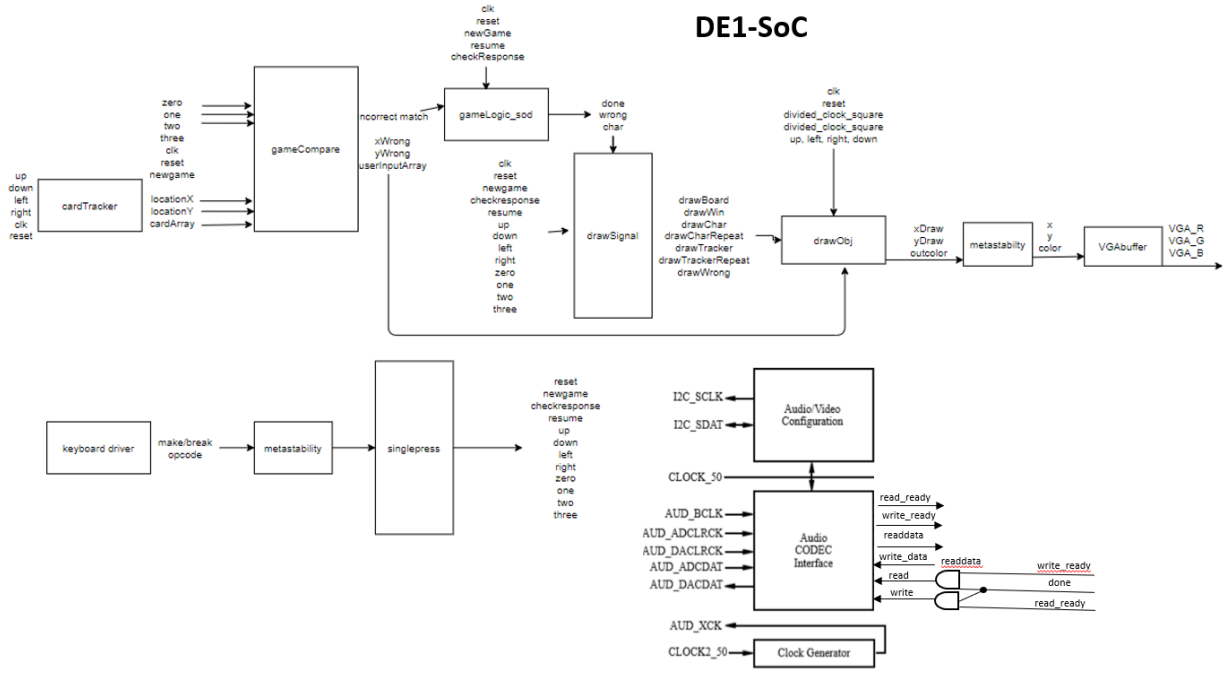
Lab group: Kanika Aggarwal (1660543)
Michelle Chuang (1734793)

Lab 6 Report – Character Sudoku Project

Design Procedure

In this lab, we learned how to connect all of the components worked with separately in previous labs. We used a VGA display, speakers as an audio output, and a keyboard as user input for our game. All of these were connected to the DE1_SoC board. The overall goal of the lab was to choose our own project. We chose to make the Character Sudoku game. A brief overview of the finished project is presented in the Results section. A block diagram of the overall system is shown below and is followed by a detailed description of each module.



DE1_SoC

This module connects all of the below module to create the Character Sudoku system. We made the design decision to have a hex display light up “YAY” and have a moving LEDR display when the game was done. To do this we implemented logic in the DE1_SoC that used the done signal from game logic(discussed below) to trigger the logic for the hex displays and the LEDR display. This module also took care of the keyboard driver outcodes to connect them to logic that got inputted into various methods in our system so that different keys could trigger different actions. It also managed audio, outputting sound when triggered by the done signal. It also worked with the VGA display to send signals from the methods relating to drawing to the objects on the screen.

