

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

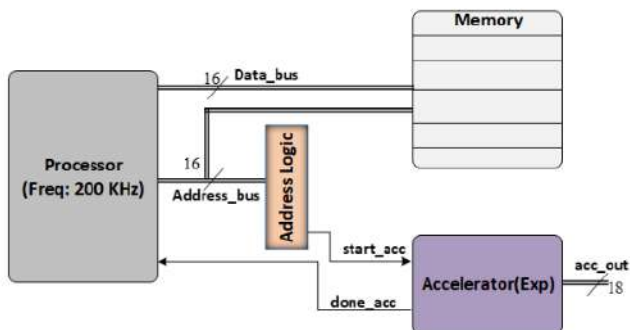


Fig. 1 Block diagram of a typical integrated circuit. Any components that is in communication with CPU talks to CPU via signals "start" and "done".

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	

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:

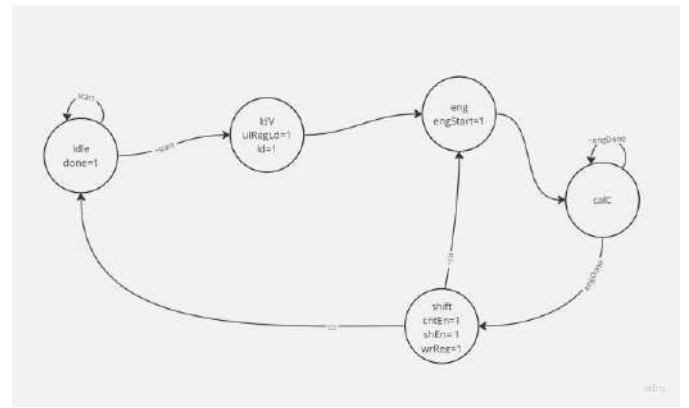


Fig. 5 Wrapper controller state diagram

2) Verilog description:

```

1 timescale 1ns/1ns
2 module ControllerWrapper(clk, rst, start, engDone, done, wrReq, shEn, ld, engStart, wrRegId);
3 input clk, start, rst, engDone;
4 output done, wrReq, shEn, ld, engStart, wrRegId;
5 reg done, wrRegId, 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, wrRegId, ld, engStart, cntEn, shEn, wrReq) = 7'b0;
38 case(ps)
39 Idle: done = 1'b1;
40 ldV: (wrRegId, ld) = 2'b11;
41 eng: engStart = 1'b1;
42 calc: ;
43 shift: (cntEn, shEn, wrReq) = 3'b111;
44 default: (done, wrRegId, ld, engStart, cntEn, shEn, wrReq) = 7'b0;
45 endcase
46 end
47

```

Fig. 6 controller of wrapper Verilog code

B. Verilog description of wrapper:

```

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
8 always @(posedge clk, posedge rst) begin
9 if (rst) begin
10 data = 16'b0;
11 end
12 else if (ld) begin
13 data = dataIn;
14 end
15 else if (shEn) begin
16 data = data << 1;
17 end
18 end
19 assign dataOut = data;
20 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, uiRegId, ui, uiOut);
3     input clk, uiRegId;
4     input [1:0] ui;
5     output [1:0] uiOut;
6     reg [1:0] data;
7     always @(posedge clk) begin
8         if (uiRegId) begin
9             data = ui;
10        end
11    end
12    assign uiOut = data;
13 endmodule
14
15

