Experiment 4 - Accelerator and Wrappers

Erfan Mokhtari
810100208

AmirHosein Samoudi
810100108

***Abstract*— In this experiment we have implemented an accelerator and designed a wrapper for it. Accelerators are dedicated computation units that usually execute one specific task. This single task needs a smaller and less complicated datapath which leads to a high frequency of operation for the accelerators. This imposes a low frequency of operation for CPUs.**

***Keywords*— Exponential accelerator – Wrapper – Hand shaking –SOC– FIFO - ModelSim simulation - FPGA**

I. Introduction

System on Chip (SOC)is an integrated circuit that integrates multiple components including digital, analog, hardware, and software programs in a single chip and includes processor, memory, Input/Output ports, and accelerators. Accelerator is a hardware designed for the specific tasks that is used beside the main processor to speed up the system. Processor dispute some of its tasks to the hardware accelerator. In this experiment we are given an accelerator engine that is designed to do exponential computations and we have also designed a wrapper for it to communicate with other parts of the chip (specially the processor) because accelerators are usually a lot faster than our main processor.

When the CPU needs to compute an exponential value, because of the higher estimation speed of the accelerator it asks the exponential hardware accelerator to complete this task (by issuing start signal). In this way, the CPU can complete other software tasks in parallel with the accelerator. Before starting the computation, the CPU should send a set of data from memory to the accelerator. This data will be stored in a buffer inside the accelerator. When transferring is finished, the CPU initiates the accelerator for an N-round exponential estimation. Accelerator issues ‘Done’ signal to inform the processor when the operation is finished.

II. Exponential Engine

The accelerator that you are going to use is an exponential circuit. This module receives a 16-bit input ”x” and generates a 16-bit output ”Fractional-part” and 2-bit ”Integer-part”. x value fits between zero and one. The accelerator starts working with a complete pulse on signal ”start” and when the computation is completed signal ”done” will be sent to the processor to acknowledge it. For clock generation for this module, we need to be aware of the maximum frequency of this accelerator.

First we examined the given code of the accelerator by writing a testbench for it.

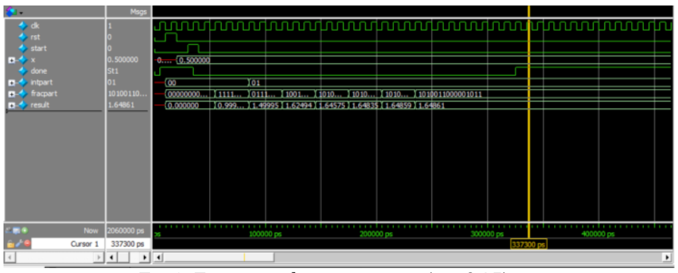


Fig 1. Exponential engine output (x = 0.25)

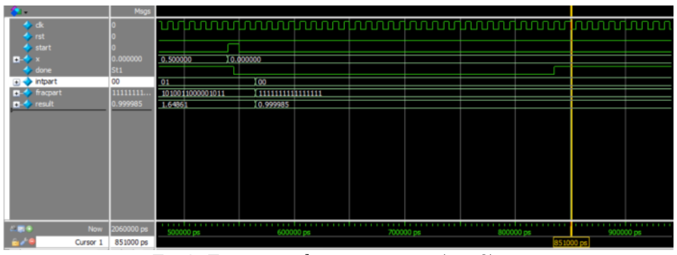


Fig 2. Exponential engine output (x = 0)

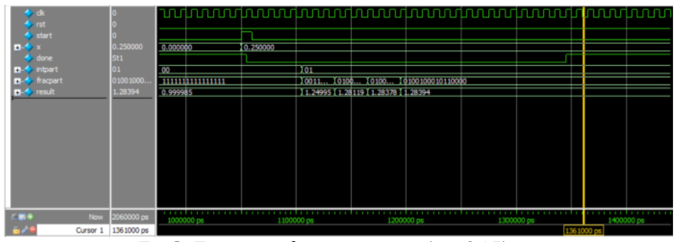
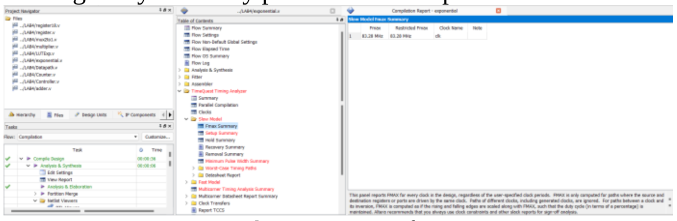


Fig 3. Exponential engine output (x = 0.25)

As we can see results are valid and accurate.

