Tetris FPGA:
A DE1-SoC Implementation of Tetris

Embedded Systems Design (CSEE4840)
Spring 2025

Michael Lippe (ml5201), Garvit Vyas (gv2361), Bhargav Sriram (bs3586)

1. Introduction

Tetris is a classic puzzle video game revolving around the strategic placement of falling geometric shapes known as Tetrominos. The goal is to rotate and arrange these pieces in such a manner that forms complete horizontal lines, which are then cleared from the screen, and points are given based on the number of lines cleared. As the game goes on, the falling speed of the blocks increases, and thus so does the difficulty.

In Tetris, gameplay revolves around manipulating geometric shapes known as Tetrominos, each consisting of exactly four squares arranged in seven distinct configurations. These configurations are labeled based on their shapes: I, O, T, S, Z, J, and L.

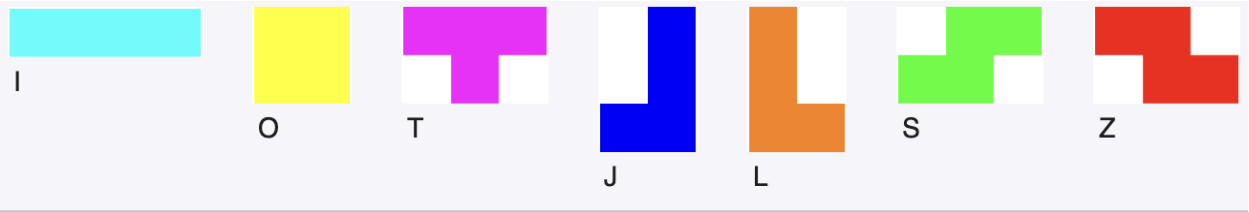


Figure 1: Tetrominos (Courtesy of Wikipedia [7])

Each Tetromino can be rotated by 90-degree increments, allowing four possible orientations for each piece (except the O-Tetromino, which remains unchanged upon rotation). Players strategically rotate and position these pieces as they descend, aiming to fill horizontal rows completely to clear them from the playing field and score points.

The specifics for each Tetromino are:

- **I-Tetromino**: Straight-line shape, can rotate between vertical and horizontal orientations.
- **O-Tetromino**: Square shape, rotation does not affect its configuration.
- **T-Tetromino**: T-shaped configuration, has four distinct rotational states.
- **S- and Z-Tetrominos**: Mirror-image zigzag shapes, each has two rotational states due to symmetry.
- **J- and L-Tetrominos**: Mirrored L-shaped configurations, each with four rotational states.

Our project implements a hardware-software system capable of playing Tetris. To accomplish our goal of playing Tetris, we use an SNES-style USB controller, the DE1-SoC FPGA board, a VGA monitor, a USB DAC, and a speaker.

2. Project Overview

The project can be broken down into five major sections: Input devices, software run on the HPS, hardware instantiated on the FPGA, built-in peripherals on the DE1-SoC board, and output devices. The block diagram for the project is shown in Figure 2, where the various components have been grouped together into the five major sections.

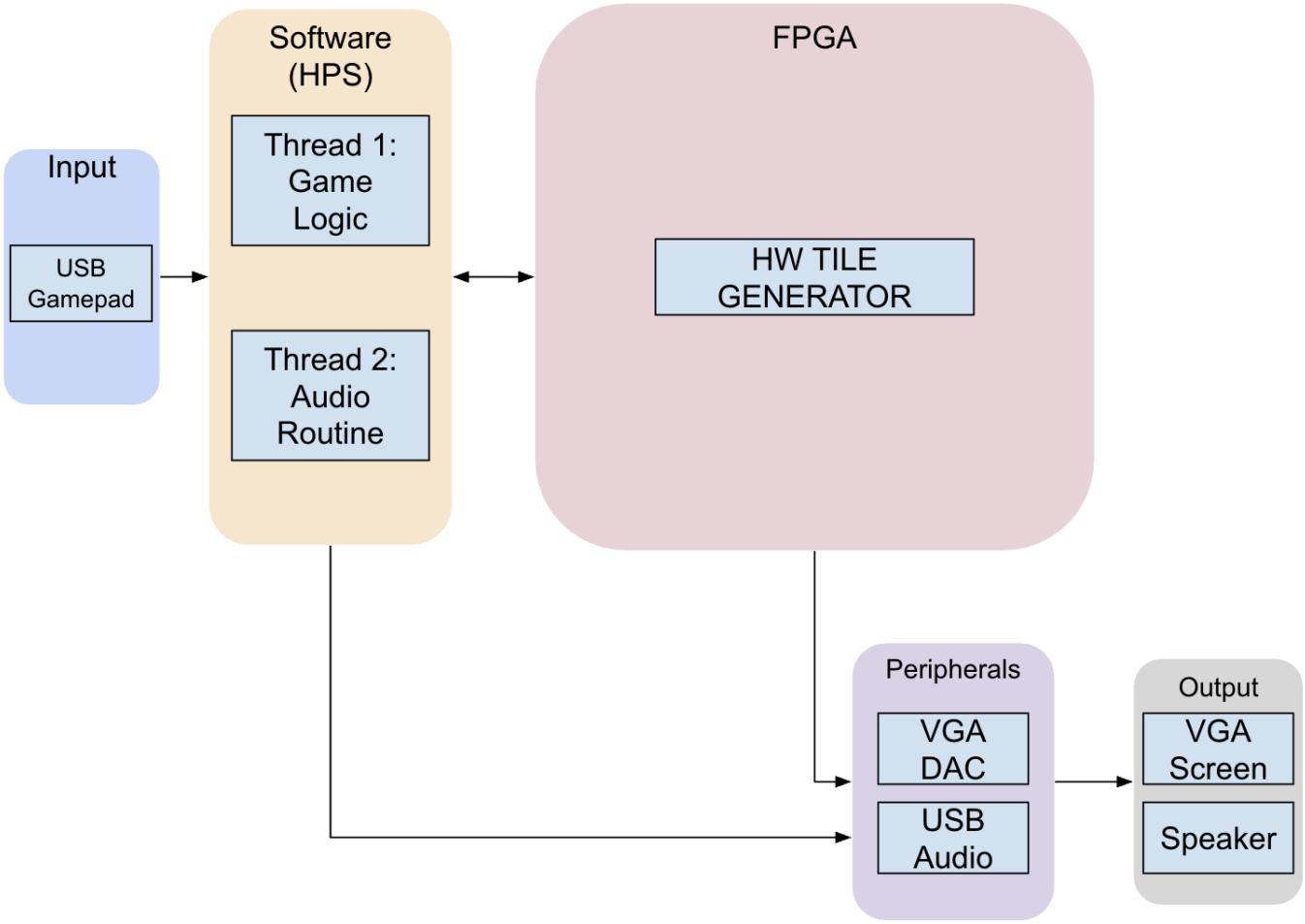


Figure 2: System Block Diagram

3. Hardware

3.1. Initial Plans

Initially, our aim for graphics was to design a Picture Processing Unit (PPU) loosely based on the on found in the Nintendo Entertainment System. However, after designing standalone testbenches for and verifying several of the verilog modules for our original design, we realized that we made our system too complex and verifying everything together within a week and a half while also figuring out the compilation and software was not feasible As such, we decided to pivot to a simpler tile only approach as Tetris does not need sprites. The original PPU hardware is still worth going over, however, as significant time and effort was put into it and the learnings from it contributed significantly to the final project.

The Picture Processing Unit (PPU) was designed to drive a 640×480 VGA display at 60Hz. Graphics were built from 16×16 tiles where each pixel was encoded in 4 bits, and each tile was referenced via 8 bits of combined attribute data, specifically a 1-bit palette ID and a 7-bit tile ID. The PPU would have supported storing 128 unique background titles in memory and 128 unique sprite tiles. The tile buffer would have held 1,200 tiles since the tile resolution of the screen would be 40 x 30. Each color palette had 4 colors, and there were a total of 2 palettes. The system would have allowed for up to 128 sprites on screen at once, with a maximum of 8 sprites per scan line. For sprites, we planed to use a priority encoder to control shift registers. To minimize the critical path for the circuit that determines which of the 8 sprites or the background a given pixel should come from, a divide and conquer approach would have been adopted so that rather than a chain of 9 muxes, it would be a chain of length 4.

Object Attribute Memory (OAM) was used for storing data about each of the 128 sprites, where each sprite would be described with an 8-byte entry and addressed via 32-bit words. Table 1 shows the per-sprite mapping of OAM.

Table 1: OAM Map Per Sprite

Words	**Field**
Word 0	[31] Vertical Flip Bit, [30] Horizontal Flip Bit, [7] Color Palette ID, [6:0] Sprite Tile ID
Word 1	[31:16] Y cord, [15:0] X cord

The main memory blocks in the PPU were the Tile Buffer, Tile Graphics Memory, Sprite Graphics Memory, Color Palette Memory, and Object Attribute Memory; each would have been implemented individually, with different widths based on the data it stores. The various PPU memories would be exposed to the HPS as a single virtual VRAM. Table 2 shows the proposed addressing scheme along with the width and size of each memory.

Table 2: VRAM Information

	Addressing Scheme Bits [12:0]	**Width**	**Size**
Tile Buffer	`0x00XXBBBBBBBBB`	32-bits	1200B
Tile Graphics	`0x01TTTTTTTTTT`	32-bits	8192B
Sprite Graphics	`0x10SSSSSSSSSS`	32-bits	8192B
Color Palettes	`0x110XXXXXXXPPP`	24-bits	24B
OAM	`0x111XXOOOOOOOO`	32-bits	1024B

An initial draft of this PPU design was implemnted in System Verilog, with the corresponding files shown in Section 8.1. Please note that none of this code is used in the final implementation of the project.

3.2. Final Hardware

3.2.1 Final Hardware Details

With the pivot to a purely tile based approach, we based our final graphics hardware implementation on Professor Stephen Edwards' tile generator [1] [2]. We modified tiles.sv to add double buffering to reduce flickering by keeping the tilemap being read by the video hardware consistent throughout a given frame.. Figure 3 shows the general block diagram for the graphics hardware.

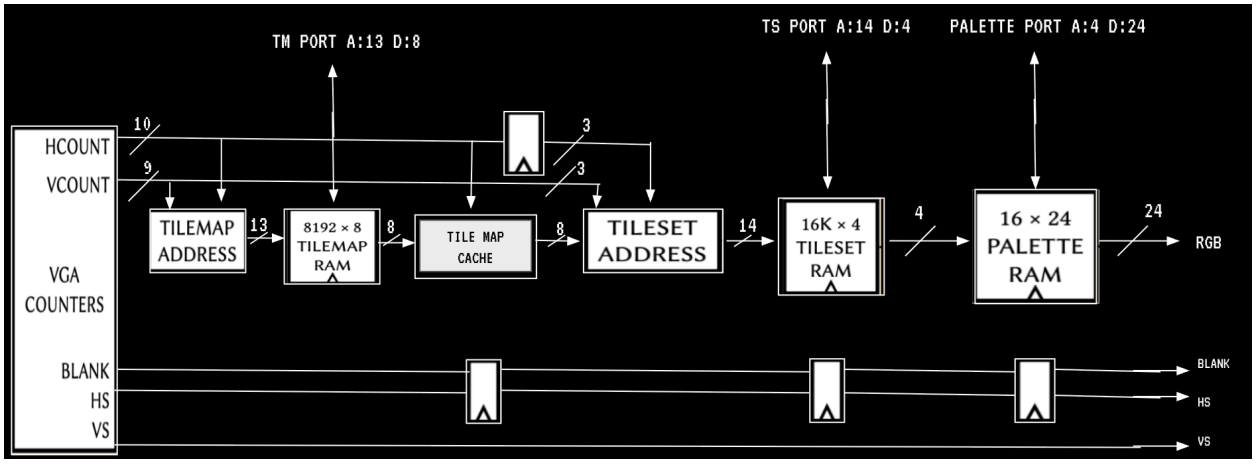


Figure 3: HW Block Diagram (Based on Professor Edwards' diagram [1])

As Figure 3 shows, the double buffering was accomplished by instantiating another 8192 x 8 BRAM that sits between the tilemap ram and the video generator. The System Verilog code for the double buffering logic is shown below:

```
//CPU side tile RAM
 //port-A (clk = mem_clk) from Avalon bus
 //port-B (clk = VGA_CLK) to frame cacher
 twoportbram #(.DATA_BITS(8), .ADDRESS_BITS(13)) tilemap_cpu (
    .clk1  ( mem_clk ),
    .addr1 ( tm_address ),
    .we1   ( tm_we ),
    .din1  ( tm_din ),
    .dout1 ( tm_dout ),

    .clk2  ( VGA_CLK ),
    .addr2 ( copy_addr ),
    .we2   ( 1'b0 ),
    .din2  ( 8'hxx ),
    .dout2 ( tm_cpu_dout )
 );

 //Display side tile cache
 //port-A (clk = VGA_CLK) from pixel pipeline
 //port-B (clk = VGA_CLK) to frame cacher (writes)
 twoportbram #(.DATA_BITS(8), .ADDRESS_BITS(13)) tilemap_disp (
    .clk1  ( VGA_CLK ),
    .addr1 ( { vcount[8:3], hcount[9:3] } ),
    .we1   ( 1'b0 ),
    .din1  ( 8'hxx ),
    .dout1 ( tilenumber ),

    .clk2  ( VGA_CLK ),
    .addr2 ( copy_addr_d1 ),
    .we2   ( copy_we ),
    .din2  ( tm_cpu_dout ),
    .dout2 ( )
 );

 //Per-frame cacher (runs during vblank)
 //starts at the first pixel of vblank (vcount = 480, hcount = 0)
 always_ff @(posedge VGA_CLK or posedge VGA_RESET) begin
    if (VGA_RESET) begin
```

```
            copying   <= 1'b0;
            copy_addr <= 13'd0;
         end else begin
            if (!copying && hcount == 10'd0 && vcount == 10'd480) begin //start of
vblank
               //start caching
               copying   <= 1'b1;
               copy_addr <= 13'd0;
            end else if (copying) begin
               copy_addr <= copy_addr + 13'd1;
               if (copy_addr == 13'd8191) //finished copying tile map into the
cache
                  copying <= 1'b0;
            end
         end
      end
```

The always_ff block we have labled as the per-frame cacher is responsible for managing the communication between the tilemap RAM and the tilemap cache. At the start of the vertical blanking period, the per-frame cacher starts copying data from the tilemap RAM into the tilemap cache. As Shown in Figure 4, given that the hardware is designed to support a resolution of 640 x 480, the total vblank period, vertical front porch plus vertical back porch, is 43 lines. With 800 pixels per line, we get 34,400 pixels, and thus clock cycles, of vblank. Since we only have 8192 addresses to copy, and a copy only takes one clock cycle, we have plenty of time to cache the state of the tilemap before we start drawing the next frame.

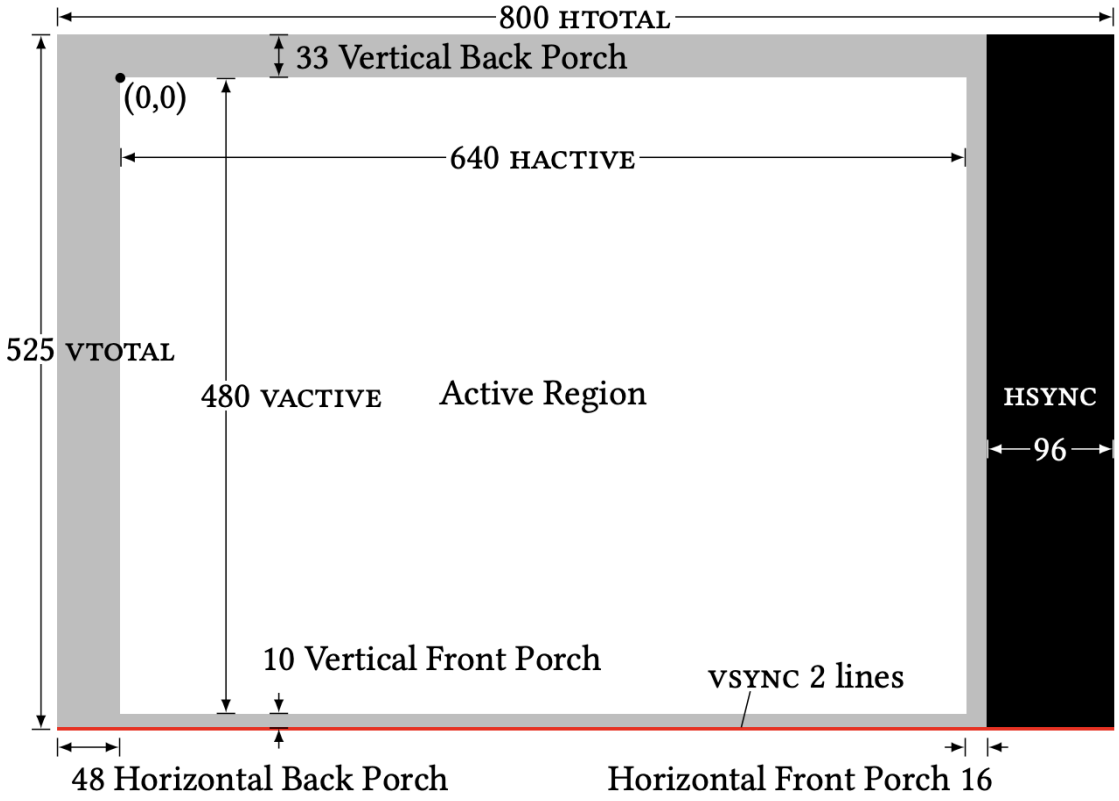


Figure 4: VGA Region Map (Courtesy of Professor Edwards [1])

In terms of the overall design of the video generator, the hardware supports a resolution of 640 x 480 at 60hz. Tiles are 8 x 8 pixels with 4 bits per pixel. There is a single color palette with 16 colors. The tileset memory supports storing the graphics for up to 256 tiles. With 8 x 8 tiles and a resolution of 640 x 480, the tile resolution of the screen is 80 x 60, for a total of 4,800 tiles being stored in the tilemap. The next power of two number of tiles across each axis from 80 x 60 is 128 x 64, so the tilemap, and thus the tile map cache, stores 8192 tiles. Since the tiles in the tile map are in row order, the tiles corresponding to off screen coordinates are not displayed. Figure 5 shows how the tilemap is layed out. The tiles on the far right and bottom are not displayed on the screen.

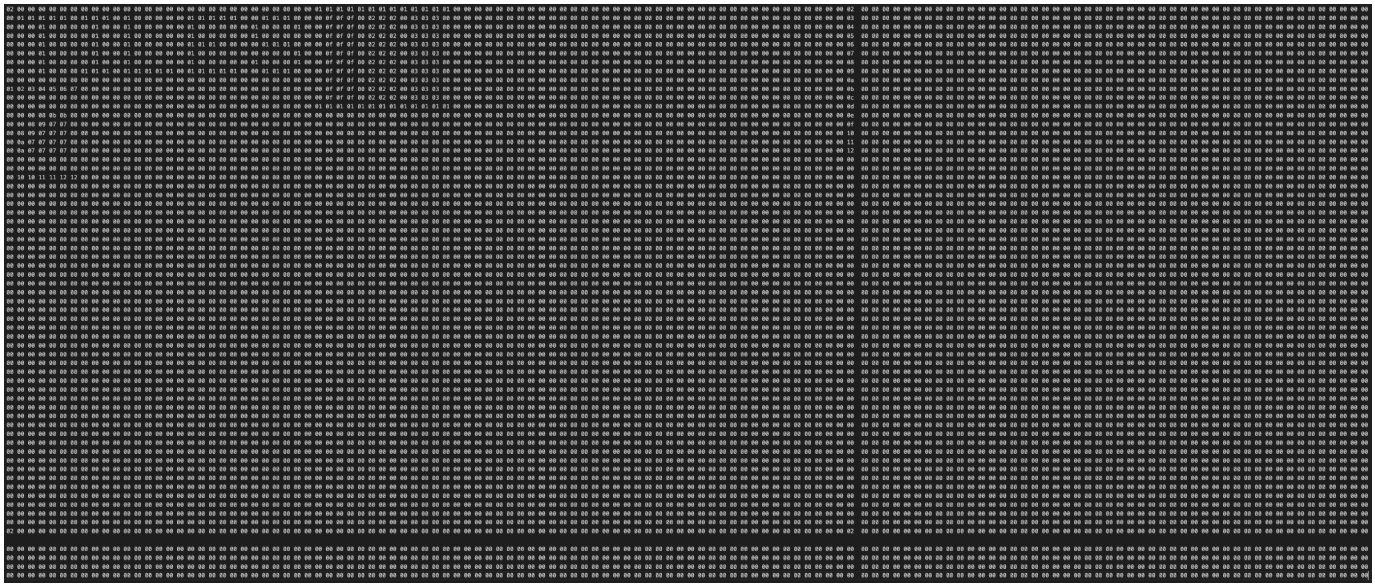


Figure 5: Tilemap Regions [2]

Each tile is 32 bytes, and each tile ID is 8-bits, since we have 256 tiles. Thus, the tileset is 256 tiles x 32 bytes = 8KiB. The tilemap and tilemap cache store 8192 tiles each with an 8-bit tile ID for a size of 8192 x 1 byte = 8KiB. The palette memory stores 16 24-bit colors for a total size of 48B. Because each memory stores different data: the tilemap needs stores the tile IDs, the tileset stores the 4-bit color codes, and the palette memory stores the 24-bit RGB value, each memory is a different width. The details for each memory are shown in Table 3. Note that the tilemap cache is not accessible to the CPU.

