MICROPROCESSOR LAB EXPERIMENT 3

Deenabandhan N ee23b021
Sai Harshith Gajendra ee23b069
Krutarth Patel ee23b137

31 August 2024

Introduction

FPGA board : Edge Artix 7

This experiment involves

- Simulating a wallace multiplier
- Displaying text on an LCD
- Displaying product of two 4-bit numbers on the LCD

Xilinx Vivado

- We were introduced to Xilinx vivado , a software that is used to synthesize and analyse the hardware description designs.
- The procedures that we followed are ,
  - Create a project with source file as our Verilog code.
  - Add constraints to the ports that we have defined in the Verilog code.
  - Run the synthesis to check whether our Verilog code has any error in it.
  - We can open the Schematics to see the design that we have coded in Verilog.
  - After running synthesis , we can run simulation using our testbench and check for logical errors.
  - Once we confirm there is no logical/syntax error , we can run the implementation which basically implements our hardware description in the board which we have chosen while creating the project.
  - After the implementation is done , we can generate bitstream which is basically a file that contains the configuration information for an FPGA.
  - Once we successfully generate the bitstream , we can connect the target source and program it with the generated bitstream.
- In this report , we have included
  - Verilog code for module instantiation.
  - The Schematics generated from Xilinx Vivado.
  - The Constraints file generated for FPGA.
  - The testbench and simulation generated
  - The analysis of reports obtained from Xilinx Vivado.

Wallace Multiplier

Data flow model

```
module unsigned_mult(output [7:0] m , input [3:0] a,b);
        wire [3:0]p[0:3];
        genvar i,j;
        generate
        for(i=0;i<4;i=i+1)
        begin
                for(j=0;j<4;j=j+1)
                begin
                        assign p[i][j]=a[j]*b[i];
                end
        end
        endgenerate
        wire s0,s1,s2,s3,s4;
        wire c0,c1,c2,c3,c4;

        wire k1,l1,k2,l2;

        assign m[0]=p[0][0];

        half_adder h0(k1,l1,p[3][0],p[2][1]);

        half_adder h1(k2,l2,p[1][3],p[2][2]);

        half_adder h2(s0,c0,p[0][1],p[1][0]);

        FA_S f1(s1,c1,p[2][0],p[1][1],p[0][2]);

        FA_S f2(s2,c2,p[0][3],p[1][2],k1);

        FA_S f3(s3,c3,p[3][1],k2,l1);

        FA_S f4(s4,c4,p[2][3],p[3][2],l2);

        wire u1,u2,u3,u4;

        assign m[1]=s0;

        half_adder h3(m[2],u1,s1,c0);

        FA_S f5(m[3],u2,u1,s2,c1);

        FA_S f6(m[4],u3,u2,s3,c2);

        FA_S f7(m[5],u4,u3,s4,c3);

        FA_S f8(m[6],m[7],u4,p[3][3],c4);
endmodule
```

Schematics :

