International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:4, No:8, 2010

A Processor with Dynamically Reconfigurable Circuit for Floating-Point Arithmetic

Yukinari Minagi , Akinori Kanasugi

***Abstract*—**This paper describes about dynamic reconfiguration to miniaturize arithmetic circuits in general-purpose processor. Dynamic reconfiguration is a technique to realize required functions by changing hardware construction during operation. The proposed arithmetic circuit performs floating-point arithmetic which is frequently used in science and technology. The data format is floating-point based on IEEE754. The proposed circuit is designed using VHDL, and verified the correct operation by simulations and experiments.

***Keywords*—**dynamic reconfiguration, floating-point arithmetic, double precision, FPGA

I. Introduction

RECENTLY, development of embedded processors is toward miniaturization and energy saving for ecology. On the other hand, high performance arithmetic circuits are required in a lot of application in science and technology. Dynamically reconfigurable processors have been developed to meet these requests. They can change circuit configuration according to instructions in program instantly during operations. This technology constructs required functions with minimum area [1]-[3].

This paper describes about dynamic reconfiguration to miniaturize arithmetic circuits in general-purpose embedded processor. We reconfigure floating-point arithmetic circuits. The proposed dynamically reconfigurable circuit is reconfigured two modes as described by the following:

(i) One double precision floating-point arithmetic,
(ii) Two parallel operations of single precision floating-point arithmetic.

We have designed an embedded processor which reduces implementation area by reconfiguring common parts of each operation [4]-[6]. The proposed processor conforms to the instruction set of "MIPS-I" that is a typical 32-bit microprocessor [7]. We evaluate implementation area and processing speed for floating-point arithmetic. Then we show effectiveness of the proposed circuit.

Y. Minagi is with Graduate School of Engineering, Tokyo Denki University, Tokyo, Japan (e-mail: 10kme63@ms.dendai.ac.jp)
A. Kanasugi is with the Department of Electrical and Electronic Engineering, Tokyo Denki University, Tokyo, Japan (e-mail: kanasugi@eee.dendai.ac.jp).
This work was supported by Tokyo Denki University Science Promotion Fund (Q06J-03)

II. Dynamic Reconfiguration

Conventional circuits require a lot of clocks for reconfiguration. As a result, time of few *milli* seconds order is required and speed-up is prevented. On the other hand, dynamic reconfiguration changes circuit construction at one clock cycle during operation without stopping circuits. Thus, dynamic reconfiguration enables reconfiguration of circuits in a few *nano* seconds.

In this paper, a dynamically reconfigurable circuit for floating-point arithmetic is proposed. The arithmetic circuit consists of two single precision floating-point arithmetic circuits. As shown in Fig. 1, the proposed circuit performs double precision floating-point arithmetic by reconfiguration [4].

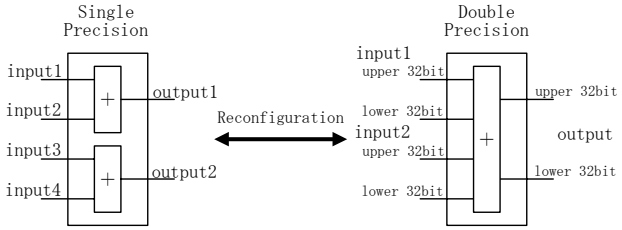

Fig. 1 Dynamic reconfiguration of arithmetic circuit

III. Floating-Point based on IEEE754 format

We use IEEE754 format which is adopted most widely at the calculation of the floating-point [8]. It decides expression format, particular value, rounding method, precision, and so on.

Fig. 2 shows expression format. The detail format is shown in Table I.

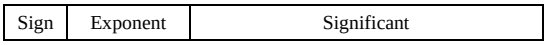

Fig. 2 Floating-point format

TABLE I
Bit width of each precision

Symbol	Single precision	Double precision
Total	32 bit	64 bit
Sign	1 bit	1 bit
Exponent	8 bit	11 bit
Significant	23 bit	52 bit

The floating point number is converted into a binary number as follows (single precision),

$$(-1)^C \times 1.S \times 2^{E-127} \quad \text{(single precision)} \qquad (1)$$

$$(-1)^C \times 1.S \times 2^{E-1023} \quad \text{(double precision)} \qquad (2)$$

where, C is sign part, E is exponent part, and S is significant part. The treatments in this paper are shown below.

