

Experiment 4 - Accelerator and Wrappers

Shahriar Attar,
810100186,
attarshahriar@gmail.com

Elahe K. Nadrabadi,
810100132,
elahekhodavrdi@gmail.com

Abstract— This document is a report for experiment #4 of Digital Logic Design Laboratory at ECE department, University of Tehran. The purpose of this experiment is to use Accelerators to improve performance and use wrappers.

Keywords— Exponential Engine, SoC, Accelerator Buffer, Accelerator Wrapper, FPGA Implementation, Altera Cyclone II

I. INTRODUCTION

This experiment aims to explain more about hardware accelerators and how we can use them to get results faster. First we need to be familiar with SoC (i.e. System on Chip). Integrated circuit which includes multiple components and the main core is a processor that handles different computational tasks. Since CPUs need to execute millions of operations in a fix time interval, they are not always optimized for specific tasks, so we use accelerators that are designed to do one thing and because of that they can work much faster. So the processor dispute some of its tasks to hardware accelerator and store results in a memory and CPU will access those results when it finished its tasks. In Fig. 1 you can see the general design. Components that are in communication with CPU use *done* and *start* signals. In our experiment the CPU will issue the *start* signal when input is provided for the accelerator, and when the accelerator has done its duty it will issue the *done* signal so the CPU can access results that are in memory.

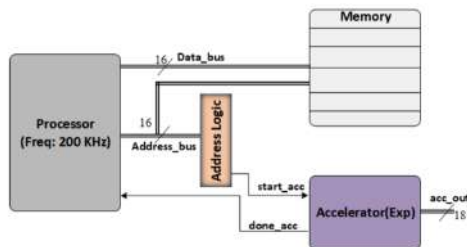


Fig. 1 Block diagram of a typical integrated circuit

II. EXPONENTIAL ENGINE

This module receives a 16-bit input x and generates an 16-bit output *Fractionalpart* and 2-bit *Integerpart*. x value lies between 0 and 1.

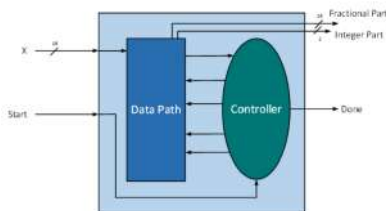


Fig. 2 Block diagram of an exponential engine

A. ModelSim simulation

The results for simulation for three different values for x are shown in Fig. 3.

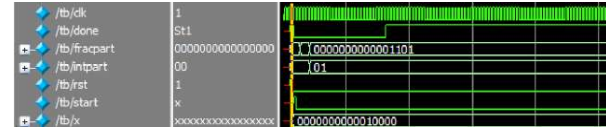


Fig. 3 Simulation results for ExpEng first input

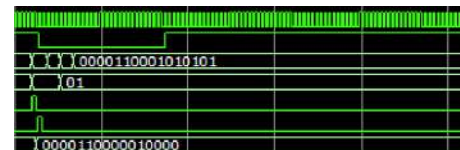


Fig. 4 Simulation results for ExpEng second input



Fig. 5 Simulation results for ExpEng third input

The results are very close to the expected output:

Input 0x0010: 0x1000D

Input 0x0C10: 0x0C55

Input 0xFFFF: 0xA60A


The difference is due to the exponential engine's accuracy.

B. Quartus synthesis and maximum frequency

The synthesis report is shown in Fig. 4 and also maximum frequency is shown in Fig. 5. For finding maximum frequency we used the Timing Analyzer report in Quartus synthesis tool.

Flow Summary	
<<Filter>>	
Flow Status	Successful - Mon Jun 12 16:35:41 2023
Quartus Prime Version	22.1std.0 Build 915 10/25/2022 SC Lite Edition
Revision Name	exponential
Top-level Entity Name	exponential
Family	Cyclone V
Device	5CGXFC7C6U19A7
Timing Models	Final
Logic utilization (In ALMs)	50 / 56,480 (< 1 %)
Total registers	70
Total pins	38 / 268 (14 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	1 / 156 (< 1 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

Fig. 6 Synthesis report

Slow 1100mV 125C Model Fmax Summary				
 <<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	108.62 MHz	108.62 MHz	clk	

III. EXPONENTIAL ACCELERATOR WRAPPER

The activation function is a mathematical function that is applied to the output of each neuron in the network. The activation function determines whether the neuron should be activated or not based on the weighted sum of its inputs.

