

Experiment 4 - Accelerator and Wrappers

Pasha Barahimi, 810199385

Misagh Mohaghegh, 810199484

Abstract— This document is a student report for the 4th experiment of the Digital Logic Laboratory course (ECE 045) at University of Tehran, Department of Electrical and Computer Engineering.

Keywords— Exponential Engine, Accelerator Buffer, Accelerator Wrapper, FPGA Implementation, Altera Cyclone II, Quartus, ModelSim

I. EXPONENTIAL ENGINE

A. Simulation

The following test-bench was written for the purpose of testing the provided exponential engine.

Three different values are given to the engine's input and the output is calculated and verified.

```
1 ~timescale 1ns/1ns
2
3 module ExpEng1BQ;
4     reg start, clk, rst;
5     reg [15:0] inp;
6
7     wire done;
8     wire [15:0] intpart;
9     wire [15:0] fracpart;
10
11     exponential exp(.x(inp), .start(start), .clk(clk), .rst(rst), .done(done), .intpart(intpart), .fracpart(fracpart));
12
13     always #5 clk = ~clk;
14
15     initial begin
16         #0 clk = 0; rst = 0; start = 0;
17         #3 rst = 1;
18         #3 rst = 0;
19
20         #10 inp = 16'h8000; start = 1;
21         #10 start = 0;
22         while (~done) #10;
23         #100;
24
25         #10 inp = 16'hCCCC; start = 1;
26         #10 start = 0;
27         while (~done) #10;
28         #100;
29
30         #10 inp = 16'h3333; start = 1;
31         #10 start = 0;
32         while (~done) #10;
33         #100;
34
35         #100 $stop;
36     end
37 endmodule
```

Fig. 1 Verilog test-bench for the exponential engine

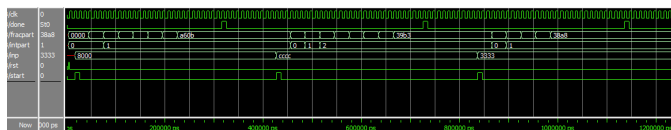


Fig. 2 Simulation results

The results are very close to the expected output:

Input 0x8000: 0x1A5E3

Input 0xCCCC: 0x233F7

Input 0x3333: 0x13581

The difference is due to the exponential engine's accuracy.

B. Synthesis

1) Results:

Flow Summary	
Flow Status	Successful - Tue May 17 05:52:58 2022
Quartus II 64-Bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	ExpAcc
Top-level Entity Name	ExpEngine
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	102 / 18,752 (< 1 %)
Total combinational functions	100 / 18,752 (< 1 %)
Dedicated logic registers	61 / 18,752 (< 1 %)
Total registers	61
Total pins	38 / 315 (12 %)
Total virtual pins	0
Total memory bits	0 / 239,616 (0 %)
Embedded Multiplier 9-bit elements	2 / 52 (4 %)
Total PLLs	0 / 4 (0 %)

Fig. 3 Synthesis results

2) Maximum Frequency:

Slow Model Fmax Summary			
	Fmax	Restricted Fmax	Clock Name
1	113.22 MHz	113.22 MHz	clk

Fig. 4 Maximum frequency

II. EXPONENTIAL ENGINE WRAPPER

A. Buffers

1) ROM:

The following .mif file is used for the initialization of the input read-only memory:

```

1  WIDTH=16;
2  DEPTH=8;
3
4  ADDRESS_RADIX=HEX;
5  DATA_RADIX=HEX;
6
7  CONTENT BEGIN
8
9  00: 8000; -- e^0.50 = 1.648 = 0x1A5E3
10 01: CCCC; -- e^0.79 = 2.203 = 0x233F7
11 02: 3333; -- e^0.19 = 1.209 = 0x13581
12 03: FD70; -- e^0.99 = 2.691 = 0x2B0E5
13 04: 0000; -- e^0.00 = 1.000 = 0x10000
14 05: 0000;
15 06: 0000;
16 07: 0000;
17
18 END;

```

Fig. 5 ROM .mif file

The first 5 memory locations are filled with different values to test the functionality of the exponential engine.

The calculation result is also commented for verification.

2) FIFO:

A FIFO queue is used for the output buffer.

After every read signal from the user, its rdreq signal is issued which will dequeue the first input's result.

