*EESM5020 Fall 2020*

20618102 QIAN KUN
20672689 DING YAYI

# A 16-bit SIMD Microprocessor

## I. Design of SIMD Microprocessor Architecture

In this project, 16-bit SIMD microprocessor is constructed. 2's compliment calculations, shifting and logical operations are implemented in this ALU. To execute each instruction, 4 state signals are needed, which are IF, ID, EX and WB, and each state consumes one clock.

For state IF, 12-bit instruction is fetched and stored into command register, then turn to state ID.

In state ID, the instruction is decoded to get opcode, register address of input data and output data as well as bitwise of data. The bitwise indicates whether data is 16 bits, or 2 sets of 8 bits, or 4 sets of 4 bits. Besides, different bitwise uses different range of register address. The opcode can select corresponding ALU operation. The input data for ALU are stored in registers in advance.

As for EX, operation enabled by opcode will be executed, and the output of the enabled algorithm part will be generated. Finally in WB stage, generated output data will be written into the register that assigned by instruction.

In order to speed up, instruction level pipeline is implemented. We use the state machine to control the stages change at first, but the instructions can only execute in serial. So we added some registers to enable each stage use independent registers for calculate and store, then four independent instructions can run at same time in different stages parallel. However, it must take four clock cycle to release the first output, and then each clock cycle will finish one instruction.

The registers are separated into 12 words for stored, each word conclude 16bits. The register [0]-[3] are for 4bits calculate, register [4]-[7] are for 8 bits and register [8]-[11] for 16bits. The structure shown in figure 1.

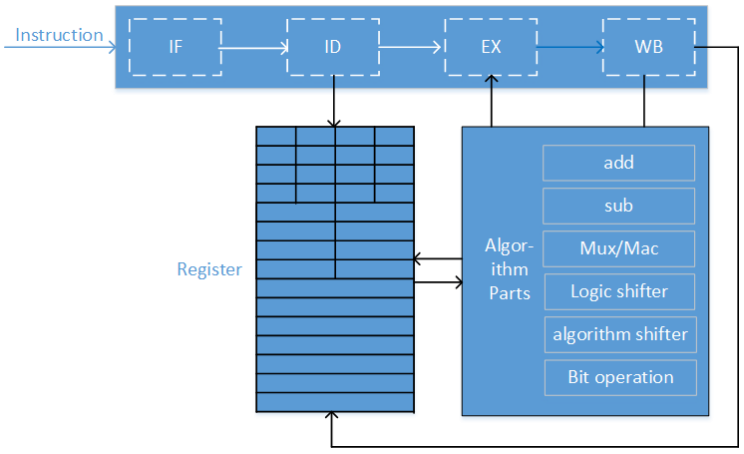



Figure 1 The structure

## II. Design of Operation in ALU

Operations in ALU have arithmetic operations, shifting operations and logical operations. Arithmetic Operations. The data used in ALU is 2's compliment data. Arithmetic operations include addition, subtraction, multiplication and multiply-and-add (MAC) operation. Shifting Operations have logical left shift, arithmetic left shift, logical right shit and arithmetic right shift. Logical operations include bitwise AND, OR, XOR and NOT. Arithmetic computation has 3 modes, which work as a single instruction on 1 set of 16-bit data, 2 sets of 8-bit data or 4 sets of 4-bit data. 2-bit Bitnum is used to indicate data mode: 00 for 4-bit input data, 01 for 8-bit data and 10 for 16-bit input data.

### 1. Addition and Subtraction

16-bit input data A and B are divided into 4 set of 4-bit data, which are the input of 4 4-bit adder as shown in figure 2 . For 4 set of 4-bit addition, carry-out of previous adder should not be taken as carry-in of the next stage adder. For 2 set of 8-bit addition, carry-out of adder0 and adder 2 should be taken as carry-in of adder1 and adder 3. For 16-bit addition, carry-out of previous adder should be taken as carry-in of the next adder.

The difference between addition and subtraction is that 4-set 4-bit input data B need to be inverted at first. When carry-in from previous stage is not needed, each set of the result of A adding B needs to add 1 to get 2's compliment data.

