjunction with the UART receiver. It makes sure the receiver is communicating at the correct baud rate using the clock speed of the board. This portion of the system is shown in a schematic in Figure 4.

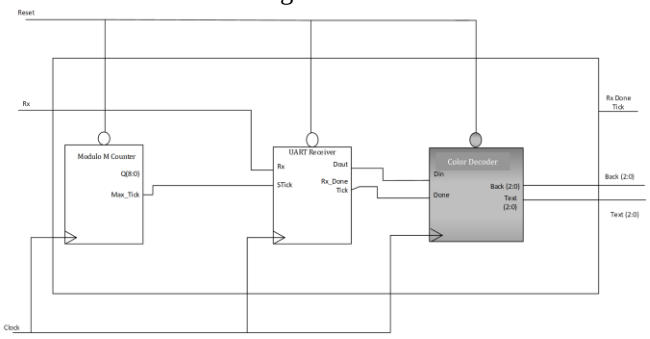

Figure 4: Schematic for the UART portion of the system

The PS2 piece of the system is shown in Figure 5. It is labeled *text_screen_top* in Figure 3. This block gives the RGB, h-sync, and v-sync outputs to the VGA output. The general outline of the block is as follows. The keyboard sends PS2 inputs to the system that are received by the PS2 receiver. The data interpreted by the receiver is used by the Text Generation FSM (discussed later) to decide when to submit a letter to the screen and move the cursor. The -*key2ascii* block takes the key code data and converts it to an ASCII. The Text Generation Unit (discussed in the next section) actually moves the cursor and submits letters to the screen and outputs it to VGA.

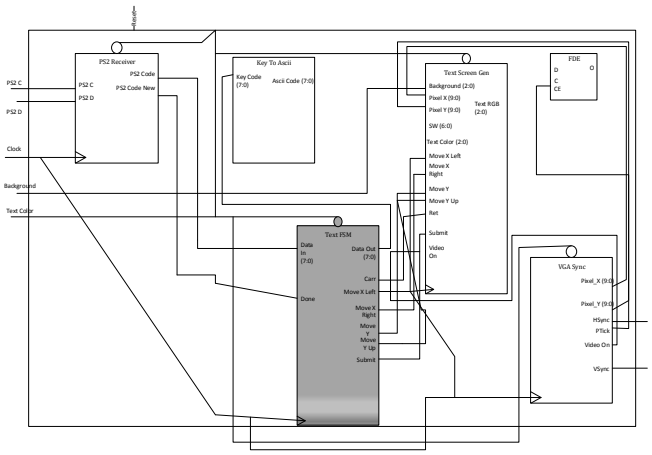

Figure 5: Schematic for the PS2 portion of the system

F. *Text Generation Block*

This block was taken from Chu's text but also modified to achieve the desired functionality. The text generation block manages the location of the cursor on the screen and the submission of text symbols. So when this block receives an ASCII code and the signal to submit, the symbol that correlates to the ASCII code is sent to the screen at wherever the cursor is located. This block also controls the color of the background and the text (it gets the colors from the color decoder mentioned below). This block was modified so that the cursor functions like other text editors. When the cursor gets to the end of the line, it moves to the start of the next line. Before, the block only allowed for the cursor to move right and down. It was modified to also move up and left so that the arrow keys could work correctly. This was implemented for easier navigation. It was also modified to easily handle the carriage return, which also moves the cursor to the start of the next line. Another slight modification was made to the memory module. The reset button did not actually clear the screen because the memory module (taken from Chu's example) kept the text stored. A reset condition was added which set all the memory array values to zero. This worked, but it extended the time to synthesize and implement the design within the ISE software significantly. The reason is unknown, but perhaps the method used to clear the array was inefficient and needed to be optimized. This is something to look into.

G. *Color decoder*

The color decoder is a simple block which takes the data from the UART receiver and translates it into a color to display to the screen. There are eight color combinations (background color and font color) which are attributed to the arbitrary keys. More color combinations could be added easily. Because the system only outputs three bits to the VGA (one for red, one for green, one for blue), these color combinations are limited. Implementing another standard, like 12 bit RGB, would have allowed for more color customization. Another improvement that could have been made in regard to text and background color is the option to choose each color separately.

H. *Text Generation FSM*

This is the control block that interprets the outputs of the PS2 receiver. The receiver outputs a done pulse each time a byte is sent over serially. So, for example, when the 'A' key is pressed, the PS2 receiver will output *1C, done, F0, done, 1C, done*. The make code (when the key is pressed) contains the first byte. The break code (when the key is released) contains the next two bytes. In this example, this finite state machine would wait for the first done pulse, which an edge detector shortens so it does not last through multiple clock ticks, and then switch to the next state. It would next wait for the *F0* code to be sent over and switch states. Lastly it would take the third code, *1C*, and tell the text generation block to submit the letter to the screen. Note, the text generation block gets the ASCII code to place on the screen from a block that converts the byte long key code to an ASCII code (taken from Chu's text). After another clock tick, it would tell the text generation block to

move the cursor to the right (so the user can continue typing without having to move the cursor). This process would then start over. The process changes slightly when an arrow key is pressed because the make code is two bytes instead of one and the break code is three codes instead of two. The FSM just waits for more done pulses to pulse a submit bit to the text generation block. The finite state machine could have been altered to add more functionality to the overall system. The FSM only allows a symbol to be displayed on the screen on key release, so holding down a key has no effect. To better emulate a normal typing experience this would have to be changed. Key combinations could also be accounted for in the FSM. This would, however, require changing other aspects of the system in order for the combinations to act normally. For example, a key combination is required to use an exclamation point. The state machine could interpret the combination but the exclamation point symbol would have to be added to the font tile library (which we got from Chu's text to save time) and the *key2ascii* block would have to translate the exclamation point code to ASCII. These changes could be added in time but were excluded for simplicity.