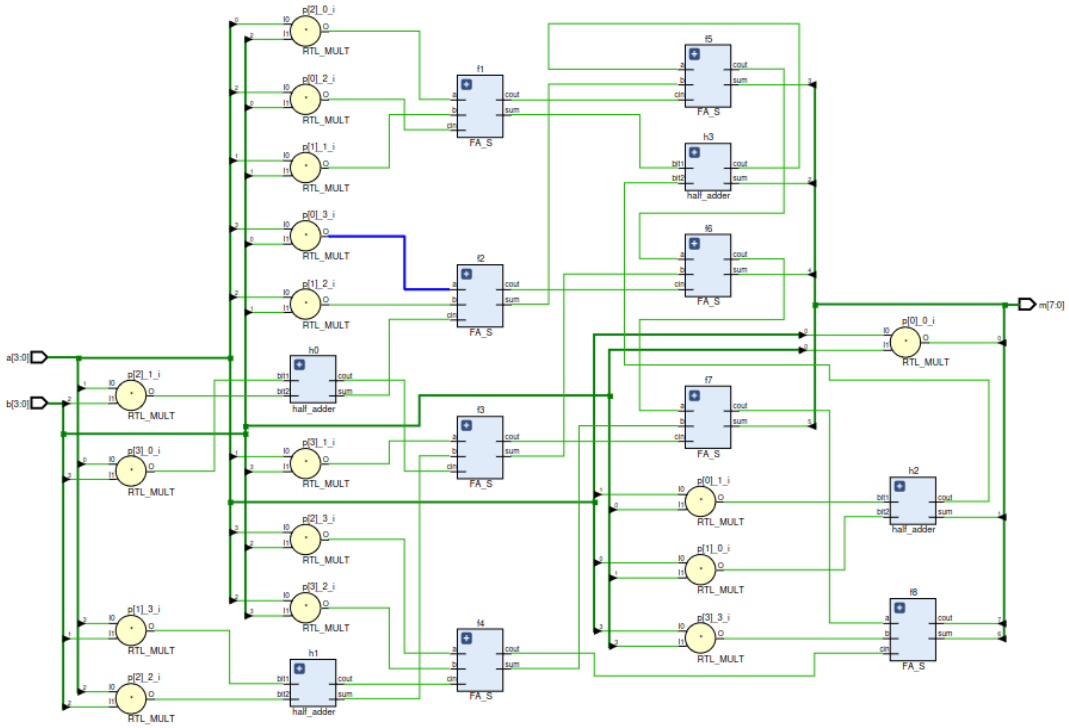


Figure 1: Schematics of the Data flow modelling

Constraints on ports of FPGA

```
set_property IOSTANDARD LVCMOS33 [get_ports {a[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[0]}]
set_property PACKAGE_PIN L5 [get_ports {a[0]}]
set_property PACKAGE_PIN L4 [get_ports {a[1]}]
set_property PACKAGE_PIN M4 [get_ports {a[2]}]
set_property PACKAGE_PIN M2 [get_ports {a[3]}]
set_property PACKAGE_PIN M1 [get_ports {b[0]}]
set_property PACKAGE_PIN N3 [get_ports {b[1]}]
set_property PACKAGE_PIN N2 [get_ports {b[2]}]
set_property PACKAGE_PIN N1 [get_ports {b[3]}]
set_property PACKAGE_PIN J3 [get_ports {m[0]}]
set_property PACKAGE_PIN H3 [get_ports {m[1]}]
set_property PACKAGE_PIN J1 [get_ports {m[2]}]
set_property PACKAGE_PIN K1 [get_ports {m[3]}]
set_property PACKAGE_PIN L3 [get_ports {m[4]}]
set_property PACKAGE_PIN L2 [get_ports {m[5]}]
set_property PACKAGE_PIN K3 [get_ports {m[6]}]
set_property PACKAGE_PIN K2 [get_ports {m[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {m[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {m[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {m[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {m[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {m[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {m[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {m[1]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {m[0]}]}
```

Reports :

Resources utilized :

Ref Name	*Used*	*Functional category*
OBUF	*8*	**IO**
IBUF	*8*	**IO**
LUT2	*2*	**LUT**
LUT3	*2*	**LUT**
LUT4	*3*	**LUT**
LUT5	*7*	**LUT**
LUT6	*6*	**LUT**

Time delays

Type of Path	*Path taken*	**Time delay (ns)**
Max Delay path	**b[3] → m[6]**	**11.016**
	b[3] → m[3]	**10.913**
	b[3] → m[7]	**10.877**
Min Delay path	**a[1] → m[2]**	**2.275**
	b[1] → m[7]	**2.297**
	b[1] → m[6]	**2.332**

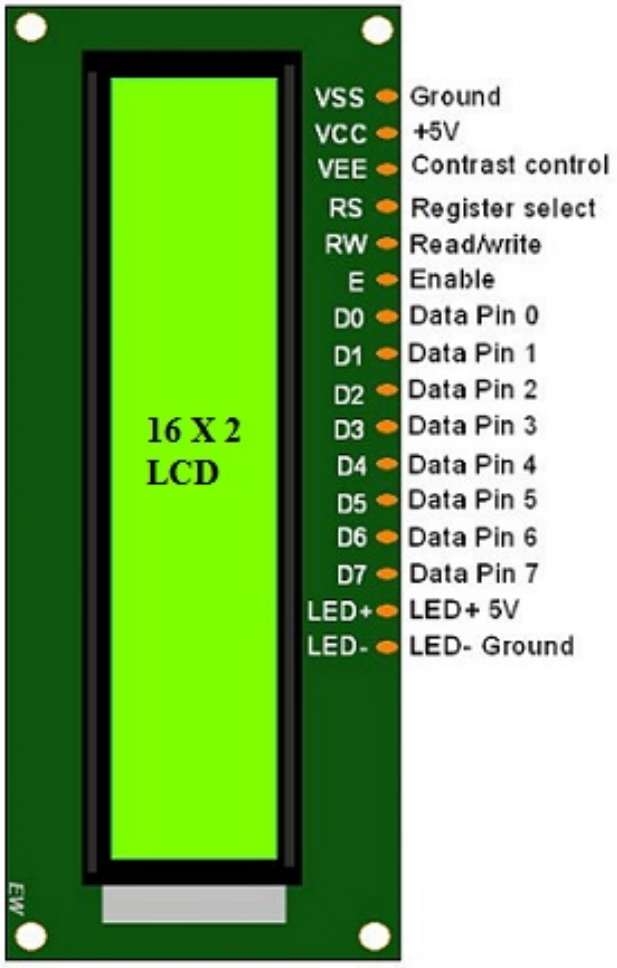
Power consumed

- The power consumed by FPGA is **0.325 W**

LCD Display

Aim :

- The aim of this part of experiment is to display some characters on the LCD display
- For this part of the experiment we are using LCD display of 16 by 2 dimension for which data is fed by FPGA.