B. Controller

1) State Diagram:

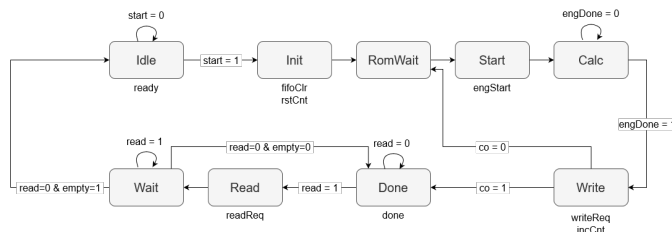


Fig. 6 Wrapper controller state diagram

2) Behavioral Description:

```

1 `define Idle 4'b0000
2 `define Init 4'b0001
3 `define RomWait 4'b0010
4 `define Start 4'b0011
5 `define Calc 4'b0100
6 `define Write 4'b0101
7 `define Done 4'b0110
8 `define Read 4'b0111
9 `define Wait 4'b1000
10
11 module WrapperController(
12     input start, engDone, coCnt, read, empty, clk, rst,
13     output reg done, ready, engStart, rstCnt, incCnt, writeReq, readReq, fifoClr
14 );
15     reg [3:0] ps, ns;
16
17     always @(posedge clk, posedge rst) begin
18         if (rst) ps <= `Idle;
19         else ps <= ns;
20     end
21
22     always @(ps, start, engDone, coCnt, read, empty) begin
23         case (ps)
24             `Idle: ns = start ? `Init : `Idle;
25             `Init: ns = `RomWait;
26             `RomWait: ns = `Start;
27             `Start: ns = `Calc;
28             `Calc: ns = engDone ? `Write : `Calc;
29             `Write: ns = coCnt ? `Done : `RomWait;
30             `Done: ns = read ? `Read : `Done;
31             `Read: ns = `Wait;
32             `Wait: ns = read ? `Wait : (empty ? `Idle : `Done);
33             default::;
34         endcase
35     end
36
37     always @(ps) begin
38         {done, ready, engStart, rstCnt, incCnt, writeReq, readReq, fifoClr} = 8'd0;
39         case (ps)
40             `Idle: ready = 1'b1;
41             `Init: {rstCnt, fifoClr} = 2'b11;
42             `RomWait::;
43             `Start: engStart = 1'b1;
44             `Calc::;
45             `Write: {writeReq, incCnt} = 2'b11;
46             `Done: done = 1'b1;
47             `Read: readReq = 1'b1;
48             `Wait::;
49             default::;
50         endcase
51     end
52 endmodule

```

Fig. 7 Wrapper controller Verilog description

3) Block Diagram:

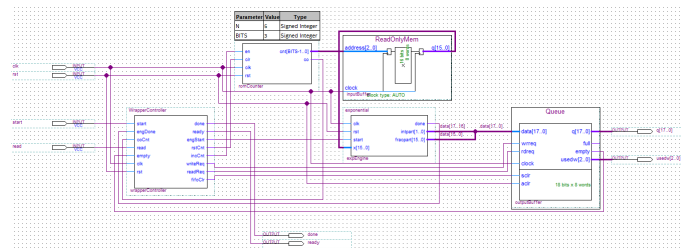


Fig. 8 The accelerator block diagram

4) Simulation:

The above accelerator was synthesized and a test-bench was written for it:

```

1 `timescale 1ns/1ns
2
3 module ExpAcc18Q;
4     reg start, read, clk, rst;
5
6     wire done, ready;
7     wire [17:0] q;
8     wire [2:0] usedw;
9
10    ExpAcc acc(.done(done), .start(start), .clk(clk), .rst(rst), .read(read), .ready(ready), .q(q), .usedw(usedw));
11
12    always #5 clk = ~clk;
13
14    initial begin
15        #0 clk = 0; rst = 0; start = 0; read = 0;
16        #3 rst = 1;
17        #3 rst = 0;
18        #10 start = 1;
19        #10 start = 0;
20        while (~done) #10;
21        #30 read = 1;
22        #30 read = 0;
23        #30 read = 1;
24        #30 read = 0;
25        #30 read = 1;
26        #30 read = 0;
27        #30 read = 1;
28        #30 read = 0;
29        #30 read = 1;
30        #30 read = 0;
31        #100 $stop;
32    end
33 endmodule

