ELEC6234 – Embedded Processors

Coursework Report – picoMIPS implementation

Xuankun Cai
Xc1m22
(MSc) Internet of Things
Tomasz Kazmierski

1. Introduction

The objective of this assignment is to implement the affine transformation algorithm by specific picoMIPS application, in which presentation will be 8-bits. The implementation is expected to use the smallest possible hardware cost and will be demonstrated in FPGA broad.

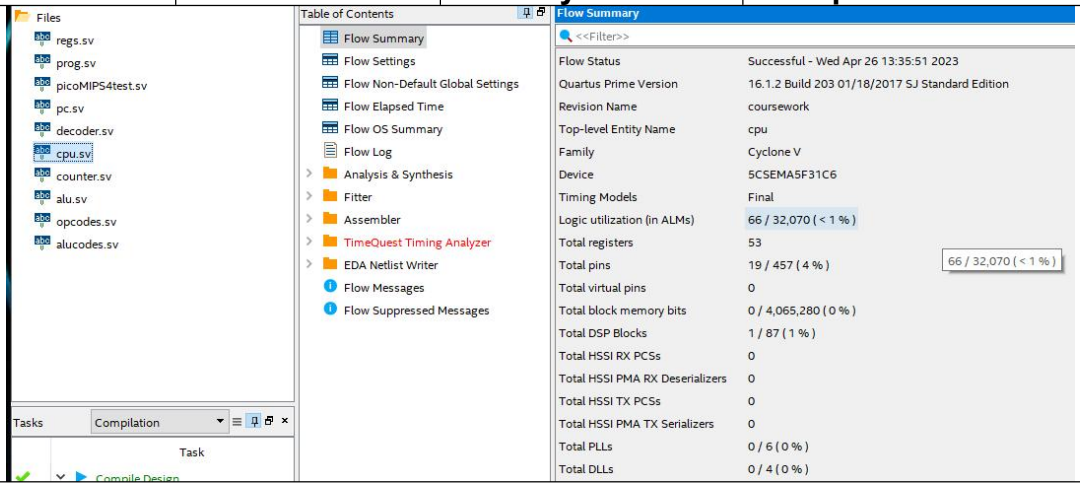
Even another structure, NISC, is allowed to use, picoMIPS is still chosen. Because, according to advantages and disadvantages of both systems, this assignment will not require such complex functions or optimization that prefer to NISC rather than picoMIPS, and picoMIPS is simpler design [1, 2].

The essential requirements have been accomplished by VHDL programme of designed system and a hexadecimal instruction set of affine transformation algorithm. According to the 8-bit structure of data, the input range is limited in -128 to 127. The output will be processed result presented by LED.

To make further optimization, the initial 24-bits instruction is reduced to 16-bits. The address of source is combined into immediate number, which represented by 8-bits. The recognition whether it's address or number is implemented by a multiplexer. Besides, the amount of instructions for implementation is simplified to reduce total cost.

Design Details Form

Total Cost: 66	ALMs: 66	Memory bits: 0	Multipliers:1



Instruction Set

There is total 10 instructions are prepared in this design. Not all of these are needed but prepared for further development.

- NOP: No operation.
- ADD: Add values from destination address and source address, the result will be written into destination address.
- ADDI: Add value from destination address and imported immediate value, the result will be written into destination address.
- SUB: Subtract values from destination address by that from source address, the result will be written into destination address.
- SUBI: Subtract value from destination address by imported immediate value, the result will be written into destination address.
- MUL: Multiple values from destination address and source address, the result will be written into destination address.
- MULI: Multiple value from destination address and imported immediate value, the result will be written into destination address.
- LDS: The operation is importing signals from switches into destination address, the ALU will not calculate and send signals directly.
- BAT: Recognize branch at designed condition by Bcon==I[7].
- SHOW: Control display of LEDs