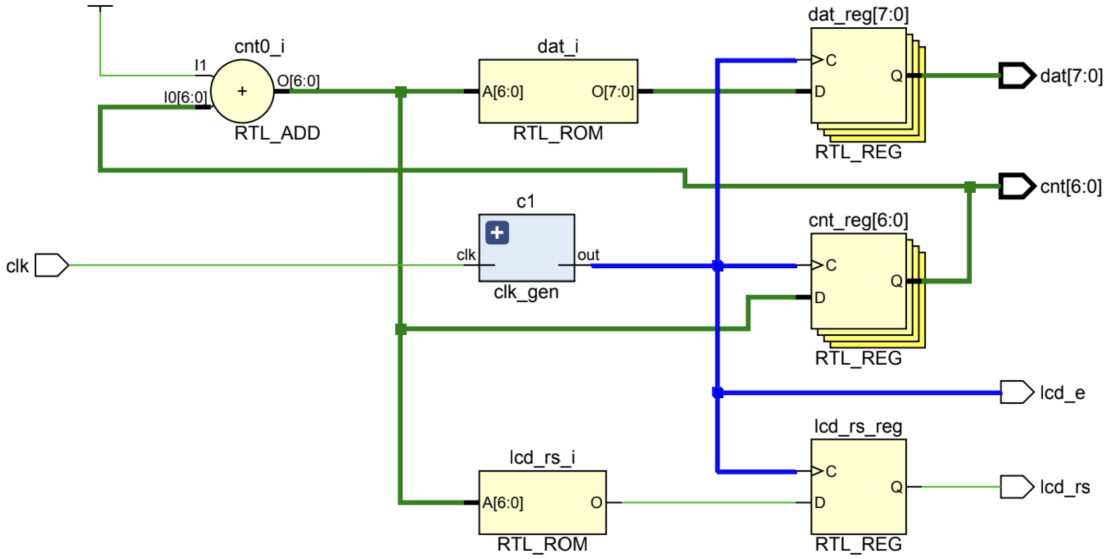
- The important terminologies that we need to note are :
  - **lcd_e** This is used to enable the lcd to accept the inputs basically acts as an clock stimulator.
  - **lcd_rs** This is used to decide whether the given data is an instruction to the lcd or the data to be displayed in the lcd.
  - **dat[7:0]** This is the data given to lcd.

Code :

```
module clk_gen(input clk,output out);
        reg out=1'b0;
        reg [31:0] count=0;
        always@(posedge clk)
        begin
                        count+=1;
                        if(count==(1<<4))
                        begin
                                out=~(out);
                                count=0;
```

```
                        end
        end
endmodule

module lcd_disp(input clk, output reg lcd_rs, output lcd_e,output reg [7:0] dat);

        wire [7:0] command[0:4];
        reg [31:0] cnt=0;

        clk_gen c1(clk,lcd_e);

        assign command[0]=8'h38;
        assign command[1]=8'h0C;
        assign command[2]=8'h06;
        assign command[3]=8'h01;
        assign command[4]=8'hC0;

        always@(posedge lcd_e) begin

                cnt=cnt+1;

                case(cnt)
                        1: begin lcd_rs=0;dat=command[0]; end
                        2: begin lcd_rs=0;dat=command[1]; end
                        3: begin lcd_rs=0;dat=command[2]; end
                        4: begin lcd_rs=0;dat=command[3]; end
                        5: begin lcd_rs=0;dat=command[4]; end
                        6: begin lcd_rs=1;dat=8'h31; end
                        7: begin lcd_rs=1;dat=8'h32; end
                        8: begin lcd_rs=1;dat=8'h33; end
                        9: begin lcd_rs=1;dat=8'h34; end
                        10: begin lcd_rs=1;dat=8'h35; end
                        11: begin lcd_rs=1;dat=8'h36; end
                        12: begin lcd_rs=1;dat=8'h37; end
                        13: begin lcd_rs=1;dat=8'h38; end
                        14: begin lcd_rs=1;dat=8'h39; end
                        15: begin lcd_rs=1;dat=8'h41; end
                        16: begin lcd_rs=1;dat=8'h42; end
                        17: begin lcd_rs=1;dat=8'h43; end
                        18: begin lcd_rs=1;dat=8'h44; end
                        19: begin lcd_rs=1;dat=8'h45; end
                        20: begin lcd_rs=1;dat=8'h46; end
                        21: begin lcd_rs=1;dat=8'h47; end

                        100: begin cnt=0; end
                        default : begin lcd_rs=0;dat=8'h02; end

                endcase
        end

endmodule
```

