# Experiment 4 - Accelerator and Wrappers

Mohammad Amin Alinejad- 810199463 / Amirali Shahriary- 810100173

***Abstract*— In this experiment, we are going to design an Exponential Accelerator Wrapper. For this design we are going to plan exponential engine, Accelerator wrapper and its controller , then perform our full design on Quartus environment that help us implement this Accelerator on FPGA and finally watch the results on FPGA board.**

***Keywords*— SoC, CP, Accelerator, Exponential Engine, handshaking in an SoC, Exponential Accelerator Wrapper, CPU, ROM, FPGA**

## 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 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 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.

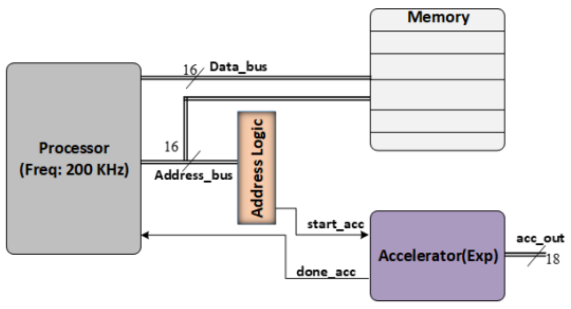


*Fig.1 Block diagram of a typical integrated circuit*

## II. Exponential Engine

this module receives a 16-bit input ”x” and generates a 16-bit output ”Fractionalpart” and 2-bit ”Integerpart”. The accelerator starts working with a complete pulse on the signal ”start” and when the computation is completed signal ”done” will be sent to the processor to acknowledge it.

code examination by running Modelsim simulation:

maximum of value that can be run is $2^{16}$=65536 so when we want to give value to e we should divide it by the maximum capacity($2^{16}$).

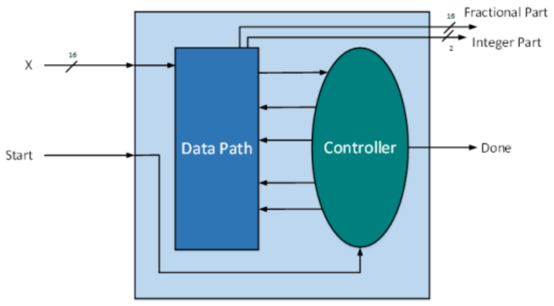
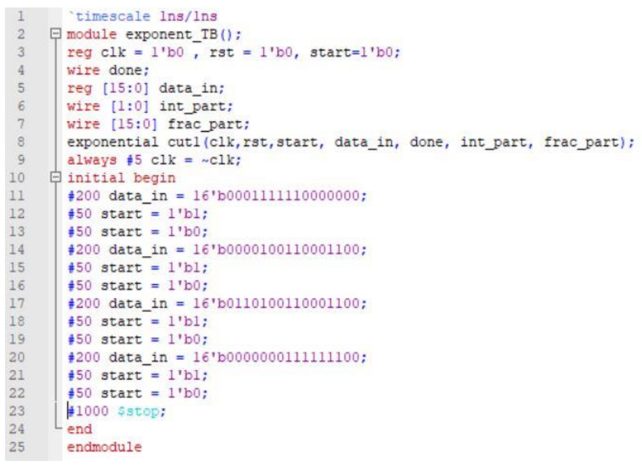


*Fig.2 Block diagram of exponential accelerator*

We simulated the code with 3 values of x = 8064 , 2444, 27020, 508. The Verilog description for calculation of (e) was given, the test bench code is below.



```
`timescale 1ns/1ns
module exponent_TB();
reg clk = 1'b0 , rst = 1'b0, start=1'b0;
wire done;
reg [15:0] data_in;
wire [1:0] int_part;
wire [15:0] frac_part;
exponential cut1(clk,rst,start, data_in, done, int_part, frac_part);
always #5 clk = ~clk;
initial begin
#200 data_in = 16'b0001111110000000;
#50 start = 1'b1;
#50 start = 1'b0;
#200 data_in = 16'b0000100110001100;
#50 start = 1'b1;
#50 start = 1'b0;
#200 data_in = 16'b0110100110001100;
#50 start = 1'b1;
#50 start = 1'b0;
#200 data_in = 16'b0000000111111100;
#50 start = 1'b1;
#50 start = 1'b0;
#1000 $stop;
end
endmodule
```