Instruction Format

Size of instruction = 16 bits. Size of data = 8 bits.
Format: 16 bits = opcode (4 bits) + destination address (4 bits) + source address / immediate value (8 bits)

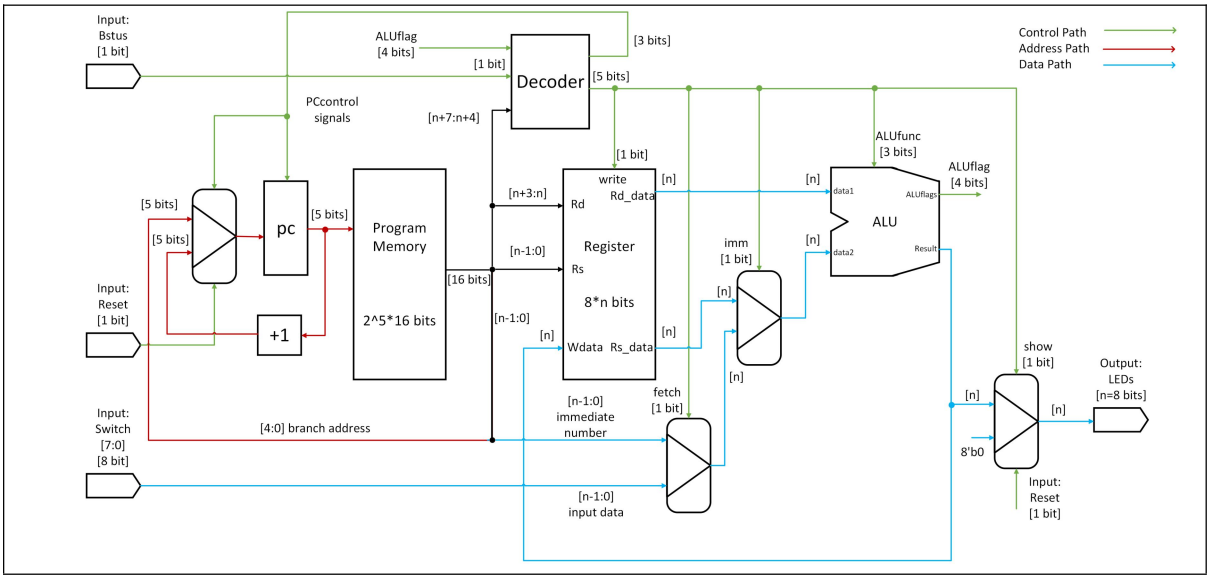
Opcode	Binary	Operation	Description
NOP	1111	No operation	
ADD	0000	ADD %d, %s	%d = %d + %s
ADDI	0001	ADDI %d, imm	%d = %d + imm
SUB	0010	SUB %d, %s	%d = %d - %s
SUBI	0011	SUBI %d, imm	%d = %d - imm
MUL	0100	MUL %d, %s	%d = %d * %s
MULI	0101	MULI %d, imm	%d = %d * imm
LDS	0110	LDS %d, imm	%d = import imm
BAT	0111	Bcon==I[7], Bstus==Bcon?	branch by condition
SHOW	1000	SHOW %d, %0	LED display control, output <= %d

