In-Order Issue Out-of-Order Execution Floating-Point Coprocessor for CalmRISC32

Cheol-Ho Jeong, Woo-Chan Park, Tack-Don Han
Media System Laboratory, Department of Computer Science, Yonsei University
Seoul, Korea,
{chjeong, chan, hantack}@kurene.yonsei.ac.kr

Sang-Woo Kim
MCU Team, System LSI Division,
Samsung Electronics Co.
Yong-In, Korea
saroun@samsung.co.kr

Moon-Key Lee
VLSI and CAD Laboratory, Department of Electrical Engineering, Yonsei University
Seoul, Korea
mklee@spark.yonsei.ac.kr

Abstract

The CalmRISC32 FPU(Floating-Point Unit) is a RISC coprocessor for embedded system applications. It supports IEEE-754 standard single precision floating-point addition, floating-point subtraction, floating-point multiplication, floating-point division, format conversion, comparison, rounding, load, store, etc. It also supports four rounding modes, and precise exception. It can execute and complete instructions out of order, if such constraints as data dependency, resource conflict, and exception prediction are resolved. Standard cell-base design techniques were used to reduce design time and expense. The first prototype operated at approximately 70MHz with the worst-case delay in gate level simulation.

1. Introduction

Currently, embedded systems are the preferred choice of major semiconductor companies and mobile device manufacturers where they need a simple, light, and low power micro-controller, not a high-performance microprocessor. Obviously, the current design goal is for lower power and higher performance within certain constraints. Both on-chip and off-chip configurations of peripheral devices must be possible to meet various market requirements.

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

In general, floating-point operation latencies vary with the arithmetic instructions in the execution pipelines [1]. Therefore, it is usual to adjust all operation latencies to the longest pipeline latency. In the CalmRISC32 FPU, out-of-order execution and completion control schemes are designed to achieve higher performance. Scoreboarding and Tomasulo's algorithm are possible methods to support out-of-order execution completion [2]. The design cost and complexity of these techniques are too great for micro-controller applications. Therefore, constraints-based dynamic scheduling was used with data dependency checking, resource conflict checking, and exception prediction. With this technique, the CalmRISC32 FPU can perform instructions out of order of execution or completion. The exception prediction technique eliminates a special hardware unit for the reorder buffer for precise exception. All operands of the arithmetic instructions are checked for exception in the first stage of the pipeline, and if exception occurs, the coprocessor executes instructions in-order to handle the exceptional condition properly, otherwise the coprocessor performs instructions out of order.

The CalmRISC32 FPU was implemented with a standard-cell library to save implementation time and expense. It supports floating-point addition, subtraction, floating-point multiplication, floating-point division, format conversion, comparison, rounding, load, store, etc. A hard-macro block is used for the large conventional block—fraction multiplier, barrel shifter, adder, and subtractor. This reduced the design time and verification effort.

Section 2 of this paper describes the architecture of the CalmRISC32 FPU; Section 3 describes the coprocessor interface to the host processor; design methods and implementation scheme are explained in Section 4; and Section 5 presents the conclusions.

0-7695-1150-3/01 $10.00 © 2001 IEEE

2. CalmRISC32 FPU architecture

The CalmRISC32 FPU is a 32-bit RISC type coprocessor that executes floating-point operations with the support of a CalmRISC32 microprocessor. It was designed for micro control units that are used in embedded systems. It can be applied to high-speed floating-point calculation, signal processing and 3D graphics applications with a multiple FPU. It is composed of a hardware floating-point arithmetic and logic unit (ALU), floating-point multiplier, and floating-point divider. It has independent instruction decoder, load/store unit, register file and coprocessor interface unit. Therefore, the CalmRISC32 FPU can be included, or excluded along with the application domain. It can execute several instructions simultaneously within some constraints and complete instructions out of order if arithmetic exception is not generated by the instruction. The details of the control scheme are described in the next section.

The CalmRISC32 FPU supports 32-bit single precision floating-point addition, subtraction, multiplication, division, comparison, type conversion and rounding operations. In addition, four IEEE-754 standard rounding modes (round towards zero, round to nearest, and round towards negative/positive infinity) and exception are supported. It has a greatly reduced interrupt recovery mechanism with a simple coprocessor interface and exception prediction technique. It has a 16×32-bit register file, one status/control register, and one exception register. Rounding modes, exception type, and comparison results are stored into these special registers.

