EE2016: Experiment 2

Group:19

Sparsh Gupta EE23B117
Charan Srikkanth EE23B127
Kaushik Iyer EE23B135

August 21, 2024

1 Introduction

This experiment focuses on designing and simulating a Wallace Multiplier using Xilinx Vivado.This is done by multiplying two four-bit numbers and getting the eight-bit result.It also focuses on how to use the LCD display in the FPGA development board.

2 Objectives

The primary objectives of this experiment are:

1. Implement Wallace multiplier using Xilinx Vivado for two 4bit numbers.
2. Implement the 8bit output on the LEDs present in the FPGA.
3. Make the LCD display work and print characters on it.
4. BONUS: Display on the LED.On the first line ”PRODUCT =”.On the next line the output product of the two 4bit numbers.

3 Wallace Multiplier

Let us assume we are multiplying two four bit numbers A and B.Wallace multiplier involves taking product of each digit of A and B and adding them according to their place value.(by using Full and Half adders.) It involves the following:

1. First find the AND between all four bits of A with all four bits of B.Place them according to their place values after multiplying.
2. You will notice that for a given place there will be more than 3 bits to add so we will need many iterations of using Full and Half adders as seen in the fig.

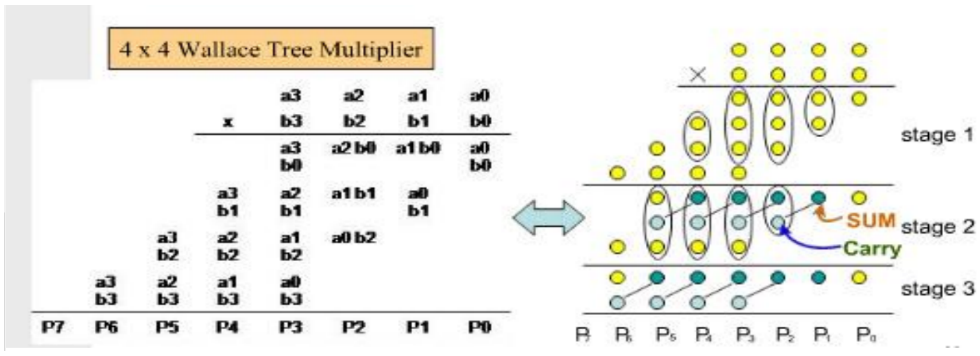

Figure 1: OUTPUT

3.1 Verilog Implementation

```
module wallace_multiplier(output wire[7:0] prod, input wire[3:0] a, input wire[3:0] b);
    wire p00, p01, p02, p03, p10, p11, p12, p13, p20, p21, p22, p23, p30, p31, p32, p33;
    wire s0, s1, s2, s3, s4, s5, s6, s7;
    wire c0, c1, c2, c3, c4, c5, c6, c7;
    wire cm3, cm4, cm5, cm6, cm7;

    and(p00, a[0], b[0]);
    and(p01, a[0], b[1]);
    and(p02, a[0], b[2]);
    and(p03, a[0], b[3]);
    and(p10, a[1], b[0]);
    and(p11, a[1], b[1]);
    and(p12, a[1], b[2]);
    and(p13, a[1], b[3]);
    and(p20, a[2], b[0]);
    and(p21, a[2], b[1]);
    and(p22, a[2], b[2]);
    and(p23, a[2], b[3]);
    and(p30, a[3], b[0]);
    and(p31, a[3], b[1]);
    and(p32, a[3], b[2]);
    and(p33, a[3], b[3]);

    half_adder h0(s0, c0, p01, p10);
    full_adder f0(s1, c1, p02, p11, p20);
    full_adder f1(s2, c2, p03, p12, p21);
    half_adder h1(s3, c3, p13, p22);

    half_adder h2(s4, c4, s1, c0);
    full_adder f2(s5, c5, s2, c1, p30);
    full_adder f3(s6, c6, s3, c2, p31);
    full_adder f4(s7, c7, p23, c3, p32);

    assign prod[0]=p00;
    assign prod[1]=s0;
    assign prod[2]=s4;
    half_adder h3(prod[3], cm3, s5, c4);
    full_adder f5(prod[4], cm4, s6, c5, cm3);
    full_adder f6(prod[5], cm5, s7, c6, cm4);
    full_adder f7(prod[6], prod[7], p33, c7, cm5);
endmodule
```