*Fig.3 Verilog description for the test bench of accelerator.*

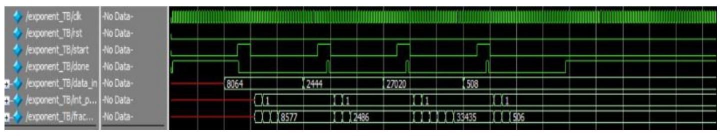


*Fig.4 Value of Exponential accelerator for all the modes.*

The calculations for each mode are listed below

1) $x = 8064$ , $8064/2^{16}$ ≈0.123046
$e^{0.123046} \approx\ 1.130936$

Output shown as shown in figure is 8577
$8577/2^{16} \approx 0.1308746$

As it turns out, the answer is almost equal to the expected output.

$$2)\quad x = 2444\,,\ 2444/2^{16} \approx 0.037292$$
$$e^{0.037292} \approx\ 1.037996$$

Output shown as shown in figure is 2486

$$2486/2^{16} \approx 0.037933$$

As it turns out, the answer is almost equal to the expected output.

$$3)\ x = 27020\,,\ 27020/2^{16} \approx 0.412292$$
$$e^{0.412292} \approx\ 1.51028$$

Output shown as shown in figure is 33435

$$33435/2^{16} \approx 0.510176$$

As it turns out, the answer is almost equal to the expected output.

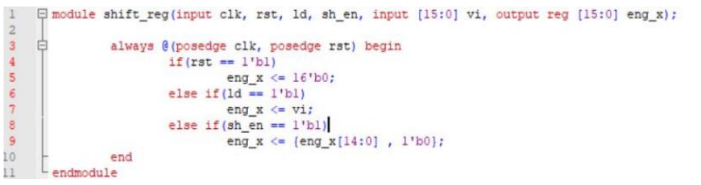
$$4)\ x = 508\,,\ 508/2^{16} \approx 0.00775146$$
$$e^{0.00775146} \approx\ 1.007751$$

Output shown as shown in figure is 506

$$506/2^{16} \approx 0.00772094$$

As it turns out, the answer is almost equal to the expected output.

Some of experiment components &controller Verilog description:



```verilog
module shift_reg(input clk, rst, ld, sh_en, input [15:0] vi, output reg [15:0] eng_x);

        always @(posedge clk, posedge rst) begin
                if(rst == 1'b1)
                        eng_x <= 16'b0;
                else if(ld == 1'b1)
                        eng_x <= vi;
                else if(sh_en == 1'b1)
                        eng_x <= {eng_x[14:0] , 1'b0};
        end
endmodule
```

Fig.5 Verilog description for shift register.

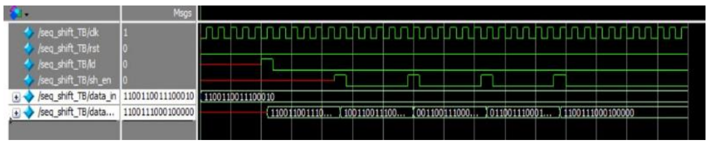
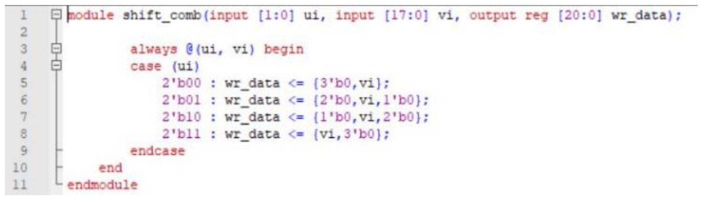


Fig.6 waveform output for shift register.



```verilog
module shift_comb(input [1:0] ui, input [17:0] vi, output reg [20:0] wr_data);

        always @(ui, vi) begin
        case (ui)
            2'b00 : wr_data <= {3'b0,vi};
            2'b01 : wr_data <= {2'b0,vi,1'b0};
            2'b10 : wr_data <= {1'b0,vi,2'b0};
            2'b11 : wr_data <= {vi,3'b0};
        endcase
    end
endmodule
```

Fig.7 Verilog description for combinational shift.