Draw_obj

Draw_obj implements makeSquare to draw the full objects involved in the Sudoku game which includes the tracker, characters, and board. We made a design decision to only erase one character at a time when dealing with mistakes in the player’s answer to help the player focus on one error at a time. We implemented this with our hide case. We used case statements to organize the order squares were drawn for each component of each character and the color for each component. We made the design decision to

draw the initial empty board slots white because white is a color that is often used for blank spaces. We implemented this by setting color to our 3 bit value for white(3'b111).

makeSquare

Make square implements horizontal line drawer to draw a square using horizontal lines. We made the design decision to start drawing the each square from the top most horizontal line because it was the most logical direction to draw in. We implemented this by initializing the incrementing y coordinate variable to 0 every time a square is initialized.

horizontal_line_drawer

Horizontal line drawer was modified from the line drawer module from a previous 371 lab. We made a design decision to draw all of our objects with horizontal lines and this module implements that by outputting coordinates to strictly draw horizontal lines. We made this decision to simplify our method of drawing our images on the VGA display. This module also takes care of the color signal for the lines drawn.

Meta, metaEleven, metaColor

These meta modules were taken from a 271 lab so we did not testbench them. We have 3 different modules that deal with metastability. The only different between the three are the input and output bit sizes. All of the internal logic is the same. Meta uses 1 bit and it deals with keyboard keys and reset. metaEleven uses 11 bits and it deals with the draw coordinate signals. MetaColor uses 3 bit and it deals with the color signal for the VGA.

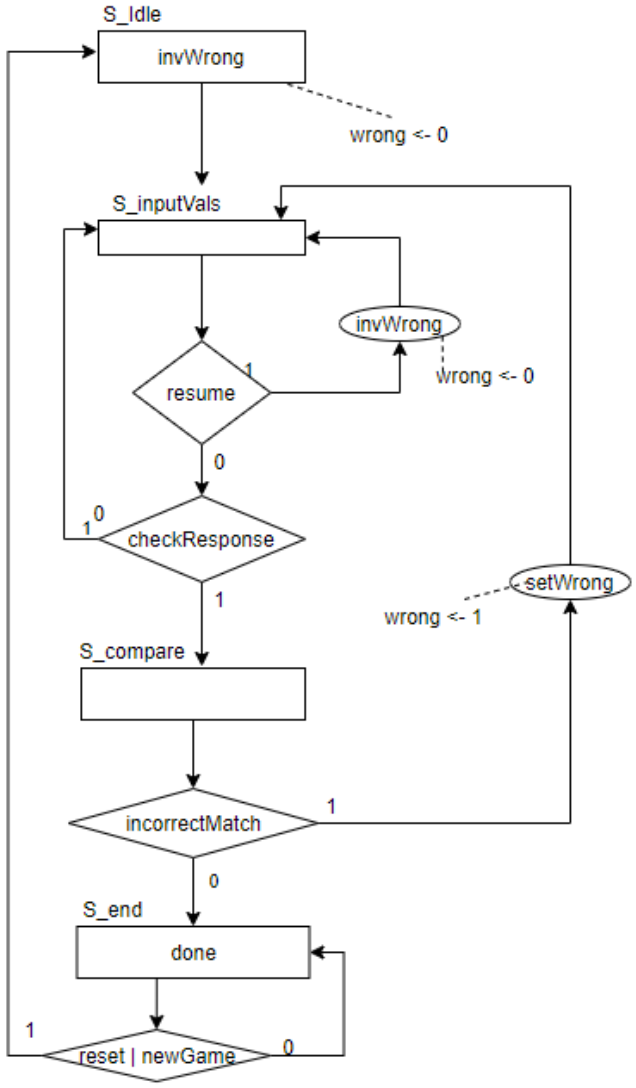
singlePress

This module was taken from an old 271 lab, so we did not testbench it. It makes sure that a button pressed for a longer period of time still only registers as one press instead of multiple presses.

Gamelogic

Lab group: Kanika Aggarwal (1660543)
Michelle Chuang (1734793)

This module's code was guided by the ASMD chart below. We made the design decision to have a resume signal work with the checkResponse signal to make sure the player was ready to resume the game after changing the answer if their answer was initially wrong. It also outputs a done signal that is used in the DE1_SoC module to trigger a VGA display, audio, the LEDRs, and the HEX light display.



drawSignal

This module keeps track of what to draw at certain situations. For example if the tracker is drawn and an arrow button signal is true, the signal for drawing tracker again is true. We implement this with a chain of if-else-if statements for each situation.

gameSel

We made a design decision to have different preset sudoku boards. This would emulate a classic sudoku notebook with preset boards for the player to fill in. We used this module to set up the answer key for each board. We had 4 different boards. Each board was a 4x4. We set each board's answer key to a different case statement.

