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

Experiment 4

Sessions 8,9

# Accelerator and Wrappers

Digital Logic Laboratory

ECE 045

Laboratory Manual

Spring 1401

## Contents

**Contents** **1**

**Introduction** **2**

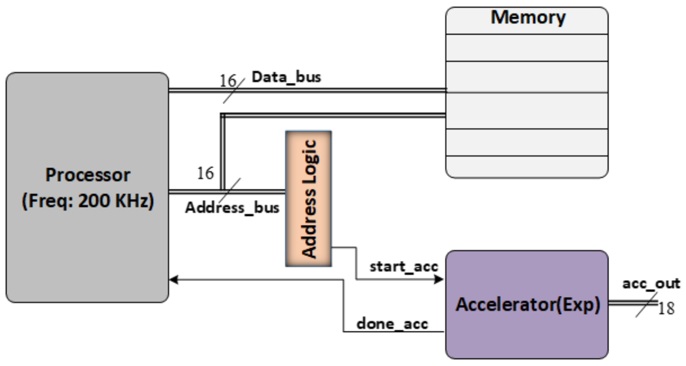
**1 Exponential Engine** **3**

**2 Exponential Accelerator Wrapper** **4**
2.1 Accelerator Buffers . . . . . . . . . . 4
2.2 The Wrapper Controller . . . . . . . . . . 5

**3 Implementing Accelerator on FPGA** **5**

**Acknowledgment** **6**

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 all 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 a memory, Input/Output ports and accelerators.

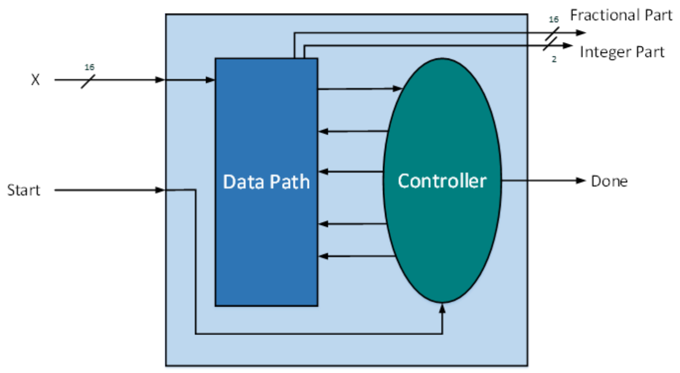
As you have learned from the Digital Logic Design course, accelerators are dedicated computation units that usually execute one specific task. This single task, needs a smaller and less complicated datapath which leads to a high frequency of operation for the accelerators. This is in contrary to CPUs in which millions of operations must be executed within a fix time interval. This impose a low frequency of operation for CPUs. To increase the speed of an SOC, hardware accelerators are usually embedded in the system. The processor will dispute some of its tasks to the hardware accelerator and during this time the accelerator performs several of the same or different operations and store 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 in an SOC.

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

- The concept of an SOC
- The concept of handshaking in an 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 is in communication with CPU talks to 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 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, CPU initiates the accelerator for an N round exponential estimation. CPU uses its address bus for initiating a component. By decoding

Figure 2: Block diagram of exponential accelerator



the address bus through an address logic as shown in figure 1, 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 details:

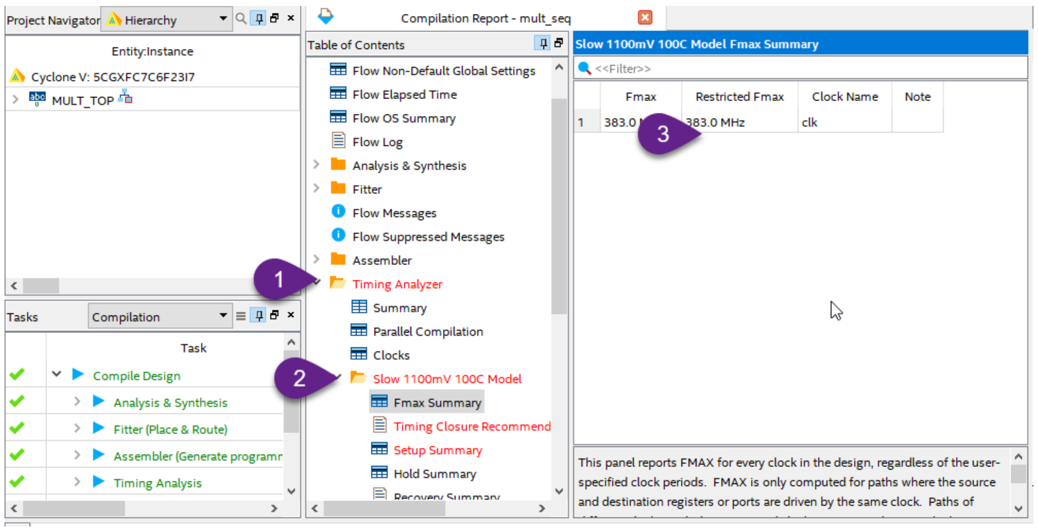
- Exponential Accelerator Engine
- Exponential Accelerator Wrapper
- Implementing Accelerator on FPGA