-Rounding process uses Unbased.
(Round to nearest even digit in the required position)
- Un-normalized number are not used.
- Exception processes are overflow, underflow, infinity, and NaN.

IV. Dynamically Reconfigurable Floating-Point Arithmetic Circuit

The floating-point arithmetic circuits contain three operations; sign part, exponent part, and significant part. In addition, normalization, rounding and exception handling are necessary. In this paper, the proposed circuit contains four floating-point arithmetics; addition, subtraction, multiplication, and division. Each circuit reconfigures two modes as indicated by the following:

(i) One double precision floating-point arithmetic,
(ii) Two parallel operations of single precision floating-point arithmetic.

The reconfiguration of circuit is controlled by "*fmt*" (format signal). The input "*fmt*" selects single precision (*fmt* = '0') or double precision (*fmt* = '1').

A. Floating-point adder and subtracter

Fig. 3 shows the floating-point adder and subtracter. This circuit calculates

$$Input\_A + Input\_B = Output,$$

or

$$Input\_A - Input\_B = Output.$$

The input "*sel*" selects addition (*sel* = '0') or subtraction (*sel* = '1'). The symmetrical circuit blocks perform single precision arithmetic in parallel. Two right shift circuits located at the center operate as one double precision circuit.

B. Floating-point multiplier

Fig. 4 shows the floating-point multiplier, which calculates

$$Input\_A \times Input\_B = Output.$$

The adder and register located at the center add partial products of single precision for double precision. Two clock cycles are necessary for double precision arithmetic.

C. Floating-point divider

Fig. 5 shows the floating-point divider. This circuit calculates

$$Input\_A / Input\_B = Output.$$

This divider uses the recovery method.

