Experiment 4 – Accelerator and Wrappers

Matin Bazrafshan, 810100093

Mohammad Amin Yousefi, 810100236

***Abstract*— This document is a student report for the fourth experiment of the Digital Logic Laboratory course at University of Tehran, ECE department. The experiment is about Accelerators and how to integrate them in a System on Chip(SoC)**

***Keywords*— SoC, Accelerators, handshaking, Exponential, Wrapper, Quartus, FPGA, FIFO, Synthesis, Frequency, Verilog**

I. Introduction

In this experiment we created wrapper with an accelerator.

Accelerator is a fast component that can compute special mathematic operation and it can work parallel with CPU. Because its high speed it computes multi amount of operation and saves them in FIFO.

We created Verilog code for components of wrapper, the we synthesize wrapper at Quartus and move it to FPGA board.

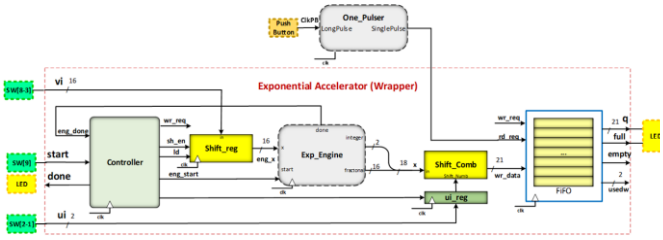


Fig.1 wrapper for exponential accelerator

II. Components and their Simulation on Verilog

A. *Exponential Engine*

For this component, we had Verilog code and we have just checked its accuracy .

An exponential engine is an accelerator which receives a 16-bit input “x” and calculates e^x (note that $0 < x < 1$) and generates an 16-bit output ”Fractional part” and 2- bit ”Integer part”. 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 first input:

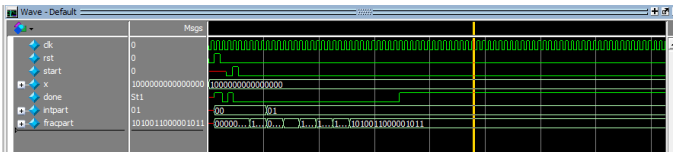


Fig. 2 first input

For second input:

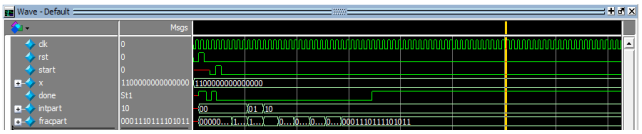


Fig. 3 second input

For third one:

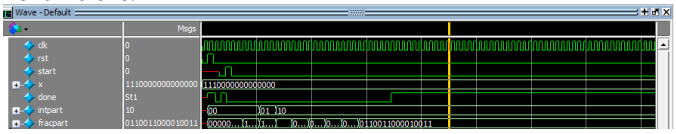


Fig.4 third input

B. *Data path of 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 have to wait for the processor to send and receive these signals with its low frequency. This imposes some timing overhead to the accelerator and hence performance reduction. In order to use this free time, the accelerator can calculate multiple exponential values.

$$x_i = z_i + v_i$$

$$e^{x_i} = e^{z_i}.e^{v_i}$$

But we can estimate integer part of result with 2 instead of e.

$$e^{x_i} = 2^{u_i}.e^{v_i}$$

$$e^{x_i} = e^{v_i} << u_i$$

And final equation is as below which can be implemented with a shifter that shifts e^{v_i} for u_i times. We are going to apply this transformation to the exponential in a wrapper around the exponential engine.

$$e^{x_i} = e^{v_i} \ll u_i$$

$$u_i = \lfloor z_i . \log_2 e \rfloor$$

And as a result considering the input values are within u_i and u_i + 1 we can calculate n number of exponentials in this range as follows:

$$e^{x_i} = \begin{cases} e^{v_i} << u_i & \\ e^{2*v_i} << u_i & \text{if } v_i < 1/2^{(n-1)} \\ \dots & \text{if } u_i < x_i < u_i + 1 \\ e^{2^{(n-1)}*v_i} << u_i & \end{cases}$$

C. controller

In fact, controller handles the computation and communicates with CPU and handshakes with accelerator. The controller is responsible for generating the load and shift enable signals for shift register, the start signal for exponential engine and the load signal for the ui-register. In fact, 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 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.