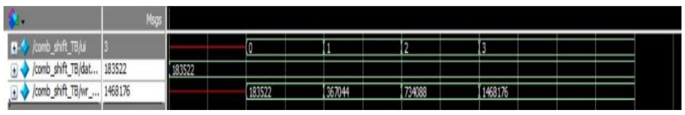
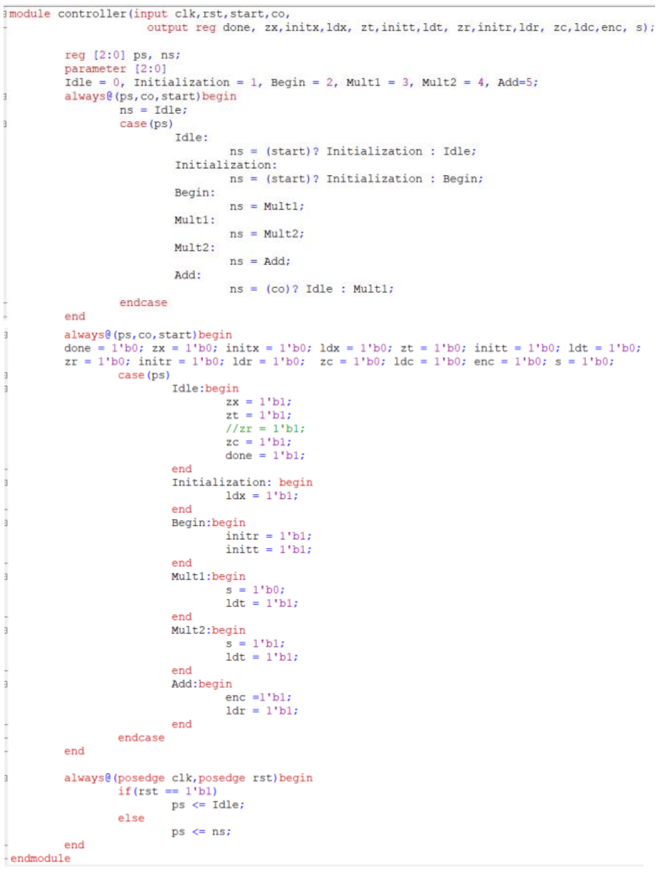


Fig.8 waveform output for combinational shift register.



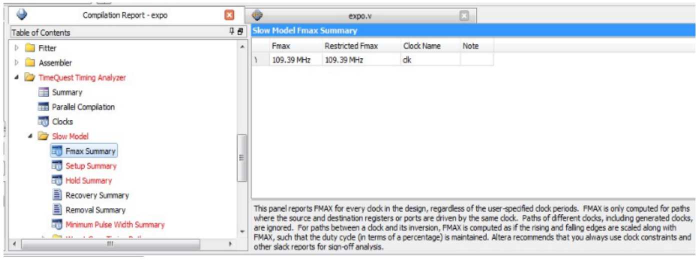
```verilog
module controller(input clk,rst,start,co,
                  output reg done, zx,initx,ldx, zt,initt,ldt, zr,initr,ldr, zc,ldc,enc, s);

        reg [2:0] ps, ns;
        parameter [2:0]
        Idle = 0, Initialization = 1, Begin = 2, Mult1 = 3, Mult2 = 4, Add=5;
        always@(ps,co,start)begin
                ns = Idle;
                case(ps)
                        Idle:
                                ns = (start)? Initialization : Idle;
                        Initialization:
                                ns = (start)? Initialization : Begin;
                        Begin:
                                ns = Mult1;
                        Mult1:
                                ns = Mult2;
                        Mult2:
                                ns = Add;
                        Add:
                                ns = (co)? Idle : Mult1;
                endcase
        end
        always@(ps,co,start)begin
        done = 1'b0; zx = 1'b0; initx = 1'b0; ldx = 1'b0; zt = 1'b0; initt = 1'b0; ldt = 1'b0;
        zr = 1'b0; initr = 1'b0; ldr = 1'b0;  zc = 1'b0; ldc = 1'b0; enc = 1'b0; s = 1'b0;
                case(ps)
                        Idle:begin
                                zx = 1'b1;
                                zt = 1'b1;
                                //zr = 1'b1;
                                zc = 1'b1;
                                done = 1'b1;
                        end
                        Initialization: begin
                                ldx = 1'b1;
                        end
                        Begin:begin
                                initr = 1'b1;
                                initt = 1'b1;
                        end
                        Mult1:begin
                                s = 1'b0;
                                ldt = 1'b1;
                        end
                        Mult2:begin
                                s = 1'b1;
                                ldt = 1'b1;
                        end
                        Add:begin
                                enc =1'b1;
                                ldr = 1'b1;
                        end
                endcase
        end

        always@(posedge clk,posedge rst)begin
                if(rst == 1'b1)
                        ps <= Idle;
                else
                        ps <= ns;
        end
endmodule
```

Fig.9 controller Verilog description.