Then we have synthesized the design on Quartus to find the maximum frequency of the circuit that is measured by evaluating delays of every potential critical paths.


Project Navigator
Table of Contents
Compilation Report - exponential
Fmax Restricted Fmax Clock Name Note
1 83.28 MHz 83.28 MHz clk
Fig 4. Accelerator maximum frequency

As we can see in figure 4 the maximum frequency of the accelerator circuit is 83.28 MHz.

III. Exponential Accelerator Wrapper

Although the accelerator is working with a higher frequency than the processor, for the handshaking signals of ”start” and ”done” the accelerator has to wait for the processor to send and receive these signals with its low frequency. This imposes some timing overhead on the accelerator and hence performance reduction. To use this free time, the accelerator can calculate multiple exponential values. Each input value is split into an integer number zi and a fractional number vi as below:

$xi = zi + vi$

$e^{xi} = e^{zi} .e^{vi}$

The second segment of this equation can be easily implemented with the exponential engine. For the first part we use the base number 2 to take place of the base number e:

$e^{xi} = 2^{ui} .e^{vi}$

this equation can be implemented with a shifter that shifts e^{vi} for ui times.

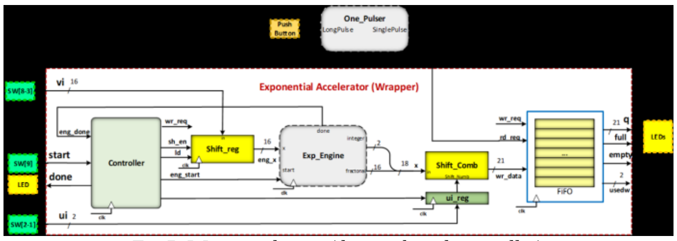


Fig 5. Wrapper design (data path and controller)

Wrapper receives single input in the form of a fractional value, vi, and an integer value ui and a start signal from the processor. 4 exponential values can be calculated. Four different values can be generated with the shift register unit by first registering the value of vi and then shifting its value one bit to the left for each exponential calculation.

The controller is responsible for generating the load and shift enable signals for the shift register, the start signal for the exponential engine, and the load signal for the ui-register. The exponential engine should start each calculation when the previous one is completely done. For this purpose engdone is fed to the controller and when done is asserted the controller generates a complete pulse on engstart. At the same time, the correct value of x should appear on the corresponding input of the exponential engine. For each exponential value estimation, the controller issues the wrreq signal for writing data to the FIFO. When all calculations are finished the controller sends a done signal on the wrapper output.

The FIFO which stands for First Input First Output is a storage element like a memory array that automatically keeps track of the order in which data enters into the module and reads the data out in the same order.

Figure 6 shows the state diagram of the controller.