III. Experimental Setup

Since several portions of the system consisted of modified pieces of code taken from other sources or relied heavily several timing inputs, for a lot of the testing, it was quicker to simply compile the code, synthesize it, and confirm the functionality it gave. In other parts (mostly the control blocks), simulations were used to verify the circuit. These were useful in instances when we knew the desired functionality and it would not take too long to simulate the timings (like in the case of the PS2 receiver). In other cases, it was decided that the better experimental method for testing the functionality was implementing the block on the Artix7 board. For example, in order to test the UART receiver (a modified receiver taken from Chu's text), the receiver was implemented so that the output of the receiver was

constrained to the LEDs on the board. This way, it was easy to tell what was coming out of the receiver and decide whether it was functioning properly. Using this method, we were able to tell when the receiver was set up for the wrong clock time and adjust accordingly. Simulations were used for the text generation FSM, and color decoder because these were most important to the functionality of the system and creating the test benches was not too time consuming. These simulations were crucial to debugging the system and probably should have been used more extensively to save time. Figure 6 shows the simulation results for the text generation state machine. It runs through the press and release of the 'A' key. The simulation shows that given the make and break codes along with done ticks, the state machine runs through it states so that it can send a submit pulse and a move right pulse, which are then used by the text generation unit to actually move the cursor and submit the letter. This functionality is expected because while typing, the user is accustomed to pressing the letter and the cursor moving to the next position.

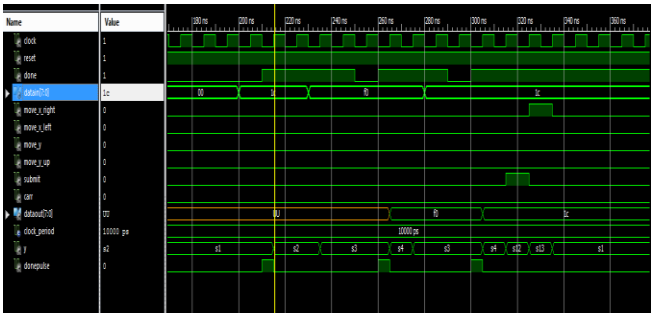

Figure 6: Behavioral simulation for the text generation state machine, showing the press and release of the 'A' key

IV. Results

Ultimately *A Text* Editor performed as expected. We w were able to display text and some symbols on a VGA screen in addition to changing the color of both the background of the VGA display and the color of the text via a PC with Putty UART. In addition to being able to display letters on the VGA screen other keys on the keyboard like the space bar, backspace key, enter key and arrow keys were also fully functional. In order to expand functionality of this program we could include more symbols and combination keys (shift+number) in our code. The following link is a video of our project functioning https://www.youtube.com/watch?v=M1KimC8q-IQ.

This project allowed us to explore different applications of VHDL and how these applications could be combined. It covered several topics covered in class, such as VGA output, state machines, flip-flops, edge detection, etc. These topics were discussed earlier in the report. This project also provided a deeper learning opportunity for specific serial communications. For example, learning about and dealing with the make and break codes given by the keyboard was at first overlooked.

V. Conclusion

From working on this project we took away a better understanding of how VHDL works and how powerful it can be. Also, by using the VGA hookup, we learned how that whole setup works. This setup was covered in class but there was additional information that we found out about the VGA, such as tile mapping. Before the project none of us knew about tile mapping and how to apply it to a VGA. By researching we were able to get a grasp on it and apply it to our code in order to view our test on the screen. We also learned about the PS2 keyboard as we progressed through the project. The original text generation finite state machine overlooked the make and break codes given by the keyboard. So instead of functioning as intended, each letter and move cursor command was executed twice (one for key press and one for key release). The test bench did not catch this issue because the make and break codes were not simulated. Another issue dealt with the clearing the screen with the reset button. At first the reset button would not actually clear the letters off the screen because they were still stored in the memory module. Adding a condition that cleared the array in the memory module worked but it extended the synthesize time significantly. Some improvements that could be made are that we could have added a 12 bit color scheme instead of the 3 bit that we used. Using the 3 bit limits us to only a few different colors to choose from. With the 12-bit scheme we could make many more color combinations. Another idea we could have added could have been to do more research on the PS2 keyboard so all of the keys on the keyboard have a function. Only the numbers, letters and a few other keys worked. Also, we could have found out how to change the keyboard from a press and let go input, to be able to hold down the key and it keeps working, such as holding backspace down. We cannot continuously hold down a key and it works. We have to press and release over and over again for the key to work.

VI. References

[1] Chu, Pong P. *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*. Hoboken, NJ: Wiley-Interscience, 2008. Print.

[2] PS/2 Keyboard Interface (VHDL) - Logic - Eewiki." *PS/2 Keyboard Interface (VHDL) - Logic - Eewiki*. N.p., n.d. Web. 14 Apr. 2015 https://eewiki.net/pages/viewpage.action?pageId=28278929