Experiment 4 - Accelerator and Wrappers

Digital Logic Design Lab - Fall 2022

Erfan Panahi
ID: 810198369

Sogol Goodarzi
ID: 810198467

ACCELERATOR AND WRAPPERS

INTRODUCTION

By the end of this experiment, you should have learned:

- The concept of an SOC
- The concept of handshaking in an SOC
- The principle of an accelerator

I. EXPONENTIAL ENGINE

Question 1. First examine the code and its accuracy by running Modelsim simulation. For this purpose, write a testbench for this design with at least three different values for input "x". Show the results by taking picture of the simulation results.
For examining the code of Exponential Engine, the test bench in figure 1 is written. The results for different input x, is shown in figure 2.

Question 2. Synthesize this design in Quartus II Software. Show the synthesis results in your report.
The results of synthesizing the circuit in Quartus is shown in figure 3.

Question 3. After synthesizing design, you can find out the maximum frequency of this accelerator by referring to the Timing Analyzer reports in Quartus synthesis tool. You can follow the steps shown in the Figure 3.
By following the steps, we can find the maximum frequency of the accelerator. This frequency is shown in figure 4.

II. EXPONENTIAL ACCELERATOR WRAPPER

Question 1. Show the state diagram of the controller and write the Verilog description of this module in a Huffman style.
The state diagram of the controller for the exponential engine is shown in figure 5. Also, the Verilog description of this module in a Huffman style is illustrated in figure 6.

Question 2. Make instance of ROM IP from Quatus Megawizard. For the ROM IP, generate a 1-PORT 16bit*256Word ROM and initialize it with an mif file containing 5 input values.
For ROM implementation, we first made an instance of ROM IP from Quartus and initialize it with an mif file containing five input values in last 5 addresses. (because our counter

```
`timescale 1 ns/1 ns
module TestBench();
    reg clk = 0;
    reg rst;
    reg start = 0;
    reg [15:0] x;
    wire done;
    wire [1:0] intpart;
    wire [15:0] fracpart;

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

    always
        begin
          #10 clk = ~clk;
      end

    initial begin
      #5 rst = 1;
      #5 rst = 0;
        #5 start = 1'b1;
        #35 start = 1'b0;
        #20 x = 16'h0000;
        #5 start = 1'b1;
        #35 start = 1'b0;
        #2000 x = 16'hFFFF;
        #5 start = 1'b1;
        #35 start = 1'b0;
        #2000 x = 16'h8000;
        #5 start = 1'b1;
        #35 start = 1'b0;
        #2000 x = 16'h2492;
        #2000 x = 16'h2000;
        #5000 $stop;
    end
endmodule
```

Fig. 1. The testbench for the exponential engine

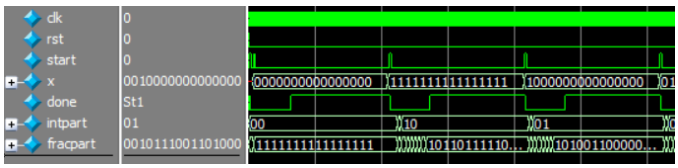

Fig. 2. The waveform of the testbench for the exponential engine

Flow Summary	
Flow Status	Successful - Thu Jan 12 21:56:48 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	exponential
Top-level Entity Name	exponential
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	101 / 18,752 (< 1 %)
Total combinational functions	100 / 18,752 (< 1 %)
Dedicated logic registers	60 / 18,752 (< 1 %)
Total registers	60
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. Compilation summary of the exponential engine

Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	117.14 MHz	117.14 MHz	clk	

Fig. 4. The maximum frequency of the accelerator

is designed to count from 251 to 255 and then repeat this counting again). Figure 7 shows the mif file.

Question 3. Write a Verilog description for the wrapper shown in Figure 4.

Before we create the Verilog description of the wrapper, we have to write the Verilog code for an 8 bit counter. The Verilog description of the wrapper is illustrated in two modules in figure 8.

Question 4. Write a testbench and make instance of this wrapper in your tesbench.

By writing the test bench shown in figure 9, we test the wrapper design and show the results and outputs in figure 10.