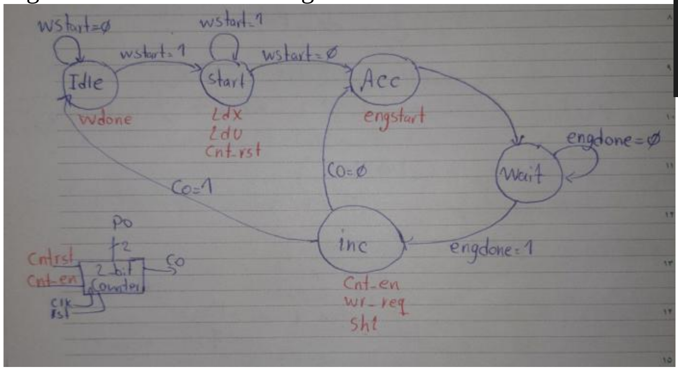
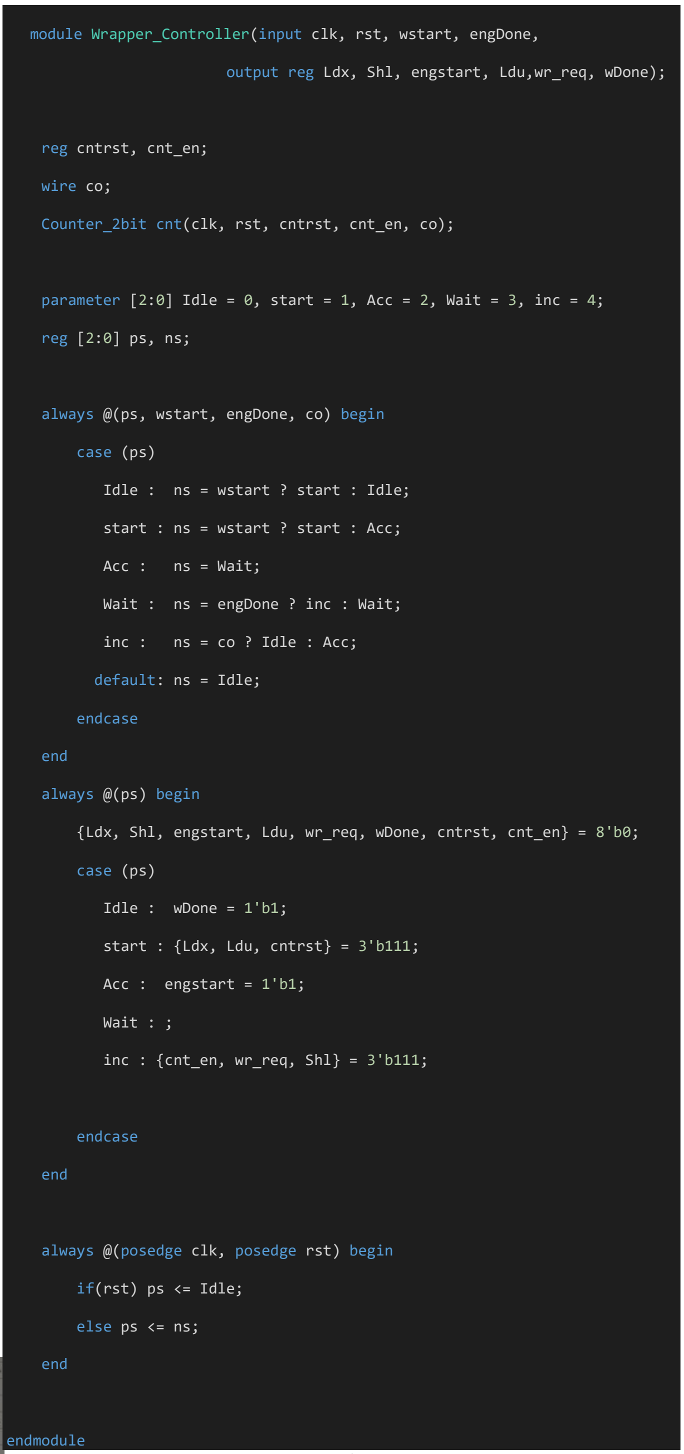
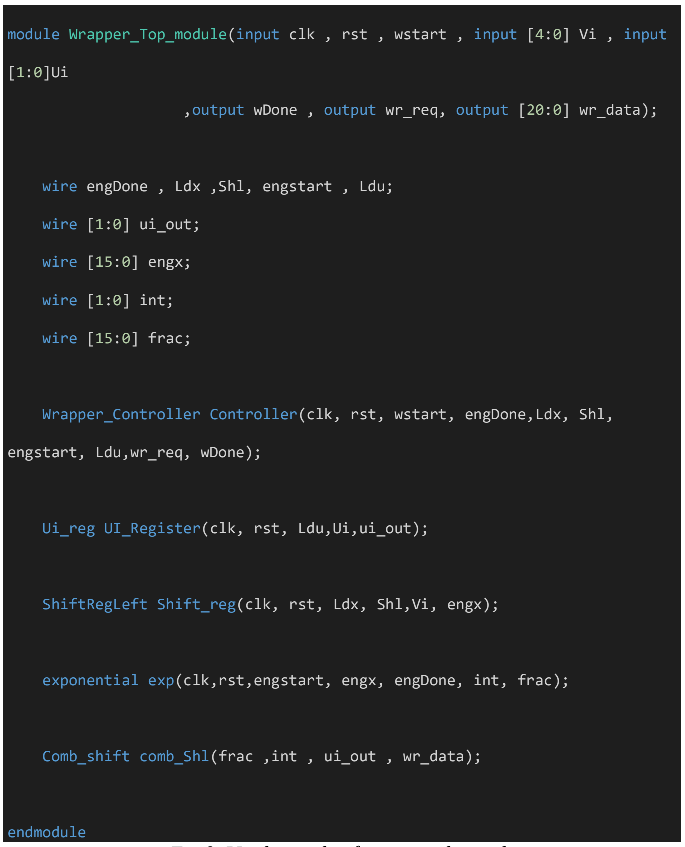


