Experiment 4 - Accelerator and Wrappers

Pasha Barahimi, 810199385
Misagh Mohaghegh, 810199484

***Abstract*— This document is a student report for the 4th experiment of the Digital Logic Laboratory course (ECE 045) at University of Tehran, Department of Electrical and Computer Engineering.**

***Keywords*— Exponential Engine, Accelerator Buffer, Accelerator Wrapper, FPGA Implementation, Altera Cyclone II, Quartus, ModelSim**

I. Exponential Engine

A. Simulation

The following test-bench was written for the purpose of testing the provided exponential engine.

Three different values are given to the engine's input and the output is calculated and verified.

```verilog
`timescale 1ns/1ns

module ExpEngTB();
    reg start, clk, rst;
    reg [15:0] inp;

    wire done;
    wire [1:0] intpart;
    wire [15:0] fracpart;

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

    always #5 clk = ~clk;

    initial begin
        #0 clk = 0; rst = 0; start = 0;
        #3 rst = 1;
        #3 rst = 0;

        #10 inp = 16'h8000; start = 1;
        #10 start = 0;
        while (~done) #10;
        #100;

        #10 inp = 16'hCCCC; start = 1;
        #10 start = 0;
        while (~done) #10;
        #100;

        #10 inp = 16'h3333; start = 1;
        #10 start = 0;
        while (~done) #10;
        #100;

        #100 $stop;
    end
endmodule
```

Fig. 1 Verilog test-bench for the exponential engine

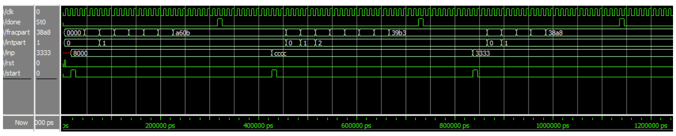

Fig. 2 Simulation results

The results are very close to the expected output:
Input 0x8000: 0x1A5E3
Input 0xCCCC: 0x233F7
Input 0x3333: 0x13581
The diffrenece is due to the exponential engine's accuracy.

B. Synthesis

1) Results:

Flow Summary	
Flow Status	Successful - Tue May 17 05:52:58 2022
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	ExpAcc
Top-level Entity Name	ExpEngine
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	102 / 18,752 (< 1 %)
Total combinational functions	100 / 18,752 (< 1 %)
Dedicated logic registers	61 / 18,752 (< 1 %)
Total registers	61
Total pins	38 / 315 (12 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	2 / 52 (4 %)
Total PLLs	0 / 4 (0 %)

Fig. 3 Synthesis results

2) Maximum Frequency:

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	113.22 MHz	113.22 MHz	clk	

Fig. 4 Maximum frequency

II. Exponential Engine Wrapper

A. Buffers

1) ROM:

The following .mif file is used for the initialization of the input read-only memory:

```
WIDTH=16;
DEPTH=8;

ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;

CONTENT BEGIN

00: 8000;  -- e^0.50 = 1.648 = 0x1A5E3
01: CCCC;  -- e^0.79 = 2.203 = 0x233F7
02: 3333;  -- e^0.19 = 1.209 = 0x13581
03: FD70;  -- e^0.99 = 2.691 = 0x2B0E5
04: 0000;  -- e^0.00 = 1.000 = 0x10000
05: 0000;
06: 0000;
07: 0000;

END;
```

Fig. 5 ROM .mif file

The first 5 memory locations are filled with different values to test the functionality of the exponential engine.

The calculation result is also commented for verification.

2) FIFO:

A FIFO queue is used for the output buffer.

After every read signal from the user, its rdreq signal is issued which will dequeue the first input's result.

