Experiment 4 – Accelerator and Wrappers

Mehdi Jamalkhah, 810100111

Mobina Mehrazar, 810100216

***Abstract*— This document is a report for experiment #3 of Digital Logic Design Laboratory at ECE department, University of Tehran. The purpose of this experiment is to generate a wide variety of waveforms with different amplitude and frequency with an Arbitrary Function Generator (AFG).**

***Keywords*—** accelerator**,** frequency**,** wrapper, exponential, CPU, Verilog, engine, SoC**.**

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 which needs a smaller and less complicated datapath, and 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 imposes 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 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.

A block diagram related to a typical Embedded system is shown in Fig. 1. This block diagram includes an accelerator and a memory.

3

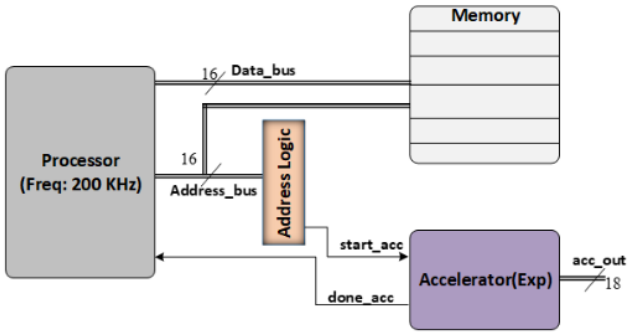


Fig. 1 Block diagram of a typical integrated circuit. Any components that is in communication with CPU talks to CPU via signals "start" and "done".

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 a round exponential estimation. CPU uses its address bus for initiating a component. By decoding the address bus through an address logic, accelerator will have its "start" signal issued when needed. For simplicity in this experiment we will implement the whole CPU and address logic inside the testbench and when implementing on a FPGA, we will feed "start" through board switches.

II. Exponential Engine

The accelerator that we are going to use is an exponential circuit. We will use the accelerator that designed in Digital Logic Design course. This accelerator starts working with a complete pulse on signal on "start" and when the computation is completed signal "done" be sent to the processor to acknowledge it. First we want to explore the design accuracy. Furthermore we need to be aware of maximum frequency of this accelerator to feed that with proper clock frequency.

A. Code examination:

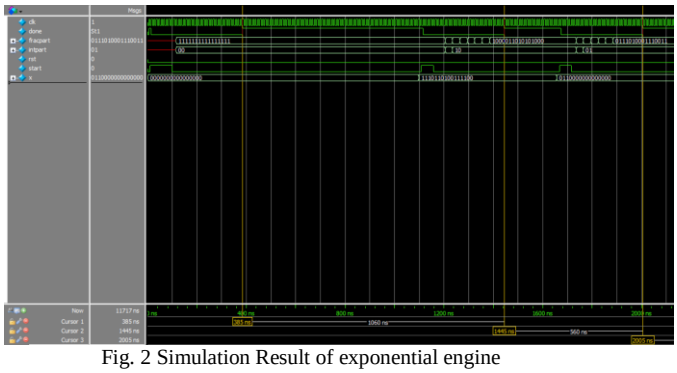


Fig. 2 Simulation Result of exponential engine

$$x_1 = 0_d = 0 \cdot 1111\_1111\_1111\_1111_b$$
$$e^{x_1} = 1_d = 00 \cdot 1111\_1111\_1111\_1111$$

$$x_2 = 0 \cdot 9266_d = 0 \cdot 1110\_1101\_0011\_1100_b$$
$$e^{x_2} = 10 \cdot 1000\_0110\_1011\_0001$$

$$x_3 = 0 \cdot 375_d = 0110\_0000\_0000\_0000_b$$
$$e^{x_3} = 1 \cdot 4549_d = 01 \cdot 0111\_0100\_0111\_0011_b$$