3.2 Testbench

```
//wallace_tb.v

module wallace_tb(in_Clk, lcd_rs, lcd_e, data, number);
reg [3:0] a=2;
reg [3:0] b=5;
wire [7:0] m;
wallace_multiplier w0(m,a,b);
initial
begin
#10 a = 3; b = 7;
```

```
#10 a = 15; b = 2;


#10 $finish;
end
endmodule
```

3.3 Output

Given below is the output seen on Xilinx vivado for the above testbench.

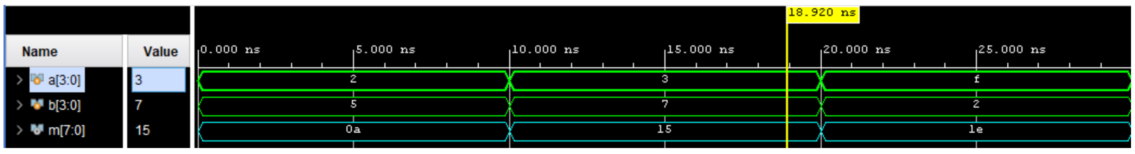

Figure 2: wallace multiplier output

3.4 Constraint file for implementing wallace multiplier on fpga with leds

```
set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 } [get_ports { m[0] }];#LSB
set_property -dict { PACKAGE_PIN H3    IOSTANDARD LVCMOS33 } [get_ports { m[1] }];
set_property -dict { PACKAGE_PIN J1    IOSTANDARD LVCMOS33 } [get_ports { m[2] }];
set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33 } [get_ports { m[3] }];
set_property -dict { PACKAGE_PIN L3    IOSTANDARD LVCMOS33 } [get_ports { m[4] }];
set_property -dict { PACKAGE_PIN L2    IOSTANDARD LVCMOS33 } [get_ports { m[5] }];
set_property -dict { PACKAGE_PIN K3    IOSTANDARD LVCMOS33 } [get_ports { m[6] }];
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { m[7] }];

set_property -dict { PACKAGE_PIN L5    IOSTANDARD LVCMOS33 } [get_ports { a[0] }];#LSB
set_property -dict { PACKAGE_PIN L4    IOSTANDARD LVCMOS33 } [get_ports { a[1] }];
set_property -dict { PACKAGE_PIN M4    IOSTANDARD LVCMOS33 } [get_ports { a[2] }];
set_property -dict { PACKAGE_PIN M2    IOSTANDARD LVCMOS33 } [get_ports { a[3] }];

set_property -dict { PACKAGE_PIN M1    IOSTANDARD LVCMOS33 } [get_ports { b[0] }];
set_property -dict { PACKAGE_PIN N3    IOSTANDARD LVCMOS33 } [get_ports { b[1] }];
set_property -dict { PACKAGE_PIN N2    IOSTANDARD LVCMOS33 } [get_ports { b[2] }];
set_property -dict { PACKAGE_PIN N1    IOSTANDARD LVCMOS33 } [get_ports { b[3] }];
```

4 LCD Display

The next part of the experiment was to setup the LCD display and display characters on it. This is done by writing a verilog code which has the following:

1. Setup a clock to set the rate to write to the LCD display.This is done with a clock divider which we implemented last experiment.
2. Set lcd_rs=0 to set the initial commands.
3. Set lcd_rs=1 when you want to send data.