B. Controller

1) State Diagram:

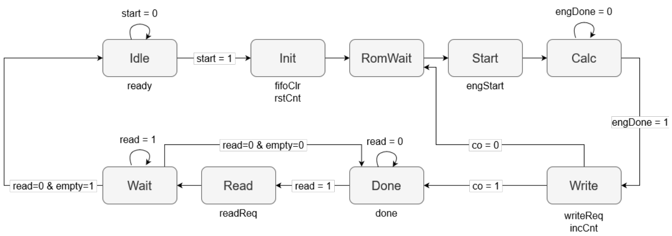

Fig. 6 Wrapper controller state diagram

2) Behavioral Description:

```
`define Idle    4'b0000
`define Init    4'b0001
`define RomWait 4'b0010
`define Start   4'b0011
`define Calc    4'b0100
`define Write   4'b0101
`define Done    4'b0110
`define Read    4'b0111
`define Wait    4'b1000

module WrapperController(
    input start, engDone, coCnt, read, empty, clk, rst,
    output reg done, ready, engStart, rstCnt, incCnt, writeReq, readReq, fifoClr
);
    reg [3:0] ps, ns;

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

    always @(ps, start, engDone, coCnt, read, empty) begin
        case (ps)
            `Idle:    ns = start ? `Init : `Idle;
            `Init:    ns = `RomWait;
            `RomWait: ns = `Start;
            `Start:   ns = `Calc;
            `Calc:    ns = engDone ? `Write : `Calc;
            `Write:   ns = coCnt ? `Done : `RomWait;
            `Done:    ns = read ? `Read : `Done;
            `Read:    ns = `Wait;
            `Wait:    ns = read ? `Wait : (empty ? `Idle : `Done);
            default:;
        endcase
    end

    always @(ps) begin
        {done, ready, engStart, rstCnt, incCnt, writeReq, readReq, fifoClr} = 8'd0;
        case (ps)
            `Idle: ready = 1'b1;
            `Init: {rstCnt, fifoClr} = 2'b11;
            `RomWait:;
            `Start: engStart = 1'b1;
            `Calc:;
            `Write: {writeReq, incCnt} = 2'b11;
            `Done: done = 1'b1;
            `Read: readReq = 1'b1;
            `Wait:;
            default:;
        endcase
    end
endmodule
```

Fig. 7 Wrapper controller Verilog description

3) Block Diagram:

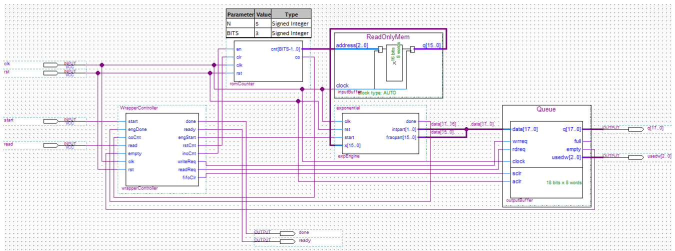

Fig. 8 The accelerator block diagram

4) Simulation:

The above accelerator was synthesized and a test-bench was written for it:

```verilog
`timescale 1ns/1ns

module ExpAccTB();
    reg start, read, clk, rst;

    wire done, ready;
    wire [17:0] q;
    wire [2:0] usedw;

    ExpAcc acc(.done(done), .start(start), .clk(clk), .rst(rst), .read(read), .ready(ready), .q(q), .usedw(usedw));

    always #5 clk = ~clk;

    initial begin
        #0 clk = 0; rst = 0; start = 0; read = 0;
        #3 rst = 1;
        #3 rst = 0;
        #10 start = 1;
        #10 start = 0;
        while (~done) #10;
        #30 read = 1;
        #30 read = 0;
        #30 read = 1;
        #30 read = 0;
        #30 read = 1;
        #30 read = 0;
        #30 read = 1;
        #30 read = 0;
        #30 read = 1;
        #30 read = 0;
        #100 $stop;
    end
endmodule
```