```

Fig. 9 u_i register Verilog code

```

Ln#
1 timescale 1ns/1ns
2 module DatapathWrapper(clk, rst, id, start, uiRegId, shEn, ui, fracInput, done, vData);
3     input clk, rst, id, start, uiRegId, shEn;
4     input [4:0] fracInput;
5     input [1:0] ui;
6     output done;
7     output [20:0] vData;
8     wire [1:0] uiOut;
9     wire [15:0] exp;
10    wire [15:0] exp;
11    wire [15:0] exp;
12    shiftReg shiftReg (.clk(clk), .shEn(shEn), .id(id), .rst(rst), .dataIn({3'b0, fracInput, 0'b0}), .dataOut(exp));
13    shiftComb andComb (.shiftComb(uiOut), .dataIn(intpart, fracPart), .dataOut(vData));
14    assign shg (.clk(clk), .uiRegId(uiRegId), .id(ui), .uiOut(uiOut));
15    expomental exp (.clk(clk), .rst(rst), .start(start), .shEn(shEn), .done(done), .intpart(intpart), .fracpart(fracPart));
16 endmodule
17

```

Fig. 10 Wrapper datapath Verilog code

```

Ln#
1 timescale 1ns/1ns
2 module Accelerator(clk, rst, vStart, ui, vi, done, vData, vReq);
3     input clk, rst, vStart, rst;
4     input [1:0] ui;
5     input [4:0] vi;
6     output done;
7     output [20:0] vData;
8     output [20:0] vData;
9     output [20:0] vData;
10    wire [1:0] uiOut;
11    wire [1:0] uiOut;
12    wire [1:0] uiOut;
13    ControlWrapper controlWrapper (.clk(clk), .rst(rst), .start(vStart), .shDone(shDone), .uiRegId(uiRegId));
14    DatapathWrapper datapathWrapper (.clk(clk), .rst(rst), .id(id), .start(vStart), .uiRegId(uiRegId), .shEn(shEn), .ui(ui), .fracInput(vi), .done(done), .vData(vData));
15 endmodule
16
17

```

Fig. 11 accelerator Verilog code

C. Simulation result of wrapper in Modelsim:

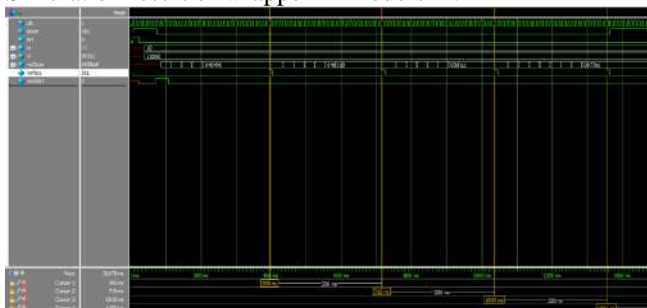


Fig. 12 simulation result part1

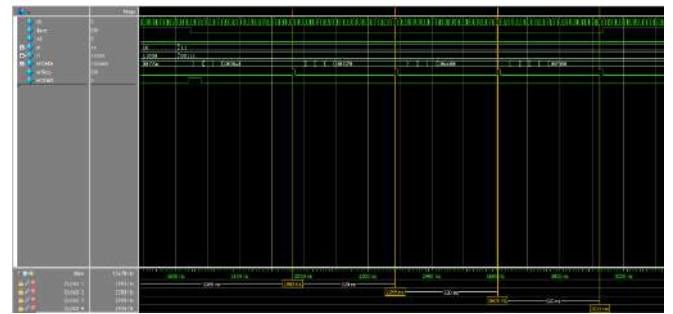


Fig. 13 simulation result part2

$$x = z + v = 2 + 0.0937$$

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

$$e_{expected}^{x_1} = 8 \cdot 115290514356445$$

$$e_{achieved}^{x_1} = 4 \cdot 392822265625$$

$$e_{expected}^{x_2} = 8 \cdot 912902981198736944925$$

$$e_{achieved}^{x_2} = 4 \cdot 8245849609375$$

$$e_{expected}^{x_3} = 10 \cdot 75101318607$$

$$e_{achieved}^{x_3} = 5 \cdot 81951904296875$$

$$e_{expected}^{x_4} = 15 \cdot 6426318841$$

$$e_{achieved}^{x_4} = 8 \cdot 46746826171875$$

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	
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	EP4CE6E22CG
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 %)

