Maxwell Stonham
Dylan Cazares
CPE200L

The Mental Math Machine

Project Overview

The Mental Math Machine is a game that incorporates the use of the DE2-115 board and its switches, pushbuttons, seven segment display and 16x2 LCD screen. The math questions will be limited to addition and multiplication questions with numbers ranging from 0-50/. Players can play up to 3 rounds per "game" and for each round, the player needs to answer 9 different problems (10 seconds each) with a difficulty increase within each round. Round 1 will be easy, round 2 will be medium and round 3 will be hard. The seven segment displays will be used to tally right and wrong answers, implement the timer, as well as display the user input. The questions will be shown on the LCD screen of the board alongside any display messages. If the user answers a question correctly, their score will be shown as a tally onto the seven segment displays. For each 9 problems per round, after the user answers (or misses), a message under the question (second line of the LCD display) will display showing whether or not the user answered correctly or incorrectly. A performance tracker will also be implemented to increase the difficulty of the questions (i.e., how big the numbers are) depending on how well the player does for each round. After each round ends, the tally will reset to 0-0 and the difficulty will increase depending how the user did the previous round. If the user gets 6/9 questions right, then they are prompted a question to proceed to the next round, otherwise they are prompted to restart the game from round 1.

Block Diagram

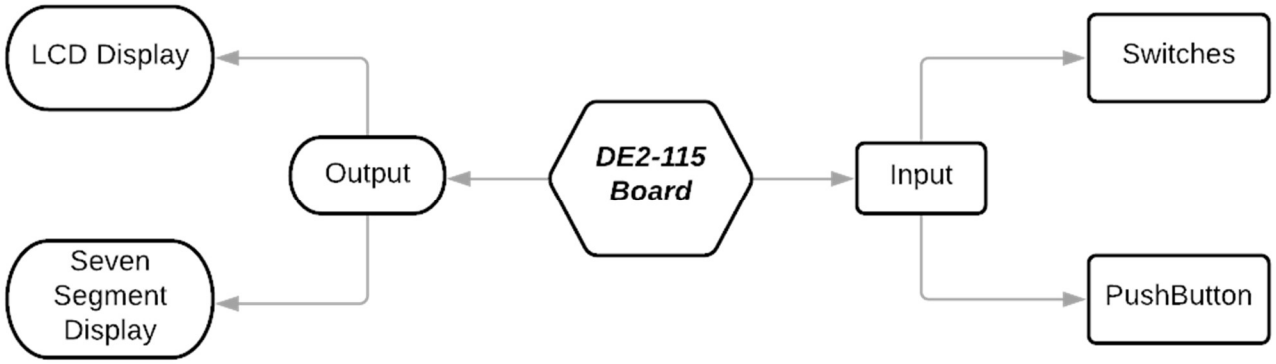


Figure 1: Overall System

The basic overall system is quite simple with the user inputs being the switches and pushbuttons while the outputs are the LCD and seven segment displays, as seen in Figure 1.

