Group S3-12
Members: Poh Yu Wen A0271715E, John Woo Yi Kai A0272561E, Roma Shrikant Joshi A0281973R, Asher Tan Shih A0283041N

PERSONAL AND TEAM IMPROVEMENTS

Student and Improvement Name	Improvement Description	Images/Photos

Team S3_12

"Cooked"

Overview
Cooked is a single-player cooking game inspired by *Overcooked*. The objective is to fulfil as many food orders as possible within a 5-minute time limit by collecting, processing, and serving the right combinations of ingredients across different cooking zones. Strategic movement, time management, and precision in mini-games are key to scoring well.

How to play?
1. Pick up ingredients from the ingredient zones.
2. Process ingredients by bringing them to the chopping/boiling stations. A mini-game will appear on the 7 segment display. Complete it to finish chopping/boiling. Do not leave the zone mid-task — doing so will fail the mini-game and the ingredient will be lost. Upon completing mini-game, an animation will start, so feel free to come back after it is done to collect the ingredient!
3. Serve orders by carrying ingredients to the serving station. If the submitted ingredients match an active order, it will be fulfilled automatically.

Controls

Action	Input
Move (Up/Down/Left/Right)	btnU, btnD, btnL, btnR
Pick Up / Put down ingredients	btnC
Open Doors (Top/Bottom/Left/Right)	sw[3] / sw[2] / sw[1] / sw[0]
Perform tasks (Chopping/Boiling)	sw[13:10] / sw[15:8]

Displays
- **Left OLED**: Shows active orders, inventory and ingredients needed
- **Right OLED**: Shows player position and map layout
- **7-Segment**: Displays timer, tasks and final score

Map
The map is split into 4 quadrants by walls, where the player can move from quadrant to quadrant through doors. Only **one door** is allowed to be open at a time. Turning on multiple door switches will **shut all doors**.

Zones
- **Ingredients Zone** – pick up raw ingredients (onion, rice, chicken, tomato)
- **Stations** – put in ingredients to cook/serve (chopping, boiling, serving)

Mini Games
Chopping Station:
The **7-segment display** will show **4 different letters**, each cycling like a slot machine. When the **correct letter** appears, its corresponding **LED** below will light up. Quickly **flip the switch** that matches the lit LED to "lock in" that letter. Repeat this for all 4 letters.

Boiling Station:
A set of LEDs from LED[15:8] will light up. Match the lit LEDs **exactly** by flipping on/off the **corresponding switches** (i.e. sw[15:8]). This repeats for **4 rounds**.

How to Win?
Complete as many food orders as possible **within 5 minutes**. The game ends when the **timer reaches 0**, or **all orders are completed** where you will then get your final score.

Cooked!

Student A:

Asher Tan Shih

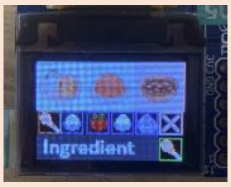
Generate Random Orders, Left OLED, Doors Logic, Attempted UART

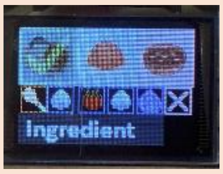
Generate Random Orders
When the game is started, 3 orders are selected from a list of 4 possible dishes (repeats allowed). This is done via a LFSR which takes in the seed of time from program start, until btnC pressed (for proceeding from start screen). This ensures a pseudorandom seed each time. Once the dishes are selected, their corresponding required ingredients to make the dish are also selected and displayed.

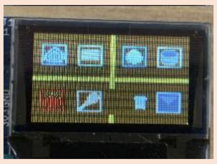
Left OLED
Programmed the left OLED display. This involved taking the generated dishes, extracting the required ingredients for that dish, then displaying it. Also took in the inventory wire from higher modules to display the current ingredient in inventory. Also took in the completed dishes to display a visual indicator (overlay tick) showing that the dishes are done. Designed and laid out the assets, also had to flip the sprites and layout in order to adapt it from a right-side OLED (JB) to a left-side OLED (JXADC).

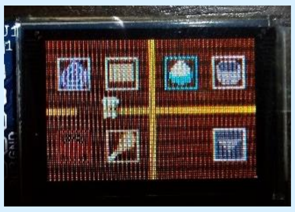
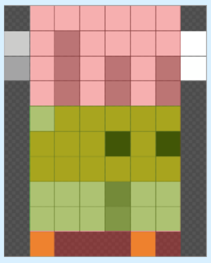
Doors Logic
When a user toggles any of the sw[3:0], the corresponding door between the quadrant on the map is opened. However, only one door can be opened at any time. If the user tries to open more than 1 door, all the doors will close. This involved modifying the collision logic to add additional conditions as well as modifying the map drawing to correspondingly display/remove the doors. This was integrated into the working program.

UART
Attempted UART for multiplayer between 2 Basys3 Boards. Was unable to get a stable connection between the devices via the PMOD headers in order to transfer data consistently, scrapped the feature afterward.





