Experiment #4 – Accelerator and Wrappers

Ava Mirmohammadmahdi 810199501
Nesa Abbasi 810199457

1. Exponential Engine

1.1

```
`timescale 1ns/1ns

module ExpTB();
    wire [15:0] fractionalPart;
    wire [1:0] integerPart;
    wire done;

    reg [15:0] x;
    reg clk = 1'b0, rst = 1'b0, start = 1'b0;

    exponential Exp(.x(x), .start(start), .clk(clk), .rst(rst), .done(done), .intpart(integerPart), .fracpart(fractionalPart));

    always #5 clk = ~clk;

    initial begin
        #3 rst = 1'b1;
        #3 rst = 1'b0;
        #10 x = 16'h8000;
        #0 start = 1'b1;
        #10 start = 1'b0;
        while (~done) #10;
        #100;
        #10 x = 16'hCCCC;
        #0 start = 1'b1;
        #10 start = 1'b0;
        while (~done) #10;
        #100;
        #10 x = 16'h3333;
        #0 start = 1'b1;
        #10 start = 1'b0;
        while (~done) #10;
        #100;
        #10 x = 16'hFD70;
        #0 start = 1'b1;
        #10 start = 1'b0;
        while (~done) #10;
        #100;
        #10 x = 16'h028F;
        #0 start = 1'b1;
        #10 start = 1'b0;
        while (~done) #10;
        #100;
        #100 $stop;
    end
endmodule
```

Fig. 1 Verilog Testbench

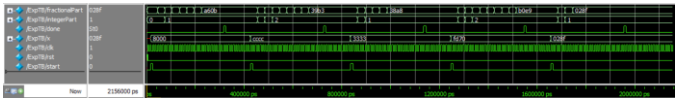

Fig. 2 Simulation

Input 0x8000 ➞ 0x0A5E3
Input 0xCCC ➞ 0x233F7
Input 0x3333 ➞ 0x13581
Input 0xFD70 ➞ 0x2B01E5
Input 0x028F ➞ 0x10000
The difference is because of the exponential engine.

1.2

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 result

1.3

Slow Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	113.22 MHz	113.22 MHz	clk	

Fig. 4 Maximum frequency

2. The Wrapper Controller

2.1

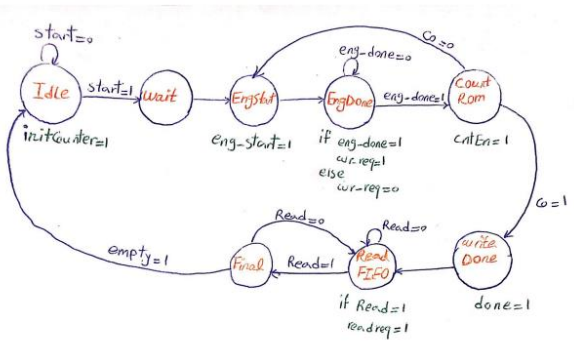

Fig. 5 State Diagram

```
module ControllerAcc(input clk, rst, start, read, eng_done, empty, output reg eng_start, writeReq, done, readReq,
output reg[2:0] counter);
    reg [2:0] ps, ns;
    reg cntEn, initCounter;
    wire co;
    parameter [2:0] Idle = 0, Wait = 1, EngStart = 2, EngDone = 3, CountRom = 4, WriteDone = 5, ReadFIFO = 6,
              Final = 7;
    always @(ps, start, eng_done, co, read, empty)begin
        ns = Idle;
        {initCounter, cntEn, writeReq, eng_start, done, readReq} = 7'd0;
        case(ps)
            Idle: begin
                ns = start ? Wait : Idle;
                initCounter = 1'b1;
            end
            Wait: begin
                ns = EngStart;
            end
            EngStart: begin
                ns = EngDone;
                eng_start = 1'b1;
            end
            EngDone: begin
                ns = eng_done ? CountRom : EngDone;
                writeReq = eng_done ? 1'b1 : 1'b0;
            end
            CountRom: begin
                ns = co ? WriteDone : EngStart;
                cntEn = 1'b1;
            end
            WriteDone: begin
                ns = ReadFIFO;
                done = 1'b1;
            end
            ReadFIFO: begin
                ns = read ? Final : ReadFIFO;
                readReq = read ? 1'b1 : 1'b0;
        done = 1'b1;
            end
            Final: begin
                ns = empty ? Idle : ~read ? ReadFIFO : Final;
        done = 1'b1;
            end
        endcase
    end

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



    always @(posedge clk , posedge rst) begin
        if(rst)
            counter <= 3'd0;
        else if(initCounter)
            counter <= 3'd0;
        else if(cntEn)
            counter <= counter + 1'b1;
    end
    assign co = &counter;
endmodule
```