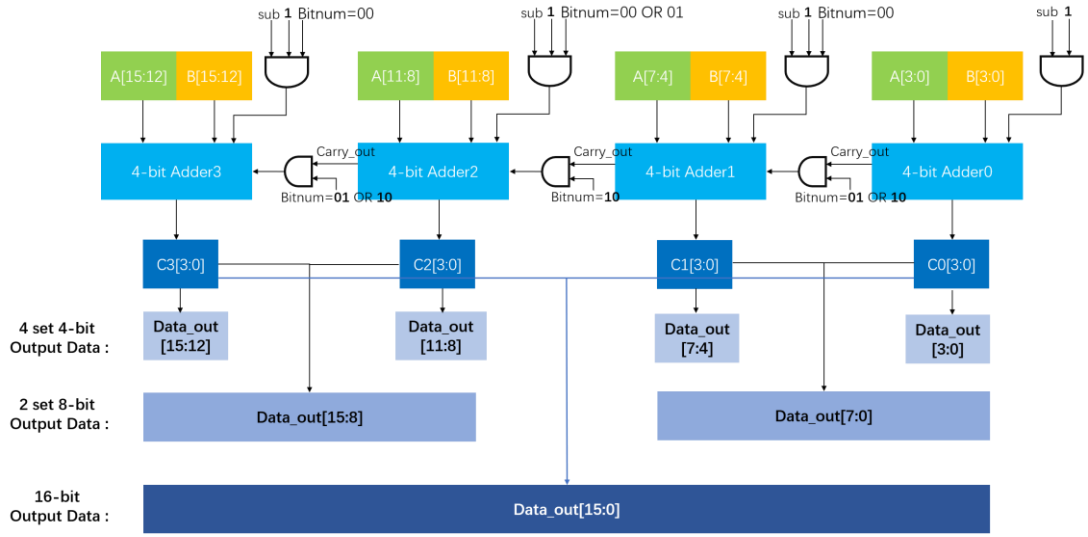



Figure 2. Structure of addition and subtraction

### 2. Multiplication

Assuming that 16-bit input data are A and B. Using B[i] AND A[15:0] respectively to get 16 16-bit partial product Mi[15:0].

Sign extension and shifting are needed when adding partial product. MSB of input data indicates the sign bit. For example, Mi[15] is the sign bit for 16x16 multiplication, Mi[15] and Mi[7] are sign bits for 8x8 multiplication, and Mi[3], Mi[7], Mi[11], Mi[15] are sign bits for 4x4 multiplication. Corresponding partial product need to do left shift and then adding shift results.

When sign bit multiplies A, it is necessary to convert the product into 2's complement form, which is indicated as ‘Mi[n] x Sign bit’ in the figure of multiplication structure.