Your affine transformation program

```
// HEX ////////// BINARY  ////////////////////// ASSEMBLER /////////


7080 // 16'b0111_0000_10000000    //   BAT1 %-, %-, 1;      until Bstus=0 goto next Instruction
7000 // 16'b0111_0000_00000000    //   BAT0 %-, %-, 0;      until Bstus=1 goto next Instruction
6100 // 16'b0110_0001_00000000    //   LDS  %1, %x, -;      REG 1 <= import x1
6200 // 16'b0110_0010_00000000    //   LDS  %2, %x, -;      REG 2 <= import x1
7080 // 16'b0111_0000_10000000    //   BAT1 %-, %-, 1;      until Bstus=0 goto next Instruction
7000 // 16'b0111_0000_00000000    //   BAT0 %-, %-, 0;      until Bstus=1 goto next Instruction
6300 // 16'b0110_0011_00000000    //   LDS  %3, %y, -;      REG 3 <= import y1
6400 // 16'b0110_0100_00000000    //   LDS  %4, %y, -;      REG 4 <= import y1
7080 // 16'b0111_0000_10000000    //   BAT1 %-, %-, 1;      until Bstus=0 goto next Instruction
5160 // 16'b0101_0001_01100000    //   MULI %1, %1, 0.75;   0.75x1
52C0 // 16'b0101_0010_11000000    //   MULI %2, %2, -0.5;   -0.5x1
5340 // 16'b0101_0011_01000000    //   MULI %3, %3, 0.5 ;   0.5y1
5460 // 16'b0101_0100_01100000    //   MULI %4, %4, 0.75;   0.75y1
0103 // 16'b0000_0001_00000011    //   ADD  %1, %3  -;      0.75x1 + 0.5y1
0204 // 16'b0000_0010_00000100    //   ADD  %2, %4  -;      -0.5x1 + 0.75y1
1114 // 16'b0001_0001_00010100    //   ADDI %1, %1, 20  ;   x2 = 0.75x1 + 0.5y1 + 20
7000 // 16'b0111_0000_00000000    //   BAT0 %-, %-, 0;       until Bstus=1 goto next Instruction
12EC // 16'b0001_0010_11101100    //   ADDI %2, %2, -20;     y2 = -0.5x1 + 0.75y1 - 20
7081 // 16'b0111_0000_10000001    //   BAT1 %-, %-, 2;       until Bstus=0 goto The 2 Instruction
```

Design Block Diagram



2. Overall architecture of the design and simulations

*As the RAM is not needed in this case, the used components of processor architecture are Decoder, Register, ALU module, assigned counter to transform fast clock, Program Counter and Memory for 2^5*16 bits [1]. The entire architecture is to implement affine transformation in picoMIPS. As shown in the design diagram shown above, blocks are connected. The input is configured as switches, in which Switch[8] is set as "Bstus" and Switch[9] is "reset". The equation of affine transformation is presented as follow with config of matrix A and vector B. The test simulation result of each components will be discussed in following subsections.*

$$\begin{vmatrix} x_2 \\ y_2 \end{vmatrix} = A \times \begin{vmatrix} x_1 \\ y_1 \end{vmatrix} + B$$

$$A = \begin{bmatrix} 0.75 & 0.5 \\ -0.5 & 0.75 \end{bmatrix} \qquad B = \begin{bmatrix} 20 \\ -20 \end{bmatrix}$$

Program Counter:
The program counter contains three inputs for branch control, a "PCincr" for normal step forward, and a reset control signal. PCout will send output referring to address of instruction. The Figure.

2.1 directly illustrates its step forward and a relative branch operation at 30-35 ns. Then the "reset" function work at 40 ns as expected.

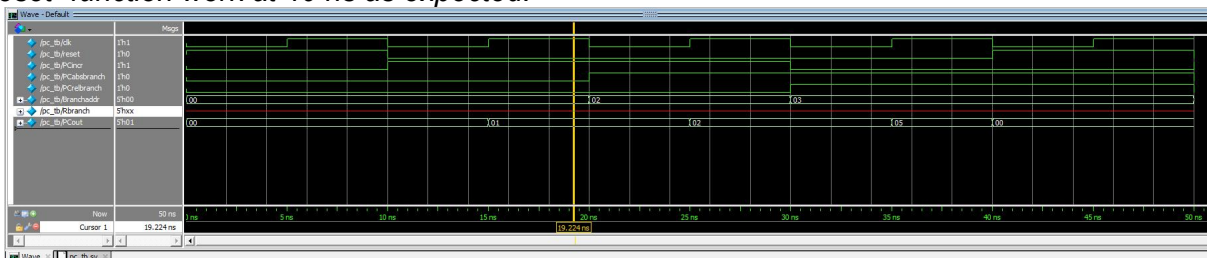


Figure. 2.1 Simulation result of Program Counter

Program Memory:
The output signal from Program Counter will be received by Program Memory and corresponding instructions will be output. The Figure. 2.2 presents all instructions in hexadecimal, which is correct. And there is no output since "address-12", because total amount of instruction is 13.