Fig. 6 Controller Verilog

2.2

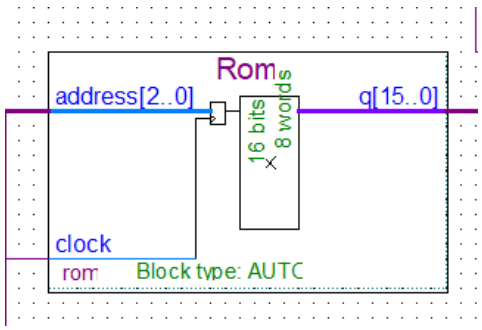

Fig. 7 Rom IP

```
WIDTH=16;
DEPTH=8;
ADDRESS_RADIX=HEX;
DATA_RADIX=HEX;
CONTENT BEGIN
00 : 8000;
01 : CCCC;
02 : 3333;
03 : FD70;
04 : 028F;
05 : 0000;
06 : 0000;
07 : 0000;
END;
```

Fig. 8 Mif File

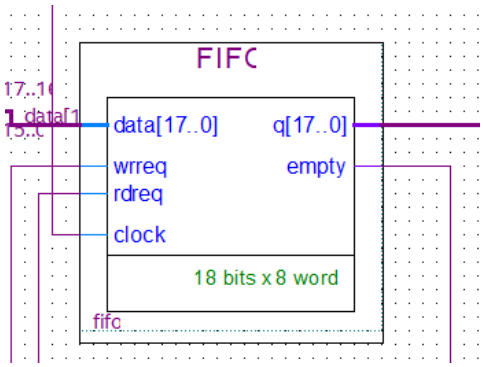

Fig. 9 FIFO IP

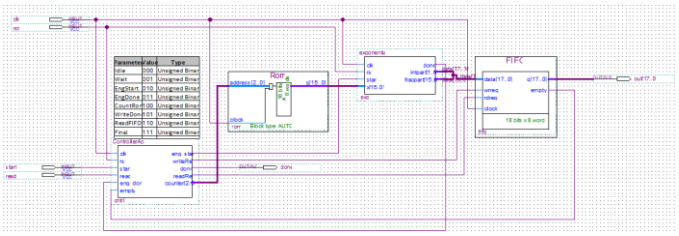

Fig. 10 The accelerator block diagram

2.4

```
module AcceleratorTB();
    reg start, read, clk, rst;

    wire done;
    wire [17:0] out;

    Accelerator acc(.done(done), .start(start), .clk(clk), .rst(rst), .read(read), .out(out));

    always #5 clk = ~clk;

    initial begin
        #0 clk = 0; rst = 0; start = 0; read = 0;
        #3 rst = 1;
        #3 rst = 0;
        #10 start = 1;
        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;
        #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. 11 Accelerator Testbench

2.5

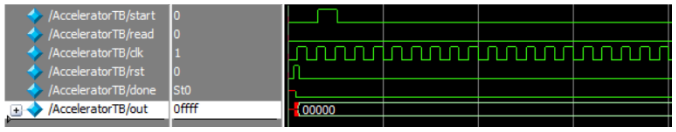

Fig. 12 Complete pulse on signal “start”

2.6

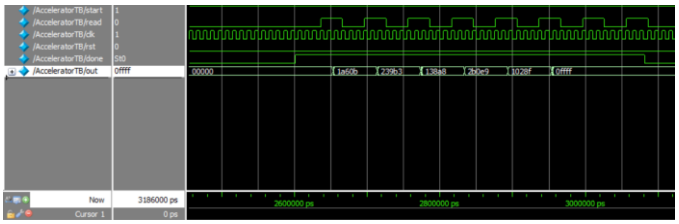

Fig. 13 Simulation result

3.Implementing Accelerator on FPGA

3.1

Flow Summary	
Flow Status	Successful - Sat Jun 18 20:19:05 2022
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	Accelerator
Top-level Entity Name	Accelerator
Family	Cyclone IV E
Total logic elements	135 / 6,272 (2 %)
Total registers	83
Total pins	23 / 92 (25 %)
Total virtual pins	0
Total memory bits	272 / 276,480 (< 1 %)
Embedded Multiplier 9-bit elements	2 / 30 (7 %)
Total PLLs	0 / 2 (0 %)
Device	EP4CE6E22C6
Timing Models	Final

Fig. 14 Synthesis report

3.2

We connected Done signal to LED 9, Start pin to SW9 and Read to SW0. We read the results one by one by setting SW0 to one. After LED9 turned on.

3.3