The Verilog code for the LCD is given below:

4.1 Verilog Implementation

```
module lcd_test_driver(input in_Clk, output reg lcd_rs, output lcd_e, output reg[7:0] data);
    reg [31:0] count = 0;
    assign lcd_e = in_Clk;

    always@(posedge lcd_e) begin
        count = count + 1;

        case(count)
            1: begin lcd_rs = 0; data = 8'h38; end // control signal to display on two lines
            2: begin lcd_rs = 0; data = 8'h0C; end // keep display on but cursor off
            3: begin lcd_rs = 0; data = 8'h06; end // increment the cursor
            4: begin lcd_rs = 0; data = 8'h01; end // clear the display
            5: begin lcd_rs = 0; data = 8'hC0; end // choose the second line
            6: begin lcd_rs = 1; data = 8'h30; end // fill in hex corresp to ASCII for 1
            7: begin lcd_rs = 1; data = 8'h31; end // 2
            8: begin lcd_rs = 1; data = 8'h32; end // 3
            9: begin lcd_rs = 1; data = 8'h33; end // 4
            10: begin lcd_rs = 1; data = 8'h34; end
            11: begin lcd_rs = 1; data = 8'h35; end
            12: begin lcd_rs = 1; data = 8'h36; end
            13: begin lcd_rs = 1; data = 8'h37; end
            14: begin lcd_rs = 1; data = 8'h38; end
            15: begin lcd_rs = 1; data = 8'h41; end
            16: begin lcd_rs = 1; data = 8'h42; end
            17: begin lcd_rs = 1; data = 8'h43; end
            18: begin lcd_rs = 1; data = 8'h44; end
            19: begin lcd_rs = 1; data = 8'h45; end // E
            20: begin lcd_rs = 1; data = 8'h46; end // F
            21: begin lcd_rs = 1; data = 8'h47; end // hex corresp to ASCII for G

            default: begin lcd_rs = 0; data = 8'h80; end // return cursor to home
        endcase
    end
endmodule


module circuit_lcd_test(in_Clk, lcd_rs, lcd_e, lcd_data);
    input in_Clk;
    output lcd_rs, lcd_e;
    output [7:0] lcd_data;

    wire divided_clk;
    clk_div cdiv(in_Clk, divided_clk);
    lcd_test_driver(divided_clk, lcd_rs, lcd_e, lcd_data);
endmodule
```

```
module clk_div(input inclk, output reg outclk);
    reg[32:0] clock_count;
    initial clock_count = 0;

    always @(posedge inclk) begin
        clock_count = clock_count + 1;
        if(clock_count == 5000000) begin
            outclk = ~outclk;
            clock_count = 0;
        end
    end
endmodule
```

