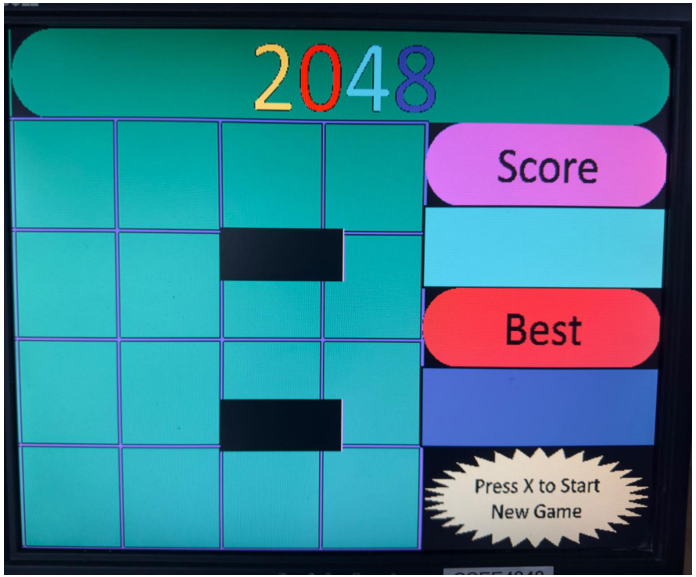
CSEE 4840 Final Project
2048

Kanghui Lin (kl3521) Yunhao Xing(yx2812) Jingtian Lin(jl6589)

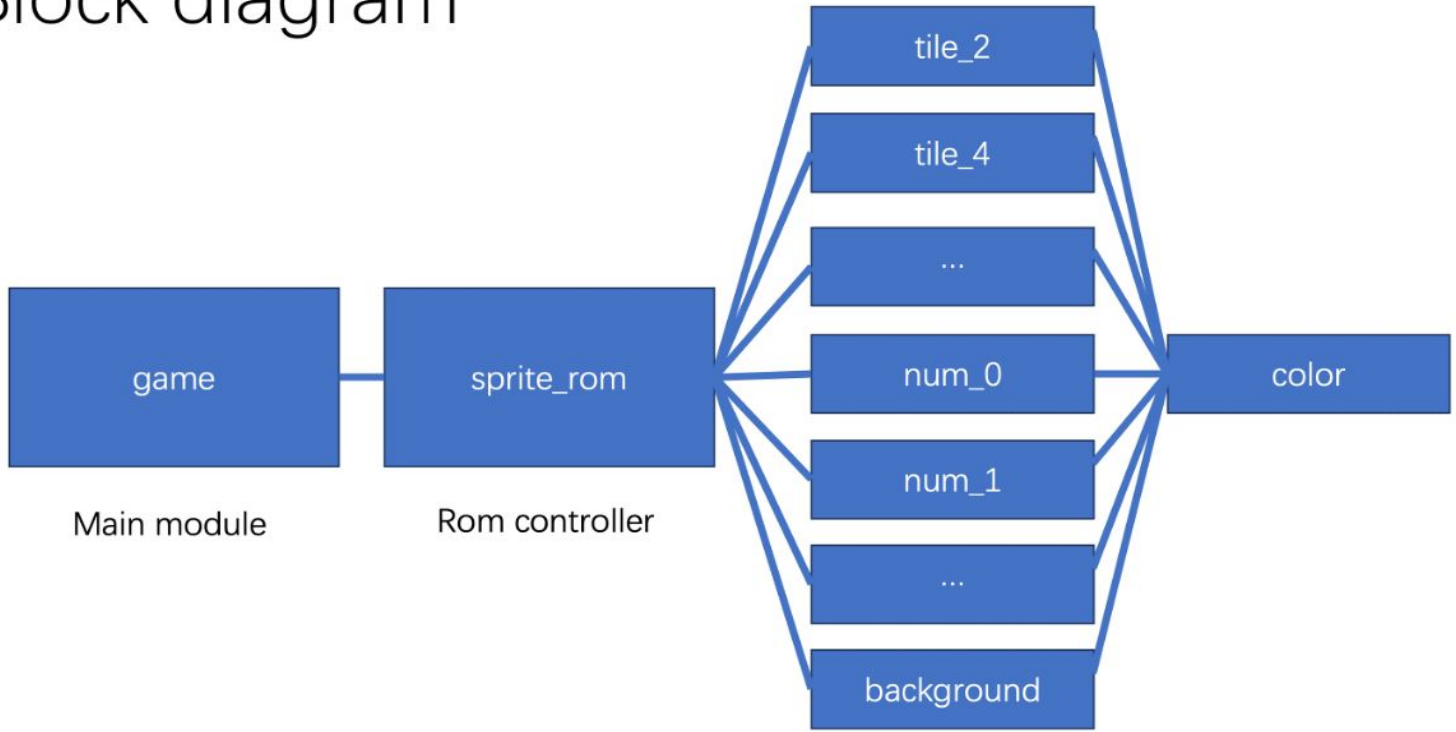
Introduction

A classical 2048 video game based on Altera DE1 board.