B. Design synthesize:

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Jun 07 02:19:32 2023
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	exponential
Top-level Entity Name	exponential
Family	Cyclone IV E
Device	EP4CE6E22C6
Timing Models	Final
Total logic elements	103 / 6,272 (2 %)
Total registers	60
Total pins	38 / 92 (41 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	2 / 30 (7 %)
Total PLLs	0 / 2 (0 %)

Fig. 3 Flow Summary of exponential engine

C. Maximum Frequency:

Slow 1200mV 85C Model Fmax Summary

<<Filter>>

	Fmax	Restricted Fmax	Clock Name	Note
1	162.05 MHz	162.05 MHz	clk	

162.05 MHz

Fig. 4 Maximum Frequency of exponential engine

III. 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.

A. Controller

1) State diagram:

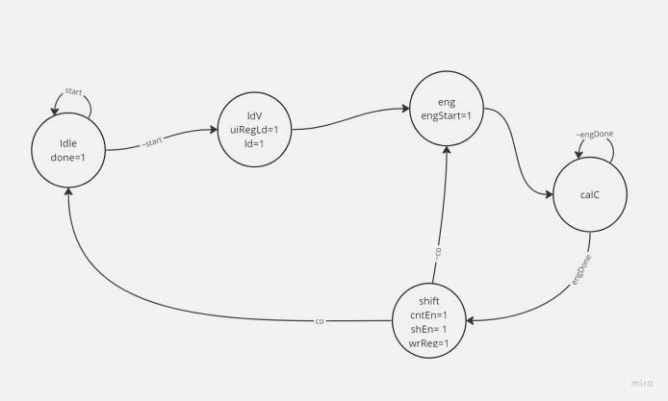


Fig. 5 Wrapper controller state diagram

2) Verilog description:

```
Ln#
 1  `timescale 1ns/1ns
 2  module ControlerWrapper(clk, rst, start, engDone, done, wrReq, shEn, ld, engStart, uiRegLd);
 3      input clk, start, rst, engDone;
 4      output done, wrReq, shEn, ld, engStart, uiRegLd;
 5      reg done, uiRegLd, ld, engStart, shEn, wrReq;
 6
 7      parameter [2:0] Idle = 3'd0, ldV = 3'd1, eng = 3'd2, calc = 3'd3, shift = 3'd4;
 8
 9      reg [2:0] ps, ns;
10      reg cntEn;
11      wire co;
12
13      twoBitCounter twobitcounter(.clk(clk), .rst(rst), .cntEn(cntEn), .co(co));
14
15      always @(posedge clk, posedge rst) begin
16          if (rst) begin
17              ps = Idle;
18          end
19          else begin
20              ps = ns;
21          end
22      end
23
24      always @(ps, start, engDone, co) begin
25          ns = Idle;
26          case(ps)
27              Idle: ns = start? ldV : Idle;
28              ldV: ns = eng;
29              eng: ns = calc;
30              calc: ns = engDone ? shift : calc;
31              shift: ns = co ? Idle : eng;
32              default: ns = Idle;
33          endcase
34      end
35
36      always @(ps) begin
37          {done, uiRegLd, ld, engStart, cntEn, shEn, wrReq} = 7'b0;
38          case(ps)
39              Idle: done = 1'b1;
40              ldV: {uiRegLd, ld} = 2'b11;
41              eng: engStart = 1'b1;
42              calc: ;
43              shift: {cntEn, shEn, wrReq} = 3'b111;
44              default: {done, uiRegLd, ld, engStart, cntEn, shEn, wrReq} = 7'b0;
45          endcase
46      end
```

Fig. 6 controller of wrapper Verilog code

B. Verilog description of wrapper:

```
Ln#
 1  `timescale 1ns/1ns
 2  module shiftReg(clk, shEn, ld, rst, dataIn, dataOut);
 3      input clk, shEn, ld, rst;
 4      input [15:0] dataIn;
 5      output [15:0] dataOut;
 6      reg [15:0] data;
 7      always @(posedge clk, posedge rst) begin
 8          if (rst) begin
 9              data = 16'b0;
10          end
11          else if (ld) begin
12              data = dataIn;
13          end
14          else if (shEn) begin
15              data = data << 1;
16          end
17      end
18      assign dataOut = data;
19  endmodule
```