4.2 Output

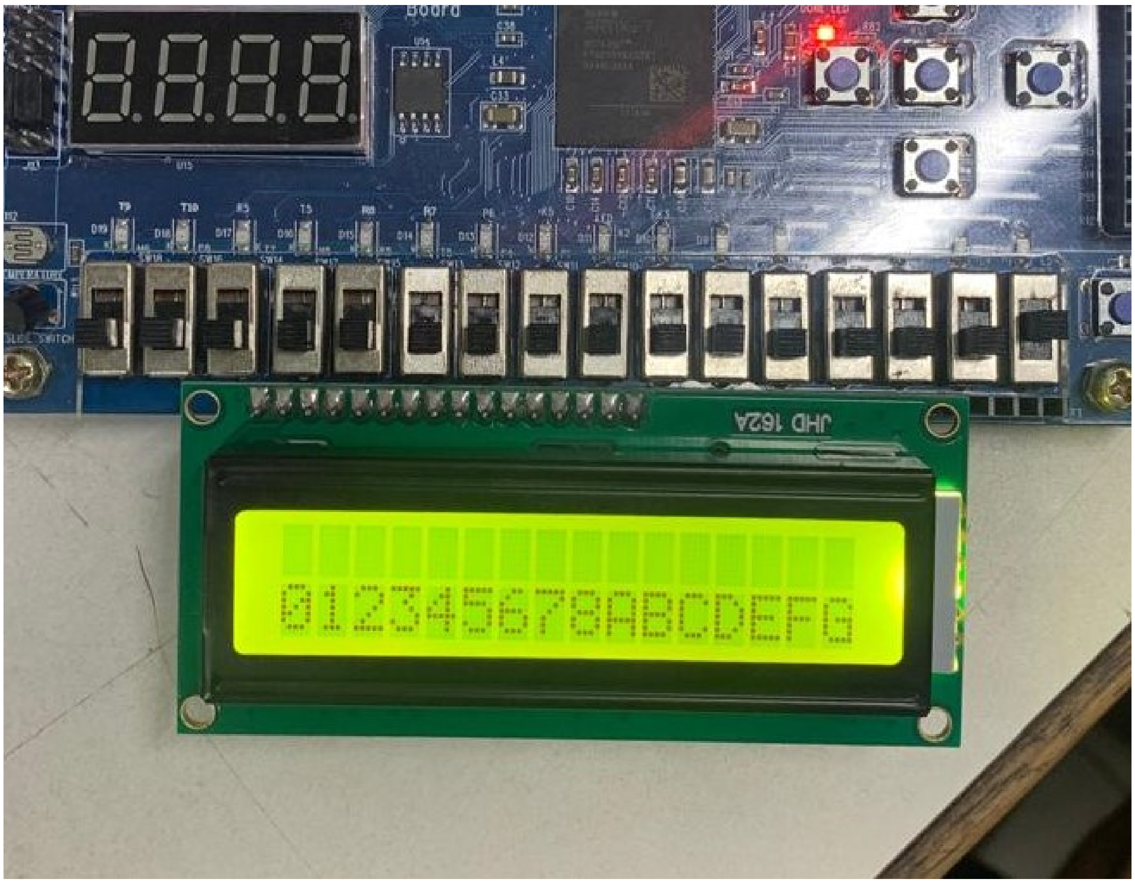

Figure 3: LCD output

4.3 Constraint file for LCD display

```
set_property -dict { PACKAGE_PIN N11    IOSTANDARD LVCMOS33 } [get_ports { in_Clk }];
set_property -dict { PACKAGE_PIN P3 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[7]}];
set_property -dict { PACKAGE_PIN M5 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[6]}];
set_property -dict { PACKAGE_PIN N4 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[5]}];
set_property -dict { PACKAGE_PIN R2 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[4]}];
set_property -dict { PACKAGE_PIN R1 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[3]}];
set_property -dict { PACKAGE_PIN R3 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[2]}];
set_property -dict { PACKAGE_PIN T2 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[1]}];
set_property -dict { PACKAGE_PIN T4 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[0]}];
set_property -dict { PACKAGE_PIN T3 IOSTANDARD LVCMOS33 } [get_ports {lcd_e}];
set_property -dict { PACKAGE_PIN P5 IOSTANDARD LVCMOS33 } [get_ports {lcd_rs}];
#LCD R/W pin is connected to ground by default.No need to assign LCD R/W Pin.
```

5 LCD Display for Wallace multiplier

The next part of the experiment was to display on the first line of the LCD display ”PRODUCT =” and the next line the result of the multiplication. To achieve this we had to:

1. Take the inputs of the multiplication from the switches and calculate the result using the wallace multiplier module.
2. Parse the 8 bit output into 3 digit output.Print this 3 digit output on the LCD screen.

The way we did this is:

1. We know that the inputs were 4bit numbers therefore they require two bcd digits.The output is 8bit and requires three bcd digits.
2. To convert to bcd we first found the remainder when divided by 10 to get the value of the digit(ones place).Then we divided the number by 10(integer division).This left us with a number with the tens and hundreds digit.
3. We then repeated this process with the new number we formed to get the tens and hundreds digit.
4. After finding the numbers we then printed them on the LCD by using their ascii values.

NOTE:We update the LCD display every time there is a change in the input otherwise the LCD display remains the same. If the input changes while text is being written on the LCD display then it first writes then after finishing writing it checks change in input. Given below is the verilog implementation of the above:

5.1 Verilog Implementation

```
module lcd_number_driver(in_Clk, lcd_rs, lcd_e, data, number, a, b);
input in_Clk;
output reg [7:0] data;
output reg lcd_rs;
output lcd_e;
input [7:0] number;
input [3:0] a;
input [3:0] b;

wire [3:0] pbcd0;
wire [3:0] pbcd1;
wire [3:0] pbcd2
```

```verilog
binary2bcd8 b2bp(pbcd0, pbcd1, pbcd2, number);

wire [3:0] abcd0;
wire [3:0] abcd1;
binary2bcd4 b2ba(abcd0, abcd1, a);

wire [3:0] bbcd0;
wire [3:0] bbcd1;
binary2bcd4 b2bb(bbcd0, bbcd1, b);

reg[7:0] old_number;
initial old_number = number;

reg [31:0] count=0;

assign lcd_e = in_Clk;

always@(posedge lcd_e) begin
    count = count + 1;

    case(count)
        1: begin lcd_rs = 0; data = 8'h38; end // control signal to display on two lines
        2: begin lcd_rs = 0; data = 8'h0C; end // keep display on but cursor off
        3: begin lcd_rs = 0; data = 8'h06; end // increment the cursor
        4: begin lcd_rs = 0; data = 8'h01; end // clear the display
        5: begin lcd_rs = 0; data = 8'h80; end // go to home
        6: begin lcd_rs = 1; data = 8'h50; end // fill in hex corresp to P
        7: begin lcd_rs = 1; data = 8'h52; end // fill in hex corresp to R
        8: begin lcd_rs = 1; data = 8'h4f; end // fill in hex corresp to O
        9: begin lcd_rs = 1; data = 8'h44; end // fill in hex corresp to D
        10: begin lcd_rs = 1; data = 8'h55; end // fill in hex corresp to U
        11: begin lcd_rs = 1; data = 8'h43; end // fill in hex corresp to C
        12: begin lcd_rs = 1; data = 8'h54; end // fill in hex corresp to T
        13: begin lcd_rs = 1; data = 8'h20; end // fill in hex corresp to
        14: begin lcd_rs = 1; data = 8'h30 + abcd1; end // a1
        15: begin lcd_rs = 1; data = 8'h30 + abcd0; end // a2
        16: begin lcd_rs = 1; data = 8'h20; end // fill in hex corresp to
        17: begin lcd_rs = 1; data = 8'h58; end // fill in hex corresp to X
        18: begin lcd_rs = 1; data = 8'h20; end // fill in hex corresp to
        19: begin lcd_rs = 1; data = 8'h30 + bbcd1; end // b1
        20: begin lcd_rs = 1; data = 8'h30 + bbcd0; end // b2
        21: begin lcd_rs = 1; data = 8'h20; end // fill in hex corresp to
        22: begin lcd_rs = 1; data = 8'h20; end // fill in hex corresp to
        23: begin lcd_rs = 0; data = 8'hC0; end // choose the second line
        24: begin lcd_rs = 1; data = 8'h3d; end // fill in hex corresp to =
        25: begin lcd_rs = 1; data = 8'h30 + pbcd2; end // fill in hex corresp to ASCII for 1
        26: begin lcd_rs = 1; data = 8'h30 + pbcd1; end // 2
        27: begin lcd_rs = 1; data = 8'h30 + pbcd0; end // 3
        default: begin
            lcd_rs = 0; data = 8'h80;

            if (number == old_number) count = 28;
            else begin
                old_number = number;
                count = 0;
            end
            end // go back to home
        endcase
    end
endmodule



module binary2bcd8(
    bcd0, bcd1, bcd2 ,binary
);
    input [7:0]binary;
    output reg[3:0] bcd0;
    output reg[3:0] bcd1;
    output reg[3:0] bcd2;

    always @(binary) begin
        bcd0 = binary % 10;
        bcd1 = (binary / 10) % 10;
        bcd2 = binary / 100;
    end
endmodule

module binary2bcd4(
    bcd0, bcd1 ,binary
);
    input [7:0]binary;
    output reg[3:0] bcd0;
    output reg[3:0] bcd1;

    always @(binary) begin
        bcd0 = binary % 10;
        bcd1 = (binary / 10);
    end
endmodule

module clk_div(input inclk, output reg outclk);
    reg[32:0] clock_count;
    initial clock_count = 0;

    always @(posedge inclk) begin
        clock_count = clock_count + 1;
        if(clock_count == 5000000) begin
            outclk = ~outclk;
            clock_count = 0;
        end
    end
endmodule
```