Lab group: Kanika Aggarwal (1660543)
Michelle Chuang (1734793)

gameCompare

This module compares the user's input to the answer key from game sel. We use if statements to check each card position. This module also initialized characters on the diagonal of 4x4 board to keep consistency throughout preset boards. We implemented this by setting different parts of a 2d array to certain indexes of another array representing different characters.

cardTracker

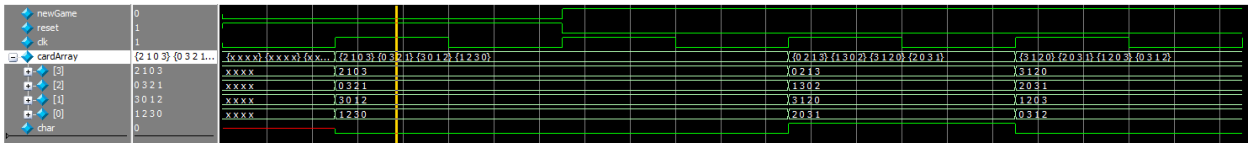
This module keeps track of which location the player is in. We made the design decision that the coordinate will move according to the first direction user chooses. We implemented this by using else if statements in the always_ff block to section out the different movement.

Results

Our final project satisfied the overall requirements of this lab. We chose to make a Character Sudoku game. In our project, we implemented the Audio and keyboard drivers as well as the VGA_framebuffer. Our Sudoku game used a 4x4 board. We displayed blocky images of TV characters including a minion, evil minion, pig, bee, SpongeBob SquarePants, Patrick Star, Toad, and Toadette. The player is able to check their answers

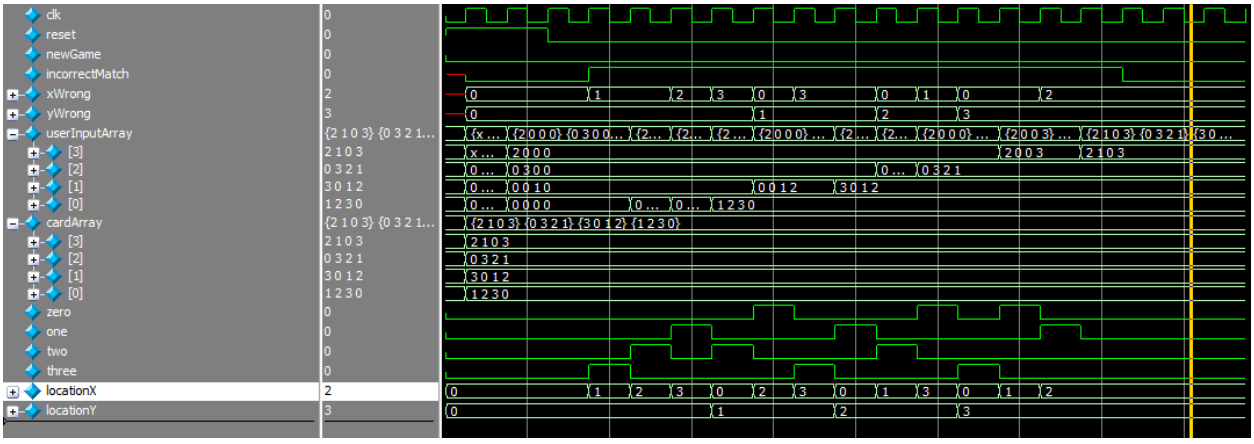
GameselSod

As shown below, when the board is reset it sets to the board value for game counter 0. Every time new game is selected, the board updates and char changes.



