Experiment #4 – Accelerator and Wrappers

Parsa Nassery, 810199561 Shayan Kashefi, 810199585

***Abstract*— This document is a student report to experiment #4 of Digital Logic Laboratory course at ECE Department, University of Tehran. In this series of experiments, Exponential Accelerator was created.**

***Keywords*— Exponential Accelerator Engine, Exponential Accelerator Wrapper, Accelerator Buffer, FPGA Implementation.**

I.INTRODUCTION

We are given the Exponential engine to build Exponential Accelerator. To complete it we made three more pieces, a ROM, a Wrapper and a Counter.

II. EXPONENTIAL ENGINE

A. Exponential Testbench

```
`timescale 1ns/1ns
module AccTB();
    reg clk=1'b0, rst=1'b0, start=1'b1;
    wire done;
    reg [15:0] x = 16'b0;
    wire [15:0] fracpart;
    wire [1:0] intpart;
    exponential inst(.clk(clk), .rst(rst),
    .start(start), .x(x), .done(done),
    .intpart(intpart), .fracpart(fracpart));
    initial repeat(10000) #50 clk = ~clk;
    initial begin
    #220 start = 1'b0;
    #300 x = 16'b1111111111111111;
    #3000 start = 1'b1;
    #100 start = 1'b0;
    #300 x = 16'b1000000000000000;
    #3000 start = 1'b1;
    #100 start = 1'b0;
    end
endmodule
```

Fig. 1 Exponential Engine Test Bench

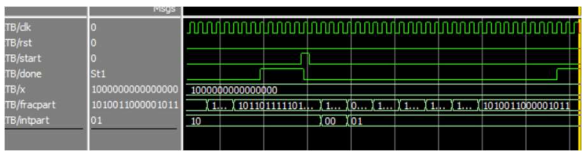

Fig. 2 Exponential Engine Test Bench Result

We put 0.5 in x, So the output must be 1,64. In Fig 3 you can see that the output is near 1,64.

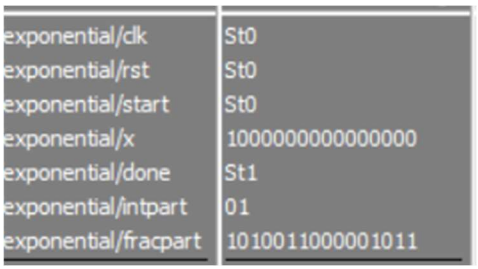

Fig. 3 Expo TB result

B. Synthesize Result

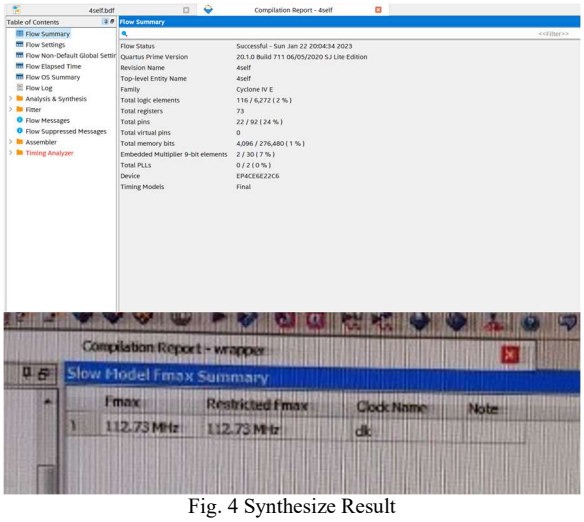

Fig. 4 Synthesize Result

The maximum frequency was 112.73 MHz.

III. EXPONENTIAL ACCELERATOR WAPPER

A. Controller

The Controller Verilog is shown in the figure below.

