ELEC6234 – Embedded Processors

Coursework Report – picoMIPS implementation

Haosen Yu
hy3u21
MSc Electronic Engineering
Dr Harold Chong

1. Introduction

The objective of the assignment was to produce a n bits picoMIPS architecture processor for the affine transformation of graphic pixels, and implementation on an Altera FPGA development board with minimal logic resources.

This objective has all been achieved, and a machine-level program designed to handle a generic affine transformation algorithm based on a customized instruction sets has been completed. After inputting X1 and Y1 in the range -128 to 127, X2 and Y2 are output through the affine transformation algorithm.

In this machine-level program, the 6 constants given in the assessment requirements (the data sets 1) are used, and X1 and Y1 are input via SW0-SW7 to a multiplexer which is connected to the ALU. An instruction named "ADDF" is defined to store the input datas fetched from SW0-SW7 into the destination register. An instruction named "BAT" is defined to implement branch at specified conditions: Bstus(connect to SW[8]) ==Instruction [7], and this is also known as the handshake function.

In addition, to make the results easier to observe, I designed a new module for displaying the output results on both the LEDs and the 7-segment displays at the same time. This additional display module is designed to convert the result of the 2's complement format to Gray Code for decimal display, and also the sign bits (positive or negative) can be displayed correctly. A total of four 7-segment displays are used, with HEX0-HEX2 being used to display decimal results and HEX3 being used to display positive and negative signs. Details (including source code files) are provided in zip format on the SOTON handin systems.

Design Details Form

Total Cost: 58	ALMs: 58	Memory bits: 0	Multipliers: 1

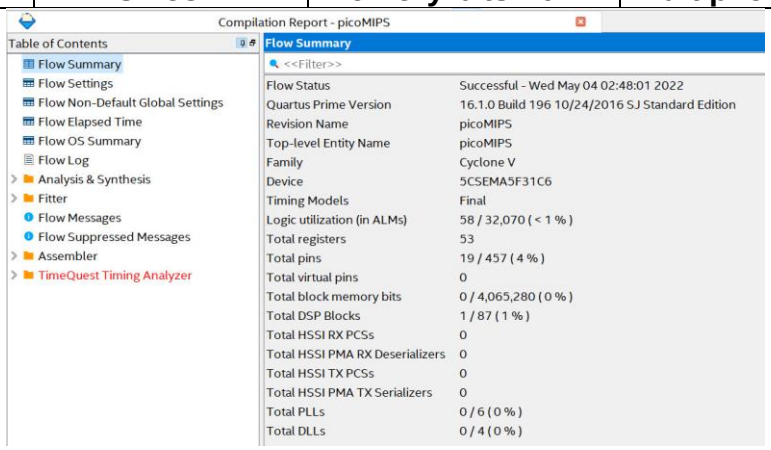

Fig.1-1 Quartus Flow Summary

Instruction Set

A total of 10 instructions have been implemented to ensure that the various bas operations are satisfied. There are some instructions such as SUB that are not used this design.

Details:

ADD	Adds the values stored in the source and destination registers and stores the result in the destination register.
ADDI	Add the immediate value to the value stored in the source register and store the result in the destination register.
ADDF	Add the immediate value fetched from the input (switchs) and the value stored in the source register, and store the result in the destination register.
SUB	Subtracts the values stored in the source and destination registers and stores the result in the destination register.
SUBI	Subtracts the immediate value to the value stored in the source register and store the result in the destination register.
MUL	Multiplies the values stored in the source and destination registers and stores the result in the destination register.
MULI	Multiplies the immediate value to the value stored in the source register and store the result in the destination register.
BAT	Branch at specified conditions: Bstus(connect to SW[8])==Instruction [7].
SHOW	Display the operation result of ALU on LEDs and 7-segment displays.
NOP	No operation.

Instruction Format

Databus size = 8 bits, Instructions size = 20 bits,
Format: 6bits opcode, 3bits address for destination register (%d), 3bits address for souce register (%s), 8bits immediate or address.

