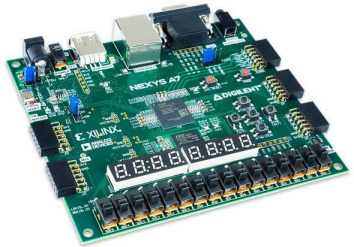
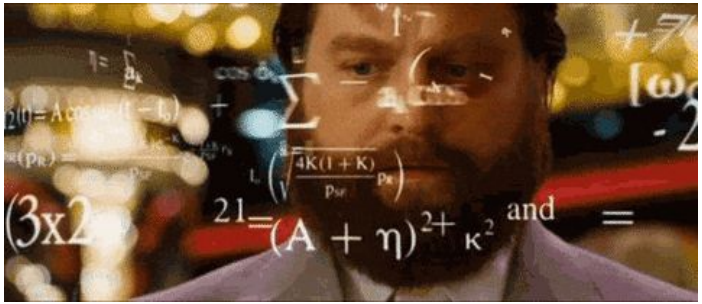
2048

Team 17
Christian Arthur, Youssef Atti,
Asbel Fontanez, Abdullah Gramish

Goals and Motivations

- Something fun but also challenging with Verilog and algorithms
- Recreate the 2048 game on an FPGA
  - What if you only had an FPGA!!?
  - Different platform to play 2048





Functionality

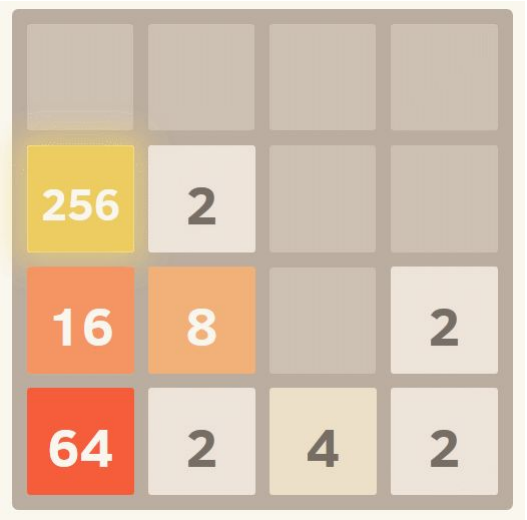
- Two blocks placed randomly with a value of 2 or 4
- When boxes with the same value are moved into each other, they will combine and their values will add up
- After each move a random valued block will be placed on the grid
- If user gets a 2048 block they WIN
- If there are no more possible moves GAME is OVER