```
module Final(input [15:0] in, output [6:0] HEX0, HEX1, HEX2, HEX3);
    Hexdisplay HD1 (in[3:0], HEX0);
    Hexdisplay HD2 (in[7:4], HEX1);
    Hexdisplay HD3 (in[11:8], HEX2);
    Hexdisplay HD4 (in[15:12], HEX3);
endmodule
```

Fig. 15 Converter for 7 Segment display

To convert data for 7 segment display, we omitted 2 least significant bits.

3.4

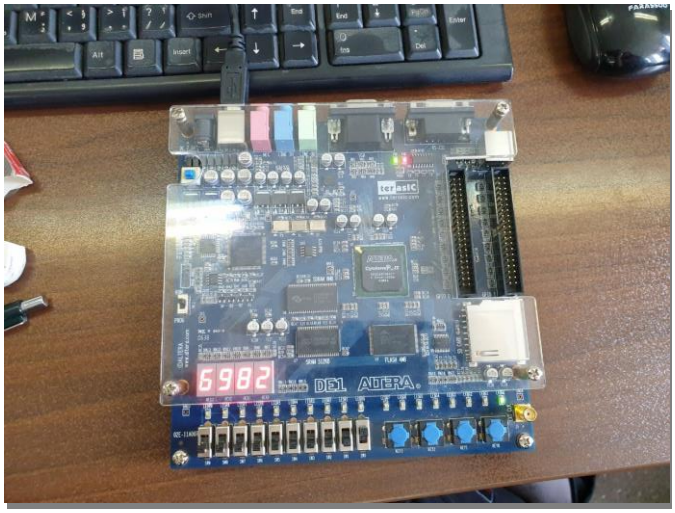

Fig. 16 First output

18 bits binary ➞ Omit 2 least significant bits ➞ Hex

011010011000001011 ➞ 01101001100000101~~11~~ ➞ 6982

Fig. 17 Second output

18 bits binary ➞ Omit 2 least significant bits ➞ Hex

100011100110110011 ➞ 10001110011011001~~11~~ ➞ 8E6C

Fig. 18 Third output

18 bits binary ➞ Omit 2 least significant bits ➞ Hex