Table 3: Memory Details

Memory	**Size**	**Data Width**	**Number of Addresses**
Tileset	8KiB	4 bits	16,384
Tilemap	8KiB	8 bits	8,192
Tilemap Cache	8KiB	8 bits	8,192
Palette	24B	24 bits	16
Total	24.624KB	-	-

The total memory used of 24.624KB is a fraction of the available M10K memory on the FPGA which is arround 480KB. Since M10K blocks can be dual-ported each memory is configured as dual-port so that the CPU can get one port and the video generator can get the other port.

Because of the dual-buffering setup, the dual-porting of the tilemap and tilemap cache are used slightly differently to how it is used for the other memories. For the tilemap, one port goes to the CPU and the other goes to the tilemap cache (data) and buffer controller (address and write control). For the

tilemap cache, one port goes to the tilemap memory (data) and buffer controller (address and write control) and the other goes to the video generator.

The hardware also implements pipelining to account for the one cycle delay between sending an address to memory and getting the corresponding data back. Because of this one cycle delay, the hcount, VGA_BLANK_n, and VGA_HS signals, would be one cycle ahead of the current pixel data, leading to graphical issues, To solve this, the signals are pipelined to introduce a one cycle delay to keep them in sync with the current pixel the color data is being calculated for. Ideally, vcount and VGA_VS would be pipelined as well to account for this one cycle delay as well, but because they signals only change once per scan line, the delay is not as relevant. The provided tile hardware [2] was functional without pipelining for these two signals so we saw no need to modify the pipelining behavior or add pipelining to these two signals.[1]

3.2.2 Hardware-Software Interface

No driver is required for the hardware; instead, the tile map, color palette, and tileset memories are directly mapped to memory addresses which the CPU can read and write to. Table 4 shows the memory map and corresponding addresses used, while Figure 6 shows the addressing scheme. The addresses were derived from the example U-Boot code given in the tiles tutorial [2]; it was found that the same addresses work from within Linux.

Table 4: Memory Map

Region	**Base Address**	**Address Range**
Tile Map	0xFF200000	0xFF200000 - 0xFF201FFF
Palette	0xFF202000	0xFF202000 - 0xFF20203F
Tileset	0xFF204000	0xFF204000 - 0xFF207FFF

a_{14}	a_{13}	a_{12}	a_{11}	a_{10}	a_9	a_8	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0	
0	0	m_{12}	m_{11}	m_{10}	m_9	m_8	m_7	m_6	m_5	m_4	m_3	m_2	m_1	m_0	Tilemap
0	1	0	0	0	0	0	0	0	p_3	p_2	p_1	p_0	b_1	b_0	Palette
1	s_{13}	s_{12}	s_{11}	s_{10}	s_9	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0	Tileset

Figure 6: Address Encoding Scheme (Courtesy of Stephen Edwards [2])

To communicate with the tiles hardware from within Linux, we simply write or read from the addresses listed in Table 4. For the tilemap, indexing it is made easier by the fact that bits [13:7] of the address correspond to the row while bits [6:0] corresponding to the column.

[1] For more details on how tile graphics hardware we used and modified for our project was developed, please see Professor Edwards' writeup [1].

3.2.3 Final Hardware Resource Utilization

Fitter Summary	
Fitter Status	Successful - Sun May 11 13:31:59 2025
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	soc_system
Top-level Entity Name	soc_system_top
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	416 / 32,070 (1 %)
Total registers	622
Total pins	362 / 457 (79 %)
Total virtual pins	0
Total block memory bits	196,992 / 4,065,280 (5 %)
Total RAM Blocks	26 / 397 (7 %)
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	1 / 4 (25 %)

Figure 7: FPGA Resource Utilization

Figure 7 shows the FPGA resource utilization for the final implementation of the design. We have plenty of resources to spare, with the most used resource being the pins.

4. Software

4.1 USB SNES Controller

Our game is controlled using the Kiwitata USB SNES style controller shown in Figure 8.

Figure 8: USB SNES Controller [8]

To read the controller, the Linux kernel's built-in HID driver parses the device descriptors and creates an input node under /dev/input/event*. To open the controller for reading, our code calls open_controller() which iterates through event0–event31, using the EVIOCGNAME and EVIOCGID ioctls to match the SNES pad before opening it in non-blocking mode so that our game loop can poll it without blocking inputs.

Controller Input	**Tetris Method**	**Description**
D-pad Left	move_left()	Shift the active tetromino one column left
D-pad Right	move_right()	Shift the active tetromino one column right
D-pad Down, A, Y	soft_drop()	Advance the tetromino one row downward

B	hard_drop()	Instantly lock the tetromino at bottom
L, R, X	rotate()	Rotate the tetromino 90° clockwise
Select	toggle_pause()	Pause / resume the game
Start	spawn() / reset()	Begin play from START/OVER

During each 60 Hz cycle, our poll_input() routine drains all pending HID reports from the controller's non blocking file descriptor and immediately translates them into game actions. Left and right D-pad movements slide the current tetromino one column via move_left() or move_right(), while pressing down on the pad—or tapping either the A or Y button—invokes soft_drop(), advancing the piece a single row. The three shoulder inputs (L, R) and X all call rotate(), causing a quarter‑turn clockwise spin about the piece's pivot; tapping B executes a hard_drop(), locking the tetromino at its lowest valid position. Meanwhile, Select toggles the pause flag—freezing both gravity and input handling—and pressing Start in either the START or OVER state clears the screen and transitions into PLAY (or resets a completed game).

4.2. Game Logic

4.2.1 Overall Game FSM

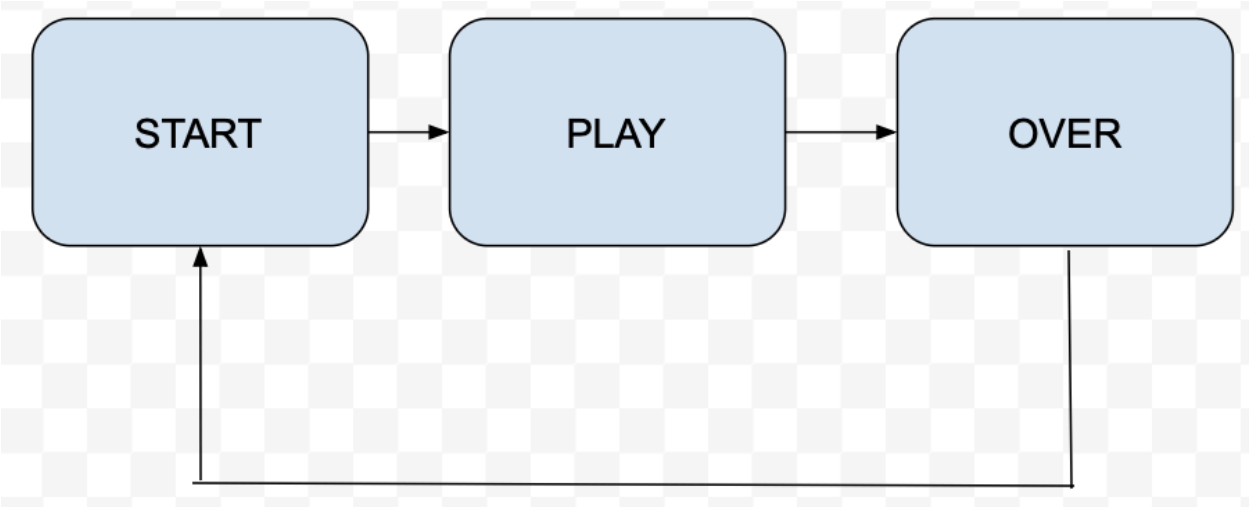


Figure 9: Overall Game State FSM

The overall game FSM, shown in Figure 9, has three states: START, PLAY and OVER. The START state is when the game starts up and the press start to play screen appears. This transitions into initializing and displaying the playscren and the first piece spawns at the center of the playfield when it enters the PLAY state where the game logic is run. A game over then trigger the OVER condition and upon pressing the start button to restart, we return to the START condition.

4.2.2 Start Screen

Upon first booting up the game, the start-screen, shown in Figure 10, is displayed. The main loop enters the START state and waits for the start button to be pressed. Once the start button is pressed, tilemap is cleared, and the FSM state is changed from START to PLAY, handing control over to the active game loop.

Figure 10: Tetris Start Screen

4.2.3 Gameplay Logic

In the PLAY state the Tetris code executes a deterministic, 60 Hz loop that integrates piece generation, user input, automatic descent, collision resolution, row clearing, level increasing, speed increasing, and end-of-game handling.

Each iteration of the loop advances the game by precisely one frame, ensuring that player actions and gravity-driven events are applied in lockstep with rendering. The general logic loop is shown in Figure 11.

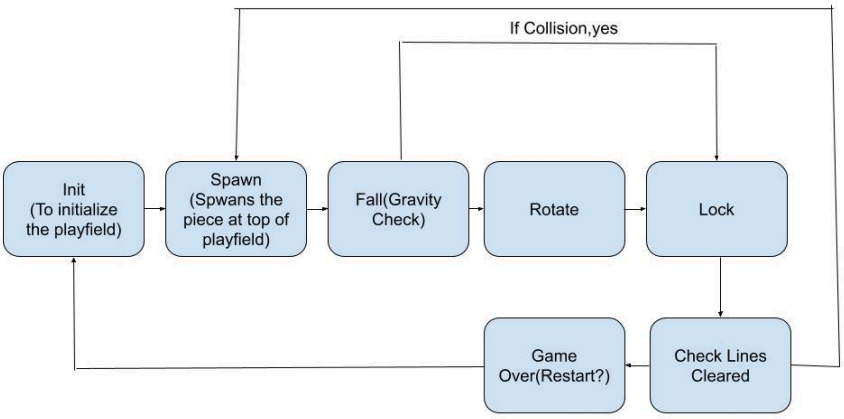


Figure 11: General Diagram of Gameplay Logic

Key stages of the loop include:

- spawn()
  Initializes a new tetromino by moving the next-piece buffer into the active piece (cur = nxt), generating a fresh random preview (nxt = rnd_piece()), and resetting position and orientation (px = 5, py = 0, rot = 0). An immediate collision at this location—checked via collision(px, py, cur, rot)—flags the game as over.

- poll_input()
  Reads all pending struct input_event records from the nonblocking controller descriptor. D-pad left/right (ABS_HAT0X 0/255) invoke move_left()/move_right(), D-pad down (ABS_HAT0Y 255) and buttons A/Y (codes 289/291) call soft_drop(), shoulder buttons L/R/X (292/293/288) trigger rotate(), B (290) executes hard_drop(), and Select (296) toggles pause.

- Gravity Tick (step)
  Increments an internal tick counter and, when tick % max(1, 30/level) == 0, attempts an automatic descent. If collision(px, py + 1, cur, rot) is false, the piece moves down; otherwise, lock_piece() is invoked.

- lock_piece() & clear_lines()
  Embeds the rotated tetromino mask into the 20×10 playfield array, then scans for full rows in clear_lines(). Each complete line is removed by shifting all above rows downward and inserting an empty row at the top. The routine increments lines_cleared, recalculates level = 1 + (lines_cleared / 10), and updates score_val according to the 100–300–500–800 table.

- Chain spawn()
  After clearing lines, spawn() is called again to introduce the next piece or, if the spawn location is obstructed, to set the game-over flag.

- draw_ghost()
  Ghost Piece Projection runs before each frame's rendering: simulates a hard drop without modifying game state, locating the lowest valid position and overlaying the tetromino in the designated ghost tile (Tile 13), thereby giving the player a preview of its landing spot.

- Next-Piece Preview is rendered each frame via draw_next(), which displays the upcoming tetromino in a reserved tilemap region, granting players foresight into subsequent pieces and enabling deeper tactical play.

- Game-Over & Reset
  When over is true, the loop transitions to the OVER state and displays the final score. A Start-button press (EV_KEY code 297) then invokes reset(), which clears the playfield, zeros score and lines, resets level and tick counters, and returns to spawn() to begin a new game.

- Level Progression ties difficulty to performance: each ten lines cleared increments the level by one, shortening the gravity interval via interval = max(1, 30/level) and accelerating automatic drops to increase challenge.

By organizing these stages into a single frame-driven cycle and leveraging the Super-Rotation System, nonblocking I/O, and efficient tilemap updates, the implementation delivers precise, low-latency gameplay that reproduces classic Tetris mechanics on the DE1-SoC. The core game logic is loosely based on an open-source C++ Tetris implementation [3].

Figure 12: Tetris Gameplay

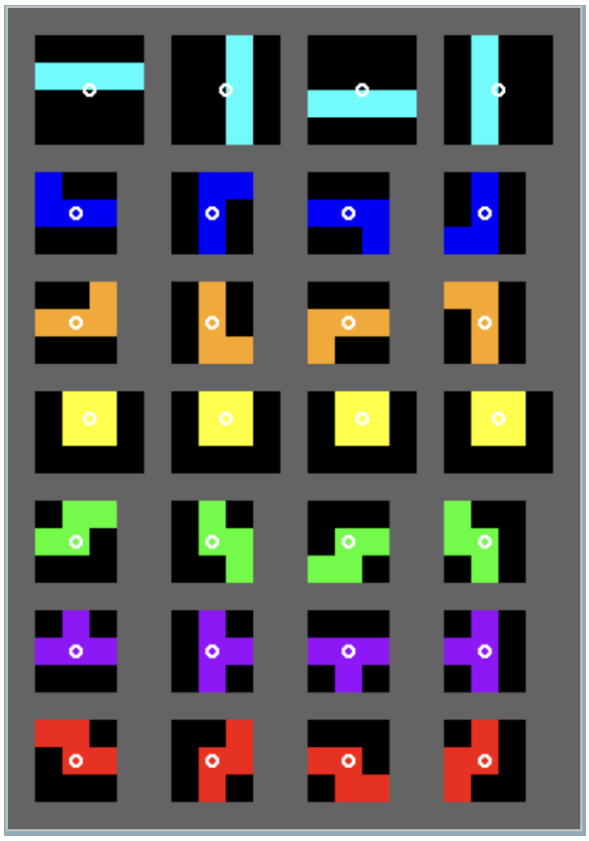
Figure 12 shows the Tetris gameplay screen. Here you can see all the various aspects of the game, including the playfield, next piece, level, score, lines, current block, and ghost block.
Scoring follows the familiar non-linear scheme to reward multi-line clears: 100 points for a single line, 300 for a double, 500 for a triple, and 800 for a four-line "Tetris," encouraging strategic stacking and line-clear planning.

4.2.4 SRS Rotation System

In our implementation, each tetromino adheres to the Super‑Rotation System (SRS), cycling through four discrete states—0°, 90°, 180°, and 270°—with every clockwise turn simply computed as (state + 1) % 4. This rotation always occurs around a well‑defined pivot within the 4×4 mask, ensuring that pieces maintain consistent geometry as they spin. By embedding this logic directly in the Tetris::rotate() method, we guarantee that each button press advances the orientation predictably, without ambiguity in how the block will appear on the playfield.

However, a naive rotation can collide with walls or existing blocks, so SRS supplements the raw turn with a series of "wall‑kick" trials: when a rotation initially overlaps the playfield, the system tests up to five small positional offsets to find a valid placement before abandoning the rotation. J, L, S, T, and Z pieces use the JLSTZ kick table—(0,0), (–1,0), (–1,+1), (0,–2), and (–1,–2)—while the I-tetromino applies its own wider offsets of (0,0), (–2,0), (+1,0), (–2,–1), and (+1,+2) to accommodate its elongated shape. The square (O) piece, thanks to its perfect symmetry, simply rotates in place without any kicks. Importantly, SRS accepts the first offset that resolves the collision and cancels the turn only if all five

attempts fail, yielding the familiar, forgiving rotation behavior that players expect from modern Tetris. Our SRS rotation is based on the table below, however only for clockwise rotation as we don't have anti-clockwise rotation.



J, L, S, T, Z Tetromino Wall Kick Data

	Test 1	Test 2	Test 3	Test 4	Test 5
0->R	(0, 0)	(−1, 0)	(−1,+1)	(0,−2)	(−1,−2)
R->0	(0, 0)	(+1, 0)	(+1,−1)	(0,+2)	(+1,+2)
R->2	(0, 0)	(+1, 0)	(+1,−1)	(0,+2)	(+1,+2)
2->R	(0, 0)	(−1, 0)	(−1,+1)	(0,−2)	(−1,−2)
2->L	(0, 0)	(+1, 0)	(+1,+1)	(0,−2)	(+1,−2)
L->2	(0, 0)	(−1, 0)	(−1,−1)	(0,+2)	(−1,+2)
L->0	(0, 0)	(−1, 0)	(−1,−1)	(0,+2)	(−1,+2)
0->L	(0, 0)	(+1, 0)	(+1,+1)	(0,−2)	(+1,−2)

I Tetromino Wall Kick Data

	Test 1	Test 2	Test 3	Test 4	Test 5
0->R	(0, 0)	(−2, 0)	(+1, 0)	(−2,−1)	(+1,+2)
R->0	(0, 0)	(+2, 0)	(−1, 0)	(+2,+1)	(−1,−2)
R->2	(0, 0)	(−1, 0)	(+2, 0)	(−1,+2)	(+2,−1)
2->R	(0, 0)	(+1, 0)	(−2, 0)	(+1,−2)	(−2,+1)
2->L	(0, 0)	(+2, 0)	(−1, 0)	(+2,+1)	(−1,−2)
L->2	(0, 0)	(−2, 0)	(+1, 0)	(−2,−1)	(+1,+2)
L->0	(0, 0)	(+1, 0)	(−2, 0)	(+1,−2)	(−2,+1)
0->L	(0, 0)	(−1, 0)	(+2, 0)	(−1,+2)	(+2,−1)

Figure 13: SRS Rotation System (Courtesy of Harddrop Wiki [4])

4.2.5 Game Over Screen

When the playfield can no longer accommodate a new tetromino, the game enters the OVER state and invokes show_game_over(). This routine begins by wiping the tilemap clean with memset(TM, 0), removing any remnants of the previous game. It then displays "GAME OVER" as shown in the figure below at (10, 10), followed by the final score and total lines cleared at (10, 20) and (10, 30), respectively. To invite another attempt, it draws "START: RESTART" at (10, 40), matching the report's end‑screen illustration. The main loop continues at 60 Hz but deliberately ignores all inputs until it sees the Start button (EV_KEY code 297). When Start is pressed, reset() reinitializes the internal playfield array, zeros the score and line counters, and restores the level and tick counters to their defaults before calling spawn() to introduce the first tetromino of the next game.

Figure 14: Tetris Game Over Screen

4.3. Display Logic

The rendering subsystem begins by memory-mapping the three FPGA regions—tilemap (TM), palette (PA), and tileset (TS)—via map_fpga(), then populating PA and TS in load_assets(). Once the tile graphics and 16-color palette are resident in VRAM, the game uses a small set of draw primitives (put(), rect(), frame(), draw_char(), and draw_string()) to update the scene each 60 Hz frame. In practice, each function writes tile indices into the 8 KB TM region: draw_borders() frames the playfield and next-piece box, draw_playfield() and draw_piece() paint locked and active tetrominos, draw_ghost() overlays a faded preview, draw_next() renders the upcoming piece, and draw_hud() blits score, lines, and level using text tiles. Clearing and re-drawing only changed regions (e.g. individual digits) minimizes memory writes, ensuring smooth animation on the DE1-SoC's hardware tile engine.

The complete tileset is illustrated in Figure 15. Ten distinct tiles are used for all graphics and text:

- **Tile 0:** Background (empty playfield)
- **Tile 1:** Walls, playfield border, and next-piece bounding box

- **Tiles 2–7:** Solid blocks for the seven tetromino types (I, J, L, O, S, T, Z)
- **Tile 13:** Ghost-piece overlay and all alphanumeric characters rendered by draw_string()

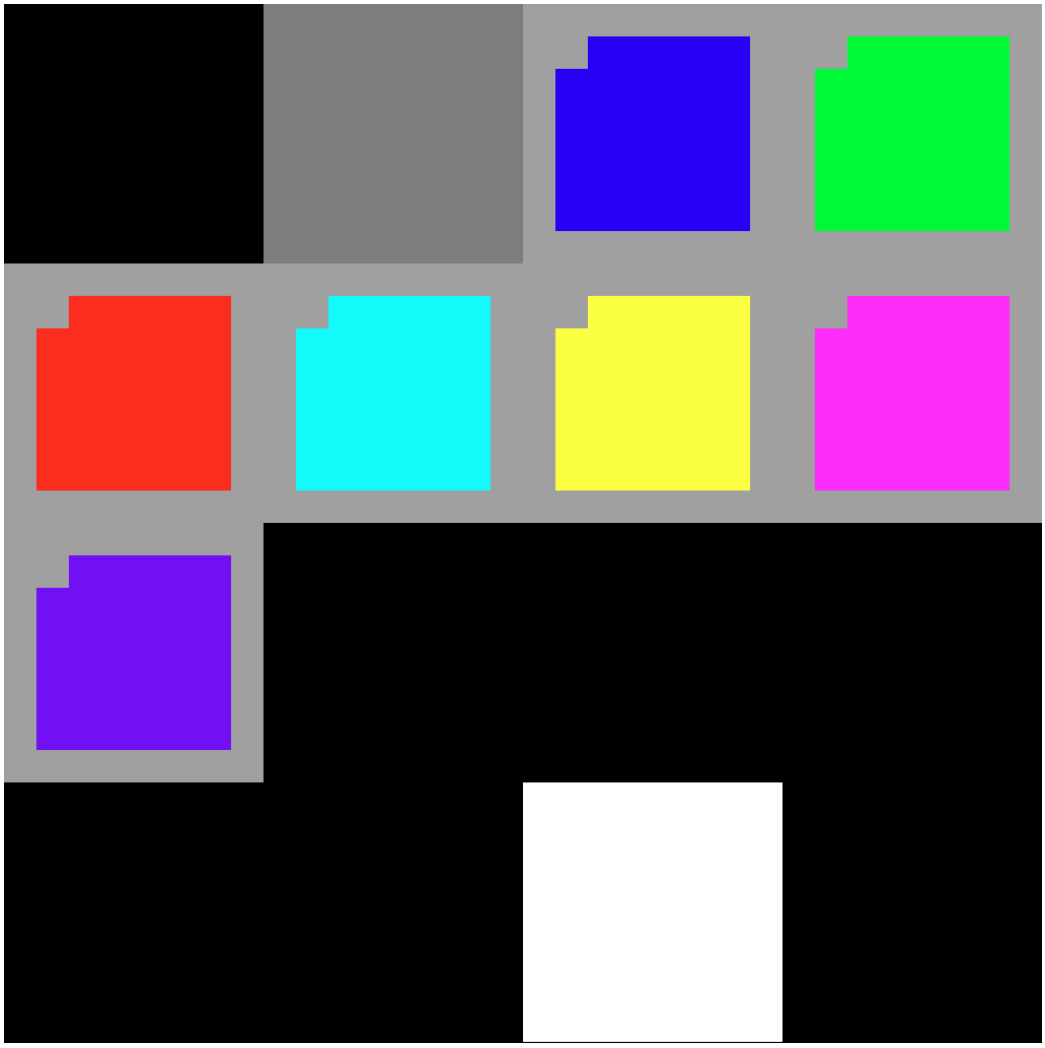


Figure 15: Tetris Tileset

4.4. Dual Threads and Audio

We separate the audio playback and the gameplay logic using two threads. Thread 1 is used for the game logic which runs the game loop and handles the inputs.screen rendering and thread 2 is used to read and decode the mp3 audio file we provide in our case we are using tetris.mp3 which is an mp3 file from Internet Archive [9]. The file Runner.cpp creates two threads using POSIX threads which runs both the threads in parallel such that the audio plays while we are playing the game, one thread runs the audio executable while the other thread runs the tetris executable.