2048

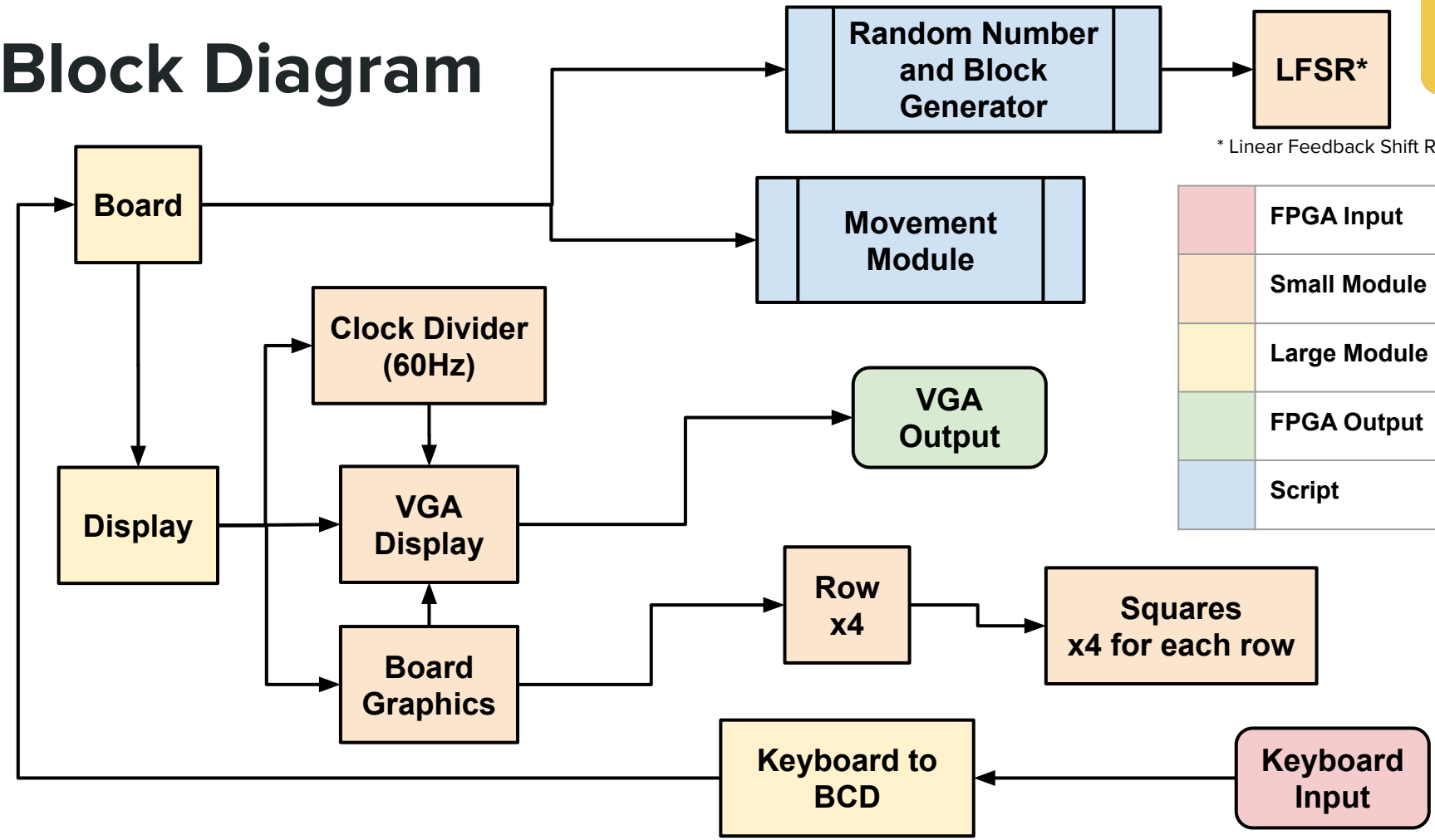
Specifications

- Specs:
  - 2048 is played on a 4x4 game board
  - The game would also receive user inputs to ‘slide’ the number boxes all together in a certain direction (up, down, left, and right)

- Constraints:
  - Directional Input (Keyboard)
  - External display (VGA)



Block Diagram



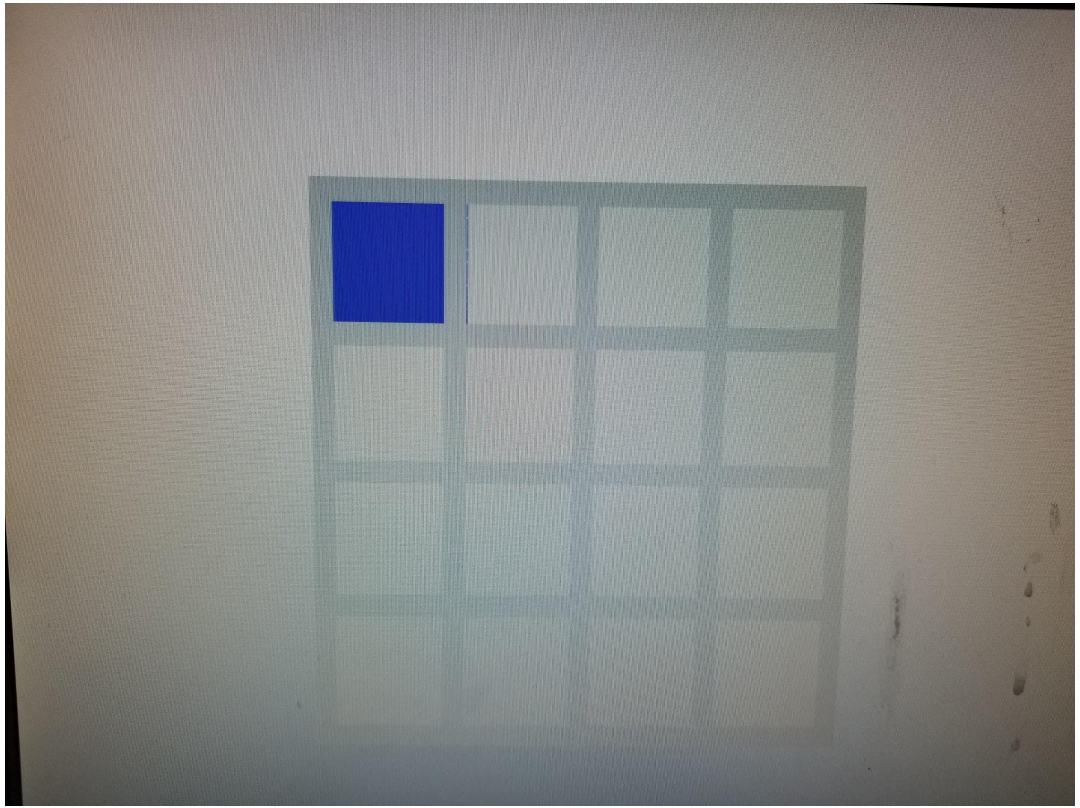
Code Snippet (vga_controller)

```
CLK60HZ rr(.CLK100MHZ(CLK100MHZ), .stb(pix_stb));
vga640x480 display(.CLK100MHZ(CLK100MHZ), .pix_stb(pix_stb), .reset(~CPU_RESETN),
                   .hs(VGA_HS), .vs(VGA_VS), .de(de), .x(cx), .y(cy));
board_disp board_graphics(.CLK100MHZ(CLK100MHZ), .data(data), .cx(cx), .cy(cy), .pix_stb(pix_stb),
                          .board_r(board_color[11:8]), .board_g(board_color[7:4]),
                          .board_b(board_color[3:0]), .row(row), .col(col), .draw(board_draw));

assign VGA_R = de ? cur_r: 4'h0;
assign VGA_G = de ? cur_g: 4'h0;
assign VGA_B = de ? cur_b: 4'h0;

always @(posedge pix_stb)begin
    if(board_draw)begin
        cur_r <= board_color[11:8];
        cur_g <= board_color[7:4];
        cur_b <= board_color[3:0];
    end else begin
        cur_r <= 4'hf;
        cur_g <= 4'hf;
        cur_b <= 4'he;
    end
```