Fig. 9 Accelerator test-bench

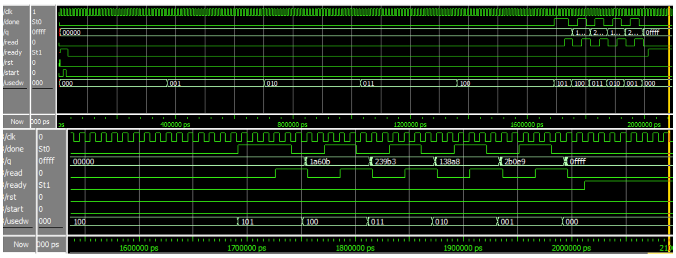

Fig. 10 Simulation results

III. FPGA IMPLEMENTATION

A. *Pin Planner*

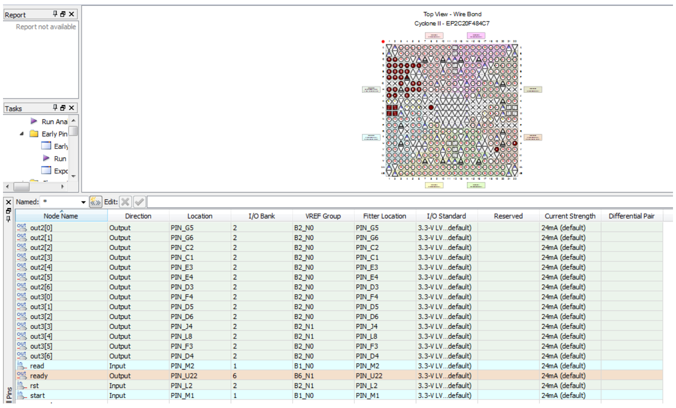

Fig. 11 Quartus pin planner. Assigning toggle buttons to our inputs, connecting the output to the 7-segment displays, using a green LED for the *ready* signal, and a red LED for the *done* signal.

B. *Implementation*

The accelerator was put in a top-level module dividing and connecting its output to 4 HexDisplay modules.

1) *HEX Display:*

The exponential engine output is 18-bits. 2 int parts and 16 fractional parts.

The output here is shown on the 7-segment displays. The first display (leftmost) shows the int part and the rest of the displays each show 4 bit of the fractional part.

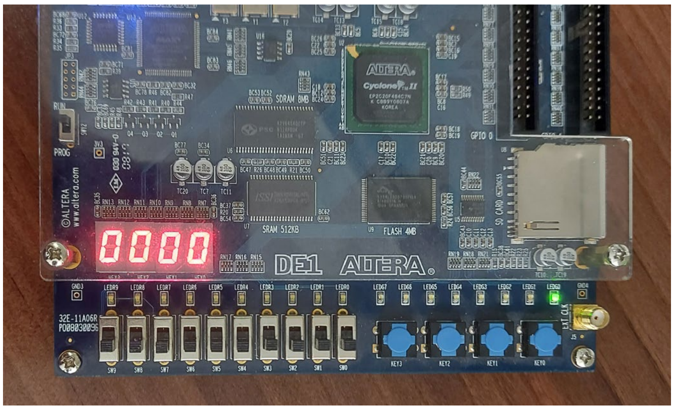

Fig. 12 Ready signal is issued

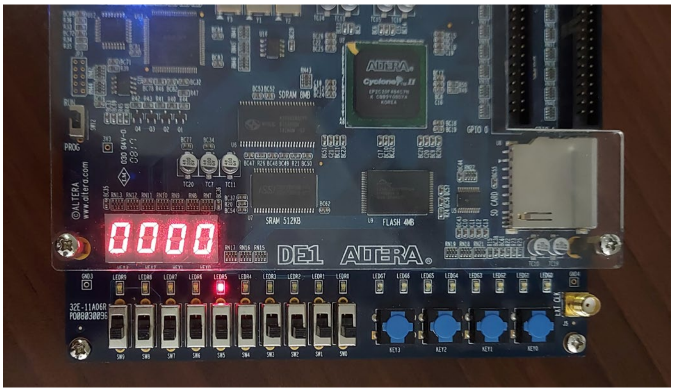

Fig. 13 After a full pulse on the start input, the done signal is issued meaning that the calculations are done

