

Experiment 4 - Accelerator and Wrappers

Bitra 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:

```
2 `define clk_dur 50
3 module ExpEngineTB ();
4
5 reg rst = 1'b1, clk = 1'b1, start = 1'b0;
6 reg [15:0] x;
7 wire done;
8 wire [1:0] int;
9 wire [15:0] fraction;
10
11 exponential test (.clk(clk), .rst(rst), .start(start), .x(x),
12                .done(done), .int(int), .fraction(fraction));
13
14 initial begin
15     #(`clk_dur) rst = 1'b0;
16
17     #(2 * `clk_dur) start = 1'b1;
18     #(2 * `clk_dur) start = 1'b0;
19     x = 16'b0100110110110111;
20     while (done == 1'b0) #(`clk_dur);
21
22     #(2 * `clk_dur) start = 1'b1;
23     #(2 * `clk_dur) start = 1'b0;
24     x = 16'b00111101101000100;
25     while (done == 1'b0) #(`clk_dur);
26
27     #(2 * `clk_dur) start = 1'b1;
28     #(2 * `clk_dur) start = 1'b0;
29     x = 16'b0111110010100011;
30     while (done == 1'b0) #(`clk_dur);
31
32     #(2 * `clk_dur) $stop;
33 end
34
35 always begin
36     #`clk_dur clk = ~clk;
37 end
38 endmodule
```

Fig.1 Testbench of Exponential engine

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

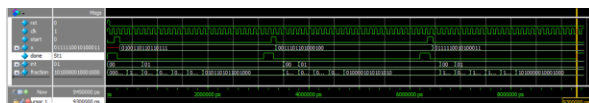


Fig. 2 Exponential engine waveform

2. The flow summary:

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.

Compilation Report - exponential			
Slow Model Fmax Summary			
Table of Contents	Fmax	Restricted Fmax	Clock Name
Flow Elapsed Time	109.39 MHz	109.39 MHz	clk
Flow OS Summary			
Flow Log			
Analysis & Synthesis			
Fitter			
Assembler			
TimeQuest Timing Analyzer			
Summary			
Parallel Compilation			
Clocks			
Slow Model			
Fmax Summary			
Setup Summary			
Hold Summary			
Recovery Summary			
Removal Summary			
Minimum Pulse Width Summ			
Worst-Case Timing Paths			

Fig. 4 Accelerator maximum frequency

Experiment 4 - Accelerator and Wrappers

Bitra 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:

```
1  WIDTH=16;  
2  DEPTH=8;  
3  
4  ADDRESS_RADIX=HEX;  
5  DATA_RADIX=HEX;  
6  
7  CONTENT BEGIN  
8  
9  00: 8000; -- e^0.50 = 1.648 = 0x1A5E3  
10 01: CCCC; -- e^0.79 = 2.203 = 0x233F7  
11 02: 3333; -- e^0.19 = 1.209 = 0x13581  
12 03: FD70; -- e^0.99 = 2.691 = 0x2B0E5  
13 04: 0000; -- e^0.00 = 1.000 = 0x10000  
14 05: 0000;  
15 06: 0000;  
16 07: 0000;  
17  
18 END;
```

Fig. 5 ROM .mif file

2.2 The Wrapper Controller

1. controller:

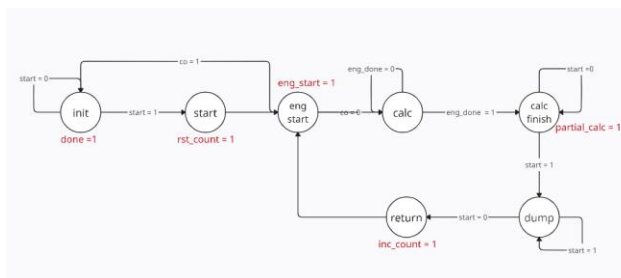


Fig. 6 Wrapper controller state diagram

3. Wrapper verilog description:

```
1  module ExpAccWrapperPartial (  
2      int_part,  
3      frac_part,  
4      partial_calc,  
5      done,  
6      rom_addr,  
7      start,  
8      eng_x,  
9      clk,  
10     rst  
11 );  
12  
13 output [1:0] int_part;  
14 output [15:0] frac_part;  
15 output partial_calc, done;  
16 output [7:0] rom_addr;  
17  
18 input[15:0] eng_x;  
19 input start, clk, rst;  
20  
21 wire co, inc_count, rst_count;  
22 wire eng_start, eng_done;  
23  
24 exponential exp (  
25     .clk(clk),  
26     .rst(rst),  
27     .start(eng_start),  
28     .x(eng_x),  
29     .done(eng_done),  
30     .intpart(int_part),  
31     .fracpart(frac_part)  
32 );  
33  
34  
35 Counter counter (  
36     .count(rom_addr),  
37     .co(co),  
38     .inc(inc_count),  
39     .clk(clk),  
40     .rst(rst_count)  
41 );  
42  
43 ExpAccController controller (  
44     .eng_start(eng_start),  
45     .rst_count(rst_count),  
46     .inc_count(inc_count),  
47     .partial_calc(partial_calc),  
48     .done(done),  
49     .start(start),  
50     .eng_done(eng_done),  
51     .co(co),  
52     .clk(clk),  
53     .rst(rst)  
54 );  
55  
56 endmodule
```

Fig. 7 Wrapper partial code

Experiment 4 - Accelerator and Wrappers

Bitra Nasiri Zarandi
810199504

Ali Padyav
810199388

```
1  module ExpAcc (  
2      done_partial,  
3      int_out,  
4      frac_out,  
5      done,  
6      clk,  
7      rst,  
8      start  
9  );  
10  
11      output [1:0] int_out;  
12      output [15:0] frac_out;  
13      output done_partial, done;  
14  
15      input start, clk, rst;  
16  
17      wire [7:0] rom_address;  
18      wire [15:0] eng_x;  
19  
20      ExpAccWrapperPartial wrapper_partial (  
21          .int_part(int_out),  
22          .frac_part(frac_out),  
23          .done(done),  
24          .partial_calc(done_partial),  
25          .rom_addr (rom_address),  
26          .start (start),  
27          .eng_x(eng_x),  
28          .clk (clk),  
29          .rst (rst)  
30      );  
31  
32      ROM rom (  
33          .address (rom_address),  
34          .clock(clk),  
35          .q(eng_x)  
36      );  
37  endmodule
```

Fig. 8 Wrapper code

4. Wrapper testbench:

```
1  `timescale 1ns / 1ns  
2  `define clk_dur 50  
3  
4  module ExpAccTB();  
5  
6      reg rst = 1'b1, clk = 1'b1, start = 1'b0;  
7  
8      wire done;  
9      wire partial_done;  
10     wire [1:0] intpart;  
11     wire [15:0] fracpart;  
12  
13     ExpAcc to_be_tested (  
14         .clk(clk),  
15         .rst(rst),  
16         .start(start),  
17         .done(done),  
18         .partial_calc(partial_done),  
19         .int_part(intpart),  
20         .frac_part(fracpart)  
21     );  
22  
23     initial begin  
24         #(`clk_dur) rst = 1'b0;  
25  
26         #(2 * `clk_dur) start = 1'b1;  
27         #(2 * `clk_dur) start = 1'b0;  
28         while (partial_done == 1'b0) #(`clk_dur);  
29  
30         #(2 * `clk_dur) start = 1'b1;  
31         #(2 * `clk_dur) start = 1'b0;  
32         while (partial_done == 1'b0) #(`clk_dur);  
33  
34         #(2 * `clk_dur) start = 1'b1;  
35         #(2 * `clk_dur) start = 1'b0;  
36         while (partial_done == 1'b0) #(`clk_dur);  
37  
38         #(2 * `clk_dur) start = 1'b1;  
39         #(2 * `clk_dur) start = 1'b0;  
40         while (partial_done == 1'b0) #(`clk_dur);  
41  
42         #(2 * `clk_dur) start = 1'b1;  
43         #(2 * `clk_dur) start = 1'b0;  
44         while (partial_done == 1'b0) #(`clk_dur);  
45  
46         #(2 * `clk_dur) $stop;  
47         end  
48  
49     always begin  
50         #`clk_dur clk = ~clk;  
51         end  
52  
53  endmodule
```

Fig. 9 Wrapper testbench

Experiment 4 - Accelerator and Wrappers

Bitra Nasiri Zarandi
810199504

Ali Padyav
810199388

5.