5. USB Audio Implementation

We initially explored FPGA‑based audio cores but encountered persistent driver and timing conflicts under the SoCFPGA Linux environment, so we elected to use a class-compliant USB audio approach instead. Figure 16 shows the USB DAC used.

Figure 16: Griffen iMic USB Audio DAC [10]

SInce the provided kernel did not have usb audio support, we had to build the snd-usb audio kernel module ourselves. After cloning the linux-socfpga v4.19 repository and building with socfpga_defconfig, we enabled the snd-usb-audio module via make menuconfig (under Sound → USB) and loaded it with modprobe snd_usb_audio. As a result, any connected USB DAC now appears as an ALSA sound card at boot.

In user space, libmpg123 decodes our Tetris.mp3 file into raw PCM samples, which libao then feeds into ALSA's PCM interface; ALSA buffers these frames and dispatches them over isochronous USB transfers through the snd-usb-audio driver to the external DAC, which converts them to analog and drives the speakers. We also had to edit the dtb to force the ports into host mode. Finally, audio playback runs in its own POSIX thread, decoupling it from the main game loop to ensure seamless, uninterrupted music alongside the gameplay. Figure 17 shows the USB audio flow.

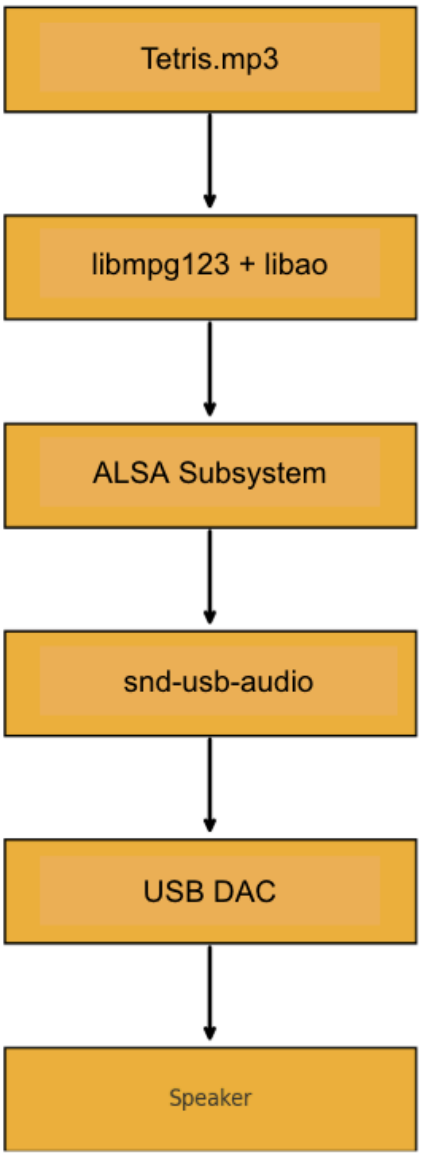


Figure 17: USB Audio Flow

6. Contributions and Lessons Learned

6.1. Lessons Learned

We learned to start a project with a minimum viable product(MVP) where we started with a sprite-based approach when we should have started with just the tile-based approach required by tetris. While building the testbenches for the initial sprite-based approach we realised that due to time constraints we would not be able to test this custom hardware for all cases and complete the game which motivated us to pivot to a simpler approach which is convenient for a game like tetris which uses tiles and we decided to make the gameplay smoother by adding double buffering to the tile-based approach which is easier to test.

We also learnt the importance of using open-source hardware and software and also the importance of hardware- software integration to get a working product, which is a game in this case, if we are limited by time.

6.2. Contributions

- Michael Lippe: Developed the initial design for the sprite hardware. Implemented the sprite PPU in system verilog. Modified the provided tiles hardware to add double buffering. Synthesized the hardware to run on the FPGA. Got hardware software interface working to edit the tiles memory from within Linux. Found and configured correct Linux Kernel with USB Audio support. Decompiled and recompiled DTB to force USB ports on the DE1-SoC to operate in host mode. Got USB Audio working with USB DAC and speaker. Designed the tileset artwork. Implemented the ghost piece and level system. Sourced font.
- Garvit Vyas: Developed TestBenches for the initial sprite based hardware design to verify functionality and debug DUT on Modelsim with waveforms. Worked on finding the correct USB Audio Linux Kernel to get USB audio to work. Tried testing other hardware based Audio open source implementations for DE1-SoC before moving forward with USB audio. Implemented various gameplay functions for the software game logic.
- Bhargav Sriram: Worked on the software game logic to implement various functions. Worked on finding the correct Linux Kernel build for the USB Audio+getting it to work . Helped develop testbenches of the initial sprite based hardware based design for verifying functionality

7. References

1. Edwards, S. VGA tile graphics on an FPGA: A tutorial. https://www.cs.columbia.edu/~sedwards/classes/2025/4840-spring/tiles.pdf
2. Edwards, S. Source files for VGA TIle Graphics on an FPGA https://www.cs.columbia.edu/~sedwards/classes/2025/4840-spring/tiles.tar.gz
3. Milon. *Milon/Tetris: My First Project Using C++*. GitHub. https://github.com/milon/Tetris
4. *SRS*. Hard Drop Tetris Wiki. https://harddrop.com/wiki/SRS
5. Altera-Fpga. *Altera-FPGA/Linux-socfpga at v4.19*. GitHub. https://github.com/altera-fpga/linux-socfpga/tree/v4.19
6. Ameba8195. *Arduino/hardware_v2/cores/arduino/font5x7.h at master · AMEBA8195/Arduino*. GitHub. https://github.com/Ameba8195/Arduino/blob/master/hardware_v2/cores/arduino/font5x7.h
7. Wikimedia Foundation. (2025, April 27). *Tetromino*. Wikipedia. https://en.wikipedia.org/wiki/Tetromino
8. Amazon.com: Kiwitata 2X Classic SNES USB controller for Retro Gamings, SNES wired USB Joypad Game Controller for windows PC mac raspberry pi : Video games. https://www.amazon.com/Classic-Controller-kiwitat%C3%A1-Joystick-Raspberry/dp/B01JYGYAX8
9. *Tetris theme music : Free Download, borrow, and streaming*. Internet Archive. https://archive.org/details/TetrisThemeMusic
10. Amazon.com: Griffin Technology IMIC - the original USB stereo input and Output Audio Adapter : Griffin Technology: Electronics. https://www.amazon.com/Griffin-Technology-iMic-original-Adapter/dp/B003Y5D776

8. Code

8.1. Sprite Hardware (Not Using)

8.1.1 address_decoder.sv

```
// hw/address_decoder.sv
`timescale 1ns/1ps

module addr_decode(
   addr,
   write_data,
   chip_select,
   write,
   rw_tile_buffer,
   rw_tile_graphics,
   rw_sprite_graphics,
   rw_color_palettes,
   rw_OAM,
   write_data_tile_buffer,
   write_data_tile_graphics,
   write_data_sprite_graphics,
   write_data_OAM,
   write_data_color_palettes,
   addr_tile_buffer,
   addr_tile_graphics,
   addr_sprite_graphics,
   addr_color_palettes,
   addr_OAM
);
 // port declarations
 input  logic [12:0] addr;
 input  logic [31:0] write_data;
 input  logic        chip_select;
 input  logic        write;

 output logic        rw_tile_buffer;
 output logic        rw_tile_graphics;
 output logic        rw_sprite_graphics;
```

```
output logic         rw_color_palettes;
output logic         rw_OAM;

output logic [31:0] write_data_tile_buffer;
output logic [31:0] write_data_tile_graphics;
output logic [31:0] write_data_sprite_graphics;
output logic [31:0] write_data_OAM;
output logic [23:0] write_data_color_palettes;

output logic [ 8:0] addr_tile_buffer;
output logic [10:0] addr_tile_graphics;
output logic [10:0] addr_sprite_graphics;
output logic [ 2:0] addr_color_palettes;
output logic [ 7:0] addr_OAM;

always @(*) begin
  // defaults
  rw_tile_buffer          = 1'b0;
  rw_tile_graphics        = 1'b0;
  rw_sprite_graphics      = 1'b0;
  rw_color_palettes       = 1'b0;
  rw_OAM                  = 1'b0;

  write_data_tile_buffer       = 32'h0;
  write_data_tile_graphics     = 32'h0;
  write_data_sprite_graphics   = 32'h0;
  write_data_OAM               = 32'h0;
  write_data_color_palettes    = 24'h0;

  addr_tile_buffer         = 9'h0;
  addr_tile_graphics       = 11'h0;
  addr_sprite_graphics     = 11'h0;
  addr_color_palettes      = 3'h0;
  addr_OAM                 = 8'h0;

  // decode top 3 bits (addr[12:10])
  casez (addr[12:10])
    3'b00z: begin  // Tile Buffer (00x)
      if (chip_select && write) begin
        rw_tile_buffer        = 1'b1;
        addr_tile_buffer      = addr[8:0];
        write_data_tile_buffer= write_data;
```

```
      end
    end

    3'b01z: begin  // Tile Graphics (01x)
      if (chip_select && write) begin
        rw_tile_graphics         = 1'b1;
        addr_tile_graphics       = addr[10:0];
        write_data_tile_graphics = write_data;
      end
    end

    3'b10z: begin  // Sprite Graphics (10x)
      if (chip_select && write) begin
        rw_sprite_graphics         = 1'b1;
        addr_sprite_graphics       = addr[10:0];
        write_data_sprite_graphics = write_data;
      end
    end

    3'b110: begin // Color Palettes
      if (chip_select && write) begin
        rw_color_palettes         = 1'b1;
        addr_color_palettes       = addr[2:0];
        write_data_color_palettes = write_data[23:0];
      end
    end

    3'b111: begin // OAM
      if (chip_select && write) begin
        rw_OAM        = 1'b1;
        addr_OAM      = addr[7:0];
        write_data_OAM= write_data;
      end
    end
  endcase
 end
endmodule
```

8.1.2 memories.sv

```
module tile_buffer(
    input logic clk, rw_1, rw_2,
    input logic [31:0] write_data_1, write_data_2,
    input logic [8:0] addr_1, addr_2,
    output logic [31:0] read_data_1, read_data_2
);

    logic [31:0] tile_buffer_array [299:0];

    always @(posedge clk) begin
        if (rw_1) tile_buffer_array[addr_1] <= write_data_1;
        else read_data_1 <= tile_buffer_array[addr_1];
        if(rw_2) tile_buffer_array[addr_2] <= write_data_2;
        else read_data_2 <= tile_buffer_array[addr_2];
    end

endmodule

module tile_graphics(
    input logic clk, rw_1, rw_2,
    input logic [31:0] write_data_1, write_data_2,
    input [10:0] addr_1, addr_2,
    output logic [31:0] read_data_1, read_data_2
);

    logic [31:0] tile_graphics_array [2047:0];

    always @(posedge clk) begin
        if (rw_1) tile_graphics_array[addr_1] <= write_data_1;
        else read_data_1 <= tile_graphics_array[addr_1];
        if(rw_2) tile_graphics_array[addr_2] <= write_data_2;
        else read_data_2 <= tile_graphics_array[addr_2];
    end

endmodule

module sprite_graphics(
    input logic clk, rw_1, rw_2,
    input logic [31:0] write_data_1, write_data_2,
    input [10:0] addr_1, addr_2,
    output logic [31:0] read_data_1, read_data_2
```

```
);

    logic [31:0] sprite_graphics_array [2047:0];

    always @(posedge clk) begin
        if (rw_1) sprite_graphics_array[addr_1] <= write_data_1;
        else read_data_1 <= sprite_graphics_array[addr_1];
        if(rw_2) sprite_graphics_array[addr_2] <= write_data_2;
        else read_data_2 <= sprite_graphics_array[addr_2];
    end

endmodule

module color_palettes(
    input logic clk, rw_1, rw_2,
    input logic [23:0] write_data_1, write_data_2,
    input [2:0] addr_1, addr_2,
    output logic [23:0] read_data_1, read_data_2
);

    logic [23:0] color_palette_array [7:0];

    always @(posedge clk) begin
        if (rw_1) color_palette_array[addr_1] <= write_data_1;
        else read_data_1 <= color_palette_array[addr_1];
        if(rw_2) color_palette_array[addr_2] <= write_data_2;
        else read_data_2 <= color_palette_array[addr_2];
    end

endmodule

module OAM (
    input logic clk, rw_1, rw_2,
    input logic [31:0] write_data_1, write_data_2,
    input [7:0] addr_1, addr_2,
    output logic [31:0] read_data_1, read_data_2
);

    logic [31:0] OAM_array [255:0];

    always @(posedge clk) begin
        if (rw_1) OAM_array[addr_1] <= write_data_1;
```

```
        else read_data_1 <= OAM_array[addr_1];
        if(rw_2) OAM_array[addr_2] <= write_data_2;
        else read_data_2 <= OAM_array[addr_2];
    end

endmodule
```