```

Fig. 9 Accelerator test-bench

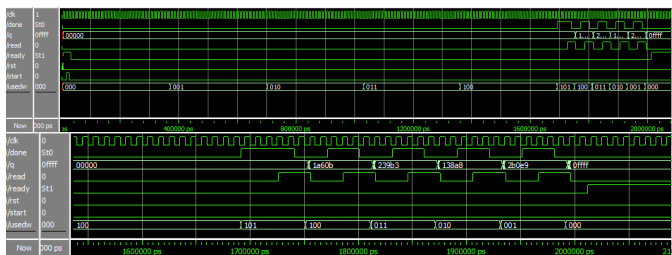


Fig. 10 Simulation results

III. FPGA IMPLEMENTATION

A. Pin Planner

The figure shows the Quartus pin planner interface. The top part displays a top-level view of the FPGA pin grid. The bottom part shows a table of pin assignments.

Node Name	Direction	Location	I/O Bank	VREF Group	Filter Location	I/O Standard	Reserved	Current Strength	Differential Pair
acc18q[0]	Output	PP1_05	2	B2_00	PP1_05	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[1]	Output	PP1_06	2	B2_00	PP1_06	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[2]	Output	PP1_C2	2	B2_00	PP1_C2	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[3]	Output	PP1_C1	2	B2_00	PP1_C1	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[4]	Output	PP1_E3	2	B2_00	PP1_E3	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[5]	Output	PP1_E4	2	B2_00	PP1_E4	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[6]	Output	PP1_D3	2	B2_00	PP1_D3	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[7]	Output	PP1_F4	2	B2_00	PP1_F4	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[8]	Output	PP1_D5	2	B2_00	PP1_D5	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[9]	Output	PP1_D6	2	B2_00	PP1_D6	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[10]	Output	PP1_J4	2	B2_01	PP1_J4	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[11]	Output	PP1_L8	2	B2_01	PP1_L8	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[12]	Output	PP1_F3	2	B2_00	PP1_F3	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[13]	Output	PP1_D4	2	B2_00	PP1_D4	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[14]	Output	PP1_H2	1	B1_00	PP1_H2	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[15]	Output	PP1_L2	2	B2_01	PP1_L2	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[16]	Output	PP1_U22	6	B6_01	PP1_U22	3.3V LV (default)	2mA (default)	2mA (default)	
acc18q[17]	Output	PP1_J2	2	B2_01	PP1_J2	3.3V LV (default)	2mA (default)	2mA (default)	
start	Input	PP1_01	1	B1_00	PP1_01	3.3V LV (default)	2mA (default)	2mA (default)	

Fig. 11 Quartus pin planner. Assigning toggle buttons to our inputs, connecting the output to the 7-segment displays, using a green LED for the *ready* signal, and a red LED for the *done* signal.

B. Implementation

The accelerator was put in a top-level module dividing and connecting its output to 4 HexDisplay modules.

1) HEX Display:

The exponential engine output is 18-bits. 2 int parts and 16 fractional parts.

The output here is shown on the 7-segment displays. The first display (leftmost) shows the int part and the rest of the displays each show 4 bit of the fractional part.