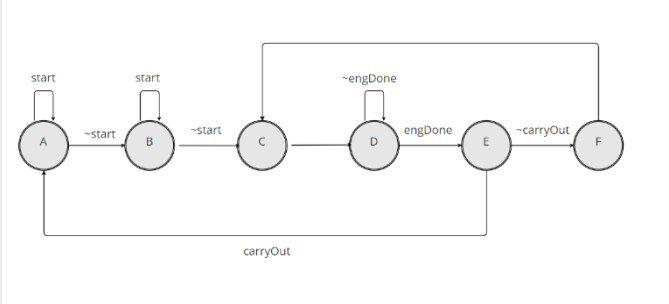


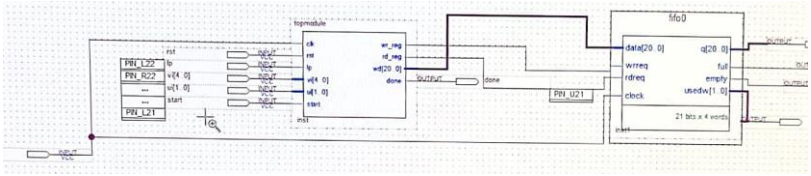
Fig. 5 controllers state diagram

D. FIFO

When an exponential value is calculated then it should be stored in a FIFO so when the CPU finishes it's work it can retrieve all the results. The FIFO which stands for First Input First Output is an 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.

III. Synthesis and Simulation

After designing every module, we get an instantiate in Quartus, and then we connect them properly.



Fig, 6 Block Diagram

And for simulation on:

u = 3;
v = 16'b0001111100000000;
ans = $e^v \times 2^u$

we have:
first: v = 0.12109375
ans = 9.029845
ans_bin = 01001.00000

second: 2v = 0.2421875
ans = 10.19226
ans_bin = 01010.00110

and third: 4v = 0.484375
ans = 12.98528
ans_bin = 01100.11111

and fourth: 8v = 0.96875
ans = 21.07719
ans_bin = 10101.00010

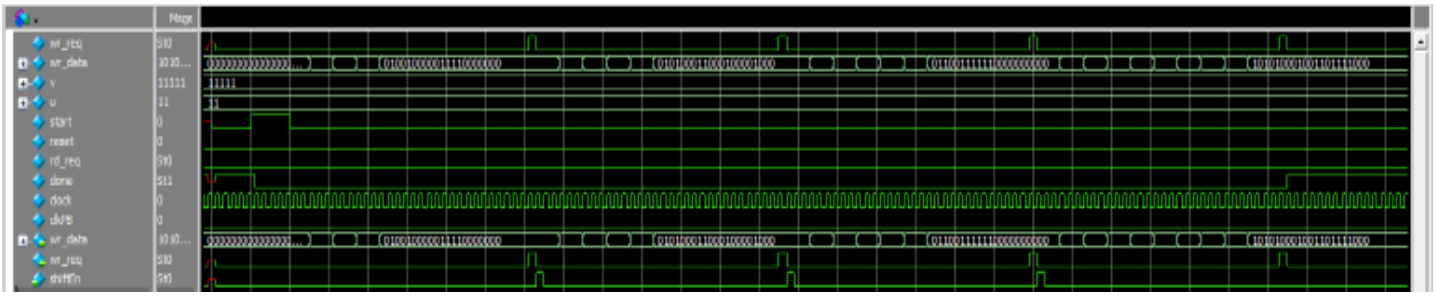


Fig. 6 Simulation result of exponential accelerator wrapper

IV. IMPLEMENTING ON FPGA

In this part, we are to synthesize the wrapper and implement it on the FPGA.

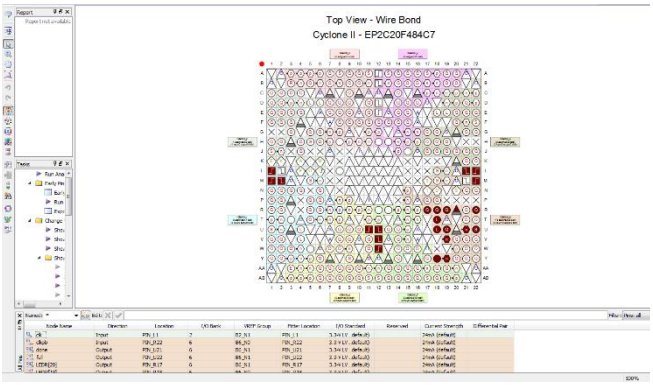


Fig. 7 pin planner

Slow Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	112.92 MHz	112.92 MHz	clk	

