SINdoku

EE354 - Spring 2021

Pilar Luiz : Sophomore, CECS

Lauren Tsai : Sophomore, CECS

Abstract

Our project SINdoku, uses a VGA monitor to display a sudoku puzzle that the user can solve using the Nexys-4 FPGA board. The player will be able to use the left, right, up, and down buttons on the board to move to different cells in the puzzle and can enter by pressing the center button after using the FPGA switches to select a number. After filling out all the empty cells on the board, the user can flip the "check solution" switch to verify their puzzle with the solution. If the user was correct, an LED on the board will light up to indicate it. Otherwise, a different LED will light up to indicate an incorrect answer.

Introduction and Background

We based our project on the GCD lab design. Since the user was able to enter numbers using the FPGA and also used debounced button signals, we thought it would be a reasonable starting point for our project. The top file for GCD also took care of the SSDs and cathode values which made it easy to display the numbers the user selected. However, we could have also used the divider lab as well.

The Design

I. Design description of our implementation

Our main objective was to create a Sudoku game that resembled Android and iPhone Sudoku games. In order to traverse the puzzle, we can use the left, right, up, and down buttons on the FPGA to move in their respective directions. The cell we currently reside in is highlighted red, and will update as we move through the game. Once you're on a blank square, you can use the switches to select a number between zero (eraser) and 9 which is displayed on the SSD. With your number selected, pressing the center button on the FPGA inputs the number in your current cell, updating the state of the board. Once we're finished, we can flip one of the switches on the FPGA to move into the check solution state. An LED lights up, indicating that we are in the correct or incorrect state, which allows us to check our answers. We can flip the reset switch to attempt the puzzle again. The VGA monitor displays the puzzle and inputted numbers while also highlighting our current cell in red.

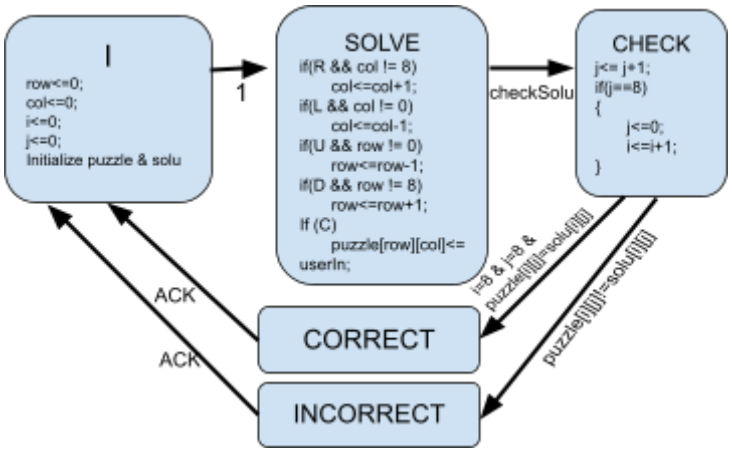
II. Explanation of the state machine

Our state machine has 5 states: initial, solve, check, correct, and incorrect. In the initial state, we reset all the variables i, j, row, and col to be 0. The row and col variables keep track of the location of the cell the user is currently selecting. By resetting the row and col to be 0, we ensure that the user starts entering the numbers to the puzzle on the top left corner when in the SOLVE state. The variables i and j are used in the CHECK state to move one cell at a time in

order to compare the number the user inputted to the number in the solution. In the SOLVE state, the code checks whether or not a button has been pressed. If the right button is pressed and the user is not at the rightmost column of the board, then the col will increment. If the left button is pressed and the user is not at the leftmost column of the board, then the col will decrement. If the up button is pressed and the user is not at the topmost row of the board, then the row will decrement. If the down button is pressed and the user is not at the bottommost row of the board, then the row will increment. If the center button is pressed, the number that the user entered with the switches on the FPGA will be deposited into the puzzle at the location specified by the row and col variables. When the user flips the checkSolu switch high, the state will change to CHECK. At each iteration through the CHECK state, the state machine compares the number the user entered in the cell puzzle[i][j] to the number in solu[i][j]. If they differ, it goes to the INCORRECT state. If they are equal, it increments j (which corresponds to the col) unless it is already at the rightmost column. In that case, j is set back to 0 and i (which corresponds to the row) increments. When at the bottom right corner of the puzzle, if the user's input is the same as the solution, it goes to the CORRECT state. For both the CORRECT and INCORRECT state, it waits for the ACK signal before going back to the initial state.