## 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 an 16-bit output "Fractionalpart" and 2-bit "Integerpart". Remember that the x value fits within 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 the purpose of 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. First 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 picture of the simulation results.
2. Synthesize this design in Quartus II Software. Show the synthesis results in your report.
3. After synthesizing design, you can find out the maximum frequency of this accelerator by referring to the Timing Analyzer reports in Quartus synthesis tool. You can follow the steps shown in the Figure 3

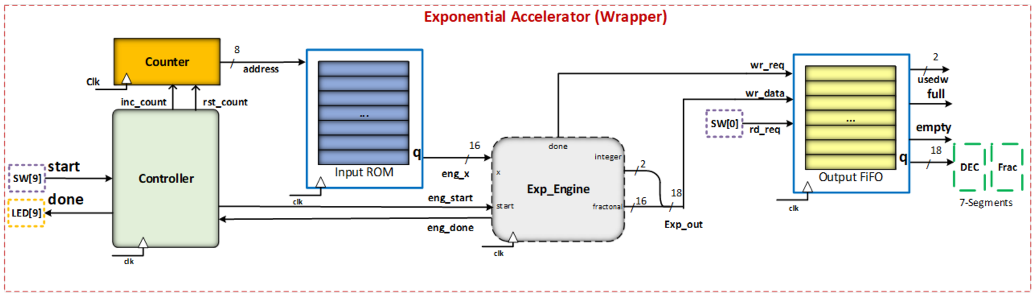
Figure 3: Steps for observing maximum frequency



| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 383.0 | 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

Figure 4: Exponential accelerator wrapper



## 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 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. For this purpose the exponential engine that is provided to you should be wrapped in a wrapper as shown in Figure 4.

### 2.1 Accelerator Buffers

Since the accelerator data will be accessed before and after completing CPU task, the data has to be stored in memory elements in the accelerator wrapper when CPU is busy with other works. Two memory elements are needed in this experiment, one is an input ROM for storing the input data and a FIFO for storing the output data. You have worked with ROM in the previous experiment. For the simplicity, the input data for the accelerator will be stored as a mif file inside the ROM.However in real applications, a RAM or FIFO is usually used to be able to write new data every time. 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. 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.You will use a FIFO to store the results of exponential engine. In this experiment you will use the FIFO and ROM IP provided in Quartus Megawizard. All you need is to connect the proper signals. The connection of buffers are shown in Figure 4.

## 2.2 The Wrapper Controller

The controller in this wrapper is responsible for generating the ”*start*” signal for exponential engine and the address of each input data reading from the input ROM. 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. To do this the controller issues the ”*inccount*” signal for reading data from the ROM. When all calculations are finished the controller sends a done signal on the wrapper output and issues the ”*rstcount*” signal to reset the counter for the next round of estimations.

1. Show the state diagram of the controller and write the Verilog description of this module in a Huffman style.
2. Make instances of FIFO and ROM IPs from Quatus Megawizard. For the ROM IP, generate a 1-PORT 16bit*256Word ROM and initialize it with an mif file containing 5 input values. For the FIFO buffer, generate an 18bit*5Word FIFO IP.
3. Write a Verilog description for the wrapper shown in Figure 4.
4. Write a testbench and make instance of this wrapper in your tesbench.
5. Generate a complete pulse on the signal ”*start*” for the accelerator wrapper.
6. Include expected and achieved results (Exp-out) in your report and make a comparison.

## 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 5-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. Connect the ”*readreq*” of the FIFO to SW[0]. After each round is finished and LED[9] turns on, you can read the results one by one by setting SW[0] to one.

3. To show the 18-bit result values use 7-segments. Note that you need a converter for 7-segment display.
4. 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, PHD student of Digital Systems at University of Tehran, under the supervision of professor Zain Navabi.