Schematics



Constraints

```
set_property IOSTANDARD LVCMOS33 [get_ports {dat[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[0]}]
set_property PACKAGE_PIN P3 [get_ports {dat[7]}]
set_property PACKAGE_PIN M5 [get_ports {dat[6]}]
set_property PACKAGE_PIN N4 [get_ports {dat[5]}]
set_property PACKAGE_PIN R2 [get_ports {dat[4]}]
set_property PACKAGE_PIN R1 [get_ports {dat[3]}]
set_property PACKAGE_PIN R3 [get_ports {dat[2]}]
set_property PACKAGE_PIN T2 [get_ports {dat[1]}]
set_property PACKAGE_PIN T4 [get_ports {dat[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN N11 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports lcd_e]
set_property IOSTANDARD LVCMOS33 [get_ports lcd_rs]
set_property PACKAGE_PIN T3 [get_ports lcd_e]
set_property PACKAGE_PIN P5 [get_ports lcd_rs]
```

Reports :

Resources utilized :

- 19 **SLICE LUT** were used, where LUT is used as a logic and not memory.
- 53 Slice Registers were used as Flip Flop.

Ref Name	*Used*	*Functional category*
OBUF	*8*	**IO**
IBUF	*8*	**IO**
LUT2	*2*	**LUT**
LUT3	*2*	**LUT**
LUT4	*3*	**LUT**
LUT5	*7*	**LUT**
LUT6	*6*	**LUT**

Clock :

- only 1 Global Clock was utilized.

Time delays

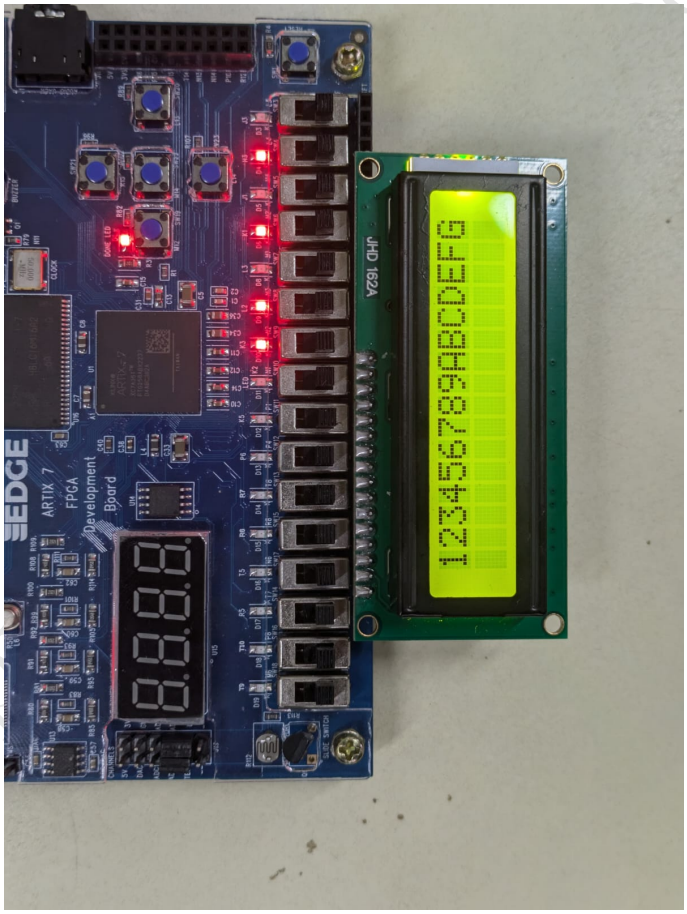
Type of Path	*Path taken*	**Time delay (ns)**
Min Delay path	**cnt_reg[2]/C → dat_reg[3]/D**	*0.340*
Max Delay path	**lcd_rs_reg/C → lcd_rs**	*6.142*
	c1/out_reg/C → lcd_e	*6.078*
	dat_reg[1]/C → dat[1]	*6.064*

Power consumed

- The power consumed by FPGA is **12.261W**
- It is show high because we used LED's to display the counter variable as we were not getting results for a long time.

Final result :

- We generated the desired output and the picture is given below.



- The important issue that we faced here was the picture given in the experiment assigns D0 to the P7 port of the fpga.
- So we thought that it meant the date[0] should be given to P7 and we weren't getting the output for a long time.
- We rectified it by the end of the class.

Bonus Question

Approach to the question :

- We have solved bonus question by using the same code that we have used in the lcd display part with the slight modification.
- The important aspect of this question is to convert the binary numbers to its decimal equivalent in ascii code.
- In this part of the report , we will see the verilog code to convert the binary to its decimal equivalent in ascii form.