Fig. 7 shift register Verilog code

```
Ln#
1   `timescale 1ns/1ns
2   module shiftComb(shiftNumb, dataIn, dataOut);
3       input [1:0] shiftNumb;
4       input [17:0] dataIn;
5       output [20:0] dataOut;
6       assign dataOut = {3'b0, dataIn} << shiftNumb;
7   endmodule
8
```

Fig. 8 shift combinational Verilog code

```
Ln#
1   `timescale 1ns/1ns
2   module uiReg(clk, uiRegLd, ui, uiOut);
3       input clk, uiRegLd;
4       input [1:0] ui;
5       output [1:0] uiOut;
6       reg [1:0] data;
7       always @(posedge clk) begin
8           if (uiRegLd) begin
9               data = ui;
10          end
11      end
12
13      assign uiOut = data;
14  endmodule
15
```

Fig. 9 u_i register Verilog code

```
Ln#
1   `timescale 1ns/1ns
2
3   module DataPathWrapper(clk, rst, ld, start, uiRegLd, shEn, ui, fracInput, done, wrData);
4       input clk, rst, ld, start, uiRegLd, shEn;
5       input [4:0] fracInput;
6       input [1:0] ui;
7       output done;
8       output [20:0] wrData;
9       wire [1:0] uiOut, intpart;
10      wire [15:0] engX, fracPart;
11
12      shiftReg shftRg (.clk(clk), .shEn(shEn), .ld(ld), .rst(rst), .dataIn({3'b0, fracInput, 8'b0}), .dataOut(engX));
13      shiftComb shftCmb (.shiftNumb(uiOut), .dataIn({intpart, fracPart}), .dataOut(wrData));
14      uiReg uRg (.clk(clk), .uiRegLd(uiRegLd), .ui(ui), .uiOut(uiOut));
15      exponential exp (.clk(clk), .rst(rst), .start(start), .x(engX), .done(done), .intpart(intpart), .fracpart(fracPart));
16
17  endmodule
```

Fig. 10 Wrapper datapath Verilog code

```
Ln#
1   `timescale 1ns/1ns
2
3   module Accelerator(clk, rst, wrStart, ui, vi,  done, wrData, wrReq);
4       input clk, wrStart, rst;
5       input [1:0] ui;
6       input [4:0] vi;
7       output done, wrReq;
8       output [20:0] wrData;
9
10
11      wire co, ld, engStart, uiRegLd, shEn, engDone;
12      ControlerWrapper cntrllerWrapper(.clk(clk), .rst(rst), .start(wrStart), .engDone(engDone),
13                                       .done(done), .wrReq(wrReq), .shEn(shEn), .ld(ld), .engStart(engStart), .uiRegLd(uiRegLd));
14      DataPathWrapper dPthWrapper (.clk(clk), .rst(rst), .ld(ld), .start(engStart), .uiRegLd(uiRegLd), .shEn(shEn),
15                                   .ui(ui), .fracInput(vi), .done(engDone), .wrData(wrData));
16  endmodule
17
```