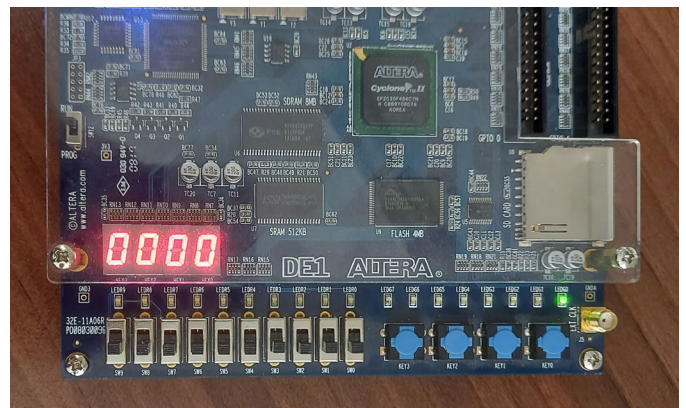


Fig. 12 Ready signal is issued

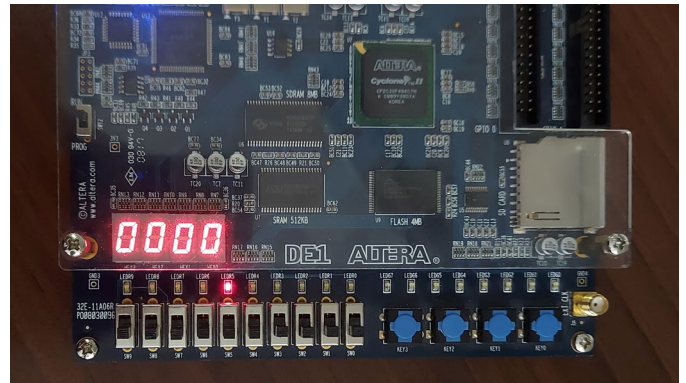


Fig. 13 After a full pulse on the start input, the done signal is issued meaning that the calculations are done

