The Design and Implementation of CalmRISC32 Floating Point Unit

Cheol-Ho Jeong*, Woo-Chan Park*, Sang-Woo Kim and Tack-Don Han***

* Department of Computer Science, Yonsei University, Seoul 120-749 Korea
**MCU Team, System LSI Division, Samsung Electronics Co., Yong-In, Korea
*E-mail: chjeong@kurene.yonsei.ac.kr

Abstract

The CalmRISC32 FPU is RISC style coprocessor for embedded system. It supports IEEE-754 standard single precision addition/subtraction, floating-point multiplication, floating-point division, format conversion, comparison, rounding, load/store, and etc. It also supports four rounding modes, and precise exception. It can execute and complete instructions out of order if some constraint is resolved – data dependency, resource conflict, and exception prediction. Standard cell base design technique is used to save design time and cost. First prototype is running at about 70Mhz with worst-case delay in gate level simulation.

I. INTRODUCTION

Currently, embedded system is hailed by major semiconductor and mobile device manufacturer. They need simple, light, and low power micro-controller, not high performance microprocessor. Obviously current design goal is low-power and higher performance within given constraints. And both on-chip and off-chip configuration of peripheral device must be possible for various market demands.

CalmRISC32 FPU is design for embedded system based on above characteristics. It is RISC type coprocessor, and on-chip or off-chip configuration is possible with host processor. It supports IEEE-754 single precision data type, four rounding mode, and precise exception. It has five separate pipelines, and optimized for fast floating-point addition/subtraction, floating-point multiplication, and floating-point comparison. Also, coprocessor instructions can be executed simultaneously in all pipelines and can be completed out of order.

In general, floating-point operation latencies are varied by arithmetic instructions that executed [1]. Therefore, it is general to adjust all operation latencies to longest pipeline latency. But, In CalmRISC32 FPU, out of order execution and completion control scheme is designed to achieve high performance. Scoreboarding and tomasulro's algorithm can be a candidate to support out of order execution/completion [2]. But, design cost and complexity of these techniques is too high for micro-controller. Therefore, constraints base dynamic scheduling is used with data dependency checking, resource conflict checking, and exception prediction technique. With this technique CalmRISC32 FPU can perform instructions out of order execution/completion. And exception prediction technique eliminates special hardware unit – reorder buffer or reservation station. All operands of arithmetic instructions are checked for exception in the first stage of pipeline, and if exception can be happen, then coprocessor executes instruction in order to handle exceptional condition properly, otherwise coprocessor performs instruction out of order.

CalmRISC32 FPU is implemented with standard-cell library to save implementation time and cost. It supports floating-point addition/subtraction, floating-point multiplication, floating-point division, format conversion, comparison, rounding, load/store, and etc. hard-macro block is used for large conventional block—fraction multiplier, adder, and subtractor. And it reduces design time and verification effort.

The remainder of this paper organized as follows. Section 2 shows the architecture of CalmRISC43 FPU and section 3 describes the coprocessor interface to host processor. Design methods and implementation scheme is explained in section 4. Finally, section 5 describes conclusions.

II. CalmRISC32 FPU ARCHITECTURE

The CalmRISC32 FPU is a 32bit RISC type co-processor that executes floating-point operation with the support of CalmRISC32 microprocessor. It was designed for micro-controller that is used for embedded system. And, It can be applied to high-speed floating-point number calculation, signal processing and 3D graphics application with a multiple FPU. It is composed of hardware floating-point ALU (Arithmetic and Logic Unit), floating-point multiplier, and floating-point divider. And, It has independent instruction decoder, load/store unit, register file and co-processor interface unit. Therefore, CalmRISC32 FPU can be included, or excluded along with application domain. It can execute several instructions simultaneously within some constraint and complete instruction out of order if arithmetic exception is not generated by the instruction.