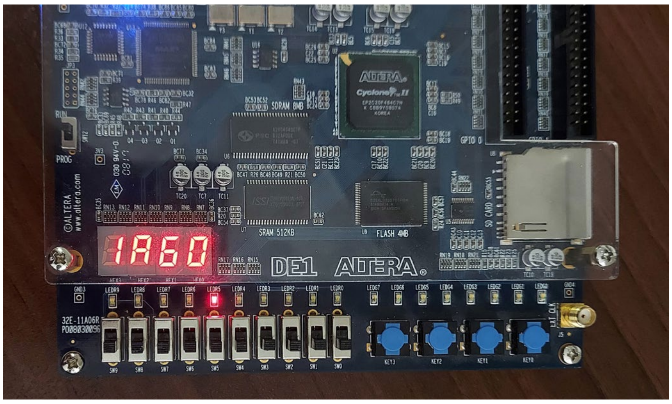

Fig. 14 Read 1

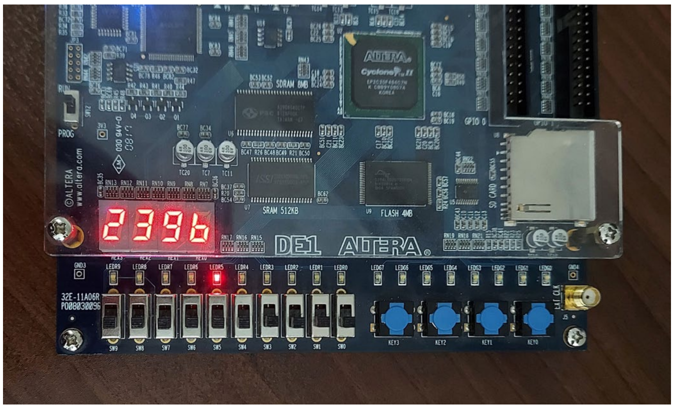

Fig. 15 Read 2

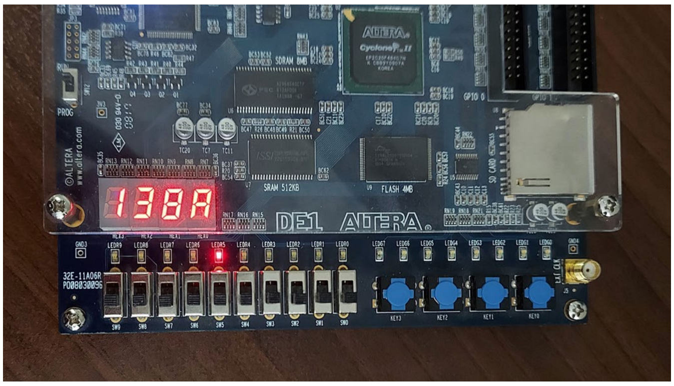

Fig. 16 Read 3

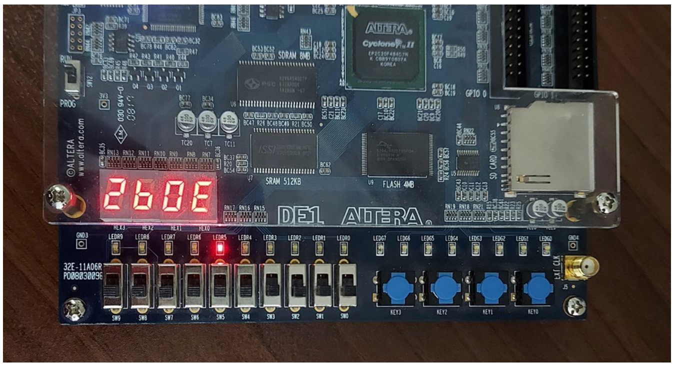

Fig. 17 Read 4

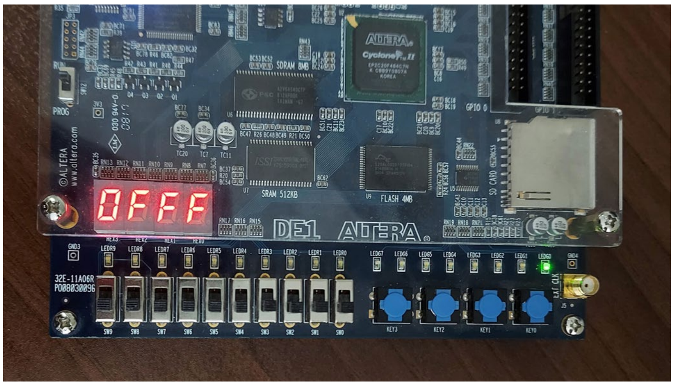

Fig. 18 Read 5. The controller is back to its idle state and the ready signal is issued again

2) Decimal Display:

Here, the output shown on the 7-segment displays is in a BCD format as follows: (considering d0 to be the leftmost display)

d0: Int part
d1: Fractional part
d2: H for separating
