University of Tehran
College of Engineering
School of Electrical & Computer Engineering

Experiment 4
Sessions 9,10,11

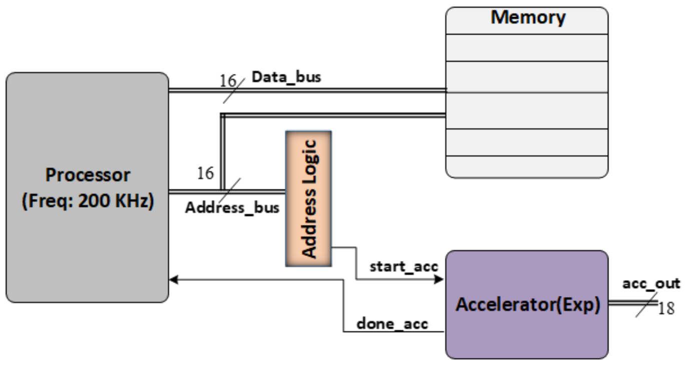
Accelerator and Wrappers

Digital Logic Laboratory
ECE 045
Laboratory Manual

Fall 1402

Contents

Figure 1: Block diagram of a typical integrated circuit



Introduction

System on Chip is an integrated circuit that integrates multiple components including digital, analog, hardware, and software programs in a single chip. The main core of an SoC is a processor that handles different computational tasks within the system. In addition to the processor, the system includes memory, Input/Output ports, and accelerators.

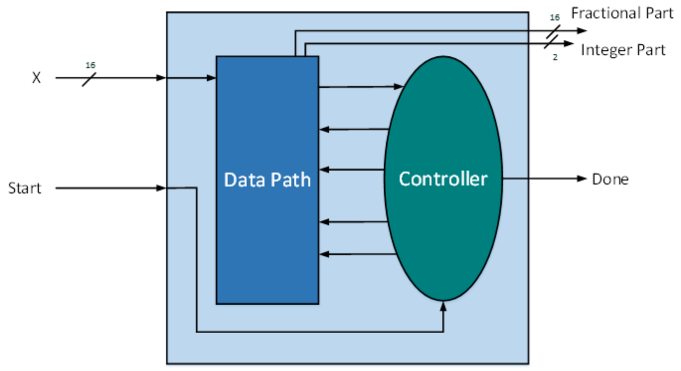
As you have learned from the Digital Logic Design course, accelerators are dedicated computation units that usually execute one specific task. This single task needs a smaller and less complicated datapath which leads to a high frequency of operation for the accelerators. This is contrary to CPUs in which millions of operations must be executed within a fixed time interval. This imposes a low frequency of operation for CPUs. To increase the speed of a SOC, hardware accelerators are usually embedded in the system. The processor will dispute some of its tasks to the hardware accelerator. During this time the accelerator performs several of the same or different operations and stores the result values in a memory. The CPU will access these results when it finishes its tasks. The focus of this experiment is on accelerators and how to integrate them into a SOC.

By the end of this experiment, you should have learned:

- The concept of a SOC
- The concept of handshaking in SoC
- The principle of an accelerator

Figure 1 shows the block diagram of a typical embedded system including a processor, an accelerator, and a memory. Any components that are in communication with the CPU talk to the CPU via signals "*start*" and "*done*". The embedded system shown in this figure works as follows: 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. 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. CPU uses its address bus for initiating

Figure 2: Block diagram of exponential accelerator



a component. By decoding the address bus through an address logic as shown in figure 1, the accelerator will have its ”*start*” signal issued when needed. For simplicity, in this experiment, you will implement the whole CPU and address logic inside the testbench and when implementing on an FPGA, you will feed ”*start*” through board switches.