Module to convert binary to decimal with 2 digits

```
module hex_dec_ascii_2(output reg [7:0] tens,ones,input [3:0] bin);

        wire [3:0] dec;
        assign dec=bin;

        always@(*) begin

                ones = (dec%10) + 8'h30 ; // Takes the unit digit and converts to ascii

                tens = ((dec/10)%10) + 8'h30 ; // Takes the tens digit and converts to ascii

        end

endmodule
```

Module to convert binary to decimal with 3 digits

```
module hex_dec_ascii_3(output reg [7:0] hundreds,tens,ones,input [7:0] bin);

        wire [7:0] dec;
        assign dec=bin;

        always@(*) begin

                ones = (dec%10) + 8'h30 ; // Takes the unit digit and converts to ascii

                tens = ((dec/10)%10) + 8'h30 ; // Takes the tens digit and converts to ascii

                hundreds = ((dec/100)%100) + 8'h30 ; // Takes the hundred's digit and converts to asc

        end

endmodule
```

- **Key lesson learnt :**
  - The division operation assigns extra bits to the output i.e if the input is 8 bits the division of input by 10 will have 8 bits.
  - To avoid it we take modulo by 10 so that the resultant won't overflow.
  - We were getting errors due to this error and rectified it after multiple trials.
- Regarding changing the lcd display , we can do it either by synchronous or asynchronous trigger.
- Synchronous will mean that the display will be refreshed after every time period and asynchronous would mean that display will change only when the inputs are changed.

Synchronous reset

```
module half_adder(output sum,cout
                 ,input bit1,bit2);

    assign sum = bit1 ^ bit2;
    assign cout = bit1 & bit2;

endmodule

module hex_dec_ascii_2(output reg [7:0] tens,ones,input [3:0] bin);

        wire [3:0] dec;
        assign dec=bin;

        always@(*) begin

                ones = (dec%10) + 8'h30 ;

                tens = ((dec/10)%10) + 8'h30 ;

        end

endmodule

module hex_dec_ascii_3(output reg [7:0] hundreds,tens,ones,input [7:0] bin);

        wire [7:0] dec;
        assign dec=bin;

        always@(*) begin

                ones = (dec%10) + 8'h30 ;

                tens = ((dec/10)%10) + 8'h30 ;

                hundreds = ((dec/100)%100) + 8'h30 ;

        end

endmodule

module FA_S(output sum,cout,
            input a,b,cin);
        wire s1,s2,s3,s4,s5;

        xor x1(s1,a,b);
        xor x2(sum,s1,cin);

        and a1(s2,a,b);
        and a2(s3,a,cin);
        and a3(s4,b,cin);
        or o1(s5,s2,s3);
        or o2(cout,s5,s4);
endmodule

module clk_gen(input clk,output out);
        reg out=1'b0;
        reg [31:0] count=0;
        always@(posedge clk)
        begin
                        count+=1;
```

```
                        if(count==(1<<4))
                        begin
                                out=~(out);
                                count=0;
                        end
        end
endmodule

module unsigned_mult(output [7:0] m , input [3:0] a,b);
        wire [3:0]p[0:3];
        genvar i,j;
        generate
        for(i=0;i<4;i=i+1)
        begin
                for(j=0;j<4;j=j+1)
                begin