As shown in the figure below, The maximum frequency that this exponential calculator can operate is 109.39MHz which we got access to from Fmax summary.



Slow Model Fmax Summary

| | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 109.39 MHz | 109.39 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 different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera recommends that you always use clock constraints and other slack reports for sign-off analysis.

Fig.10 Fmax of Exponential Engine.

## III. EXPONENTIAL ACCELERATOR WRAPPER

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

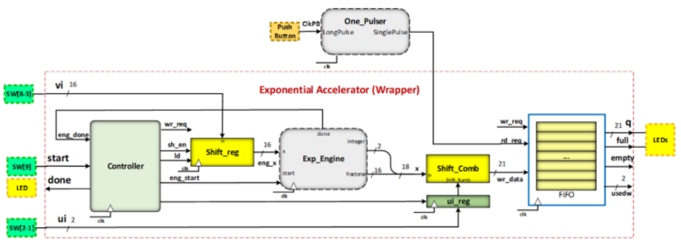


Fig.11 Exponential accelerator wrapper

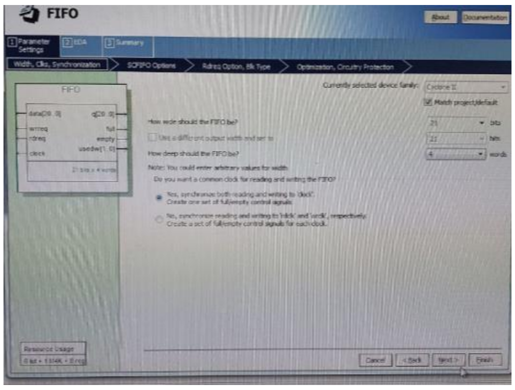
busy with other works. The memory element required in this experiment is an input ROM for storing the input data.

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

## IV. Implementing Accelerator on FPGA

By using mega wizard unit, we declare a FIFO unit. FIFO is a group of register which we can store data and it has both read and write option.

The specific option of FIFO is that the reading data are sorted as they entered into the FIFO unit.



Currently selected device family: Cyclone II

☑ Match project/default

How wide should the FIFO be? 21 bits

☐ Use a different output width and set to 21 bits

How deep should the FIFO be? 4 words

Note: You could enter arbitrary values for width

Do you want a common clock for reading and writing the FIFO?

● Yes, synchronize both reading and writing to 'clock'. Create one set of full/empty control signals

○ No, synchronize reading and writing to 'rdclk' and 'wrclk', respectively. Create a set of full/empty control signals for each clock.

Fig.12 Creating FIFO.

In this part, we are about to connect the main circuit into the FIFO unit and one-pulser. After that we define the inputs and outputs to plan them in pin planner.

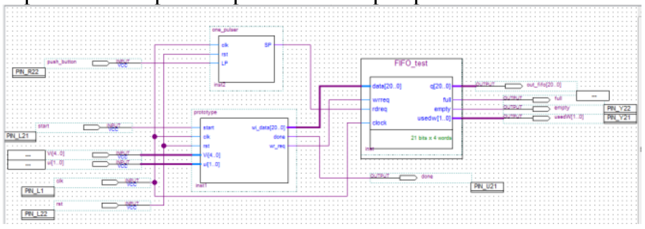


Fig.13 schematic of the circuit.

To map the circuit on device we need pin assignment using *Cyclone EP2C20F484C7* manual.

