ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRN : 21336

LECTURER : Prof. Dr. Mustafa Ersel Kamaşak

GROUP MEMBERS:

150200905 : Mawada Khaled

820220319 : Ahmet Enes Yılmaz

SPRING 2024

Contents

1 INTRODUCTION

In this project, our focus revolves around the initial stages of constructing a basic computer system using the Verilog hardware description language. Through this implementation, we aim to simulate fundamental functionalities essential for a computer system. These functionalities encompass arithmetic and logical operations, alongside the ability to store resultant values in memory or registers. Furthermore, our design facilitates sequential execution of these operations, ensuring coherence and order in processing tasks.

2 MATERIALS AND METHODS

Throughout this project, we have designed and implemented a comprehensive Arithmetic Logic Unit.

We have integrated various components, including Register File, Address Register File (ARF), and Instruction Register (IR).

2.1 PART 1

In the initial phase of the project, we developed a 16-bit general register essential for subsequent modules such as the Register File and the Address Register File (ARF). This register offers eight functionalities when it is enabled:

- Decrement by 1.
- Increment by 1.
- Load data.
- Clear contents to 0.
- Clear the upper 8 bits and write the lower 8 bits.
- Write only the lower 8 bits.
- Write only the upper 8 bits.
- Sign-extend the upper 8 bits from the 7th bit of the input, and write the lower 8 bits.

2.2 PART 2

2.2.1 PART 2.a

In this part, we introduce the design of a 16-bit Instruction Register (IR), crucial for storing binary instructions within our system. Although the IR is capable of holding 16 bits, it receives input data of only 8 bits. To manage this, an additional signal, L'H, determines whether the lower or upper half of the IR will be affected by the operation.

The IR operates as follows:

- When L'H is 0, the IR maintains its current value.
- If L'H is 1 and Write is 1, the lower 8 bits load input data (LSB).
- If L'H is 1 and Write is 1, the upper 8 bits load input data (MSB).

To implement this functionality, we employ behavioral Verilog, utilizing conditional statements to handle operations on the relevant half of the IR.
The detailed design of our implementation:

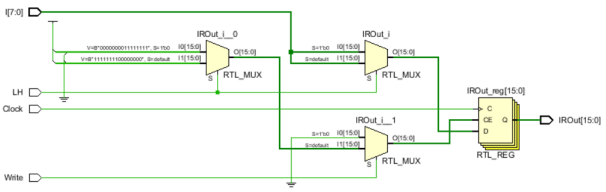

Figure 1: Schematic for Instruction register

2.2.2 PART 2.b

In this new segment, we introduce the design of a system comprising four 16-bit general-purpose registers (R1, R2, R3, R4) and four 16-bit scratch registers (S1, S2, S3, S4). These registers play a crucial role in storing data within the system. The system operates based on several inputs and outputs as detailed below:
Inputs:

- OutASel and OutBSel: These signals determine which registers' contents are outputted to OutA and OutB, respectively.
- RegSel and ScrSel: These 4-bit signals select the registers to apply functions determined by the 3-bit FunSel signal. The selection of registers based on RegSel and ScrSel.

Outputs:

- OutA and OutB: These outputs carry the contents of the selected registers, as determined by OutASel and OutBSel.

The functionality of the system is governed by the FunSel control input, which specifies the operation to be performed on the selected registers. The operations include decrementing, incrementing, loading, clearing, and various write operations.
To implement this system, we employ behavioral Verilog, leveraging multiplexers to select the appropriate registers based on the input selectors. Additionally, we use conditional statements to execute the specified operations on the selected registers as dictated by the FunSel signal. This design ensures efficient storage and manipulation of data within the system's registers, facilitating smooth operation of the overall system.
The detailed design of our implementation:

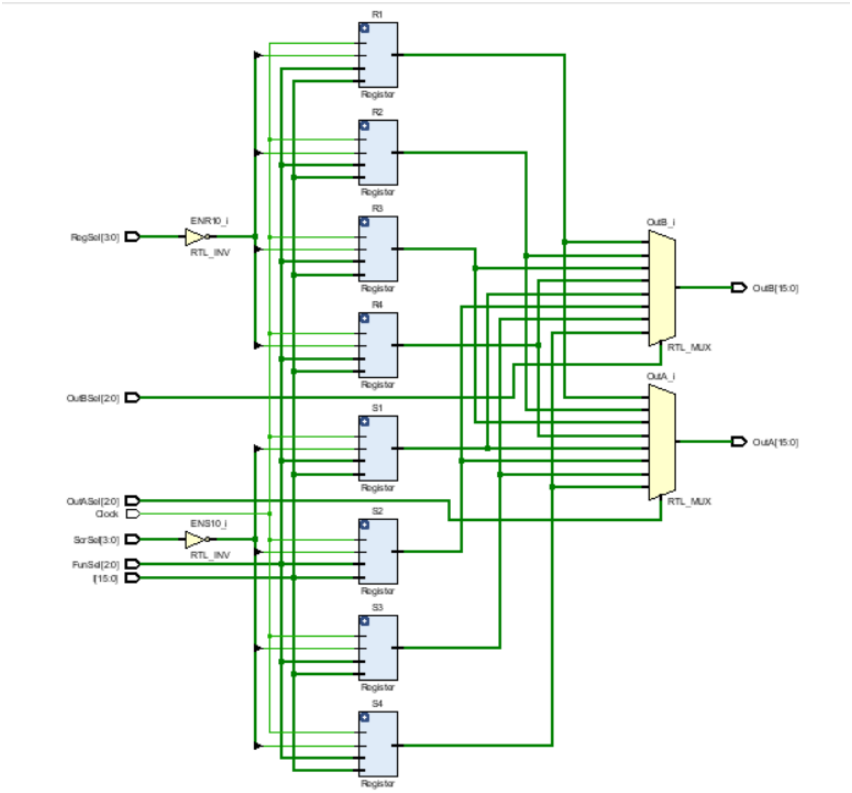

Figure 2: Schematic for Register File

2.2.3 PART 2.c

In this section, we introduce the implementation of an Address Register File (ARF) comprising three registers: PC, AR, SP. The design involves six inputs, including a load signal, a register selector, two output selectors, a function selector, and the clock signal.

To initialize the registers, we utilize the 16-bit register module developed in the initial phase. Similar to the Register File design, the load signal, function selector, and clock are provided as inputs to each register. The register selector, represented by RegSel, is used to enable individual registers, following a pattern similar to the Register File. Specifically, each bit of RegSel is mapped to a distinct wire, which acts as an enable input for the corresponding register.

Upon being enabled, each register undergoes modification based on a function selected by FunSel. Two outputs, OutA and OutB, are generated from the outputs of all registers using multiplexers. These outputs are determined by the OutCSel and OutDSel selectors, respectively.

The Verilog code provided encapsulates this design, with the AddressRegisterFile module instantiating the individual registers (PC, AR, SP) and facilitating the selection of output registers through multiplexers. This design ensures efficient storage and manipulation of data within the Address Register File, essential for system operation.

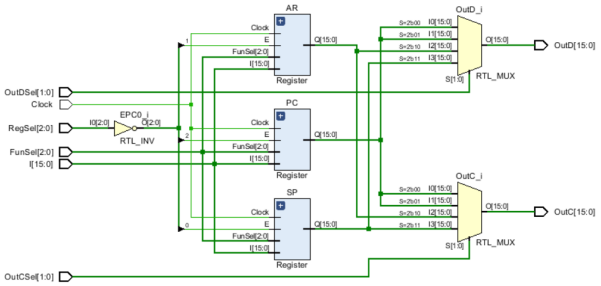

Figure 3: Schematic for Address Register File

2.3 PART 3

In this new section, we present the design and implementation of an Arithmetic Logic Unit (ALU), which serves as a combinational circuit performing various microoperations based on the input function selector (FunSel). The ALU operates on two 16-bit inputs (A and B) and produces a 16-bit output (ALUOut) along with four flag outputs: zero (Z), negative (N), carry (C), and overflow (O).