5.2 TestBench

```
module circuit_wallace_display(a, b, prod, in_Clk, lcd_rs, lcd_e, lcd_data);
    input [3:0] a, b;
    output [7:0] prod;
    input in_Clk;
    output lcd_rs, lcd_e;
    output [7:0] lcd_data;

    wallace_multiplier(prod, a, b);

    wire divided_clk;
    clk_div cdiv(in_Clk, divided_clk);

    lcd_number_driver(divided_clk, lcd_rs, lcd_e, lcd_data, prod, a, b);
endmodule
```

5.3 Constrain file for LCD with Wallace

```
set_property -dict { PACKAGE_PIN N11    IOSTANDARD LVCMOS33 } [get_ports { in_Clk }];

set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 } [get_ports { prod[0] }];#LSB
set_property -dict { PACKAGE_PIN H3    IOSTANDARD LVCMOS33 } [get_ports { prod[1] }];
set_property -dict { PACKAGE_PIN J1    IOSTANDARD LVCMOS33 } [get_ports { prod[2] }];
set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33 } [get_ports { prod[3] }];
set_property -dict { PACKAGE_PIN L3    IOSTANDARD LVCMOS33 } [get_ports { prod[4] }];
set_property -dict { PACKAGE_PIN L2    IOSTANDARD LVCMOS33 } [get_ports { prod[5] }];
set_property -dict { PACKAGE_PIN K3    IOSTANDARD LVCMOS33 } [get_ports { prod[6] }];
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { prod[7] }];

set_property -dict { PACKAGE_PIN L5    IOSTANDARD LVCMOS33 } [get_ports { a[0] }];#LSB
set_property -dict { PACKAGE_PIN L4    IOSTANDARD LVCMOS33 } [get_ports { a[1] }];
set_property -dict { PACKAGE_PIN M4    IOSTANDARD LVCMOS33 } [get_ports { a[2] }];
set_property -dict { PACKAGE_PIN M2    IOSTANDARD LVCMOS33 } [get_ports { a[3] }];

set_property -dict { PACKAGE_PIN M1    IOSTANDARD LVCMOS33 } [get_ports { b[0] }];
set_property -dict { PACKAGE_PIN N3    IOSTANDARD LVCMOS33 } [get_ports { b[1] }];
set_property -dict { PACKAGE_PIN N2    IOSTANDARD LVCMOS33 } [get_ports { b[2] }];
set_property -dict { PACKAGE_PIN N1    IOSTANDARD LVCMOS33 } [get_ports { b[3] }];



set_property -dict { PACKAGE_PIN P3 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[7]}];
set_property -dict { PACKAGE_PIN M5 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[6]}];
set_property -dict { PACKAGE_PIN N4 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[5]}];
set_property -dict { PACKAGE_PIN R2 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[4]}];
set_property -dict { PACKAGE_PIN R1 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[3]}];
set_property -dict { PACKAGE_PIN R3 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[2]}];
set_property -dict { PACKAGE_PIN T2 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[1]}];
set_property -dict { PACKAGE_PIN T4 IOSTANDARD LVCMOS33 } [get_ports {lcd_data[0]}];
set_property -dict { PACKAGE_PIN T3 IOSTANDARD LVCMOS33 } [get_ports {lcd_e}];
set_property -dict { PACKAGE_PIN P5 IOSTANDARD LVCMOS33 } [get_ports {lcd_rs}];
#LCD R/W pin is connected to ground by default.No need to assign LCD R/W Pin.
```