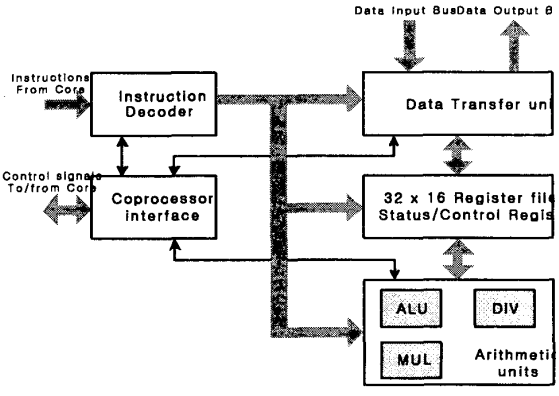

Fig. 1. Block diagram of CalmRISC32 FPU

CalmRISC32 FPU supports 32bit single precision floating-point addition, subtraction, multiplication, division, comparison, type conversion and rounding operations. And, IEEE-754 standard four rounding mode (Round to zero, round to nearest, and round to negative/positive infinity) and exception is supported. It has greatly reduced interrupt recovery mechanism

0-7803-6470-8/00/ $10.00 © 2000 IEEE

with the simple co-processor interface and exception prediction technique. And it has 16 ? 32bit register file, one status/control register, and one exception register. A rounding mode, exception type, and comparison result are stored into these special registers.

Data transfer from/to host processor (CalmRISC32) is made by data transfer unit under the control of co-processor interface unit. To reduce design complexity of co-processor, it has not instruction fetch unit and memory address generation unit. And, co-processor cannot independently access memory. Therefore, host processor takes responsibility of instruction fetch and address generation and data preparation for memory read or write operation.

Instruction is first fetched and pre-decoded by host processor. If pre-decoded instruction is co-processor instruction, the instruction code is transferred directly to co-processor with several control signals. And, then co-processor decoded the received instruction and executed appropriate operation. For load/store operation, host processor continues instruction execution for memory address generation and data that stored. Co-processor only has responsibility of data preparation and data transfer to host processor through the data bus.

Generally, the most frequent floating-point operation is floating-point addition/subtraction. And the floating-point multiplication is in the second place. Therefore, design effort is focused on the fast floating-point addition/subtraction unit and floating-point multiplication unit. For the fast program execution dedicated floating-point comparison unit is included in the floating-point ALU and it can complete floating-point comparison operation with in one clock cycle. And miscellaneous operations (register move, absolute, negation and etc.) are executed in the separate pipeline unit with one clock cycle latency.

A. FPU Execution Pipeline

Figure 2 shows pipeline diagram of CalmRISC32 FPU. CalmRISC32 FPU has five separate pipeline paths – floating-point ALU pipeline (FALU), floating-point multiplication pipeline (FMUL), floating-point division pipeline (FDIV), load/store pipeline (FLDST), and miscellaneous pipeline (Misc.). As shown in figure 2, those pipelines have different operation latencies, and all pipeline except FDIV are fully pipelines. And first stage of FDIV pipeline has iterative path, which has 15-latencies. With the in-order issue and in-order completion control scheme pipeline resource is greatly wasted because another instruction cannot be issued before current instruction is being executed. To full use of these pipelines, simple dynamic instruction scheduling is used. This dynamic scheduling can be archived by resource conflict checking in the write-back stage (FW), data dependency checking in the decode stage (FD), and exception prediction in the first stage of the each arithmetic pipelines (FDIV, FMUL, and FALU). Therefore, the host processor can continues instruction issues to co-processor until data dependency and resource conflict is founded. Issued instructions are executed simultaneously in those pipelines and complete operation out of order if resource conflict is resolved and other pipeline does not generate exception prediction signal. If exception prediction signal is generated, the host processor stops instruction issues until the instruction that generates exception prediction signal finishes its execution. If exception prediction is false, host processor continues the program execution.

In the FW stage, if two or more write-back data is available, only one write-back data is selected from these five pipelines with the predefined priority and generated pipeline stall signal to other pipeline. And in the next clock cycle the other write-back data is advanced to the FW stages.

Co-processor catches pipeline stall conditions in the case of data dependency and resource conflict. If stall conditions are founded, it generates appropriate control signals to stall host processor and stop instruction issues.

