ECE241 Project Final Report

Jiale Zhong 1004931853
Brandon Wu 1005218676

1.Introduction

Our game is a single player cooking game that has the player put together raw ingredients to prepare meals, which in our case was burgers and sushi. We chose this game because it is fairly easy to implement a base version of the game and complexity can be increased significantly if time permits. In order to prepare for the recipe, the player has to take the raw ingredients over to the stove/oven to cook first, and then the cooked ingredients are to be taken to the assembly station to prepare the final meal. The chef must cook 10 recipes under a specified time limit to win. Difficulty could be customized by changing the time limit. The player controls the chef through the use of a PS/2 keyboard and the number of burgers left and time are tracked on the board through the LEDs and the hex display respectively.

2.The Design

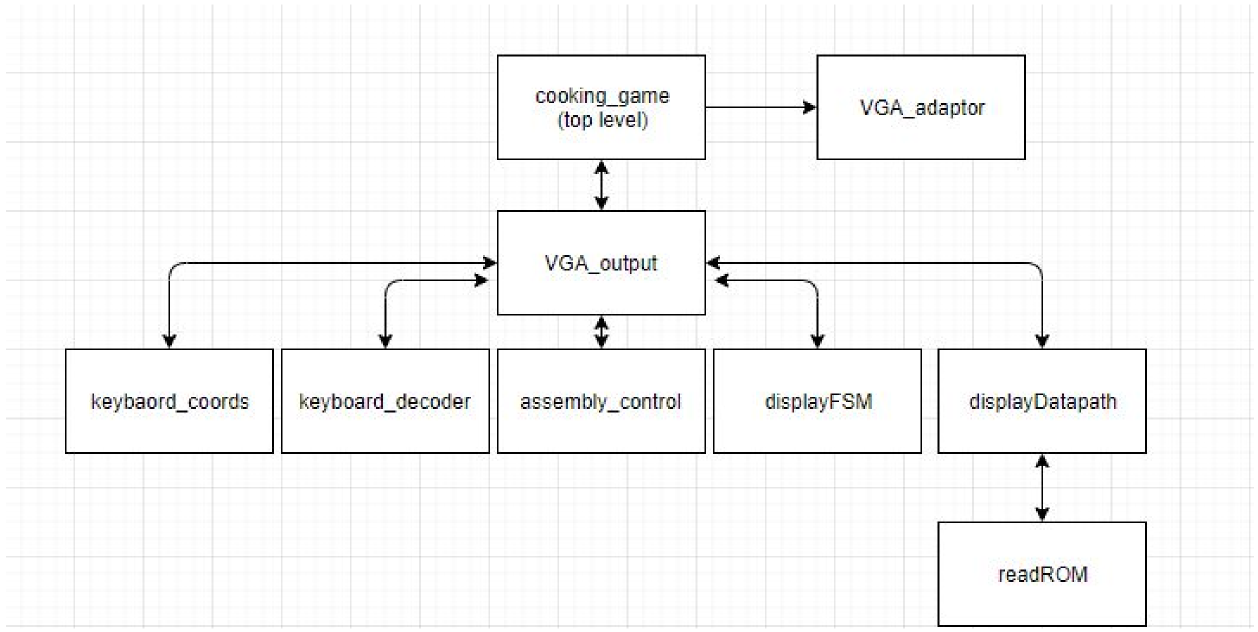

Fig 1. Block Diagram

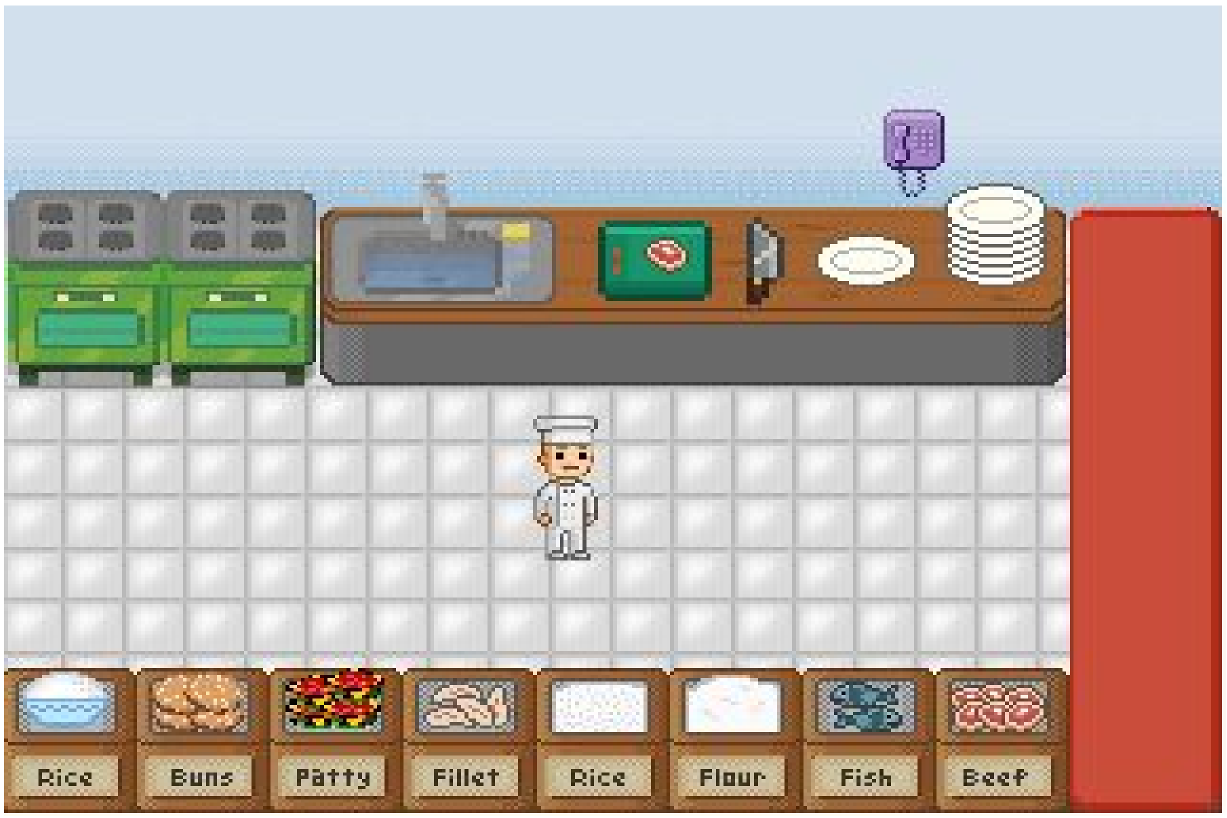

Fig 2. Game Interface(assembly station at the top right corner)

The top level module *cooking_game* instantiates 2 modules which are the *VGA_adapter* and the *VGA_output* module. The *VGA_output* module is responsible for controlling what is written to the VGA display, and does this through a series of submodules. The modules instantiated are

- keyboard_coords
- keyboard_decoder
- assembly_control
- displayFSM
- displayDatapath.

The first three modules instantiated by the *VGA_output* module contains the bulk of the game logic. The *keyboard_coords* module initializes the position of the character to the center of the screen and changes the coordinates of the player according to the keys pressed. These coordinates are then passed to the *keyboard_decoder* module, which checks if the player is in the area in front of an ingredient or the cooking/assembly station, and if so, sends an area signal to the game logic FSM in *assembly_control* when the space bar is depressed. The *assembly_control* module then checks if the area the player is in is valid for the next step of a recipe, and changes its internal state accordingly.

The state of the *assembly_control* module is then passed to the displayFSM module, which uses it to decide which module load signals to send. The displayDatapath module then takes these load signals, sends a load signal and the desired sprite to the readROM module, and waits for a done signal before allowing the displayFSM module to draw the next picture.

3.Report on Success

On presentation day we were able to demonstrate a bare-bones version of our project. The player was able to be controlled and could make burgers, and the sprites and art worked properly. Additionally, we created a down counter for the hex display, and the base functionality of the game was implemented.