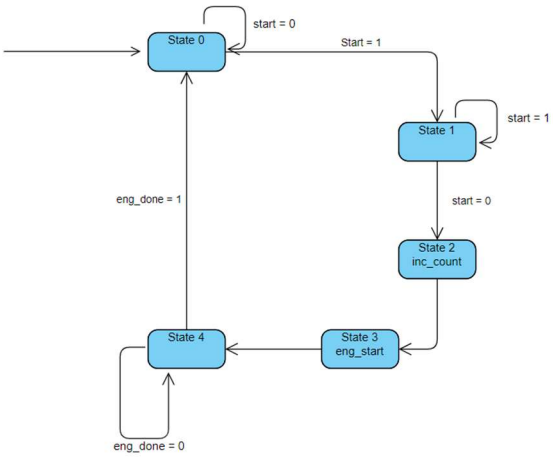

Fig. 5 Controller Verilog

B. Buffer

We put the mif file data in our ROM. As seen in Fig 6, the numbers in line 6 to 11 are given to exponential input.

```
1   WIDTH=16;
2   DEPTH=256;
3   ADDRESS_RADIX=HEX;
4   DATA_RADIX=BIN;
5   CONTENT BEGIN
6   00 : 10000000;                -- memory address : data
7   01 : 0000000000000000;
8   02 : 1111111111111111;
9   03 : 1000000000000000;
10  04 : 0100000000001100;
11  05 : 0000000010001111;
12
13  END;
```

Fig. 6 Rom

The ROM symbol is shown in Figure 7.

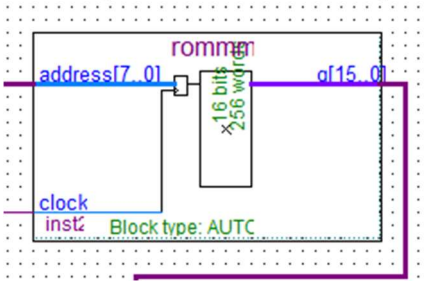

Fig.7 Rom symbol

C. Wrapper

We wrote the Wrapper Verilog and it can be seen in Figure 9.

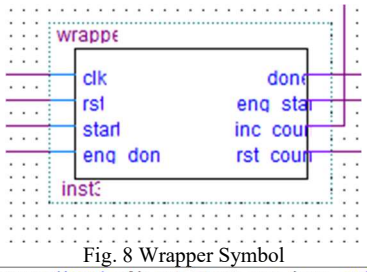

Fig. 8 Wrapper Symbol

```
module wrapper(input clk,rst,start,eng_done ,output reg done,
 eng_start, inc_count, output rst_count);
    reg [2:0] ps, ns;
    always@(ps,eng_done,start)begin
        ns = 3'd0;
        case(ps)
            3'd0: begin ns = (start)? 3'd1 : 3'd0; end
            3'd1: begin ns = (start)? 3'd1 : 3'd2; end
            3'd2: begin ns = 3'd3; end
            3'd3: begin ns = 3'd4; end
            3'd4: begin ns = (eng_done)? 3'd0 : 3'd4; end
        endcase
    end

    always@(ps)begin
        {inc_count, eng_start, done} = 3'd000;
        case(ps)
            3'd0:begin
                done = 1'b1;
            end
            3'd2:begin
                inc_count = 1'b1;
            end
            3'd3:begin
                eng_start = 1'b1;
            end
        endcase
    end


    always@(posedge clk,posedge rst)begin
        if(rst == 1'b1)
            ps <= 3'd0;
        else
            ps <= ns;
    end
    assign rst_count = rst;
endmodule
```

Fig. 9 Wrapper Verilog

D. Wrapper Verilog

Test Bench Verilog is shown in Fig 10.

```
`timescale 1ns/1ns
module WTB();
    reg clk=1'b0, rst=1'b0, start=1'b0, eng_done=1'b1;
    wire done, eng_start, rst_count;
    reg inc_start=1'b0;
    wrapper inst(.clk(clk), .rst(rst), .start(start),
    .eng_done(eng_done), .done(done), .eng_start(eng_start),
    .inc_count(inc_count), .rst_count(rst_count));
    initial repeat(10000) #50 clk = ~clk;
    initial begin
        #1000 start = 1'b1;
        #220 start = 1'b0;
        #50 eng_done = 1'b0;
        #7000 eng_done = 1'b1;
    end
endmodule
```

Fig. 10 Wrapper TB Verilog

E. Wrapper Test Bench

The Test Bench result is shown in Fig 11.

