Typing Turmoil: Escape from Photonics

Carlo Lanza, Frederick Grass,
Arnouv Nayana, Ryan Malone

Goal/Motivation

- Utilize FPGA, VGA, and keyboard interaction to create a fun way to practice typing
  - Improving the speed and accuracy of the user
- To provide people of all ages a fun way to improve their typing abilities
  - This requires an easy to use system that allows the user to focus on typing

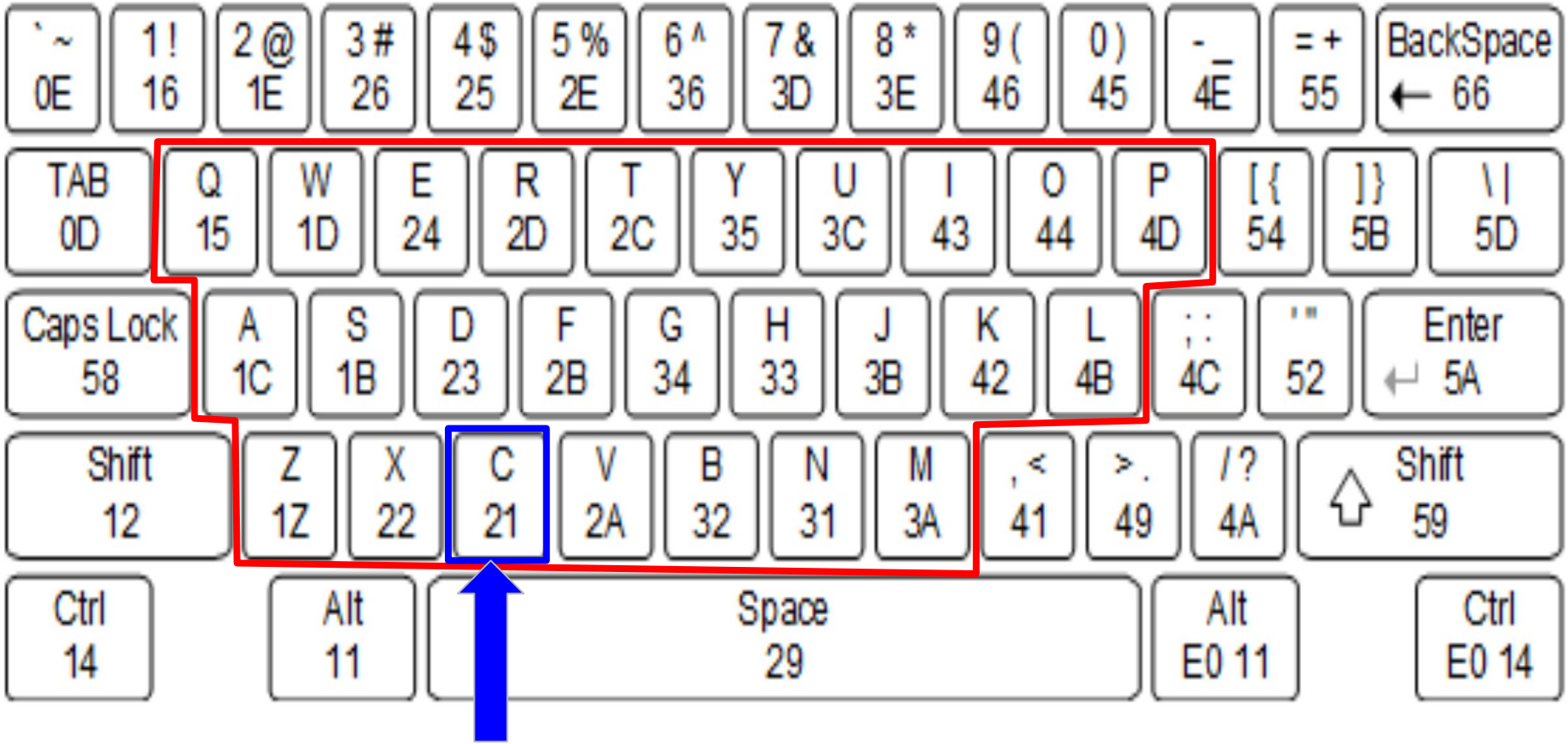
Functionality

- Words are displayed to the screen on the VGA display and the player must type the letters from left to right.
- An incorrect letter results in a ‘game over’ screen, displaying their score and their words per minute pace