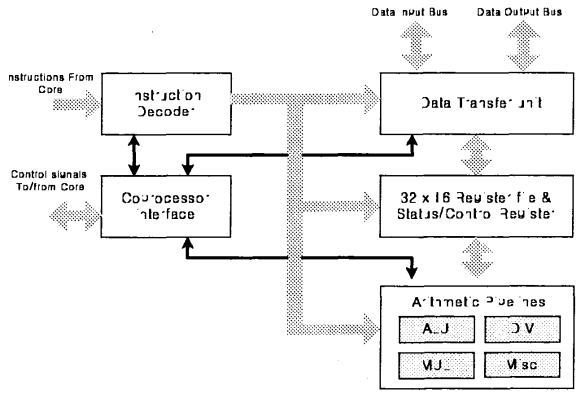

Figure 1. Block diagram of the CalmRISC32 FPU.

A data transfer unit under the control of the coprocessor interface unit performs data transfer from and to the host processor (CalmRISC32). To reduce the design complexity of the coprocessor, it has neither an instruction fetch unit nor a memory address generation unit. As the coprocessor cannot independently access memory, the host processor takes the responsibility for instruction fetching, address generation and data preparation for memory read or write operations.

An instruction is first read and pre-decoded by the host processor. If the pre-decoded instruction is a coprocessor instruction, the instruction code is transferred directly to the coprocessor with several control signals. The coprocessor decodes the received instruction and executes it on the appropriate pipeline. For load/store operations, memory address generation and data preparation for transferring to the coprocessor or to memory, the host processor continues instruction execution. The coprocessor only has responsibility for data preparation and data transfer to the host processor through the data bus.

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

2.1 FPU execution pipeline

Figure 2 shows the pipeline diagram of the CalmRISC32 FPU. The CalmRISC32 FPU has five separate pipeline paths—floating-point ALU pipeline (FALU), floating-point multiplication pipeline (FMUL), floating-point division pipeline (FDIV), floating-point load/store pipeline (FLDST), and miscellaneous pipeline (Misc.). As shown in Figure 2, these pipelines have different operation latencies, and all pipelines except the FDIV are full pipelines. The first stage of the FDIV pipeline has an iterative path that has 15 clock cycle latencies. With the in-order issue and in-order completion control scheme the FPU pipeline resources may be largely wasted, because another instruction that uses vacant pipelines cannot be issued until the current instruction completes its execution. To fully utilize these pipelines, simple dynamic instruction scheduling was used. This dynamic scheduling can be achieved by resource conflict checking at 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 continue issuing instructions to the coprocessor until a data dependency or resource conflict is found. Issued instructions are executed simultaneously in those pipelines and complete the operation out of order if resource conflict is resolved and the other pipelines do not generate an exception prediction signal. If an exception prediction

signal is generated, the host processor stops issuing instructions until the instruction that generated the exception prediction signal completes its execution. If exception prediction is false, the host processor continues the program execution.

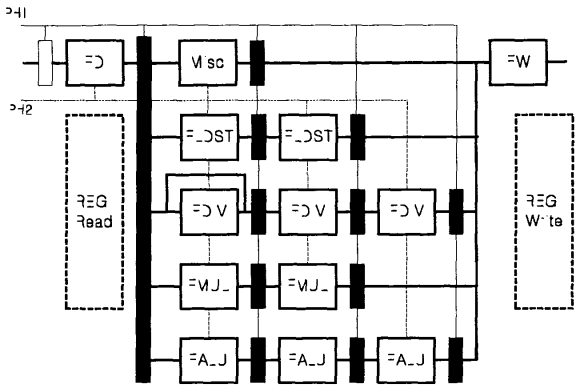

Figure 2. Pipeline diagram of the CalmRISC32 FPU

In the FW stage, if two or more write-back data are available, only one write-back data is selected from these five pipelines according to the predefined priority and a pipeline stall signal is generated to stop pipeline advance of the other pipeline, or pipelines. On the next clock cycle, the next priority write-back data is advanced to the FW stage.

The coprocessor catches pipeline stall conditions in the case of data dependency and resource conflict. If stall conditions are found, it generates appropriate control signals to stall the host processor and to stop issuing instructions.

