Simple SIMD Processor

Report of Final Project

Sachin Kumar

Instructor: Assistant Professor G. Gopal

1- Introduction of Project

This project is to implement a simple SIMD processor, core of which is a 16-bit SIMD ALU. 2's compliment calculations are implemented in this ALU. The ALU operation will take two clocks. The first clock cycle will be used to load values into the registers. The second will be for performing the operations. 6-bit opcodes are used to select the functions. The instruction code, including the opcode, will be 18-bit.

The ALU will be embedded into a simple processor based on 5-stage, delay of each stage will be 1 cycle, meeting the delay of ALU, as shown in the figure below. The 5 typical stages are IF, ID, EX, MEM and WB, without pipeline. In the stage IF, a 10-bit address will be sent to an instruction Block-RAM (BRAM) to fetch 18-bit instruction. In the stage ID, the instruction will be decoded and some of control registers will be set to control the following stage. In the stage EX, ALU will process data in registers or implement some control commands, e.g. jump. In the stage MEM, if the instruction is "store" or "load", data would be read from/ written to data BRAM, based on instruction and address. Finally, in the stage WB, data will be written back to register. The pins of clock, reset, address, data and BRAM enable will be exposed on the interface of processor. The architecture of processor is shown in **Figure 1**.

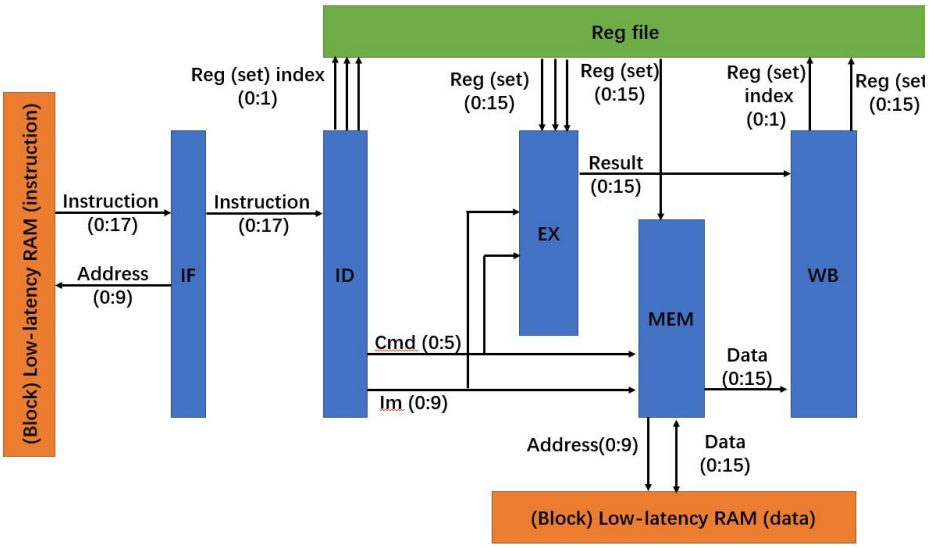
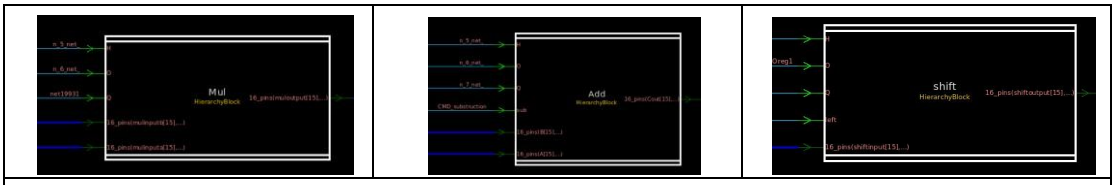


Figure 1: Simple SIMD Processor Architecture

2- Design of SIMD ALU

The ALU consists of three SIMD basic computation units: SIMD adder, SIMD multiplier and SIMD shifter. These three components can be reused for the computation of data with different width

(4-bit, 8-bit or 16-bit) and can handle all the instructions (listed in **Section 4**) in the design specification. Each of them is controlled by three input port, H (for 16-bit), O (for 8-bit) and Q (for 4-bit), indicating the kind of input data, so that the unit can handle the data properly, as shown below.