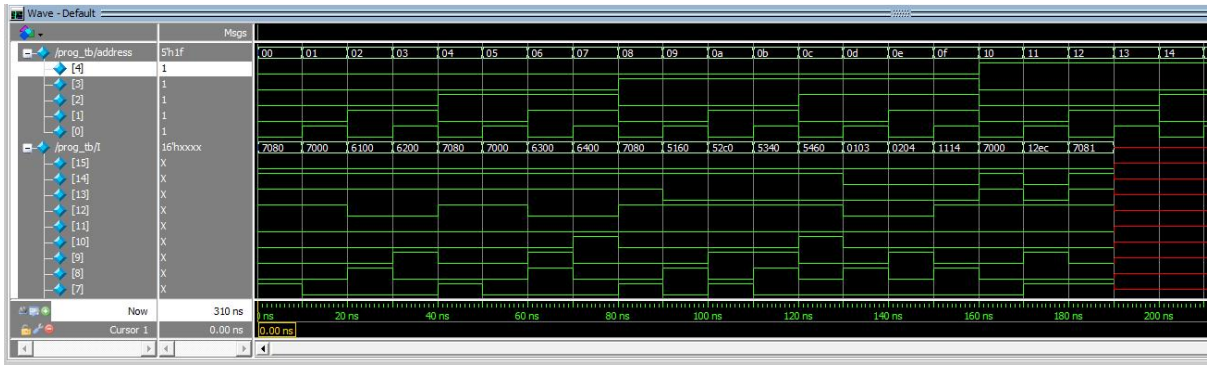


Figure. 2.2 Simulation result of Program Memory

Decoder:
Decoder will response to opcodes contained in instructions with designed control signals. The control of branch structure and some components is accomplished here. For instance, as shown between 20-30 ns in Figure. 2.3, if input opcodes is "ADDI", whose binary code is [0001], Decoder will send active signal to "imm" (a multiplexer) to extract immediate value and to Register for writing data in.

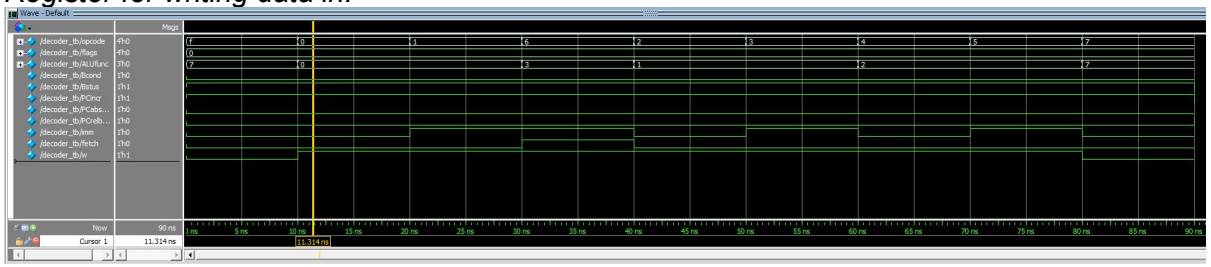


Figure. 2.3 Simulation result of Decoder

Register:
The Register operate reading process by default and operate writing process when "w" signal is high-level. The data in address [0] is fixed as [0]. Therefore, the following figure that present operation of register could be divided into two stages by "w" signal. In stage 1, the "w" signal is low-level. Only reading process will be operated, and corresponding result will be [0] for [0] address and null for other address. In stage 2, the writing process is activated and "Wdata" will be written into destination address (Raddr2). Then, the output "Rdata2" will present same value to "Wdata".