d3: The FIFO structure's used words

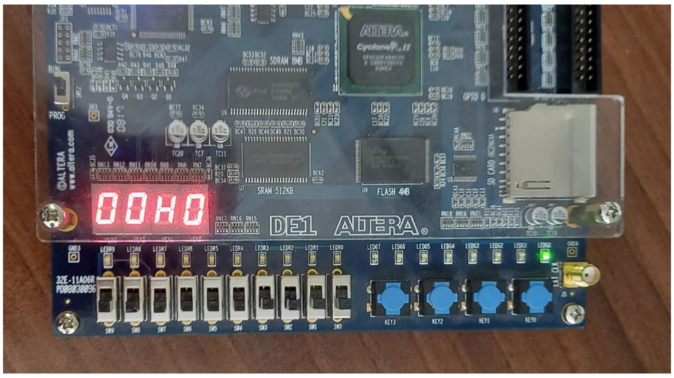

Fig. 19 Ready signal is issued

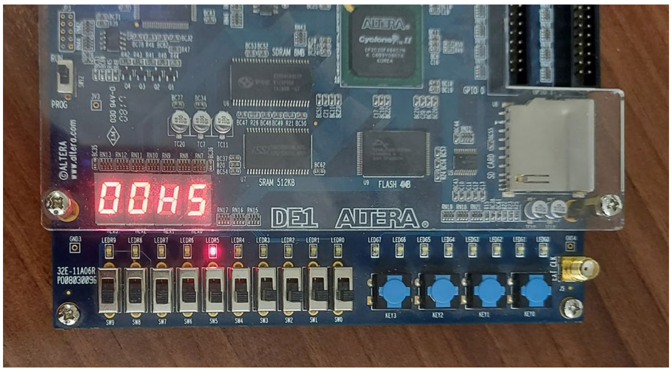

Fig. 20 After a full pulse on the start input, the done signal is issued meaning that the calculations are done. The number 5 is shown for the used words' count, which means that there are 5 entries in the output FIFO

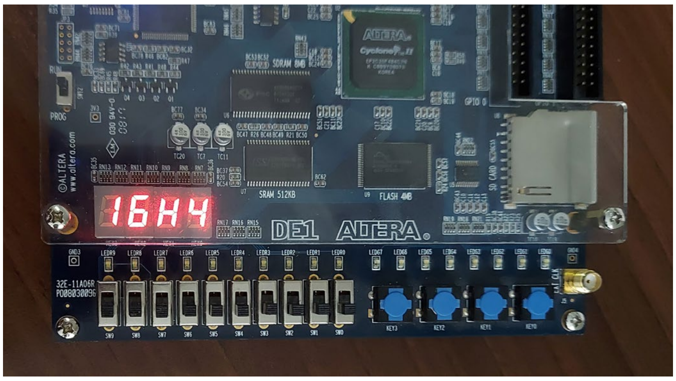

Fig. 21 Read 1. An entry from the FIFO is popped thus showing 4 used words. The first two displays are the int and fraction part of the output. 1.6 is the result here