The Interface of ALU Components

1) SIMD adder:

The SIMD adder is implemented based on 4 4-bit adders. To reuse the adder for data with different width, the input signals, H, O and Q, control the forwarding of the carries between the adders. Moreover, the adder can also support subtraction, by flipping the second operand and add one to it when the signal sub is set. The data path of the SIMD adder is shown in **Figure 2.**

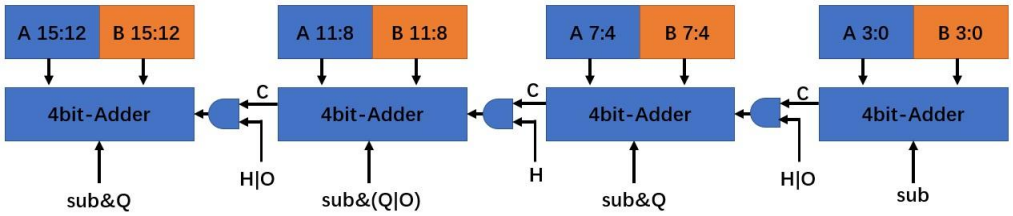


Figure 2: Datapath of SIMD Adder

2) SIMD shifter:

The SIMD shifter can also support data with different width. It is based on two 16-bit shifters and necessary overstep-correct logic. To illustrate the implementation of the shifter, the example based on logic-left-shift is shown in **Figure 3**. The shifter will determine whether the MSB of 4-bit block should be set to 0 or just inherit the value from the LSB from 4-bit block in front of it, according to the input signal, H and O.

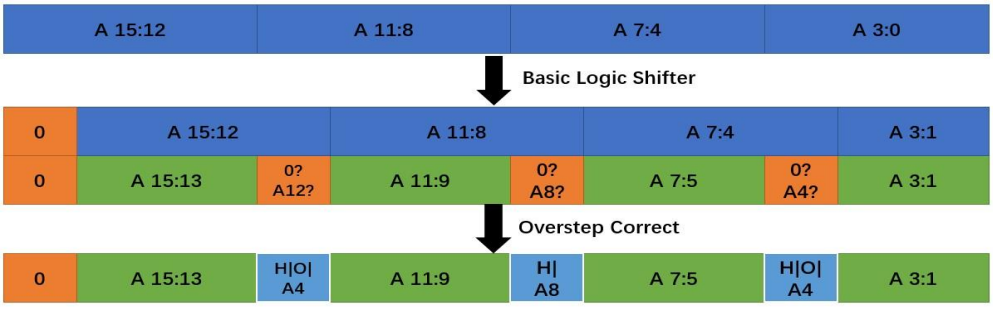


Figure 3: Example of SIMD Shifter

3) SIMD multiplier:

The SIMD multiplier is implemented with adders and shifters. To make it support multiplication with different data width, the input signal, H, O and Q, are used to control the input operands of the adders and select corresponding outputs for the target data width. The

1x16x16 multiplication is implemented as a typical multiplier, as **Figure 4**, where adder tree is utilized. The least significant 16 bits of the sum is the output of multiplication.

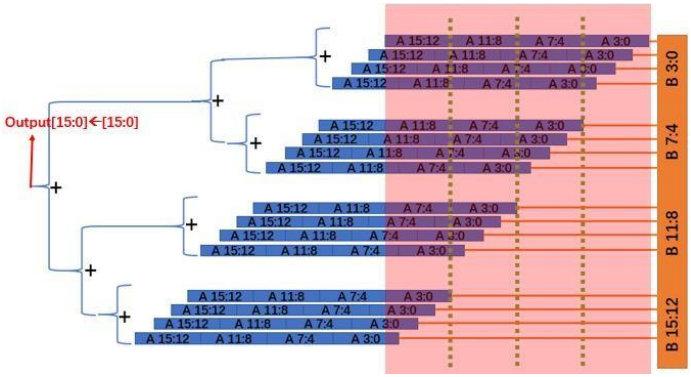


Figure 4: 1x16x16 multiplication

To reuse the structure of 1x16x16 multiplication implemented as above, for the input signal, H, O and Q, are used to select the inputs of the adders, to control the adders to only sum up the range of bits. The 4x4x4 multiplication is implemented as shown **Figure 5**, where for each adder, just a part of input bit is valid while other bits will be set to 0. Similarly, the 2x8x8 multiplication is shown in **Figure 6**.