Fig. 11 accelerator Verilog code

C. Simulation result of wrapper in Modelsim:

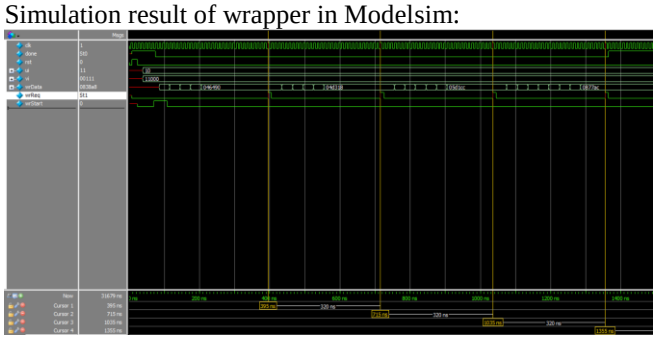


Fig. 12 simulation result part1

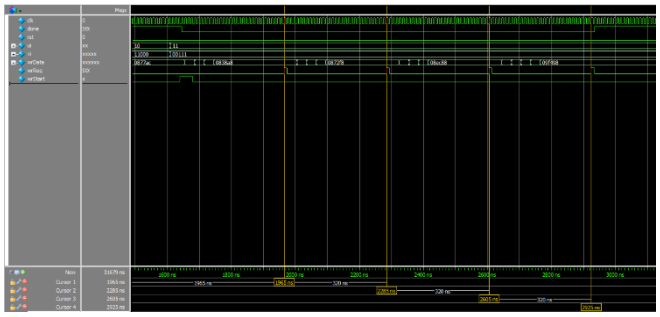


Fig. 13 simulation result part2

$$x = z + v = 2 + 0 \cdot 0937$$

$$\rightarrow u = 10, v = 11000$$

$$e^{x_1}_{expected} = 8 \cdot 115290514356445$$
$$e^{x_1}_{achieved} = 4 \cdot 392822265625$$

$$e^{x_2}_{expected} = 8 \cdot 912902981198736944925$$
$$e^{x_2}_{achieved} = 4 \cdot 8245849609375$$

$$e^{x_3}_{expected} = 10 \cdot 75101318607$$
$$e^{x_3}_{achieved} = 5 \cdot 81951904296875$$

$$e^{x_4}_{expected} = 15 \cdot 6426318841$$
$$e^{x_4}_{achieved} = 8 \cdot 46746826171875$$

This huge difference between expected value and achieved one is because of floor function in formula of z_i which mathematically achieved, which means this error is not related to the design.

D. Synthesize result in Quartus:

Flow Summary

<<Filter>>	
Flow Status	Successful - Wed Jun 07 03:14:23 2023
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	Accelerator
Top-level Entity Name	Accelerator
Family	Cyclone IV E
Device	EP4CE6E22C6
Timing Models	Final
Total logic elements	149 / 6,272 (2 %)
Total registers	69
Total pins	33 / 92 (36 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	2 / 30 (7 %)
Total PLLs	0 / 2 (0 %)

149 / 6,272 (2 %)

Fig. 14 Flow Summary

E. Maximum Frequency of this accelerator wrapper:

Slow 1200mV 85C Model Fmax Summary

<<Filter>>

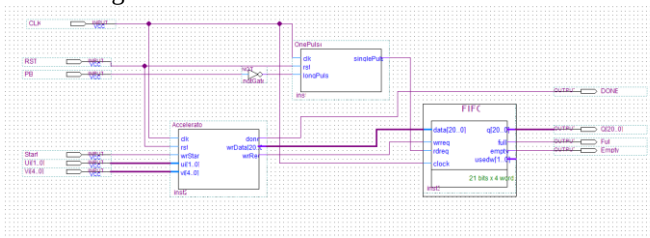
	Fmax	Restricted Fmax	Clock Name	Note
1	170.62 MHz	170.62 MHz	clk	