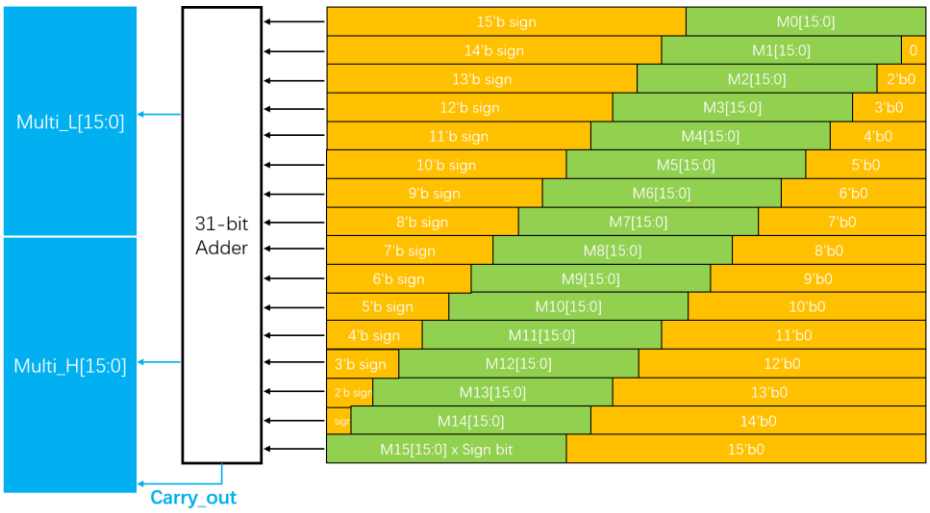



Figure 3. 1 set of 16x16 multiplication

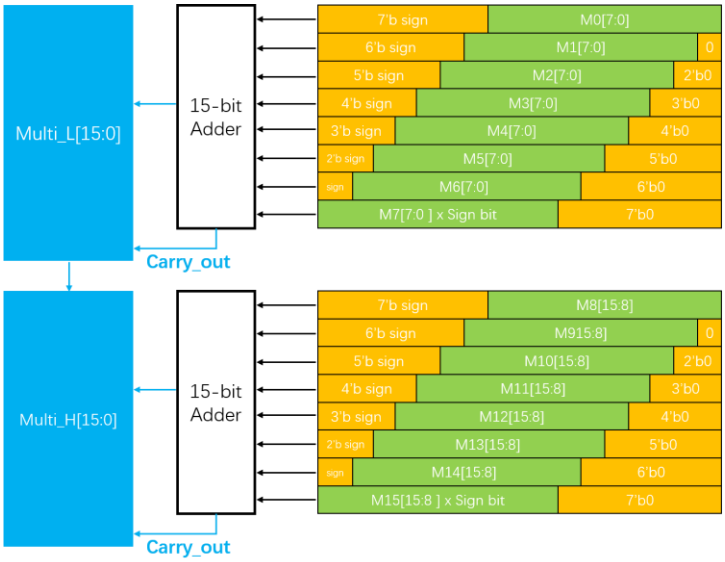



Figure 4. 2 set of 8x8 multiplication

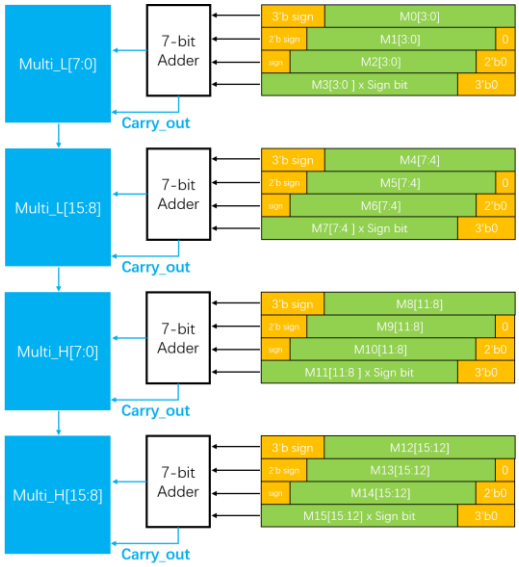



Figure 5. 4 set of 4x4 multiplication

## 3. MAC

MAC operation is the combination of multiplication and addition. Mac code is included in multi.v file, 'select' signal is used to select MAC operation or multiplication. When select=0, it is a multiplication operation of input A and B, and summand input C is set to be 0. While it will be MAC operation when select=1, and summand input C will be given.

## 4. Shifter

Shifting is a bitwise operation. For both arithmetic and logical left shift, LSB should be set 0, which is shown in figure 6.

For right shift, arithmetic right shift shown should do sign extension and MSB should be the same as sign bit. While logical right shift needs to set MSB to be 0, sign bit in figure 7 should be replaced to be 0.

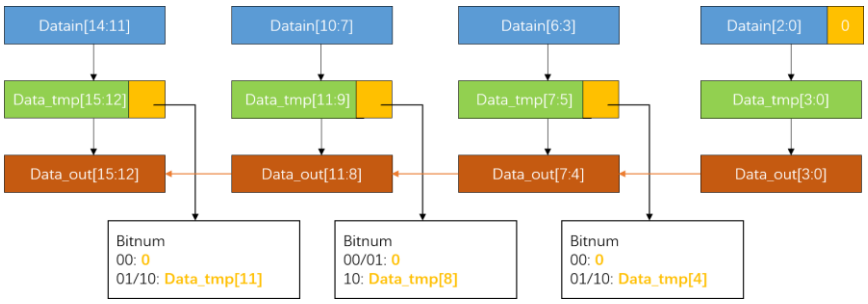



Figure 6. Structure of left shift

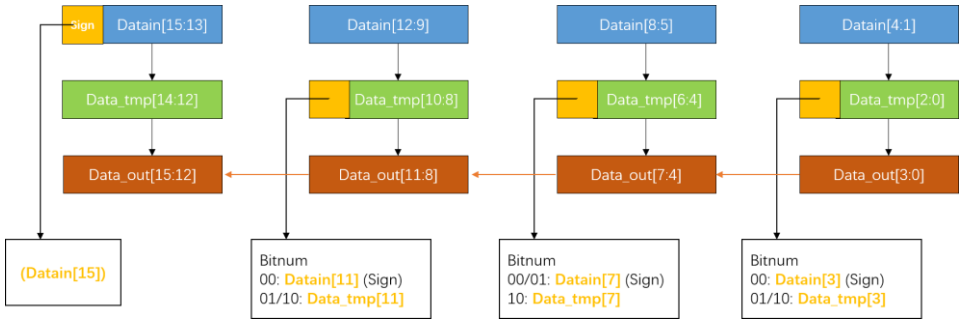



Figure 7. Structure of arithmetic right shift

### 5. Logical Operation

Bitwise AND operation use '&' to implement and bitwise OR operation is implemented by '|'. For NOT operation, using '!' to invert input data. As for XOR operation, if input data A is equal to input B, output will be 0, otherwise output will be 1.