The ALU functions and their corresponding effects on the flags are specified in Table 6, with the Write Flag (WF) controlling whether the flags are updated. The ALU operates using 2's complement logic, with each function producing a result based on the selected inputs and operation.

The Verilog code provided encapsulates this design, with the ArithmeticLogicUnit module

defining the ALU's behavior. The ALUOut is determined by the FunSel input, which selects the specific operation to be performed. The flags are updated accordingly based on the result of the operation and stored in a register for subsequent use.
The ALU design efficiently handles a range of arithmetic and logic operations, ensuring accurate computation and flag generation essential for system functionality.

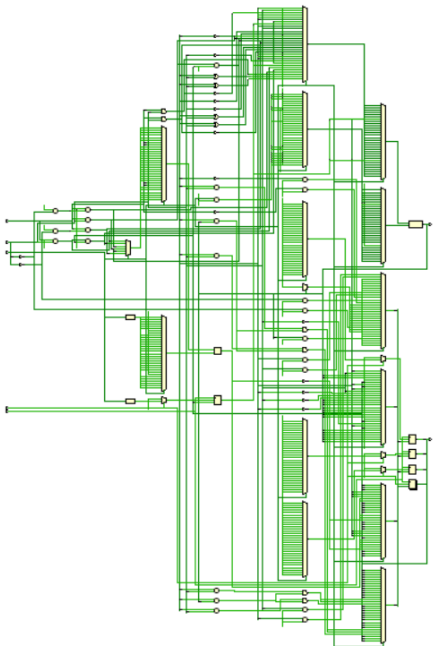

Figure 4: Schematic for ALU

2.4 PART 4

In this segment, we've developed an ALU (Arithmetic Logic Unit) System, a set of digital circuits capable of performing arithmetic and logical operations on binary integers. The system integrates modules such as the memory module from the provided files, alongside previously developed components like the ALU itself, the Register File, the Address Register File (ARF), and the Instruction Register (IR).
Data and instructions are fetched from memory, although the IR only utilizes half of its stored bits. The Register File features eight registers integrated into the circuitry to expedite consecutive operations, enhancing processing speed. Meanwhile, the Address Register File includes registers like the Program Counter (PC) and Address Register (AR), crucial for managing addresses.
Control inputs, called selectors, are pivotal in determining ALU operations and managing data flow throughout the system. These selectors ensure efficient execution of various arithmetic and logical tasks while coordinating input flow across memory, registers, and the ALU.

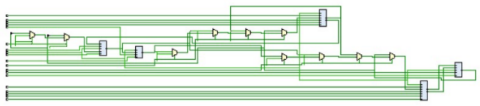

Figure 5: Schematic for ALU system

3 RESULTS

Register Simulation Started
[PASS] Test No: 1, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 2, Component: Q, Actual Value: 0x0024, Expected Value: 0x0024
[PASS] Test No: 3, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 4, Component: Q, Actual Value: 0x0026, Expected Value: 0x0026
[PASS] Test No: 5, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 6, Component: Q, Actual Value: 0x0012, Expected Value: 0x0012
[PASS] Test No: 7, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 8, Component: Q, Actual Value: 0x0000, Expected Value: 0x0000
Register Simulation Finished
0 Test Failed
8 Test Passed

Figure 6: Register Results (8/8)

```
RegisterFile Simulation Started
[PASS] Test No: 1, Component: OutA, Actual Value: 0x1234, Expected
Value: 0x1234
[PASS] Test No: 1, Component: OutB, Actual Value: 0x5678, Expected
Value: 0x5678
[PASS] Test No: 2, Component: OutA, Actual Value: 0x1234, Expected
Value: 0x1234
[PASS] Test No: 2, Component: OutB, Actual Value: 0x3548, Expected
Value: 0x3548
RegisterFile Simulation Finished
0 Test Failed
4 Test Passed
```