8.1.3 ppu_asm.sv

```
module PPU_asm(
    input logic clk, //Clock
    input logic reset, //Active High Reset

    //VGA related IO
    input logic [10:0] hcount, //VGA hcount from VGA Controller
    input logic [9:0] vcount, //VGA vcount from VGA Controller
    input logic vblank, //VGA vblank from VGA Controller
    input logic hsync, //VGA hsync from VGA Controller
    output logic [23:0] pixel_color, //Pixel Color to Send to VGA Controller

    //RW Signals to Memories
    output logic rw_tile_buffer,
    output logic rw_tile_graphics,
    output logic rw_sprite_graphics,
    output logic rw_color_palettes,
    output logic rw_OAM,

    //Write Data to Memories
    output logic [31:0] write_data_tile_buffer,
    output logic [31:0] write_data_tile_graphics,
    output logic [31:0] write_data_sprite_graphics,
    output logic [31:0] write_data_OAM,
    output logic [23:0] write_data_color_palettes,

    //Address Signals to Memories
    output logic [8:0] addr_tile_buffer,
    output logic [10:0] addr_tile_graphics,
    output logic [10:0] addr_sprite_graphics,
    output logic [2:0] addr_color_palettes,
    output logic [7:0] addr_OAM,
```

```
    //Read Data from Memories
    input logic [31:0] read_data_tile_buffer,
    input logic [31:0] read_data_tile_graphics,
    input logic [31:0] read_data_sprite_graphics,
    input logic [31:0] read_data_OAM,
    input logic [23:0] read_data_color_palettes,

    //Shift Register Signals
    output logic [31:0] shift_load_data [8:0],
    output logic [8:0] shift_enable,
    output logic shift_load_sprite,
    output logic shift_load_background,

    //Priority Encoder Signals
    output logic  priority_palette_data_out[8:0],
    input logic [1:0] priority_pixel_data_in,
    input logic priority_palette_data_in
);

    //Buffers Updated During Vblank
    logic [23:0] color_palette_buffer [8:0];
    logic [15:0] sprite_x_buffer [127:0];
    logic [15:0] sprite_y_buffer [127:0];
    logic sprite_palette_buffer [127:0];
    logic [6:0] sprite_tile_id_buffer [127:0];
    logic [1:0] sprite_rotation_buffer [127:0];

    //Buffers Updated During Hsync
    logic [31:0] background_line_graphics_buffer [39:0];
    logic [39:0] background_line_palette_buffer;
    logic [31:0] sprite_graphics_buffer [7:0];
    logic [6:0] sprites_on_line [7:0];
    logic sprites_on_line_palettes[7:0];

    //Vblank memory access trackers
    logic [7:0] cords_sprite_load;
    logic [7:0] palette_sprite_load;
    logic [3:0] palette_ram_pointer;

    //Hsync memory access trackers
    logic [5:0] background_line_pointer;
    logic [2:0] sprite_graphics_pointer;
```

```
logic [7:0] sprites_on_line_pointer;
logic [3:0] shift_register_load_pointer;
logic [3:0] sprites_found;

always @(posedge clk) begin

    //reset
    if (reset) begin
        rw_tile_buffer <= 0;
        rw_tile_graphics <= 0;
        rw_sprite_graphics <= 0;
        rw_color_palettes <= 0;
        rw_OAM <= 0;
        cords_sprite_load <= 0;
        palette_sprite_load <= 0;
        palette_ram_pointer <= 0;
        background_line_pointer <= 0;
        sprite_graphics_pointer <= 0;
        sprites_on_line_pointer <= 0;
        sprites_found <= 0;
        shift_register_load_pointer <= 0;
        shift_enable <= 0;
        shift_load_background <= 0;
        shift_load_sprite <= 0;
        background_line_palette_buffer <= 40'b0;
        for (int i = 0; i < 128; i = i + 1) begin
            sprite_x_buffer[i] <= 0;
            sprite_y_buffer[i] <= 0;
            sprite_tile_id_buffer[i] <= 0;
            sprite_rotation_buffer[i] <= 0;
    sprite_palette_buffer[i] <= 0;
        end
        for (int i = 0; i < 8; i += 1) begin
            sprite_graphics_buffer[i] <= 0;
            sprites_on_line[i] <= 0;
            shift_load_data[i] <= 0;
            sprites_on_line_palettes[i] <= 0;
        end
        for (int i = 0; i < 40; i += 1) begin
            background_line_graphics_buffer[i] <= 0;
        end
        for (int i = 0; i < 9; i += 1) begin
```

```
                priority_palette_data_out[i] <= 0;
            end

        end

        /*
        Note that reading from any of the memories in memories.sv takes 2 cycles. On
cycle 1, we give the memory the address
        we want to read from. On cycle 2, we get the data at that address on the
memories respective read_data line. As such,
        you will notice that when filling the buffers from memory, there is a one cycle
offset between sending the address and
        filling the buffer, hence the at first confusing four tiered if statements. For
the first one, we only send the address
        since we have no data to load. For the second, we load the data for the
previous address and send the next address. For
        the third one we have no new addresses to send but we still need to received
and load the final data return into the buffer.
        For the fourth one, we are done with both addresses and reading data.
        */

        //set buffers that fill once per frame
        else if (vblank) begin

            //fill color_palette_buffer
            if (palette_ram_pointer == 0) begin
                rw_color_palettes <= 0; //Set color palette memory to read
                addr_color_palettes <= palette_ram_pointer;
                palette_ram_pointer <= palette_ram_pointer + 1;
            end

            else if (palette_ram_pointer < 8) begin
                rw_color_palettes <= 0; //Set color palette memory to read
                addr_color_palettes <= palette_ram_pointer;
                color_palette_buffer[palette_ram_pointer - 1] =
read_data_color_palettes;
                palette_ram_pointer <= palette_ram_pointer + 1;
            end

            else if (palette_ram_pointer == 8) begin
                rw_color_palettes <= 0; //Set color palette memory to read
                addr_color_palettes <= 0;
```

```
                color_palette_buffer[palette_ram_pointer - 1] =
read_data_color_palettes;
                palette_ram_pointer <= palette_ram_pointer + 1;
            end

            else begin
                rw_color_palettes <= 0; //Set color palette memory to read
                addr_color_palettes <= 0;
            end

            //fill sprite_x_buffer and sprite_y_buffer
            if (cords_sprite_load == 0) begin
                rw_OAM <= 0; //Set OAM memory to read
                addr_OAM <= cords_sprite_load * 2 + 1;
                cords_sprite_load <= cords_sprite_load + 1;
            end

            else if (cords_sprite_load < 128) begin
                rw_OAM <= 0; //Set OAM memory to read
                addr_OAM <= cords_sprite_load * 2 + 1;
                sprite_x_buffer[cords_sprite_load - 1] <= read_data_OAM[15:0];
                sprite_y_buffer[cords_sprite_load - 1] <= read_data_OAM[31:16];
                cords_sprite_load <= cords_sprite_load + 1;
            end

            else if (cords_sprite_load == 128) begin
                rw_OAM <= 0; //Set OAM memory to read
                addr_OAM <= palette_sprite_load;
                sprite_x_buffer[cords_sprite_load - 1] <= read_data_OAM[15:0];
                sprite_y_buffer[cords_sprite_load - 1] <= read_data_OAM[31:16];
                cords_sprite_load <= cords_sprite_load + 1;
                palette_sprite_load <= palette_sprite_load + 1;
            end

            else if (palette_sprite_load < 128) begin
                rw_OAM <= 0; //Set OAM memory to read
                addr_OAM <= palette_sprite_load * 2;
                sprite_palette_buffer[palette_sprite_load - 1] <= read_data_OAM[7];
                sprite_tile_id_buffer[palette_sprite_load - 1] <= read_data_OAM[6:0];
                sprite_rotation_buffer[palette_sprite_load - 1] <=
read_data_OAM[31:30];
                palette_sprite_load <= palette_sprite_load + 1;
```

```
            end

            else if (palette_sprite_load == 128) begin
                rw_OAM <= 0; //Set OAM memory to read
                addr_OAM <= 0;
                sprite_palette_buffer[palette_sprite_load - 1] <= read_data_OAM[7];
                sprite_tile_id_buffer[palette_sprite_load - 1] <= read_data_OAM[6:0];
                sprite_rotation_buffer[palette_sprite_load - 1] <=
read_data_OAM[31:30];
                palette_sprite_load <= palette_sprite_load + 1;
            end

            else begin
                rw_OAM <= 0; //Set OAM memory to read
                addr_OAM <= 0;
            end

        end

        //Set buffers that fill once per line
        else if (hsync) begin

            //Load background tiles into buffer

            if (background_line_pointer == 0) begin

                rw_tile_buffer <= 0; //Set tile buffer memory to read

                /*Calculate address into the tile buffer for tile at the start of the
current line.
                We do *10 and not *40 since each 32-bit entry of the tile-buffer holds
4 tile IDs */
                addr_tile_buffer <= vcount * 10;

                background_line_pointer <= background_line_pointer + 1;

            end

            else if (background_line_pointer == 1) begin

                rw_tile_buffer <= 0; //Set tile buffer memory to read
                rw_tile_graphics <= 0;
```

```
                /*Calculate address into the tile buffer for current tile being
processed.
                We do >> 2 since each 32-bit entry of the tile-buffer holds 4 tile IDs
*/
                addr_tile_buffer <= (vcount * 10) + (background_line_pointer >> 2);


                /* Calculate the address into the tile graphics memory for the current
line
                of the current tile being processed */
                case (background_line_pointer[1:0])

                    1: begin
                        addr_tile_graphics <= (read_data_tile_buffer[6:0] * 16) +
(background_line_pointer - 1);
                        background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[7];
                    end

                    2: begin
                        addr_tile_graphics <= (read_data_tile_buffer[14:8] * 16) +
(background_line_pointer - 1);
                        background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[15];

                    end

                    3: begin
                        addr_tile_graphics <= (read_data_tile_buffer[22:16] * 16) +
(background_line_pointer - 1);
                        background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[23];

                    end

                    0: begin
                        addr_tile_graphics <= (read_data_tile_buffer[30:24] * 16) +
(background_line_pointer - 1);
                        background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[31];
                    end
```

```
                endcase



                background_line_pointer <= background_line_pointer + 1;

            end

            else if (background_line_pointer < 40) begin

                rw_tile_buffer <= 0; //Set tile buffer memory to read
                rw_tile_graphics <= 0;

                /*Calculate address into the tile buffer for current tile being
processed.
                We do >> 2 since each 32-bit entry of the tile-buffer holds 4 tile IDs
*/
                addr_tile_buffer <= (vcount * 10) + (background_line_pointer >> 2);


                /* Calculate the address into the tile graphics memory for the current
line
                of the current tile being processed */
                case (background_line_pointer[1:0])

                    1: begin
                        addr_tile_graphics <= (read_data_tile_buffer[6:0] * 16) +
(background_line_pointer - 1);
                        background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[7];
                    end

                    2: begin
                        addr_tile_graphics <= (read_data_tile_buffer[14:8] * 16) +
(background_line_pointer - 1);
                        background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[15];

                    end

                    3: begin
```

```
                    addr_tile_graphics <= (read_data_tile_buffer[22:16] * 16) +
(background_line_pointer - 1);
                    background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[23];

                end

                0: begin
                    addr_tile_graphics <= (read_data_tile_buffer[30:24] * 16) +
(background_line_pointer - 1);
                    background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[31];
                end

            endcase

            background_line_graphics_buffer[(background_line_pointer - 1)] <=
read_data_tile_graphics;

            background_line_pointer <= background_line_pointer + 1;

        end

        else if (background_line_pointer == 40) begin

            rw_tile_buffer <= 0; //Set tile buffer memory to read
            rw_tile_graphics <= 0;

            addr_tile_buffer <= 0;

            /* Calculate the address into the tile graphics memory for the current
line
            of the current tile being processed */
            case (background_line_pointer[1:0])

                1: begin
                    addr_tile_graphics <= (read_data_tile_buffer[6:0] * 16) +
(background_line_pointer - 1);
                    background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[7];
                end
```

```
                    2: begin
                        addr_tile_graphics <= (read_data_tile_buffer[14:8] * 16) +
(background_line_pointer - 1);
                        background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[15];

                    end

                    3: begin
                        addr_tile_graphics <= (read_data_tile_buffer[22:16] * 16) +
(background_line_pointer - 1);
                        background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[23];

                    end

                    0: begin
                        addr_tile_graphics <= (read_data_tile_buffer[30:24] * 16) +
(background_line_pointer - 1);
                        background_line_palette_buffer[background_line_pointer - 1] <=
read_data_tile_buffer[31];
                    end

                endcase

                background_line_graphics_buffer[(background_line_pointer - 1)] <=
read_data_tile_graphics;

                background_line_pointer <= background_line_pointer + 1;

            end

            else if (background_line_pointer == 41) begin

                rw_tile_buffer <= 0; //Set tile buffer memory to read
                rw_tile_graphics <= 0;

                addr_tile_buffer <= 0;
                addr_tile_graphics <= 0;

                background_line_graphics_buffer[background_line_pointer - 1] <=
read_data_tile_graphics;
```

```
                background_line_pointer <= background_line_pointer + 1;

            end

            else begin
                rw_tile_buffer <= 0; //Set tile buffer memory to read
                rw_tile_graphics <= 0;

                addr_tile_buffer <= 0;
                addr_tile_graphics <= 0;
            end

            //Detect which sprites are on the line
            if (sprites_on_line_pointer < 128) begin

                //If current sprite is on the line and we have not filled all the
sprite slots
                if (sprites_found < 8 && (vcount >=
sprite_y_buffer[sprites_on_line_pointer]) && (vcount <
sprite_y_buffer[sprites_on_line_pointer] + 16)) begin
                    sprites_on_line[sprites_found] <= sprites_on_line_pointer;
                    sprites_on_line_palettes[sprites_found] <=
sprite_palette_buffer[sprites_on_line_pointer];
                    sprites_found <= sprites_found + 1;
                end

                sprites_on_line_pointer <= sprites_on_line_pointer + 1;
            end

            //Calculate pointers to sprite graphics based on rotation flags, what line
we are on, and the sprites' Y positions
            else if (shift_register_load_pointer == 0) begin
                rw_sprite_graphics <= 0;

                // If vertical flip bit is set
                if
(sprite_rotation_buffer[sprites_on_line[shift_register_load_pointer]][1])
addr_sprite_graphics <=
((sprite_tile_id_buffer[sprites_on_line[shift_register_load_pointer]]) * 16) + (15 -
(vcount - sprite_y_buffer[sprites_on_line[shift_register_load_pointer]]));
```

```
                // If vertical flip bit is not set
                else addr_sprite_graphics <=
((sprite_tile_id_buffer[sprites_on_line[shift_register_load_pointer]]) * 16) + (vcount
- sprite_y_buffer[sprites_on_line[shift_register_load_pointer]]);

                shift_register_load_pointer <= shift_register_load_pointer + 1;

            end

            else if (shift_register_load_pointer < 8) begin

                rw_sprite_graphics <= 0;

                // If vertical flip bit is set
                if
(sprite_rotation_buffer[sprites_on_line[shift_register_load_pointer]][1])
addr_sprite_graphics <=
((sprite_tile_id_buffer[sprites_on_line[shift_register_load_pointer]]) * 16) + (15 -
(vcount - sprite_y_buffer[sprites_on_line[shift_register_load_pointer]]));

                // If vertical flip bit is not set
                else addr_sprite_graphics <=
((sprite_tile_id_buffer[sprites_on_line[shift_register_load_pointer]]) * 16) + (vcount
- sprite_y_buffer[sprites_on_line[shift_register_load_pointer]]);

                //Check against sprites_found to make sure we don't load garbage data
into the graphics buffers
                if (sprites_on_line_pointer <= sprites_found) begin
                    // If horizontal flip bit is set
                    //if
(sprite_rotation_buffer[sprites_on_line[shift_register_load_pointer]][0])
sprite_graphics_buffer[shift_register_load_pointer - 1][31:0] <=
read_data_sprite_graphics[0:31];
                    if
(sprite_rotation_buffer[sprites_on_line[shift_register_load_pointer]][0]) begin
                      // horizontal - flip: reverse all bits
                      sprite_graphics_buffer[shift_register_load_pointer-1] <=
bit_reverse32(read_data_sprite_graphics);
                    end // If horizontal flip bit is not set
                    else sprite_graphics_buffer[shift_register_load_pointer - 1] <=
read_data_sprite_graphics;
                end
```

```
                //If sprite slot is empty, fill place in sprite graphics buffer with
zeros
                else sprite_graphics_buffer[shift_register_load_pointer - 1] <= 0;

                shift_register_load_pointer <= shift_register_load_pointer + 1;

            end

            else if (shift_register_load_pointer == 8) begin

                rw_sprite_graphics <= 0;
                addr_sprite_graphics <= 0;

                //Check against sprites_found to make sure we don't load garbage data
into the graphic buffers
                if (sprites_on_line_pointer <= sprites_found) begin
                    // If horizontal flip bit is set
                    //if
(sprite_rotation_buffer[sprites_on_line[shift_register_load_pointer]][0])
sprite_graphics_buffer[shift_register_load_pointer - 1][31:0] <=
read_data_sprite_graphics[0:31];
                    if
(sprite_rotation_buffer[sprites_on_line[shift_register_load_pointer]][0]) begin
                      // horizontal - flip: reverse all 32 bits
                          sprite_graphics_buffer[shift_register_load_pointer-1]
                            <= bit_reverse32(read_data_sprite_graphics);
// If horizontal flip bit is not set
                    end else sprite_graphics_buffer[shift_register_load_pointer - 1] <=
read_data_sprite_graphics;
                end
                //If sprite slot is empty, fill place in sprite graphics buffer with
zeros
                else sprite_graphics_buffer[shift_register_load_pointer - 1] <= 0;

                shift_register_load_pointer <= shift_register_load_pointer + 1;

            end

            else if (shift_register_load_pointer == 9) begin
                rw_sprite_graphics <= 0;
                addr_sprite_graphics <= 0;
```

```
                shift_load_sprite <= 1;
                shift_load_data[7:0] <= sprite_graphics_buffer;

                shift_load_background <= 1;
                shift_load_data[8] <= background_line_graphics_buffer[0];

                shift_register_load_pointer <= shift_register_load_pointer + 1;

            end

            else begin
                shift_load_sprite <= 0;
                rw_sprite_graphics <= 0;
                addr_sprite_graphics <= 0;
            end
        end

        else begin
            //Reset vblank and hsync memory pointers
            cords_sprite_load <= 0;
            palette_sprite_load <= 0;
            palette_ram_pointer <= 0;
            background_line_pointer <= 0;
            sprite_graphics_pointer <= 0;
            sprites_on_line_pointer <= 0;
            sprites_found <= 0;
            shift_register_load_pointer <= 0;
            for (int i = 0; i < 8; i += 1) begin
                sprites_on_line_palettes[i] <= 0;
            end
            shift_load_sprite <= 0;
            priority_palette_data_out <= {background_line_palette_buffer[hcount[10:5]],
sprites_on_line_palettes};

            //Logic to load new background tile and palette into shift registers
            if (hcount[4:0] == 5'b11111) begin

                shift_load_data[8] <= background_line_graphics_buffer[hcount[10:5]];

                shift_load_background <= 1;

            end
```

```
            else shift_load_background <= 0;

            //Logic to handle pixel doubling
            if (hcount[0]) begin

                shift_enable[8] <= 1;

                //Logic to enable and disable shift registers
                for (int i = 0; i < 8; i += 1) begin
                    if ((sprite_x_buffer[sprites_on_line[i]] >= hcount[10:1]) &&
((sprite_x_buffer[sprites_on_line[i]] < hcount[10:1] + 16)))
                        shift_enable[i] <= 1;
                    else
                        shift_enable[i] <= 0;
                end
            end else shift_enable <= 0;


            //Convert pixel data to colors
            pixel_color <= color_palette_buffer[priority_pixel_data_in + (4 *
priority_palette_data_in)];

        end
    end

function automatic logic [31:0] bit_reverse32(input logic [31:0] in);
 for (int i = 0; i < 32; i++)
   bit_reverse32[i] = in[31 - i];
endfunction

endmodule
```

8.1.4 ppu_top.sv

```
module ppu_top(

   input logic clk,
   input logic reset,
   input logic [31:0] write_data,
   input logic [12:0] address,
   input logic write,
   input chipselect,
   output logic irq,

   output logic [7:0] VGA_R,
   output logic [7:0] VGA_G,
   output logic [7:0] VGA_B,
   output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n
);

   //PPU RW Signals to Memories
   logic ppu_rw_tile_buffer;
   logic ppu_rw_tile_graphics;
   logic ppu_rw_sprite_graphics;
   logic ppu_rw_color_palettes;
   logic ppu_rw_OAM;

   //CPU RW Signals to Memories
   logic cpu_rw_tile_buffer;
   logic cpu_rw_tile_graphics;
   logic cpu_rw_sprite_graphics;
   logic cpu_rw_color_palettes;
   logic cpu_rw_OAM;

   //PPU Write Data to Memories
   logic [31:0] ppu_write_data_tile_buffer;
   logic [31:0] ppu_write_data_tile_graphics;
   logic [31:0] ppu_write_data_sprite_graphics;
   logic [31:0] ppu_write_data_OAM;
   logic [23:0] ppu_write_data_color_palettes;

   //CPU Write Data to Memories
   logic [31:0] cpu_write_data_tile_buffer;
   logic [31:0] cpu_write_data_tile_graphics;
   logic [31:0] cpu_write_data_sprite_graphics;
```

```
logic [31:0] cpu_write_data_OAM;
logic [23:0] cpu_write_data_color_palettes;

//PPU Address Signals to Memories
logic [8:0] ppu_addr_tile_buffer;
logic [10:0] ppu_addr_tile_graphics;
logic [10:0] ppu_addr_sprite_graphics;
logic [2:0] ppu_addr_color_palettes;
logic [7:0] ppu_addr_OAM;

//CPU Address Signals to Memories
logic [8:0] cpu_addr_tile_buffer;
logic [10:0] cpu_addr_tile_graphics;
logic [10:0] cpu_addr_sprite_graphics;
logic [2:0] cpu_addr_color_palettes;
logic [7:0] cpu_addr_OAM;

//PPU Read Data from Memories
logic [31:0] ppu_read_data_tile_buffer;
logic [31:0] ppu_read_data_tile_graphics;
logic [31:0] ppu_read_data_sprite_graphics;
logic [31:0] ppu_read_data_OAM;
logic [23:0] ppu_read_data_color_palettes;

//VGA Singals
logic [10:0] hcount;
logic [9:0] vcount;
logic [23:0] pixel_color;
logic vblank;

//Shift Register Signals
logic [31:0] shift_load_data [8:0];
logic [8:0] shift_enable;
logic shift_load_sprite;
logic shift_load_background;

//Priority Encoder Signals
logic priority_palette_data_asm_to_encoder [8:0];
logic [1:0] priority_pixel_data_encoder_to_asm;
logic priority_palette_data_encoder_to_asm;
logic [1:0] priority_pixel_data_shifter_to_encoder [8:0];
```

```
  addr_decode decoder(
    .addr(address),
    .write_data(write_data),
    .chip_select(chipselect),
    .write(write),
    .rw_tile_buffer(cpu_rw_tile_buffer),
    .rw_tile_graphics(cpu_rw_tile_graphics),
    .rw_sprite_graphics(cpu_rw_sprite_graphics),
    .rw_color_palettes(cpu_rw_color_palettes),
    .rw_OAM(cpu_rw_OAM),
    .write_data_tile_buffer(cpu_write_data_tile_buffer),
    .write_data_tile_graphics(cpu_write_data_tile_graphics),
    .write_data_sprite_graphics(cpu_write_data_sprite_graphics),
    .write_data_OAM(cpu_write_data_OAM),
    .write_data_color_palettes(cpu_write_data_color_palettes),
    .addr_tile_buffer(cpu_addr_tile_buffer),
    .addr_tile_graphics(cpu_addr_tile_graphics),
    .addr_sprite_graphics(cpu_addr_sprite_graphics),
    .addr_color_palettes(cpu_addr_color_palettes),
    .addr_OAM(cpu_addr_OAM)
);


tile_buffer tile_buffer_mem (
    .clk(clk),
    .rw_1(ppu_rw_tile_buffer),
    .rw_2(cpu_rw_tile_buffer),
    .write_data_1(ppu_write_data_tile_buffer),
    .write_data_2(cpu_write_data_tile_buffer),
    .addr_1(ppu_addr_tile_buffer),
    .addr_2(cpu_addr_tile_buffer),
    .read_data_1(ppu_read_data_tile_buffer),
    .read_data_2()
);

tile_graphics tile_graphics_mem (
    .clk(clk),
    .rw_1(ppu_rw_tile_graphics),
    .rw_2(cpu_rw_tile_graphics),
    .write_data_1(ppu_write_data_tile_graphics),
    .write_data_2(cpu_write_data_tile_graphics),
    .addr_1(ppu_addr_tile_graphics),
```

```
    .addr_2(cpu_addr_tile_graphics),
    .read_data_1(ppu_read_data_tile_graphics),
    .read_data_2()
);


sprite_graphics sprite_graphics_mem (
    .clk(clk),
    .rw_1(ppu_rw_sprite_graphics),
    .rw_2(cpu_rw_sprite_graphics),
    .write_data_1(ppu_write_data_sprite_graphics),
    .write_data_2(cpu_write_data_sprite_graphics),
    .addr_1(ppu_addr_sprite_graphics),
    .addr_2(cpu_addr_sprite_graphics),
    .read_data_1(ppu_read_data_sprite_graphics),
    .read_data_2()
);


color_palettes color_palettes_mem(
    .clk(clk),
    .rw_1(ppu_rw_color_palettes),
    .rw_2(cpu_rw_color_palettes),
    .write_data_1(ppu_write_data_color_palettes),
    .write_data_2(cpu_write_data_color_palettes),
    .addr_1(ppu_addr_color_palettes),
    .addr_2(cpu_addr_color_palettes),
    .read_data_1(ppu_read_data_color_palettes),
    .read_data_2()
);

OAM OAM_mem (
    .clk(clk),
    .rw_1(ppu_rw_OAM),
    .rw_2(cpu_rw_OAM),
    .write_data_1(ppu_write_data_OAM),
    .write_data_2(cpu_write_data_OAM),
    .addr_1(ppu_addr_OAM),
    .addr_2(cpu_addr_OAM),
    .read_data_1(ppu_read_data_OAM),
    .read_data_2()
);
```

```
PPU_asm asm(
    .clk(clk),
    .reset(reset),
    .hcount(hcount),
    .vcount(vcount),
    .vblank(vblank),
    .hsync(VGA_HS),
    .pixel_color(pixel_color),
    .rw_tile_buffer(ppu_rw_tile_buffer),
    .rw_tile_graphics(ppu_rw_tile_graphics),
    .rw_sprite_graphics(ppu_rw_sprite_graphics),
    .rw_color_palettes(ppu_rw_color_palettes),
    .rw_OAM(ppu_rw_OAM),
    .write_data_tile_buffer(ppu_write_data_tile_buffer),
    .write_data_tile_graphics(ppu_write_data_tile_graphics),
    .write_data_sprite_graphics(ppu_write_data_sprite_graphics),
    .write_data_OAM(ppu_write_data_OAM),
    .write_data_color_palettes(ppu_write_data_color_palettes),
    .addr_tile_buffer(ppu_addr_tile_buffer),
    .addr_tile_graphics(ppu_addr_tile_graphics),
    .addr_sprite_graphics(ppu_addr_sprite_graphics),
    .addr_color_palettes(ppu_addr_color_palettes),
    .addr_OAM(ppu_addr_OAM),
    .read_data_tile_buffer(ppu_read_data_tile_buffer),
    .read_data_tile_graphics(ppu_read_data_tile_graphics),
    .read_data_sprite_graphics(ppu_read_data_sprite_graphics),
    .read_data_OAM(ppu_read_data_OAM),
    .read_data_color_palettes(ppu_read_data_color_palettes),
    .shift_load_data(shift_load_data),
    .shift_enable(shift_enable),
    .shift_load_sprite(shift_load_sprite),
    .shift_load_background(shift_load_background),
    .priority_palette_data_out(priority_palette_data_asm_to_encoder),
    .priority_pixel_data_in(priority_pixel_data_encoder_to_asm),
    .priority_palette_data_in(priority_palette_data_encoder_to_asm)
);

combined_priority_encoder priority_encoder (
    .pixel_data_in(priority_pixel_data_shifter_to_encoder),
    .palette_data_in(priority_palette_data_asm_to_encoder),
    .pixel_data_out(priority_pixel_data_encoder_to_asm),
```

```
        .palette_data_out(priority_palette_data_encoder_to_asm)
    );

    shift_register_block shift_registers (
        .load_data(shift_load_data),
        .clk(clk),
        .reset(reset),
        .load_sprite(shift_load_sprite),
        .load_background(shift_load_background),
        .enable(shift_enable),
        .out_data(priority_pixel_data_shifter_to_encoder)
    );

    vga vga_inst(
        .clk(clk),
        .reset(reset),
        .pixel_color(pixel_color),
        .VGA_R(VGA_R),
        .VGA_G(VGA_G),
        .VGA_B(VGA_B),
        .VGA_CLK(VGA_CLK),
        .VGA_HS(VGA_HS),
        .VGA_VS(VGA_VS),
        .VGA_BLANK_n(VGA_BLANK_n),
        .VGA_SYNC_n(VGA_SYNC_n),
        .VGA_vBLANK(vblank),
        .hcount(hcount),
        .vcount(vcount)
    );



endmodule
```