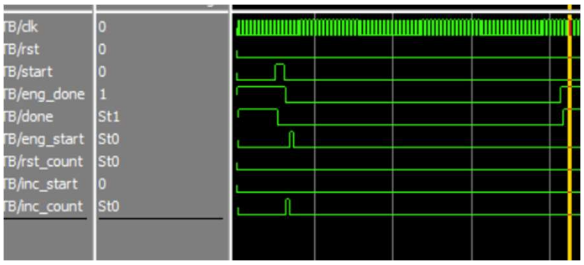

Fig. 11 Wrapper TB result

IV. TOTAL DESIGN

A. Block Diagram

To make Total Design in Quartus, we added parts of Wrapper and Exponential Engine and ROM and an 8-bit Counter to Quartus.
We got three inputs: start, clock and reset.
We gave the clock to all four parts and reset to Wrapper and Exponential Engine. And we connected the start only to the Wrapper.
We connected enable and reset of the Counter to the Wrapper and gave the output of the Counter to the ROM Address.
We gave the output of ROM, which is a number whose it's exponent should be calculated, to the input of the Exponential Engine and also took the Exp. Engine start from the Wrapper.
And we connected the done of Exp. Engine to the Wrapper.
In the end, we created three outputs. A one-bit output for the Total Design done, a 2-bit output for the Integer part of the answer and a 16-bit output for the Decimal part of the answer.
The Total Design is shown in Fig 12.

Fig.

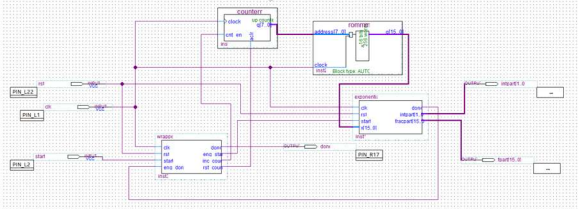
Fig. 12 Total Design

V. FPGA IMPLEMENTATION

A. Pin Planner

We connected clock to 50 MHz FPGA clock.
We connected done to the most left LED and 17 bits of the final answer to right of it. The two LEDs after done connected for 2 integer part of answer and the 15 right LEDs for 15 decimal part of answer. The 16th decimal is connected to the first seven segment.

clk	Unknown	PIN_L1	3.3-V ...fault)
done	Unknown	PIN_R17	3.3-V ...fault)
rst	Unknown	PIN_L22	3.3-V ...fault)
start	Unknown	PIN_L2	3.3-V ...fault)
intpart[1]	Unknown	PIN_R18	3.3-V ...fault)
intpart[0]	Unknown	PIN_U18	3.3-V ...fault)
fpart[15]	Unknown	PIN_Y18	3.3-V ...fault)
fpart[14]	Unknown	PIN_V19	3.3-V ...fault)
fpart[13]	Unknown	PIN_T18	3.3-V ...fault)
fpart[12]	Unknown	PIN_Y19	3.3-V ...fault)
fpart[11]	Unknown	PIN_U19	3.3-V ...fault)
fpart[10]	Unknown	PIN_R19	3.3-V ...fault)
fpart[9]	Unknown	PIN_R20	3.3-V ...fault)
fpart[8]	Unknown	PIN_Y21	3.3-V ...fault)
fpart[7]	Unknown	PIN_Y22	3.3-V ...fault)
fpart[6]	Unknown	PIN_W21	3.3-V ...fault)
fpart[5]	Unknown	PIN_W22	3.3-V ...fault)
fpart[4]	Unknown	PIN_V21	3.3-V ...fault)
fpart[3]	Unknown	PIN_V22	3.3-V ...fault)
fpart[2]	Unknown	PIN_U21	3.3-V ...fault)
fpart[1]	Unknown	PIN_U22	3.3-V ...fault)
fpart[0]	Unknown	PIN_J2	3.3-V ...fault)