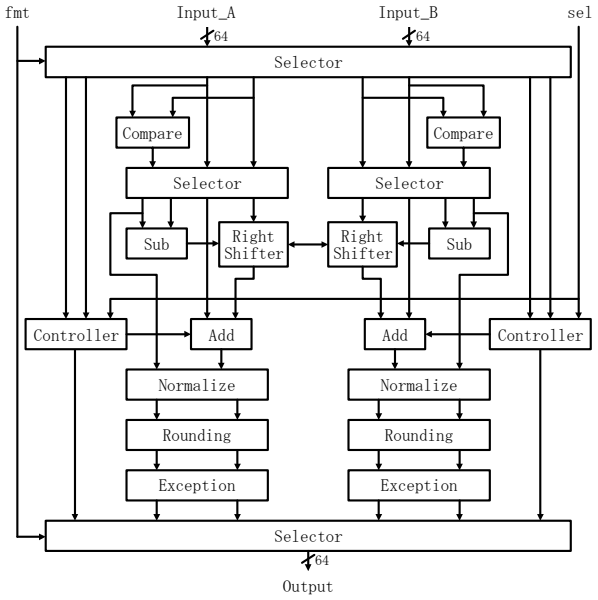

Fig. 3 Floating-point adder and subtracter

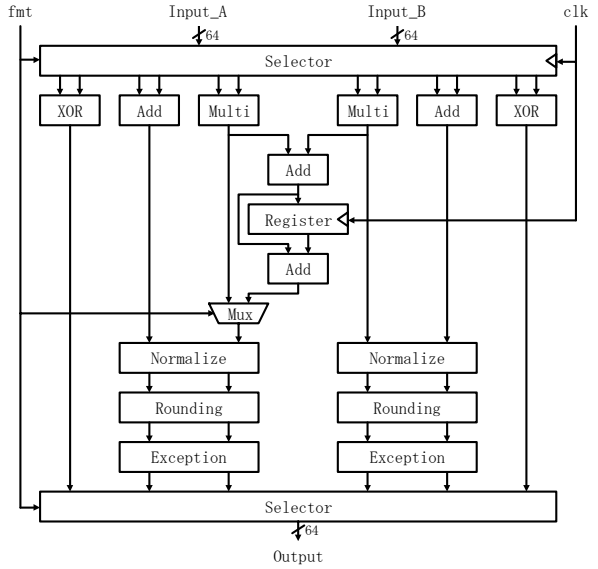

Fig. 4 Floating-point multiplier

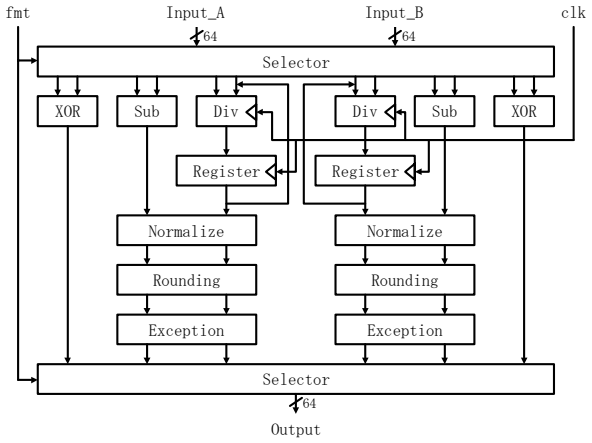

Fig. 5 Floating-point divider

V. Processor Architecture

The instruction set architecture of the proposed processor conforms to MIPS-I (MIPS Technologies, Inc.) subset that is a typical microprocessor [7]. MIPS-I has 32-bit RISC architecture. The pipeline consists of five stages (IF (instruction fetch), ID (instruction decode), EX (execution), MEM (memory access), and WB (write back)). The registers are PC (program counter), 32 GPR (general purpose register), and HI/LO (multiplication and division register). The size of instruction memory and data memory is 1 (KB). The integer arithmetic circuit has ALU (arithmetic logical unit) and MDU (multiplication and division unit).

In addition, it has two CP (coprocessors). CP0 controls status management, exception, and interrupt. CP1 controls floating-point arithmetic. CP1 controls dynamically reconfigurable arithmetic circuit designed in this paper. Fig. 6 and Fig.7 show the block diagram of processor and co-processor, respectively. The "DR FPU" in Fig.7 means the dynamically reconfigurable floating-point unit.

Table II summarizes the implemented instructions which cover 80% of instruction set for MIPS-I processor.