                        assign p[i][j]=a[j]*b[i];
                end
        end
        endgenerate

        wire s0,s1,s2,s3,s4;
        wire c0,c1,c2,c3,c4;

        wire k1,l1,k2,l2;

        assign m[0]=p[0][0];

        half_adder h0(k1,l1,p[3][0],p[2][1]);

        half_adder h1(k2,l2,p[1][3],p[2][2]);

        half_adder h2(s0,c0,p[0][1],p[1][0]);

        FA_S f1(s1,c1,p[2][0],p[1][1],p[0][2]);

        FA_S f2(s2,c2,p[0][3],p[1][2],k1);

        FA_S f3(s3,c3,p[3][1],k2,l1);

        FA_S f4(s4,c4,p[2][3],p[3][2],l2);

        wire u1,u2,u3,u4;

        assign m[1]=s0;

        half_adder h3(m[2],u1,s1,c0);

        FA_S f5(m[3],u2,u1,s2,c1);

        FA_S f6(m[4],u3,u2,s3,c2);

        FA_S f7(m[5],u4,u3,s4,c3);

        FA_S f8(m[6],m[7],u4,p[3][3],c4);


endmodule


module lcd_disp(input clk,input [3:0] a,b, output reg lcd_rs, output lcd_e,output reg [7:0] dat);
```

```
        wire [7:0] command[0:5];
        reg [31:0] cnt=0;

        clk_gen c1(clk,lcd_e);

        assign command[0]=8'h38;
        assign command[1]=8'h0C;
        assign command[2]=8'h06;
        assign command[3]=8'h01;
        assign command[4]=8'h80;
        assign command[5]=8'hC0;

        wire [7:0] out;
        wire [7:0] i0,i1,i2,i3,o0,o1,o2;

        hex_dec_ascii_2 h1(i0,i1,a);

        hex_dec_ascii_2 h2(i2,i3,b);

        unsigned_mult m1(out,a,b);

        hex_dec_ascii_3 h3(o0,o1,o2,out);


        always@(posedge lcd_e) begin

                cnt=cnt+1;
                case(cnt)
                        1: begin lcd_rs=0;dat=command[0]; end
                        2: begin lcd_rs=0;dat=command[1]; end
                        3: begin lcd_rs=0;dat=command[2]; end
                        4: begin lcd_rs=0;dat=command[3]; end
                        5: begin lcd_rs=0;dat=command[4]; end
                        6: begin lcd_rs=1;dat=8'h50; end \\ P
                        7: begin lcd_rs=1;dat=8'h52; end \\ R
                        8: begin lcd_rs=1;dat=8'h4F; end \\ O
                        9: begin lcd_rs=1;dat=8'h44; end \\ D
                        10: begin lcd_rs=1;dat=8'h55; end \\ U
                        11: begin lcd_rs=1;dat=8'h43; end \\ C
                        12: begin lcd_rs=1;dat=8'h54; end \\ T
                        13: begin lcd_rs=1;dat=8'h20; end \\ <space>
                        14: begin lcd_rs=1;dat=8'h4F; end \\ O
                        15: begin lcd_rs=1;dat=8'h46; end \\ F
                        16: begin lcd_rs=1;dat=i0; end
                        17: begin lcd_rs=1;dat=i1; end
                        18: begin lcd_rs=1;dat=8'h2A; end \\ *
                        19: begin lcd_rs=1;dat=i2; end
                        20: begin lcd_rs=1;dat=i3; end
                        21: begin lcd_rs=0;dat=command[5]; end
                        22: begin lcd_rs=1;dat=8'h3D; end \\ =
                        22: begin lcd_rs=1;dat=o0; end
                        23: begin lcd_rs=1;dat=o1; end
                        24: begin lcd_rs=1;dat=o2; end
                        100: begin cnt=0; end
                        default : begin lcd_rs=0;dat=8'h02; end

                endcase
        end

endmodule
```