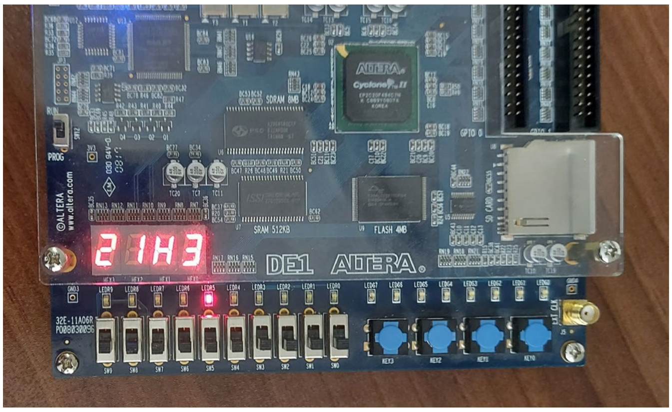

Fig. 22 Read 2

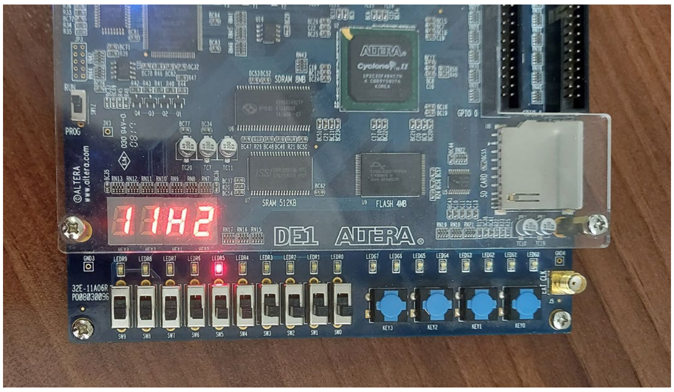

Fig. 23 Read 3

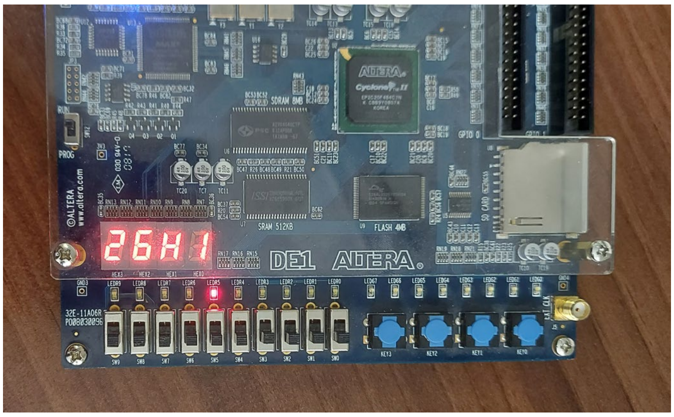

Fig. 24 Read 4

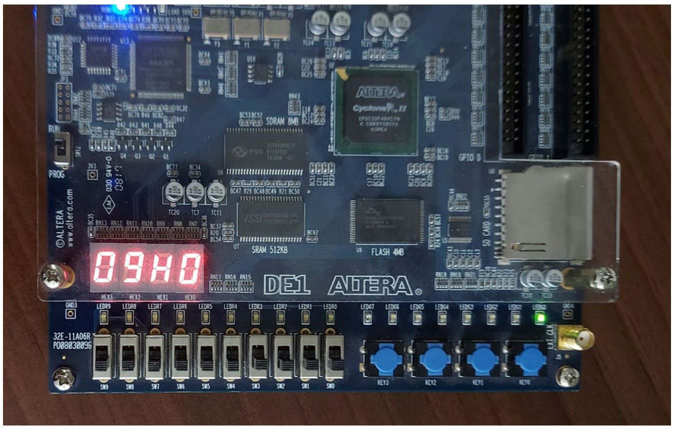

Fig. 25 Read 5. The controller is back to its idle state and the ready signal is issued again. The FIFO is now empty