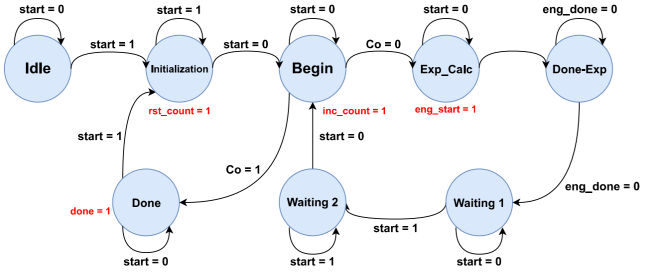

Fig. 5. The state machine of the wrapper

```
module Controller_Exp(input clk, rst, start, Co, eng_done,
            output reg done, inc_count, rst_count, eng_start);
    reg [2:0] ps, ns;
    parameter [2:0] Idle = 0, Initialization = 1, Begin = 2,
    Exp_calc = 3, Done_Exp = 4, Done = 5, Waiting1 = 6, Waiting2 = 7;
    always@(ps, Co, start, eng_done)begin
        ns = Idle;
        case(ps)
            Idle:
                ns = (start) ? Initialization : Idle;
            Initialization:
                ns = (start) ? Initialization : Begin;
            Begin:
                ns = Co ? Done : Exp_calc;
            Exp_calc:
                ns = Done_Exp;
            Done_Exp:
                ns = eng_done ? Waiting1 : Done_Exp;
            Done:
                ns = start ? Initialization : Done;
            Waiting1:
              ns = start ? Waiting2 : Waiting1;
            Waiting2:
              ns = start ? Waiting2 : Begin;
        endcase
    end
    always@(ps, Co, start, eng_done)begin
    done = 1'b0; rst_count = 1'b0; inc_count = 1'b0; eng_start = 1'b0;
        case(ps)
            //Idle:
            //new_state:
            Initialization: begin
                rst_count = 1'b1;
            end
            Begin: begin
                inc_count = 1'b1;
            end
            Exp_calc: begin
                eng_start = 1'b1;
            end
            //Done_Exp:
            Done: begin
                done = 1'b1;
            end
        endcase
    end
    always@(posedge clk,posedge rst)begin
        if(rst == 1'b1)
            ps <= Idle;
        else
            ps <= ns;
    end
endmodule
```

Fig. 6. The controller code of the wrapper

```
-- Quartus II generated Memory Initialization File (.mif)

WIDTH=16;
DEPTH=256;

ADDRESS_RADIX=UNS;
DATA_RADIX=UNS;

CONTENT BEGIN
    [0..251]  :   0;
    252   :   65535;
    253   :   32768;
    254   :   21845;
    255   :   16384;
END;
```

Fig. 7. The mif file of the ROM

```
module Datapath_Exp(input clk, rst, rst_count, inc_count, eng_start
                    ,output Co, eng_done, output[1:0] intpart, output[15:0] fracpart);
    wire [7:0] address;
    wire [15:0] data;
    Count_Exp cntexp(clk, rst_count, inc_count,
                     address, Co);
    ROM rom1(clk, address, data);
    exponential exp(clk, rst, eng_start, data,
                    eng_done, intpart, fracpart);
endmodule

module Exp_Accelerator(input clk, rst, start,
                       output done, output[1:0] intpart, output [15:0]fracpart);
    wire Co, eng_start, eng_done, inc_count, rst_count;
    Controller_Exp controlexp(clk, rst, start, Co, eng_done,
                              done, inc_count, rst_count, eng_start);
    Datapath_Exp dPexp(clk, rst, rst_count, inc_count, eng_start
                       ,Co, eng_done, intpart, fracpart);
endmodule
```

Fig. 8. The datapath and top-module of the wrapper

```
`timescale 1ns/1ns
module Exp_Accelerator_TB();
    reg clk = 0;
    reg rst = 0;
    reg start = 0;
    wire done;
    wire [1:0] intpart;
    wire [15:0] fracpart;
    Exp_Accelerator exp_acc(clk, rst, start,
                            done, intpart, fracpart);
    always
        begin
          #10 clk = ~clk;

    end
    initial begin
        #5 rst = 1;
        #5 rst = 0;
        #15 start = 1'b1;
        #25 start = 1'b0;
        #2000 start = 1'b1;
        #25 start = 1'b0;
        #2000 start = 1'b1;
        #25 start = 1'b0;
        #2000 start = 1'b1;
        #25 start = 1'b0;
        #2000 start = 1'b1;
        #25 start = 1'b0;
        #2000 start = 1'b1;
        #25 start = 1'b0;
        #2000 start = 1'b1;
        #25 start = 1'b0;
        #2000 start = 1'b1;
        #25 start = 1'b0;
        #2000 $stop;
    end
endmodule
```

Fig. 9. The testbench of the wrapper (Pre-synthesis)

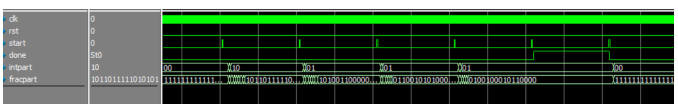

Fig. 10. The result of the wrapper (Pre-synthesis)

Question 5. Generate a complete pulse on the signal "start" for the accelerator wrapper.