Use PS5 controller to merge the tile.



Block diagram



Introduction

What we have done:

Software:

•The game logic in software part is verified and tested.

Hardware：

•The game audio file is generated but not implemented to the board yet.

•The basic background and framework of the game can be displayed in VGA monitor

•The controller signal is tested but not implemented to the board yet.

Software

Board & Frames
the board is a 4*4 matrix where each position in the board represents the value held in the corresponding board.
The Frames are generated during player actions (shift_left, shift_right, shift_up, and shift_down), and tile generations.
There are in total of 6 frames generated per each player move.

Frame

frames are consists of 6 frame matrices;
frame1 entails what should the hardware render on first frame;
frame2 entails what should the hardware render on second frame
frame3 …
each frame matrix is 16*4 where 16 correspond to the 16 tiles on the board
for each tile there are 4 arguments where:
arg0 specifies the x position that the tile should be placed on
arg1 specifies the y position that the tile should be placed on
arg2 (value from 1-4) specifies the size of the tile that should be rendered where x \in (1-4) represents 0.25*x*100*100 (tile size)
arg3 specifies the value on the tile

Frames

We split the shift move into two parts. The first part is the slide which move all tiles to the leftmost / rightmost / farthest to the top / farthest to the bottom according to player move. The second part merges the tile with the adjacent tile that has the same value.

The frame on the screen is generated accordingly.
frame 1-2 is the animation of the slide
frame 3-4 is the animation of the merge
frame 5-6 is the generation of new tile

Frame

For instance, suppose a row (0, 2, 2, 0) performs a left_move() by player.
Frame 1-2 will generate the animation of sliding the two 2 tiles to the leftmost position. At frame 2 the screen will display (2, 2, 0, 0).
Frame 3-4 will generate the animation of merging two 2 tiles. At frame 4 the screen will display (4, 0, 0, 0)
then suppose a tile 4 is generated at the rightmost position of the row
Frame 5-6 will generate the animation of zooming out a new tile in the corresponding position. At frame 6 the screen will display (4, 0, 0, 4)

Hardware

Game:

- Main module of our game
- Supposed to handle audio and video parts
- For video part, use sprite_rom to get correct pixel info for our vga display

Game code:

```
logic           drawing, transparent;
logic [19:0]    current_score, best_score;
logic [23:0]    pix_out;
logic [31:0]    tiles_0, tiles_1, tiles_2, tiles_3,
                tiles_4, tiles_5, tiles_6, tiles_7,
                tiles_8, tiles_9, tiles_10, tiles_11,
                tiles_12, tiles_13, tiles_14, tiles_15;

sprite_rom sr(.clk(clk), .rst(reset), .hc(hcount), .vc(vcount),
              .tiles_0(tiles_0), .tiles_1(tiles_1), .tiles_2(tiles_2),
              .tiles_3(tiles_3), .tiles_4(tiles_4), .tiles_5(tiles_5),
              .tiles_6(tiles_6), .tiles_7(tiles_7), .tiles_8(tiles_8),
              .tiles_9(tiles_9), .tiles_10(tiles_10), .tiles_11(tiles_11),
              .tiles_12(tiles_12), .tiles_13(tiles_13), .tiles_14(tiles_14),
              .tiles_15(tiles_15), .current_score(current_score),
              .best_score(best_score),

              .drawing(drawing), .transparent(transparent) .pix_out(pix_out));
```

Sprite_ROM:

- A rom controller, integrate all roms so that the whole structure is more clear and comprehensive
- Fetch pixel info by different component types
- Include a state machine to render screen

Sprite_Rom coe:

```
module sprite_rom (
  always_ff @(posedge clk) begin
    else begin
      case (state)

        DRAW_TILE:begin
          if(!num_active) state <= WAIT;
          else state <= DRAW_NUM;

          if ((hc >= tiles_0[9:0] - 45 && hc <= tiles_0[9:0] + 45
            && vc >= tiles_0[18:10] - 45 && vc <= tiles_0[18:10] + 45)) begin
            tile_addr = (hc - (tiles_0[9:0] - 45)) +
                        (vc - (tiles_0[18:10] - 45)) << 6 +
                        (vc - (tiles_0[18:10] - 45)) << 4 +
                        (vc - (tiles_0[18:10] - 45)) << 3 +
                        (vc - (tiles_0[18:10] - 45)) << 1;
            tile_type = tiles_0[25:22];
          end
          else if ((hc >= tiles_1[9:0] - 45 && hc <= tiles_1[9:0] + 45
            && vc >= tiles_1[18:10] - 45 && vc <= tiles_1[18:10] + 45)) begin
            tile_addr = (hc - (tiles_1[9:0] - 45)) +
                        (vc - (tiles_1[18:10] - 45)) << 6 +
                        (vc - (tiles_1[18:10] - 45)) << 4 +
```