B. Floating-point ALU

Floating-point ALU pipeline is composed of floating-point addition/subtraction unit, comparison unit and exception prediction unit. It can handles floating-point addition, subtraction, type conversion, rounding, comparison operation. Generally floating-point addition/subtraction takes four processing steps – alignment, fraction addition/subtraction, normalization, and rounding. It needs addition fraction adder for rounding and increase processing time and area. It causes renormalization step by the overflow in rounding processing. In order to reduce that renormalization overhead, parallel-rounding algorithms is implemented [4]. With this algorithm fraction addition/subtraction and rounding is executed simultaneously in the second pipeline stage. Also it supports IEEE standard four rounding mode. Therefore, It can perform floating-point addition/subtraction and the other ALU operations within three clock cycles – rounding operation and format conversion operation (Integer to floating-point or floating-point to integer). In addition, it does not need renormalization step because rounding takes place before normalization, and additional adder is eliminated.

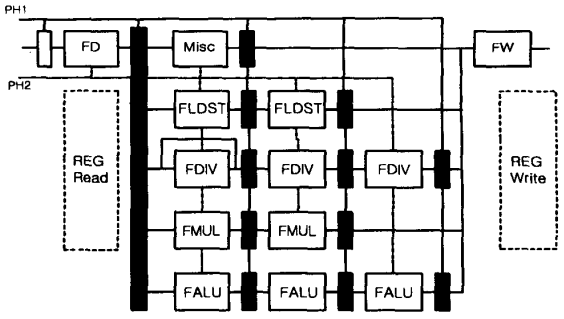

Fig. 2. Pipeline diagram of CalmRISC32 FPU

For the exception prediction the exponents of two operands are examined in the first stage. Exponent addition or subtraction is performed according to the ALU operation, and invalid number checking of input operands is executed. If one of input operands is invalid format, and calculation of the two operands may cause exception, ALU asserts exception prediction signal to prevent further instruction issue by host processor. The last ALU pipeline stage ascertains the truth of arithmetic exception by the status/control register setting values to serve an exception handing. And setting of status/control register value to all zero can ignore the exception generated by the arithmetic pipeline.

C. Floating-Point MUL

Floating-point MUL pipeline has two stages. In the first stage floating-point fraction multiplication and addition of partial production is performed. And in the next stage, fraction rounding and normalization is executed. In order to save design time and effort, integer multiplier hard macro in target library is applied and it was designed with the conventional floating-point multiplication steps [5]. In addition, in the first stage the exponents of two operands are examined for exception prediction. If one of the two operand's exponent and fraction is invalid number format (Not a Number, Infinity number or De-normalized number), or the addition of the two exponents may cause overflow or underflow exception, the exception prediction signal is generated to stop issuing instruction according to status/control register setting. The exact exception signal is generated in the last stage of FMUL pipeline to process exception handling.

D. Floating-Point DIV and Load/Store

Floating-point DIV pipeline has iterative first stage and non-iterative the other stages. In the first stage, radix-4 SRT division algorithm is used for implementation [6], [7]. In the second stage quotient addition is performed, and rounding and normalization step is executed in the last stage. Also the exponents of two operands are examined for exception prediction in the first stage. If one of the two operand's exponent and fraction is invalid number or zero divisor (Division by Zero), or the subtraction of the two exponents may cause overflow, underflow exception, the exception prediction signal is generated to stop issuing instruction. In the last stage of FDIV pipeline exact exception signal is generated according to status/control register setting to process exception handling.

Load/Store pipeline (FLDST) has two stages comply with memory access stage (MEM) in host processor. In the first stage of FLDST, there is any operation, but in the next stage data read or write operation is executed thought the data input bus or output bus. For multiple cycle load/store instruction or pipeline control signal is used to stall co-processor. In the next section explains host and co-processor interface mechanism.

Misc. pipeline can executed register move, absolute value, negation value and constant (0.0 or 1.0) load operation.