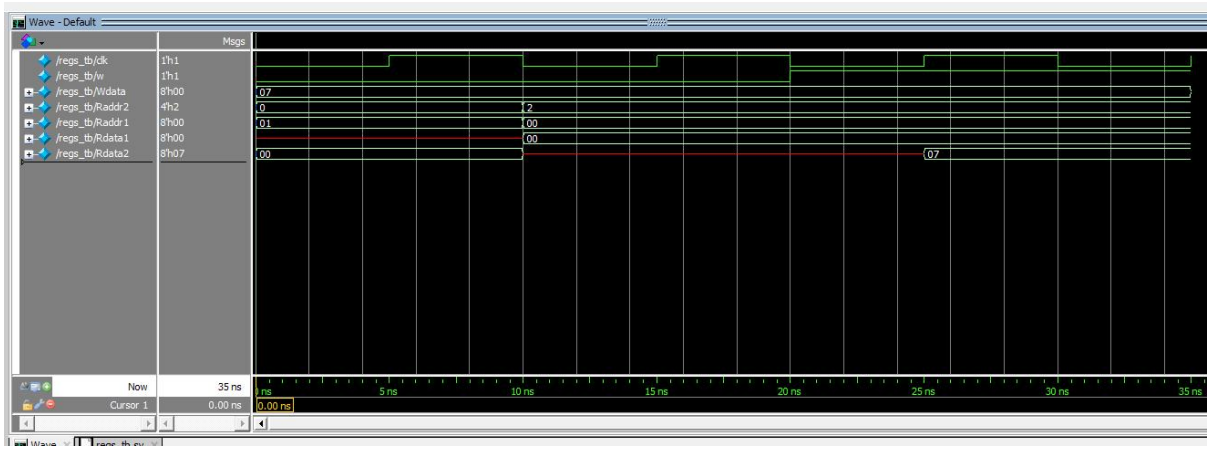


Figure. 2.4 Simulation result of Register

ALU:

The ALU component will make arithmetic logic operation for two inputs. The sort of operation depends on input "ALUfunc" defined in "alucodes.sv". The following illustration shows adding and subtracting operation, whose defined function code is 000 and 001, and it present correct output, which is 7 (4+3) and 1 (4-3).

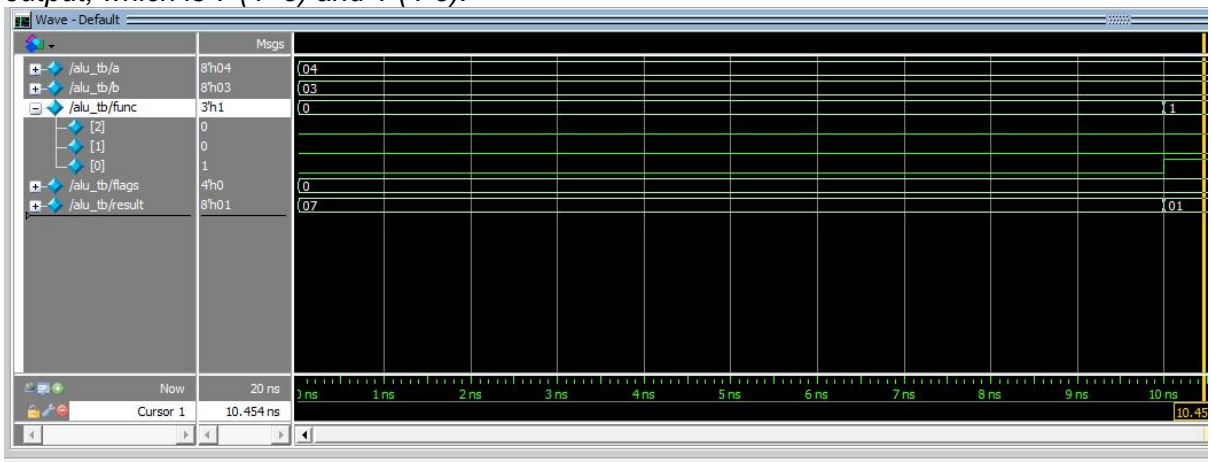


Figure. 2.5 Simulation result of ALU

CPU:

After all components are designed and tested, the integration system is tested. The X1 is imported as [01] and Y1 is imported as [02]. The system calculates by assigned algorithm and outputs correct answer that is [15] and [ec].