Details:

ADD	ADD %d, %s;	%d = %d + %s
ADDI	ADDI %d, %s, imm;	%d = %s + immediate value
ADDF	ADDF %d, %0;	%d = inport immediate value
SUB	SUB %d, %s;	%d = %d − %s
SUBI	SUBI %d, %s, imm;	%d = %s − immediate value
MUL	MUL %d, %s;	%d = %d × %s
MULI	MULI %d, %s, imm;	%d = %s × immediate value
BAT	BAT address;	branch at specified conditions (Bstus==I[7])
SHOW	SHOW %d, %0;	output display <= %d
NOP	No operation	No operation

Your affine transformation program

```
1  ADDI %0, %0, 0;     clear REG 0
2  BAT  1, 0;          until Bstus=0 goto next Instruction
3  BAT  0, 0;          until Bstus=1 goto next Instruction[REPEAT HERE]
4  ADDF %1, %0;        REG 1 <= inport x1
5  ADDF %2, %0;        REG 2 <= inport x1
6  BAT  1, 0;          until Bstus=0 goto next Instruction
7  BAT  0, 0;          until Bstus=1 goto next Instruction
8  ADDF %3, %0;        REG 3 <= inport y1
9  ADDF %4, %0;        REG 4 <= inport y1
10 BAT  1, 0;          until Bstus=0 goto next Instruction
11 MULI %1, %1, 0.75; %1 = %1 * 0.75; // 0.75x1
12 MULI %2, %2, -0.5; %2 = %2 * -0.5; // -0.5x1
13 MULI %3, %3, 0.5;  %3 = %3 * 0.5;  // 0.5y1
```

```
14  MULI %4, %4, 0.75; %4 = %4 * 0.75; // 0.75y1
15  ADD  %1, %3;       %1 = %1 + %3;   // 0.75x1 + 0.5y1
16  ADD  %2, %4;       %2 = %2 + %4;   // -0.5x1 + 0.75y1
17  ADDI %1, %1, 20;   %1 = %1 + 20;   // x2 = 0.75x1 + 0.5y1 + 20
18  SHOW %1, %0;       SHOW %1
19  BAT  0, 0;         until Bstus=1 goto next Instruction
20  ADDI %2, %2, -20;  %2 = %2 + -20;  // y2 = -0.5x1 + 0.75y1 - 20
21  SHOW %2, %0;       SHOW %2
22  BAT  1, 3;         until Bstus=0 goto LINE3 Instruction[BEGIN REPEAT]
```