Figure 7: RF Results (4/4)

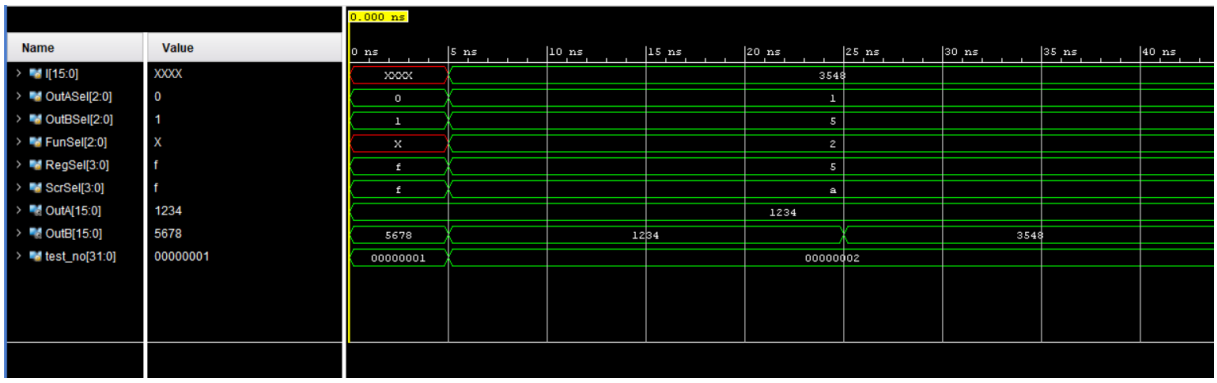

Figure 8: RF Results

```
AddressRegisterFile Simulation Started
[PASS] Test No: 1, Component: OutC, Actual Value: 0x1234, Expected
Value: 0x1234
[PASS] Test No: 1, Component: OutD, Actual Value: 0x5678, Expected
Value: 0x5678
[PASS] Test No: 2, Component: OutA, Actual Value: 0x1234, Expected
Value: 0x1234
[PASS] Test No: 2, Component: OutB, Actual Value: 0x3548, Expected
Value: 0x3548
AddressRegisterFile Simulation Finished
0 Test Failed
4 Test Passed
```

Figure 9: ARF Results (4/4)

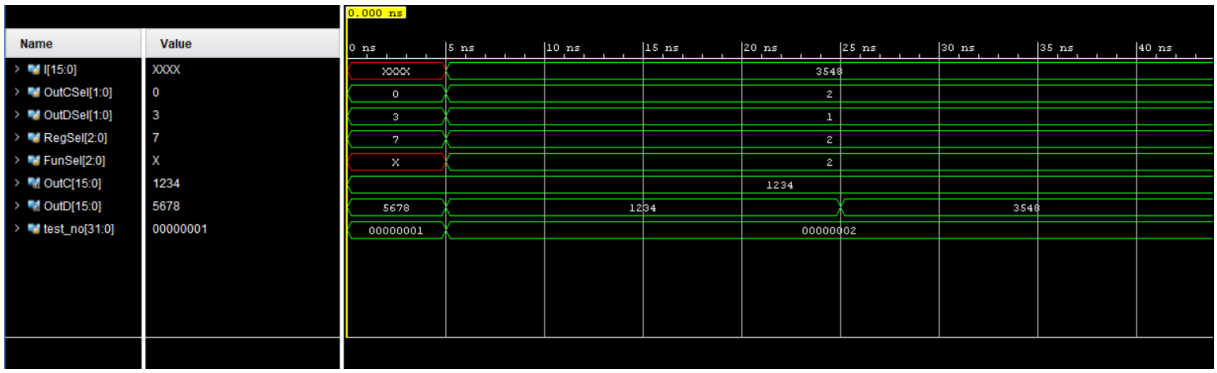

Figure 10: ARF Results (4/4)

```
InstructionRegister Simulation Started
[PASS] Test No: 1, Component: IROut, Actual Value: 0x2315, Expected
Value: 0x2315
[PASS] Test No: 2, Component: IROut, Actual Value: 0x1567, Expected
Value: 0x1567
InstructionRegister Simulation Finished
0 Test Failed
2 Test Passed
```