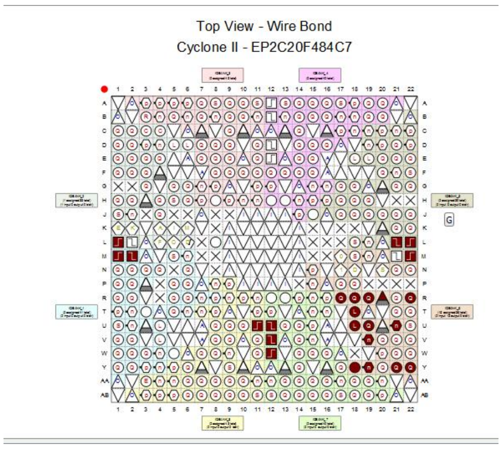
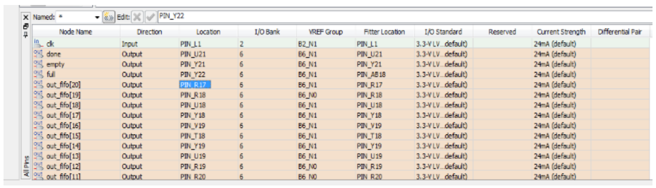
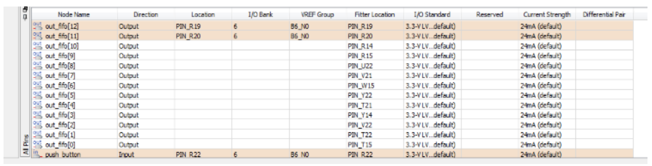


Fig.14 Pin assignment top view.

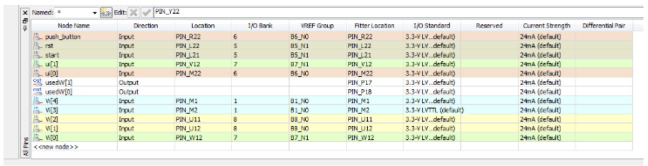


Named: * Edit: PIN_Y22

| Node Name | Direction | Location | I/O Bank | VREF Group | Fitter Location | I/O Standard | Reserved | Current Strength | Differential Pair |
|---|---|---|---|---|---|---|---|---|---|
| clk | Input | PIN_L1 | 2 | B2_N1 | PIN_L1 | 3.3-V LV...default) | | 24mA (default) | |
| done | Output | PIN_U21 | 6 | B6_N1 | PIN_U21 | 3.3-V LV...default) | | 24mA (default) | |
| empty | Output | PIN_Y21 | 6 | B6_N1 | PIN_Y21 | 3.3-V LV...default) | | 24mA (default) | |
| full | Output | PIN_Y22 | 6 | B6_N1 | PIN_AB18 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[20] | Output | PIN_R17 | 6 | B6_N1 | PIN_R17 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[19] | Output | PIN_R18 | 6 | B6_N0 | PIN_R18 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[18] | Output | PIN_U18 | 6 | B6_N1 | PIN_U18 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[17] | Output | PIN_Y18 | 6 | B6_N1 | PIN_Y18 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[16] | Output | PIN_V19 | 6 | B6_N1 | PIN_V19 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[15] | Output | PIN_T18 | 6 | B6_N1 | PIN_T18 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[14] | Output | PIN_Y19 | 6 | B6_N1 | PIN_Y19 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[13] | Output | PIN_U19 | 6 | B6_N1 | PIN_U19 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[12] | Output | PIN_R19 | 6 | B6_N0 | PIN_R19 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[11] | Output | PIN_R20 | 6 | B6_N0 | PIN_R20 | 3.3-V LV...default) | | 24mA (default) | |



| Node Name | Direction | Location | I/O Bank | VREF Group | Fitter Location | I/O Standard | Reserved | Current Strength | Differential Pair |
|---|---|---|---|---|---|---|---|---|---|
| out_fifo[12] | Output | PIN_R19 | 6 | B6_N0 | PIN_R19 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[11] | Output | PIN_R20 | 6 | B6_N0 | PIN_R20 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[10] | Output | | | | PIN_R14 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[9] | Output | | | | PIN_R15 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[8] | Output | | | | PIN_U22 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[7] | Output | | | | PIN_V21 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[6] | Output | | | | PIN_W15 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[5] | Output | | | | PIN_Y22 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[4] | Output | | | | PIN_T21 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[3] | Output | | | | PIN_Y14 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[2] | Output | | | | PIN_V22 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[1] | Output | | | | PIN_T22 | 3.3-V LV...default) | | 24mA (default) | |
| out_fifo[0] | Output | | | | PIN_T15 | 3.3-V LV...default) | | 24mA (default) | |
| push_button | Input | PIN_R22 | 6 | B6_N0 | PIN_R22 | 3.3-V LV...default) | | 24mA (default) | |