Schematics :

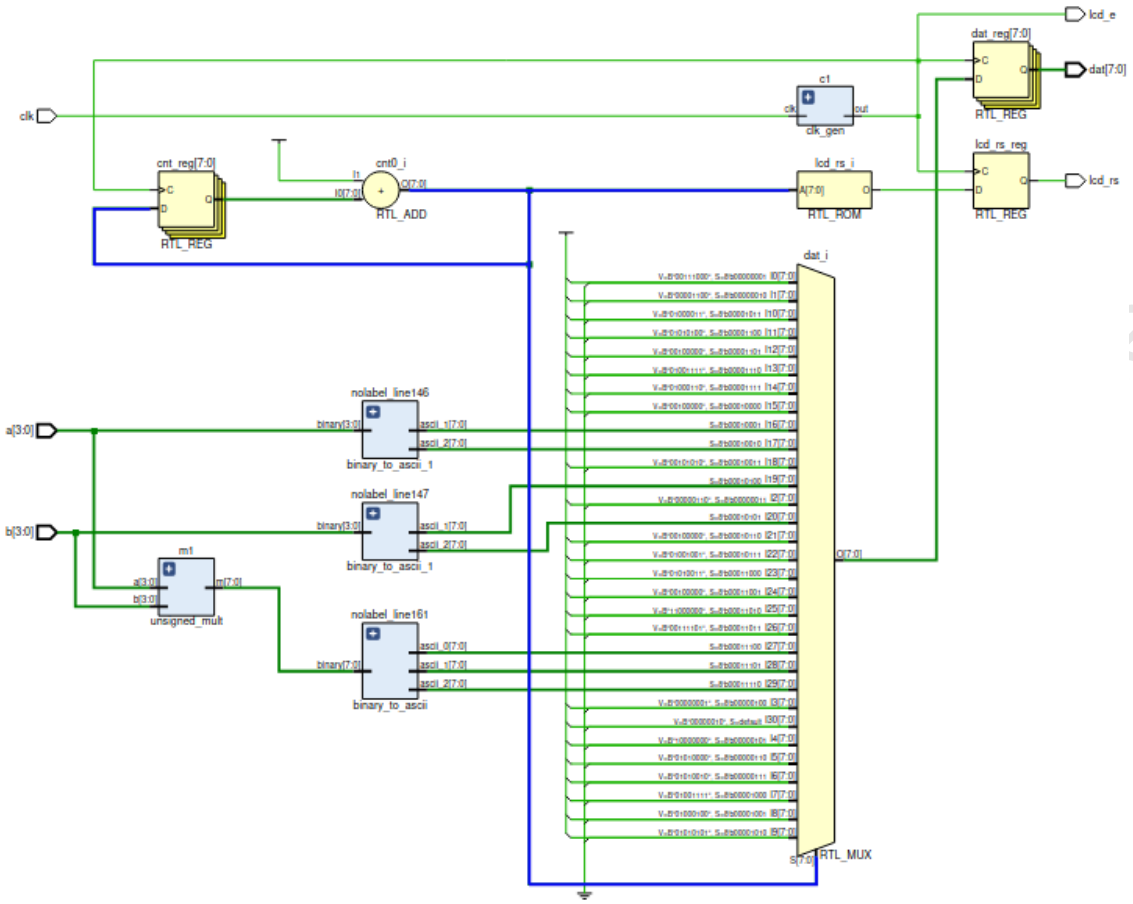


Figure 2: Schematics of the lcd display

Asynchronous reset

- The only change will be made in lcd_disp module and so only that module is given below.

```
module lcd_disp(input clk,input [3:0] a,b, output reg lcd_rs, output lcd_e,output reg [7:0] dat);

        wire [7:0] command[0:5];
        reg [31:0] cnt=0;
        reg [7:0] old_var;

        clk_gen c1(clk,lcd_e);

        assign command[0]=8'h38;
        assign command[1]=8'h0C;
        assign command[2]=8'h06;
        assign command[3]=8'h01;
        assign command[4]=8'h80;
        assign command[5]=8'hC0;

        wire [7:0] out;
        wire [7:0] i0,i1,i2,i3,o0,o1,o2;

        hex_dec_ascii_2 h1(i0,i1,a);

        hex_dec_ascii_2 h2(i2,i3,b);

        unsigned_mult m1(out,a,b);
```

```
        hex_dec_ascii_3 h3(o0,o1,o2,out);


        always@(posedge lcd_e) begin

                case(cnt)
                        0: begin old_var=out; end
                        1: begin lcd_rs=0;dat=command[0]; end
                        2: begin lcd_rs=0;dat=command[1]; end
                        3: begin lcd_rs=0;dat=command[2]; end
                        4: begin lcd_rs=0;dat=command[3]; end
                        5: begin lcd_rs=0;dat=command[4]; end
                        6: begin lcd_rs=1;dat=8'h50; end \\ P
                        7: begin lcd_rs=1;dat=8'h52; end \\ R
                        8: begin lcd_rs=1;dat=8'h4F; end \\ O
                        9: begin lcd_rs=1;dat=8'h44; end \\ D
                        10: begin lcd_rs=1;dat=8'h55; end \\ U
                        11: begin lcd_rs=1;dat=8'h43; end \\ C
                        12: begin lcd_rs=1;dat=8'h54; end \\ T
                        13: begin lcd_rs=1;dat=8'h20; end \\ <space>
                        14: begin lcd_rs=1;dat=8'h4F; end \\ O
                        15: begin lcd_rs=1;dat=8'h46; end \\ F
                        16: begin lcd_rs=1;dat=i0; end
                        17: begin lcd_rs=1;dat=i1; end
                        18: begin lcd_rs=1;dat=8'h2A; end \\ *
                        19: begin lcd_rs=1;dat=i2; end
                        20: begin lcd_rs=1;dat=i3; end
                        21: begin lcd_rs=0;dat=command[5]; end
                        22: begin lcd_rs=1;dat=8'h3D; end \\ =
                        22: begin lcd_rs=1;dat=o0; end
                        23: begin lcd_rs=1;dat=o1; end
                        24: begin lcd_rs=1;dat=o2; end
                        25: begin
                                if(old_var==out) begin
                                        cnt=cnt-1;
                                end
                                else begin
                                        cnt=-1;
                                end
                        end
                        100: begin cnt=0; end
                        default : begin lcd_rs=0;dat=8'h02; end

                endcase
                cnt=cnt+1;
        end

endmodule
```