Design Block Diagram

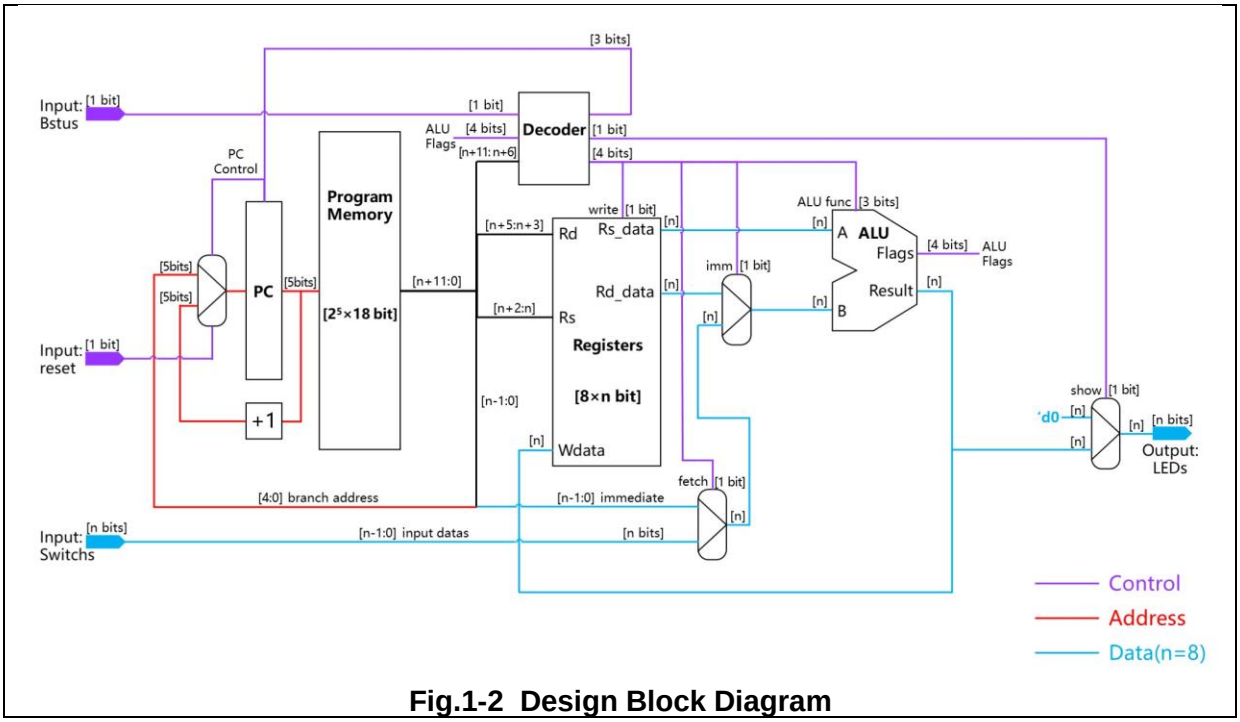

Fig.1-2 Design Block Diagram

2. Overall architecture of the design and simulations

In this project, an n-bit picoMIPS architecture processor for the affine transformation of graphics pixels needs to be designed. And in my design, the architecture uses an arithmetic logic unit(ALU), a general-purpose register(GPR), a 20 bits instruction decoder, a program counter(PC) and a program memory which stores the machine program of the affine transformation algorithm. These blocks are connected with the other blocks as shown in the design block diagram shown above. The input switches carrying the coordinate datas are connected to the ALU block via multiplexers.

The affine transformation is a transformation that preserves co linearity. This means that the points will still lie on the same line as it was before after undergoing transformation. It can be expressed by the equation shown below.

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

Here [x1, y1] are the coordinates of the pixel before the transform and [x2, y2] are the coordinates after the transformation We use the following dataset values which is given in assessment requirements to implement the transform.

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

Based on the above mathematical equations, we can get the following equation.

$$X2 = A_{11} \times X1 + A_{12} \times Y1 + B_1$$

$$Y2 = A_{21} \times X1 + A_{22} \times Y1 + B_2$$

As we can see from the above equation, we need 4 multiplication operations and 4 addition operations. We use the embedded multipliers in the dsp block to implement the multiplication operation. The picoMIPS architecture is more than enough to implement the affine transformation. We use a 8 bits data bus width for the picoMIPS and 20 bits instruction.