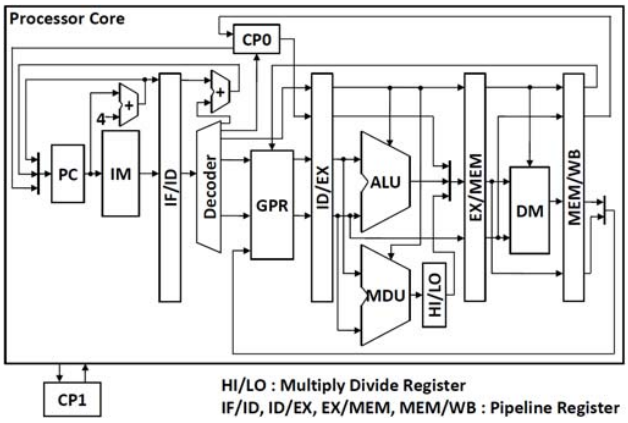

Fig. 6 Block diagram of processor core

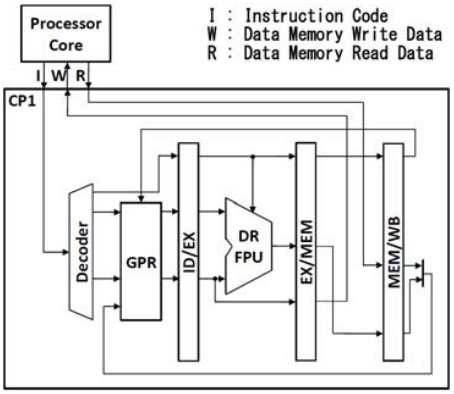

Fig. 7 Block diagram of floating-point co-processor

TABLE II
Instruction List

Format	Instruction
R (Register)	ADD, ADDU, SUB, SUBU, MULT, MULTU, DIV, DIVU, SLT, SLTU, AND, OR, NOR, XOR, MFHI, MFLO, MTHI, MTLO, SLL, SLLV, SRL, SRLV, SRA, SRAV, JR, JALR
I (Immediate)	LW, SW, LB, LBU, SB, LH, LHU, SH, BEQ, BNE, BLEZ, BGEZ, BLTZ, BGTZ, BGEZAL, BLTZAL, ADDI, ADDIU, SLTI, SLTIU, ANDI, ORI, XORI
J (Jump)	J, JAL
floating-point	LWC1, SWC1, ADD.fmt, SUB.fmt, MUL.fmt, DIV.fmt, ABS.fmt, MOV.fmt, NEG.fmt, C.EQ.fmt, C.LT.fmt, C.LE.fmt, BC1T, BC1F

(fmt: S = Single-precision, D = Double-precision, P = Pair Single-precision)

VI. Verification

A. Simulation

The dynamically reconfigurable arithmetic circuit was synthesized using ISE11.2 CAD software (Xilinx Inc.). It was simulated using ModelSim XE III 6.4b logic simulator (Mentor Graphics Corp.). The circuit was verified with a sample program that solves simultaneous linear equation by Gauss-Jordanian method in double precision floating-point.

In the simulation, the following simultaneous linear equations are solved.

$$\begin{aligned} 2x + 3y + z &= 4 \\ 4x + y - 3z &= -2 \\ -x + 2y + 2z &= 2 \end{aligned} \tag{3}$$

Then, the following solution matrix is stored in data memory.

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 3 \end{bmatrix} \tag{4}$$

The results in double precision floating-point format are obtained as shown in table 3. For example,

$$3FF0\ 0000\ 0000\ 0000_{(16)} \quad (5)$$

equals

$$0011\ 1111\ 1111\ 0...0_{(2)} \quad (6)$$

Then, we have

$$C = 0 \quad (7)$$

$$E = 011\ 1111\ 1111_{(2)} = 1023_{(10)} \quad (8)$$

$$S = 0 \quad (9)$$

Therefore

$$(-1)^0 \times 1.0 \times 2^{1023-1023} = 1 \quad (10)$$

Fig.8 shows the part of sample program. The object program size is 456 byte.

Fig.9 shows the simulation result. The result showed that the value of data memory is equal to the solution in Table III. Thus, the correct operations of proposed circuit were verified.

TABLE III
FLOATING-POINT VALUE OF SOLUTION

row	column	value	Floating-point (hex)
1	1	1.0	3FF00000 00000000
	2	0.0	00000000 00000000
	3	0.0	00000000 00000000
	4	2.0	40000000 00000000
2	1	0.0	00000000 00000000
	2	1.0	3FF00000 00000000
	3	0.0	00000000 00000000
	4	-1.0	BFF00000 00000000
3	1	0.0	00000000 00000000
	2	0.0	00000000 00000000
	3	1.0	3FF00000 00000000
	4	3.0	40080000 00000000

```
0000        LW      r25, 0x7FFC(r0)     # t1 = *(0x7FFC);
0004        ADDI    r21, r0,  4         # y = 4;
0008        ADDI    r22, r0,  8         # x = 8;
000C        ADD     r1,  r0,  r0        # k = 0;
0010    L1: MOV.D   f2,  f0             # max = 0.0;
0014        ADD     r2,  r1,  r0        # s = k;
0018        ADD     r3,  r1,  r0        # j = k;
001C        MULT    r1,  r21            # LO = k * y;
0020        MFLO    r20                 # z = LO;
0024        ADD     r4,  r20, r0        # l = z;
0028    L2: ADD     r5,  r4,  r1        # m = l + k;
002C        MULT    r5,  r22            # LO = m * x;
0030        MFLO    r5                  # m = LO;
0034        LWC1    f4,  4(r5)          # FPR4 = a[m];
0038        LWC1    f5,  0(r5)          #
003C        ABS.D   f6,  f4             # dumy = fabs(FPR4);
0040        C.LT.D  f2,  f6             # if (max < dumy)
0044        BC1F    3                   #
0048        NOP                         # // branch delay slot
004C        MOV.D   f2,  f6             #     max = dumy;
0050        ADD     r2,  r3,  r0        #     s = j;
0054        ADDI    r4,  r4,  4         # l = l + 4;
0058        ADDI    r3,  r3,  1         # j++;
005C        SLTI    r6,  r3,  3         # if (j < 3)
0060        BNE     r6,  r0,  -15       #     goto L2;
0064        NOP                         # // branch delay slot
0068        C.EQ.D  f2,  f0             # if (max == 0.0)
006C        BC1T    82                  #     goto ER:
0070        NOP                         # // branch delay slot
0074        MULT    r2,  r21            # LO = s * y;
0078        MFLO    r5                  # m = LO;
```