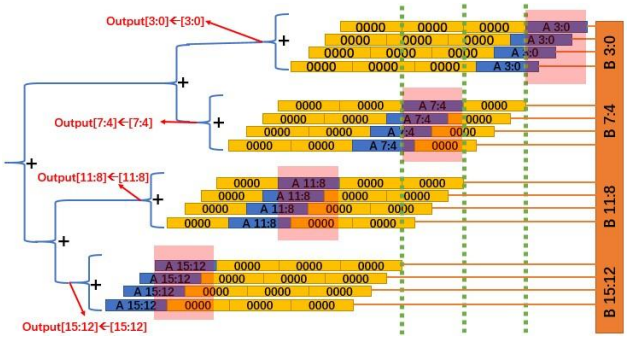


Figure 5: 4x4x4 multiplication

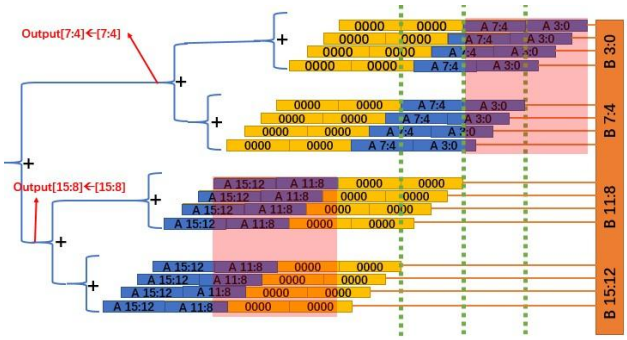


Figure 6: 2x8x8 multiplication

3- Register File

There are registers which are used to store the data for computation.

16-bit registers: H1, H2, H3, H4

8-bit registers (3 sets): {O1,O2}, {O3, O4}, {O5, O6}

4-bit registers (3 sets): {Q1, Q2, Q3, Q4}, {Q5, Q6, Q7, Q8}, {Q9, Q10, Q11, Q12}
Loop counter (for loop control): LC

4- Instruction Set

The following operations are supported by the processor:

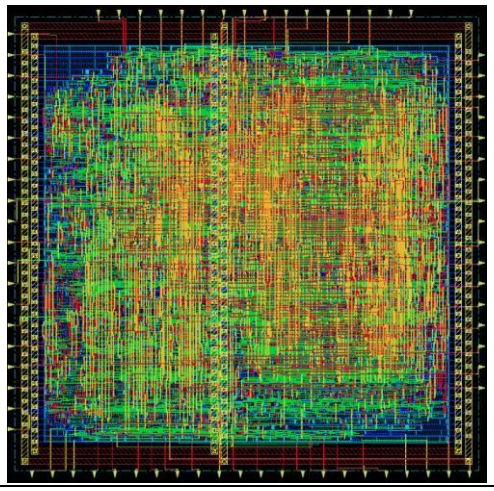
No.	Instruction	opcode(6bit)	Operand(12bit)
0	add16bit	000000	00000000 x1 x2 (x1,x2:2bit, select H1…4)
1	add8bit	000001	00000000 x1 x2 (x1,x2:2bit, select 2 sets from O)
2	add4bit	000010	00000000 x1 x2 (x1,x2:2bit, select 2 sets from Q)
3	add16im	000011	x1 im (x1:2bit, select H1..4, im:10bit)
4	add8im	000100	x1 00 im (x1:2bit, select 1 set from O, im:8bit)
5	add4im	000101	x1 000000 im (x1:2bit, select 1 set from Q, im:4bit)
6	sub16bit	000110	00000000 x1 x2 (x1,x2:2bit, select H1…4)
7	sub8bit	000111	00000000 x1 x2 (x1,x2:2bit, select 2 sets from O)
8	sub4bit	001000	00000000 x1 x2 (x1,x2:2bit, select 2 sets from Q)
9	sub16im	001001	x1 im (x1:2bit, select H1..4, im:10bit)
10	sub8im	001010	x1 00 im (x1:2bit, select 1 set from O, im:8bit)
11	sub4im	001011	x1 000000 im (x1:2bit, select 1 set from Q, im:4bit)
12	mul16bit	001100	00000000 x1 x2 (x1,x2:2bit, select H1…4)
13	mul8bit	001101	00000000 x1 x2 (x1,x2:2bit, select 2 sets from O)
14	mul4bit	001110	00000000 x1 x2 (x1,x2:2bit, select 2 sets from Q)
15	mul16im	001111	x1 im (x1:2bit, select H1..4, im:10bit)
16	mul8im	010000	x1 00 im (x1:2bit, select 1 set from O, im:8bit)
17	mul4im	010001	x1 000000 im (x1:2bit, select 1 set from Q, im:4bit)
18	mac16bit	010010	000000 x1 x2 x3 (x1,x2,x3:2bit, select H1…4)
19	mac8bit	010011	000000 x1 x2 x3 (x1,x2,x3:2bit, select 3 sets from O)
20	mac4bit	010100	000000 x1 x2 x3 (x1,x2,x3:2bit, select 3 sets from Q)
21	lsl16bit	010101	0000000000 x1 (x1:2bit, select H1…4)
22	lsl8bit	010110	0000000000 x1 (x1:2bit, select 1 set from O)
23	lsl4bit	010111	0000000000 x1 (x1:2bit, select 1 set from Q)
24	lsr16bit	011000	0000000000 x1 (x1:2bit, select H1…4)
25	lsr8bit	011001	0000000000 x1 (x1:2bit, select 1 set from O)
26	lsr4bit	011010	0000000000 x1 (x1:2bit, select 1 set from Q)
27	and16bit	011011	00000000 x1 x2 (x1,x2:2bit, select H1…4)
28	and8bit	011100	00000000 x1 x2 (x1,x2:2bit, select 2 sets from O)
29	and4bit	011101	00000000 x1 x2 (x1,x2:2bit, select 2 sets from Q)
30	or16bit	011110	00000000 x1 x2 (x1,x2:2bit, select H1…4)
31	or8bit	011111	00000000 x1 x2 (x1,x2:2bit, select 2 sets from O)
32	or4bit	100000	00000000 x1 x2 (x1,x2:2bit, select 2 sets from Q)