Figure 11: IR Results (2/2)

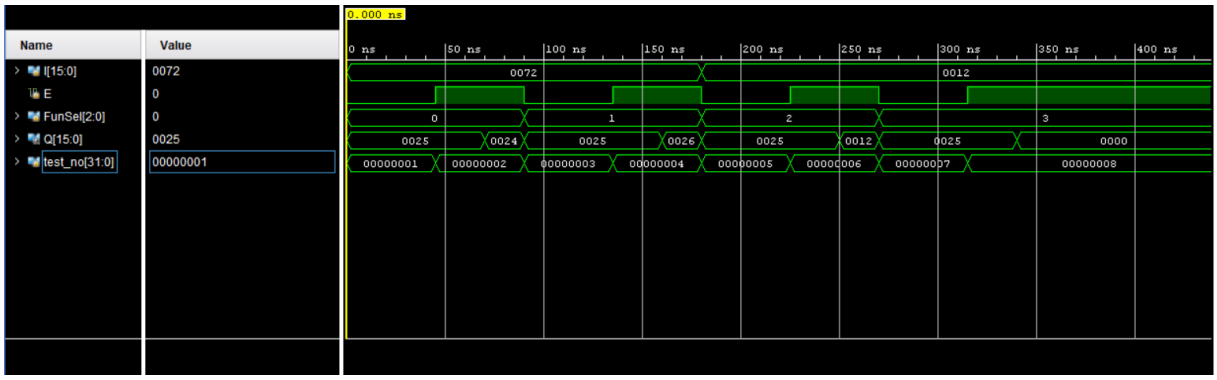

Figure 12: IR Results

ArithmeticLogicUnit Simulation Started
[PASS] Test No: 1, Component: ALUOut, Actual Value: 0x5555, Expected Value: 0x5555
[PASS] Test No: 1, Component: Z, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: C, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: N, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: O, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 2, Component: ALUOut, Actual Value: 0x5555, Expected Value: 0x5555
[PASS] Test No: 2, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: O, Actual Value: 0x0000, Expected Value: 0x0000

Figure 13: ALU Results (145/145)

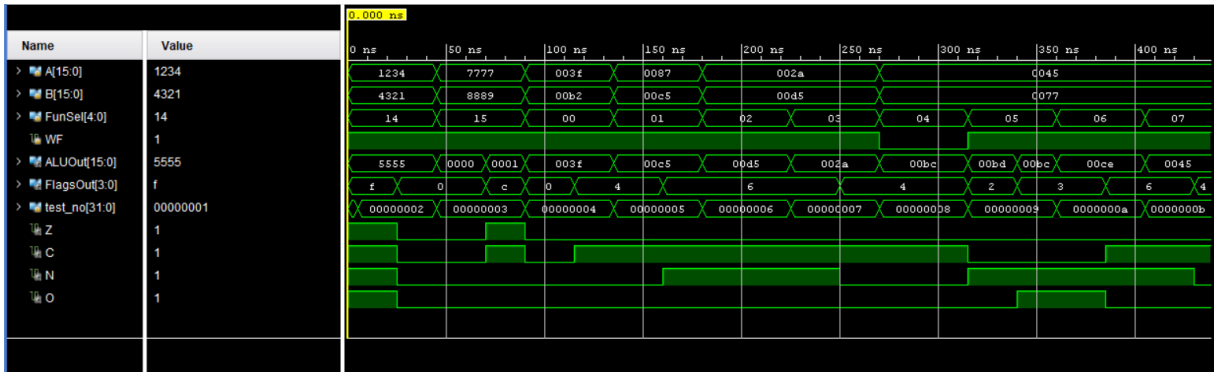

Figure 14: ALU Results

ArithmeticLogicUnitSystem Simulation Started
[PASS] Test No: 1, Component: OutA, Actual Value: 0x7777, Expected Value: 0x7777
[PASS] Test No: 1, Component: OutB, Actual Value: 0x8887, Expected Value: 0x8887
[PASS] Test No: 1, Component: ALUOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: MuxAOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: MuxBOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: MuxCOut, Actual Value: 0x00fe, Expected Value: 0x00fe
[PASS] Test No: 1, Component: R2, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: S3, Actual Value: 0x0000, Expected Value: 0x0000