The exponential function is a commonly used activation function in DNNs. However, calculating the exponential function can be computationally expensive, especially when dealing with large datasets.

To address this issue, the accelerator can calculate multiple exponential values during the free time it has while waiting for the processor to send and receive the handshaking signals. These pre-calculated exponential values can then be stored in memory and used by the DNN during the forward pass. This can significantly improve the performance of the DNN by reducing the computational overhead associated with calculating the exponential function.

$$f(x_i) = e^{x_i}, \{i = 1, 2, 3, \dots, N\}$$

Instead of using one accelerator for each we use one and in order to reduce hardware resources requisite for softmax exponential function, we use some mathematical transformation. z_i stands for integer part and v_i stands for fractional part.

$$\begin{aligned} x_i &= z_i + v_i \\ e^{x_i} &= e^{z_i} \cdot e^{v_i} \end{aligned}$$

And then for more simplicity we use base number 2 so it would change to:

$$e^{x_i} = 2^{u_i} \cdot e^{v_i} = e^{v_i} \ll u_i \ (u_i = \lfloor z_i \cdot \log_2 e \rfloor)$$

So we can solve the problem by just shifting. Another problem would be how many values are we going to calculate? Considering the input values are between u_i and

$u_i + 1$ and $v_i < 1/2^{(n-1)}$, the possible n numbers of exponential in this range are as follows:

$$e^{x_i} = \left\{ e^{v_i} \ll u_i, e^{2v_i} \ll u_i, \dots, e^{2^{(n-1)}v_i} \ll u_i \right\}$$

In this experiment $n = 4$, so we compute 4 different values between handshakes. We do so by first computing the exponential part and then shifting it.

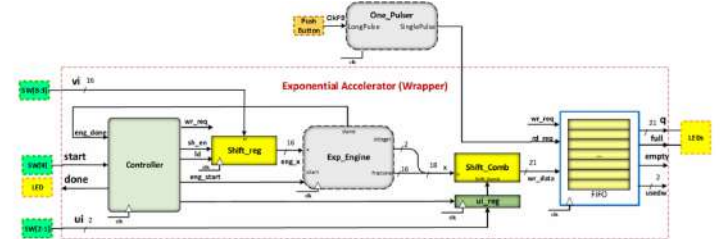


Fig. 8 Block diagram of Exponential accelerator wrapper

As can be seen in the above figure the controller is responsible for generating *load* and *shift enable* signals for shift register, the *start* signal, and *load signal* for `ui_reg`. After each calculation the engine starts the next one and for implementing this we use *engdone* signal. This is the state diagram of the controller.

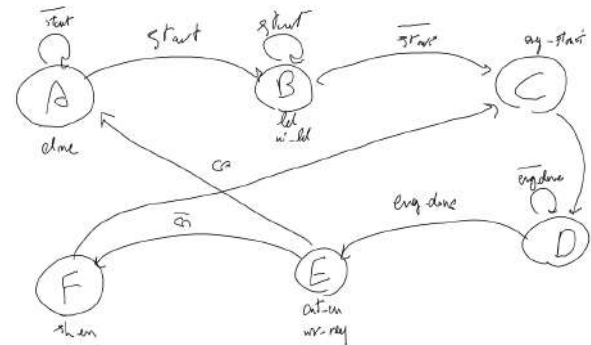


Fig. 9 State diagram of controller

After each computation we store the result in a FIFO using the *writereq* signal. And when all computations are done and written in our FIFO (which stands for First Input First Output), the *done* signal will be issued to acknowledge it.