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

IO Bank	IO Action	Location	IO Bank	IO Bank	IO Bank	IO Bank	IO Bank	IO Bank	IO Bank
IO0	IO0_0	IO0_0	IO0	IO0_0	IO0_0	IO0	IO0_0	IO0_0	IO0_0
IO0	IO0_1	IO0_1	IO0	IO0_1	IO0_1	IO0	IO0_1	IO0_1	IO0_1
IO0	IO0_2	IO0_2	IO0	IO0_2	IO0_2	IO0	IO0_2	IO0_2	IO0_2
IO0	IO0_3	IO0_3	IO0	IO0_3	IO0_3	IO0	IO0_3	IO0_3	IO0_3
IO0	IO0_4	IO0_4	IO0	IO0_4	IO0_4	IO0	IO0_4	IO0_4	IO0_4
IO0	IO0_5	IO0_5	IO0	IO0_5	IO0_5	IO0	IO0_5	IO0_5	IO0_5
IO0	IO0_6	IO0_6	IO0	IO0_6	IO0_6	IO0	IO0_6	IO0_6	IO0_6
IO0	IO0_7	IO0_7	IO0	IO0_7	IO0_7	IO0	IO0_7	IO0_7	IO0_7
IO0	IO0_8	IO0_8	IO0	IO0_8	IO0_8	IO0	IO0_8	IO0_8	IO0_8
IO0	IO0_9	IO0_9	IO0	IO0_9	IO0_9	IO0	IO0_9	IO0_9	IO0_9
IO0	IO0_10	IO0_10	IO0	IO0_10	IO0_10	IO0	IO0_10	IO0_10	IO0_10
IO0	IO0_11	IO0_11	IO0	IO0_11	IO0_11	IO0	IO0_11	IO0_11	IO0_11
IO0	IO0_12	IO0_12	IO0	IO0_12	IO0_12	IO0	IO0_12	IO0_12	IO0_12
IO0	IO0_13	IO0_13	IO0	IO0_13	IO0_13	IO0	IO0_13	IO0_13	IO0_13
IO0	IO0_14	IO0_14	IO0	IO0_14	IO0_14	IO0	IO0_14	IO0_14	IO0_14
IO0	IO0_15	IO0_15	IO0	IO0_15	IO0_15	IO0	IO0_15	IO0_15	IO0_15
IO0	IO0_16	IO0_16	IO0	IO0_16	IO0_16	IO0	IO0_16	IO0_16	IO0_16
IO0	IO0_17	IO0_17	IO0	IO0_17	IO0_17	IO0	IO0_17	IO0_17	IO0_17
IO0	IO0_18	IO0_18	IO0	IO0_18	IO0_18	IO0	IO0_18	IO0_18	IO0_18
IO0	IO0_19	IO0_19	IO0	IO0_19	IO0_19	IO0	IO0_19	IO0_19	IO0_19
IO0	IO0_20	IO0_20	IO0	IO0_20	IO0_20	IO0	IO0_20	IO0_20	IO0_20
IO0	IO0_21	IO0_21	IO0	IO0_21	IO0_21	IO0	IO0_21	IO0_21	IO0_21
IO0	IO0_22	IO0_22	IO0	IO0_22	IO0_22	IO0	IO0_22	IO0_22	IO0_22
IO0	IO0_23	IO0_23	IO0	IO0_23	IO0_23	IO0	IO0_23	IO0_23	IO0_23
IO0	IO0_24	IO0_24	IO0	IO0_24	IO0_24	IO0	IO0_24	IO0_24	IO0_24
IO0	IO0_25	IO0_25	IO0	IO0_25	IO0_25	IO0	IO0_25	IO0_25	IO0_25
IO0	IO0_26	IO0_26	IO0	IO0_26	IO0_26	IO0	IO0_26	IO0_26	IO0_26
IO0	IO0_27	IO0_27	IO0	IO0_27	IO0_27	IO0	IO0_27	IO0_27	IO0_27
IO0	IO0_28	IO0_28	IO0	IO0_28	IO0_28	IO0	IO0_28	IO0_28	IO0_28
IO0	IO0_29	IO0_29	IO0	IO0_29	IO0_29	IO0	IO0_29	IO0_29	IO0_29
IO0	IO0_30	IO0_30	IO0	IO0_30	IO0_30	IO0	IO0_30	IO0_30	IO0_30
IO0	IO0_31	IO0_31	IO0	IO0_31	IO0_31	IO0	IO0_31	IO0_31	IO0_31