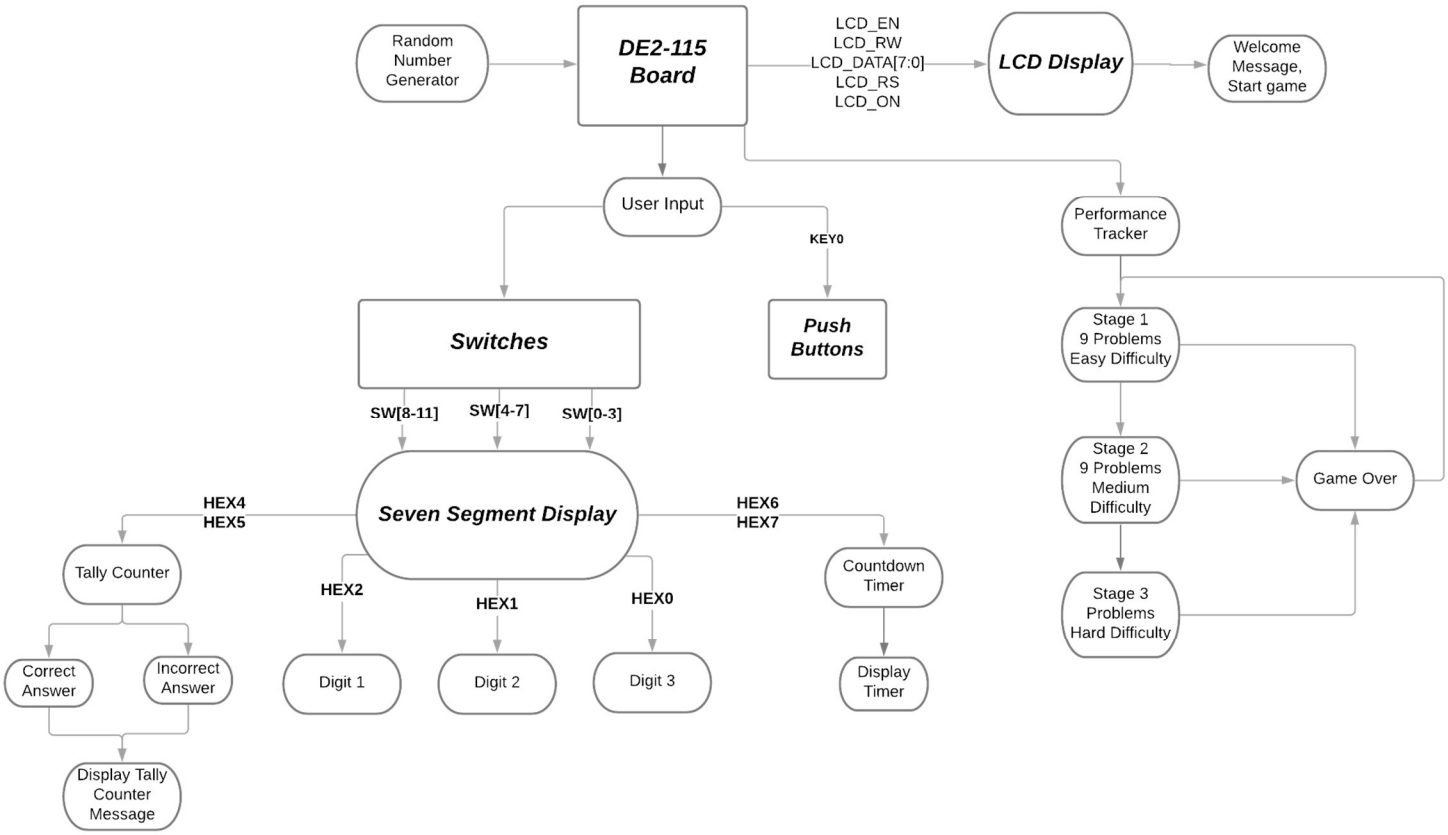


Figure 2: Submodules

The game will implement the use of a counter, timer, performance tracker (using if/else statements) and random number generator, as seen in Figure 2. To start the game, the player is prompted with a message displaying whether they are ready to begin. The player presses the right-most pushbutton (KEY0) to begin, and as soon as they begin, the first question is displayed along with a 10 second timer counting down on the seven segment display. The timer will be implemented onto HEX5 and HEX4 while the tally will be implemented onto HEX7 (incorrect answers) and HEX6 (correct answers). If the player answers the question right, the tally for correct answers will increment using a counter, and if a player gets the answer wrong or does not answer within 5 seconds, then the tally for incorrect answers will increment.

