Experiment #4 – Integrated System

Mohammad
Saadati,
810198410

***Abstract*— This document is a student report to experiment #4 of Digital Logic Laboratory course at ECE Department, University of Tehran. In this experiment, a processor is simulated alongside a frequency multiplier and an exponential calculator block (accelerator). The frequency multiplier generates a higher frequency clock for the accelerator using a clock divider (implemented in previous experiments) so the accelerator can work faster.**

***Keywords*— Clock, System-On-Chip, SOC, Frequency Multiplier, Accelerator, Modelsim, Quartus**

I. 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. 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 on operation. This is in contrary to CPU in which millions of operations must be executed within a fix time interval. This imposes a low frequency of operation for CPUs.

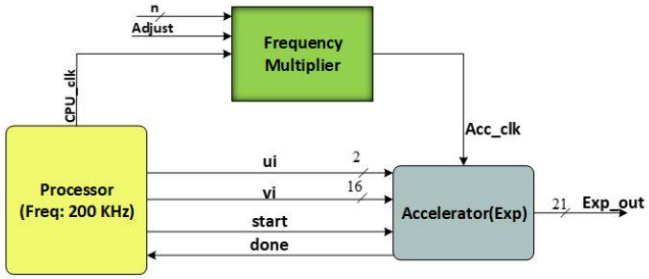

Fig. 1 Block diagram of a typical integrated circuit

II. Frequency Multiplier

This module receives a 4-bit number n[3:0] and a signal called f (inFreq) and outputs a signal with frequency $f \times 2^n$. The output signal is generated using a clock divider implemented in the previous experiment. The clock divider is consisted of an 8 bit counter and a T flip flop toggling each time the carry out bit of the counter becomes high. In this scenario, a refereneClk (Approximately 200MHz) is being divided.

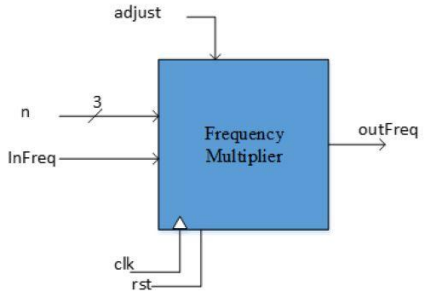

Fig. 2 Block diagram of the frequency multiplier

$$f \times 2^n = 200 \div k$$
$$k = 200 \div (f \times 2^n)$$

Dividing 200 by f is done by calculating the number of pulses on *f* during a complete pulse on the referenceClk (200MHz) similar to the idea of the *display* module in the previous projects. Dividing by 2^n is equivalent to shifting to right by n units.

By now we have calculated the value of k. Now we have to set a suitable parallel load for the counters so the output of the T-Flip Flop would have the desired frequency. The suitable value would be 255 – (k ÷ 2). Division by 2 again could be done using a shifter. The T-Flop-Flop would toggle with every k/2 clock cycles (Because it takes k/2 clock cycles for the counters to reach 255), therefore the output of the T-Flip-Flop would have frequency 200/k.

This module starts operating when receiving a complete pulse on *adjust*. This signal is issued by the CPU.

