Experiment 4 - Accelerator and Wrappers

Mohammad Ghareh Hasanloo, 810198461

Danial Saeedi, 810198571

Abstract— **This is the report of experiment 4 and in this experiment, at first a testbench is implemented for exponential engine functionality. Then exponential accelerator wrapper is added to control handshaking by a controller and an input buffer. At last the accelerator is implemented on FPGA to see output with LEDs.**

Keywords— **Exponential Engine, Exponential Accelerator Wrapper, Accelerator Buffer, Huffman Style, Handshaking**

1 Exponential Engine

In this part a testbench is written to check functionality of exponential engine. Code and its functionality can be seen below.

exponential_TB.v

```
`timescale 1ns/1ns

module exponential_TB();
    reg start=0, clk=0, rst=0;
    reg [15:0] x;
    wire done;
    wire [1:0] intpart;
    wire [15:0] fracpart;

    exponential exp(.x(x), .start(start), .clk(clk),
     .rst(rst), .done(done), .intpart(intpart), .fracpart(fracpart));

    always #10 clk = ~clk;

    initial begin
        #20 x = 16'h8000; start = 1;
        #20 start = 0;
        while (~done) #20;
        #20 x = 16'b1000000000000001; start = 1;
        #20 start = 0;
        while (~done) #20;
        #20 x = 16'b1000000000000010; start = 1;
        #20 start = 0;
        while (~done) #20;
        #100 $stop;

    end
endmodule
```

Fig. 1 Testbench

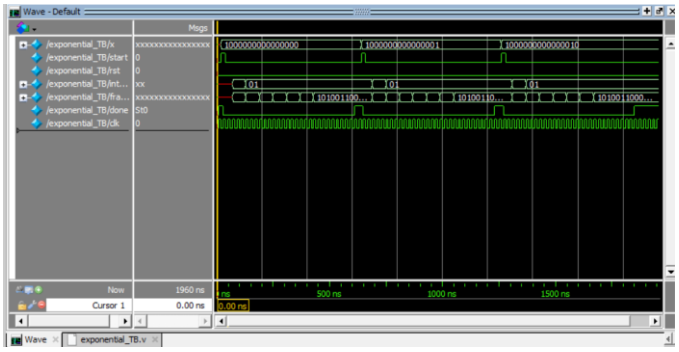

Fig. 2 Testbench Result with Three Inputs

The maximum frequency of this accelerator after synthesizing design is shown below.

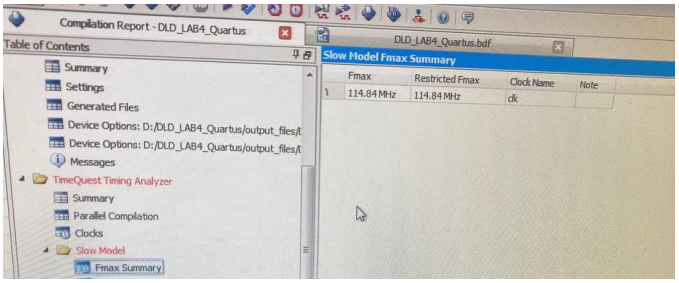

Fig. 3 Maximum Frequency Calculated by Quartus

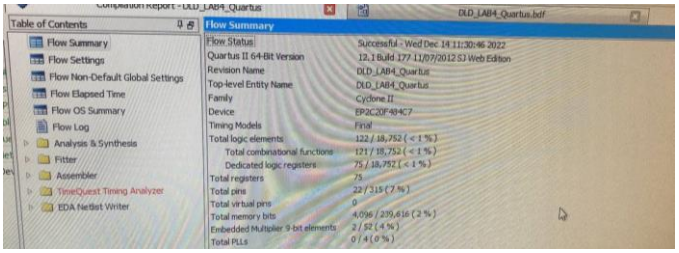

Fig. 4 Synthesis's Result

2 Exponential Accelerator Wrapper

This part is designed to accelerator calculates multiple exponential values. CPU saves input values into input buffer and exponential engine uses these values to compute corresponding outputs. In this project, an input buffer is just implemented and the memory element is ROM. A top level wasn't implemented because of Quartus option to make top

level. The input values are stored as mif file inside ROM. The ROM is shown below.