Specifications/Constraints

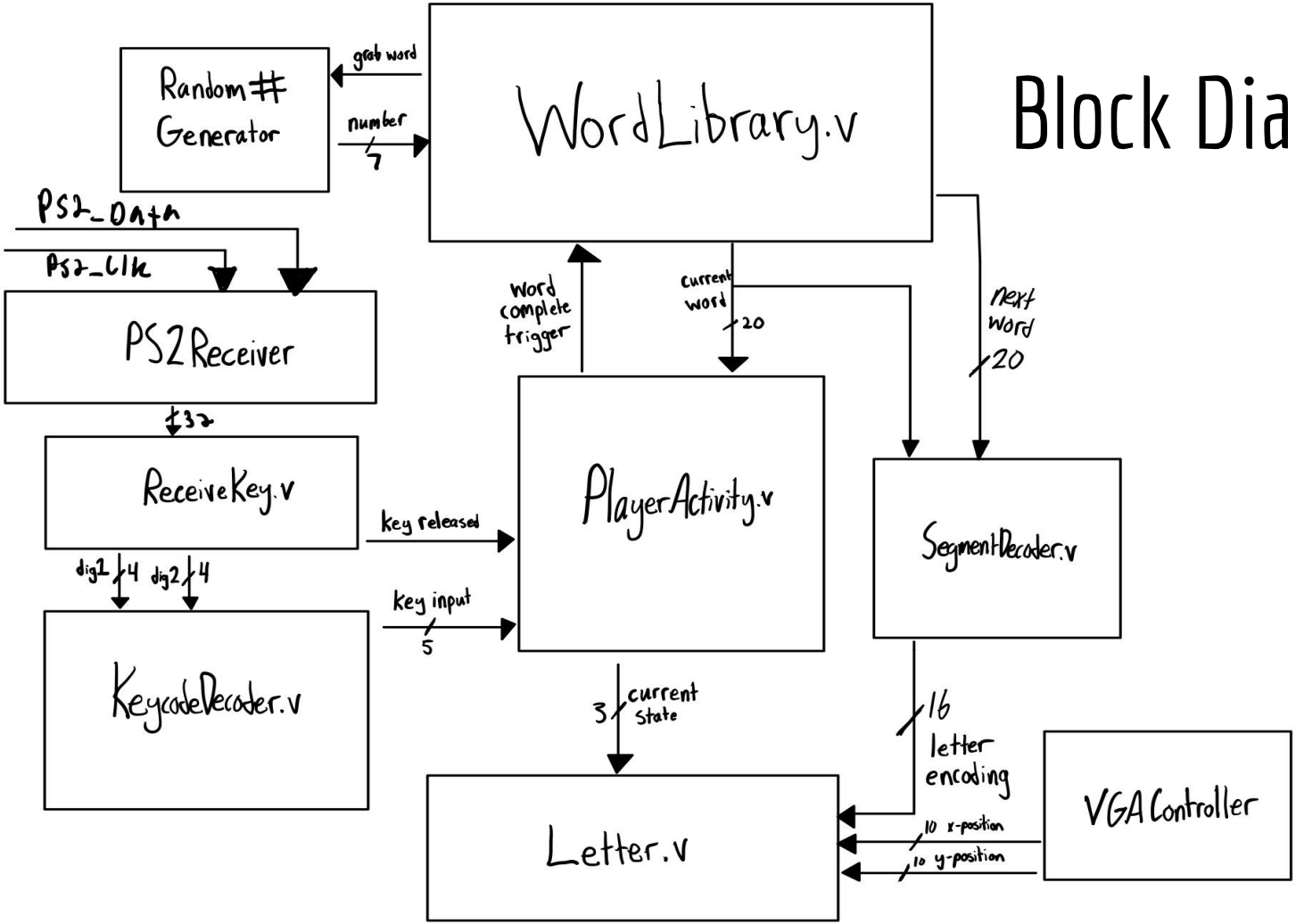
- FPGA:
  - Words hard coded
  - Has limited memory
- Input delays
  - VGA must refresh/react to user extremely fast
  - Debouncer needs to handle rapid inputs
- Keyboard
  - Cannot change how data sent
  - Must decode into understandable sequence of bits
  - Picks a most recent for simultaneous presses

keyboard[31:0]=prev1[3:0] prev2[3:0] kr[15:0]dig1[3:0]dig2[3:0]



dig1=0010 dig2=0001 → Char = 5'b00101

Block Diagram:



Verilog Snippet (WordLibrary.v)

```
`timescale 1ns / 1ps

module WordDelivery(
    clk,
    wordComplete,
    reset,

    currentWord,
    nextWord
    );

input clk;
input wordComplete;
input reset;

output reg [19:0] currentWord;
output reg [19:0] nextWord;

reg [19:0] word [0:99];
wire [6:0] nextWordLocation;

RandomNumberGenerator randomizer(.grabWord(wordComplete), .reset(reset), .random_num(nextWordLocation));

always @ (posedge wordComplete) begin
    currentWord = nextWord;
    nextWord = word[nextWordLocation];
end
```

```
`timescale 1ns / 1ps


module RandomNumberGenerator(
    input grabWord,
    input reset,
    output reg [6:0] random_num // 7 bits to represent numbers up to 100
    );

    // LFSR internal state (using a 7-bit register for simplicity)
    reg [6:0] lfsr;

    // Taps for 7-bit LFSR for maximal length sequence
    wire feedback = lfsr[6] ^ lfsr[5] ^ lfsr[3] ^ lfsr[2];

    initial begin
        lfsr <= 7'b0000001;
    end

    always @(posedge grabWord or posedge reset) begin
        if (reset) begin
            // Reset the LFSR to a non-zero value
            lfsr <= 7'b0000001;
        end else begin
            // Shift the LFSR and insert the feedback bit
            lfsr <= {lfsr[5:0], feedback};
        end
    end

    // Scale LFSR value to 1-100 range
    always @(lfsr) begin
        random_num <= (lfsr % 100) + 1;
    end

endmodule
```

WordLibrary.v

```
reg [19:0] word [0:99];

initial begin

    word[0] = 20'b00001100100000010111;
    word[1] = 20'b00010101010000000001;
    word[2] = 20'b01010100100101100100;
    word[3] = 20'b01111100101001001011;
    word[4] = 20'b00010000000111101111;
    word[5] = 20'b00011100100011110110;
    word[6] = 20'b10011100101010101011;
    word[7] = 20'b10101100101001010000;
    word[8] = 20'b10111001000001001000;
    word[9] = 20'b00000011111011010010;
    word[10] = 20'b11101001001011010111;
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    word[90] = 20'b01111000001010101011;
    word[91] = 20'b10000100110101001000;
    word[92] = 20'b10001000001001100100;
    word[93] = 20'b10010100101110100100;
    word[94] = 20'b10011110000001001011;
    word[95] = 20'b10101100100000010000;
    word[96] = 20'b10110001000010010011;
    word[97] = 20'b10111110001011001011;
    word[98] = 20'b11000011111000100000;
    word[99] = 20'b11001001000010010101;
end
```

Word encoding

```
letter_bit = {
    "a": "00000",
    "b": "00001",
    "c": "00010",
    "d": "00011",
    "e": "00100",
    "f": "00101",
    "g": "00111",
    "h": "01000",
    "i": "01001",
    "j": "01010",
    "k": "01011",
    "l": "01111",
    "m": "10000",
    "n": "10001",
    "o": "10010",
    "p": "10011",
    "q": "10100",
    "r": "10101",
    "s": "10110",
    "t": "10111",
    "u": "11000",
    "v": "11001",
    "w": "11010",
    "x": "11011",
    "y": "11100",
    "z": "11101",
}
```

type_racer_words_bits.txt

```
00001100100000010111
00010101010000000001
01010100100101100100
01111100101001001011
00010000000111101111
00011100100011110110
10011100101010101011
10101100101001010000
10111001000001001000
00000011111011010010
11101001001011010111
11100000001010110001
10100110000100110011
01010010011000111011
11010001000111100011
10010001110111100100
10000001000111100011
01111110001011001000
01010100110111001000
01010100100111110111
```

type_racer_words.txt

```
boat
crab
joke
look
call
dogs
pork
room
tech
also
zest
yarn
quip
jinx
weld
ogle
meld
lush
kite
jolt
```

PlayerActivity.v

- currentWord Received from WordLibrary
  - Parameterized into Letters
- FSM operated with each letter check corresponding to a state