The most important success of our project was actually the scalability of our game. The modules were all connected in a way that including more game logic would only require minor modifications to our game logic FSM, display FSM/datapath, and ROMs. Functionality such as difficulty settings would basically have an additional FSM state for the start screen, and animating the chef for the different directions that they could face would take in the last pressed key and load the correct sprite accordingly. Carrying food items and serving of the dish would work similarly, with the main challenge being creating a whole bunch of MIF files.

In the scramble to finish our project we failed to develop a few things, most notably the PS2 keyboard module. This module is supposed to recognize the make and break codes from the keyboard and raise and lower the signals which represented the keys. The reason why it failed was that it devoting resources to it was a poor use of time considering we could already use the onboard keys as input. The resources available on the internet to solve some of the problems we had were sparse, and ultimately, we were only able to implement the module without reading the break codes which did not suit our purposes. Another issue was keeping track of the number of food items left to cook in under the time limit. We intended on using the LEDs, with all of them being powered on by default and bit-shifting the register to turn off the LEDs when a recipe was completed. The failure of this module lies in the code written in the assembly_control module. The increment function of the game logic sits in a state which is followed by a delay, resulting in all of the bits being shifted to 0. If time permitted, fixing this issue would not have been tremendously difficult as it would likely just require the addition of another state.

4. What would we do differently

1. Initially we both came up with a list of ideas. Although we finally decided on this cooking game, our visualization on the game interface and the game mechanics was very different. Therefore, during the first week, when we were working towards our first milestone, we wasted time because we each completed our own part of the work but soon realized that our own ideas of the game require different implementations than what our partner wrote. If we had a chance of redoing the project, we would make sure that we start off with the same understanding of the game (perhaps hand draw a game interface or make a chart to illustrate the flow of the game).
2. In retrospect, our goals were broad and didn't focus on the important parts of making our project work. Things like creating the display module FSM was infinitely more important than advanced game logic modules like keeping track of the number of ingredients and cooking it. If we could redo the project, we would spend significantly more time at the beginning of the project laying out the barebones functionality and only later implementing the more complex logic.

Appendix: All verilog code and schematics

See attached files.