The player will use switches SW[0-11] to input their answers and use KEY0 as an "enter" key to submit their answer. SW[0-3] will be used to answer the right-most digit, SW[4-7] to answer the middle digit and SW[8-11] to answer the left-most digit with a maximum of three digits being used (not necessary to use all of them, depending on the question). SW[0-3] will correlate to HEX0 of the seven segment display, SW[4-7] to HEX1 and SW[8-11] to HEX2 in order to display what the user inputs before they submit their answer. What may seem somewhat interesting to players would be that the switches correlate to binary numbers split to three portions for each three digits, hence why those switches are chosen. For each digit a user enters, the switches they flip will correlate to binary inputs from 0-9. An example: to enter the number "56" onto the right most display, the user must flip SW6, SW4, SW2 and SW1 before submitting their answer since "5"

and “6” is “0101” and “0110” in binary. This can potentially make the game more challenging to users since it tests their binary conversions as well.

A random number generator will be used to generate the random questions. The performance tracker is implemented by using an if/else statement to determine whether or not the user gets to go to the next round depending on what the users score within a round. If a user gets 6/9 questions correct, then the user can proceed to the next round and the difficulty will increase. Otherwise, the user will need to restart the game from round 1. Difficulty will increase each round by means of asking users to calculate a larger range of numbers.

Adjustments Made and Lessons Learned

The feedback given from the mid-project report were to show design files in appendix, how the random numbers are generated and to implement the performance tracker and timer in our simulation before implementing VGA and keyboard interface. The random number through simulation is generated by the $urandom_range function call which calls upon random numbers within the range that it is given to. Implementing the timer would not be ideal to do in simulation, but instead would be better demonstrated in hardware. The implementation of the performance tracker is a bit difficult because there were many errors when simulating through ModelSim. However, a rather large component of our project is adjusted, that being the implementation of the monitor and keyboard which is the reason we are resorting to our original idea of simply using the board alone without the use of any external devices.

The reason being for this adjustment was mainly due to how inefficient it was to display text onto the VGA monitor, let alone randomly generated text. We realized later on that the easiest way to display text onto VGA was by having the board read individual txt files before being able to display it onto the monitor. While displaying text alone isn’t too difficult, the issue we were having was displaying randomly generated numbers since we would then have to download txt files for each individual number onto the board, on top of the letters of the alphabet. This proved to be more difficult than we thought since we weren’t quite sure how to implement it onto VGA properly. Another issue we were having was trying to use the PS/2 ports to implement the keyboard. The issue we faced while trying to implement the keyboard was registering the user inputs. We were able to find the scancodes of the keyboard however we got stuck on trying to make the module for it. The keys we were going to use were the numbers 0-9 on the number pad for the user input and the enter key on the number pad to confirm their answer. These adjustments were made due to difficulties in the implementation, so we decided to resort back to the LCD display and switches. This adjustment isn’t so bad since it would feel more like an arcade game by using the switches on the board to input answers displayed on the small LCD screen.

While doing the timer something we learned is that it began to be more difficult to implement it onto the board. The main code we understood where the top module is just a decrementing counter, the next module was just the slowclock that was provided. What we learned from the slow clock is that in order for the board to understand how a second works it utilizes its frequency which is 50Mhz which is also one second per pulse, for 1 millisecond it is just 50Mhz times 0.1, with that you’re able to find the bit width of the value which in this case the bit width

was 25 bits. However, the difficult part where we struggled on was how to implement the output of the counter onto a seven segment display. We were able to figure out the
seconds in the ten's spot but had a difficult time trying to add the timer on the one's spot since for a seven segment display it requires 4 inputs (Data[0-3]) but we only had one output to go into 4 inputs.

Another lesson we learned is that simulating the testbench proved to be much easier than it is to implement onto hardware. We realized that we spent too much time working on the testbench to test it out in simulation, but when it came to implementing it onto hardware, it was much more difficult. We forgot that the testbench file can't be uploaded onto Quartus directly, which was an issue because we spent a lot of time working on the testbench and simulating it through ModelSim that we forgot it was totally different than doing it in Quartus. After many compilation errors, we also realized that we couldn't get random numbers through Quartus, even though it compiled through ModelSim. This proved to be an issue considering our main game is based on having randomly generated numbers output the question.