GameCompare_sod

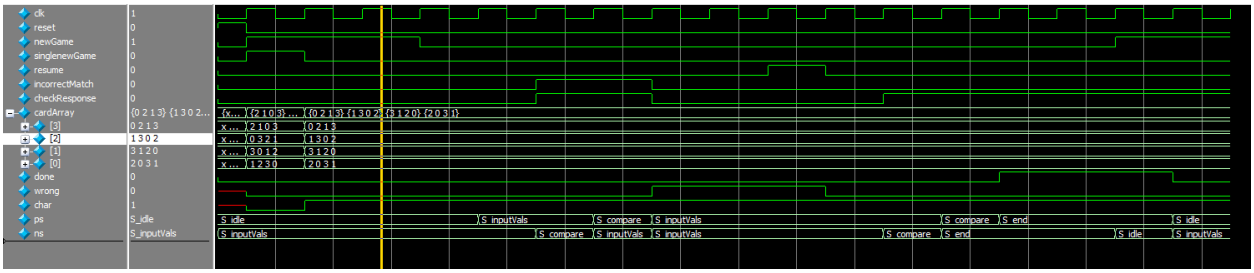
As shown below, user input array is initally all 0's except the locations that are initalized. Then I fill up every element of user input array with the correct values by altering input locationX, locationY, and zero/one/two/three. Once all the correct elements are in userInputArray, the array matches cardArray and incorrectMatch is set to 0.



Lab group: Kanika Aggarwal (1660543)
Michelle Chuang (1734793)

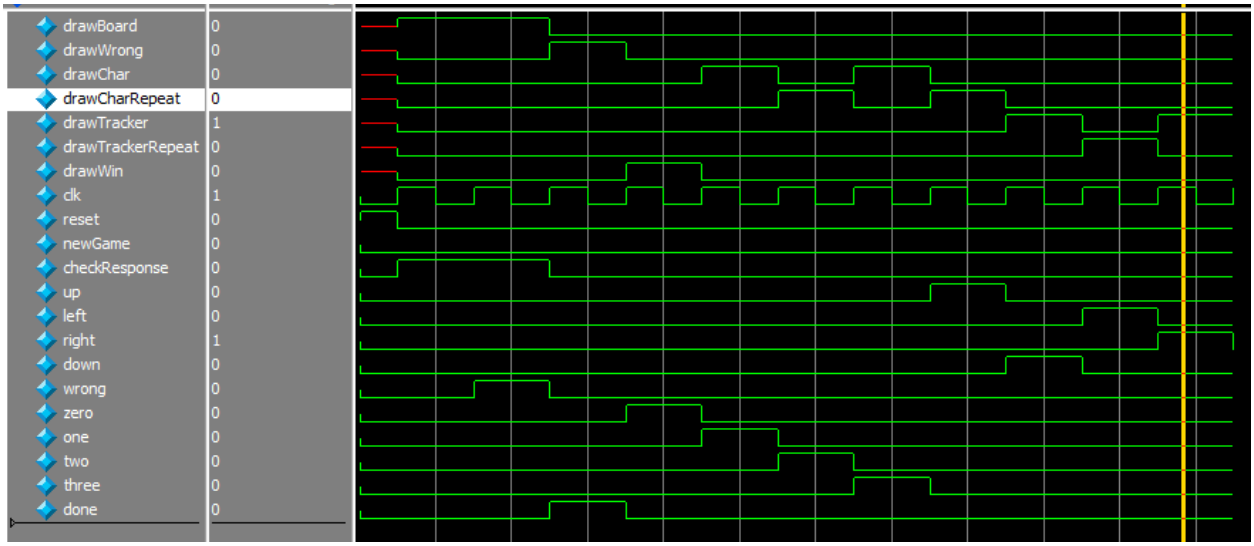
Gamelogic_sod

As shown below when reset is selected the board initalizes to the first iteration and every time new game is selected the board sets to a different game. When reset and new game are deselected the module goes into inputthe inputVal state where it waits for the user to check its responses. When it checks responses and there is an incorrect match the wrong variable is set to true and it goes back to state inputVals. When the user clicks resume the wrong signal switched off. Then when the user clicks check response and there are no incorrect matches the game goes back into S_idle



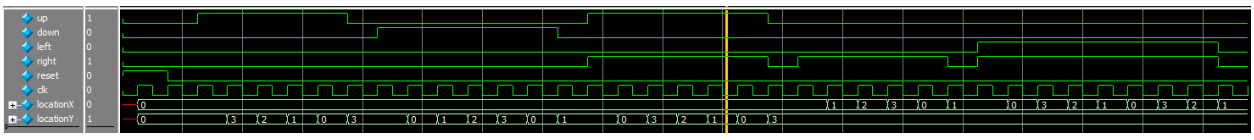
DrawSignal

As shown below only one draw signal is true at a time. When reset is true drawBoard is true. Then I set done to 1 to test drawWin. Then I test the zero/one/two keys to test drawChar and drawCharRepeat. Then I test the up down arrows to test draw tracker and drawtracker repeat.