Constraints on ports of FPGA

```
set_property IOSTANDARD LVCMOS33 [get_ports {a[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {a[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {a[1]}]

set_property IOSTANDARD LVCMOS33 [get_ports {a[0]}]

set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {dat[0]}]
set_property PACKAGE_PIN P3 [get_ports {dat[7]}]
set_property PACKAGE_PIN M5 [get_ports {dat[6]}]
set_property PACKAGE_PIN N4 [get_ports {dat[5]}]
set_property PACKAGE_PIN R2 [get_ports {dat[4]}]
set_property PACKAGE_PIN R1 [get_ports {dat[3]}]
set_property PACKAGE_PIN R3 [get_ports {dat[2]}]
set_property PACKAGE_PIN T2 [get_ports {dat[1]}]
set_property PACKAGE_PIN T4 [get_ports {dat[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports lcd_e]
set_property IOSTANDARD LVCMOS33 [get_ports lcd_rs]
set_property PACKAGE_PIN N11 [get_ports clk]
set_property PACKAGE_PIN T3 [get_ports lcd_e]
set_property PACKAGE_PIN P5 [get_ports lcd_rs]
set_property PACKAGE_PIN L5 [get_ports {a[3]}]
set_property PACKAGE_PIN L4 [get_ports {a[2]}]
set_property PACKAGE_PIN M4 [get_ports {a[1]}]
set_property PACKAGE_PIN M2 [get_ports {a[0]}]
set_property PACKAGE_PIN M1 [get_ports {b[3]}]
set_property PACKAGE_PIN N3 [get_ports {b[2]}]
set_property PACKAGE_PIN N2 [get_ports {b[1]}]
set_property PACKAGE_PIN N1 [get_ports {b[0]}]
```

Reports :

Resources utilized :

Ref Name	*Used*	*Functional category*
OBUF	*8*	**IO**
IBUF	*8*	**IO**
LUT2	*2*	**LUT**
LUT3	*2*	**LUT**
LUT4	*3*	**LUT**
LUT5	*7*	**LUT**
LUT6	*6*	**LUT**

- It is interesting to note that both wallace multiplier and this part has same memory distribution as we have 8 inputs a and b and 8 outputs other than the fact that in the first part , output is given to led and in this part output is given to lcd.
- 108 **SLICE LUT** were used, where LUT is used as a logic and not memory.
- 50 Slice Registers were used as Flip Flop.
- There are F7 and F8 muxes implemented in this circuit.

Clock :

- only 1 Global Clock was utilized.
- This is use to utilize the lcd_e.

Time delays

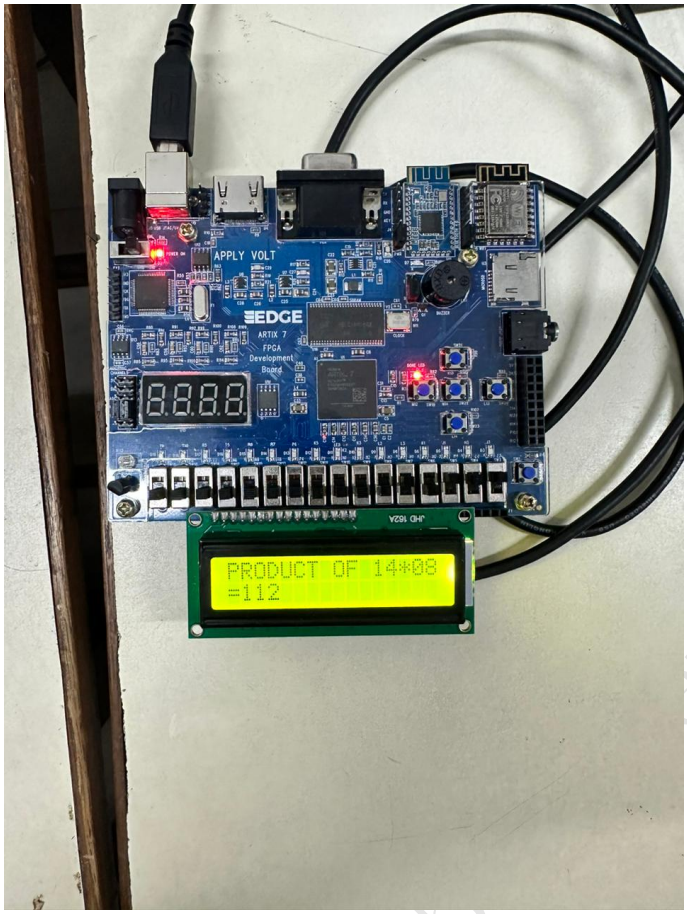
Type of Path	*Path taken*	**Time delay (ns)**
Min Delay path	**cnt_reg[6]/C → dat_reg[2]/D**	*0.367*
	cnt_reg[7]/C → dat_reg[2]/D	*0.353*
Max Delay path	**lcd_rs_reg → lcd_rs**	*5.829*
	a[2] → dat_reg[0]	*9.478*
	b[0] → dat_reg[0]	*7.170*
	dat_reg[7]/C → dat[7]	*5.841*

Power consumed

- The power consumed by FPGA is **1.533W**

Final result :

- We generated the desired output and the picture is given below.



- The important issue that we faced here was initially our clock frequency was slow so we were confused that the lcd display was not working.