ROM.mif - Notepad

File Edit Format View Help

```
WIDTH=16;
DEPTH=256;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN

00: 8000;
01: CCCC;
02: 3333;
03: FD70;
```

Fig. 5 ROM Contents

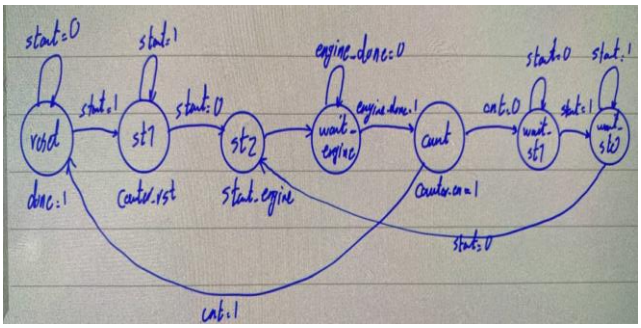

Fig. 6 State Diagram of the Controller

WrapperController.v

```
module WrapperController(
    input start, engDone, clk, rst ,incCounter,
    output reg done, engStart, rstCounter, cnt_en
);
    reg [2:0] ps, ns;

    always @(posedge clk, posedge rst) begin
        if (rst) ps <= `reset;
        else ps <= ns;
    end

    always @(ps, start, engDone,incCounter) begin
        case (ps)
            `reset:    ns = start ? `st1 : `reset;
            `st1:    ns = start ? `st1 : `st2;
            `st2: ns = `wait_engine;
            `wait_engine:   ns = engDone ? `count : `wait_engine;
            `count:    ns = incCounter ? `reset : `wait_st1;
            `wait_st1:   ns = start ? `wait_st2 : `wait_st1;
            `wait_st2:    ns = start ? `wait_st2 : `st2;
            default:;
        endcase
    end

    always @(ps) begin
        {done, engStart, rstCounter, cnt_en} = 4'b0;
        case (ps)
            `reset: done = 1'b1;
            `st1: rstCounter = 1'b1;
            `st2: engStart = 1'b1;
            `wait_engine: ;
            `count: cnt_en = 1'b1;
            `wait_st1: ;
            `wait_st2: ;
            default:;
        endcase
    end
endmodule
```

Fig. 7 Wrapper's Controller

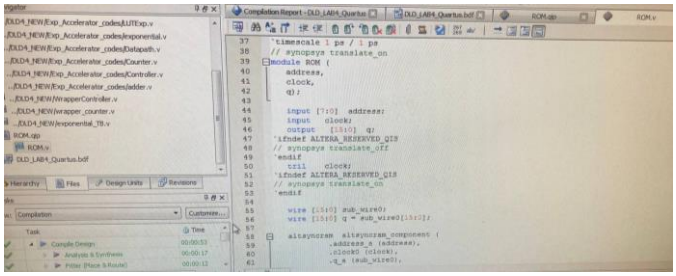

Fig. 8 Instance of ROM IP

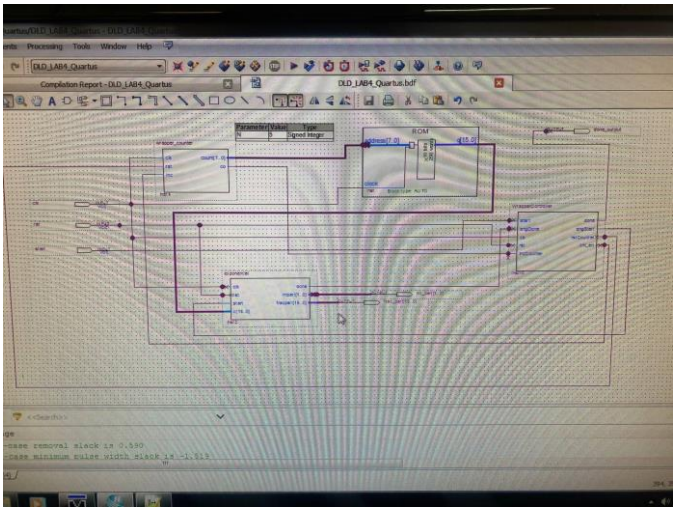

Fig. 9 Block Diagram

3 Implementing Accelerator on FPGA

The wrapper is synthesized and implemented on FPGA board and the outputs are shown step by step below. the inputs has been shown in figure 5.