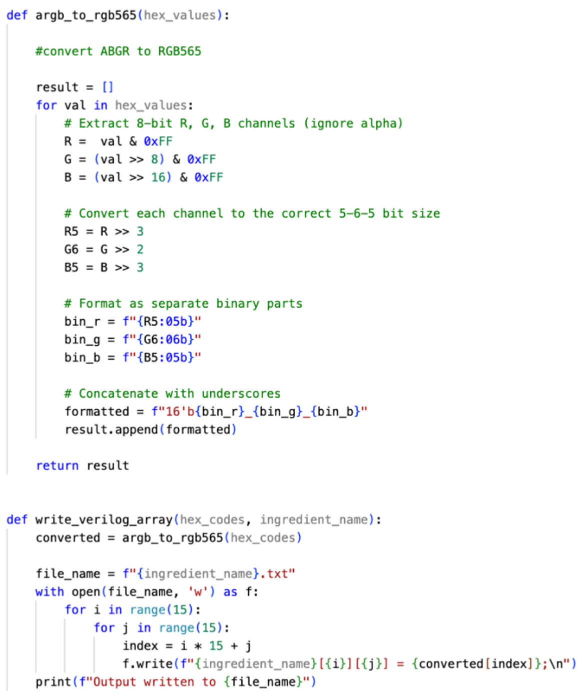


Student B: **Poh Yu Wen** **Map, Movement, Collision, Zone detection, Ingredient logic, btnC debouncing, integration**	**Map & Movement** Rendered a map with four quadrants separated by walls with narrow **gaps** on the right OLED (JB), where the character can move through by using **btnU, btnD, btnL, and btnR**. Movement is **discrete** instead of continuous to allow the character to stop precisely in zones. **Collision Logic** Implemented collision detection for the OLED screen edges and the walls of the map. The player is also able to **collide with the walls inside the gaps**. Collision parameters such as wall thickness, gap size, and player dimensions are all **customizable** through Verilog parameters. A custom 6×10 rectangular hitbox is used for the player, slightly smaller than the 8×10 character sprite, allowing the character to move naturally **without colliding with non-visible parts** like the gap between the head and body. **Zone Detection Logic** Seven zones of size 15×15 pixels are placed across the map, demarcated by their respective icons. A character is considered within a zone only when the **entire player hitbox** is inside the zone boundaries. A **separate** zone detection hitbox of **6×6 pixels**, allows for easier player alignment within zones, especially given the character's large rectangular shape and the limited precision of pushbutton controls. **Ingredient Logic** Expanded Student C's inventory management framework to work with the map where players are allowed to pick up **only** from ingredient zones and completed stations. Implemented chopping/boiling logic where any ingredient can be **boiled, chopped or both**, and players can view the ingredient through the inventory on the left OLED. Implemented serving logic where ingredients placed at the serving station are **checked against active orders** and removed from the server inventory when completed. In addition, If a player fails a mini-game or exits a zone prematurely, the ingredient is **removed from the inventory**, reflecting a failed cooking attempt. **Centre Button Debouncing** Implemented robust and responsive picking up and putting down of ingredients **every 100ms** using the **same** push button btnC. A pulse of one clock cycle is generated regardless of the duration of the press, and it is used to toggle the PickUp flag. **Integration & Enhancement** Integrated **all** modules, including each of the FSM written for minigames, animation, timer, left OLED etc.	 **Player collision with the walls of the gaps**  **Character Sprite** Red overlay indicate collision hitbox and green overlay indicate the zone detection hitbox
Student C: **Roma Shrikant Joshi** **Minigame, ingredient management, and graphics**	**Chopping minigame** Implemented chopping minigame for one station involving 7-segment display, LEDs and switches. **Inventory management** Implemented inventories for the player, ingredient stores and the chop/boil/serve stations. Implemented logic for player picking up and putting down items. **Station timing logic** Implemented timeout and animations for chop and boil stations. **Graphics** Created all in-game graphics: • Start and end screens • Station icons, animations and timer animations • Character sprites • Inventory icons ○ Different icons depending on state of ingredient (raw vs boiled vs chopped) Start screen was created by using the Picture2Pixel library to generate a .mem file. The rest of the graphics were created on Piskel and converted from Hex to RGB565 format using a python script (in appendix). **UART** Attempted UART for multiplayer between 2 Basys3 Boards. Was unable to get a stable connection between the devices via the PMOD headers in order to transfer data consistently.	 **Chopping minigame** **Character sprites and inventory and timer icons**
Student D: **John Woo Yi Kai** **Boiling Minigame, Countdown Timer and Score**	**Boiling Minigame Logic** Implemented the boiling minigame. Once started, pseudo-random LEDs (LED[15:8]) and switches (sw[15:8]) combinations are generated using a **LFSR (Linear Feedback Shift Register)**. Switches will then have to be **simultaneously switched on or off** to match the corresponding LEDs blinking. This is checked using a 3Hz clock to allow for some leeway. Once done, a new pattern will be shown to follow. Users will have to match 4 different patterns to pass the task. The 7-segment display will light up, with each anode showing how many patterns have been matched so far. If the user fails to match the corresponding LEDs or did not flip the switches simultaneously in time, the task is considered to be failed. **Countdown Timer Logic** Implemented a timer with tenth digits. Once the centre button btnC is pressed at the "Start Game" screen, the 7-segment will display a timer starting with a minute, ten second, second and tenth digit displayed on corresponding anodes an[3:0]. The decimal points on the minute anode and second anodes are used to annotate the cutoff from minutes to ten seconds, and seconds to tenths. Getting the tenth display countdown to work was based on a 7-segment stopwatch (in appendix). **Score Logic** Implemented the final score. Once the game has ended by completing all orders, the 7-segment will switch to show your overall score for the game. The score is calculated by having the **number of orders completed * 3 + seconds left** on the 7-segment timer when game is completed.	