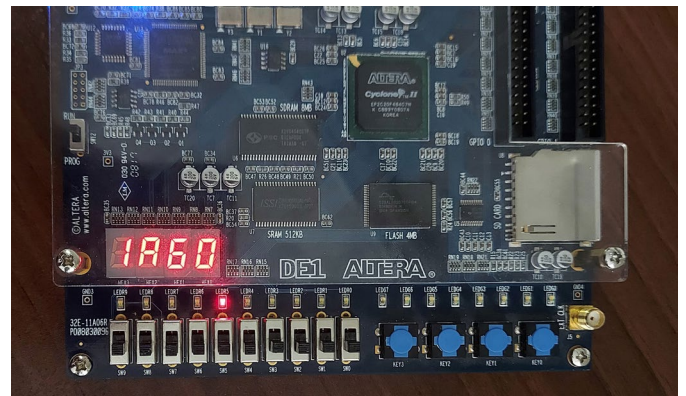


Fig. 14 Read 1

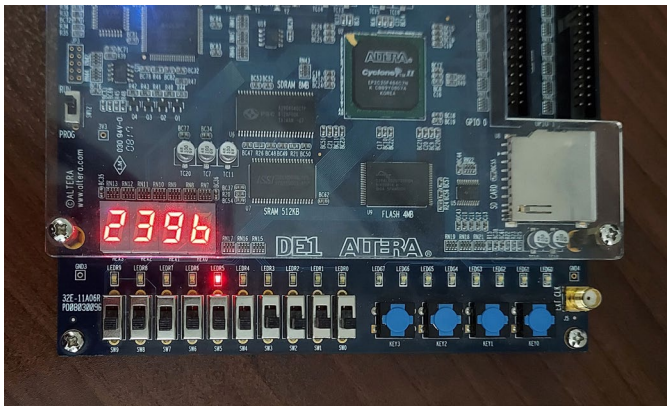


Fig. 15 Read 2

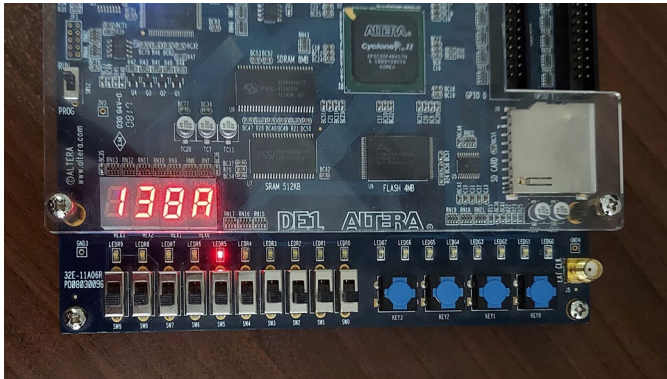


Fig. 16 Read 3

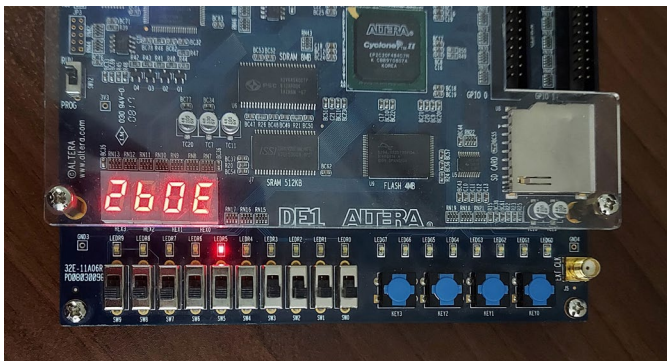


Fig. 17 Read 4

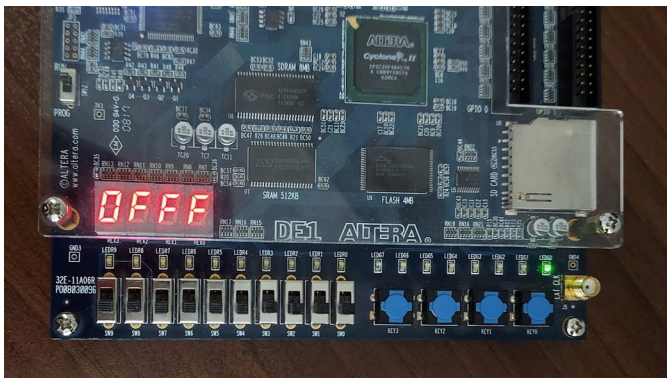


Fig. 18 Read 5. The controller is back to its idle state and the ready signal is issued again

2) Decimal Display:

Here, the output shown on the 7-segment displays is in a BCD format as follows: (considering d0 to be the leftmost display)

d0: Int part

d1: Fractional part

d2: H for separating

d3: The FIFO structure's used words

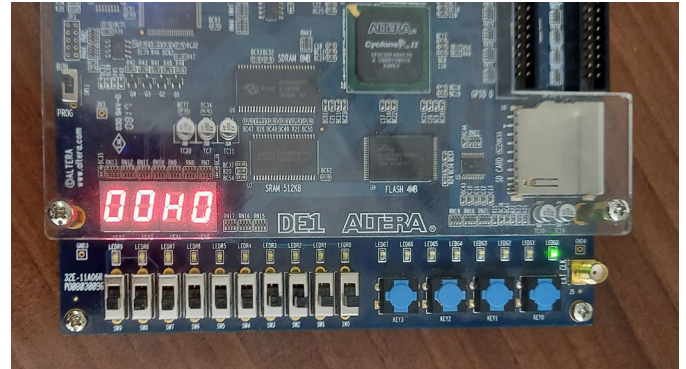


Fig. 19 Ready signal is issued

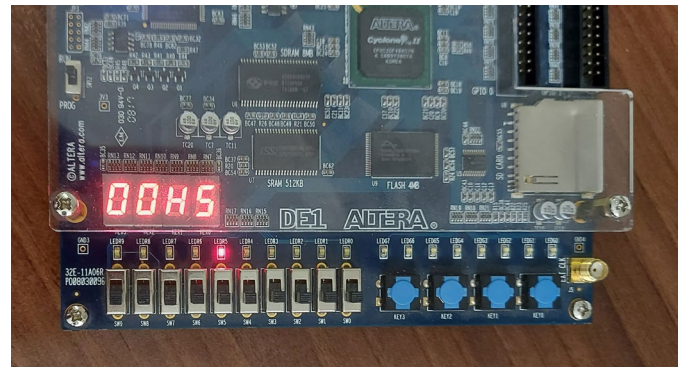


Fig. 20 After a full pulse on the start input, the done signal is issued meaning that the calculations are done. The number 5 is shown for the used words' count, which means that there are 5 entries in the output FIFO

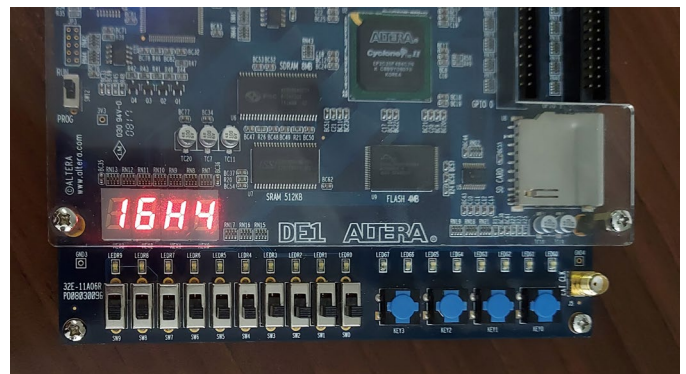


Fig. 21 Read 1. An entry from the FIFO is popped thus showing 4 used words. The first two displays are the int and fraction part of the output. 1.6 is the result here