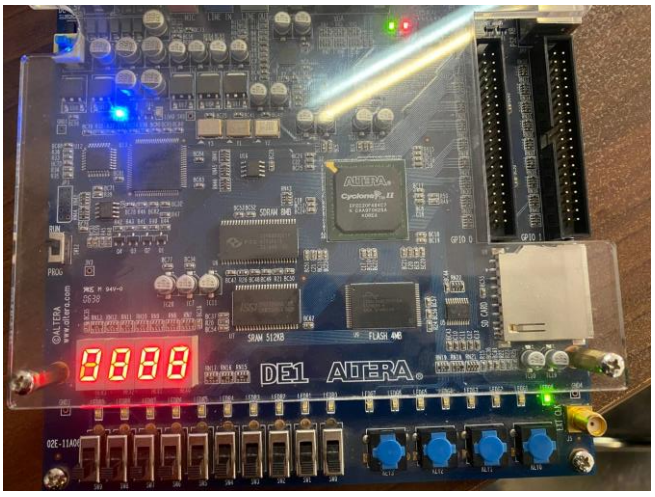

Fig. 10 The Picture before Starting Calculations

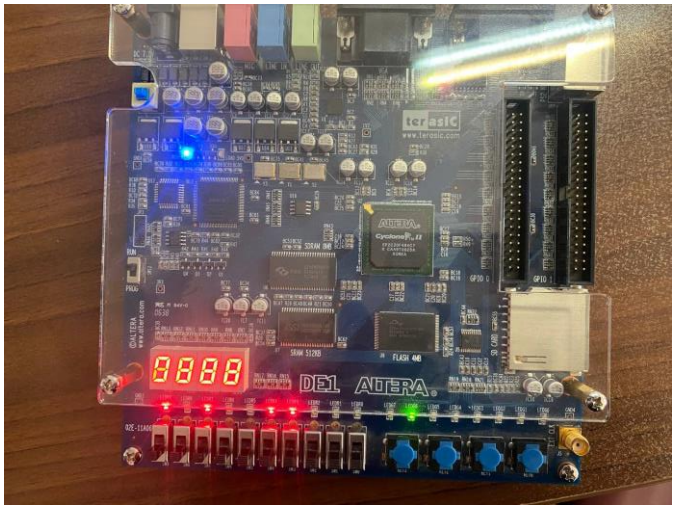

Fig. 11 Output for First Input

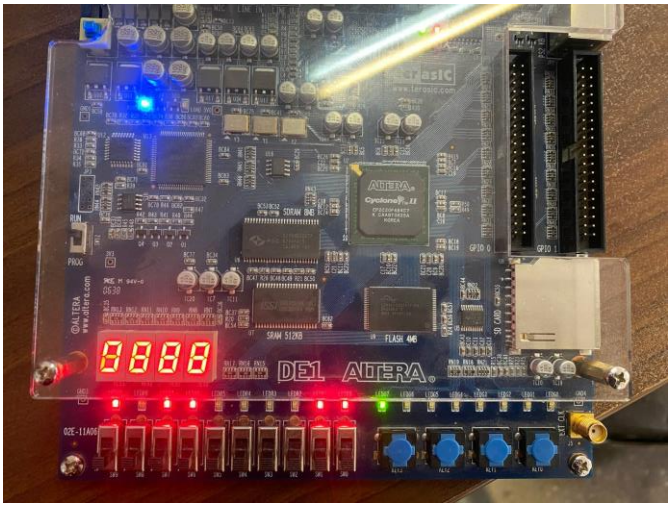

Fig. 14 Output for Forth Input

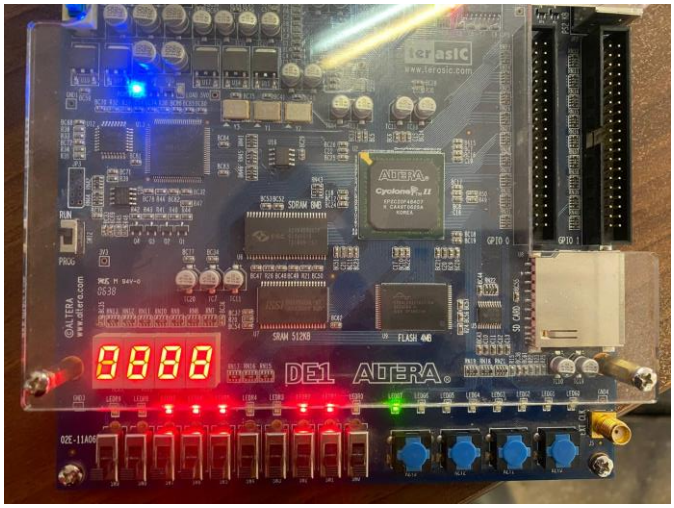