Parameterization

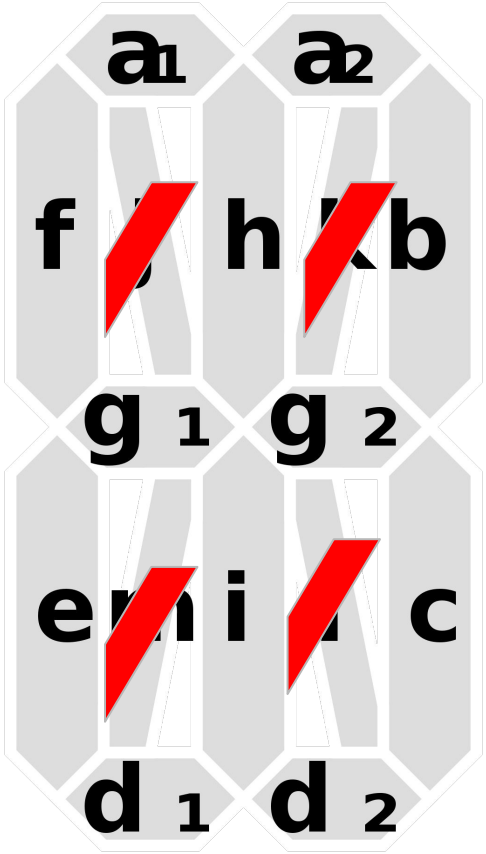
```
module PlayerActivity(
    currentWord,
    keystroke,
    keyReleased,

    wordComplete,
    gameOver
    );

    input [19:0] currentWord;
    input [4:0] keystroke;
    input keyReleased;

    output reg wordComplete;
    output reg gameOver;

    wire [4:0] L1 = currentWord[19:15];
    wire [4:0] L2 = currentWord[14:10];
    wire [4:0] L3 = currentWord[9:5];
    wire [4:0] L4 = currentWord[4:0];

    reg [1:0] currentState;

    initial begin
        currentState = 1;
    end
```

FSM

```
always @ (posedge keyReleased) begin
    case(currentState)
        1: begin
            if(keystroke == L1) begin
                currentState = 2;
                wordComplete = 0;
            end else begin
                gameOver = 1;
            end

        end

        2: begin
            if(keystroke == L2) begin
                currentState = 3;
                wordComplete = 0;
            end else begin
                gameOver = 1;
            end
        end

        3: begin
            if(keystroke == L3) begin
                currentState = 4;
                wordComplete = 0;
            end else begin
                gameOver = 1;
            end
        end

        4: begin
            if(keystroke == L4) begin
                currentState = 1;
                wordComplete = 1;
            end else begin
                gameOver = 1;
            end
        end

    endcase
```