Code Snippet (board_move)

```
case(direction)
    0: begin //move board left
            for(i = 3; i > 0; i = i - 1)begin
                for(j = 3; j > 0; j = j - 1)begin
                    if(newBoard[i][j-1] == 0)begin
                        newBoard[i][j-1] <= newBoard[i][j];
                        newBoard[i][j] <= 0;
                        moving <= 1;
                    end else if (newBoard[i][j-1] == newBoard[i][j])begin
                        newBoard[i][j-1] <= newBoard[i][j-1] + newBoard[i][j];
                        newBoard[i][j] <= 0;
                        moving <= 1;
                    end
                end
            end
       end
```

Project Visualization (start screen)



Successes

As of December 7, 2020

A more “complex” version of 2048 game to the FPGA

Enabling the use of:

- The VGA output to display the game board
- Keyboard inputs

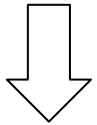
Notable Successes in...

- Spawning random valued blocks in random empty cells
- Displaying the grid on the VGA output

Successes

More Notable Successes...

- The usage of a simple ‘square’ module to build a row of 4 cells, 4 rows are combined with a big square to make the game board



Generating UI using only 2 modules that with some further logic can change cell colors depending on the board cells value

- Usage of one-hot encoding
  - We use multi-bit register to represent parts of the screen
  - The highest bot represent the object on the very top of the screen while the lowest bit represents the background
  - Then reading this register to decide color, it is read in descending order and depending which bit is high first tells the program to draw that objects color on the screen

Failures

As of December 7, 2020

Dynamically displaying text (numbers for scores and block values) is quite difficult, so the current compromise is combining same color blocks rather than blocks with the same values

Animation of the block movements is not perfect yet with it ‘blinking’ or ‘jumping’ from one cell to another (debugging is currently being done to fix this)

Failures

As of December 9, 2020

Clock issues in addition to issues with blocking vs non-blocking statements are messing up the data for the board

- As we implemented the move block logic we started to run into issues with the clock being to slow when using blocking statements
- Our first solution was to switch to non-blocking statements but this causes the issue where logic may be activated improperly due to the parallelization of non-blocking statements (timing issues)
- Our next solution was to try to do some moving logic during the blanking period while other moving logic occurred at the very beginning of the non-blanking period this still prove to be futile

Our Solution

WILD WEST EDITION

2048

Our Solution

- ‘Modified’ Game Mechanics from 2048
  - Same controls and basic mechanics from the basic 2048 game with some twists
  - The game turns into a puzzle game! The goal is to combine blocks of the same color to not allow the game board to be flooded by blocks
  - Combining these blocks become difficult overtime with the introduction of multi-color blocks and even invisible blocks
- Implementation of a ‘failure’ as a feature
  - Utilizing the ‘randomness’ created from from the clock and non-blocking statements to make difficult puzzle game that is endless and you can only lose

Thank you for listening!
Any questions?