Experiment 4 - Accelerator and Wrappers

Bita Nasiri Zarandi
810199504

Ali Padyav
810199388

***Abstract*— In This report we explain about our activities for experiment 4 in DLD lab and it's related to Accelerator and wrappers.**

***Keywords*— SOC, accelerator, Exponential Engine, wrapper.**

1. Exponential Engine

1. First we have our testbench here as you can see and there is three different values for the design:

```
`define clk_dur 50
module ExpEngineTB ();

  reg rst = 1'b1, clk = 1'b1, start = 1'b0;
  reg [15:0] x;
  wire done;
  wire [1:0] int;
  wire [15:0] fraction;

  exponential test (.clk(clk), .rst(rst), .start(start), .x(x),
                    .done(done), .int(int),.fraction(fraction));

  initial begin
    #(`clk_dur) rst = 1'b0;

    #(2 * `clk_dur) start = 1'b1;
    #(2 * `clk_dur) start = 1'b0;
    x = 16'b0100110110110111;
    while (done == 1'b0) #(`clk_dur);

    #(2 * `clk_dur) start = 1'b1;
    #(2 * `clk_dur) start = 1'b0;
    x = 16'b0011101101000100;
    while (done == 1'b0) #(`clk_dur);

    #(2 * `clk_dur) start = 1'b1;
    #(2 * `clk_dur) start = 1'b0;
    x = 16'b0111110010100011;
    while (done == 1'b0) #(`clk_dur);

    #(2 * `clk_dur) $stop;
  end

  always begin
    #`clk_dur clk = ~clk;
  end
endmodule
```

Fig.1 Testbench of Exponential engine

Here is the result of testbench for exponential engine in ModelSim:

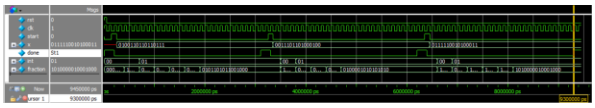


Fig. 2 Exponential engine waveform

2. The flow summery:

Flow Summary	
Flow Status	Successful - Sat Dec 10 01:58:02 2022
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Exp4
Top-level Entity Name	Exp4
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	97 / 18,752 (< 1 %)
Total combinational functions	96 / 18,752 (< 1 %)
Dedicated logic registers	58 / 18,752 (< 1 %)
Total registers	58
Total pins	23 / 315 (7 %)
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

3. Maximum frequency of this accelerator which is 109.39 MHz.

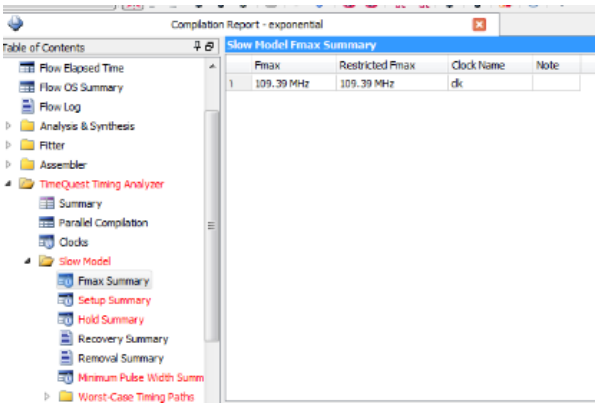


Fig. 4 Accelerator maximum frequency

Experiment 4 - Accelerator and Wrappers

Bita Nasiri Zarandi 810199504

Ali Padyav 810199388

2. Exponential Accelerator Wrapper

2.1 Accelerator Buffer

Here is .mif file that 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

2.2 The Wrapper Controller

1. controller:

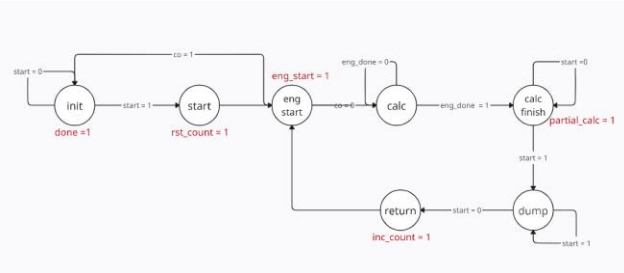


Fig. 6 Wrapper controller state diagram

3. Wrapper verilog description:

```
module ExpAccWrapperPartial (
    int_part,
    frac_part,
    partial_calc,
    done,
    rom_addr,
    start,
    eng_x,
    clk,
    rst
);

  output [1:0] int_part;
  output [15:0] frac_part;
  output partial_calc, done;
  output [7:0] rom_addr;

  input[15:0] eng_x;
  input start, clk, rst;

  wire co, inc_count, rst_count;
  wire eng_start, eng_done;

  exponential exp (
      .clk(clk),
      .rst(rst),
      .start(eng_start),
      .x(eng_x),
      .done(eng_done),
      .intpart(int_part),
      .fracpart(frac_part)
  );

  Counter counter (
      .count(rom_addr),
      .co(co),
      .inc(inc_count),
      .clk(clk),
      .rst(rst_count)
  );

  ExpAccController controller (
      .eng_start(eng_start),
      .rst_count(rst_count),
      .inc_count(inc_count),
      .partial_calc(partial_calc),
      .done(done),
      .start(start),
      .eng_done(eng_done),
      .co(co),
      .clk(clk),
      .rst(rst)
  );

endmodule
```

Fig. 7 Wrapper partial code

Experiment 4 - Accelerator and Wrappers

Bita Nasiri Zarandi 810199504

Ali Padyav 810199388

```
module ExpAcc (
    done_partial,
     int_out,
    frac_out,
    done,
    clk,
    rst,
     start
);

    output [1:0] int_out;
    output [15:0] frac_out;
    output done_partial, done;

    input start, clk, rst;

    wire [7:0] rom_address;
    wire [15:0] eng_x;

    ExpAccWrapperPartial wrapper_partial (
        .int_part(int_out),
        .frac_part(frac_out),
        .done(done),
        .partial_calc(done_partial),
        .rom_addr   (rom_address),
        .start (start),
        .eng_x(eng_x),
        .clk (clk),
        .rst (rst)
    );

    ROM rom (
        .address (rom_address),
        .clock(clk),
        .q(eng_x)
    );
endmodule
```

Fig. 8 Wrapper code

4. Wrapper testbench:

```
`timescale 1ns / 1ns
`define clk_dur 50

module ExpAccTB();

        reg rst = 1'b1, clk = 1'b1, start = 1'b0;

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

        ExpAcc to_be_tested (
          .clk(clk),
          .rst(rst),
          .start(start),
          .done(done),
          .partial_calc(partial_done),
          .int_part(intpart),
          .frac_part(fracpart)
        );

         initial begin
        #(`clk_dur) rst = 1'b0;

        #(2 * `clk_dur) start = 1'b1;
        #(2 * `clk_dur) start = 1'b0;
        while (partial_done == 1'b0) #(`clk_dur);

        #(2 * `clk_dur) start = 1'b1;
        #(2 * `clk_dur) start = 1'b0;
        while (partial_done == 1'b0) #(`clk_dur);
```

```
        #(2 * `clk_dur) start = 1'b1;
        #(2 * `clk_dur) start = 1'b0;
        while (partial_done == 1'b0) #(`clk_dur);

        #(2 * `clk_dur) start = 1'b1;
        #(2 * `clk_dur) start = 1'b0;
        while (partial_done == 1'b0) #(`clk_dur);

        #(2 * `clk_dur) start = 1'b1;
        #(2 * `clk_dur) start = 1'b0;
        while (partial_done == 1'b0) #(`clk_dur);

        #(2 * `clk_dur) $stop;
         end

         always begin
        #`clk_dur clk = ~clk;
         end

endmodule
```

Fig. 9 Wrapper testbech

Experiment 4 - Accelerator and Wrappers

Bita Nasiri Zarandi 810199504

Ali Padyav 810199388

5.