III. COPROCESSOR INTERFACE

To deduce the design complexity and effort of coprocessor exception recovery mechanism and memory access unit is excluded. But, a simple host – coprocessor interface unit and some constraints, can support these services. That is, instruction stream is scheduled with a host processor and interface signals. Instructions are issued in order by host processor, but instruction completions can be done out of order by the operation latency. Coprocessor and core processor need to be synchronized on some cases – instruction issue may not be allowed since coprocessor suffer from stalls and can not get more instruction from core processor, core processor is stalled and can not provide data for coprocessor data transfer instruction, etc. Therefore, co-processor and host processor have to synchronize with each other special control signals – STXEN, STMEN, STWEN, COPXEN, COPMEN, and COPWEN. These signals active in low state, "ST" mean host processor status, and "COP" means co-processor status. And, "X", "M" and "W" are stand for execute stage, memory stage, "M" and "W" are stand for execute stage, memory stage, and write-back stage, respectively. The last letters "EN" stand for enable. That is, if STMEN is high, it means that the instructions in the host processor can advanced to next memory stage. If it goes to low state, the instruction in the host processor can not advanced to the next stage. And if COPXEN is in the low state, it means that some pipeline stall conditions (data dependency, resource conflict, and pipeline full) are happened, therefore, instruction issues must be stop until COPXEN goes to high. Like this, instruction issue control is archived with these control signals.

In the coprocessor decode stage decodes instructions from host processor and checks dependency. COPXEN signals generated to stop instruction issuing if data dependency happens. As soon as data dependency is resolved, COPXEN signal goes to high.

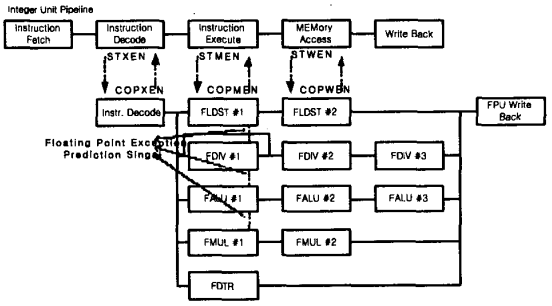

Fig. 3. Host and co-processor interface signals

Two resource conflicts can be happens in the coprocessor. First, resource conflict can happens if two or more pipeline attempts to advance to the coprocessor write-back stage, because only one write-back data can advance to write-back stage. And the other instructions in the pipelines must wait until next clock cycles. Write-back data selection is scheduled with the priority. If one pipeline is full of instructions due to low priority, no more instructions that use the pipeline can be executed. But, the instruction that uses vacant pipeline can execute if data dependency is not happen. Write-back conflict is resolved by the priority scheduling, and pipeline resource conflict is controlled by the COPXEN signals for instruction scheduling in the host processor.

For data load/store operation, the load/store pipeline is design to fit with a host processor pipeline. Because coprocessor cannot access memory, host processor has memory write data to global data bus so that coprocessor reads that data when coprocessor data load operation is executed. And for coprocessor data store operation host processor generates memory address and coprocessor writes data to global bus so as to be stored. In general, one clock cycle load/store operation, no control signals are generated by the host processor. But, in the multiple cycle load/store operation, coprocessor must wait for the end of memory access and data preparation in host processor. In that case one or more control signals from host processor goes low to stall a co-processor load/store pipeline. But the other pipeline in the coprocessor can execute another instruction if data dependency, resource conflict and exception prediction is not happen.

For the support interrupt recovery exception prediction technique is used. Actually, CalmRISC32 FPU has no special exception recovery unit but every instruction is predicted for arithmetic exception and no further instruction issuing if exception prediction signal is active. And un-active exception prediction signal must guarantee that arithmetic exception never happens on the execution of the instruction. This technique has an advantage in area and design cost because special hardware unit (reorder buffer) is not needed.

In the first stage of arithmetic pipeline every instruction is checked for the possibility of exception. If the instruction can make an exception, exception prediction signal is generated and these signals make host processor stall with COPMEN and COPXEN signals. In the last stage of arithmetic pipeline true exception is generated, if exception is not happen, host processor continues instruction issuing, or happens, host processor jump to the coprocessor exception handling routine.