Named: * Edit: PIN_Y22

| Node Name | Direction | Location | I/O Bank | VREF Group | Fitter Location | I/O Standard | Reserved | Current Strength | Differential Pair |
|---|---|---|---|---|---|---|---|---|---|
| push_button | Input | PIN_R22 | 6 | B6_N0 | PIN_R22 | 3.3-V LV...default) | | 24mA (default) | |
| rst | Input | PIN_L22 | 5 | B5_N1 | PIN_L22 | 3.3-V LV...default) | | 24mA (default) | |
| start | Input | PIN_L21 | 5 | B5_N1 | PIN_L21 | 3.3-V LV...default) | | 24mA (default) | |
| ui[1] | Input | PIN_V12 | 7 | B7_N1 | PIN_V12 | 3.3-V LV...default) | | 24mA (default) | |
| ui[0] | Input | PIN_M22 | 6 | B6_N0 | PIN_M22 | 3.3-V LV...default) | | 24mA (default) | |
| usedW[1] | Output | | | | PIN_P17 | 3.3-V LV...default) | | 24mA (default) | |
| usedW[0] | Output | | | | PIN_P18 | 3.3-V LV...default) | | 24mA (default) | |
| Vi[4] | Input | PIN_M1 | 1 | B1_N0 | PIN_M1 | 3.3-V LV...default) | | 24mA (default) | |
| Vi[3] | Input | PIN_M2 | 1 | B1_N0 | PIN_M2 | 3.3-V LVTTL (default) | | 24mA (default) | |
| Vi[2] | Input | PIN_U11 | 8 | B8_N0 | PIN_U11 | 3.3-V LV...default) | | 24mA (default) | |
| Vi[1] | Input | PIN_U12 | 8 | B8_N0 | PIN_U12 | 3.3-V LV...default) | | 24mA (default) | |
| Vi[0] | Input | PIN_W12 | 7 | B7_N1 | PIN_W12 | 3.3-V LV...default) | | 24mA (default) | |
| <<new node>> | | | | | | | | | |

Fig.15,16&17 Pin assignment.

We assign push button of one-pulser into Key0. To declare done, empty and full signals we use LEDG1, LEDG6 and LEDG7 respectively.

The output result will be seen on LEDRs and initialization of $V_i$ and $U_i$ will be done on SWs.

After this pin planning process, it’s time to run the design and investigate the results.

First, we can watch our chip planner which show us that both logic and memory units are used to implement this design.

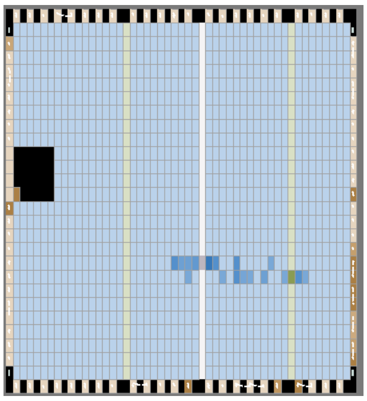
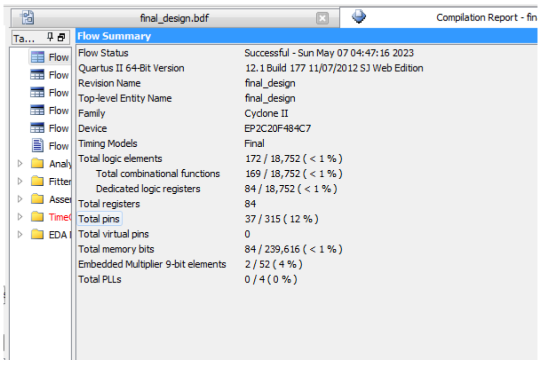


Fig.18 Chip planner.

We can check the summary report of synthesis too.



Flow Summary

| | |
|---|---|
| Flow Status | Successful - Sun May 07 04:47:16 2023 |
| Quartus II 64-Bit Version | 12.1 Build 177 11/07/2012 SJ Web Edition |
| Revision Name | final_design |
| Top-level Entity Name | final_design |
| Family | Cyclone II |
| Device | EP2C20F484C7 |
| Timing Models | Final |
| Total logic elements | 172 / 18,752 ( < 1 % ) |
| Total combinational functions | 169 / 18,752 ( < 1 % ) |
| Dedicated logic registers | 84 / 18,752 ( < 1 % ) |
| Total registers | 84 |
| Total pins | 37 / 315 ( 12 % ) |
| Total virtual pins | 0 |
| Total memory bits | 84 / 239,616 ( < 1 % ) |
| Embedded Multiplier 9-bit elements | 2 / 52 ( 4 % ) |
| Total PLLs | 0 / 4 ( 0 % ) |

Fig.19 Flow summary of synthesis.