Fig. 8 A part of sample program for verification

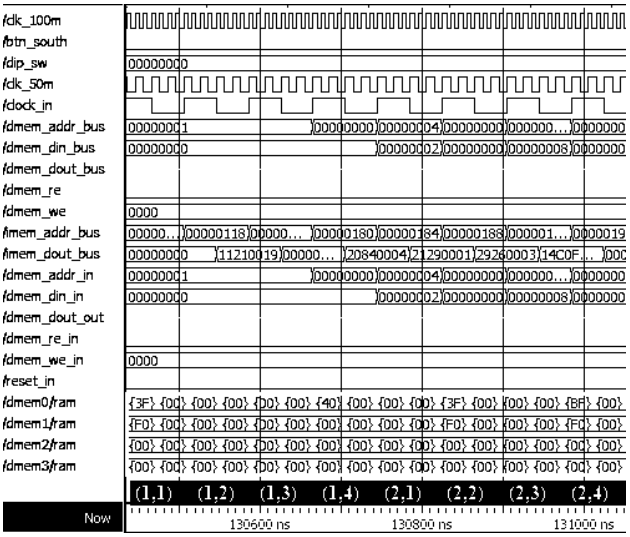

Fig. 9 Simulation result

B. Experiments

The actual operations were verified by experiments. Fig. 10 shows the block diagram for experiments. The verification program (Fig. 8) was assigned on a VHDL source as initial value of memory.

The processor was implemented on Xilinx Virtex-4 (XC4VLX25) ML401 evaluation platform [9] (Fig. 11) using Xilinx ISE 11.2 CAD software. The calculation results are displayed in LCD (Liquid Crystal Display) on the evaluation platform. The memory display is changed by the DIP switch (dip_sw).

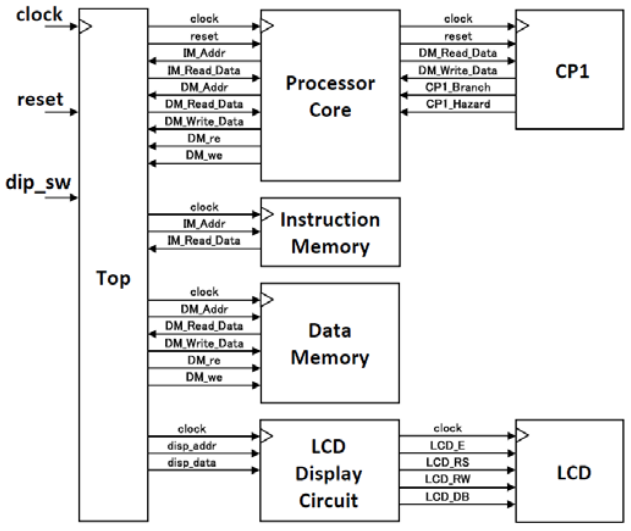

Fig. 10 Block diagram for experiments

Fig. 12 shows the first row of solution matrix as an example of experimental results. The result is equal to the simulation result of Fig. 9. Similarly, other results were verified.

Table IV shows the circuit size of proposed processor and the number of total clock cycles of verification program. The conventional floating-point arithmetic circuit was measured for the comparison. Table IV showed that the proposed circuit was reduced the circuit size more than the conventional circuit, with a very little number of clocks (1.1%).

VII. Conclusion

This paper proposed a general-purpose processor with dynamically reconfigurable floating-point arithmetic. The correct operation was verified by simulations and experiments. In addition, the implementation result showed that the proposed circuit could reduce the circuit size, with a very little number of clocks.

The future work is a design of dynamically reconfigurable VLIW processor.