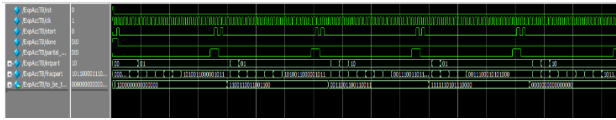


Fig. 10 Wrapper waveform

3. Implementing Accelerator on FPGA

2. First state result in FPGA

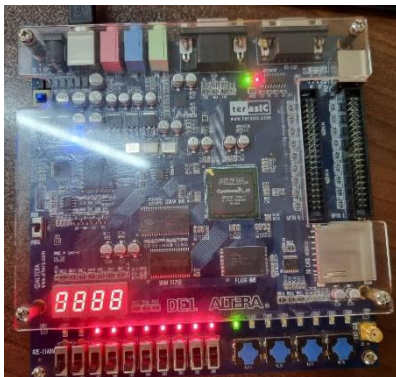


Fig. 11 first state

3. All FPGA results

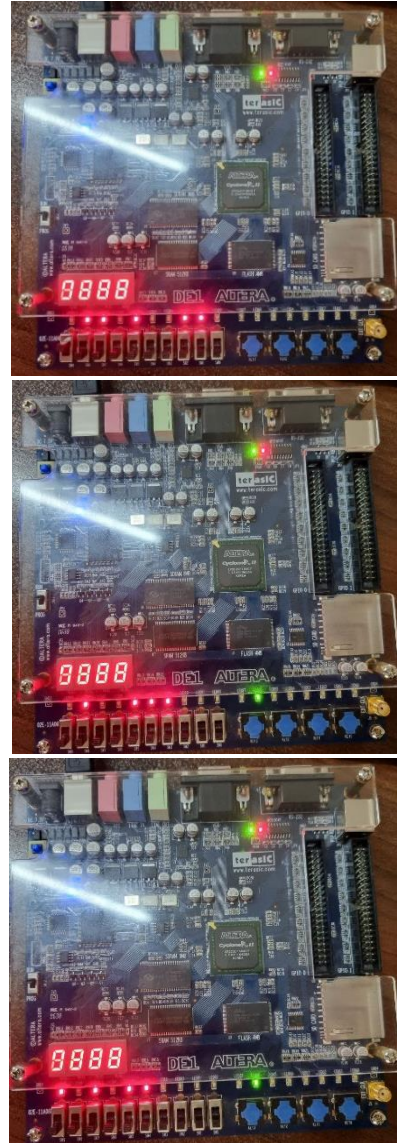
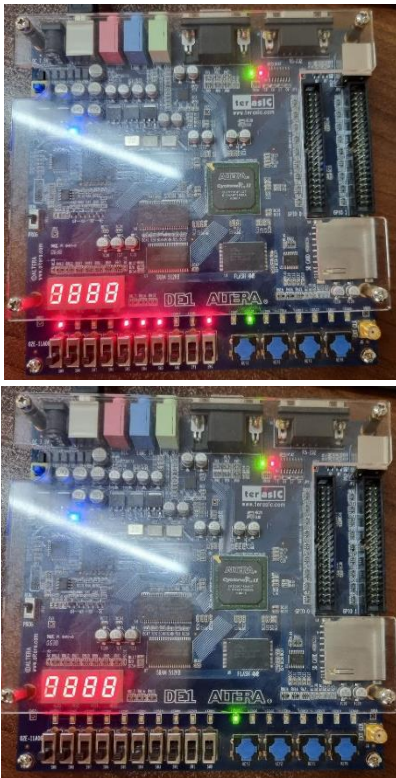


Fig. 12 FPGA result