2.2 Floating-Point ALU

The floating-point ALU pipeline is composed of a floating-point addition/subtraction unit, comparison unit and exception prediction unit. It handles floating-point addition, subtraction, type conversion, rounding and comparison operations. Generally, floating-point addition/subtraction takes four processing steps—alignment, fraction addition/subtraction, normalization, and rounding. It needs an additional fraction adder for rounding and this increases processing time and die area. It also has a re-normalization step caused by overflow in rounding processing. In order to reduce that re-normalization overhead, parallel-rounding algorithms are implemented [3]. With this algorithm, fraction addition/subtraction and rounding are executed simultaneously in the second pipeline stage. It also supports four IEEE standard rounding modes. Therefore, it can perform floating-point addition/subtraction and the other ALU operations within three clock cycles, i.e., rounding operation and format conversion operation (integer to floating-point or floating-point to integer). In addition, it does not need a re-normalization step because the rounding takes place before normalization, and the additional adder is eliminated.

For 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 the input operands is an invalid format, or the exponent calculation of the two operands may cause exception, the ALU generates an exception prediction signal to prevent further instruction issue by the host processor. The last ALU pipeline stage ascertains the truth of arithmetic exception by the status/control register setting values and the operation result to derive an exception handling condition. Setting the status/control register value to all zeros can ignore the exception generated by the arithmetic pipeline.

2.3 Floating-Point multiplier

The FMUL pipeline has two stages. In the first stage, floating-point fraction multiplication and addition of partial products is performed. In the next stage, fraction rounding and normalization is executed. It was designed with conventional floating-point multiplication [4]. In order to save design time and effort, an integer multiplier hard macro in a target library was used. In addition, in the first stage, the exponents of two operands are examined for exception prediction. If one of the two operand's exponents or fractions has an invalid number format (Not a Number, Infinity number or De-normalized number), or the addition of the two exponents may cause an overflow or underflow exception, the exception prediction signal is generated to stop instruction issuing according to the status/control register setting. The exact exception signal is generated in the final stage of the FMUL pipeline to process exception handling.

2.4 Floating-Point divider and Load/Store unit

The FDIV pipeline has an iterative first stage and non-iterative other stages. In the first stage, a radix-4 SRT division algorithm is used [5], [6], [7]. In the second stage, quotient addition is performed, and rounding and normalization are performed in the final stage. The exponents of the two operands are examined for exception prediction in the first stage. If either of the two operand's exponents or fractions is an invalid number or zero divisor (Division by Zero), or the subtraction of the two exponents may cause overflow, or underflow exception, the exception prediction signal is generated to stop issuing instructions. In the last stage of the FDIV pipeline, an exact exception signal is generated according to the

status/control register setting to process exception handling.

The FLDST has two stages to comply with the memory access stage (MEM) in the host processor. In the first stage of FLDST, there is no operation, but in the next stage a data read or write operation is executed through the data input bus or output bus. For multiple cycle load/store instructions, the pipeline control signal is used to stall the coprocessor. The next section describes the host and coprocessor interface mechanism.

The Misc. pipeline can execute register move, absolute value, negation value and constant (0.0 or 1.0) load operations.

3. Coprocessor interface and control scheme

To reduce the design complexity and effort for the coprocessor exception recovery and memory access units are excluded. A simple host-coprocessor interface unit can support precise exception and dynamic scheduling with some constraints. That is the instruction stream is scheduled with host processor and interface signals. The host processor issues instructions sequentially, but the issued instruction completions can be performed out of order in the coprocessor.

3.1 Coprocessor interface

The coprocessor and host processor need to be synchronized in some cases, e.g., instruction issue may not be allowed since the coprocessor has suffered a stall and can get no further instructions from the core processor; the host processor is stalled and cannot provide data for coprocessor data transfer instruction; etc. Therefore, the coprocessor and host processor are synchronized with each other by special control signals—STXEN, STMEN, STWEN, COPXEN, COPMEN, and COPWEN. These signals are active in low state, where "ST" means the host processor status; "COP" means the coprocessor status; and "X", "M" and "W" stand for execute stage, memory stage, and write-back stage, respectively. The last letters "EN" denote enable. That is, for example, if STMEN is high, it means that the instructions in the host processor can advance to the next memory stage; if low, the instruction in the host processor cannot advance to the next stage. If COPXEN is in the low state, it means that some pipeline stall conditions (data dependency, resource conflict, and pipeline full) have occurred and instruction issue must be stopped until COPXEN goes high. Instruction issue control is therefore achieved with these control signals.