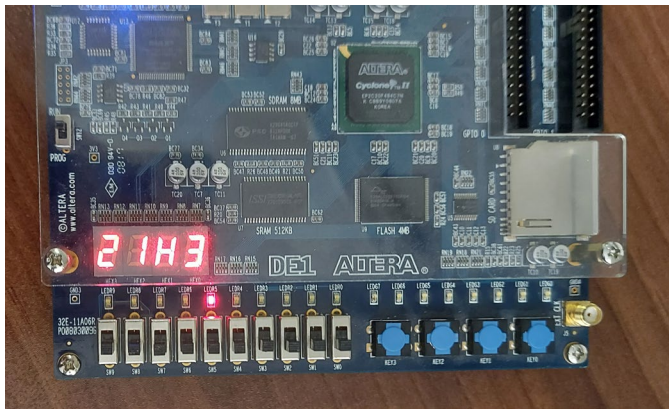


Fig. 22 Read 2

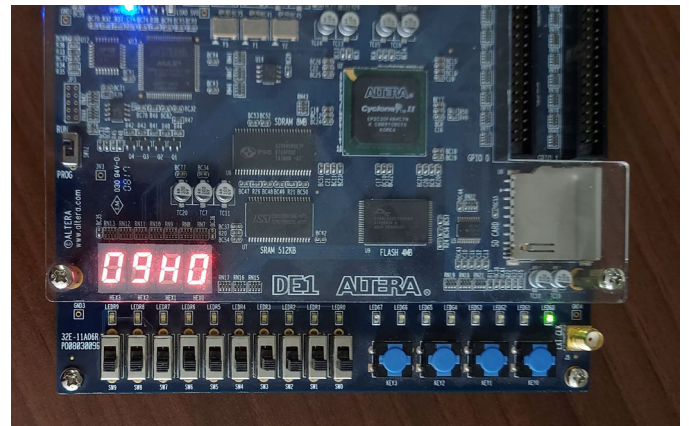


Fig. 25 Read 5. The controller is back to its idle state and the ready signal is issued again. The FIFO is now empty

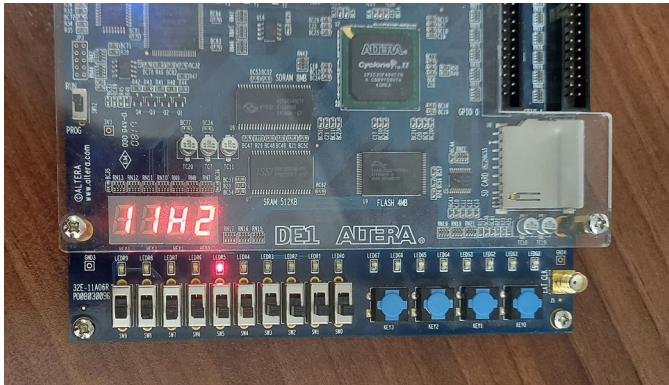


Fig. 23 Read 3

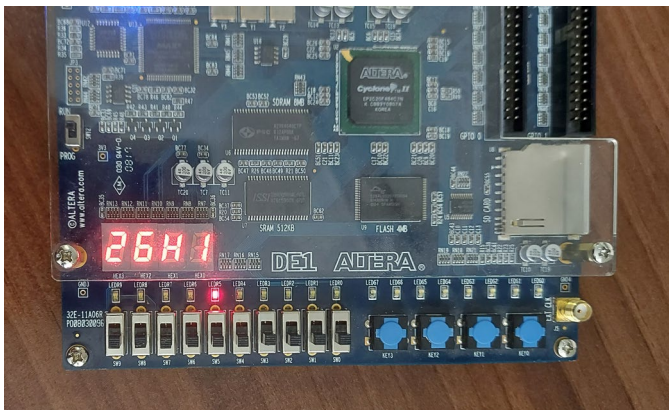


Fig. 24 Read 4