IO0	IO0_32	IO0_32	IO0	IO0_32	IO0_32	IO0	IO0_32	IO0_32	IO0_32
IO0	IO0_33	IO0_33	IO0	IO0_33	IO0_33	IO0	IO0_33	IO0_33	IO0_33
IO0	IO0_34	IO0_34	IO0	IO0_34	IO0_34	IO0	IO0_34	IO0_34	IO0_34
IO0	IO0_35	IO0_35	IO0	IO0_35	IO0_35	IO0	IO0_35	IO0_35	IO0_35
IO0	IO0_36	IO0_36	IO0	IO0_36	IO0_36	IO0	IO0_36	IO0_36	IO0_36
IO0	IO0_37	IO0_37	IO0	IO0_37	IO0_37	IO0	IO0_37	IO0_37	IO0_37
IO0	IO0_38	IO0_38	IO0	IO0_38	IO0_38	IO0	IO0_38	IO0_38	IO0_38
IO0	IO0_39	IO0_39	IO0	IO0_39	IO0_39	IO0	IO0_39	IO0_39	IO0_39
IO0	IO0_40	IO0_40	IO0	IO0_40	IO0_40	IO0	IO0_40	IO0_40	IO0_40
IO0	IO0_41	IO0_41	IO0	IO0_41	IO0_41	IO0	IO0_41	IO0_41	IO0_41
IO0	IO0_42	IO0_42	IO0	IO0_42	IO0_42	IO0	IO0_42	IO0_42	IO0_42
IO0	IO0_43	IO0_43	IO0	IO0_43	IO0_43	IO0	IO0_43	IO0_43	IO0_43
IO0	IO0_44	IO0_44	IO0	IO0_44	IO0_44	IO0	IO0_44	IO0_44	IO0_44
IO0	IO0_45	IO0_45	IO0	IO0_45	IO0_45	IO0	IO0_45	IO0_45	IO0_45
IO0	IO0_46	IO0_46	IO0	IO0_46	IO0_46	IO0	IO0_46	IO0_46	IO0_46
IO0	IO0_47	IO0_47	IO0	IO0_47	IO0_47	IO0	IO0_47	IO0_47	IO0_47
IO0	IO0_48	IO0_48	IO0	IO0_48	IO0_48	IO0	IO0_48	IO0_48	IO0_48
IO0	IO0_49	IO0_49	IO0	IO0_49	IO0_49	IO0	IO0_49	IO0_49	IO0_49
IO0	IO0_50	IO0_50	IO0	IO0_50	IO0_50	IO0	IO0_50	IO0_50	IO0_50
IO0	IO0_51	IO0_51	IO0	IO0_51	IO0_51	IO0	IO0_51	IO0_51	IO0_51
IO0	IO0_52	IO0_52	IO0	IO0_52	IO0_52	IO0	IO0_52	IO0_52	IO0_52
IO0	IO0_53	IO0_53	IO0	IO0_53	IO0_53	IO0	IO0_53	IO0_53	IO0_53
IO0	IO0_54	IO0_54	IO0	IO0_54	IO0_54	IO0	IO0_54	IO0_54	IO0_54
IO0	IO0_55	IO0_55	IO0	IO0_55	IO0_55	IO0	IO0_55	IO0_55	IO0_55
IO0	IO0_56	IO0_56	IO0	IO0_56	IO0_56	IO0	IO0_56	IO0_56	IO0_56
IO0	IO0_57	IO0_57	IO0	IO0_57	IO0_57	IO0	IO0_57	IO0_57	IO0_57
IO0	IO0_58	IO0_58	IO0	IO0_58	IO0_58	IO0	IO0_58	IO0_58	IO0_58
IO0	IO0_59	IO0_59	IO0	IO0_59	IO0_59	IO0	IO0_59	IO0_59	IO0_59
IO0	IO0_60	IO0_60	IO0	IO0_60	IO0_60	IO0	IO0_60	IO0_60	IO0_60
IO0	IO0_61	IO0_61	IO0	IO0_61	IO0_61	IO0	IO0_61	IO0_61	IO0_61
IO0	IO0_62	IO0_62	IO0	IO0_62	IO0_62	IO0	IO0_62	IO0_62	IO0_62
IO0	IO0_63	IO0_63	IO0	IO0_63	IO0_63	IO0	IO0_63	IO0_63	IO0_63