## III . Opcode List

The 4-bit opcode of ALU operation and 2-bit bitwise are shown below:

```
// Arithmetic Operations
// parameter add4bit    = 0000_00
// parameter add8bit    = 0000_01
// parameter add16bit = 0000_10_
// parameter sub4bit    = 0001_00_
// parameter sub8bit    = 0001_01_
// parameter sub16bit = 0001_10_
// parameter mux4bit    = 0010_00_
// parameter mux8bit    = 0010_01_
// parameter mux16bit = 0010_10_
// parameter mac4bit    = 0011_00_
// parameter mac8bit    = 0011_01_
// parameter mac16bit = 0011_10_
```

```
//Shifting Operations
// parameter shift_lg_4bit    = 0100_00_
// parameter shift_lg_8bit    = 0100_01_
// parameter shift_lg_16bit = 0100_10_
//
// parameter shift_am_4bit    = 0110_00_
// parameter shift_am_8bit    = 0110_01_
// parameter shift_am_16bit = 0110_10_

// Logical Operations
// parameter and_bit    = 1000
// parameter or_bit     = 1001
// parameter xor_bit    = 1010
// parameter not_bit    = 1011
```

## IV．Simulation Results

### 1. Behavior-Level Simulation

The testbench has tested all operations in ALU.

Instruction 1: add8bit reg4 + reg5 = 0000_ffd1
Instruction 2: AND reg8 & reg9 = 0000_1200
Instruction 3: sub16bit reg8 - reg9 = 0000_1ba6
Instruction 4: mux8bit reg4 x reg5 = f9e8_db7c
Instruction 5: logical left shift 16bit (reg8) : 0000_2c94
Instruction 6: XOR reg2 xor reg 1 = 0000_fee8
Instruction 7: MAC8bit 0000_ffd1 + f9e8_db7c = f9e7_db4d
Instruction 8: NOT reg1: 0000_e77b
Instruction 9: arithmetic right shift 8bit (reg4): 0000_1326
Instruction 10: OR reg2 or reg1 = 0000_feec

Initial data stored in hex in registers is shown below:

Table 1: Initial data stored in registers

| Reg0 | Reg1 | Reg2 | Reg3 | Reg4 | Reg5 | Reg6 | Reg7 |
|---|---|---|---|---|---|---|---|
| 0000 | 1884 | e66c | 0000 | 274c | d885 | 0000 | 0000 |
| Reg8 | Reg9 | Reg10 | Reg11 | Reg12 | Reg13 | Reg14 | Reg15 |
| 164a | faa4 | 0000 | 0000 | 0000 | 1884 | 0000 | 0000 |