IV. IMPLEMENTATION

CalmRISC32 FPU is design with 0.25?m standard-cell library because design time is very important in embedded system market. CalmRISC32 FPU support 32bit single precision floating-point arithmetic instruction – floating-point addition/subtraction (FADD/FSUB), floating-point multiplication (FMUL), floating-point division (FDIV), format conversion (FTOI, ITOF), comparison (FCMP), rounding (FRND), load/store (CLD), and etc. and IEEE-754 standard rounding mode and exception signals are supported.

For fast floating-point addition/subtraction unit, parallel rounding algorithm is implemented. It can eliminate FALU pipeline stage and delivers fast floating-point ALU operation results. And to save design complexity of floating-point multiplier, multiplier hard-macro block is used, and the other adder and subtractor hard-macro blocks are used for another data-path design. These hard-macro blocks can save design and simulation time. In the design of floating-point divider special control block is design for the first division step because the first stage of divider has iterative property. With a simple co-processor interface and load/store unit hardware design cost and effort is reduced.

Instruction	Latency/throughput
FADD/SUB	3/1
FMUL	2/1
FDIV	17/15
Load/Store	2/1
Conversion	3/1
FRND	3/1
FCMP	1/1
Etc.	1/1

Table 1. Instruction latencies

First, Behavioral HDL model is implemented and verified with simple test vector. All arithmetic data-path is verified by the comparison of data-path calculation result and C program result. And then, the synopsys' s design analyzer generates Gate level model. Next, Timing verification with Samsung's in house tool – CubicWare is done. First prototype is running at about 70Mhz with worst-case delay in gate level simulation. The further gate level model optimization is needed for the best result.

Table.1 shows supported instruction latencies in CalmRISC32 FPU. Floating-point addition/subtraction, floating-point multiplication pipelines are optimized for fast program execution. And for comparison operation, dedicated comparison unit is design and it delivers one-clock cycles latency. The latency of floating-point division takes longer clock cycles than that of the other operation. But, this processor can execute another instructions on the middle of instruction execution. It can provide the improvement of program execution time, program optimization, and the improvement of overall performance.

V. CONCLUSIONS

The CalmRISC32 FPU is RISC type coprocessor. It is configured with CalmRISC32 micro controller. It supports IEEE-754 standard single precision floating – point addition/subtraction, multiplication, division, format conversion, comparison, rounding, load/store, and etc. It also supports four rounding modes, and exception signals. It can execute and complete instructions out of order if some constraint is resolved – data dependency, resource conflict, and exception prediction. It has simple co-processor interface, and has high performance due to constraint base dynamic scheduling. Standard cell base design technique is used to curtail design time and cost.

Acknowledgement

This research was supported by Samsung Electronics Co., Ltd. (1999)

REFERENCES

[1] Israel Koren, "Computer Arithmetic Algorithms," John Wiley & Sons, 1993

[2] D. Goldberg "Appendix A, Computer Arithmetic," in J.L. Hennessy and D.A. Patterson, Computer Architecture: A quantitative Approach, Morgan Kaufman Publisher, San Francisco, 1996

[3] Amos R. Omondi, "Computer Arithmetic Systems: Algorithms, Architecture and Implementation," Prentice Hall, 1994

[4] Woo-Chan Park, Shi-Wha Lee, Oh-Young Kwon and Tack-Don Han, "Floating point Adder/Subtractor Performing IEEE Rounding and Addition/Subtraction in Parallel," IEICE Trans. Inf.&Syst., Vol.E79-D, No.4, Apr. 1996

[5] Woo-Chan Park, Cheol-ho Jeong, Jin-ki Yang and Tack-don Han, "design of floating point multiplier performing the IEEE rounding and addition in parallel," Journal of the Korean institute of telemetics and electronics, Vol. 34, No.11, pp. 897-904, Nov. 1997

[6] D. E. Atkins, "Higher-Radix Division Using Estimates of the divisor and partial Remainder," IEEE Transaction on computers, 17, No 10, pp.925-934, Oct. 1968

[7] Milos D. Ercegovac and Tomas Lang, Division and Square Root: Digit-Recurrence Algorithms and Implementations, Kluwer Academic Press, 1994