Fig. 13 Pin Planner

LED	17	16	15	14	13	12	11	10	9
Source	done	int[1]	int[0]	dec[15]	dec[14]	dec[13]	dec[12]	dec[11]	dec[10]
LED	8	7	6	5	4	3	2	1	0
Source	dec[9]	dec[8]	dec[7]	dec[6]	dec[5]	dec[4]	dec[3]	dec[2]	dec[1]
Seven segment									
dec[0]									

Fig. 14 Pin on FPGA

B. Implementation

After Implementation, we tested the results on FPGA.
In the Fig 15, the first FPGA state is indicated. As you can see, LED done is on and FPGA is waiting for start to start its work.

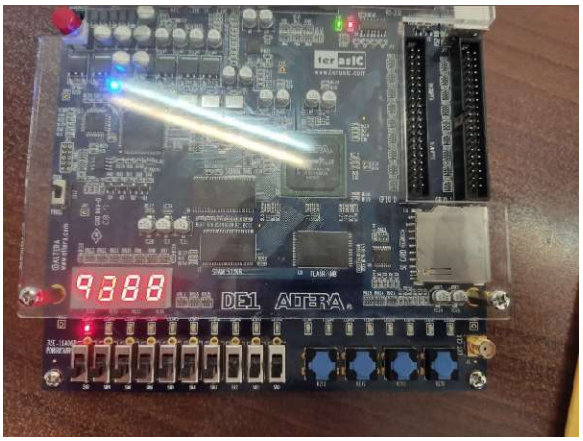

Fig. 15 FPGA wait for start

The first number that the Accelerator should be calculate was 16'd65536 which is equal to 0,5. In Figure 17, the answer can be seen in ModelSim. In Fig 16 you can the result on FPGA.

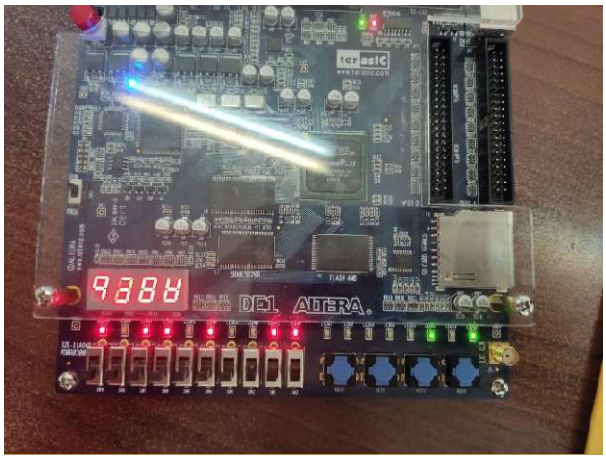

Fig. 16 FPGA test

/exponential/x	1000000000000000
/exponential/done	St1
/exponential/intpart	01
/exponential/fracpart	1010011000001011

Fig. 17 ModelSim Test Result

The second number was all 1. In Figure 18, the answer can be seen in ModelSim. In Fig 19 you can the result on FPGA.

/exponential/x	1111111111111111
/exponential/done	St1
/exponential/intpart	10
/exponential/fracpart	1011011111010101

Fig. 18 ModelSim Test Result

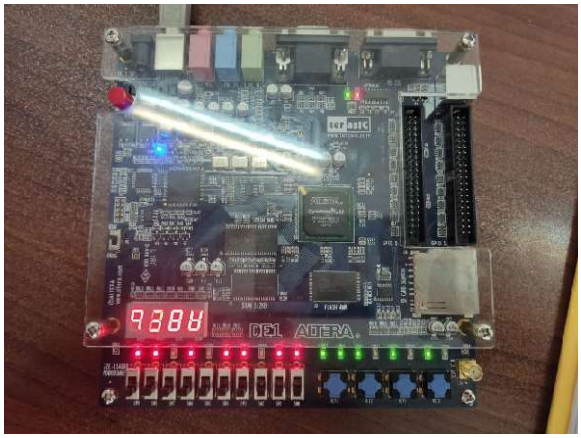

Fig. 19 FPGA Test Result