IO0	IO0_64	IO0_64	IO0	IO0_64	IO0_64	IO0	IO0_64	IO0_64	IO0_64
IO0	IO0_65	IO0_65	IO0	IO0_65	IO0_65	IO0	IO0_65	IO0_65	IO0_65
IO0	IO0_66	IO0_66	IO0	IO0_66	IO0_66	IO0	IO0_66	IO0_66	IO0_66
IO0	IO0_67	IO0_67	IO0	IO0_67	IO0_67	IO0	IO0_67	IO0_67	IO0_67
IO0	IO0_68	IO0_68	IO0	IO0_68	IO0_68	IO0	IO0_68	IO0_68	IO0_68
IO0	IO0_69	IO0_69	IO0	IO0_69	IO0_69	IO0	IO0_69	IO0_69	IO0_69
IO0	IO0_70	IO0_70	IO0	IO0_70	IO0_70	IO0	IO0_70	IO0_70	IO0_70
IO0	IO0_71	IO0_71	IO0	IO0_71	IO0_71	IO0	IO0_71	IO0_71	IO0_71
IO0	IO0_72	IO0_72	IO0	IO0_72	IO0_72	IO0	IO0_72	IO0_72	IO0_72
IO0	IO0_73	IO0_73	IO0	IO0_73	IO0_73	IO0	IO0_73	IO0_73	IO0_73
IO0	IO0_74	IO0_74	IO0	IO0_74	IO0_74	IO0	IO0_74	IO0_74	IO0_74
IO0	IO0_75	IO0_75	IO0	IO0_75	IO0_75	IO0	IO0_75	IO0_75	IO0_75
IO0	IO0_76	IO0_76	IO0	IO0_76	IO0_76	IO0	IO0_76	IO0_76	IO0_76
IO0	IO0_77	IO0_77	IO0	IO0_77	IO0_77	IO0	IO0_77	IO0_77	IO0_77
IO0	IO0_78	IO0_78	IO0	IO0_78	IO0_78	IO0	IO0_78	IO0_78	IO0_78
IO0	IO0_79	IO0_79	IO0	IO0_79	IO0_79	IO0	IO0_79	IO0_79	IO0_79
IO0	IO0_80	IO0_80	IO0	IO0_80	IO0_80	IO0	IO0_80	IO0_80	IO0_80
IO0	IO0_81	IO0_81	IO0	IO0_81	IO0_81	IO0	IO0_81	IO0_81	IO0_81
IO0	IO0_82	IO0_82	IO0	IO0_82	IO0_82	IO0	IO0_82	IO0_82	IO0_82
IO0	IO0_83	IO0_83	IO0	IO0_83	IO0_83	IO0	IO0_83	IO0_83	IO0_83
IO0	IO0_84	IO0_84	IO0	IO0_84	IO0_84	IO0	IO0_84	IO0_84	IO0_84
IO0	IO0_85	IO0_85	IO0	IO0_85	IO0_85	IO0	IO0_85	IO0_85	IO0_85
IO0	IO0_86	IO0_86	IO0	IO0_86	IO0_86	IO0	IO0_86	IO0_86	IO0_86
IO0	IO0_87	IO0_87	IO0	IO0_87	IO0_87	IO0	IO0_87	IO0_87	IO0_87
IO0	IO0_88	IO0_88	IO0	IO0_88	IO0_88	IO0	IO0_88	IO0_88	IO0_88
IO0	IO0_89	IO0_89	IO0	IO0_89	IO0_89	IO0	IO0_89	IO0_89	IO0_89
IO0	IO0_90	IO0_90	IO0	IO0_90	IO0_90	IO0	IO0_90	IO0_90	IO0_90
IO0	IO0_91	IO0_91	IO0	IO0_91	IO0_91	IO0	IO0_91	IO0_91	IO0_91
IO0	IO0_92	IO0_92	IO0	IO0_92	IO0_92	IO0	IO0_92	IO0_92	IO0_92
IO0	IO0_93	IO0_93	IO0	IO0_93	IO0_93	IO0	IO0_93	IO0_93	IO0_93
IO0	IO0_94	IO0_94	IO0	IO0_94	IO0_94	IO0	IO0_94	IO0_94	IO0_94
IO0	IO0_95	IO0_95	IO0	IO0_95	IO0_95	IO0	IO0_95	IO0_95	IO0_95
IO0	IO0_96	IO0_96	IO0	IO0_96	IO0_96	IO0	IO0_96	IO0_96	IO0_96