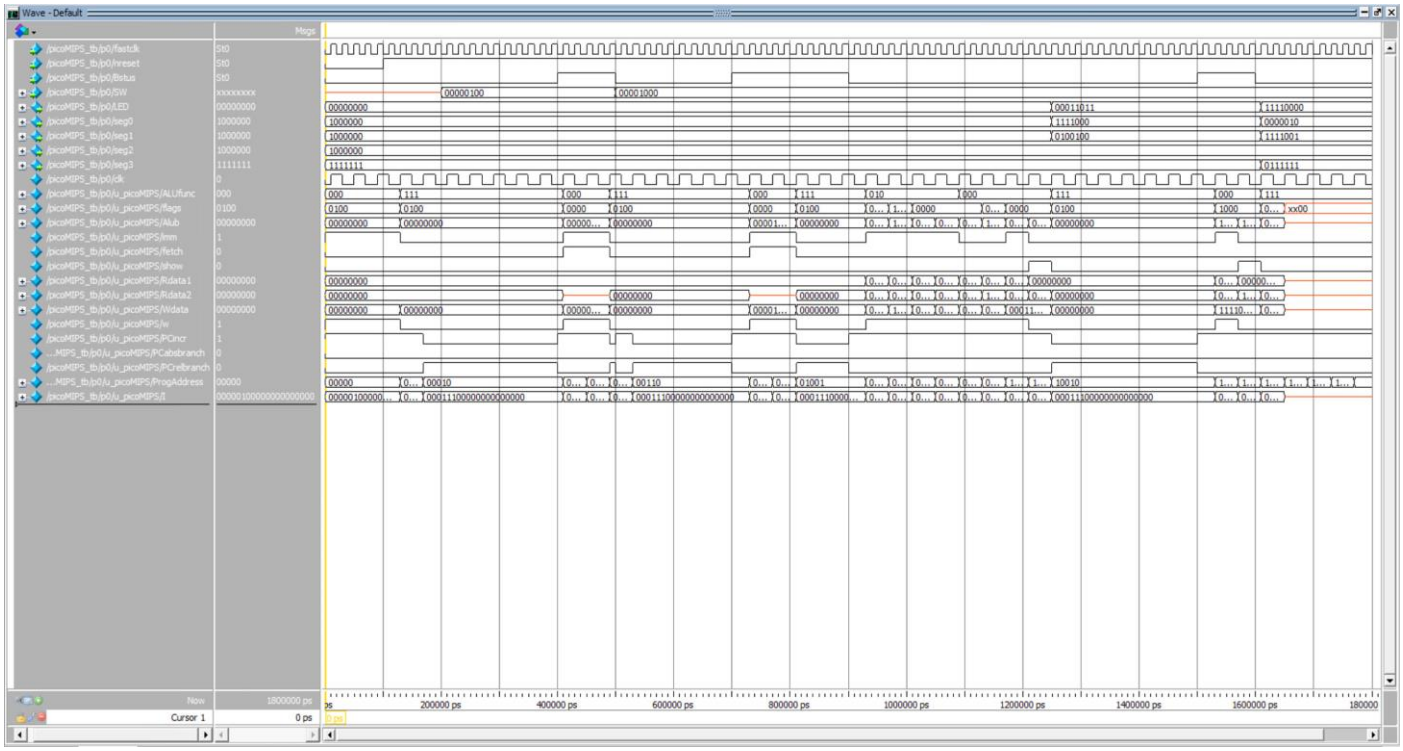

Fig.2-1 picoMIPS Modelsim Waveform Graph

The design was simulated using modelsim software and the results are shown in the waveforms above. The simulation shows clear working of the states and displays X1, Y1, X2 and Y2.

[GPR:]

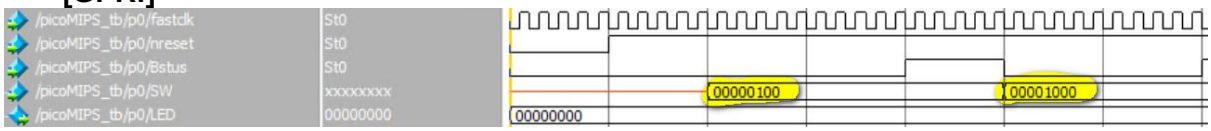

Fig.2-2 picoMIPS Modelsim Waveform Area Screenshot

As shown above, the system is reset when nreset signal is set low, SW is the input port and the values of X1(00000100_2) and Y1(00001000_2) are input one by one. A total of 4 general purpose registers are used, %1 and %2 to store X1 and %3 and %4 to store Y1.

[Decoder & ALU:]

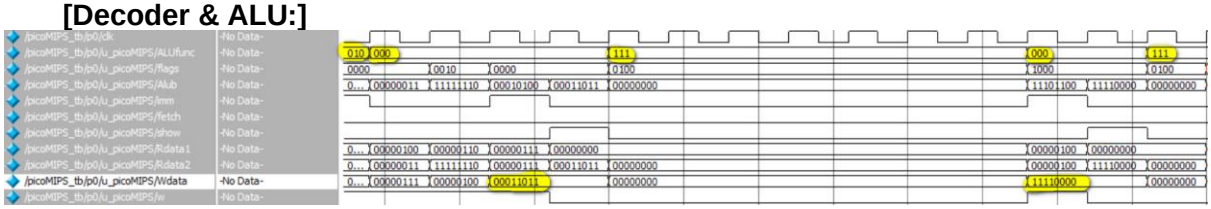

Fig.2-3 picoMIPS Modelsim Waveform Area Screenshot

As shown above, the ALU performs different operations depending on the different ALUfunc from the decoder, e.g. 010, 100, 111, etc. By performing different operations, the correct result is obtained, as shown in the diagram above, X2 equals 00011011_2 and Y2 equals 11110000_2.

Table.2-1 ALU output selection

Func value	Result
RADD (3'b000)	a + b
RSUB (3'b001)	a + (-b)
RMUL (3'b010)	a*b=>result[14:7] (fraction part is discarded)
RNOP (3'b111)	No operation

[PC:]

Fig.2-4 picoMIPS Modelsim Waveform Area Screenshot

As shown in the figure above, the program memory which stores the machine program of the affine transformation algorithm outputs the corresponding instructions (Green background) with a length of 24bits according to the different ProgAddresses with a length of 5bits that from program counter(PC).