5.4 Output

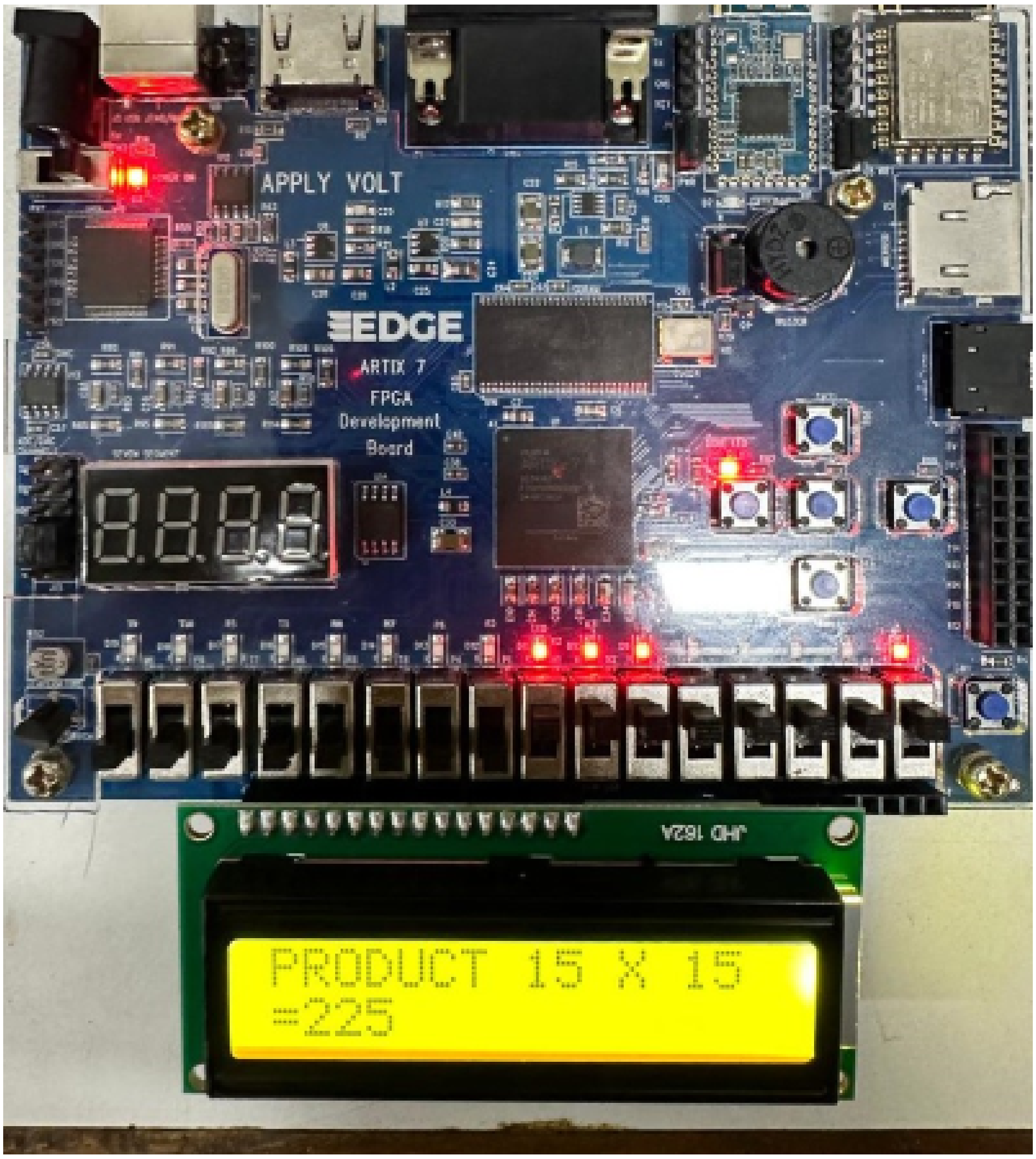

Figure 4: LCD output

6 Comments

6.1 Debugging

6.1.1 Wallace Multiplier

While implementing the Wallace multiplier initially we were confused on how to approach multiple layers of Full Adder addition.However once we brainstormed we realised that we could stack the Full Adders to achieve adding more than 3 bits.

6.1.2 LCD issue

Initially the LCD display was not working there was only one row of black boxes which did not change.However after a long time of debugging we saw that we did not connect the LCD properly.We also realised there were some issues with our code which after being rectified started working.

6.1.3 LCD with Wallace

There were many hurdles on how to approach the problem.Our main issue was with how to convert binary to decimal characters to feed to the LCD display. After some thinking we finally got the idea which we implemented.