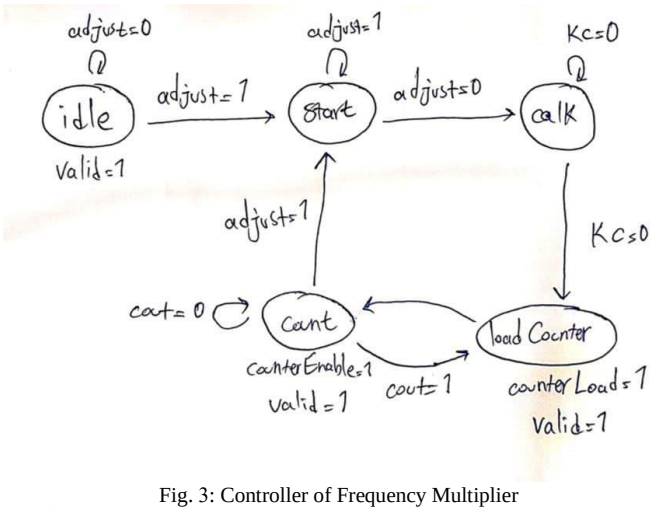

Fig. 3: Controller of Frequency Multiplier

After seeing a negative edge on adjust, the process of calculating *k* starts (Similar to display module). When calculation of *k* is done (kc = 1), it goes to the state which sets the appropriate parallel load for the counter. It starts counting afterwards until the carry out bit of the counter becomes high and

then it sets the appropriate parallel loads and starts counting again. Whenever there is a new pulse on adjust, it returns to start state.
A register could be used to store the value of k since it is changing but there is no problem caused here since the value loaded on the counters is calculated using the right value of k and it hasn't started counting again from zero.

Here are some examples used to test this module:

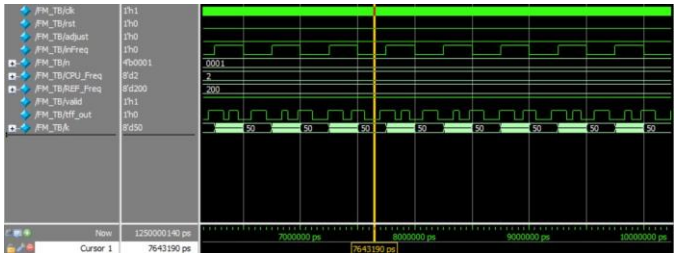

Fig. 4: CPU_Freq = 2MHz , n = 1
$k = 200 / (2 \times 2^1) = 50$
Within a period of tff_out, there are about 50 clock pulses.

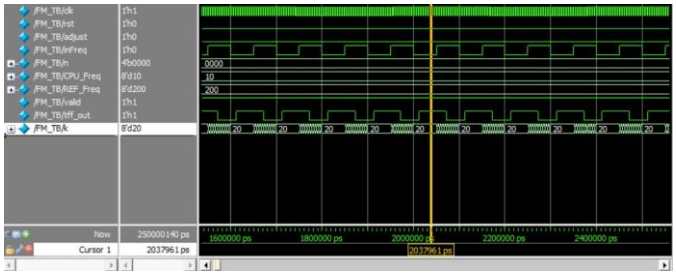

Fig. 5: CPU_Freq = 10MHz , n = 0
$k = 200 / (10 \times 2^0) = 20$
Within a period of tff_out, there are about 20 clock pulses.

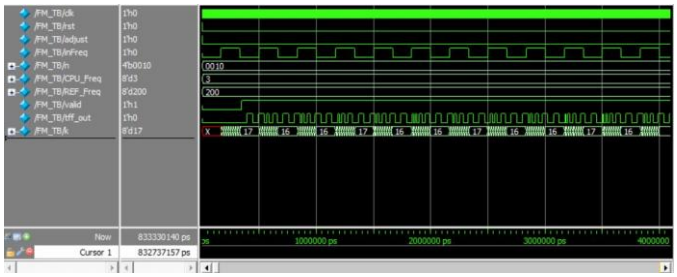

Fig. 10: CPU_Freq = 3MHz , n = 2
$k = 200 / (3 \times 2^2) = 16.67$
Within a period of tff_out, there are about 17 clock pulses.

III. EXPONENTIAL ACCELERATOR

A. *Exponential Engine*

This module receives a 16-bit input "x" and generates a 16-bit output "FractionalPart" and a 2-bit "IntegerPart". The accelerator starts working with a complete pulse on start.

When the computation is finished a "done" is signal is issued to be high so that the processor would understand this.

