Yu Chang Ou and Kendra Crawford

ECE 318 Digital Design

Professor Traver

Final Project Report- Typing Tutor

Abstract

This project was to design a typing game, which allows the user to learn how to type quickly on the keyboard by implementing the educational aspect as the objective of the game. We designed a system, which takes inputs from the keyboard control and output the game’s results on the screen. Intermediate outputs include the use of the LCD display on the FPGA which will keep track of the “time” it took the user to initiate and complete the entire list of words in the game. The game begins with an opening screen, which prompts the user to begin the game. When the game begins, a word is displayed in the middle of the screen to the left. The user must type the word that is being displayed. When they type the letter correctly, the the indicator LED will flicker green. When the letter is typed incorrectly, the indicator LED will not flicker until the correct letter is hit, and the user is only then allowed to continue.

There will be one system asynchronous reset button, which will be the KEY 3 push-down button on the DEII board. The system-reset will tell the system to return to the beginning screen and return the timer back to 0. All other inputs of this system will solely be triggered by the commands on the keyboard. When the correct key is pressed on the keyboard the game will flicker the indicator LED. If the incorrect key is pressed then the game will turn the LED will not flicker and the game will wait till the correct key is pressed.

System Overview

|  |
| --- |
|  |
| Figure 1. Top Level Design of Entire System |

The system was designed to get information from a keyboard then that data is transferred over to the VGAcomp. The keyboard required the inputs of the DE2 board keyboard clock, and the data line this allowed the keyboard to run properly. It then had the indicator LED to allow the user to know if there was anything wrong with the make code and break code being the same. The q output is the data that is sent to the VGAcomp that is compared to the character displayed on the screen. The VGAcomp uses KEY[3] as a system reset, and the clock to synchronize the data. The VGAcomp displays aw word from set of pre-listed words that are compared to the keyboard inputted data. Once the letters, then words are inputted correctly the next word is displayed from the list. The outputs of the VGAcomp are the outputs on the DE2 board that connect the board with the VGA display. It also has an output that is connected to a green LED that allows the system to indicate that the letter and the keyboard inputted data matched. The timing component of this system is connected to KEY[3] which is the reset, KEY[1], SW[3] and the clock. The KEY[1] is the pushbutton of the DE2 board this is the start and stop of the system. This meant when the person started the game they would push that button the time would start then when they ended the game they would push it again and the time would stop. The SW[3] was a reset enable this meant that when that switch was high and the reset was pressed not only would the current time become zero but the best time would also zero out. If the enable was low it meant that only the current time would be zero when the reset was pressed. The output of the timing block where the outputs on the DE2 board that connected the system to the LCD screen which is what we used to display the time. The clock keeps the timing component synchronized with the rest of the system.

Subsystems

VGA Components

|  |
| --- |
| VGA top level design.JPG |
| Figure 2. Top Level Design of VGA Display |

The VGA display is composed of six different parts as seen in Figure 2. The first is the VGA\_sync, a sequential block provided by the DE2core library. This block outputs information based on the pixel row and column it is currently adjusting, and takes the input of what color the specified area should be. The data that is inputted into the system by the red, green, and blue inputs is processed by this block and sent to the output on the De2 board to the VGA display so the VGA display “lights up” the color specified by these inputs.

The next block was based off the DE2core library block, large\_char, which defined the amount of space the characters would take up on the screen. The block can be found if Figure 2 connected to WordStorage and VGA\_SYNC blocks.Char\_display defines the amount of space the maximum ten characters WordStorage outputs will take up on the screen. We wanted to make it in the middle of the screen and smaller than large\_char so the user could understand the characters easily. This a combination block since it depends on the output pixel\_row and pixel\_column not a clock.Pixel\_row and pixel\_colum define the space on the screen that is currently changing based on the red green and blue inputs. Figure 3 shows some the VHDL that defines how the screen is portioned to show the characters in a specified space. The VHDL also shows that when there is no char\_one, two, three etc the VGA should display a space. Which is described by the BCD hex value 40.

|  |
| --- |
| VGA Display Char_display.JPG |
| Figure 3. VHDL Showing the Portioning of VGA Display |

The third block is word storage this is a sequential block that outputs a set of characters to the comparator block, and to the display block. The word changes when the word is typed correctly. WordStorage can store as many words as the designer wants, that fit within the ten character limit. Figure 2 shows where the block is placed in the top level design of the VGA Display. Figure 4 shows how a set of characters were defined in VHDL and how the changing of states changed the word. It also has the letter that is represented by the HEX value to see the full chart of letters to BCD HEX values look at the appendix Figure 1.

|  |
| --- |
| VGA Display WordStorgae.JPG |
| Figure 4. VHDL Showing Process to Change a Word and How Words are Being Stored in WordStorage Block |

The next block is the StoredWordRom which is the DE2core character\_rom. This block is sequential to the VGA clock since it sends the signal that states that the screen should be a color based off the character display.