By giving the values 1 and 0 for the 'start' signal in test bench, we generated a complete pulse on 'start' signal.

Question 6. Include expected and achieved results (Exp-out) in your report and make a comparison.

The achieved results are shown in figure 11.

Input	Expected Result		Achieved Result
	Decimal	Binary	Binary
0	1	00.1111111111111111	00.1111111111111111
1	2.71828	10.1011011111100001	10.1011011111010101
$\frac{1}{2}$	1.64872	01.1010011000010010	01.1010011000001011
$\frac{1}{3}$	1.39561	01.0110010101000110	01.0110010101000000
$\frac{1}{4}$	1.28402	01.0100100010110101	01.0100100010110000

Fig. 11. The expected and acheived results of the wrapper (Pre-synthesis)

III. Implementing Accelerator on FPGA

Question 1. Synthesize the design and include the synthesis report in your documents.

The final block diagram and the synthesis report of the circuit is shown in figures 12 and 13.

Question 2. Connect the wrapper "done" signal to LED[9].

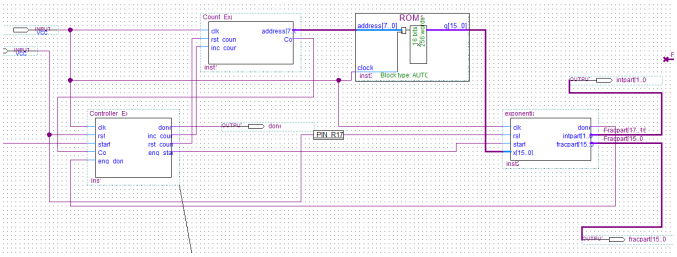

Fig. 12. The block diagram of the wrapper

After each round of estimating 5-values this LED will turn on. Connect the wrapper "start" pin to SW[9] on the board. You need to issue the start" at the beginning of each round

We do the assigning of the pins and program the FPGA board.

Question 3. To show the 18-bit result values use 7-segments. Note that you need a converter for 7-segment display.Alternatively you can use LEDs for displaying the results. Use one LED for the signal done, 2 LEDs for integer part and the rest of LEDs for the most significant bits of the fractional part.

To show the result values containing integer part and fractional part, we have to use seven segments. For this purpose, we need to use a converter so we use the Verilog code of Seven Segment

Flow Summary	
Flow Status	Successful - Thu Jan 12 21:36:56 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Lab4
Top-level Entity Name	Lab4
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	153 / 18,752 (< 1 %)
Total combinational functions	153 / 18,752 (< 1 %)
Dedicated logic registers	76 / 18,752 (< 1 %)
Total registers	76
Total pins	50 / 315 (16 %)
Total virtual pins	0
Total memory bits	4,096 / 239,616 (2 %)
Embedded Multiplier 9-bit elements	2 / 52 (4 %)
Total PLLs	0 / 4 (0 %)

Fig. 13. The compilation summary of the wrapper (Post-synthesis)

Decoder (SSD) which we designed in Lab2. We used an LED for the signal 'done' and after each round of estimating 5-values this LED will turn on. We show both integer and fractional part of the result by seven segments. The left seven segment is for 2 bits of integer part and 2 bits of fractional part and the other three seven segments are for the rest bits of fractional part. For the 5 values we saved in addresses in mif file, we test the design and the results are shown in figure 13. As we can see after doing operations in all 5 values, the LED assigned to the 'done' signal turns on and then again in the next loop it turns off.