Accordingly, Below are the topics that are explained in the following of this experiment in detail:

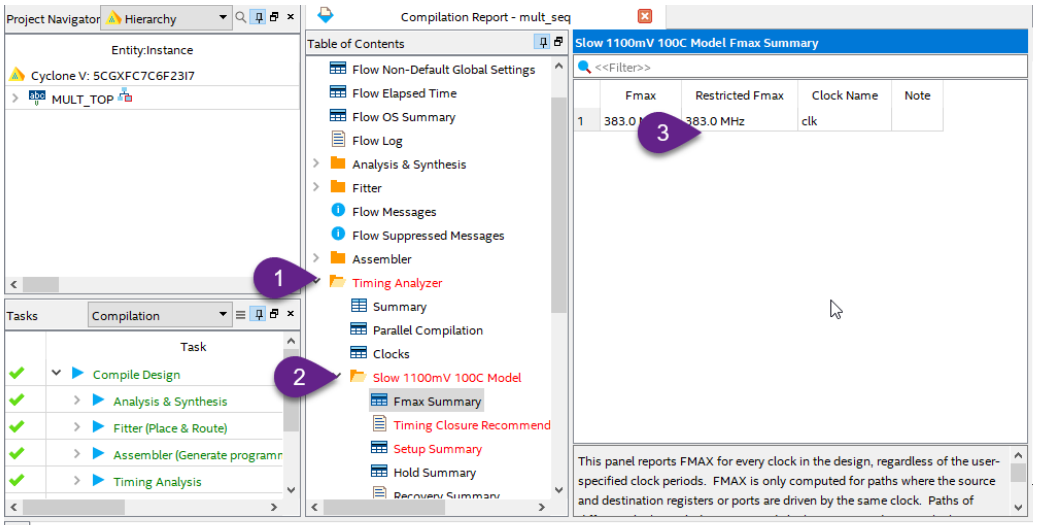
- Exponential Accelerator Engine
- Exponential Accelerator Wrapper
- Implementing Accelerator on FPGA

1 Exponential Engine

The accelerator that you are going to use is an exponential circuit. You are familiar with this accelerator design. As Figure 2 shows, this module receives a 16-bit input ”x” and generates a 16-bit output ”Fractionalpart” and 2-bit ”Integerpart”. Remember that the 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, first, you need to explore the design accuracy. Furthermore you as a designer need to be aware of the maximum frequency of this accelerator. The Verilog description code for this module is provided to you.

1. Examine the code and its accuracy by running Modelsim simulation. For this purpose, write a testbench for this design with at least three different values for input ”x”. Show the results by taking pictures of the simulation results.
2. Synthesize this design in Quartus II Software. Show the synthesis results in your report.
3. After synthesizing the design, you can find out the maximum frequency of this accelerator by referring to the Timing Analyzer reports in the Quartus synthesis tool. You can follow the steps shown in the Figure 3

Figure 3: Steps for observing maximum frequency



	Fmax	Restricted Fmax	Clock Name	Note
1	383.0 M	383.0 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

2 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. One of the applications that makes use of such multi-value exponential calculation is an activation function in Deep Neural Networks (DNN).

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

Multiple of these exponential values can be calculated with one accelerator instead of using one accelerator unit for each xi. To reduce the required hardware resources for the softmax exponential function, a mathematical transformation would be useful. Each input value is split into an integer number zi and a fractional number vi as below:

$$x_i = z_i + v_i$$

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

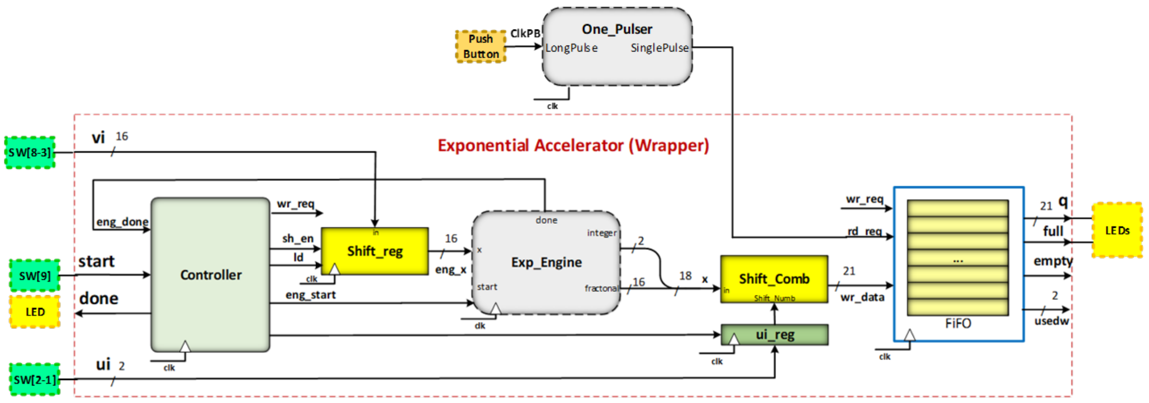
The second segment of this equation can be easily implemented with the exponential engine. To reduce the complexity of the hardware implementation of this equation, we use the base number 2 to take place of the base number e and the exponential function will be changed to:

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

or

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

Figure 4: Wrapper for exponential accelerator



where

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

As can be seen, this equation 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. Some other tasks are also included in the wrapper that are explained below:

- Referring to Figure 4, the wrapper receives single input in the form of a fractional value, v_i, and an integer value u_i and a start signal from the processor.
- Considering the input values are within u_i and u_i+1 we can calculate n number of exponential in this range as follows:

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

  In this experiment n=4. Based on these assumptions, 4 exponential values can be calculated. Four different values can be generated with the shift register unit by first registering the value of v_i 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.

- For shifting the output of the exponential engine, a combinational shifter is required. The input of this shifter is the u_i value that is provided outside the wrapper. To store the value of u_i, a register called *uireg* is used.
- When an exponential value is calculated then it should be stored in a FIFO so that when the CPU finishes its work it can retrieve all the results. 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. As shown in the Figure 4, the FIFO has a *writereq* input and a write data for writing into the buffer and a *readreq* for reading the outputs. In this experiment, you will use the FIFO IP provided in Quartus Megawizard. All you need is to connect the proper signals. Since we are limited to four calculations in this experiment the FIFO size would be four as well.

1. Show the state diagram of the controller and write the Verilog description of this module in a Huffman style.
2. Write a Verilog description for the wrapper shown in Figure 4.
3. Write a testbench and make an instance of this wrapper in your testbench.
4. At first provide u_i and v_i values and also generate a complete pulse on the signal "start" for the accelerator wrapper.
5. Test your design for at least 2 values of u_i and v_i. Include expected and achieved results in your report and make a comparison.
6. After synthesizing design, find out the maximum frequency of this accelerator wrapper by referring to the Timing Analyzer reports in the Quartus synthesis tool. You can follow the steps shown in the Figure 3

3 Implementing Accelerator on FPGA

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

1. Synthesize the design and include the synthesis report in your documents.
2. Connect 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. You need to issue the *start*" at the beginning of each round.
3. To feed the 16-bit fractional values of v_i to your design, you will use 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. Assign six switches to the rest six most significant bits of v_i. For u_i use two switches SW[2] and SW[1]. SW[0] will be used for the reset signal. Use the 50 MHz clock frequency of FPGA as the clock of the wrapper.
4. 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 you designed in experiment 2. Use a push-button as the input of a one-pulser circuit.

5. To show the 21-bit result values use LEDs on the board. Use one LED for the signal done, 5 LEDs for the integer part, and the rest of the LEDs for the most significant bits of the fractional part.

6. Test your design and include the results in your report. Report the values of ”Exp-out” and include images of board implementation.

Acknowledgment

This lab manual was prepared and developed by Katayoon Basharkhah, Ph.D. student of Digital Systems at the University of Tehran, under the supervision of Professor Zain Navabi.

This manual has been revised and edited by Zahra Jahanpeima, Ph.D. student of Digital Systems at University of Tehran.