The fifth block for the VGA display was the the color\_changer block, this was a sequential block that depended on the VGA clock. This block used the output of StoredWordRom which indicated that there was going to be something written there if it was high. In our design this meant the color should be black since those are the letters besides that the rest should have been white. Figure 2 shows the block and how the outputs are connected to the VGA\_SYNC block. This is also the block that causes the indicator LED to go high when the character and keyboard input match. Figure 5 shows the VHDL that used the inputs to choose the correct time for the red, green, and blue outputs to go high.

|  |
| --- |
| VGA Display Color_changer.JPG |
| Figure 5. VHDL Showing the Change in Red, Green and Blue output in Color\_Changer |

The final block is a comparator block, which compares the input from the keyboard and with the current character from the stored word block to check for a match. The comparator block is combinational logic block. If the letters indeed match, the LED will flicker green prompting the stored word rom to send it the next character to display. If the key pressed on the keyboard was not correct then the letter will turn red and waits till the correct key is pressed. Figure 2 shows how the block is connected to others in the VGA display top level and how the data is sent. Figure 6 shows the VHDL code that was responsible for comparing the keyboard data to the current character and what output were sent when. If the character was “000000” it meant that it was the last letter of the word and to go to the end state.

|  |
| --- |
| VGA Display comparing.JPG |
| Figure 7. VHDL code for comparing Block in VGAcomp |

Keyboard Components

|  |
| --- |
| Keyboard top level design.JPG |
| Figure 7. Top Level Design of Keyboard |

The keyboard is composed also composed of four blocks. The first block is the keyboard DE2core block. The keyboard block is a sequential block that has a scan ready, read, keyboard data, and clock input. There is also a scan code output. This is a block that takes the data from when the key is pressed and outputs the address that is connected to that key.

The second block is the keyboard control block communicates with the keyboard and captures the key values sent from the keyboard. This is a combinational logic block so it does not rely on a clock. The third block is a store and error check block these store the “make” and “break” scan codes into 8 bit registers, and then registers are checked for errors by the compare.

The final block is the convert and display block. This is a sequential logic block and has a clock to send out the correct data. This block is responsible for converting the break codes into VGA display code. This is done with a 256x8 ROM that uses the break code as an address and stores the BCD hex value that corresponds to that letter for the VGA display. See the appendix Figure 2 for the list of the keys on the keyboard and the correspond make and break code. Figure 8 show the rom data and if compared to figure 1 and figure 2 in the appendix they verify that those are the correct values.

|  |
| --- |
| Keyboard ROM.JPG |
| Figure 8. ROM used for Ipm\_rom0 in Keyboardcomp |

Time Keeper Components

|  |
| --- |
|  |
| Figure 8. Top Level Design of Time Keeper |

The time keeper component is based on the incremental timing unit used in lab 3 where we link cascading *lpm\_rom* modded (by the megawizard function) to receive *clk\_div* (clock step down module) signals. Each cascading lpm\_rom also has a *enablewait* which facilitates the transition of the cascading ripple signal through the counters. The count is then displayed onto an LCD screen using the *LCD\_display* component. After the LCD display receives signal from the cascading time blocks, another signal is sent to *timestore* which controls the storage of the “best time” recorded in the game in which the player finish typing the series of words.

Lpm\_rom is a component that acts like a counter in which when triggered by the clock will count to its modded number. The speed of its count is determined by the clock speed to which it is linked to. The lpm\_rom is connected to the *clk\_div* found in lab 3 which decreases the frequency of the clock. Here, we connected out lpm\_rom to a 1Hz clock signal which sets the lpm\_rom to count once per a second acting like an actual second counter. We also modded the LPM\_rom to have both a mod 10 and mod 7 configurations because this will allow us to have counters that count from 0-9 and 0-6 to implement the cascading timer with a 60 minute : 60 second count.

Because the lpm\_rom blocks uses a ripple counter mechanism to count, the transitions between one number to the next is not synchronized to the clock. There will be instances where the number will count to 5 in one signal rather than 1 due to the prolonged nature of the signal of Cout in the LPM\_rom. Therefore, the enablewait module implements a clocked input that is on the same frequency as with the rest of the components on the time keeper which allows for accuracy and regulation in the transition of the cascading timer

We then sent the time outputted by the LPM\_Rom to the LCD display so it would display the count time of the module since the start of the game (triggered by the press of key 1). A similar signal is also sent to timestore at the end of the game when key 3 is triggered to compare the current end game time with the previous lowest end game time. The timestore component is a 1-to-1 comparison of the newest numbers given by the lpm\_rom, and the previous numbers stored as signals given by the lowest time thus far. When the newer time is lower than the past recorded time, timestore will update the “best time” with the new fastest time.