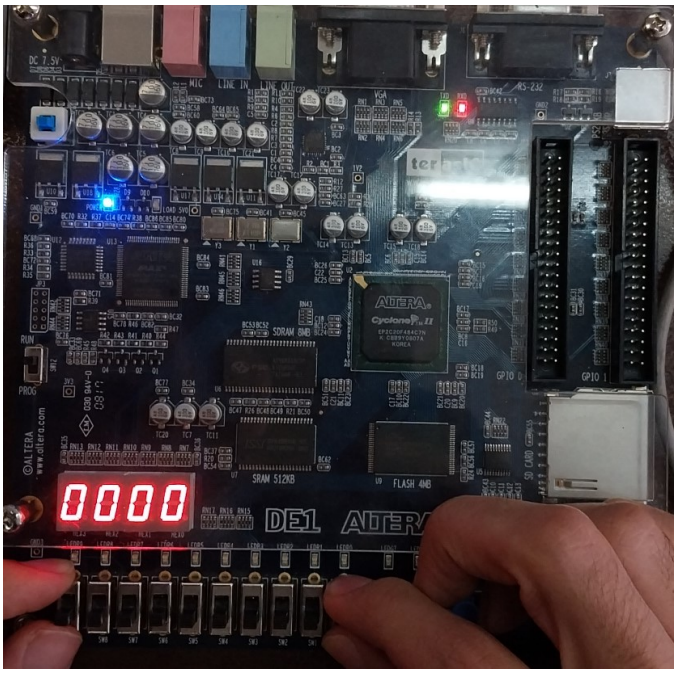

Fig. 14. The result after pressing the 'rst'

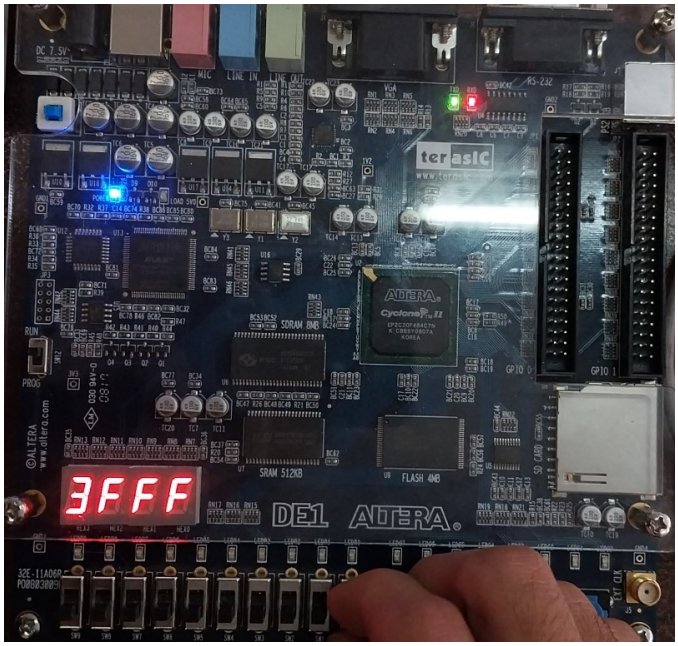

Fig. 15. The result of the 'input = 16'h0000'

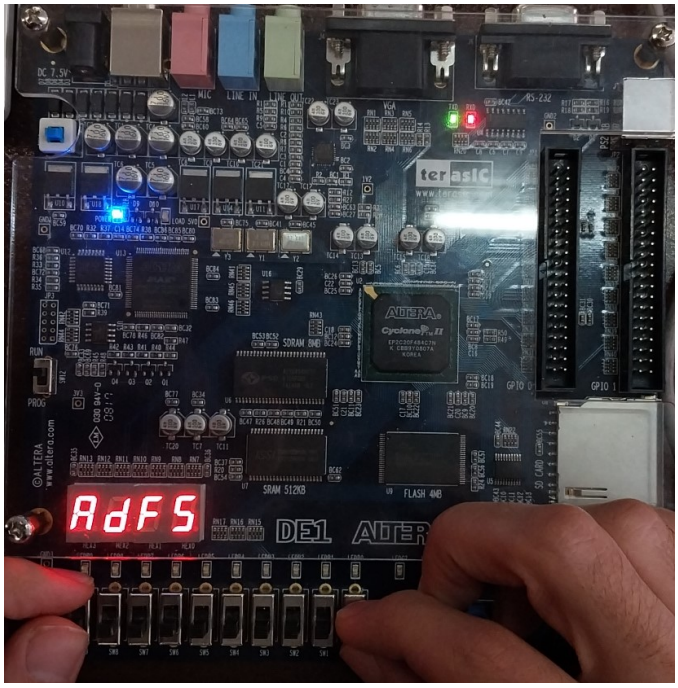

Fig. 16. The result of the 'input = 16'hFFFF'