For shifting after computing the exponential part we used a combinational shifter and the output of this module is stored in FIFO.

When *done* signal is received the CPU can retrieve data from FIFO, because of the structure of FIFO it keeps track of the order in which data enters into the module and reads from it in the same order. To use FIFO you will issue the *writereq* signal and it will store whatever data is on the buffer and by using *readreq* you can retrieve them in the same order. In this experiment we used the FIFO IP provided in Quartus

Megawizard, and since we want to store four different values we only need a FIFO of size 4.

A. ModelSim simulation

The testbench is shown in Fig. 19 and Fig. 11-14 show simulation results in ModelSim for wrapper.

```
initial begin
    rst = 1'b1; start = 1'b0; V = 5'b10000; U = 2'b01;
    #30 rst = 1'b0;
    #10 start = 1'b1;
    #20 start = 1'b0;
    #2000 $stop;
end
```

Fig. 10 accelerator wrapper test bench

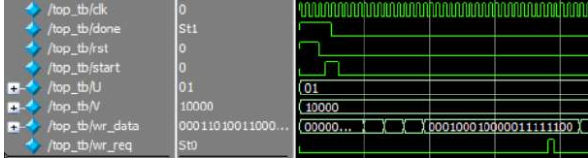


Fig. 11 Simulation of wrapper (first computed result)

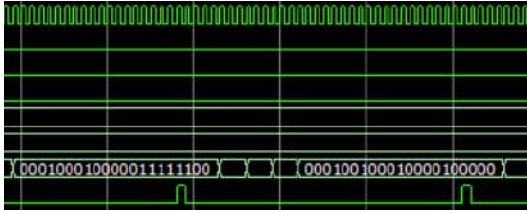


Fig. 12 Simulation of wrapper (second computed result)

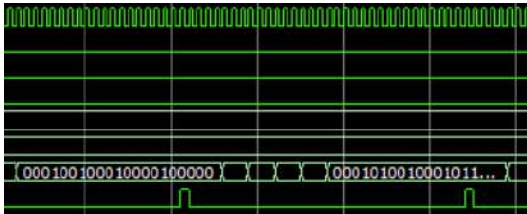


Fig. 13 Simulation of wrapper (third computed result)

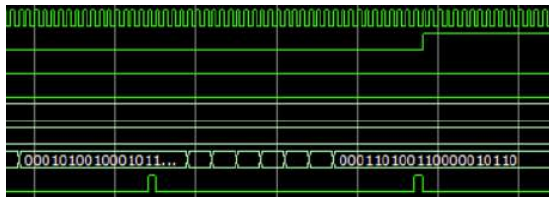


Fig. 14 Simulation of wrapper (fourth computed result)

B. Quartus

After that, we designed the exponential accelerator wrapper in Quartus, you can see the result in Fig. 15.

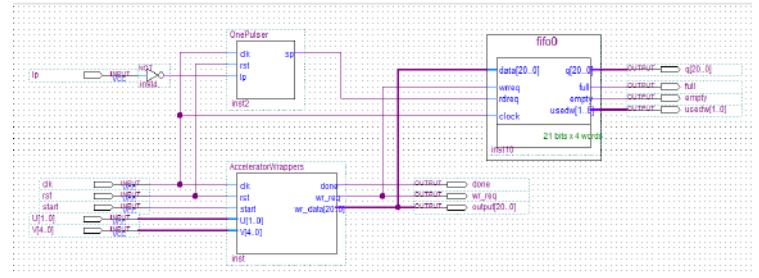


Fig. 15 The accelerator wrapper block diagram

The OnePulser module provides a *wrreq* input for the FIFO. This is used for controlling the clock when the circuit is implemented on an FPGA board. The one-pulser connects to a push-button on your board and when pressed it creates a single pulse that is synchronized with the system clock. The main idea behind this module is that since humans are not as fast to change input in every clock toggle (since they usually have a duration of about few nanoseconds) we need a module to determine when we want to enable the clock, and since we don't want to interfere with timing, it has to be synchronized with the system clock.