010011100010101000 ➞ 01001110001010100~~00~~ ➞ 4E2A

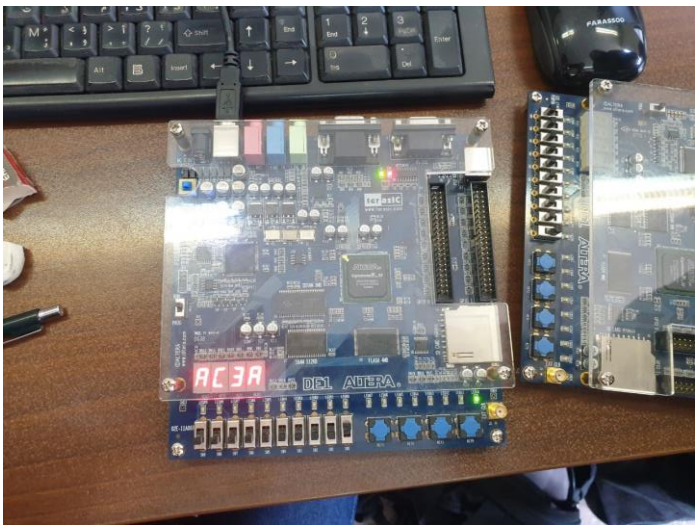

Fig. 19 Fourth output

18 bits binary ➞ Omit 2 least significant bits ➞ Hex

101011000011101001 ➞ 10101100001110100~~01~~ ➞ AC3A

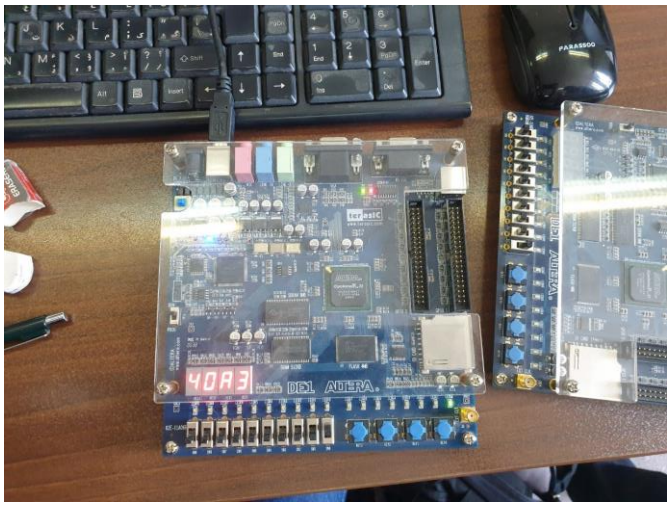

Fig. 20 Fifth output

18 bits binary ➞ Omit 2 least significant bits ➞ Hex

010000001010001111 ➞ 0100000010100011~~11~~ ➞ 40A3

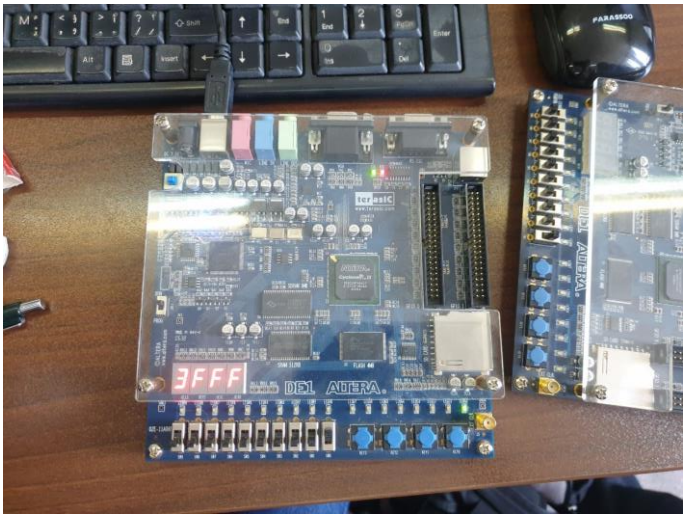

Fig. 21 Sixth output

18 bits binary ➞ Omit 2 least significant bits ➞ Hex

001111111111111111 ➞ 0011111111111111~~11~~ ➞ 3FFF

If we convert our 18 bit input from hex to binary and ommit 2 least significant bits of it and then convert it to hex the output is the same as we ecpected.