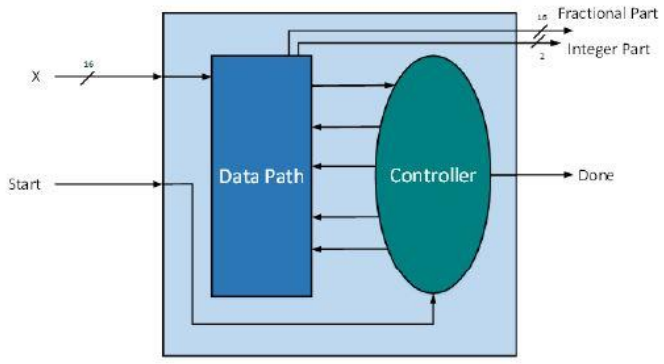

Fig. 7: Block diagram of exponential accelerator

The correctness of the exponential accelerator is evaluated below given 3 different values for x[15:0].

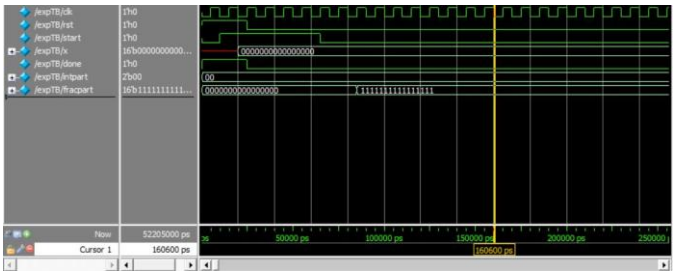

Fig. 8: x = 0
$e0 = 1 \approx 00.1111111111111111$

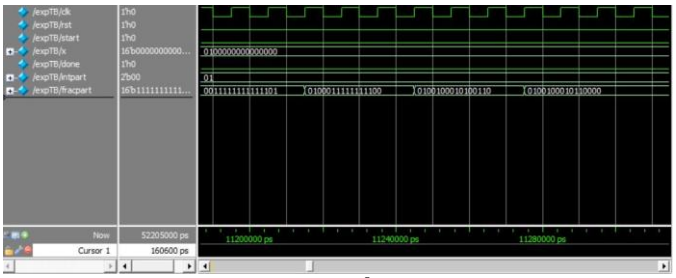

Fig. 9: $x = 2^{-2} = 0.25$
$e^{0.25} \approx 1.28402541669 \approx 01.0100100010110000$

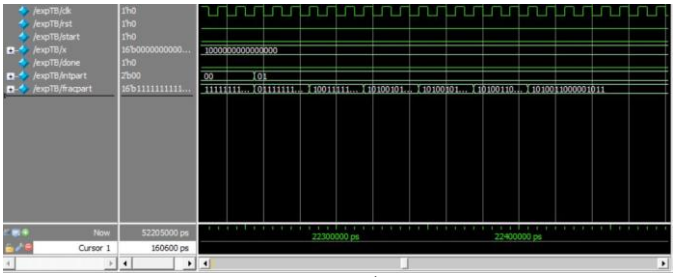

Fig. 10: $x = 2^{-1} = 0.5$
$e^{0.5} \approx 1.6487212707 \approx 01.1010011000001011$

Synthesis shows that the maximum frequency that this module can operate with is 134.86MHz on EP4CE6E22A7 Slow 1200mV 125C model and 156.13MHz on EP4CE6E22A7 Slow 1200mV -40C.

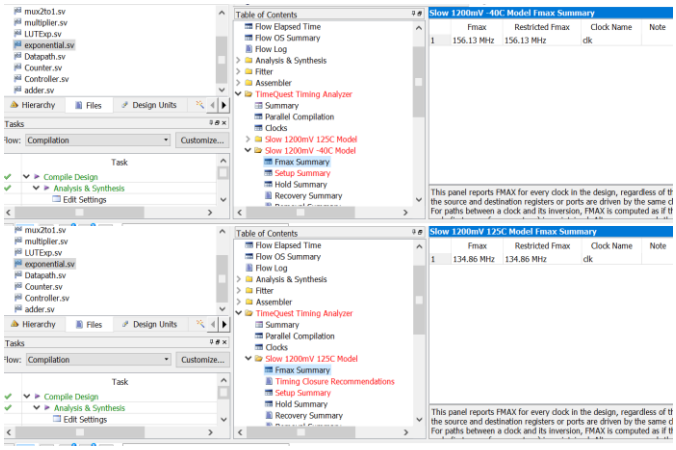

Fig. 11 Synthesis result

IV. INTEGRATED CIRCUIT

The block diagram of the typical integrated figure is shown in Fig 1. Before synthesis, a test bench is written for testing.