Fig. 12 Output for Second Input

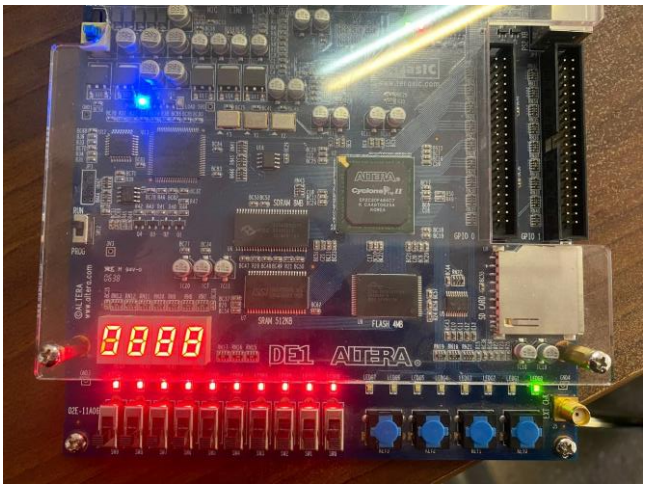

Fig. 15 Board after Finishing Calculations

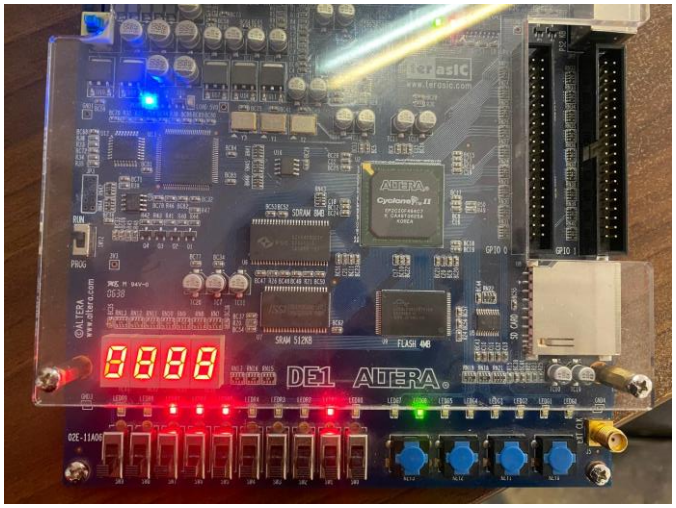

Fig. 13 Output for Third Input

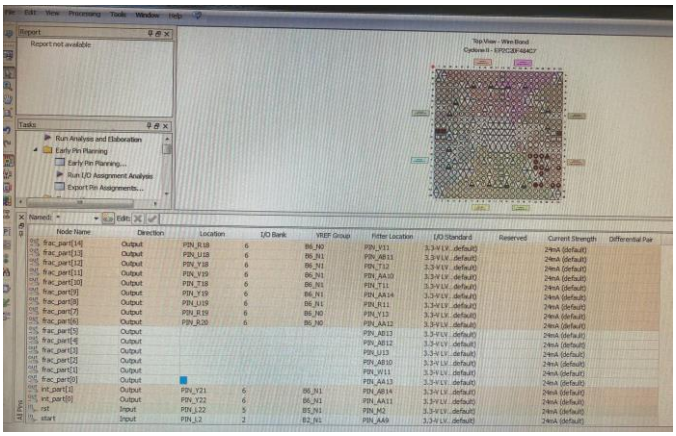

Fig. 16 Pin Assignments