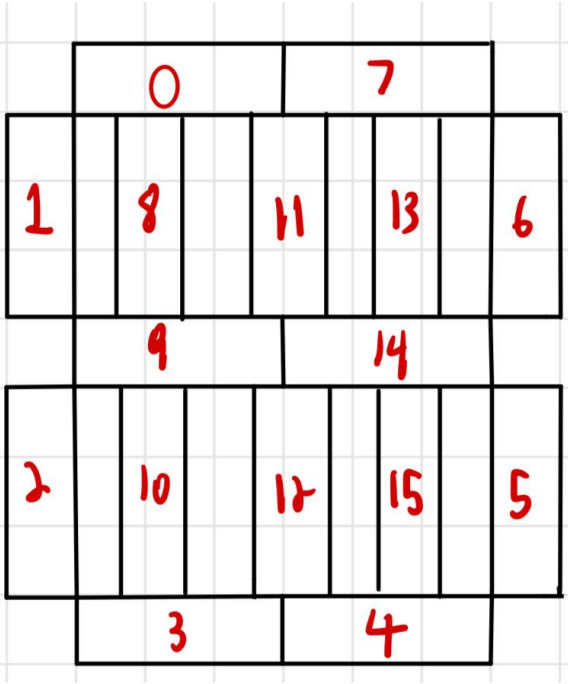
VGA Display

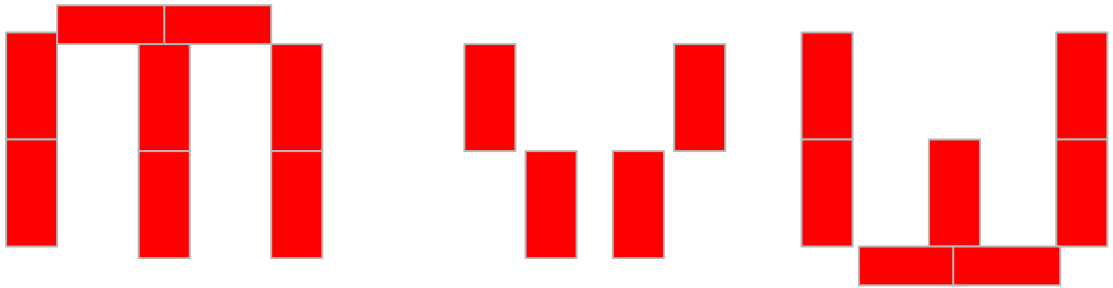
- Utilize a 16-segment display format for the letters
- Use letter.v to display the given letter based on a 16-bit encoding



Modified 16 Segment Display

```
always @(char) begin
    case (char)
        // Assign 16-bit values for each alphabet letter
        5'b00000: segments = 16'b1110011101011010; // A
        5'b00001: segments = 16'b1111111101011010; // B
5'b01100: segments = 16'b1110011100011000; // M
5'b01101: segments = 16'b1110111000011000; // N
5'b10101: segments = 16'b0100001000100001; // V
5'b10110: segments = 16'b0111111000001000; // W
```





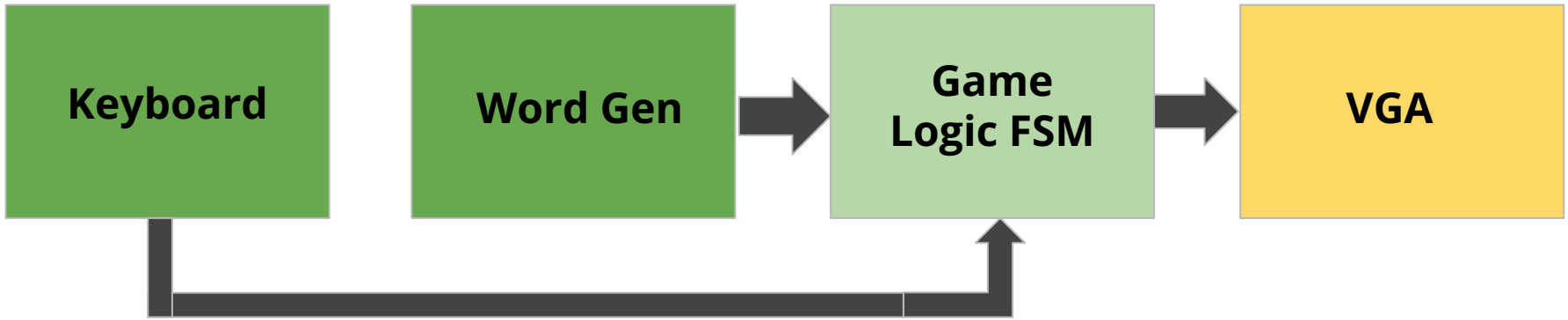
Letter.v

```
always @ (xpos, ypos) begin
    if (xpos > basex + width && xpos < basex + length + width && ypos > basey && ypos < basey + width && encoding[15] == 1) begin // 0
        red_reg <= 4'hF;
        blue_reg <= 0;
        green_reg <= 0;
        change_reg <= 1;
     end
     else if (xpos > basex && xpos < basex + width && ypos > basey + width && ypos < basey + length + width && encoding[14] == 1) begin // 1
        red_reg <= 4'hF;
        blue_reg <= 0;
        green_reg <= 0;
        change_reg <= 1;
     end
     else if (xpos > basex && xpos < basex + width && ypos > basey + length + width && ypos < basey + doubleLength + width && encoding[13] == 1) begin // 2
        red_reg <= 4'hF;
        blue_reg <= 0;
        green_reg <= 0;
        change_reg <= 1;
     end
```

```
assign red = (change_reg == 1) ? red_reg : red;
assign blue = (change_reg == 1) ? blue_reg : blue;
assign green = (change_reg == 1) ? green_reg : green;
assign change = change_reg;
```

Successes - so far

- Creating and encoding a library of words
- Generating a pseudo-random index to access words array
- Keyboard code expanded to accept all letters, track key released/depressed
- Working game logic: player activity checks accuracy of stroke



Challenges/Next Steps

- Push to the VGA Display
- Create logic to restart the game
- Implement timer to analyze and present words per minute

Thank you