CardTracker

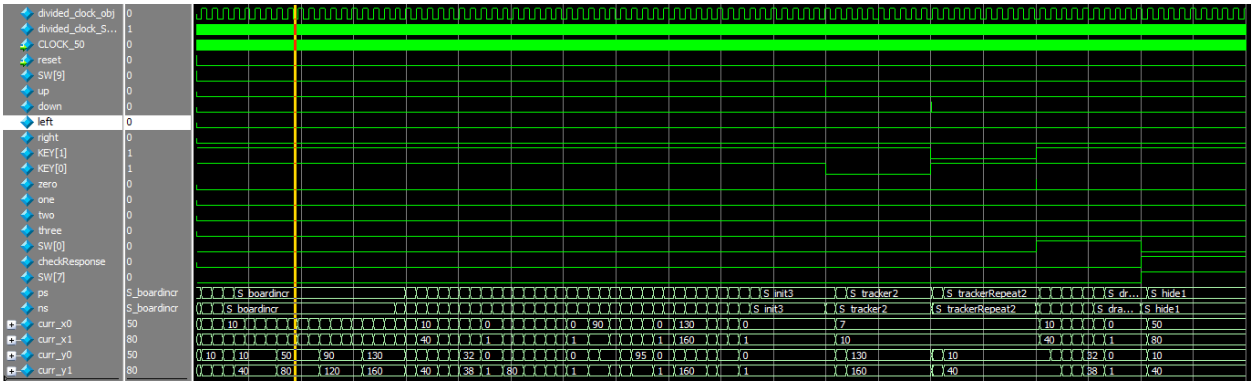
As shown below when up is true location Y increments, when down is true location Y decrements, when right is true location X increments, when left is true location X decrements. If multiple signals are true, only one signal is actually implemented.



De1_soC

Lab group: Kanika Aggarwal (1660543)
Michelle Chuang (1734793)

First, I test reset. As shown below the board correctly goes into the reset state, goes to board incr, and then initializes the sudoku board. Then I test the tracker by changing the inputs for up down. As shown below, it changes to the S_tracker state and the S_trackerRepeat state. Finally, I test check responses, and the module correctly goes into S_hide.

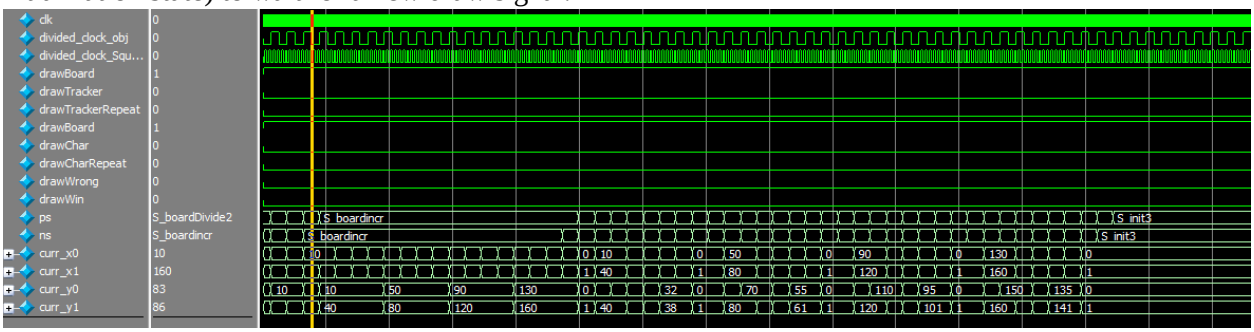


Flow Status	Successful - Wed Jun 10 21:30:24 2020
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC_sodM
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	774 / 32,070 (2 %)
Total registers	726
Total pins	143 / 457 (31 %)
Total virtual pins	0
Total block memory bits	933,888 / 4,065,280 (23 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	1 / 6 (17 %)
Total DLLs	0 / 4 (0 %)

Draw_obj_sod

As shown below, when draw board is true, the inputs into make square set to the correct x0, y0, x1, y1, and color. The states parse through the states to draw the board and then parse through the states to initialize the sodoku board. It draws toad 4 times along the diagonal then halts at S_init3 (the final