Fig. 15 Maximum Frequency of accelerator wrapper

IV. Implementing Accelerator on FPGA

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

A. Block Diagram:



B. Pin Planner:

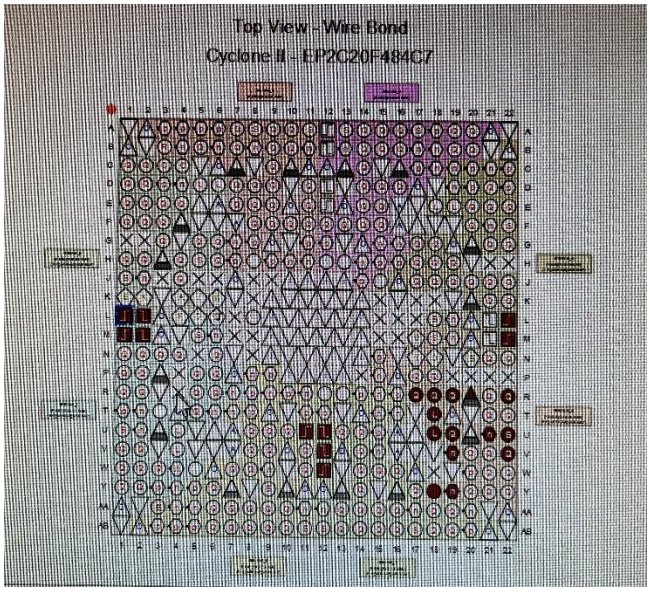


Fig. 16 Pin Planner part1

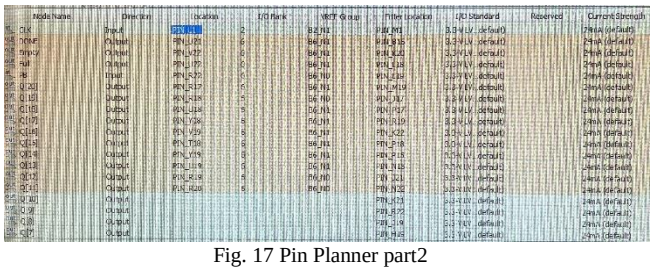


Fig. 17 Pin Planner part2

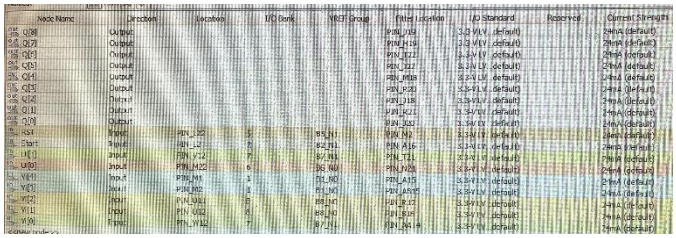


Fig. 18 Pin Planner part3

C. Design synthesizing (flow summary):

Flow Summary

Flow Status	Successful - Fri May 05 10:54:59 2023
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	Accelerator
Top-level Entity Name	Block1
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	161 / 18,752 (< 1 %)
Total combinational functions	160 / 18,752 (< 1 %)
Dedicated logic registers	79 / 18,752 (< 1 %)
Total registers	79
Total pins	35 / 315 (11 %)
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