Fig 6. FSM of the wrapper


```
module Wrapper_Controller(input clk, rst, wstart, engDone,
                          output reg Ldx, Shl, engstart, Ldu,wr_req, wDone);

 reg cntrst, cnt_en;
 wire co;
 Counter_2bit cnt(clk, rst, cntrst, cnt_en, co);

 parameter [2:0] Idle = 0, start = 1, Acc = 2, Wait = 3, inc = 4;
 reg [2:0] ps, ns;

 always @(ps, wstart, engDone, co) begin
     case (ps)
        Idle :  ns = wstart ? start : Idle;
        start : ns = wstart ? start : Acc;
        Acc :   ns = Wait;
        Wait :  ns = engDone ? inc : Wait;
        inc :   ns = co ? Idle : Acc;
       default: ns = Idle;
     endcase
 end
 always @(ps) begin
     {Ldx, Shl, engstart, Ldu, wr_req, wDone, cntrst, cnt_en} = 8'b0;
     case (ps)
        Idle :  wDone = 1'b1;
        start : {Ldx, Ldu, cntrst} = 3'b111;
        Acc :  engstart = 1'b1;
        Wait : ;
        inc : {cnt_en, wr_req, Shl} = 3'b111;

     endcase
 end

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

endmodule
```
Fig 7. Verilog code of wrapper controller



```
module Wrapper_Top_module(input clk , rst , wstart , input [4:0] Vi , input
[1:0]Ui
                    ,output wDone , output wr_req, output [20:0] wr_data);

    wire engDone , Ldx ,Shl, engstart , Ldu;
    wire [1:0] ui_out;
    wire [15:0] engx;
    wire [1:0] int;
    wire [15:0] frac;

    Wrapper_Controller Controller(clk, rst, wstart, engDone,Ldx, Shl,
engstart, Ldu,wr_req, wDone);

    Ui_reg UI_Register(clk, rst, Ldu,Ui,ui_out);

    ShiftRegLeft Shift_reg(clk, rst, Ldx, Shl,Vi, engx);

    exponential exp(clk,rst,engstart, engx, engDone, int, frac);

    Comb_shift comb_Shl(frac ,int , ui_out , wr_data);

endmodule
```

Fig 8. Verilog code of wrapper datapath

Then we wrote a testbench to validate the design. The simulation results for 2 different values are in figure 9 and 10.