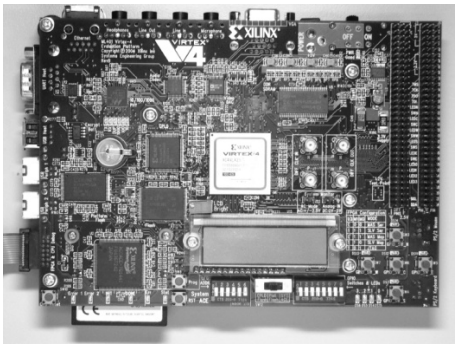

Fig. 11 Evaluation Platform

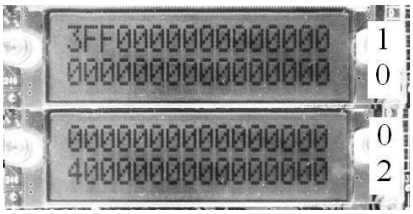

Fig. 12 First row of solution matrix

TABLE IV
Logic synthesis results of processor

	Static configuration	Dynamic reconfiguration	Reduction rate
Slice	8,459	8,009	-5.3 %
Flip Flop	3,347	3,221	-3.8 %
4-inputs LUT	15,435	14,617	-5.3 %
DSP48	29	16	-44.8 %
Clock Cycles	1,628	1,646	+1.1%

References

[1] T. J. Todman, et al, "Reconfigurable computing: architectures and design methods", *IEE Proc.-Computers & Digital Techniques*, vol. 152, no. 2, pp. 193 - 207, 2005.
[2] T. Sato, H. Watanabe, K. Shiba, "Implementation of dynamically reconfigurable processor DAPDNA-2", *VLSI Design, Automation and Test, 2005 IEEE VLSI-TSA International Symposium*, pp. 323-324, 2005.
[3] M J. Myjak, J. G. Delgado-Frias, "A Medium-Grain Reconfigurable Architecture for DSP: VLSI Design, Benchmark Mapping, and Performance", *IEEE Trans. on VLSI Systems*, vol.16, no.1, pp.14-23, Jan 2008.
[4] H. Shimada, Y. Hayakawa and A. Kanasugi, "An Architecture of Dynamically Reconfigurable Arithmetic Circuit", *Proc. of 2009 Int. Conf. on Electronics Packaging*, pp.963-966, 2009.
[5] A. Kanasugi, A. Tsukahara, "A Processor for Genetic Algorithm using Dynamically Reconfigurable Memory", *Journal of Convergence Information Technology*, vol.2, no.1, pp. 4-15, 2007.
[6] A. Tsukahara, A. Kanasugi, "Genetic Algorithm with Dynamic Variable Number of Individuals and Accuracy", *International Journal of Control, Automation, and Systems*, vol. 7, no. 1, pp.1-6, 2009.
[7] MIPS Technologies, Inc., *MIPS32 Architecture for Programmers Volume I: Introduction to the MIPS32 Architecture*, July 1, 2005.
[8] IEEE Std. 754 - 1985, *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE, 1985.
[9] Xilinx, Inc., *ML401/ML402/ML403 Evaluation Platform User Guide UG080 (v2.5)*, May 2006

Yukinari Minagi was born at Yamaguchi in March 20, 1973. He received the B.E. from Tokyo Denki University, Japan, in 2010.

He is currently a master course student in Graduate School of Engineering, Tokyo Denki University. His research interests include VLSI design for dynamically reconfigurable circuits.

Mr. Minagi is a member of the Institute of Electronics, Information and Communication Engineers.

Akinori Kanasugi was born at Tokyo in July 2, 1960. He received the B.E., M.E. and Ph. D from Saitama University, Japan, in 1983, 1985 and 1994, respectively.

After a research associate in Saitama University, he moved to Tokyo Denki University in 2002, where he is currently a professor in the faculty of engineering. His current research interests are in the development of VLSI systems such as reconfigurable processor, GA processor, and rough sets processor.

Prof. A. Kanasugi is a member of the Institute of Electronics, Information and Communication Engineers, Information Processing Society of Japan, the Japanese Society for Artificial Intelligence, and Japan Institute of Electronics Packaging.