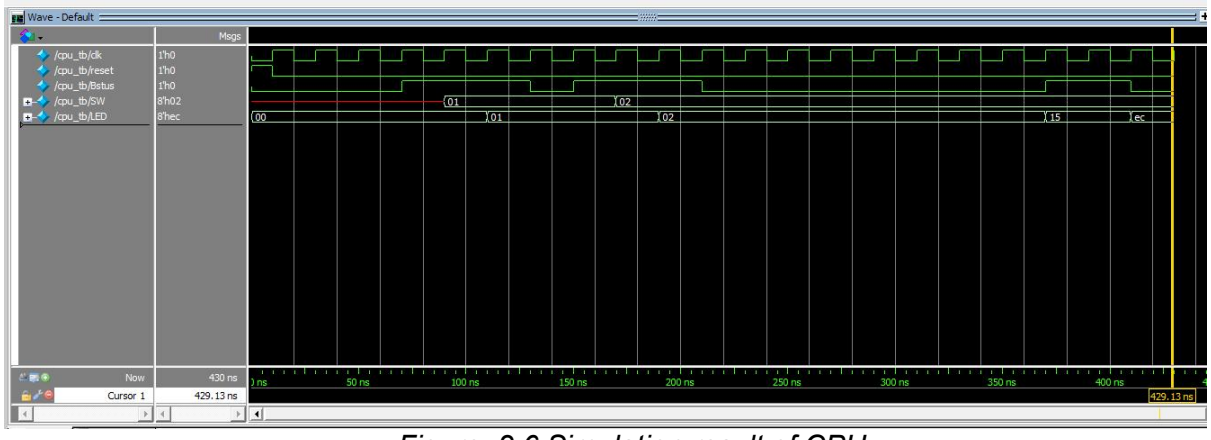


Figure. 2.6 Simulation result of CPU

3. FPGA implementation

After simulation is finished, the programme will be tested on FPGA broad, with device "5CSEMA5F31C6N". The top-level of architecture is set to "picoMIPS4test.sv". The programme installation consists of two sections: Pin management and connection configuration. The information of Pins and steps of connection configuration are presented in DE1-SoC manual [3], so that the details will not be repeated here.

After all above configuration completed, the broad is able to operate assigned function. The input and expected output data is listed in Table. 3.1, and the performance and result are illustrated in Figure. 3.1-3.3. It shows that the function is implemented correctly.

Table. 3.1 Input data and expected output data

Input Data	*Expected Output Data*
X1: 00000100 (4)	*X2: 00011011 (27)*
Y1: 00001000 (8)	*Y2: 11110000 (-16)*