8.1.5 priority_encoder.sv

```
module combined_priority_encoder(
   input  logic [1:0] pixel_data_in   [8:0],
   input  logic       palette_data_in [8:0],
   output logic [1:0] pixel_data_out,
   output logic       palette_data_out
);
localparam logic [1:0] TRANSPARENT = 2'b00;

logic [2:0] stage_0 [8:0];
logic [2:0] stage_1 [3:0];
logic [2:0] stage_2 [1:0];
logic [2:0] stage_3;

always_comb begin
   // pack palette+pixel
   for (int i = 0; i < 9; i += 1)
       stage_0[i] = { palette_data_in[i], pixel_data_in[i] };

   // First Stage (pairwise)
   if (stage_0[0][1:0] == TRANSPARENT) stage_1[0] = stage_0[1];
   else                                stage_1[0] = stage_0[0];

   if (stage_0[2][1:0] == TRANSPARENT) stage_1[1] = stage_0[3];
   else                                stage_1[1] = stage_0[2];

   if (stage_0[4][1:0] == TRANSPARENT) stage_1[2] = stage_0[5];
   else                                stage_1[2] = stage_0[4];

   if (stage_0[6][1:0] == TRANSPARENT) stage_1[3] = stage_0[7];
   else                                stage_1[3] = stage_0[6];

   // Second Stage
   if (stage_1[0][1:0] == TRANSPARENT) stage_2[0] = stage_1[1];
   else                                stage_2[0] = stage_1[0];

   if (stage_1[2][1:0] == TRANSPARENT) stage_2[1] = stage_1[3];
   else                                stage_2[1] = stage_1[2];

   // Third Stage
   if (stage_2[0][1:0] == TRANSPARENT) stage_3 = stage_2[1];
   else                                stage_3 = stage_2[0];
```

```
    // Fourth Stage (vs slot 8)
    if (stage_3[1:0] == TRANSPARENT)
        { palette_data_out, pixel_data_out } = stage_0[8];
    else
        { palette_data_out, pixel_data_out } = stage_3;
end

endmodule
```

8.1.6 shift_registers.sv

```
module shift_register(
    input logic [31:0] load_data,
    input logic enable, clk, reset, load,
    output logic [1:0] out_data
);

logic [31:0] shift_buffer;

assign out_data = shift_buffer[1:0];

always @(posedge clk) begin
    if (reset) shift_buffer <= 0;
    else if (load) shift_buffer <= load_data;
    else if (enable) shift_buffer <= shift_buffer >> 2;
end

endmodule

module shift_register_block(
    input logic [31:0] load_data [8:0],
    input logic clk, reset, load_sprite, load_background,
    input logic [8:0] enable,
    output logic [1:0] out_data [8:0]
);
    generate
        genvar i;
```

```
      for (i=0; i<8; i = i + 1) shift_register sprite_shift (load_data[i], enable[i],
clk, reset, load_sprite, out_data[i]);
   endgenerate
   shift_register background_shift (load_data[8], enable[8], clk, reset,
load_background, out_data[8]);

endmodule
```

8.1.7 vga.sv

```
module vga(
   input logic clk,
   input logic reset,
   input logic [23:0] pixel_color,
   output logic [7:0] VGA_R, VGA_G, VGA_B,
   output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n, VGA_vBLANK,
   output logic [10:0] hcount,
   output logic [9:0] vcount
);

vga_counters counters(.clk50(clk), .*);

always_ff @(posedge clk)
   if (reset) begin
     VGA_R <= 8'h0;
     VGA_G <= 8'h0;
     VGA_B <= 8'h0;
   end else begin
     VGA_R <= pixel_color[23:16];
     VGA_G <= pixel_color[15:8];
     VGA_B <= pixel_color[7:0];
   end

endmodule


module vga_counters(
input logic       clk50, reset,
output logic [10:0] hcount,  // hcount[10:1] is pixel column
output logic [9:0]  vcount,  // vcount[9:0] is pixel row
output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n, VGA_vBLANK);
```

```
/*
* 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
*
* HCOUNT 1599 0             1279       1599 0
*            _______________              ________
* ___________|    Video      |____________|  Video
*
*
* |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
*       _____________________      _____________
* |____|       VGA_HS          |____|
*/
  // Parameters for hcount
  parameter HACTIVE      = 11'd 1280,
            HFRONT_PORCH = 11'd 32,
            HSYNC        = 11'd 192,
            HBACK_PORCH  = 11'd 96,
            HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                           HBACK_PORCH; // 1600

  // Parameters for vcount
  parameter VACTIVE      = 10'd 480,
            VFRONT_PORCH = 10'd 10,
            VSYNC        = 10'd 2,
            VBACK_PORCH  = 10'd 33,
            VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                           VBACK_PORCH; // 525

  logic endOfLine;

  always_ff @(posedge clk50 or posedge reset)
    if (reset)          hcount <= 0;
    else if (endOfLine) hcount <= 0;
    else                hcount <= hcount + 11'd 1;

  assign endOfLine = hcount == HTOTAL - 1;

  logic endOfField;

  always_ff @(posedge clk50 or posedge reset)
    if (reset)          vcount <= 0;
    else if (endOfLine)
```

```
    if (endOfField)   vcount <= 0;
    else              vcount <= vcount + 10'd 1;

  assign endOfField = vcount == VTOTAL - 1;

  // Horizontal sync: from 0x520 to 0x5DF (0x57F)
  // 101 0010 0000 to 101 1101 1111
  assign VGA_HS = !( (hcount[10:8] == 3'b101) &
       !(hcount[7:5] == 3'b111));
  assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

  assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused

  // Horizontal active: 0 to 1279     Vertical active: 0 to 479
  // 101 0000 0000  1280        01 1110 0000  480
  // 110 0011 1111  1599        10 0000 1100  524
  assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
     !( vcount[9] | (vcount[8:5] == 4'b1111) );

  assign VGA_vBLANK = ( vcount[9] | (vcount[8:5] == 4'b1111) );

  /* VGA_CLK is 25 MHz
   *             __    __    __
   * clk50    __|  |__|  |__|
   *
   *             _____       __
   * hcount[0]__|     |_____|
   */
  assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule
```

8.2. Tile Hardware

8.2.1 tiles.sv

```
/*
Modified version of tiles.sv. Original version provided by Professor Stephen Edwards
https://www.cs.columbia.edu/~sedwards/classes/2025/4840-spring/tiles.pdf
*/

module tiles
 (input logic         VGA_CLK, VGA_RESET,
  output logic [7:0]  VGA_R, VGA_G, VGA_B,
  output logic        VGA_HS, VGA_VS, VGA_BLANK_n,

  input logic           mem_clk,         // Clock for memory ports

  input logic [12:0]  tm_address,      // Tilemap memory port
  input logic           tm_we,
  input logic [7:0]   tm_din,
  output logic [7:0]  tm_dout,

  input logic [13:0]  ts_address,      // Tileset memory port
  input logic           ts_we,
  input logic [3:0]   ts_din,
  output logic [3:0]  ts_dout,

  input logic [3:0]   palette_address, // Palette memory port
  input logic           palette_we,
  input logic [23:0]  palette_din,
  output logic [23:0] palette_dout);

  logic [9:0]           hcount;          // From counters
  logic [8:0]           vcount;

  logic [2:0]           hcount1;         // Pipeline registers
  logic           VGA_HS0, VGA_HS1, VGA_HS2;
  logic           VGA_BLANK_n0, VGA_BLANK_n1, VGA_BLANK_n2;

  logic [7:0]           tilenumber;      // Memory outputs
  logic [3:0]           colorindex;
```

```
  /* verilator lint_off UNUSED */
  logic               unconnected; // Extra vcount bit from counters
  /* verilator lint_on UNUSED */

  // Frame-snapshot helper signals
  logic [12:0] copy_addr;   //8 KiB tile map
  logic        copying;     //Actively copying
  logic        copy_we;     //write-enable for the display side cache
  logic [7:0]  tm_cpu_dout; //Data coming from the CPU side RAM to the display side
cache

  logic        copying_d1;
  logic [12:0] copy_addr_d1;

  always_ff @(posedge VGA_CLK) begin
     copying_d1   <= copying;
     copy_addr_d1 <= copy_addr;
  end

  assign copy_we = copying_d1;

  vga_counters cntrs(.vcount( {unconnected, vcount} ), // VGA Counters
           .VGA_BLANK_n( VGA_BLANK_n0 ),
           .VGA_HS( VGA_HS0 ),
           .*);

  //CPU side tile RAM
  //port-A (clk = mem_clk) from Avalon bus
  //port-B (clk = VGA_CLK) to frame cacher
  twoportbram #(.DATA_BITS(8), .ADDRESS_BITS(13)) tilemap_cpu (
     .clk1  ( mem_clk ),
     .addr1 ( tm_address ),
     .we1   ( tm_we ),
     .din1  ( tm_din ),
     .dout1 ( tm_dout ),

     .clk2  ( VGA_CLK ),
     .addr2 ( copy_addr ),
     .we2   ( 1'b0 ),
     .din2  ( 8'hxx ),
     .dout2 ( tm_cpu_dout )
  );
```

```
//Display side tile cache
//port-A (clk = VGA_CLK) from pixel pipeline
//port-B (clk = VGA_CLK) to frame cacher (writes)
twoportbram #(.DATA_BITS(8), .ADDRESS_BITS(13)) tilemap_disp (
   .clk1  ( VGA_CLK ),
   .addr1 ( { vcount[8:3], hcount[9:3] } ),
   .we1   ( 1'b0 ),
   .din1  ( 8'hxx ),
   .dout1 ( tilenumber ),

   .clk2  ( VGA_CLK ),
   .addr2 ( copy_addr_d1 ),
   .we2   ( copy_we ),
   .din2  ( tm_cpu_dout ),
   .dout2 ( )
);

//Per-frame cacher (runs during vblank)
//starts at the first pixel of vblank (vcount = 480, hcount = 0)
always_ff @(posedge VGA_CLK or posedge VGA_RESET) begin
   if (VGA_RESET) begin
      copying   <= 1'b0;
      copy_addr <= 13'd0;
   end else begin
      if (!copying && hcount == 10'd0 && vcount == 10'd480) begin //start of vblank
         //start caching
         copying   <= 1'b1;
         copy_addr <= 13'd0;
      end else if (copying) begin
         copy_addr <= copy_addr + 13'd1;
         if (copy_addr == 13'd8191) //finished copying tile map into the cache
            copying <= 1'b0;
      end
   end
end


always_ff @(posedge VGA_CLK)                        // Pipeline registers
  { hcount1, VGA_BLANK_n1, VGA_HS1 } <=
    { hcount[2:0], VGA_BLANK_n0, VGA_HS0 };
```

```
  twoportbram #(.DATA_BITS(4), .ADDRESS_BITS(14))  // Tile Set
  tileset(.clk1  ( VGA_CLK ), .clk2 ( mem_clk ),
     .addr1 ( { tilenumber, vcount[2:0], hcount1 } ),
     .we1   ( 1'b0 ), .din1( 4'hx), .dout1( colorindex ),
     .addr2 ( ts_address ),
     .we2   ( ts_we ), .din2( ts_din ), .dout2( ts_dout ));

  always_ff @(posedge VGA_CLK)                     // Pipeline registers
    { VGA_BLANK_n2, VGA_HS2 } <= { VGA_BLANK_n1, VGA_HS1 };

  twoportbram #(.DATA_BITS(24), .ADDRESS_BITS(4))  // Palette
  palette(.clk1  ( VGA_CLK ), .clk2 ( mem_clk ),
     .addr1 ( colorindex ),
     .we1   ( 1'b0 ), .din1( 24'hx), .dout1( { VGA_B, VGA_G, VGA_R } ),
     .addr2 ( palette_address ),
     .we2   ( palette_we ), .din2( palette_din ), .dout2( palette_dout ));

  always_ff @(posedge VGA_CLK)                     // Pipeline registers
    { VGA_BLANK_n, VGA_HS } <= { VGA_BLANK_n2, VGA_HS2 };

endmodule
```

8.2.2 twoportbram.sv

```
/*
twoportbram.sv provided by Professor Stephen Edwards
https://www.cs.columbia.edu/~sedwards/classes/2025/4840-spring/tiles.pdf
*/


module twoportbram
 #(parameter int DATA_BITS = 8,  ADDRESS_BITS = 10)
  (input logic                  clk1,  clk2,
   input logic [ADDRESS_BITS-1:0] addr1, addr2,
   input logic [DATA_BITS-1:0]    din1,  din2,
   input logic                  we1,   we2,
   output logic [DATA_BITS-1:0]   dout1, dout2);

  localparam WORDS = 1 << ADDRESS_BITS;
```

```
  /* verilator lint_off MULTIDRIVEN */
  logic [DATA_BITS-1:0]         mem [WORDS-1:0];
  /* verilator lint_on MULTIDRIVEN */

  always_ff @(posedge clk1)
    if (we1) begin
    mem[addr1] <= din1;
    dout1 <= din1;
    end else dout1 <= mem[addr1];

  always_ff @(posedge clk2)
    if (we2) begin
    mem[addr2] <= din2;
    dout2 <= din2;
    end else dout2 <= mem[addr2];

endmodule
```

8.2.3 vga_counters.sv

```
/*
vga_counters.sv provided by Professor Stephen Edwards
https://www.cs.columbia.edu/~sedwards/classes/2025/4840-spring/tiles.pdf
*/


module vga_counters(
  input  logic       VGA_CLK, VGA_RESET,
  output logic [9:0] hcount, // 0-639 active, 640-799 blank/sync
  output logic [9:0] vcount, // 0-479 active, 480-524 blank/sync
  output logic       VGA_HS, VGA_VS, VGA_BLANK_n);
  logic endOfLine;
  assign endOfLine = hcount == 10'd 799;

  always_ff @(posedge VGA_CLK or posedge VGA_RESET)
    if (VGA_RESET)      hcount <= 10'd 797;
    else if (endOfLine) hcount <= 0;
    else                hcount <= hcount + 10'd 1;

  logic endOfFrame;
  assign endOfFrame = vcount == 10'd 524;
```

```
  always_ff @(posedge VGA_CLK or posedge VGA_RESET)
    if (VGA_RESET)      vcount <= 10'd 524;
    else if (endOfLine)
      if (endOfFrame)   vcount <= 10'd 0;
      else              vcount <= vcount + 10'd 1;

  // 656 <= hcount <= 751
  assign VGA_HS = !( hcount[9:7] == 3'b101 &
                     hcount[6:4] != 3'b000 & hcount[6:4] != 3'b111 );
  assign VGA_VS = !( vcount[9:1] == 9'd 245 ); // Lines 490 and 491

  // hcount < 640 && vcount < 480
  assign VGA_BLANK_n = !( hcount[9] & (hcount[8] | hcount[7]) ) &
                       !( vcount[9] | (vcount[8:5] == 4'b1111) );
endmodule
```

8.2.4 vga_tiles.sv

```
/*
vga_tiles.sv provided by Professor Stephen Edwards
https://www.cs.columbia.edu/~sedwards/classes/2025/4840-spring/tiles.pdf
*/


/*
* Avalon memory-mapped agent peripheral that produces a VGA tile display
*
* Stephen A. Edwards
* Columbia University
*
* Memory map:
*
* 0000 - 1FFF Tilemap (8K, tile number is 8 bits per byte)
* 2000 - 203F Palette (64, 24 bits every 4 bytes)
* 4000 - 7FFF Tileset (16K, color index is lower 4 bits of each byte)
*
* 00m mmmm mmmm mmmm  Tilemap
* 010 0000 00pp ppbb  Palette
* 1ss ssss ssss ssss  Tileset
*
* In the 64-byte palette region, every color occupies 4 bytes, although
```

```
* only 24 bits are stored.  Writing to the first 3 bytes in each group
* writes a byte into the 24-bit color register.  Writing to the fourth
* byte writes the color register to the palette memory; any data written to
* these addresses is ignored; they always read 0.
*
* | Offset |    On Write          |  On Read          |
* +--------+----------------------+-------------------+
* |   0    | creg[7:0] <- data    | palette[0].red    |
* |   1    | creg[15:8] <- data   | palette[0].green  |
* |   2    | creg[23:16] <- data  | palette[0].blue   |
* |   3    | palette[0] <- creg   | Always 0          |
* |   4    | creg[7:0] <- data    | palette[1].red    |
* |   5    | creg[15:8] <- data   | palette[1].green  |
* |   6    | creg[23:16] <- data  | palette[1].blue   |
* |   7    | palette[1] <- creg   | Always 0          |
*  ...
* |   60    | creg[7:0] <- data   | palette[15].red   |
* |   61    | creg[15:8] <- data  | palette[15].green |
* |   62    | creg[23:16] <- data | palette[15].blue  |
* |   63    | palette[15] <- creg | Always 0          |
*
*/
module vga_tiles
 (input logic          clk, reset,                    // Avalon MM Agent port
  input logic          chipselect, write,             // read == chipselect & !write
  input logic [14:0] address,                       // 32K window
  input logic [7:0]  writedata,                     // 8-bit interface
  output logic [7:0] readdata,

  input logic        vga_clk_in, VGA_RESET,         // VGA signals
  output logic [7:0] VGA_R, VGA_G, VGA_B,
  output logic       VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n);

  logic [2:0]          creg_write;                   // Latch enable per byte
  logic          tm_we, ts_we, palette_we;      // Memory write enables
  logic [7:0]          tm_dout;                      // Data from tilemap
  logic [3:0]          ts_dout;                      // Data from tileset
  logic [23:0]       creg, palette_dout;           // Data to/from palette

  tiles tiles(.mem_clk        ( clk           ),
        .tm_address     ( address[12:0] ), .tm_din     ( writedata      ),
        .ts_address     ( address[13:0] ), .ts_din     ( writedata[3:0] ),
```

```
        .palette_address( address[5:2]  ), .palette_din( creg           ), .*);
  assign VGA_CLK = vga_clk_in;

  always_comb begin                                    // Address Decoder
     {tm_we, ts_we, palette_we, creg_write, readdata } = { 6'b 0, 8'h xx };
     if (chipselect)
  if (address[14] == 1'b 1) begin                 // Tileset 1--------------
     ts_we    = write;                            //  Write to tileset mem
     readdata = { 4'h 0, ts_dout };               //  Read lower 4 bits; pad upper
  end else if (address[13] == 1'b 0) begin        // Tilemap 00-------------
     tm_we    = write;                            //  Write to tilemap mem
     readdata = tm_dout;                          //  Read 8 bits
  end else if ( address[12:6] == 7'b 0_0000_00 ) // Palette 010000000------
     case (address[1:0])
       2'h 0 : begin readdata = palette_dout[7:0];   // Read red byte
                 creg_write[0] = write;          // creg <- red
                end
       2'h 1 : begin readdata = palette_dout[15:8];  // Read green byte
                           creg_write[1] = write;          // creg <- green
                     end
             2'h 2 : begin readdata = palette_dout[23:16]; // Read blue byte
                           creg_write[2] = write;          // creg <- blue
                     end
       2'h 3 : begin readdata = 8'h 00;              // Always reads as 00
                           palette_we = write;             // mem <- creg
                     end
     endcase
  end

  always_ff @(posedge clk or posedge reset)
    if (reset) creg <= 24'b 0; else begin
  if (creg_write[0]) creg[7:0]   <= writedata;    // Write byte (color)
  if (creg_write[1]) creg[15:8]  <= writedata;    // to creg according to
  if (creg_write[2]) creg[23:16] <= writedata;    // creg_write bits
    end
endmodule
```