IO0	IO0_97	IO0_97	IO0	IO0_97	IO0_97	IO0	IO0_97	IO0_97	IO0_97
IO0	IO0_98	IO0_98	IO0	IO0_98	IO0_98	IO0	IO0_98	IO0_98	IO0_98
IO0	IO0_99	IO0_99	IO0	IO0_99	IO0_99	IO0	IO0_99	IO0_99	IO0_99
IO0	IO0_100	IO0_100	IO0	IO0_100	IO0_100	IO0	IO0_100	IO0_100	IO0_100
IO0	IO0_101	IO0_101	IO0	IO0_101	IO0_101	IO0	IO0_101	IO0_101	IO0_101
IO0	IO0_102	IO0_102	IO0	IO0_102	IO0_102	IO0	IO0_102	IO0_102	IO0_102
IO0	IO0_103	IO0_103	IO0	IO0_103	IO0_103	IO0	IO0_103	IO0_103	IO0_103
IO0	IO0_104	IO0_104	IO0	IO0_104	IO0_104	IO0	IO0_104	IO0_104	IO0_104
IO0	IO0_105	IO0_105	IO0	IO0_105	IO0_105	IO0	IO0_105	IO0_105	IO0_105
IO0	IO0_106	IO0_106	IO0	IO0_106	IO0_106	IO0	IO0_106	IO0_106	IO0_106
IO0	IO0_107	IO0_107	IO0	IO0_107	IO0_107	IO0	IO0_107	IO0_107	IO0_107
IO0	IO0_108	IO0_108	IO0	IO0_108	IO0_108	IO0	IO0_108	IO0_108	IO0_108
IO0	IO0_109	IO0_109	IO0	IO0_109	IO0_109	IO0	IO0_109	IO0_109	IO0_109
IO0	IO0_110	IO0_110	IO0	IO0_110	IO0_110	IO0	IO0_110	IO0_110	IO0_110
IO0	IO0_111	IO0_111	IO0	IO0_111	IO0_111	IO0	IO0_111	IO0_111	IO0_111
IO0	IO0_112	IO0_112	IO0	IO0_112	IO0_112	IO0	IO0_112	IO0_112	IO0_112
IO0	IO0_113	IO0_113	IO0	IO0_113	IO0_113	IO0	IO0_113	IO0_113	IO0_113
IO0	IO0_114	IO0_114	IO0	IO0_114	IO0_114	IO0	IO0_114	IO0_114	IO0_114
IO0	IO0_115	IO0_115	IO0	IO0_115	IO0_115	IO0	IO0_115	IO0_115	IO0_115
IO0	IO0_116	IO0_116	IO0	IO0_116	IO0_116	IO0	IO0_116	IO0_116	IO0_116
IO0	IO0_117	IO0_117	IO0	IO0_117	IO0_117	IO0	IO0_117	IO0_117	IO0_117
IO0	IO0_118	IO0_118	IO0	IO0_118	IO0_118	IO0	IO0_118	IO0_118	IO0_118
IO0	IO0_119	IO0_119	IO0	IO0_119	IO0_119	IO0	IO0_119	IO0_119	IO0_119
IO0	IO0_120	IO0_120	IO0	IO0_120	IO0_120	IO0	IO0_120	IO0_120	IO0_120
IO0	IO0_121	IO0_121	IO0	IO0_121	IO0_121	IO0	IO0_121	IO0_121	IO0_121
IO0	IO0_122	IO0_122	IO0	IO0_122	IO0_122	IO0	IO0_122	IO0_122	IO0_122
IO0	IO0_123	IO0_123	IO0	IO0_123	IO0_123	IO0	IO0_123	IO0_123	IO0_123
IO0	IO0_124	IO0_124	IO0	IO0_124	IO0_124	IO0	IO0_124	IO0_124	IO0_124
IO0	IO0_125	IO0_125	IO0	IO0_125	IO0_125	IO0	IO0_125	IO0_125	IO0_125
IO0	IO0_126	IO0_126	IO0	IO0_126	IO0_126	IO0	IO0_126	IO0_126	IO0_126