Flow Summary	
<<Filter>>	
Flow Status	Successful - Mon Jun 12 16:43:50 2023
Quartus Prime Version	22.1std.0 Build 915 10/25/2022 SC Lite Edition
Revision Name	acceleratorwrapper
Top-level Entity Name	acceleratorwrapper
Family	Cyclone V
Device	5CGXFC7C6U19A7
Timing Models	Final
Logic utilization (in ALMs)	77 / 56,480 (< 1 %)
Total registers	83
Total pins	33 / 268 (12 %)
Total virtual pins	0
Total block memory bits	0 / 7,024,640 (0 %)
Total DSP Blocks	1 / 156 (< 1 %)
Total HSSI RX PCSs	0 / 6 (0 %)
Total HSSI PMA RX Deserializers	0 / 6 (0 %)
Total HSSI TX PCSs	0 / 6 (0 %)
Total HSSI PMA TX Serializers	0 / 6 (0 %)
Total PLLs	0 / 13 (0 %)
Total DLLs	0 / 4 (0 %)

Fig. 16 Synthesis report for accelerator wrapper

Slow 1100mV 125C Model Fmax Summary				
<<Filter>>				
	Fmax	Restricted Fmax	Clock Name	Note
1	110.93 MHz	110.93 MHz	clk	

Fig. 17 maximum frequency for accelerator wrapper

IV. FPGA IMPLEMENTATION

A. Pin Planner

We connected the wrapper *done* signal to LED[9]. After each round of estimating 4-values this LED will turn on.

Connect the wrapper *start* pin to SW[9] on the board. To feed the 16-bit fractional values of v_i , we used six switches SW[8] to SW[3]. Since the fractional value should be less than $1/2^{(n-1)}$, the three most significant bits of the fractional part have to be always zero, assigned six switches to the rest six most significant bits of v_i . For u_i used two switches SW[2] and SW[1]. SW[0] will be used for the reset signal. We use the 50 MHz clock frequency of FPGA as the clock of wrapper.

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, so the output of one-pulser module is connected to *readreq* of FIFO.

For showing the 21-bit result values LEDs on the board are used. One LED for the signal *done*, 5 LEDs for *integer part* and the rest of LEDs for the most significant bits of the *fractional part*.

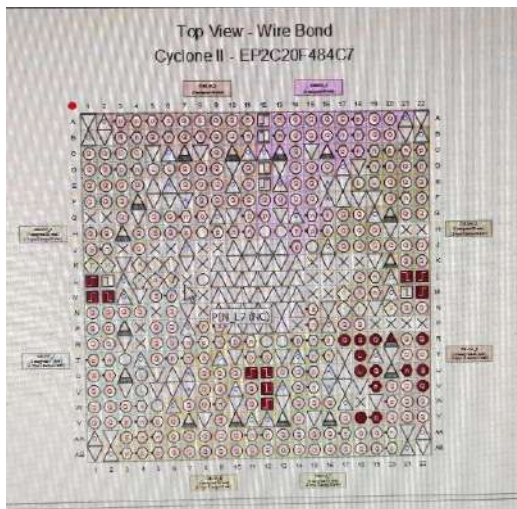


Fig. 18 Pin planner design

B. FPGA Implementation



Fig. 19 Accelerator has computed all values and are store in



Fig. 20 First number read from FIFO



Fig. 21 Second number read from FIFO



Fig. 22 Third number read from FIFO



Fig. 23 Fourth number read from FIFO

V. CONCLUSIONS

The general ideas discussed in this experiment were:

- SoC
- Accelerators
- Exponential Engine
- Module Maximum Frequency
- FPGA

ACKNOWLEDGMENT

This report was prepared by Elahe K. Nadrabadi and Shahriar Attar for 8, 9 sessions of DLD Laboratory at ECE department in 1401-1402 spring semester.

REFERENCES

- [1] SoC Labs Accelerator Wrapper [Online]. Available: <https://soclabs.org/technology/soc-labs-accelerator-wrapper/>