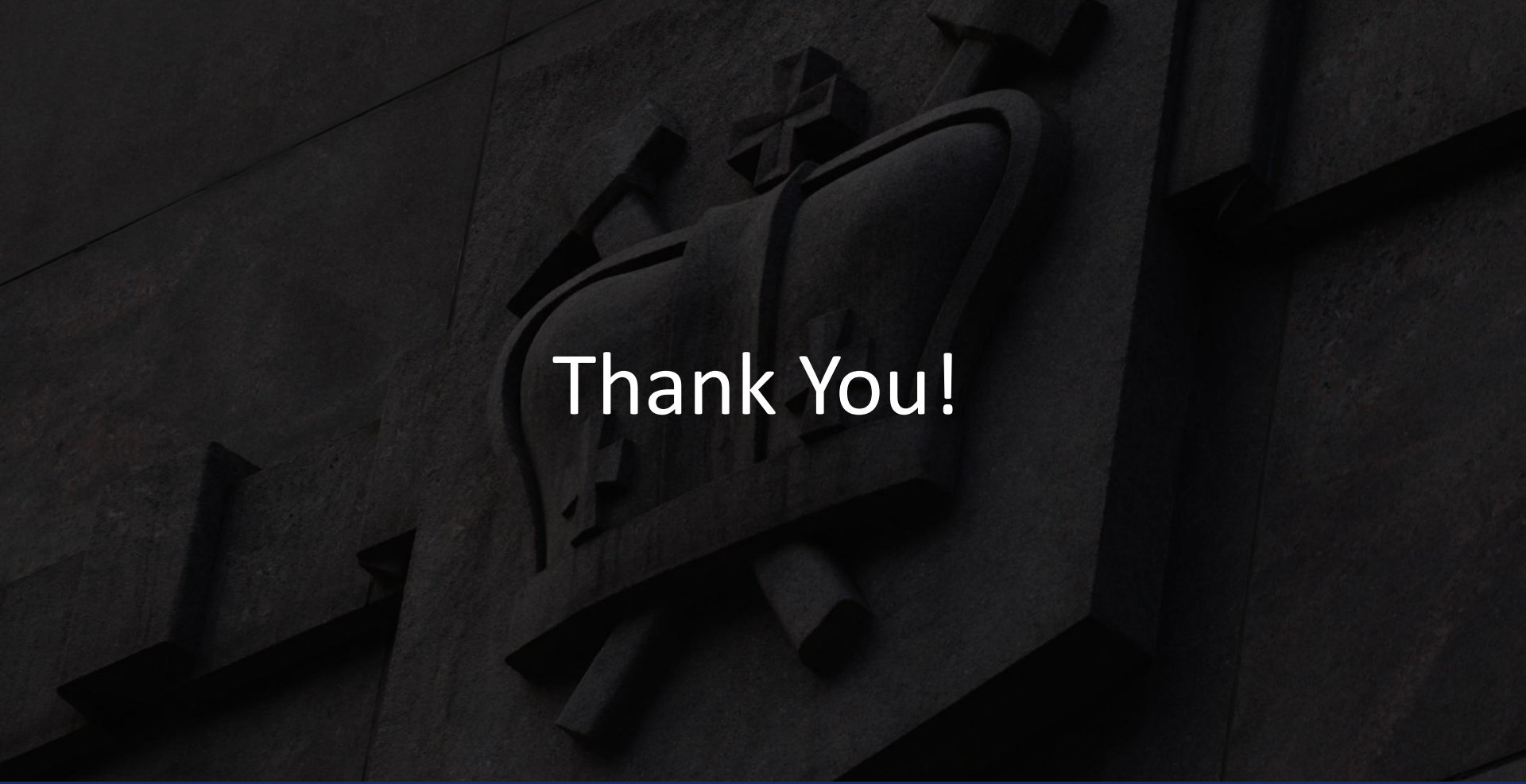
Hardware

ROM: •Init roms using .mif files

•.mif files are generated by Python program using images

•Except for color module, each rom takes an address and provides color type

•For color module, it takes the color type and rom type as the address and provides the actual color info (RGB value)


Thank You!