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).

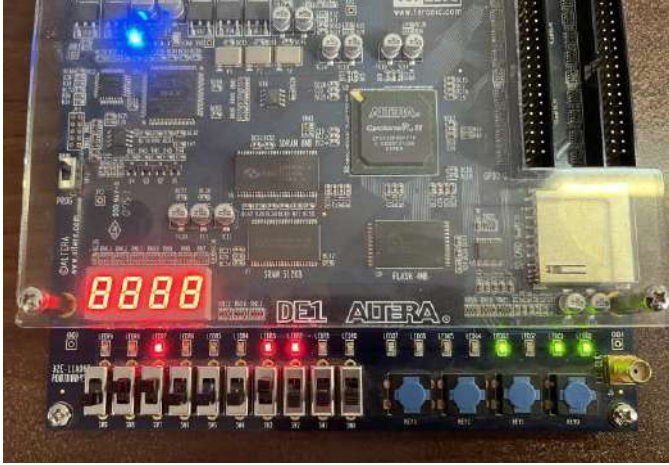


Fig. 21 First value of accelerator calculation: LED G_0 is on which means that FIFO is full. The output value is equal to $00100.01100_b = 4.375_d$.

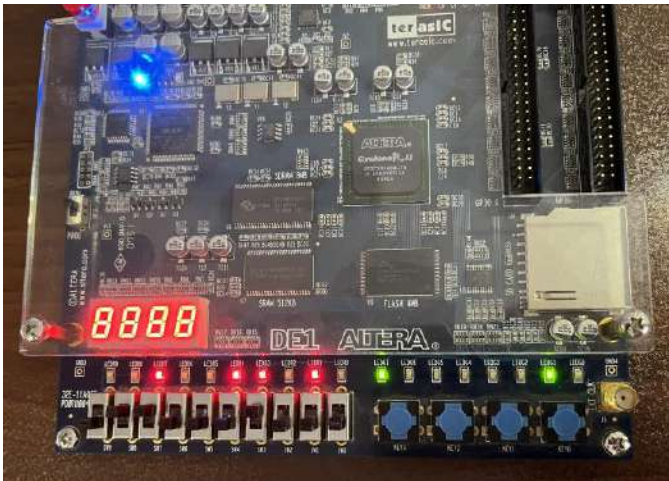


Fig. 22 Second value of accelerator calculation: the output value is equal to $0010011010_b = 4.8125_d$.