8.2.5 soc_system_top.sv

```
// ====================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ====================================================================
//
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ====================================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

  ///////// ADC /////////
  inout   ADC_CS_N,
  output    ADC_DIN,
  input   ADC_DOUT,
  output    ADC_SCLK,

  ///////// AUD /////////
```

```
input   AUD_ADCDAT,
inout   AUD_ADCLRCK,
inout   AUD_BCLK,
output    AUD_DACDAT,
inout   AUD_DACLRCK,
output    AUD_XCK,

///////// CLOCK2 /////////
input   CLOCK2_50,

///////// CLOCK3 /////////
input   CLOCK3_50,

///////// CLOCK4 /////////
input   CLOCK4_50,

///////// CLOCK /////////
input   CLOCK_50,

///////// DRAM /////////
output [12:0] DRAM_ADDR,
output [1:0]  DRAM_BA,
output    DRAM_CAS_N,
output    DRAM_CKE,
output    DRAM_CLK,
output    DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output    DRAM_LDQM,
output    DRAM_RAS_N,
output    DRAM_UDQM,
output    DRAM_WE_N,

///////// FAN /////////
output    FAN_CTRL,

///////// FPGA /////////
output    FPGA_I2C_SCLK,
inout   FPGA_I2C_SDAT,

///////// GPIO /////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,
```

```
//////// HEX0 ////////
output [6:0]  HEX0,

//////// HEX1 ////////
output [6:0]  HEX1,

//////// HEX2 ////////
output [6:0]  HEX2,

//////// HEX3 ////////
output [6:0]  HEX3,

//////// HEX4 ////////
output [6:0]  HEX4,

//////// HEX5 ////////
output [6:0]  HEX5,

//////// HPS ////////
inout   HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
output [2:0]  HPS_DDR3_BA,
output    HPS_DDR3_CAS_N,
output    HPS_DDR3_CKE,
output    HPS_DDR3_CK_N,
output    HPS_DDR3_CK_P,
output    HPS_DDR3_CS_N,
output [3:0]  HPS_DDR3_DM,
inout [31:0]  HPS_DDR3_DQ,
inout [3:0]   HPS_DDR3_DQS_N,
inout [3:0]   HPS_DDR3_DQS_P,
output    HPS_DDR3_ODT,
output    HPS_DDR3_RAS_N,
output    HPS_DDR3_RESET_N,
input   HPS_DDR3_RZQ,
output    HPS_DDR3_WE_N,
output    HPS_ENET_GTX_CLK,
inout   HPS_ENET_INT_N,
output    HPS_ENET_MDC,
inout   HPS_ENET_MDIO,
input   HPS_ENET_RX_CLK,
```

```
input [3:0]   HPS_ENET_RX_DATA,
input   HPS_ENET_RX_DV,
output [3:0]  HPS_ENET_TX_DATA,
output    HPS_ENET_TX_EN,
inout   HPS_GSENSOR_INT,
inout   HPS_I2C1_SCLK,
inout   HPS_I2C1_SDAT,
inout   HPS_I2C2_SCLK,
inout   HPS_I2C2_SDAT,
inout   HPS_I2C_CONTROL,
inout   HPS_KEY,
inout   HPS_LED,
inout   HPS_LTC_GPIO,
output    HPS_SD_CLK,
inout   HPS_SD_CMD,
inout [3:0]   HPS_SD_DATA,
output    HPS_SPIM_CLK,
input   HPS_SPIM_MISO,
output    HPS_SPIM_MOSI,
inout   HPS_SPIM_SS,
input   HPS_UART_RX,
output    HPS_UART_TX,
input   HPS_USB_CLKOUT,
inout [7:0]   HPS_USB_DATA,
input   HPS_USB_DIR,
input   HPS_USB_NXT,
output    HPS_USB_STP,

//////// IRDA ////////
input   IRDA_RXD,
output    IRDA_TXD,

//////// KEY ////////
input [3:0]   KEY,

//////// LEDR ////////
output [9:0]  LEDR,

//////// PS2 ////////
inout   PS2_CLK,
inout   PS2_CLK2,
inout   PS2_DAT,
```

```
  inout   PS2_DAT2,

  ///////// SW /////////
  input [9:0]   SW,

  ///////// TD /////////
  input   TD_CLK27,
  input [7:0]   TD_DATA,
  input   TD_HS,
  output    TD_RESET_N,
  input   TD_VS,

///////// VGA /////////
  output [7:0]  VGA_B,
  output    VGA_BLANK_N,
  output    VGA_CLK,
  output [7:0]  VGA_G,
  output    VGA_HS,
  output [7:0]  VGA_R,
  output    VGA_SYNC_N,
  output    VGA_VS
  );

  soc_system soc_system0
    (
     .clk_clk                       ( CLOCK_50 ),
     .reset_reset_n                 ( 1'b1 ),

     .hps_ddr3_mem_a                ( HPS_DDR3_ADDR ),
     .hps_ddr3_mem_ba               ( HPS_DDR3_BA ),
     .hps_ddr3_mem_ck               ( HPS_DDR3_CK_P ),
     .hps_ddr3_mem_ck_n             ( HPS_DDR3_CK_N ),
     .hps_ddr3_mem_cke              ( HPS_DDR3_CKE ),
     .hps_ddr3_mem_cs_n             ( HPS_DDR3_CS_N ),
     .hps_ddr3_mem_ras_n            ( HPS_DDR3_RAS_N ),
     .hps_ddr3_mem_cas_n            ( HPS_DDR3_CAS_N ),
     .hps_ddr3_mem_we_n             ( HPS_DDR3_WE_N ),
     .hps_ddr3_mem_reset_n          ( HPS_DDR3_RESET_N ),
     .hps_ddr3_mem_dq               ( HPS_DDR3_DQ ),
     .hps_ddr3_mem_dqs              ( HPS_DDR3_DQS_P ),
     .hps_ddr3_mem_dqs_n            ( HPS_DDR3_DQS_N ),
     .hps_ddr3_mem_odt              ( HPS_DDR3_ODT ),
```

```
    .hps_ddr3_mem_dm              ( HPS_DDR3_DM ),
    .hps_ddr3_oct_rzqin           ( HPS_DDR3_RZQ ),

    .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
    .hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
    .hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
    .hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
    .hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
    .hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
    .hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
    .hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
    .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
    .hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
    .hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
    .hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
    .hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
    .hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

    .hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD          ),
    .hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]      ),
    .hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]      ),
    .hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK          ),
    .hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]      ),
    .hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]      ),

    .hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]     ),
    .hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]     ),
    .hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]     ),
    .hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]     ),
    .hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]     ),
    .hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]     ),
    .hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]     ),
    .hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]     ),
    .hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT      ),
    .hps_hps_io_usb1_inst_STP     ( HPS_USB_STP         ),
    .hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR         ),
    .hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT         ),

    .hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
    .hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
    .hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
    .hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),
```

```
    .hps_hps_io_uart0_inst_RX     ( HPS_UART_RX     ),
    .hps_hps_io_uart0_inst_TX     ( HPS_UART_TX     ),

    .hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT     ),
    .hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK     ),

    .hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT     ),
    .hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK     ),

    .hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
    .hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
    .hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

    .hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
    .hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
    .hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
    .hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),

    .vga_r                        ( VGA_R ),
    .vga_g                        ( VGA_G ),
    .vga_b                        ( VGA_B ),
    .vga_clk                      ( VGA_CLK ),
    .vga_hs                       ( VGA_HS ),
    .vga_vs                       ( VGA_VS ),
    .vga_blank_n                  ( VGA_BLANK_N )
);

 // The following quiet the "no driver" warnings for output
 // pins and should be removed if you use any of these peripherals

 assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
 assign ADC_DIN = SW[0];
 assign ADC_SCLK = SW[0];

 assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
 assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
 assign AUD_DACDAT = SW[0];
 assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
 assign AUD_XCK = SW[0];

 assign DRAM_ADDR = { 13{ SW[0] } };
```

```
  assign DRAM_BA = { 2{ SW[0] } };
  assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : { 16{ 1'bZ } };
  assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
    DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };

  assign FAN_CTRL = SW[0];

  assign FPGA_I2C_SCLK = SW[0];
  assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

  assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : { 36{ 1'bZ } };
  assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : { 36{ 1'bZ } };

  assign HEX0 = { 7{ SW[1] } };
  assign HEX1 = { 7{ SW[2] } };
  assign HEX2 = { 7{ SW[3] } };
  assign HEX3 = { 7{ SW[4] } };
  assign HEX4 = { 7{ SW[5] } };
  assign HEX5 = { 7{ SW[6] } };

  assign IRDA_TXD = SW[0];

  assign LEDR = { 10{SW[7]} };

  assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
  assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
  assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
  assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

  assign TD_RESET_N = SW[0];

  assign VGA_SYNC_N = 1'b0; // For composite sync on green (unused)

endmodule
```

8.3. Software Code

8.3.1 assets.h

```
#ifndef ASSETS_H
#define ASSETS_H
#include <cstdint>

//Hardware layout constants
constexpr int TILE_COLS = 80; //Tile map width
constexpr int TILE_ROWS = 60; //Tile map height
constexpr int TM_STRIDE = 128; //Bytes per tile map row

//Playfield: 15×20 interior, with framed 1-tile border = 17×22
constexpr int PF_WIDTH  = 17;
constexpr int PF_HEIGHT = 22;

//Coordinates to center the playfield
constexpr int PF_LEFT  = (TILE_COLS - PF_WIDTH ) / 2;
constexpr int PF_TOP   = (TILE_ROWS - PF_HEIGHT) / 4;

//Coordinates for score / lines
constexpr int HUD_COL = 10;
constexpr int HUD_SCORE_ROW = 42;
constexpr int HUD_LINES_ROW = HUD_SCORE_ROW + 9;
constexpr int LEVEL_ROW = HUD_SCORE_ROW - 9;

//Coordinates for next box
constexpr int NEXT_COL = PF_LEFT + PF_WIDTH + 3;
constexpr int NEXT_ROW = PF_TOP + 1;



//Palette Definition (Bytes are in reverse order (BGR))
static constexpr uint32_t PALETTE24[16] = {
   0x000000, //Color 0
   0xFF0000, //Color 1
   0x00FF00, //Color 2
   0x0000FF, //Color 3
   0xFFFF00, //Color 4
   0x00FFFF, //Color 5
```

```
    0xFF00FF, //Color 6
    0x808080, //Color 7
    0xfc036f, //Color 8
    0x606060, //Color 9
    0xA0A0A0, //Color 10
    0xC0C0C0, //Color 11
    0xE0E0E0, //Color 12
    0xF0F0F0, //Color 13
    0xFFFFFF, //Color 14
    0xFFFFFF //Color 15
};

//Tile indexes
enum : uint8_t {
    TILE_EMPTY = 0,
    TILE_WALL = 1,
    TILE_RED = 2,
    TILE_GREEN = 3,
    TILE_BLUE = 4,
    TILE_YELLOW = 5,
    TILE_CYAN = 6,
    TILE_MAG = 7,
    TILE_PURPLE = 8,
    TILE_WHITE = 14
};

//Palette indexes
static constexpr uint8_t TILE2PAL(uint8_t tile) {
    switch(tile){
        case TILE_WALL: return 7;
        case TILE_RED: return 1;
        case TILE_GREEN: return 2;
        case TILE_BLUE: return 3;
        case TILE_YELLOW: return 4;
        case TILE_CYAN: return 5;
        case TILE_MAG: return 6;
        case TILE_PURPLE: return 8;
        case TILE_WHITE: return 14;
        default: return 0;
    }
}
```

```
//Tile set
static uint8_t TILESET[16384];
inline void build_tileset() {
   for (int tile = 0; tile < 256; ++tile) {
       uint8_t col = TILE2PAL(tile);
       for (int pixel = 0; pixel < 64; ++pixel)
           TILESET[tile * 64 + pixel] = col;
   }
}

#endif
```

8.3.2 audio.cpp

```
#include <iostream>
#include <mpg123.h>
#include <ao/ao.h>

int main() {
   const char *filename = "Tetris.mp3"; //MP3 Filename

   //Initialize mpg123
   if (mpg123_init() != MPG123_OK) {
       std::cerr << "Unable to initialize mpg123\n";
       return 1;
   }

   int music_handler_error;
   mpg123_handle *music_handler = mpg123_new(NULL, &music_handler_error);
   if (!music_handler) {
       std::cerr << "Failed to create mpg123_handle: " <<
mpg123_plain_strerror(music_handler_error) << "\n";
       mpg123_exit();
       return 1;
   }

   //Open the MP3 file
   if (mpg123_open(music_handler, filename) != MPG123_OK) {
       std::cerr << "Error opening `" << filename << "`\n";
       mpg123_delete(music_handler);
       mpg123_exit();
       return 1;
```

```
}

//Get audio format
long rate;
int channels, encoding;
if (mpg123_getformat(music_handler, &rate, &channels, &encoding) != MPG123_OK) {
    std::cerr << "Failed to get MP3 format information\n";
    mpg123_close(music_handler);
    mpg123_delete(music_handler);
    mpg123_exit();
    return 1;
}

//Initialize libao with format from mp3 file
ao_initialize();
ao_sample_format format;
format.bits = mpg123_encsize(encoding) * 8;
format.rate = rate;
format.channels = channels;
format.byte_format = AO_FMT_NATIVE;
format.matrix = nullptr;

//Open the ALSA drivers
int driver = ao_driver_id("alsa");
if (driver < 0) {
    std::cerr << "ALSA driver not available\n";
    ao_shutdown();
    mpg123_close(music_handler);
    mpg123_delete(music_handler);
    mpg123_exit();
    return 1;
}

//Explicitly open card 1, device 0 (Our audio device)
ao_option ao_opts[] = {
    {"dev", "plughw:1,0"}
};

ao_device *usb_audio_device = ao_open_live(driver, &format, ao_opts);
if (!usb_audio_device) {
    std::cerr << "ao_open_live failed\n";
    ao_shutdown();
```

```
        mpg123_close(music_handler);
        mpg123_delete(music_handler);
        mpg123_exit();
        return 1;
    }

    //Decode and play audio
    unsigned char buffer[8192];
    size_t done = 0;

    //Loop playback forever
    while (true) {

        //Rewind to start
        mpg123_seek(music_handler, 0, SEEK_SET);

        //Play track
        while (mpg123_read(music_handler, buffer, 8192, &done) == MPG123_OK && done >
0) {
            ao_play(usb_audio_device, reinterpret_cast<char*>(buffer), done);
        }

    }
    return 0;
}
```