3.2 Coprocessor control

The instruction that is received from the host processor is decoded and checked for data dependency and resource conflict. If data dependency is found, a COPXEN signal is generated to stop instruction issue. As soon as the data dependency is resolved, the COPXEN signal goes high.

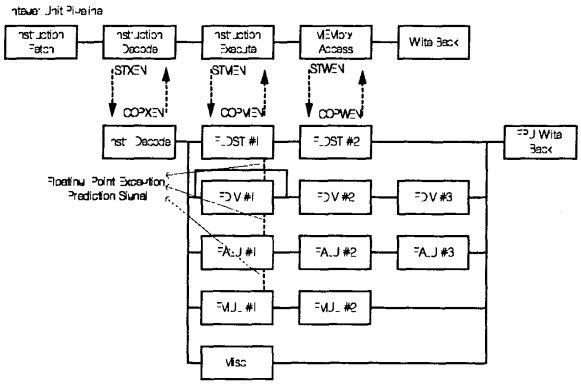

Figure 3. Host and coprocessor interface signals

If resource conflict is found, the COPXEN signal is used to stop instruction issue as in the case of data dependency. In the coprocessor, there are two possible resource conflicts. First, resource conflict can happen if two or more pipelines attempt to advance simultaneously to the coprocessor write-back stage. Because only one write-back data can advance to write-back stage at each clock cycle, the other instructions in the pipelines must wait until the next clock cycle. Write-back data selection is scheduled with first priority. Second, resource conflict can happen if one pipeline is full of instructions due to low priority; no more instructions that use the pipeline can be executed, but an instruction that uses a vacant pipeline can execute if data dependency has not occurred. Write-back conflict is resolved by priority scheduling, and pipeline resource conflict is controlled with the COPXEN signals for instruction scheduling in the host processor.

The coprocessor instructions are issued sequentially by the host processor, but the coprocessor can execute and complete the instructions out of order because it has five pipelines varying from one stage to three stages. That is, if data dependency and resource conflict are found in the coprocessor decode stage, the later issued instruction having shorter cycle latency than that of the previous instruction, need not wait for the completion of the previous instruction and can complete its own execution. To implement this out-of-order execution control, the register lock bits, pipeline status bits and operand ready bits are added in the control block. The register lock bits show whether or not the register is the write-back destination of any instruction. Pipeline status bits show which pipeline is full of instructions and those that are incomplete or empty. Operand ready bits show whether or not two source operands are available for execution. That is, if data is forwarded from the FW stage, or the lock bits

of two source registers are not set, the operand ready bit is set for instruction execution (coprocessor instruction is two-operand type).

As soon as the coprocessor instruction is decoded, the lock bit of the destination register is set for data dependency checking, and the pipeline status bits and operand ready bits determine whether the instruction is issued. If one of these bits is not set, then data dependency or resource conflict is found. The coprocessor cannot issue the instruction and the host processor cannot issue coprocessor instructions to the coprocessor until these bits are set—data dependency or write-back scheduling is complete.

For the data load/store operation, the load/store pipeline is designed to communicate with a host processor pipeline. Because the coprocessor cannot access memory, the host processor reads memory and writes data to a global data bus, and the coprocessor reads that data when a coprocessor data load operation is executed. For the coprocessor data store operation, the host processor generates memory addresses and the coprocessor writes data to the global bus to be stored. In general, one clock cycle load/store operation generates no control signals by the host processor. In the multiple cycle load/store operations, the coprocessor must wait for the end of memory access and data preparation in the host processor. In that case, one or more control signals from the host processor goes low to stall a coprocessor load/store pipeline (STMEN, STWEN), but the other pipelines in the coprocessor can execute another instruction if data dependency, resource conflict or exception prediction does not happen.

3.3 Precise exception

An exception prediction technique is used for the support of interrupt recovery. CalmRISC32 FPU has no special exception recovery unit, but every instruction is checked before execution for arithmetic exception, and no further instruction is issued if the exception prediction signal is active. A non-active exception prediction signal must guarantee that arithmetic exceptions never happen on the execution of the instruction. This technique has an advantage in area and design cost because a special hardware unit (reorder buffer) is not required.