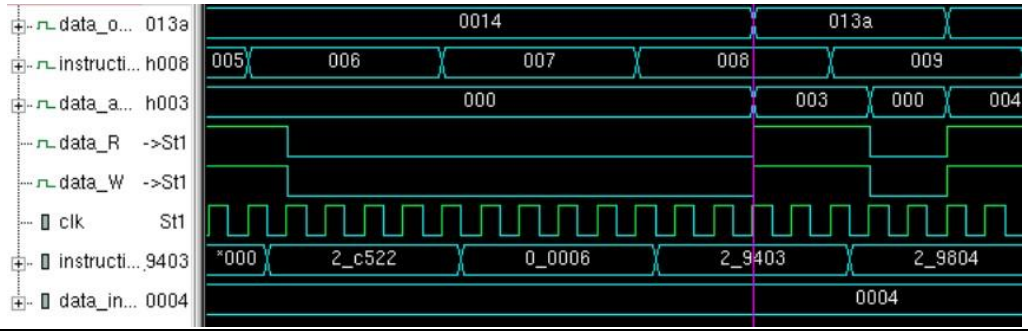
33	not16bit	100001	0000000000 x1 (x1:2bit, select H1…4)
34	not8bit	100010	0000000000 x1 (x1:2bit, select 1 set from O)
35	not4bit	100011	0000000000 x1 (x1:2bit, select 1 set from Q)
36	loopjump	100100	00 im (im:10bit, address)
37	setloop	100101	00 im (im:10bit, LC value)
38	load16bit	100110	x1 im (x1:2bit, select H1..4, im:10bit-address)
39	load8bit	100111	x1 im (x1:2bit,, select 1 set from O, im:10bit-addr)
40	load4bit	101000	x1 im (x1:2bit,, select 1 set from Q, im:10bit-addr)
41	store16bit	101001	x1 im (x1:2bit, select H1..4, im:10bit-address)
42	store8bit	101010	x1 im (x1:2bit,, select 1 set from O, im:10bit-addr)
43	store4bit	101011	x1 im (x1:2bit,, select 1 set from Q, im:10bit-addr)
44	set16bit	101100	x1 im (x1:2bit, select H1..4, im:10bit)
45	set8bit	101101	x1 00 im (x1:2bit,, select 1 set from O, im:8bit-value)
46	Set4bit	101110	x1 000000 im (x1:2bit,, select 1 set from Q, im:4bit-value)
47	halt	111111	000000000000

5- Cost and Performance of the SIMD processor

With Synopsys VCS and Cadence Encounter, the Verilog code of processor are synthesized, and, after placement and routing, related statistics are collected as below. The post-layout simulation is also conducted.

Timing constraint	20 ns
Area	5585.734 um2
Critical path	10.334 ns
Leakage power	6.927448e+4 nW
Dynamic power	261.161 uW
Total power	330.436 uW



Layout

Post-Layout Simulation