8.3.3 font.h

```
#ifndef _FONT5X7_H_
#define _FONT5X7_H_

//from
https://github.com/Ameba8195/Arduino/blob/master/hardware_v2/cores/arduino/font5x7.h

/*
* Take 'A' as example.
* 'A' use 5 byte to denote:
*     0x7C, 0x12, 0x11, 0x12, 0x7C
*
* and we represent it in base 2:
*     0x7C: 01111100
*     0x12: 00010010
```

```
*     0x11: 00010001
*     0x12: 00010010
*     0x7C: 01111100
* where 1 is font color, and 0 is background color
*
* So it's 'A' if we look it in counter-clockwise for 90 degree.
* In general case, we also add a background line to seperate from other character:
*     0x7C: 01111100
*     0x12: 00010010
*     0x11: 00010001
*     0x12: 00010010
*     0x7C: 01111100
*     0x00: 00000000
*
**/

// standard ascii 5x7 font
static unsigned char font5x7[] = {
   0x00, 0x00, 0x00, 0x00, 0x00,  // 0x00 (nul)
   0x3E, 0x5B, 0x4F, 0x5B, 0x3E,  // 0x01 (soh)
   0x3E, 0x6B, 0x4F, 0x6B, 0x3E,  // 0x02 (stx)
   0x1C, 0x3E, 0x7C, 0x3E, 0x1C,  // 0x03 (etx)
   0x18, 0x3C, 0x7E, 0x3C, 0x18,  // 0x04 (eot)
   0x1C, 0x57, 0x7D, 0x57, 0x1C,  // 0x05 (enq)
   0x1C, 0x5E, 0x7F, 0x5E, 0x1C,  // 0x06 (ack)
   0x00, 0x18, 0x3C, 0x18, 0x00,  // 0x07 (bel)
   0xFF, 0xE7, 0xC3, 0xE7, 0xFF,  // 0x08 (bs)
   0x00, 0x18, 0x24, 0x18, 0x00,  // 0x09 (tab)
   0xFF, 0xE7, 0xDB, 0xE7, 0xFF,  // 0x0A (lf)
   0x30, 0x48, 0x3A, 0x06, 0x0E,  // 0x0B (vt)
   0x26, 0x29, 0x79, 0x29, 0x26,  // 0x0C (np)
   0x40, 0x7F, 0x05, 0x05, 0x07,  // 0x0D (cr)
   0x40, 0x7F, 0x05, 0x25, 0x3F,  // 0x0E (so)
   0x5A, 0x3C, 0xE7, 0x3C, 0x5A,  // 0x0F (si)
   0x7F, 0x3E, 0x1C, 0x1C, 0x08,  // 0x10 (dle)
   0x08, 0x1C, 0x1C, 0x3E, 0x7F,  // 0x11 (dc1)
   0x14, 0x22, 0x7F, 0x22, 0x14,  // 0x12 (dc2)
   0x5F, 0x5F, 0x00, 0x5F, 0x5F,  // 0x13 (dc3)
   0x06, 0x09, 0x7F, 0x01, 0x7F,  // 0x14 (dc4)
   0x00, 0x66, 0x89, 0x95, 0x6A,  // 0x15 (nak)
   0x60, 0x60, 0x60, 0x60, 0x60,  // 0x16 (syn)
   0x94, 0xA2, 0xFF, 0xA2, 0x94,  // 0x17 (etb)
```

```
  0x08, 0x04, 0x7E, 0x04, 0x08,  // 0x18 (can)
  0x10, 0x20, 0x7E, 0x20, 0x10,  // 0x19 (em)
  0x08, 0x08, 0x2A, 0x1C, 0x08,  // 0x1A (eof)
  0x08, 0x1C, 0x2A, 0x08, 0x08,  // 0x1B (esc)
  0x1E, 0x10, 0x10, 0x10, 0x10,  // 0x1C (fs)
  0x0C, 0x1E, 0x0C, 0x1E, 0x0C,  // 0x1D (gs)
  0x30, 0x38, 0x3E, 0x38, 0x30,  // 0x1E (rs)
  0x06, 0x0E, 0x3E, 0x0E, 0x06,  // 0x1F (us)
  0x00, 0x00, 0x00, 0x00, 0x00,  // 0x20
  0x00, 0x00, 0x5F, 0x00, 0x00,  // 0x21 !
  0x00, 0x07, 0x00, 0x07, 0x00,  // 0x22 "
  0x14, 0x7F, 0x14, 0x7F, 0x14,  // 0x23 #
  0x24, 0x2A, 0x7F, 0x2A, 0x12,  // 0x24 $
  0x23, 0x13, 0x08, 0x64, 0x62,  // 0x25 %
  0x36, 0x49, 0x56, 0x20, 0x50,  // 0x26 &
  0x00, 0x08, 0x07, 0x03, 0x00,  // 0x27 '
  0x00, 0x1C, 0x22, 0x41, 0x00,  // 0x28 (
  0x00, 0x41, 0x22, 0x1C, 0x00,  // 0x29 )
  0x2A, 0x1C, 0x7F, 0x1C, 0x2A,  // 0x2A *
  0x08, 0x08, 0x3E, 0x08, 0x08,  // 0x2B +
  0x00, 0x80, 0x70, 0x30, 0x00,  // 0x2C ,
  0x08, 0x08, 0x08, 0x08, 0x08,  // 0x2D -
  0x00, 0x00, 0x60, 0x60, 0x00,  // 0x2E .
  0x20, 0x10, 0x08, 0x04, 0x02,  // 0x2F /
  0x3E, 0x51, 0x49, 0x45, 0x3E,  // 0x30 0
  0x00, 0x42, 0x7F, 0x40, 0x00,  // 0x31 1
  0x72, 0x49, 0x49, 0x49, 0x46,  // 0x32 2
  0x21, 0x41, 0x49, 0x4D, 0x33,  // 0x33 3
  0x18, 0x14, 0x12, 0x7F, 0x10,  // 0x34 4
  0x27, 0x45, 0x45, 0x45, 0x39,  // 0x35 5
  0x3C, 0x4A, 0x49, 0x49, 0x31,  // 0x36 6
  0x41, 0x21, 0x11, 0x09, 0x07,  // 0x37 7
  0x36, 0x49, 0x49, 0x49, 0x36,  // 0x38 8
  0x46, 0x49, 0x49, 0x29, 0x1E,  // 0x39 9
  0x00, 0x00, 0x14, 0x00, 0x00,  // 0x3A :
  0x00, 0x40, 0x34, 0x00, 0x00,  // 0x3B ;
  0x00, 0x08, 0x14, 0x22, 0x41,  // 0x3C <
  0x14, 0x14, 0x14, 0x14, 0x14,  // 0x3D =
  0x00, 0x41, 0x22, 0x14, 0x08,  // 0x3E >
  0x02, 0x01, 0x59, 0x09, 0x06,  // 0x3F ?
  0x3E, 0x41, 0x5D, 0x59, 0x4E,  // 0x40 @
  0x7C, 0x12, 0x11, 0x12, 0x7C,  // 0x41 A
```

```
  0x7F, 0x49, 0x49, 0x49, 0x36,  // 0x42 B
  0x3E, 0x41, 0x41, 0x41, 0x22,  // 0x43 C
  0x7F, 0x41, 0x41, 0x41, 0x3E,  // 0x44 D
  0x7F, 0x49, 0x49, 0x49, 0x41,  // 0x45 E
  0x7F, 0x09, 0x09, 0x09, 0x01,  // 0x46 F
  0x3E, 0x41, 0x41, 0x51, 0x73,  // 0x47 G
  0x7F, 0x08, 0x08, 0x08, 0x7F,  // 0x48 H
  0x00, 0x41, 0x7F, 0x41, 0x00,  // 0x49 I
  0x20, 0x40, 0x41, 0x3F, 0x01,  // 0x4A J
  0x7F, 0x08, 0x14, 0x22, 0x41,  // 0x4B K
  0x7F, 0x40, 0x40, 0x40, 0x40,  // 0x4C L
  0x7F, 0x02, 0x1C, 0x02, 0x7F,  // 0x4D M
  0x7F, 0x04, 0x08, 0x10, 0x7F,  // 0x4E N
  0x3E, 0x41, 0x41, 0x41, 0x3E,  // 0x4F O
  0x7F, 0x09, 0x09, 0x09, 0x06,  // 0x50 P
  0x3E, 0x41, 0x51, 0x21, 0x5E,  // 0x51 Q
  0x7F, 0x09, 0x19, 0x29, 0x46,  // 0x52 R
  0x26, 0x49, 0x49, 0x49, 0x32,  // 0x53 S
  0x03, 0x01, 0x7F, 0x01, 0x03,  // 0x54 T
  0x3F, 0x40, 0x40, 0x40, 0x3F,  // 0x55 U
  0x1F, 0x20, 0x40, 0x20, 0x1F,  // 0x56 V
  0x3F, 0x40, 0x38, 0x40, 0x3F,  // 0x57 W
  0x63, 0x14, 0x08, 0x14, 0x63,  // 0x58 X
  0x03, 0x04, 0x78, 0x04, 0x03,  // 0x59 Y
  0x61, 0x59, 0x49, 0x4D, 0x43,  // 0x5A Z
  0x00, 0x7F, 0x41, 0x41, 0x41,  // 0x5B [
  0x02, 0x04, 0x08, 0x10, 0x20,  // 0x5C backslash
  0x00, 0x41, 0x41, 0x41, 0x7F,  // 0x5D ]
  0x04, 0x02, 0x01, 0x02, 0x04,  // 0x5E ^
  0x40, 0x40, 0x40, 0x40, 0x40,  // 0x5F _
  0x00, 0x03, 0x07, 0x08, 0x00,  // 0x60 `
  0x20, 0x54, 0x54, 0x78, 0x40,  // 0x61 a
  0x7F, 0x28, 0x44, 0x44, 0x38,  // 0x62 b
  0x38, 0x44, 0x44, 0x44, 0x28,  // 0x63 c
  0x38, 0x44, 0x44, 0x28, 0x7F,  // 0x64 d
  0x38, 0x54, 0x54, 0x54, 0x18,  // 0x65 e
  0x00, 0x08, 0x7E, 0x09, 0x02,  // 0x66 f
  0x18, 0xA4, 0xA4, 0x9C, 0x78,  // 0x67 g
  0x7F, 0x08, 0x04, 0x04, 0x78,  // 0x68 h
  0x00, 0x44, 0x7D, 0x40, 0x00,  // 0x69 i
  0x20, 0x40, 0x40, 0x3D, 0x00,  // 0x6A j
  0x7F, 0x10, 0x28, 0x44, 0x00,  // 0x6B k
```

```
  0x00, 0x41, 0x7F, 0x40, 0x00,  // 0x6C l
  0x7C, 0x04, 0x78, 0x04, 0x78,  // 0x6D m
  0x7C, 0x08, 0x04, 0x04, 0x78,  // 0x6E n
  0x38, 0x44, 0x44, 0x44, 0x38,  // 0x6F o
  0xFC, 0x18, 0x24, 0x24, 0x18,  // 0x70 p
  0x18, 0x24, 0x24, 0x18, 0xFC,  // 0x71 q
  0x7C, 0x08, 0x04, 0x04, 0x08,  // 0x72 r
  0x48, 0x54, 0x54, 0x54, 0x24,  // 0x73 s
  0x04, 0x04, 0x3F, 0x44, 0x24,  // 0x74 t
  0x3C, 0x40, 0x40, 0x20, 0x7C,  // 0x75 u
  0x1C, 0x20, 0x40, 0x20, 0x1C,  // 0x76 v
  0x3C, 0x40, 0x30, 0x40, 0x3C,  // 0x77 w
  0x44, 0x28, 0x10, 0x28, 0x44,  // 0x78 x
  0x4C, 0x90, 0x90, 0x90, 0x7C,  // 0x79 y
  0x44, 0x64, 0x54, 0x4C, 0x44,  // 0x7A z
  0x00, 0x08, 0x36, 0x41, 0x00,  // 0x7B {
  0x00, 0x00, 0x77, 0x00, 0x00,  // 0x7C |
  0x00, 0x41, 0x36, 0x08, 0x00,  // 0x7D }
  0x02, 0x01, 0x02, 0x04, 0x02,  // 0x7E ~
  0x3C, 0x26, 0x23, 0x26, 0x3C,  // 0x7F
  0x1E, 0xA1, 0xA1, 0x61, 0x12,  // 0x80
  0x3A, 0x40, 0x40, 0x20, 0x7A,  // 0x81
  0x38, 0x54, 0x54, 0x55, 0x59,  // 0x82
  0x21, 0x55, 0x55, 0x79, 0x41,  // 0x83
  0x22, 0x54, 0x54, 0x78, 0x42,  // 0x84
  0x21, 0x55, 0x54, 0x78, 0x40,  // 0x85
  0x20, 0x54, 0x55, 0x79, 0x40,  // 0x86
  0x0C, 0x1E, 0x52, 0x72, 0x12,  // 0x87
  0x39, 0x55, 0x55, 0x55, 0x59,  // 0x88
  0x39, 0x54, 0x54, 0x54, 0x59,  // 0x89
  0x39, 0x55, 0x54, 0x54, 0x58,  // 0x8A
  0x00, 0x00, 0x45, 0x7C, 0x41,  // 0x8B
  0x00, 0x02, 0x45, 0x7D, 0x42,  // 0x8C
  0x00, 0x01, 0x45, 0x7C, 0x40,  // 0x8D
  0x7D, 0x12, 0x11, 0x12, 0x7D,  // 0x8E
  0xF0, 0x28, 0x25, 0x28, 0xF0,  // 0x8F
  0x7C, 0x54, 0x55, 0x45, 0x00,  // 0x90
  0x20, 0x54, 0x54, 0x7C, 0x54,  // 0x91
  0x7C, 0x0A, 0x09, 0x7F, 0x49,  // 0x92
  0x32, 0x49, 0x49, 0x49, 0x32,  // 0x93
  0x3A, 0x44, 0x44, 0x44, 0x3A,  // 0x94
  0x32, 0x4A, 0x48, 0x48, 0x30,  // 0x95
```

```
  0x3A, 0x41, 0x41, 0x21, 0x7A,  // 0x96
  0x3A, 0x42, 0x40, 0x20, 0x78,  // 0x97
  0x00, 0x9D, 0xA0, 0xA0, 0x7D,  // 0x98
  0x3D, 0x42, 0x42, 0x42, 0x3D,  // 0x99
  0x3D, 0x40, 0x40, 0x40, 0x3D,  // 0x9A
  0x3C, 0x24, 0xFF, 0x24, 0x24,  // 0x9B
  0x48, 0x7E, 0x49, 0x43, 0x66,  // 0x9C
  0x2B, 0x2F, 0xFC, 0x2F, 0x2B,  // 0x9D
  0xFF, 0x09, 0x29, 0xF6, 0x20,  // 0x9E
  0xC0, 0x88, 0x7E, 0x09, 0x03,  // 0x9F
  0x20, 0x54, 0x54, 0x79, 0x41,  // 0xA0
  0x00, 0x00, 0x44, 0x7D, 0x41,  // 0xA1
  0x30, 0x48, 0x48, 0x4A, 0x32,  // 0xA2
  0x38, 0x40, 0x40, 0x22, 0x7A,  // 0xA3
  0x00, 0x7A, 0x0A, 0x0A, 0x72,  // 0xA4
  0x7D, 0x0D, 0x19, 0x31, 0x7D,  // 0xA5
  0x26, 0x29, 0x29, 0x2F, 0x28,  // 0xA6
  0x26, 0x29, 0x29, 0x29, 0x26,  // 0xA7
  0x30, 0x48, 0x4D, 0x40, 0x20,  // 0xA8
  0x38, 0x08, 0x08, 0x08, 0x08,  // 0xA9
  0x08, 0x08, 0x08, 0x08, 0x38,  // 0xAA
  0x2F, 0x10, 0xC8, 0xAC, 0xBA,  // 0xAB
  0x2F, 0x10, 0x28, 0x34, 0xFA,  // 0xAC
  0x00, 0x00, 0x7B, 0x00, 0x00,  // 0xAD
  0x08, 0x14, 0x2A, 0x14, 0x22,  // 0xAE
  0x22, 0x14, 0x2A, 0x14, 0x08,  // 0xAF
  0x55, 0x00, 0x55, 0x00, 0x55,  // 0xB0
  0xAA, 0x55, 0xAA, 0x55, 0xAA,  // 0xB1
  0xFF, 0x55, 0xFF, 0x55, 0xFF,  // 0xB2
  0x00, 0x00, 0x00, 0xFF, 0x00,  // 0xB3
  0x10, 0x10, 0x10, 0xFF, 0x00,  // 0xB4
  0x14, 0x14, 0x14, 0xFF, 0x00,  // 0xB5
  0x10, 0x10, 0xFF, 0x00, 0xFF,  // 0xB6
  0x10, 0x10, 0xF0, 0x10, 0xF0,  // 0xB7
  0x14, 0x14, 0x14, 0xFC, 0x00,  // 0xB8
  0x14, 0x14, 0xF7, 0x00, 0xFF,  // 0xB9
  0x00, 0x00, 0xFF, 0x00, 0xFF,  // 0xBA
  0x14, 0x14, 0xF4, 0x04, 0xFC,  // 0xBB
  0x14, 0x14, 0x17, 0x10, 0x1F,  // 0xBC
  0x10, 0x10, 0x1F, 0x10, 0x1F,  // 0xBD
  0x14, 0x14, 0x14, 0x1F, 0x00,  // 0xBE
  0x10, 0x10, 0x10, 0xF0, 0x00,  // 0xBF
```

```
  0x00, 0x00, 0x00, 0x1F, 0x10,  // 0xC0
  0x10, 0x10, 0x10, 0x1F, 0x10,  // 0xC1
  0x10, 0x10, 0x10, 0xF0, 0x10,  // 0xC2
  0x00, 0x00, 0x00, 0xFF, 0x10,  // 0xC3
  0x10, 0x10, 0x10, 0x10, 0x10,  // 0xC4
  0x10, 0x10, 0x10, 0xFF, 0x10,  // 0xC5
  0x00, 0x00, 0x00, 0xFF, 0x14,  // 0xC6
  0x00, 0x00, 0xFF, 0x00, 0xFF,  // 0xC7
  0x00, 0x00, 0x1F, 0x10, 0x17,  // 0xC8
  0x00, 0x00, 0xFC, 0x04, 0xF4,  // 0xC9
  0x14, 0x14, 0x17, 0x10, 0x17,  // 0xCA
  0x14, 0x14, 0xF4, 0x04, 0xF4,  // 0xCB
  0x00, 0x00, 0xFF, 0x00, 0xF7,  // 0xCC
  0x14, 0x14, 0x14, 0x14, 0x14,  // 0xCD
  0x14, 0x14, 0xF7, 0x00, 0xF7,  // 0xCE
  0x14, 0x14, 0x14, 0x17, 0x14,  // 0xCF
  0x10, 0x10, 0x1F, 0x10, 0x1F,  // 0xD0
  0x14, 0x14, 0x14, 0xF4, 0x14,  // 0xD1
  0x10, 0x10, 0xF0, 0x10, 0xF0,  // 0xD2
  0x00, 0x00, 0x1F, 0x10, 0x1F,  // 0xD3
  0x00, 0x00, 0x00, 0x1F, 0x14,  // 0xD4
  0x00, 0x00, 0x00, 0xFC, 0x14,  // 0xD5
  0x00, 0x00, 0xF0, 0x10, 0xF0,  // 0xD6
  0x10, 0x10, 0xFF, 0x10, 0xFF,  // 0xD7
  0x14, 0x14, 0x14, 0xFF, 0x14,  // 0xD8
  0x10, 0x10, 0x10, 0x1F, 0x00,  // 0xD9
  0x00, 0x00, 0x00, 0xF0, 0x10,  // 0xDA
  0xFF, 0xFF, 0xFF, 0xFF, 0xFF,  // 0xDB
  0xF0, 0xF0, 0xF0, 0xF0, 0xF0,  // 0xDC
  0xFF, 0xFF, 0xFF, 0x00, 0x00,  // 0xDD
  0x00, 0x00, 0x00, 0xFF, 0xFF,  // 0xDE
  0x0F, 0x0F, 0x0F, 0x0F, 0x0F,  // 0xDF
  0x38, 0x44, 0x44, 0x38, 0x44,  // 0xE0
  0xFC, 0x4A, 0x4A, 0x4A, 0x34,  // 0xE1
  0x7E, 0x02, 0x02, 0x06, 0x06,  // 0xE2
  0x02, 0x7E, 0x02, 0x7E, 0x02,  // 0xE3
  0x63, 0x55, 0x49, 0x41, 0x63,  // 0xE4
  0x38, 0x44, 0x44, 0x3C, 0x04,  // 0xE5
  0x40, 0x7E, 0x20, 0x1E, 0x20,  // 0xE6
  0x06, 0x02, 0x7E, 0x02, 0x02,  // 0xE7
  0x99, 0xA5, 0xE7, 0xA5, 0x99,  // 0xE8
  0x1C, 0x2A, 0x49, 0x2A, 0x1C,  // 0xE9
```

```
    0x4C, 0x72, 0x01, 0x72, 0x4C,  // 0xEA
    0x30, 0x4A, 0x4D, 0x4D, 0x30,  // 0xEB
    0x30, 0x48, 0x78, 0x48, 0x30,  // 0xEC
    0xBC, 0x62, 0x5A, 0x46, 0x3D,  // 0xED
    0x3E, 0x49, 0x49, 0x49, 0x00,  // 0xEE
    0x7E, 0x01, 0x01, 0x01, 0x7E,  // 0xEF
    0x2A, 0x2A, 0x2A, 0x2A, 0x2A,  // 0xF0
    0x44, 0x44, 0x5F, 0x44, 0x44,  // 0xF1
    0x40, 0x51, 0x4A, 0x44, 0x40,  // 0xF2
    0x40, 0x44, 0x4A, 0x51, 0x40,  // 0xF3
    0x00, 0x00, 0xFF, 0x01, 0x03,  // 0xF4
    0xE0, 0x80, 0xFF, 0x00, 0x00,  // 0xF5
    0x08, 0x08, 0x6B, 0x6B, 0x08,  // 0xF6
    0x36, 0x12, 0x36, 0x24, 0x36,  // 0xF7
    0x06, 0x0F, 0x09, 0x0F, 0x06,  // 0xF8
    0x00, 0x00, 0x18, 0x18, 0x00,  // 0xF9
    0x00, 0x00, 0x10, 0x10, 0x00,  // 0xFA
    0x30, 0x40, 0xFF, 0x01, 0x01,  // 0xFB
    0x00, 0x1F, 0x01, 0x01, 0x1E,  // 0xFC
    0x00, 0x19, 0x1D, 0x17, 0x12,  // 0xFD
    0x00, 0x3C, 0x3C, 0x3C, 0x3C,  // 0xFE
    0x00, 0x00, 0x00, 0x00, 0x00   // 0xFF
};

#endif
```

8.3.4 main.cpp

```
#include "assets.h"
#include "font.h"
#include "tetris.hpp"
#include <fcntl.h>
#include <linux/input.h>
#include <sys/mman.h>
#include <unistd.h>
#include <cstdio>
#include <cstring>
#include <fstream>
#include <iomanip>
/*
#####################################
Memory Mapping Functions and Constants
```

```
####################################
*/

//Memory Map
constexpr off_t PHY_TM = 0xff200000; //Tile Map
constexpr off_t PHY_PA = 0xff202000; //Color Palette
constexpr off_t PHY_TS = 0xff204000; //Tile Set
static volatile uint8_t *TM,*PA,*TS;

//Map FPGA memory
static void map_fpga() {
   int fd = open("/dev/mem", O_RDWR | O_SYNC);
   if (fd < 0) {
       perror("mem"); _exit(1);
   }
   #define MAP(base,sz,ptr) \
       ptr = (uint8_t*) mmap(nullptr, sz, PROT_READ | PROT_WRITE, MAP_SHARED, fd,
base); \
       if (ptr == MAP_FAILED){perror("mmap"); _exit(1);}
   MAP(PHY_TM, 8192, TM)
   MAP(PHY_PA, 64, PA)
   MAP(PHY_TS, 16384, TS)
   close(fd);
}

//Load palette and tile graphics
static void load_assets() {
   for (int i = 0; i < 16; ++i) {
       uint32_t color = PALETTE24[i];
       PA[i*4+0] = color & 0xFF;
       PA[i*4+1] = (color >> 8) & 0xFF;
       PA[i*4+2] = (color >> 16) & 0xFF;
       PA[i*4+3] = 0;
   }
   {
       std::ifstream tf("tiles.hex");
       if (tf) {
           tf >> std::hex;  //Parse tileset data from hex format
           for (size_t i = 0; i < sizeof(TILESET); ++i) {
               unsigned int v;
               if (!(tf >> v)) {
                   //Tileset is malformed or too short, fall back to building tileset
```

```
                    build_tileset();
                    break;
                }
                TILESET[i] = static_cast<uint8_t>(v);
            }
        } else {
            //Couldn't open tileset file, fall back to building tileset
            build_tileset();
        }
    }
    memcpy((void*) TS, TILESET, 16384);
    memset((void*) TM, 0, 8192);
}

/*
#####################################
Helper Functions to Draw Objects
#####################################
*/

//Put tile in tile map
static inline void put(int col, int row, uint8_t tile) {
    TM[row * TM_STRIDE + col] = tile;
}

//Draw a rectangle using tiles
static void rect(int x0, int y0, int w, int h, uint8_t tile) {
    for (int y = y0; y < y0 + h; ++y)
        for (int x = x0; x < x0 + w; ++x) put(x, y, tile);
}

//Draw a frame using tiles
static void frame(int x0, int y0, int w, int h, uint8_t tile){
    for (int x = x0; x < x0 + w; ++x) put(x, y0, tile), put(x, y0 + h - 1, tile);
    for (int y = y0; y < y0 + h; ++y) put(x0, y, tile), put(x0 + w - 1, y, tile);
}

//Render char using font
static void draw_char(int col, int row, char ch) {
    // each ASCII code is 5 bytes wide in font5x7[]
    const unsigned char* bmp = font5x7 + (static_cast<unsigned char>(ch) * 5);
    for (int x = 0; x < 5; ++x) {
```

```
        unsigned char column = bmp[x];
        for (int y = 0; y < 7; ++y) {
            if (column & (1 << y)) {
                put(col + x, row + y, TILE_WHITE);
            }
        }
    }
}

//Render string using draw_char
static void draw_string(int col, int row, const char*str) {
    for (int i = 0; str[i]; ++i) draw_char(col + i * 6, row, str[i]);
}

//Clear area
static void clear_area(int col, int row, int w, int h) {
    rect(col, row, w, h, TILE_EMPTY);
}

/*
#####################################
Function to Render Tetris
#####################################
*/

//Draw playfield borders
static void draw_borders() {
    frame(PF_LEFT, PF_TOP, PF_WIDTH, PF_HEIGHT, TILE_WALL);
    frame(NEXT_COL - 1, NEXT_ROW - 1, 6, 6, TILE_WALL);
}

//Draw playfield
static void draw_playfield(const Tetris& t) {
    for (int y = 0; y < ROWS; ++y)
        for (int x = 0; x < COLS; ++x)
            put(PF_LEFT + 1 + x, PF_TOP + 1 + y, t.playfield(x,y));
}

//Draw ghost block
static void draw_ghost(const Tetris& t) {
    int x = t.get_px();
    int y = t.get_py();
```

```
    //Drop down until you would collide
    while (t.can_place(x, y + 1)) {
        y++;
    }

    //Grab the 4×4 mask already rotated into place
    Tetromino orient = t.get_rotated_piece();

    //Render with TILE_WHITE tiles
    for (int dy = 0; dy < 4; ++dy) {
        for (int dx = 0; dx < 4; ++dx) {
            if (orient.mask[dy][dx]) {
                put(PF_LEFT + 1 + x + dx, PF_TOP  + 1 + y + dy, TILE_WHITE);
            }
        }
    }
}

//Draw Tetromino piece
static void draw_piece(const Tetris& t) {
    t.for_each_block([](int x, int y, uint8_t tile){
        put(PF_LEFT + 1 + x, PF_TOP + 1 + y, tile);
    });
}

//Draw next Tetromino piece
static void draw_next(const Tetris& t) {

    rect(NEXT_COL,NEXT_ROW,4,4,TILE_EMPTY); //Clear next piece square
    t.for_each_next([](int x, int y, uint8_t tile){
        put(NEXT_COL + x, NEXT_ROW + y, tile);
    });
}

//Draw HUD
static void draw_hud(const Tetris& t)
{

    static int prev_score = -1;
    static int prev_lines = -1;
    static int prev_level = -1;
```

```
static int prev_score_len = 0;
static int prev_lines_len = 0;
static int prev_level_len = 0;

//Helper function to erase the previous number
auto erase_number = [](int col, int row, int len)
{
if (len == 0) return;

//Each char is 5 pixels wide so we advance 6 pixels
const int CHAR_W = 6;
const int CHAR_H = 7;
clear_area(col, row, len * CHAR_W, CHAR_H);
};

//Draw the HUD labels once per game
static bool labels_drawn = false;
static bool was_game_over = true; //Force labels on the first game

//Detect start of a new game after game over
if (!t.game_over() && was_game_over) {
    labels_drawn   = false; //Re-enable label drawing

//Reset HUD number caches so first frame redraws them
prev_score = -1;
prev_lines = -1;
prev_level = -1;
prev_score_len = 0;
prev_lines_len = 0;
prev_level_len = 0;
}

//Remember game over status for next frame
was_game_over = t.game_over();

//Draw the score, lines, and level labels once per game
if (!labels_drawn) {
    draw_string(HUD_COL, HUD_SCORE_ROW, "SCORE");
    draw_string(HUD_COL, HUD_LINES_ROW, "LINES");
    draw_string(HUD_COL, LEVEL_ROW,     "LEVEL");
    labels_drawn = true;
```

```
    }

    char buf[8];

    //Score
    if (t.score() != prev_score) {
        erase_number(HUD_COL + 40, HUD_SCORE_ROW, prev_score_len);

        prev_score = t.score();
        sprintf(buf, "%d", prev_score);
        prev_score_len = strlen(buf);

        draw_string(HUD_COL + 40, HUD_SCORE_ROW, buf);
    }

    //Lines
    if (t.lines() != prev_lines) {
        erase_number(HUD_COL + 40, HUD_LINES_ROW, prev_lines_len);

        prev_lines = t.lines();
        sprintf(buf, "%d", prev_lines);
        prev_lines_len = strlen(buf);

        draw_string(HUD_COL + 40, HUD_LINES_ROW, buf);
    }

    //Level
    if (t.get_level() != prev_level) {
        erase_number(HUD_COL + 40, LEVEL_ROW, prev_level_len);

        prev_level = t.get_level();
        sprintf(buf, "%d", prev_level);
        prev_level_len = strlen(buf);

        draw_string(HUD_COL + 40, LEVEL_ROW, buf);
    }
}

/*
######################################
Game Logic State Machine
######################################
```

```
*/

//Game logic state machine
enum State {START, PLAY, OVER};
static State state = START;

//Open USB Controller
static int open_controller() {
   struct input_id id;
   char path[64], name[256];
   for (int i = 0; i < 32; ++i) {
       snprintf(path, sizeof(path), "/dev/input/event%d", i);
       int fd = open(path, O_RDONLY | O_NONBLOCK);
       if (fd < 0) continue;

       //Get device name
       if (ioctl(fd, EVIOCGNAME(sizeof(name)), name) < 0)
           name[0] = '\0';
       //Get vendor/product ID
       if (ioctl(fd, EVIOCGID, &id) < 0)
           memset(&id, 0, sizeof(id));

       //Match on controller being used
       if (strcmp(name, "USB Gamepad") == 0 ||
           (id.vendor == 0x0079 && id.product == 0x0011))
       {
           printf("Using controller: %s (%s)\n", name, path);
           return fd;
       }
       close(fd);
   }
   return -1;
}

//Read Controller input
static void poll_input(Tetris& t, int fd) {
   struct input_event ev;

   //Read all pending events
   while (read(fd, &ev, sizeof(ev)) == sizeof(ev)) {
       switch (state) {
           case START:
```

```
            //Start button → PLAY
            if (ev.type == EV_KEY && ev.value == 1 && ev.code == 297) {
                state = PLAY;
                clear_area(0, 0, 80, 60);
            }
            break;

        case PLAY:
            //D-pad (ABS_HAT0X = code 0, ABS_HAT0Y = code 1)
            if (ev.type == EV_ABS) {
                if (ev.code == 0) { //left/right
                    if (ev.value == 0) t.move_left();
                    else if (ev.value == 255) t.move_right();
                }
                else if (ev.code == 1) { //down
                    if (ev.value == 255) t.soft_drop();
                }
            }
            //Buttons (EV_KEY + value == 1)
            else if (ev.type == EV_KEY && ev.value == 1) {
                switch (ev.code) {
                    case 288: //X
                    case 292: //L
                    case 293: //R
                        t.rotate();
                        break;
                    case 289: //A
                    case 291: //Y
                        t.soft_drop();
                        break;
                    case 290: //B
                        t.hard_drop();
                        break;
                    case 296: //Select
                        t.toggle_pause();
                        break;
                }
            }
            break;

        case OVER:
            //Start button resets the game
```

```
                if (ev.type == EV_KEY && ev.value == 1 && ev.code == 297) {
                    t.reset();
                    clear_area(0, 0, 80, 60);
                    state = PLAY;
                }
                break;
        }
    }
}

//Show start screen
static void show_start() {
    memset((void*)TM, 0, 8192);
    draw_string(10, 20, "TETRIS FPGA");
    draw_string(10, 40, "PRESS START");
    draw_string(10, 50, "TO START");
}

//Show game over screen
static void show_game_over(Tetris& t) {
    char buf[8];
    clear_area(0, 0, 80, 60);
    draw_string(10, 10, "GAME OVER");
    draw_string(10, 40, "START:");
    draw_string(20, 50, "RESTART");
    sprintf(buf, "%d", t.score());
    draw_string(10, 20, "SCORE");
    draw_string(50, 20, buf);
    sprintf(buf, "%d", t.lines());
    draw_string(10, 30, "LINES");
    draw_string(50, 30, buf);
}

//Main program loop
int main() {
    map_fpga();
    load_assets();
    int controller = open_controller(); if (controller < 0) {perror("controller");
return 1;}

    Tetris tetris;
    show_start();
```

```
    while(true) {
        poll_input(tetris, controller);
        if (state == PLAY) {
            tetris.step();
            draw_borders();
            draw_playfield(tetris);
            draw_ghost(tetris);
            draw_piece(tetris);
            draw_next(tetris);
            draw_hud(tetris);
            if (tetris.game_over()) {
                state = OVER;
                show_game_over(tetris);
            }
        }
        usleep(16666); //Frame timer for 60Hz
    }
}
```