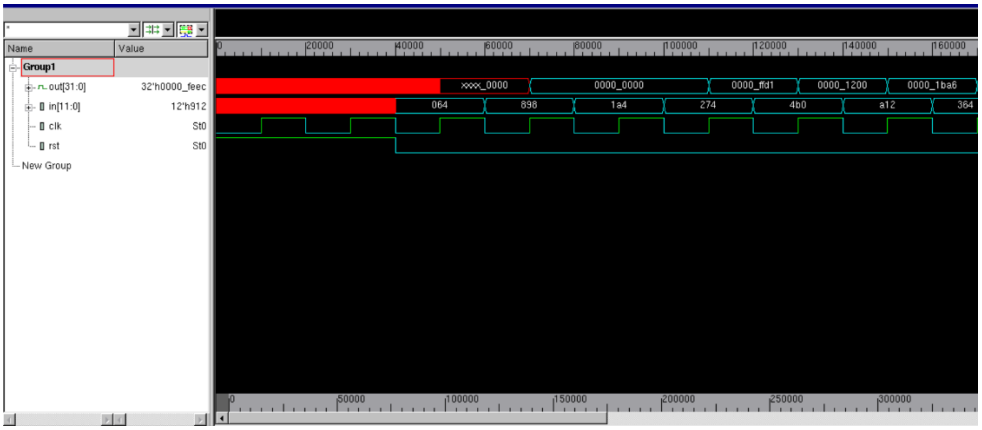



Figure 8. Behavior-level simulation result

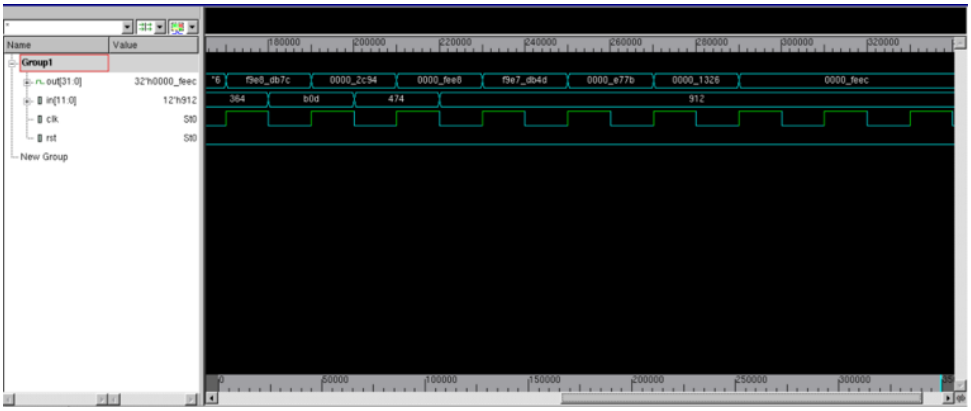



Figure 9. Behavior-level simulation result

## 2. Synthesis Simulation

Table 2: Synthesis of SIMD processor within 20ns

| 64-bit Ripple carry adder synthesized under timing constraint = 20ns | |
|---|---|
| Area | 8272.333969 um2 |
| Critical path | 19.82 ns |
| Leakage power | 90.2015 uW |
| Dynamic power | 125.4804 uW |
| Total power | 215.6818 uW |

## 3. Post-Synthesis Simulation


Figure 10. Post-synthesis simulation

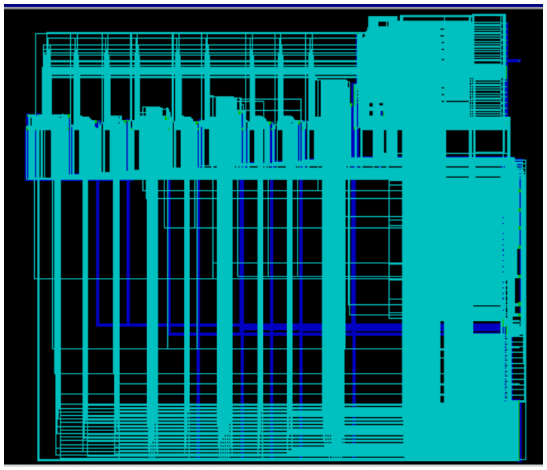
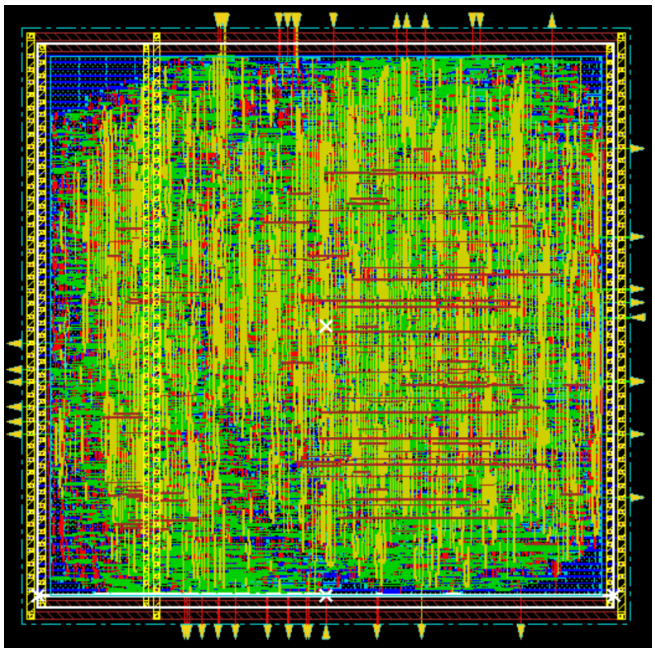


Figure 11. Post-synthesis schematic

## 4. Place and route



## 5. Post-Layout Simulation