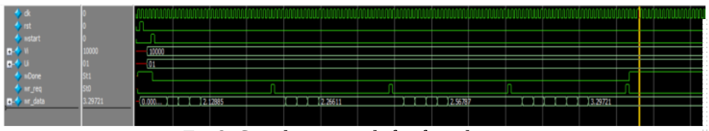


Fig 9. Simulation result for first data

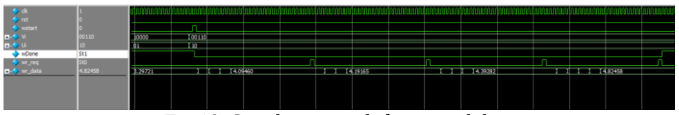
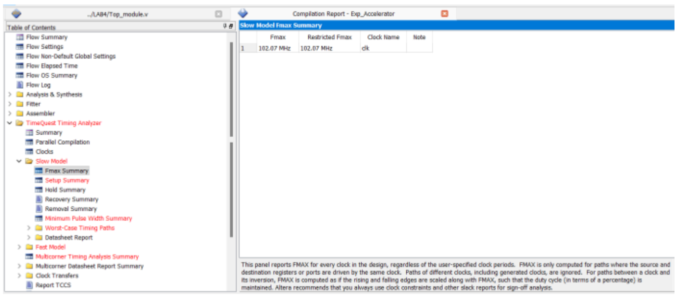


Fig 10. Simulation result for second data

In the figure 9 we have ui = 1 and vi = 0.0625. So the expected outputs are 2.1289, 2.2662, 2.5680 and 3.2974. The simulation shows the same values.

In the figure 10 we have ui = 2 and vi = 0.0234375. So the expected outputs are 4.0948, 4.1919, 4.3931 and 4.8249. The simulation shows the same values.

By synthesizing the wrapper design in Quartus we measured the maximum frequency of the wrapper which is 102.07 MHz.



Compilation Report - Exp_Accelerator

Slow Model Fmax Summary

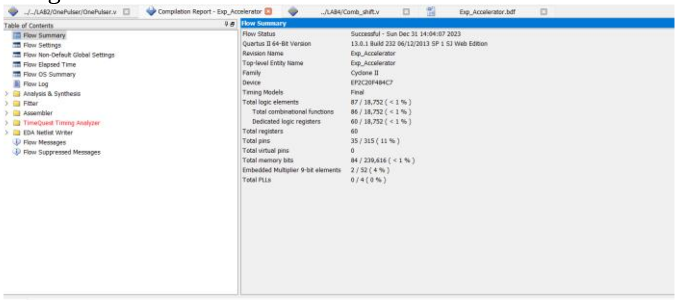
	Fmax	Restricted Fmax	Clock Name	Note
1	102.07 MHz	102.07 MHz	clk	

This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera recommends that you always use clock constraints and other slack reports for sign-off analysis.

Fig 11. Maximum frequency of the wrapper

IV. Implementing Accelerator on FPGA

We have synthesized the whole design. synthesis report is in figure 12.



Flow Summary

Flow Status	Successful - Sun Dec 31 14:04:07 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	Exp_Accelerator
Top-level Entity Name	Exp_Accelerator
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	87 / 18,752 (< 1 %)
Total combinational functions	86 / 18,752 (< 1 %)
Dedicated logic registers	60 / 18,752 (< 1 %)
Total registers	60
Total pins	35 / 315 (11 %)
Total virtual pins	0
Total memory bits	84 / 239,616 (< 1 %)
Embedded Multiplier 9-bit elements	2 / 52 (4 %)
Total PLLs	0 / 4 (0 %)

Fig 12. synthesis report

Since the readreq signal of the FIFO requires a complete pulse for reading a value from the FIFO, a one-pulser circuit is needed to issue this signal. For this purpose use the one-pulser circuit we designed in experiment 2.

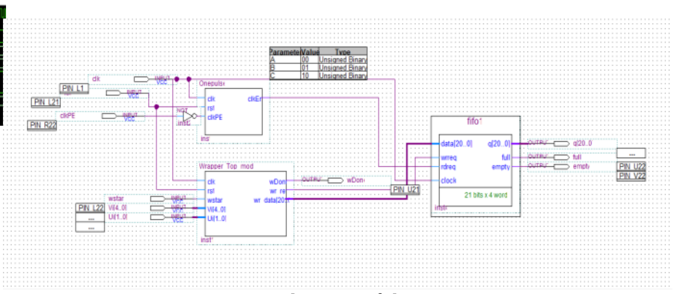


Fig 13. Schematic of the circuit

E