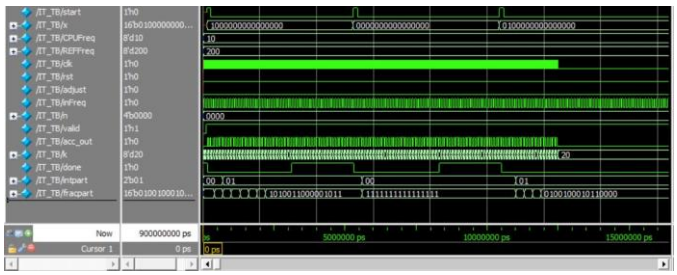

Figure 12: Simulation result IntegratedCircuitTB.v

A block of the FrequencyMultiplier and the ExponentialAccelerator is built in Quartus and the output of the FrequencyMultiplier is used for the clock of the accelerator as below:

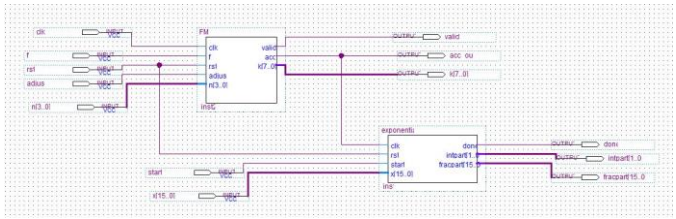

Fig 13: Block diagram of the synthesized circuit

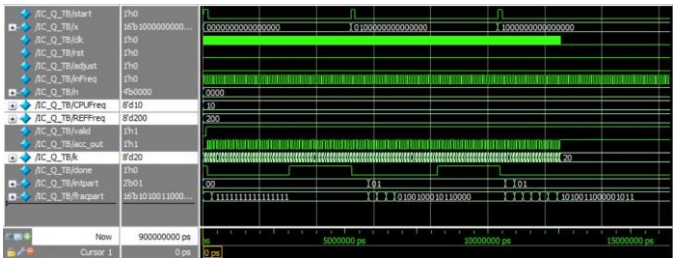

Figure 14: Simulation of the synthesized circuit with CPUFreq = 10MHz and refFreq = 200MHz and n = 0.

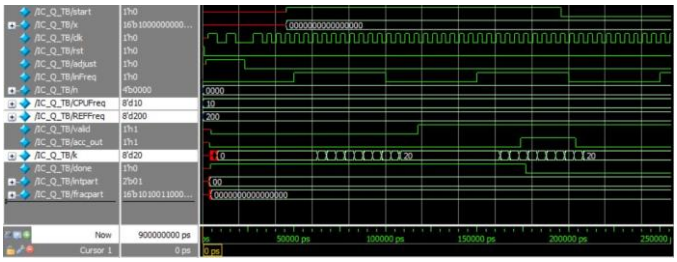

Figure 15: Beginning of simulation of the synthesized circuit with CPUFreq = 10MHz and refFreq = 200MHz and n = 0.
When valid = 1 it means that acc_out (Clock used for accelerator) is ready. Afterwards a start pulse is given to the accelerator telling it to start the computation since it's clock is ready.

When the frequency of it's clock is increased, it operates faster but we also have a upper bound limit for this frequency which is about 130MHz.
Limits for n is calculated the following way:

$$f_{CPU} \times 2^n < f_{max}$$

Considering that f_{CPU} = 10MHz and f_{max} = 134MHz (Synthesis result of part II) we have:

$$2^n < 13.4$$

$$n <= 3$$

The value of n varies based on CPU Frequency and Maximum Frequency.