Fig. 8 wrapper frequency

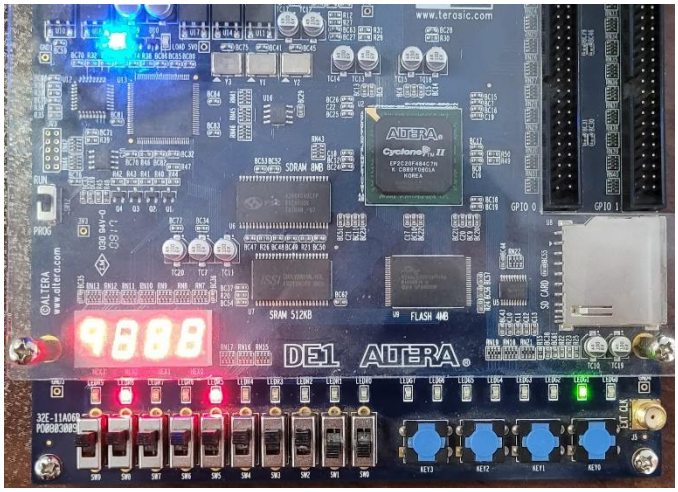


Fig. 9 The first result

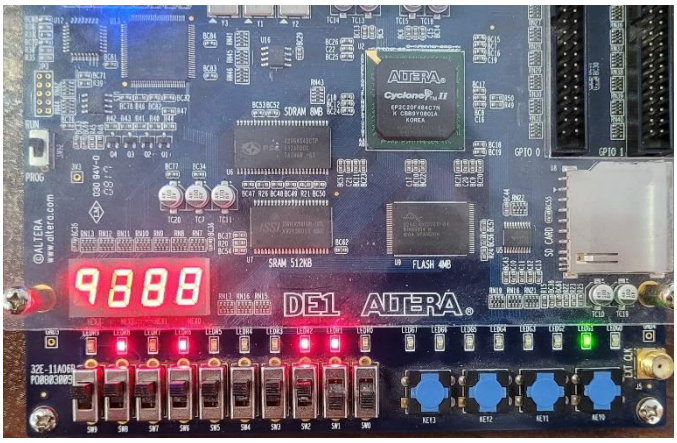


Fig. 10 The second result

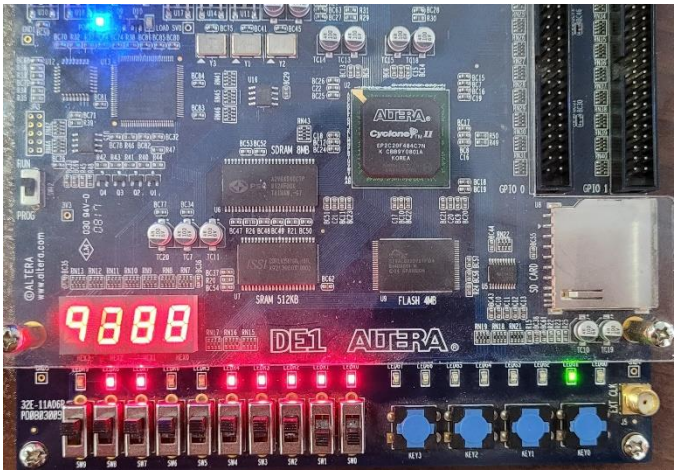


Fig. 11 The third result

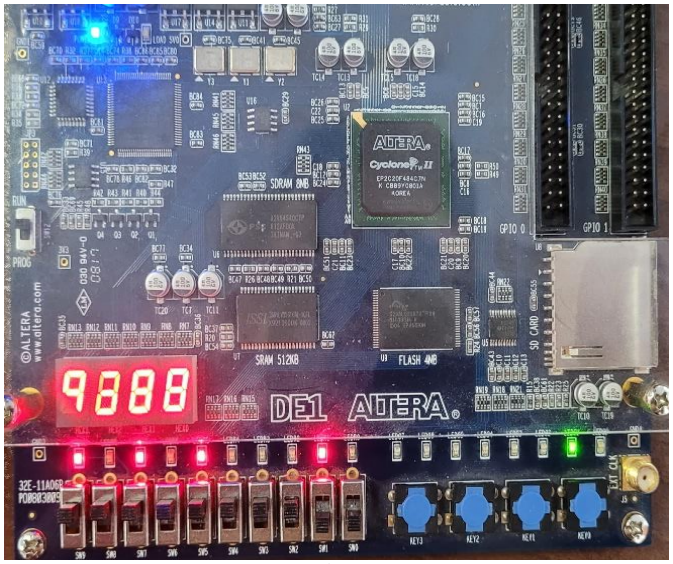


Fig. 12 The fourth result

V. Acknowledgment

This report was prepared by Mohammad Amin Yousefi and Matin Bazrafshan for fourth experiment of the DLD Laboratory at the spring of 1401-1402 semester.