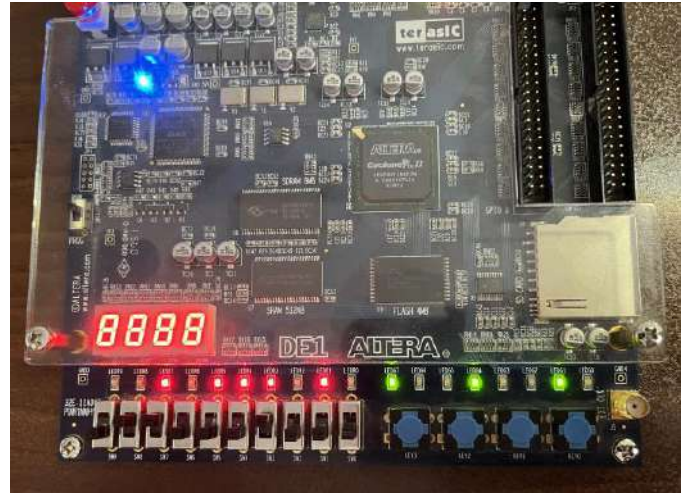


Fig. 23 Third value of accelerator calculation: the output value is equal to $00101.11010_b = 5.8125_d$.

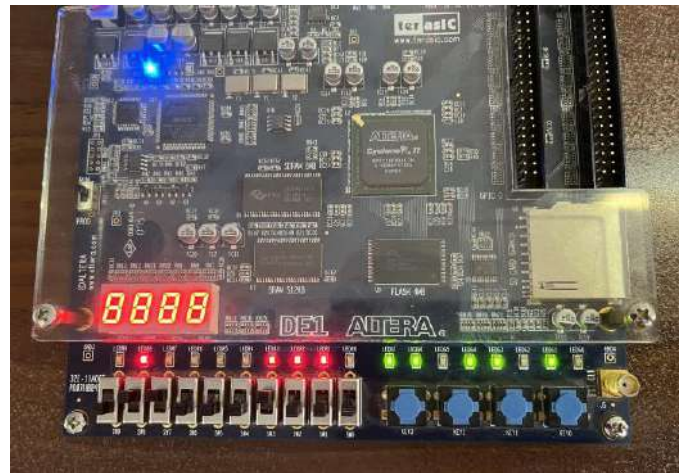


Fig. 24 Forth value of accelerator calculation: the output value is equal to $01000.01110_b = 8.4375_d$.

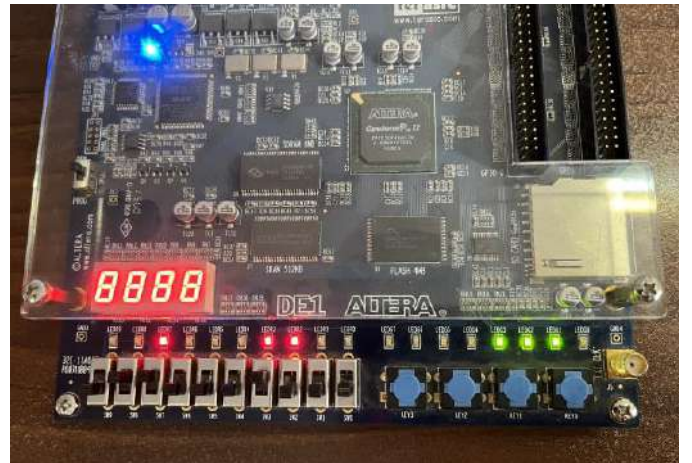


Fig. 25 Final state of accelerator: LED G_2 is on which means that FIFO is empty and all four values have been read.

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.