Implementing the LCD also was a first for us, and it was much more difficult to do than we initially thought. We had trouble figuring out which signals to use for the LCD based on the user manual. We tried using the external software that was included when Quartus was installed (Nios II and Eclipse), but we weren't able to program the LCD display because we kept getting errors when compiling the C++ code. This leads to another thing we learned, which was that the LCD display can be programmable through C or C++, but when we tried doing this, it wouldn't compile our code for "Hello World" and kept giving us errors.

Future Steps

Possible future steps for this project if it functions properly could be implementing a keyboard and a VGA display, which is what our modified mid-project decision was before resorting to this current implementation. Implementing a keyboard and a VGA display could allow more creative freedom with editing how the game looks. Of course, a big improvement that could be made would be to be able to implement subtraction and division as well, but we thought to do addition and multiplication alone was because it is much simpler for users to do addition and multiplication rather than subtraction and division, but this could be implemented in the future. However, implementing division onto hardware can also be expensive and take up a lot of memory and/or time.

Another possible modification to our game could be by giving the users the option for their time limit. While we have it at 10 seconds by default, a way to improve this could be by giving the users more variety by either increasing or decreasing the time limit. This could prove to add somewhat of a difficulty range as well since users are able to have a more flexible decision on whether or not they think the timing is too short or too long.

While we currently have our user inputs in binary, an good implementation or change could be to make the switch inputs decimal instead. We are currently using 12 switches, 4 switches for each digit with a total of 3 digits as the input. A possible modification could be to use 9 of the switches for numbers 0-9 and make use of the pushbuttons as an enter key for each digit. For

example, when entering an input of “125”, a possible way to implement this modification could be to use the switches to switch up “1” at the start, then press KEY3 to enter this as the first digit, use the switch to enter “2”, then press KEY2 to enter this second digit and switch up “5”, then press KEY1, so on the display is now “125” and users can enter KEY0 to submit their answer to see whether or not they got it right, then the next question is given. This could prove to be a somewhat “easier” alternative as opposed to having to convert the numbers to binary before using the switches.

Conclusion

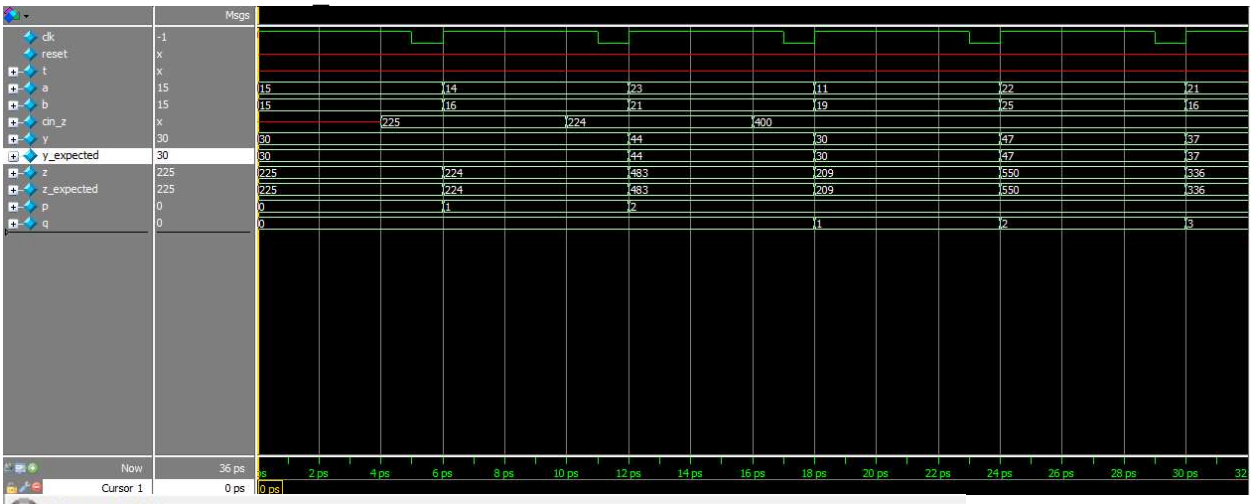
In conclusion, we came across more problems implementing our game than we initially thought we would. It seems that we were a bit too ambitious with choosing this as our first ever VHDL project and implementing it onto hardware was more difficult than we initially thought. However, we learned quite a lot of things along the way such as a basic understanding of the process on how we would implement the game onto VGA, creating our own testbench for the game’s main functionality, the relationship between the signals and the pins on the board, how there are various ways of changing the LCD screen and that random number generators don’t transfer directly from simulation to hardware.