III. User Interface

For the user interface, we used a VGA monitor to display the puzzle's current state. In the center of the screen, we display a white sudoku board with black lines varying in thickness to differentiate the 3x3 squares. Some cells start off blank, while others are filled in with the



numbers 1 through 9, which we created through if-statement logic blocks. The cell you currently reside in is highlighted red, and this will change locations according to the FPGA buttons, which are used as input. BtnR, BtnL, BtnU, and BtnD correspond to moving right, left, up, and down on the puzzle, and we don't allow the player to move beyond the boundaries of the screen. In order to get user input for the number to place in the current square, we use Sw0, Sw1, Sw2, and Sw3 to read in a number in binary, and we display this number using the rightmost SSD. The center button corresponds to placing the current number in the current position, thus updating the

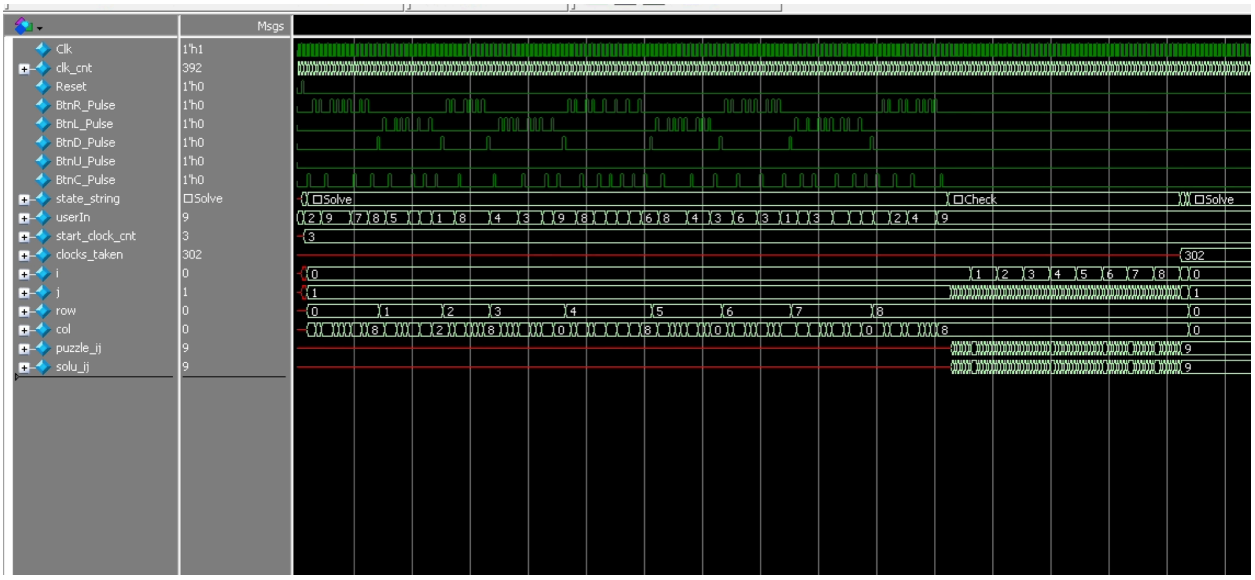
state of the puzzle. Sw13 is used as the signal to move into the check solution state where we compare the user's input with the solution puzzle. LEDs three through seven represent the different states, with Ld7 corresponding to the initial state, Ld6 to the solve state, Ld5 to the check state, Ld4 to the correct state, and Ld3 to the incorrect state. Once the user signals to check their solution, Ld5 will light up with either Ld3 and Ld4 lighting up shortly after. In order to move back into the initial state, you can use Sw14 to send the Ack signal, or Sw15 to send the reset signal. From here, you can play the game again.

Test Methodology

Core Testbench

We decided to create a testbench for the core of the SINdoku program. Since our program takes in many different inputs, we thought it would be easier to make a testbench instead of repeatedly filling out the puzzle each time we made a change. The testbench we created simulates the situation where the user enters the correct puzzle. There are signals for each of the 5 buttons (right, left, up, down, center), reset, checkSolu, ACK, and the number the user entered. We input the numbers starting from the top left and snake the way down the board. At the end of the program, we wait for the program to either go to the CORRECT or INCORRECT state before sending the ACK signal to go back to the initial state. We also create a do file to simulate the testbench and generate the waveform. Using this waveform, we could verify if the program went to the CORRECT state. If it didn't, we could use the waveform to see where the error was in our program and make changes accordingly. After we verified the correctness of our core, it was easy to change some of the testbench signals to force the program into the INCORRECT state. We also prevented the user from moving off of the board in our state logic.

Additionally, we created a testbench for the VGA display in order to quickly debug the visual interface since it took a long time to synthesize the design and program the FPGA. This streamlined testing and allowed us to catch bugs early on.



Conclusion and Future Work

Though it initially seemed straightforward, designing Sudoku was rather challenging because of the strict placement of the numbers and the inability to incorporate random logic. We had to hardcode our puzzle and its solution in the q_Initial state which was tedious and impractical if we were to add multiple puzzles for the user to choose from. In the future, we would read the puzzle and its solution in from a file because it's more flexible and requires less hardcoding. Additionally, displaying the numbers on the VGA monitor was difficult because we had to account for the possibility of any number being placed in any cell which left our implementation looking like spaghetti code. In the future, we would try to make the graphics' code more organized and use repeatable logic instead of tons of if-blocks.

We enjoyed the lab section because it allowed us to practice coding in Verilog and learn the material through trial-and-error. In particular, we really enjoyed the final project because it allowed us to incorporate our own creativity into everything we learned this semester.