After compiling the project, we run it on the FPGA. In the following pertaining images the implementation and testing can be seen.

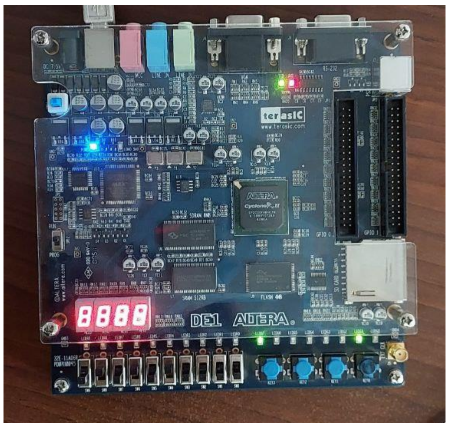


Fig.20 Accelerator and Wrappers circuit.

As it can be seen, at first the empty and done signals are active. After 1-0-1 pulse on start, the process will be started and with each pulse on push-button, we can see the output result on LEDRs. We showed the 10 most significant bits of exponential result on FPGA.

For initializing the $V_i$s, we fulfill it with 10101 and its concatenated form is 0001010100000000.

As we define this as a input data, our power numbers will be 0.081, 0.162, 0.324 and 0.648.

Now we are going to calculate the result and show it on FPGA.

$$e^{0.081} \approx 1.085$$
$$00001.00010101100101100 \approx 1.085$$

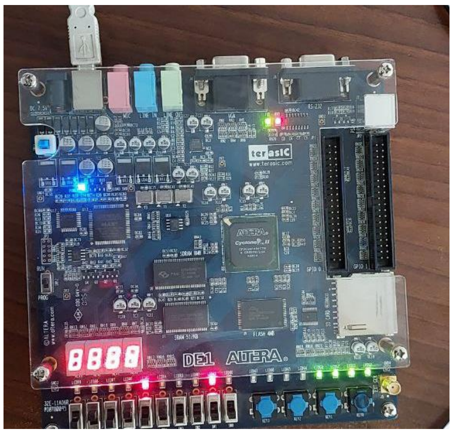


Fig.21 First step of showing result.

$$e^{0.162} \approx 1.187$$
$$00001.0010110110001010 \approx 1.187$$

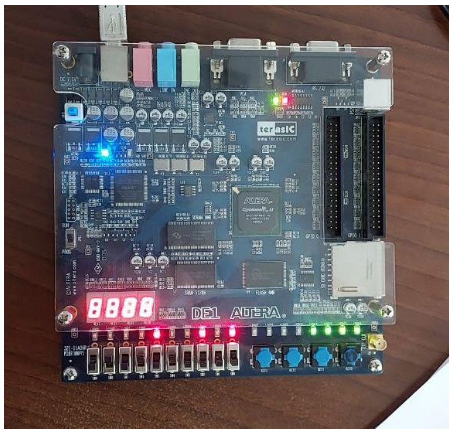


Fig.22 Second step of showing result.

$$e^{0.324} \approx 1.388$$
$$00001.0110001100110010 \approx 1.388$$

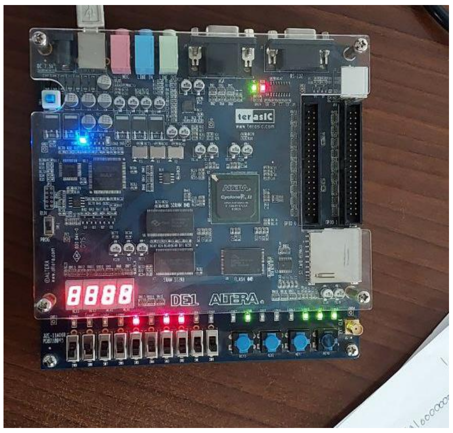


*Fig.23 Third step of showing result*.

## II. CONCLUSIONS

In this experiment, we learned how to use a accelerator and wrapper in designing a circuit and learned about the FIFO unit and how it works.

## III. REFERENCES

[1] Katayoon Basharkhah and Zahra Jahanpeim and Zain Navabi, *Digital Logic Laboratory*, University of Tehran, Fall 1402.