Despite not being able to fully implement the game onto the DE2-115 board, we managed to download the pin assignments for the input onto the board. We also managed to write a testbench for most of the functionalities of the game. The only issue we came across doing this was the performance tracker since our if/else statements seem to compile properly, but not provide the outcome we expected. This project was quite an eye opener as to how difficult it can be to implement a game onto the DE2-115 board using SystemVerilog, but an interesting experience nonetheless.

Appendix

Files:
testbench_mentalmath.sv
sevenseg_mmm_pins.qsf

Waveform for testbench_mentalmath.sv:




```
Transcript
# Loading work.testbench_mentalmath
VSIM 136> run 4
VSIM 137> force cin_z 225
VSIM 138> run 2
# Correct. Nice job!
VSIM 139> run 4
VSIM 140> force cin_z 224
VSIM 141> run 2
# Correct. Nice job!
VSIM 142> run 4
VSIM 143> force cin_z 400
VSIM 144> run 20
# Incorrect. You answered:          400 Correct answer:          483
# Incorrect. You answered:          400 Correct answer:          209
# Incorrect. You answered:          400 Correct answer:          550
# Incorrect. You answered:          400 Correct answer:          336

VSIM 145>
```

The waveform shown is what is produced when the testbench file is run in ModelSim. The waveform is formed by manually running the simulation for different periods of time. The clock generated is high for 5ps and low for 1ps (as seen in the testbench). The random number generator generates a new number with each rising edge of the clock. In the simulation shown, by manually running the waveform for 4ps, we see that the random numbers “a” and “b” are produced, “a” being the first number and “b” being the second number. We use arbitrary terms for addition and multiplication when testing. The “y” variable is the output for addition while the “z” variable is the output for multiplication. “y_expected” and “z_expected” are the expected answers for each “a” and “b” calculations for addition and multiplication, respectively.

In this testbench, the counter counts up on each positive edge. We use arbitrary terms p and q for the counter values, “p” being correct and “q” being incorrect. For testing purposes, we are only counting up for each correct/incorrect multiplication input alone since addition will be implemented the same way. “y” and “z” values are also both shown to demonstrate that the addition and multiplication of random numbers works fine (with the values for the results shown in the waveform).

When running for 4ps, we manually force cin_z (user input) to the correct answer for the addition. In this case, we force cin_z to 225 since a = 15 and b = 15, so 15x15 = 225. After forcing this value and running the simulation after the next rising edge, the counter for correct answers “p” increments by 1, while “q” stays at 0. We do this process two more times to show that the “p” counter increases for every correct value while the “q” counter stays the same. After repeating one more correct answer, we force cin_z to be incorrect the third time and the simulation shows that the “q” counter now increments by 1 while the “p” counter stays at 2. We run this simulation with this same value and the “q” counter continuously increments by 1 with every clock edge since the answer for the addition is incorrect.

This same method will be used to count the addition questions, but for testing purposes we kept it at just multiplication to keep it less complicated to show in the waveform. For each correct and incorrect answer, we also implemented a display to show up. For each correct answer, messages such as “Correct. Nice Job!” will be displayed, and for each incorrect answer, the message “Incorrect. You answered: x. Correct answer: y.” is displayed as simple display messages for each answer. The if/else statement used in our testbench doesn’t seem to work the way we intended it to. The code compiles, but our if/else statement used to implement the performance tracker doesn’t seem to be working at all.