[Outputs and Validation:]

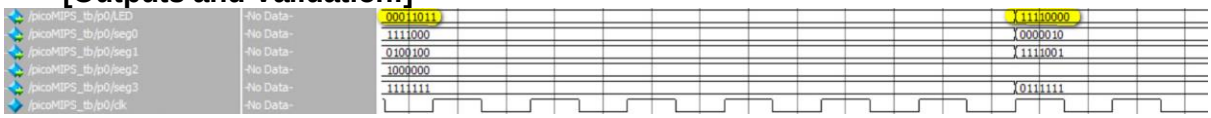

Fig.2-5 picoMIPS Modelsim Waveform Area Screenshot

In the figure above, LED is the result output port, showing the results of the calculations of X2(00011011_2), Y2(11110000_2) one by one. These results are the same as the expected results within the error range allowed.

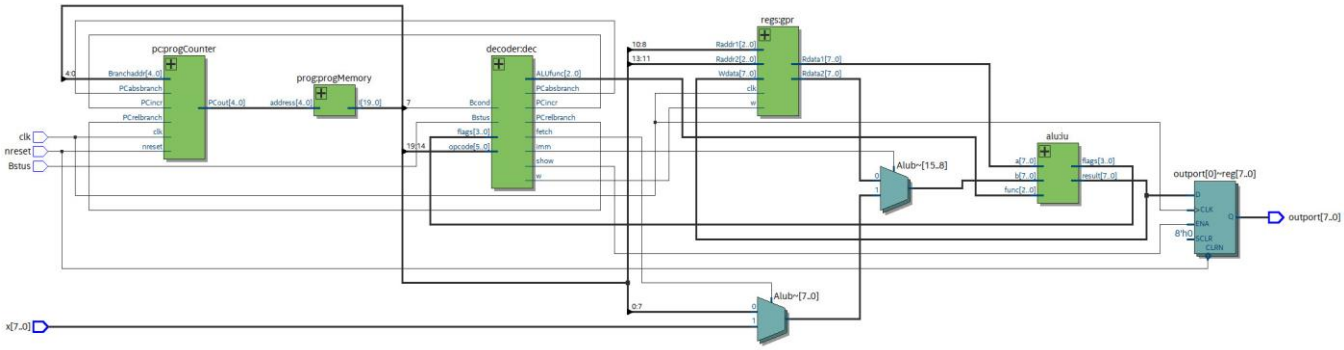

Fig.2-6 picoMIPS RTL Level Diagrams

The RTL diagram is shown above, the control path and the data processing path can be clearly seen, and the main CPU parts such as ALU, PC, Decoder, Registers and a program memory which stores the machine program of the affine transformation algorithm are also clearly visible. It seems to match the expected hardware as assessment requirements. High definition RTL level diagrams are provided in the zip attachment. By entering the test datas, the results are correctly displayed on the LEDs and the details are described in the next part.

3. FPGA implementation

The hardware was then synthesised in Quartus prime and was synthesised succesfully without any latches thanks to the use of non blocking assignments in always_ff blocks. Then the picoMIPS core was instantiated in a top-level module named "picoMIPS4test" with a clock divider. The purpose of using clock divider was to slow down the main clock to eliminate the bouncing effects of the mechanical switches on the board. Then test vectors were input through the switches, and the results were checked in compliance according to the pseudocode.

In order to test whether the system functions correctly, I chose some example data sets as input datas and observed whether the expected calculation results were displayed correctly. Also a series of separate testbench for each module were written to test each module independently. Details of the data sets used for testing are as follows.

Table.3-1 Testing Data Sets

Input Datas	
X1: 00000100_2 => 4_{10}	**Y1:** 00001000_2 => 8_{10}
Output Datas	
X2: 00011011_2 => 27_{10}	**Y2:** 11110000_2 => -16_{10}

The following photograph shows the output results of X2 on LEDs and 7-segment displays.