Group S3-12
Members: Poh Yu Wen A0271715E, John Woo Yi Kai A0272561E, Roma Shrikant Joshi A0281973R, Asher Tan Shih A0283041N

References Used:

1. Overcooked (For original idea of project) [https://www.youtube.com/watch?v=nSf5wdTcTuo] Reference video of gameplay
2. https://www.piskelapp.com/ (For creation of digital assets)
3. https://www.pixilart.com/draw?ref=home-page (For creation of digital assets)
4. https://www.comp.nus.edu.sg/~guoyi/project/picture2pixel/ (For creation of digital assets)
5. Python script to convert Piskel to RGB565 (Generated with ChatGPT)



```python
def argb_to_rgb565(hex_values):

    #convert ABGR to RGB565

    result = []
    for val in hex_values:
        # Extract 8-bit R, G, B channels (ignore alpha)
        R =  val & 0xFF
        G = (val >> 8) & 0xFF
        B = (val >> 16) & 0xFF

        # Convert each channel to the correct 5-6-5 bit size
        R5 = R >> 3
        G6 = G >> 2
        B5 = B >> 3

        # Format as separate binary parts
        bin_r = f"{R5:05b}"
        bin_g = f"{G6:06b}"
        bin_b = f"{B5:05b}"

        # Concatenate with underscores
        formatted = f"16'b{bin_r}_{bin_g}_{bin_b}"
        result.append(formatted)

    return result


def write_verilog_array(hex_codes, ingredient_name):
    converted = argb_to_rgb565(hex_codes)

    file_name = f"{ingredient_name}.txt"
    with open(file_name, 'w') as f:
        for i in range(15):
            for j in range(15):
                index = i * 15 + j
                f.write(f"{ingredient_name}[{i}][{j}] = {converted[index]};\n")
    print(f"Output written to {file_name}")
```

6. https://www.youtube.com/watch?v=_ul0hrcpkbc (John: Referenced this video to work out how to get a timer working with a tenth digit)
7. ChatGPT for:
   a. Generation of mini game ideas
   b. Refinement and feedback on Verilog code, particularly at times where we were stuck
   c. Sanity check to check for small mistakes e.g typos
   d. Generation of module instantiation with named invocations to save time from typing 20+ parameters and inputs / outputs

Feedback:

- Practical Exam:
  - The questions were too long and complex, leaving little time to plan and understand before coding. The time provided for reading could either be increased, or a demo could be shown, like the lab assignments. (Yu Wen)
  - It would be great to have more tasks that are not highly coupled with each other. (Yu Wen)
  - Practical should have slightly more time, like at least 15 minutes more for coding. (John)
  - Task B should be split up even further – it's really too big to implement as a whole, and should be modularised further into 2 different tasks. (John)
  - Evaluation can benefit from showing the intended outcome so as to have no ambiguity (Asher)
  - Evaluation could be more modular to ensure that if stuck, can still get points (Asher)
- Quizzes:
  - Allocate more space for each question as there was not enough space to write and draw diagrams, tables etc. (Yu Wen)
- Project:
  - More components should be provided to experiment with the FPGA. (John)
  - Should have more components to play around with (Asher)
  - Vivado is not a great IDE, unable to Undo after saving, and linking of files from other members is difficult (Asher)
  - Can also showcase more unique use-cases/real-world use cases of FPGAs for project references (image filter, hashing) (Asher)
- Labs:
  - Appreciated the format of having a "practice" before graded portion (Asher)
  - Some conditions and tasks were unnecessarily verbose (Using matric number to determine different task outcomes) leading to confusion and time spent on understand the desired behaviour (Asher)
  -
- General:
  - More guidance with writing Verilog code. Perhaps show the method and the partial solution for assignments, or have TAs feedback on the Verilog code submitted. This is as the mistakes would often get carried over to the project, and we want to identify them and fix them asap. (Yu Wen)
  - The timing between the quizzes, project proposal drafts, and bitstream submission, was especially tight between the week 7 – 9 period. (Yu Wen)
  - Recommend a different text editor other than the default one provided by Vivado for code editing – suggest use of VSCode to edit code and save it and use Vivado 2018.2 for the bitstream generation. (John)