Figure 15: ALU System Results (33/33)

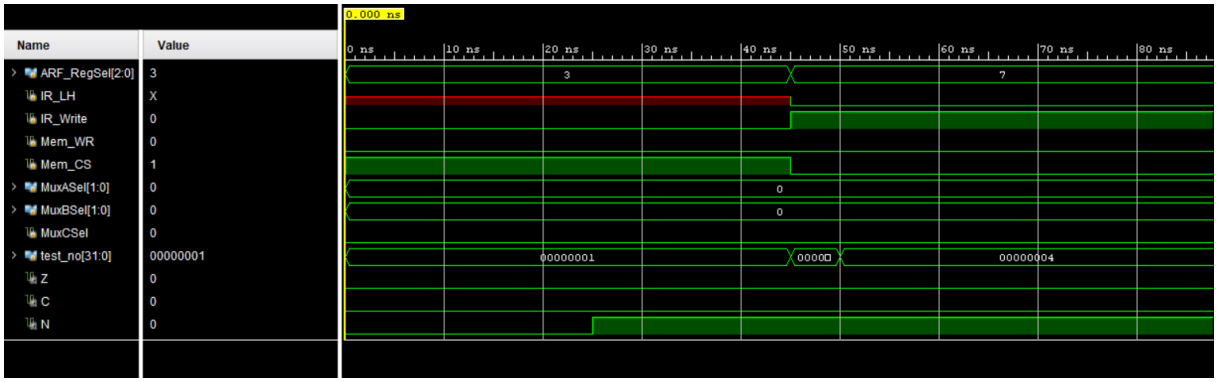

Figure 16: ALU System

4 DISCUSSION

For the first part, we designed a 16-bit general register that is utilized later on in the implementation of the Register File and the Address Register File (ARF). To design this module we used funsel to select which operation.

For second part, we designed a 16-bit Instruction Register (IR). The IR has different functions. Instruction Register also fetches data and instructions from memory, it only utilizes half of its stored bits in this implementation. The elaborated design can be seen below:

We designed also and implemented a Register File. The Register File consists of eight registers within the circuitry. The Address Register File includes registers like the Program Counter (PC) and Address Register (AR), essential for managing addresses within the system.

We have implemented a wide range of operations, including addition, subtraction, bit-wise AND/OR/XOR, left/right shifts, and more, both for 8-bit and 16-bit data. These operations are performed by selectors that have a crucial role in managing output and flagsout throughout the system. These selectors determine which operations the ALU performs and facilitate the movement of data between memory, registers, and the ALU.

The system generates flags such as zero (Z), carry (C), negative (N), and overflow (O) based on the results of ALU operations, providing crucial information about the outcome of computations.

In summary, this project represents the successful design and implementation of a functional ALU system within a digital circuit framework. Through careful integration and management of various components, we have created a versatile system capable of performing a wide range of arithmetic and logical operations on binary integers.

5 CONCLUSION

Throughout this project, we encountered challenges in designing a comprehensive ALU system. Integrating various digital circuit components, ensuring seamless communication between modules, generating accurate flags, and synchronizing operations using a single clock signal.

However, these challenges provided valuable learning experiences. We deepened our understanding of digital circuits, honed our problem-solving skills, and improved team-work and collaboration. Despite the obstacles, we successfully designed and implemented

a functional ALU system. The difficulties we have faced some points were not clear in the pdf.