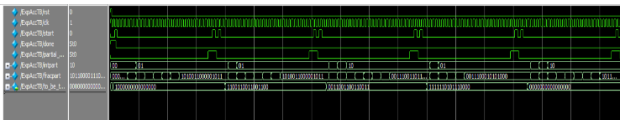


Fig. 10 Wrapper waveform

3. Implementing Accelerator on FPGA

2. First state result in FPGA

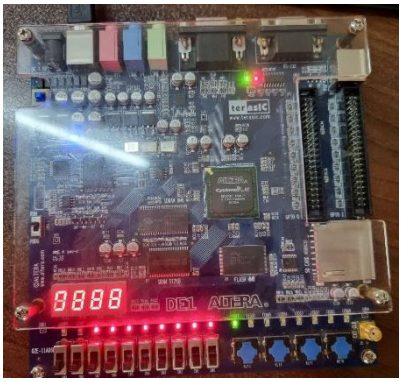
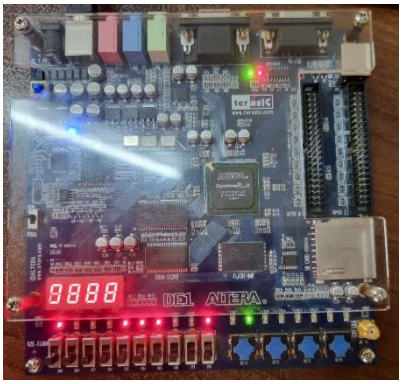
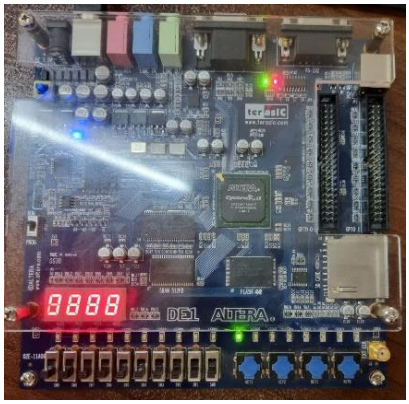
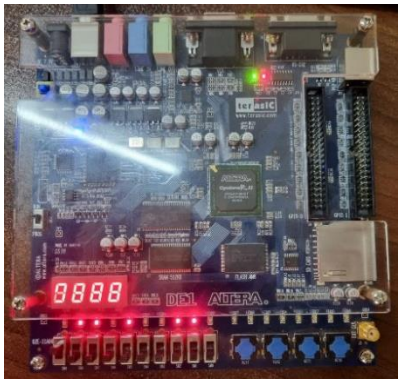


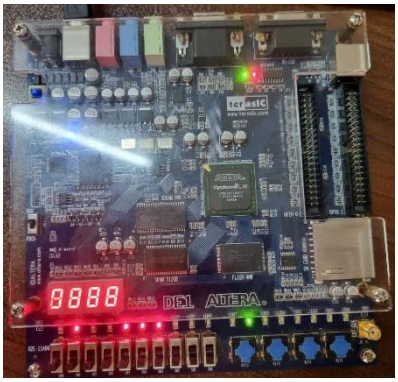
Fig. 11 first state

3. All FPGA results









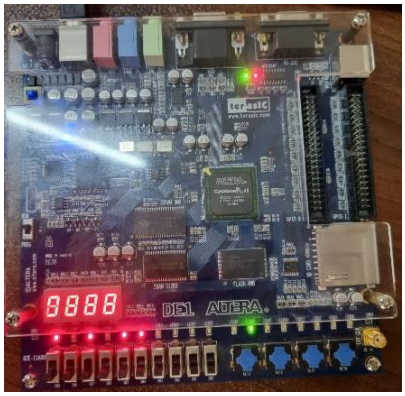


Fig. 12 FPGA result