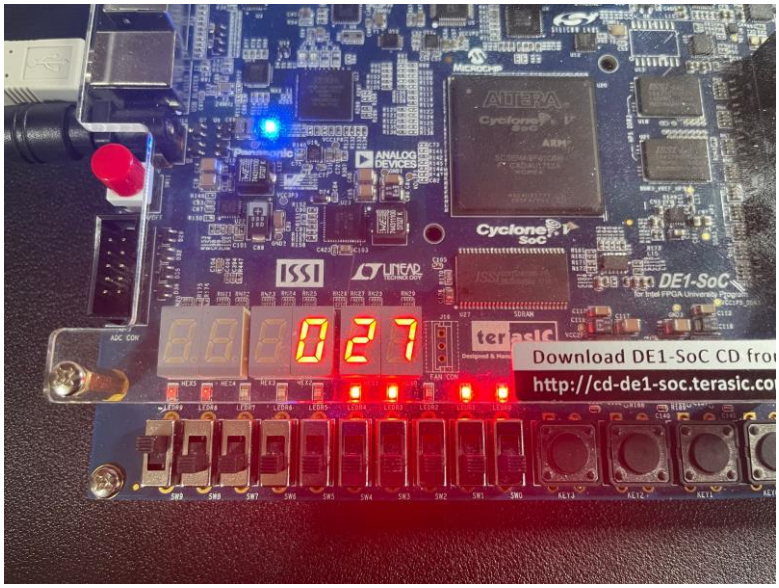

Fig.3-1 Output Results X2 on FPGA Development Board

The following photograph shows the output results of Y2 on LEDs and 7-segment displays.

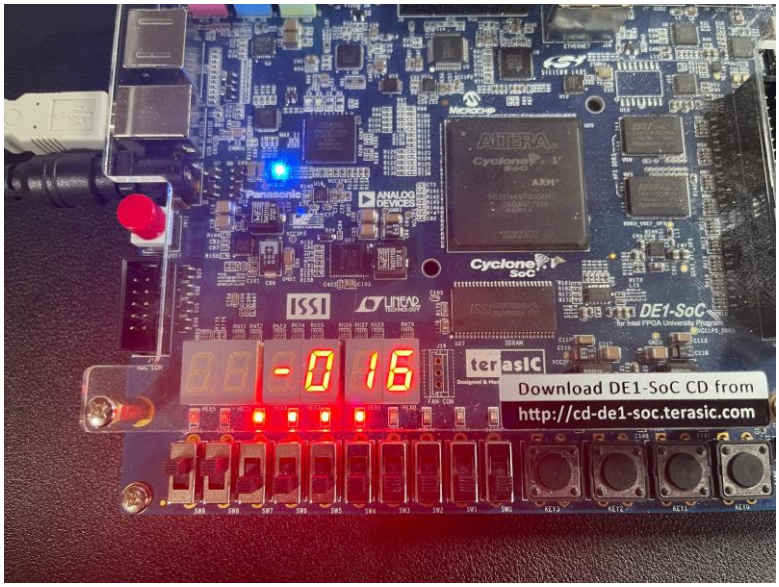

Fig.3-2 Output Results Y2 on FPGA Development Board

The specification of this design requires the design uses as fewer logic resources as possible. Therefore, when synthesising the design, the optimisation technique in the synthesis setting needs to be changed from “Balanced (default)” to “Area”. The results are shown in Table below.

Table.3-2 Effect of using synthesis optimisation technique

Technique	Resources Cost
Balanced	61
Area	58

To reduce the resources cost of design, I also have tried many other methods. The first version of this design using 24-bit instruction. The second version reduced this length to 20 bits. However, the cost figure did not change. However, by reducing Psize from 6 to 5, I gained a more significant cost reduction.

4. Conclusion

In this project, all the functional requirements and objectives have been fully achieved. A smallest picoMIPS architecture processor for the affine transformation of graphic pixels was successfully designed, and a machine-level program for the general affine transformation was developed at the same time. The processor is implemented and verified on an Altera FPGA development board with minimal logic resources.

This Assignment gives a good understanding of the process of embedded processor synthesis, and my knowledge of computer architecture has been boosted, my ability to do digital hardware design also has been reinforced. If there is an opportunity to do this project again, I would utilising the RAM to store the program to reduce the resources cost, it also can improve efficiency and optimize resource consumption.

5. References

[1] Dr Tom J Kazmierski, "ELEC6234 Embedded Processor Synthesis: Notes," University of Southampton [Online]. Available: https://secure.ecs.soton.ac.uk/notes/elec6234/

[2] ZwolińskiM., Digital system design with SystemVerilog. Upper Saddle River, Nj: Addison-Wesley, 2010.

[3] D. D. Gajski, S. Abdi, A. Gerstlauer, Gunar Schirner, and Springerlink (Online Service, Embedded System Design : Modeling, Synthesis and Verification. New York, Ny: Springer Us, 2009