D. Chip Planner:

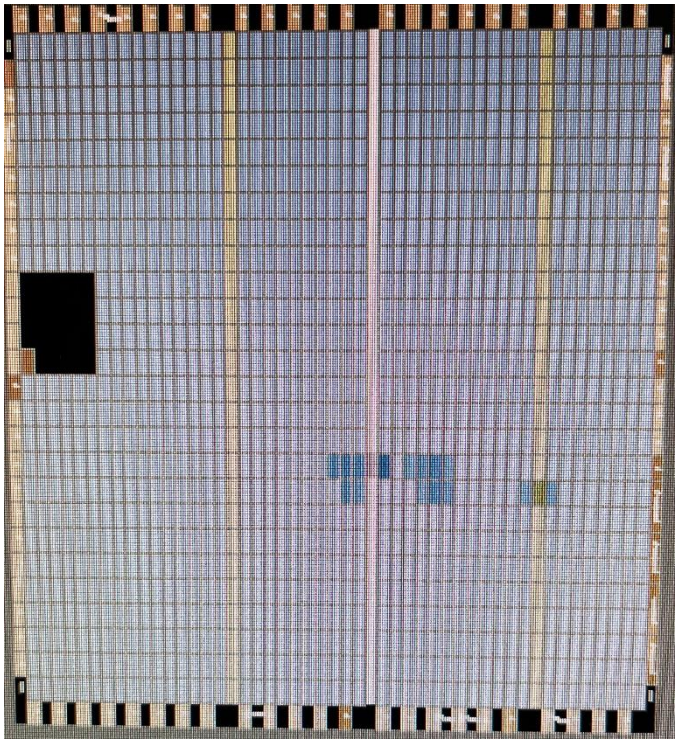


Fig. 20 Chip Planner

E. Design test:

We set the u_i to 10 and v_i to 11000 by using keys. Those values are same as given values of wrapper simulation in Modelsim which shown in previous part.

Then we generated a complete pulse on signal "start", after a moment the done signal would be issued(LED G_1 would become on).

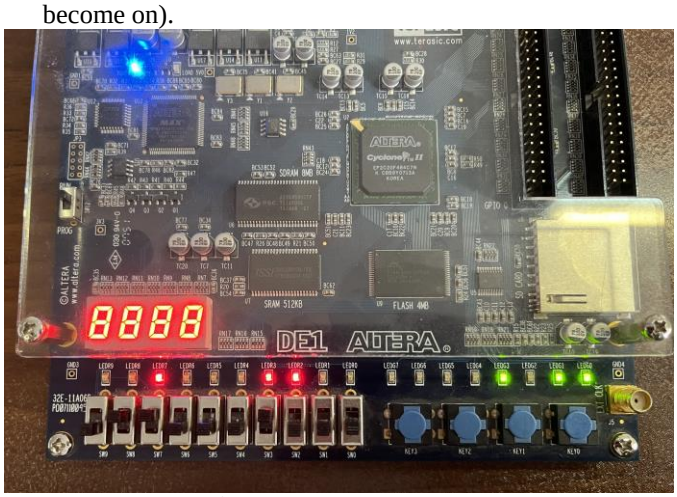


Fig. 21 First value of accelerator calculation: LED G_0 is on which means that FIFO is full. The output value is equal to $00100.01100_b = 4.375_d$.

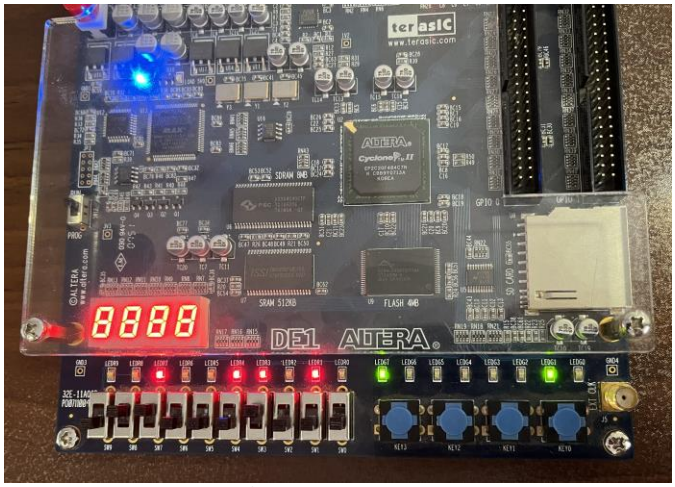


Fig. 22 Second value of accelerator calculation: the output value is equal to $0010011010_b = 4.8125_d$.