8.3.5 Makefile

```
# Makefile

CXX         := g++
CXXFLAGS    := -std=c++17 -Wall -Wextra -I.
LDLIBS      := -lmpg123 -lao

# Executables
TETRIS_EXE := tetris
MUSIC_EXE  := audio
RUN_EXE    := runner

# Sources
TETRIS_SRC := main.cpp tetris.cpp
MUSIC_SRC  := audio.cpp
RUN_SRC    := runner.cpp

.PHONY: all run clean

all: $(TETRIS_EXE) $(MUSIC_EXE) $(RUN_EXE)
```

```
$(TETRIS_EXE): $(TETRIS_SRC)
	$(CXX) $(CXXFLAGS) -o $@ $^ $(LDLIBS)

$(MUSIC_EXE): $(MUSIC_SRC)
	$(CXX) $(CXXFLAGS) -o $@ $^ $(LDLIBS)

$(RUN_EXE): $(RUN_SRC)
	$(CXX) $(CXXFLAGS) -o $@ $^ -pthread

run: all
	./$(RUN_EXE)

clean:
	rm -f $(TETRIS_EXE) $(MUSIC_EXE) $(RUN_EXE)
```

8.3.6 runner.cpp

```
#include <iostream>
#include <cstdlib>
#include <pthread.h>

void* run_music(void*) {
    std::cout << "Starting music loop..." << std::endl;
    int ret = std::system("./audio");
    std::cout << "Music loop exited with code " << ret << std::endl;
    return nullptr;
}

void* run_game(void*) {
    std::cout << "Starting Tetris..." << std::endl;
    int ret = std::system("./tetris");
    std::cout << "Tetris exited with code " << ret << std::endl;
    return nullptr;
}

int main() {
    pthread_t music_thread, game_thread;

    if (pthread_create(&music_thread, nullptr, run_music, nullptr)) {
        std::cerr << "Error creating music thread" << std::endl;
        return 1;
    }
```

```
    if (pthread_create(&game_thread, nullptr, run_game, nullptr)) {
        std::cerr << "Error creating game thread" << std::endl;
        return 1;
    }

    pthread_join(music_thread, nullptr);
    pthread_join(game_thread, nullptr);

    std::cout << "Both music and game have exited." << std::endl;
    return 0;
}
```

8.3.7 tetris.cpp

```
#include "tetris.hpp"
#include <algorithm>
#include <array>
#include <random>

/*
Game logic based on open source Tetris cpp code by Nuruzzaman Milon:
(https://github.com/milon/Tetris)
SRS system logic based on Harddrop Wiki article: (https://harddrop.com/wiki/SRS)
*/

//Full Super-Rotation System (SRS) wall-kick tables
static const int SRS_KICKS_JLSTZ[4][5][2] = {
    { {0, 0}, {-1, 0}, {-1, 1}, {0, -2}, {-1, -2} },
    { {0, 0}, {1, 0}, {1, -1}, {0, 2}, {1, 2} },
    { {0, 0}, {1, 0}, {1, 1}, {0, -2}, {1, -2} },
    { {0, 0}, {-1, 0}, {-1, -1}, { 0, 2}, {-1, 2} }
};

static const int SRS_KICKS_I[4][5][2] = {
    { {0, 0}, {-2, 0}, {1, 0}, {-2, -1}, {1, 2} },
    { {0, 0}, {-1, 0}, {2, 0}, {-1,  2}, {2, -1} },
    { {0, 0}, {2, 0}, {-1, 0}, {2, 1}, {-1, -2} },
    { {0, 0}, {1, 0}, {-2, 0}, {1, -2}, {-2, 1} }
};


//Make Tetromino
```

```
static Tetromino make_piece(std::initializer_list<const char*> rows,
                            uint8_t tile,
                            PieceType type) {
   Tetromino t{};
   t.type = type;
   int row = 0;
   for (auto line : rows) {
       for (int col = 0; col < 4 && line[col]; ++col)
           if (line[col] == '#') t.mask[row][col] = tile;
       ++row;
   }
   return t;
}

//Define Tetromino shapes and colors
static const std::array<Tetromino, 7> SHAPES = {
   make_piece({"....","####","....","...."}, BLUE, PieceType::I), // I spawn
   make_piece({"....","#...","###.","...."}, RED, PieceType::J), // J spawn
   make_piece({"....","..#.","###.","...."}, YELLOW, PieceType::L), // L spawn
   make_piece({"....",".##.",".##.","...."}, MAG, PieceType::O), // O spawn
   make_piece({"....","..##",".##.","...."}, GREEN, PieceType::S), // S spawn
   make_piece({"....",".###","..#.","...."}, CYAN, PieceType::T), // T spawn
   make_piece({"....",".##.","..##","...."}, PURPLE, PieceType::Z) // Z spawn
};

// Rotate a single 4×4 Tetromino mask 90° clockwise
static Tetromino rot_right(const Tetromino& t) {
   Tetromino r{};
   r.type = t.type;
   for (int y = 0; y < 4; ++y)
       for (int x = 0; x < 4; ++x)
           r.mask[x][3 - y] = t.mask[y][x];
   return r;
}

//Rotate Tetromino a set number of times
Tetromino Tetris::rotate_piece(const Tetromino& t, int num_rot) const {
   int rcount = ((num_rot % 4) + 4) % 4;
   Tetromino q = t;
   for(int i = 0; i < rcount; ++i) {
       q = rot_right(q);
   }
```

```
    return q;
}

//Generate random Tetromino
static Tetromino rnd_piece() {
    static std::mt19937 gen{std::random_device{}()};
    static std::uniform_int_distribution<int>d(0, 6);
    return SHAPES[d(gen)];
}

//Set Tetromino
Tetris::Tetris() {
    cur = rnd_piece();
    nxt = rnd_piece();
    spawn();
}

//Spawn current Tetromino
void Tetris::spawn() {
    px = 5;
    py = 0;
    rot = 0;
    cur = nxt;
    nxt = rnd_piece();
    if (collision(px, py, cur, rot)) {
        over = true;
    }
}

//Calculate collisions
bool Tetris::collision(int nx, int ny,
                       const Tetromino& pc,
                       int r) const
{
    Tetromino p = rotate_piece(pc, r);
    for(int y = 0; y < 4; ++y) {
        for(int x = 0; x < 4; ++x) {
            if (!p.mask[y][x]) continue;
            int gx = nx + x;
            int gy = ny + y;

            // Check again going off the left/right or below bottom
```

```
            if (gx < 0 || gx >= COLS || gy >= ROWS)
                return true;

            // Check against overlapping
            if (gy >= 0 && field[gy][gx])
                return true;
        }
    }
    return false;
}

//Move Left Function
void Tetris::move_left() {
    if (!paused && !over && !collision(px - 1, py, cur, rot)) --px;
}

//Move Right Function
void Tetris::move_right() {
    if (!paused && !over && !collision(px + 1, py, cur, rot)) ++px;
}

//Rotate Function
void Tetris::rotate() {
    if (paused || over) return;

    int old_r = rot;
    int new_r = (old_r + 1) & 3;
    const int (*kicks)[2] = nullptr;

    //Determine kick table according to Tetromino type
    if (cur.type == PieceType::I) kicks = SRS_KICKS_I[old_r]; //I
    else if (cur.type == PieceType::O) { //O
        //O Tetrominos rotate in place
        rot = new_r;
        return;
    }
    else kicks = SRS_KICKS_JLSTZ[old_r]; //JLSTZ

    //Try each of the 5 SRS tests
    for (int i = 0; i < 5; ++i) {
        int dx = kicks[i][0], dy = kicks[i][1];
        if (!collision(px + dx, py + dy, cur, new_r)) {
```

```
            px += dx;
            py += dy;
            rot = new_r;
            return;
        }
    }
}

//Soft Drop Function
void Tetris::soft_drop() {
    if (!paused && !over && !collision(px, py + 1, cur, rot)) ++py;
}

//Hard Drop Function
void Tetris::hard_drop() {
    if (paused || over) return;
    while (!collision(px, py + 1, cur, rot)) ++py;
    lock_piece();
}

//Pause Function
void Tetris::toggle_pause() {
    if (!over) paused = !paused;
}

//Gravity for Tetrominos
void Tetris::step() {
    if (paused || over) return;

    //Max speed achieved at level 30 (speed actually maxes out at a lower since
interval is an int)
    int interval = std::max(1, 30 / level);

    if (++tick % interval == 0) {
        if(!collision(px, py + 1, cur, rot)) ++py;
        else lock_piece();
    }
}

//Lock Tetromino in place
void Tetris::lock_piece() {
    Tetromino p = rotate_piece(cur, rot);
```

```
    for(int y = 0; y < 4; ++y)
        for (int x = 0; x < 4; ++x)
            if (p.mask[y][x]) field[py + y][px + x] = p.mask[y][x];
    clear_lines();
    spawn();
}

//Clear lines
void Tetris::clear_lines() {
    int cleared = 0;
    for(int y = 0; y < ROWS; ++y) {
        bool line_full = true;
        for(int x = 0; x < COLS; ++x) {
            if (!field[y][x]) { line_full = false; break; }
        }
        if (line_full) {
            for(int k = y; k > 0; --k) {
                field[k] = field[k - 1];
            }
            field[0].fill(0);
            ++cleared;
        }
    }

    if (cleared > 0) {
        lines_cleared += cleared;

        //increase level every 10 lines cleared
        level = (lines_cleared / 10) + 1;

        // Tetris scoring: 1 = 100, 2 = 300, 3 = 500, 4 = 800
        static const int SCORE_TABLE[5] = { 0, 100, 300, 500, 800 };

        // mod 5 is a failsafe for if somehow more than 4 lines are cleared at once
        score_val += SCORE_TABLE[cleared % 5];

    }
}

//Reset game state
void Tetris::reset() {
    lines_cleared = 0;
```

```
    score_val = 0;
    over = 0;
    level = 1;
    tick  = 0;
    for (int i = 0; i < 20; ++i) field[i].fill(0); //Fill playfield with empty tiles
}

//Render helper functions for rendering each of the 4 tiles that make up a Tetromino
void Tetris::for_each_block(std::function<void(int, int, uint8_t)> cb) const
{
    Tetromino t = rotate_piece(cur,rot);
    for(int y = 0; y < 4; ++y)
        for(int x = 0; x < 4; ++x)
            if(t.mask[y][x]) cb(px + x, py + y, t.mask[y][x]);
}
void Tetris::for_each_next(std::function<void(int, int, uint8_t)> cb) const
{
    for(int y=0; y<4; ++y)
        for(int x=0; x<4; ++x)
            if(nxt.mask[y][x]) cb(x, y, nxt.mask[y][x]);
}
```

8.3.8 tetris.hpp

```
#ifndef TETRIS_HPP
#define TETRIS_HPP
#include <array>
#include <cstdint>
#include <functional>


enum class PieceType { I, J, L, O, S, T, Z };

constexpr int COLS = 15;
constexpr int ROWS = 20;

//The color to palette mapping (matches assets.h)
enum Cell : uint8_t {
    EMPTY = 0,
    BLUE = 2,
    GREEN = 3,
    RED = 4,
```

```
    CYAN = 5,
    YELLOW = 6,
    MAG = 7,
    PURPLE = 8
};

struct Tetromino {
    std::array<std::array<uint8_t,4>,4> mask{};
    PieceType type = PieceType::I;  // Default of I will be overwritten later in
make_piece function
};

class Tetris {
public:
    Tetris();
    void step();
    void move_left();
    void move_right();
    void rotate();
    void soft_drop();
    void hard_drop();
    void toggle_pause();
    void reset();


    uint8_t playfield(int x, int y) const {
        return field[y][x];
    }
    void for_each_block(std::function<void(int, int, uint8_t)> cb) const;
    void for_each_next (std::function<void(int, int, uint8_t)> cb) const;
    int score() const {
        return score_val;
    }
    int lines() const {
        return lines_cleared;
    }
    int get_level() const {
        return level;
    }
    int get_px() const {
        return px;
    }
```

```
    int get_py() const {
        return py;
    }
    int get_rot() const {
        return rot;
    }
    const Tetromino& get_cur() const {
        return cur;
    }
    bool game_over() const {
        return over;
    }
    bool can_place(int nx, int ny) const {
        return !collision(nx, ny, cur, rot);
    }
    Tetromino get_rotated_piece() const {
        return rotate_piece(cur, rot);
    }

private:
    std::array<std::array<uint8_t,COLS>,ROWS> field{};
    Tetromino cur, nxt;
    int   px = 5, py = 0, rot = 0;
    int   tick = 0;
    bool  paused=false, over=false;
    int   score_val = 0;
    int   lines_cleared = 0;
    int   level         = 1;

    void spawn();
    bool collision(int nx,int ny,const Tetromino& t,int r) const;
    void lock_piece();
    void clear_lines();
    Tetromino rotate_piece(const Tetromino& t,int r) const;
};

#endif
```

8.3.9 tiles.hex

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

07 07 07 07 07 07 07 07
07 07 07 07 07 07 07 07
07 07 07 07 07 07 07 07
07 07 07 07 07 07 07 07
07 07 07 07 07 07 07 07
07 07 07 07 07 07 07 07
07 07 07 07 07 07 07 07
07 07 07 07 07 07 07 07

0A 0A 0A 0A 0A 0A 0A 0A
0A 0A 01 01 01 01 01 0A
0A 01 01 01 01 01 01 0A
0A 01 01 01 01 01 01 0A
0A 01 01 01 01 01 01 0A
0A 01 01 01 01 01 01 0A
0A 01 01 01 01 01 01 0A
0A 0A 0A 0A 0A 0A 0A 0A

0A 0A 0A 0A 0A 0A 0A 0A
0A 0A 02 02 02 02 02 0A
0A 02 02 02 02 02 02 0A
0A 02 02 02 02 02 02 0A
0A 02 02 02 02 02 02 0A
0A 02 02 02 02 02 02 0A
0A 02 02 02 02 02 02 0A
0A 0A 0A 0A 0A 0A 0A 0A

0A 0A 0A 0A 0A 0A 0A 0A
0A 0A 03 03 03 03 03 0A
0A 03 03 03 03 03 03 0A
0A 03 03 03 03 03 03 0A
0A 03 03 03 03 03 03 0A
0A 03 03 03 03 03 03 0A
```

0A 03 03 03 03 03 03 0A
0A 0A 0A 0A 0A 0A 0A 0A

0A 0A 0A 0A 0A 0A 0A 0A
0A 0A 04 04 04 04 04 0A
0A 04 04 04 04 04 04 0A
0A 04 04 04 04 04 04 0A
0A 04 04 04 04 04 04 0A
0A 04 04 04 04 04 04 0A
0A 04 04 04 04 04 04 0A
0A 0A 0A 0A 0A 0A 0A 0A

0A 0A 0A 0A 0A 0A 0A 0A
0A 0A 05 05 05 05 05 0A
0A 05 05 05 05 05 05 0A
0A 05 05 05 05 05 05 0A
0A 05 05 05 05 05 05 0A
0A 05 05 05 05 05 05 0A
0A 05 05 05 05 05 05 0A
0A 0A 0A 0A 0A 0A 0A 0A

0A 0A 0A 0A 0A 0A 0A 0A
0A 0A 06 06 06 06 06 0A
0A 06 06 06 06 06 06 0A
0A 06 06 06 06 06 06 0A
0A 06 06 06 06 06 06 0A
0A 06 06 06 06 06 06 0A
0A 06 06 06 06 06 06 0A
0A 0A 0A 0A 0A 0A 0A 0A

0A 0A 0A 0A 0A 0A 0A 0A
0A 0A 08 08 08 08 08 0A
0A 08 08 08 08 08 08 0A
0A 08 08 08 08 08 08 0A
0A 08 08 08 08 08 08 0A
0A 08 08 08 08 08 08 0A
0A 08 08 08 08 08 08 0A
0A 0A 0A 0A 0A 0A 0A 0A

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

0e 0e 0e 0e 0e 0e 0e 0e
0e 0e 0e 0e 0e 0e 0e 0e
0e 0e 0e 0e 0e 0e 0e 0e
0e 0e 0e 0e 0e 0e 0e 0e

0e 0e 0e 0e 0e 0e 0e 0e
0e 0e 0e 0e 0e 0e 0e 0e
0e 0e 0e 0e 0e 0e 0e 0e
0e 0e 0e 0e 0e 0e 0e 0e

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
```

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00