Results

The general goals of the project description were met or was replaced by another ability to substitute for any component that did not work as intended. We had three main top-level blocks in this design: the keyboard, VGA display, and LCD time keeper.

The keyboard retains full alphanumerical functionality and successfully took in inputs that were recognized by the FPGA. It could then send information of each character to be displayed by the VGA screen.

The screen of the VGA display was correctly partitioned such that the words would be displayed in the middle of the screen. The VGA correctly displays the word when prompted by the keyboard and was able to transition to a new word when the letters were typed in correctly. The integration of the keyboard and the VGA screen was successful and functions the way we expected.

However, our goal for the VGA screen was not met completely. Originally, we wanted to have each letter turn from black to either green or red depending on whether the character was typed in correctly or not. This offers the user direct feedback on their typing and would have given a better indication of the status of the word, but this would have required an output color signal on each of the letters we partitioned on the screen. Because we only came up with this new strategy to improve the design later in the process, time limitations prevented us from getting a chance to implement the change of color of each letter.

The intended behavior of this design was to have the letter change to green when typed in correctly and then wait for the next letter of the word. When the letter is typed in correctly, the letter on the screen will turn red and remain so until the correct letter is typed in. Because we never worked out the color signals, this project offered feedback to whether the correct letter was typed in on the keyboard by using the LEDG7. The LED would flicker green when correct letter is typed in, and will remain steady green when the entire word is completely typed in correctly. When the wrong key was typed in, nothing would happen and the program will continue waiting until the correct key that matches the letter on the screen is triggered.

Due to differences in clock signal, transitions from one word to another in the completion of the word is triggered by the space key. But with faster typists, the clk signal would lag at the end of the word and the new word would transition before the space key is pressed.

The LCD screen timer was the last block built for this system. On its own without integration to the main game, the timer worked as intended similar to lab the behavior derived from lab 3. The player would be timed and both their current time and best typing time thus far would be displayed on the LCD screen. The inputs to this would be to use key 3 as a reset and key 1 as the start and stop button. The timer is only enabled when switch 3 is on.

In the connection of this component, we intended the changeword signal on the VGA controller to signal the start and stop command on the time keeper, but the handshaking between the timing of the main Keyboard-VGA component to the time keeper did not agree and instead, the VGA displayed a dim flickering pattern. We attempted other methods to integrating the LCD display but due to encountering numerous errors such as the VGA display failing to work or the timer interfering with the transitions of the game, we decide to have the original external inputs at key 1 to start and stop the clock and the system’s asynchronous reset to reset the LCD timer.

Conclusion

In this project, we did meet many of our project goals. For any project goals we did not meet, we were able to find substitutes to perform similar tasks. Given more time, we would improve the feedback mechanisms for the player such as individual character color control and recording of best time to offer better indications that the letter was typed in accurately.

Our system consisted of three large components of which had more smaller components in them. This layered design made it hard to diagnose what was truly the cause of the problem. We tried to compartmentalize errors by designing each component on its own, and then integrating the systems at a later stage. Using this method, we successful integrated the screen to the keyboard to function as we intended. Although we did have a problem when putting the LCD time keeper with the already integrated system, this definitely would have been improved if given more time to debug.

One of the more frustrating things we encountered was the lag time in loading of the Quartus because our integrated system used so many parts. In total, we used 608 logic elements and 325 registers, but a brief overview of the design allows us to see that logic for components such as the timekeeper module’s counter could have been simplified by using sequential counting instead of cascading lpm\_rom. Had we simplified the design logic further, we could have generated a faster design as well.

This project did allow us to understand the integration of VHDL files and how to direct the control of the keyboard to change the output on the VGA screen. Though the complexity of the project made it difficult in integrate the different components together, we gained insightful knowledge of the uses for VHDL and the importance of simplified logic on the timing of a system.

Appendix

|  |
| --- |
| VGA Display Table .png |
| Figure 1. Table of Letters and Symbols with BCD Hex values to be sent to VGA Display |

|  |
| --- |
| Keyboard Keys and Scan Code.png |
| Keyboard list of make code.png |
| Figure 2. Figure of Keys and Table of Corresponding Make and Break Codes |