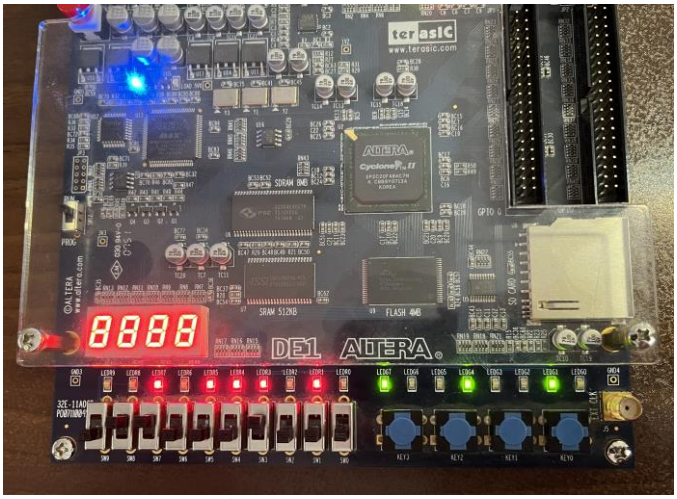


Fig. 23 Third value of accelerator calculation: the output value is equal to $00101.11010_b = 5.8125_d$.

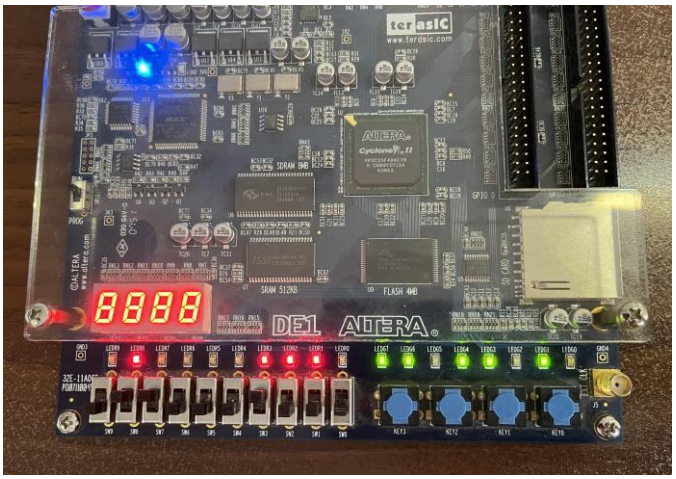


Fig. 24 Forth value of accelerator calculation: the output value is equal to $01000.01110_b = 8.4375_d$.

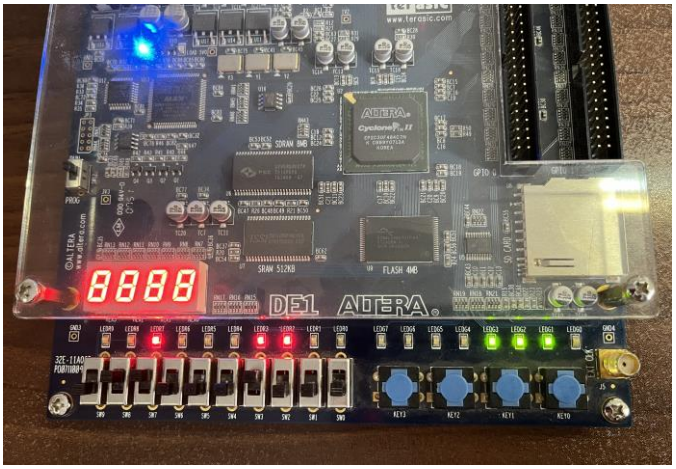


Fig. 25 Final state of accelerator: LED G_2 is on which means that FIFO is empty and all four values have been read.

V. Conclusions

In conclusion, System-on-Chip or SoC is a type of integrated circuit that combines multiple components of a computer or electronic system onto a single chip. This technology provides

several benefits, including reduced power consumption and lower costs. SoCs can also include specialized hardware accelerators that are designed to speed up performance for specific tasks.

Accelerators are used to improve performance in a wide range of computing applications, from scientific simulations and data analytics to machine learning and artificial intelligence.

A wrapper can be valuable tool for integrating an accelerator with a high frequency into a system with lower frequency CPU, helping to ensure that the system operates efficiently and effectively.

VI. Acknowledgment

This lab report was prepared and developed by Mehdi Jamalkhah and Mobina Mehrazar, bachelor students of Computer engineering at University of Tehran, under the supervision of professor Zain Navabi.