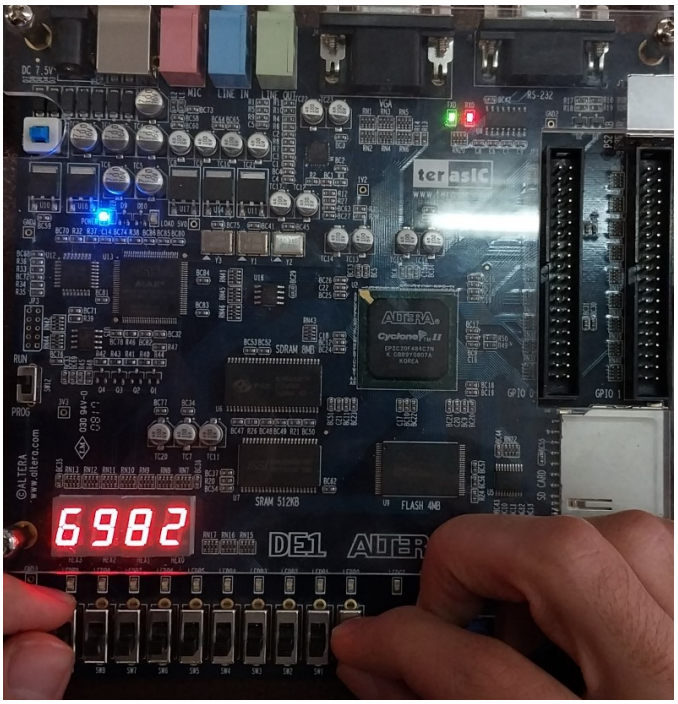

Fig. 17. The result of the 'input = 16'h8000'

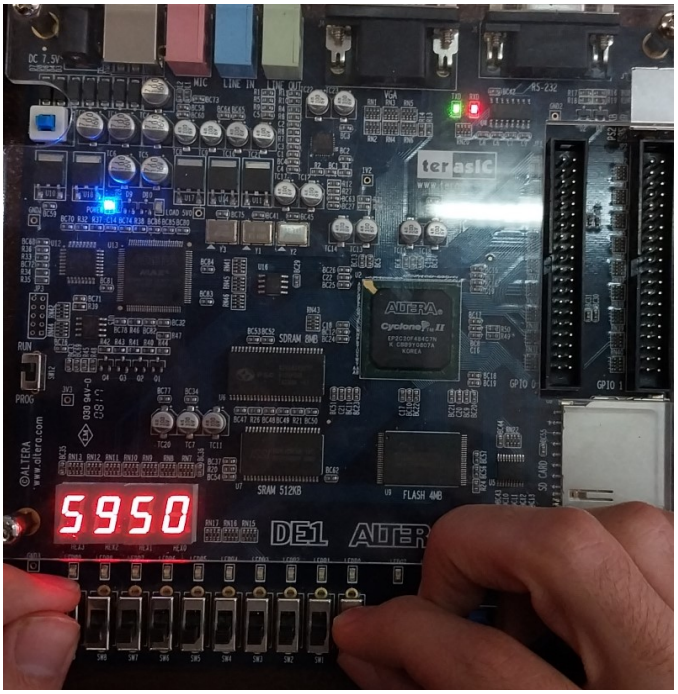

Fig. 18. The result of the 'input = 16'h5555'

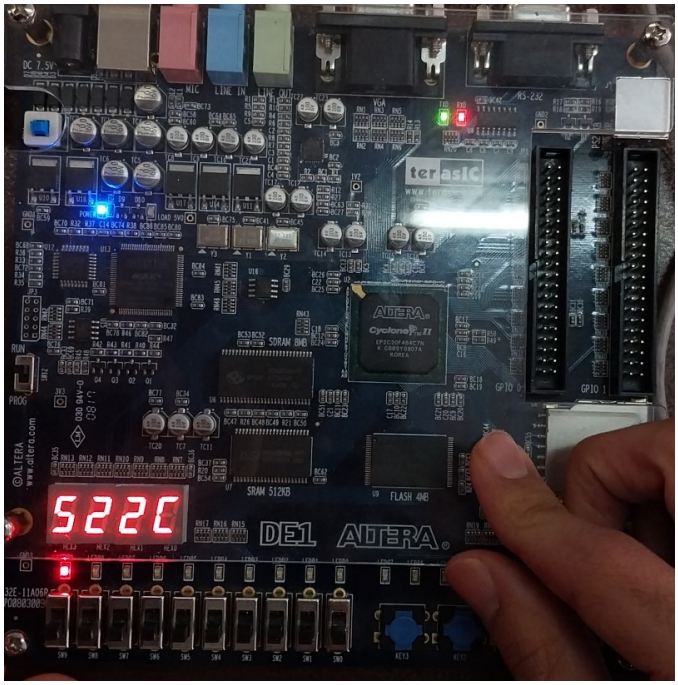

Fig. 19. The result of the 'input = 16'h4000'