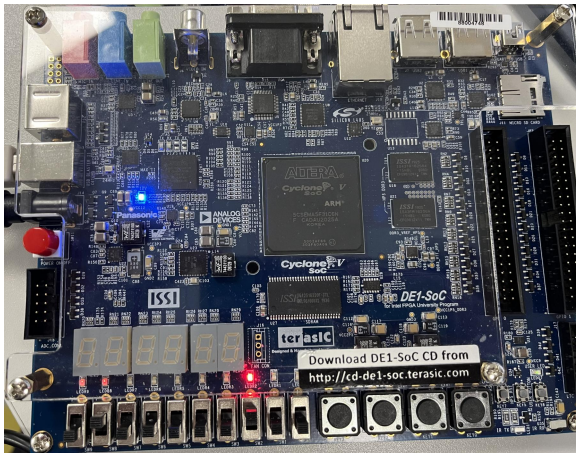


Fig. 3.1 Input X1 as 00000100

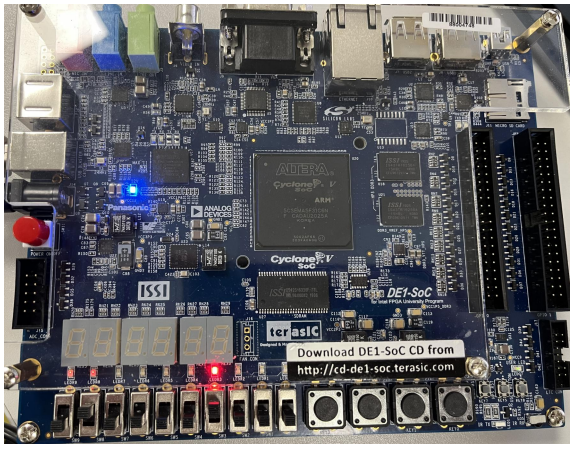


Fig. 3.2 Input Y1 as 00001000

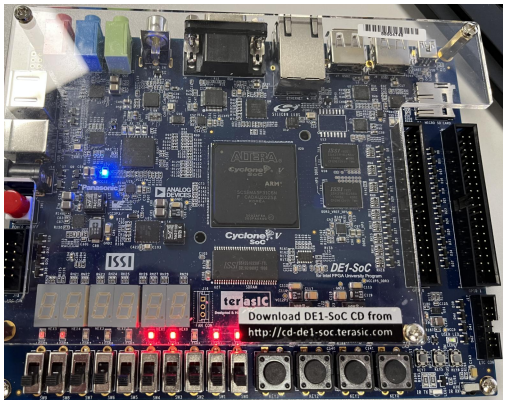


Fig. 3.3 Output X2 = 00011011

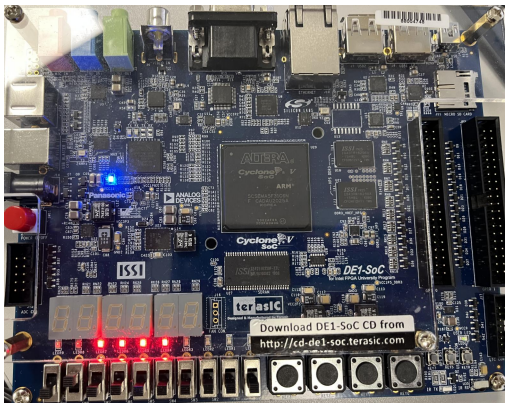


Fig. 3.4 Output Y2 = 00011011

In initial design, length of instruction is configured as 20 bits, in which address of source and immediate value are separated. To make further optimization, the address of source and immediate value are integrated. Some instructions are also integrated into its previous instruction to reduce cost. The ultimate cost is reduced from 78 to 63.

4. Conclusion

Through this assignment, a comprehensive view of the system cost is attained. The cost is mainly relative to size of Program Memory, method for arithmetic logic functions, and used RAM that is not considered in this case.

For students whose foundation of hardware is insufficient, picoMIPS is appropriate exercise as it is easy to understand and implement within specific range of time. However, to further develop this design, a NICS application or some extensions, such as segment presentation, are also considered.

5. References

[1]C. Leech and T. J. Kazmierski, "Energy Efficient Multi-Core Processing," *Electronics ETF,* vol. 18, no. 1, 2014, doi: 10.7251/els1418003l.

[2]M. Reshadi and D. Gajski, "NISC modeling and compilation," *Center for Embedded Computer Systems,* pp. 04-33, 2004.

[3]"DE1-SoC User Manual." http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/DE1-SoC_User_manual.pdf (accessed 23 April, 2023).