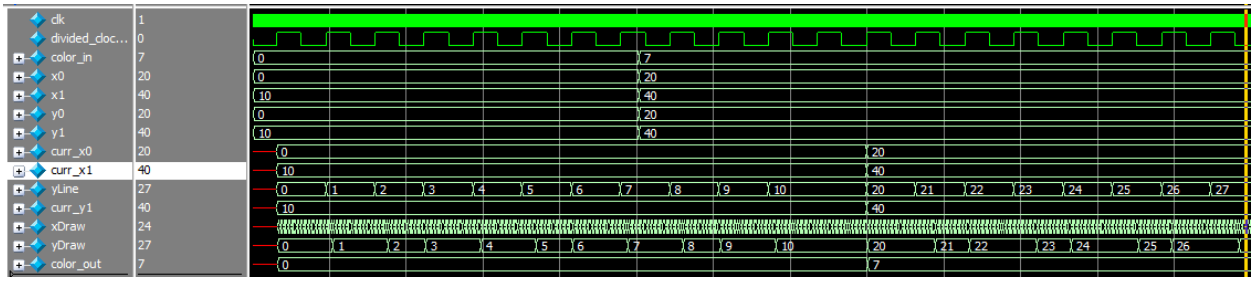
intialization state) to wait for a new draw signal.



When the draw signal changes from drawBoard to drawTracker, it smoothly transitions. S_tracker first successfully erases the previous tracker then draws a new one.

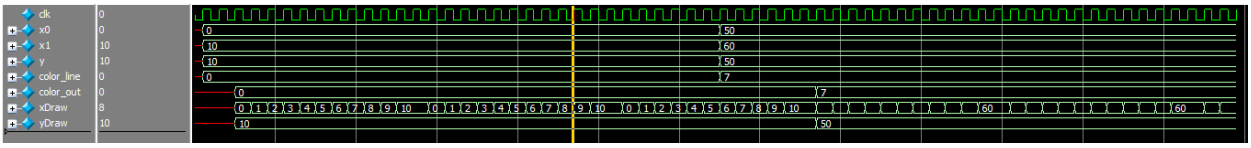
MakeSquare

Shown below, yLine increments from y0 to y1 and yDraw is set equal to yLine. When inputs change, the module waits to finish drawing the square before it updates the inputs into horizontal_line_drawer.



Horizontal_line_drawer

Shown below, xDraw increments from x0 to x1 and yDraw is set equal to y1. When inputs change, the line waits to finish drawing before it updates the x/y coordinates and the color.



Collaboration Report

Lab group: Kanika Aggarwal (1660543)
Michelle Chuang (1734793)

We worked together through the design, implementation, and debugging process for the lab. On the lab report, Kanika focus more on the results section and Michelle focused more on the design procedure. We contributed evenly for the rest of the sections. For this lab, we tried a new way to collaborate on code. We used Visual Studio Live Share which was very helpful in live collaboration. We had some trouble because Kanika's keyboard was not compatible with the DE1_SoC board. Thankfully, Michelle's keyboard worked. To work through this problem, for testing purposes, we swapped out code related to keyboard input for keys and switches so Kanika could still interact with our system. Another similar hardware problem was that Michelle did not have a male to male aux cord so she could not play audio from a device as we intended to. Kanika had the cord and we were able to demo with it. Michelle used a microphone to interact with the audio part of the system. As a tip for future labs, we will make sure to get the correct hardware. Even if we are unable to find compatible hardware, we will remember to look for other ways to substitute for what we are lacking in.

Experience Report

We had originally proposed to do a memory card game. After coding up all the modules, we testbenched the system and all of our logic was behaving as expected. However, when we uploaded to the board, the game logic was not behaving on the hardware as expected. Our character display was correct. We referred back to our experiences in past labs and decided to use a module to reduce any metastability that was causing the inconsistent behavior. Unfortunately, it did not work. We also tried using a clock divider to have some parts of the system run on different speeds. That also did not help. We then decided to shift our focus and repurpose our character display logic for a different game. That's how we decided on character Sudoku. With character Sudoku, when we used metastability and clock divider, we were able to get our desired project. A tip for the future is that if a large shift in the project is made, it is helpful to keep using the parts of the code that are working. In our case, it was the display logic that was working.