Exception prediction is the checking of the floating-point operation result for exception before execution. Exception predictors are designed individually for each arithmetic pipeline. Exception is predicted by the input pattern checking and exponent calculation. First in the input pattern checking is the examination of input data. That is, if the input number is not a number (NaN), is negative or positive infinity (±Inf) on floating-point addition, invalid exception occurs. Second, the possibility of overflow or underflow exception is checked by the exponent calculation of the two input numbers, e.g., on the floating-point multiplication, if the addition result of input number exceeds the maximum value of exponent (254), overflow exception is possible. If the exponent addition result is equal to the maximum value of exponent, it is not an actual overflow case, but on the rounding step of floating-point multiplication, depending on the fraction result, an overflow exception is possible. Therefore, the case of the exponent calculation result being equal to the maximum value of the exponent is included in the exception condition.

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

4. Implementation

The CalmRISC32 FPU was designed with a 0.25-μm standard-cell library because design time is very important in the embedded system market. This library has four metal layers. The CalmRISC32 FPU supports 32-bit single precision floating-point arithmetic instructions—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), etc., and IEEE-754 standard rounding mode and exception signals are supported.

Table 1. Instruction latencies.

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

For the fast floating-point addition/subtraction unit a parallel rounding algorithm is implemented. It can eliminate the FALU pipeline stage and delivers fast floating-point ALU operation results. To save design complexity of the floating-point multiplier, a 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 the floating-point divider, a special control block was designed for the first division step because the first stage of the divider is iterative. With the simple coprocessor interface and load/store unit, hardware design cost and effort was reduced.

4.1 Testing

First, the behavioral HDL model was implemented and verified with a simple test vector. All arithmetic data-paths were verified by the comparison of the data-path calculation result with a C program result. Then the Synopsys's design analyzer generated a gate-level model. Next, timing verification with Samsung's in-house tool, CubicWare, was performed. The first prototype ran at approximately 70MHz with worst-case delays in the gate-level simulation. Further gate-level model optimization is required for better results.

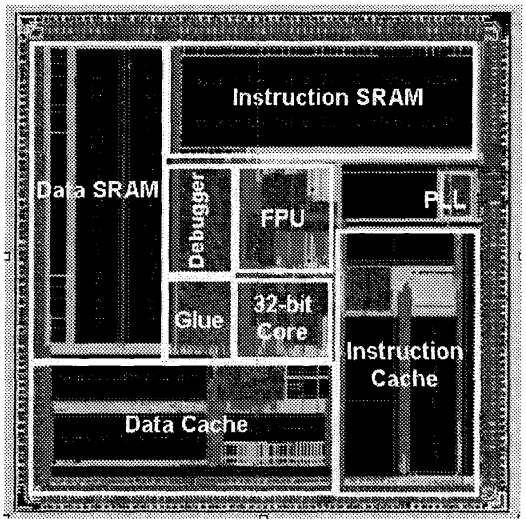

Figure 4. Die photography of the CalmRISC32 evaluation chip.

Table 1 shows supported instruction latencies in the CalmRISC32 FPU. Floating-point addition/subtraction and floating-point multiplication pipelines are optimized for fast program execution. For comparison, a dedicated comparison unit was designed and it delivered one clock cycle latency. The latency of floating-point division takes more clock cycles than any of the other operations. However, this processor can execute another instructions in the middle of instruction execution. It can provide an improvement of program execution time, program optimization and improve overall performance.

Figure 4 shows a photograph of the evaluation chip of the CalmRISC32 micro-controller, CalmRISC32 FPU, instruction cache, data cache and peripheral blocks. The size of this chip is 44.5mm^2 and the approximate size of the CalmRISC32 FPU is 2.46mm^2. The transistor count of the chip is about 879,806 and the operational clock frequency is up to 150MHz. Currently, the chip testing is in process, but the CalmRISC32 FPU may operate up to 100MHz. The dynamic power dissipation was evaluated with simple test vectors and Samsung's in-house tool, Cubic Power. The measured average power consumption of the CalmRISC32 FPU was about 983uA at 2.5 volts. The evaluation board is currently being debugged. After all bugs are fixed and the evaluation board is delivered, actual performance and power dissipation measurements will be possible.

5. Conclusions

The CalmRISC32 FPU is a RISC-type coprocessor. It is configured to operate with a CalmRISC32 micro-controller. It supports IEEE-754 standard single precision floating-point addition/subtraction, multiplication, division, format conversion, comparison, rounding, load/store, etc. It also supports four rounding modes, and exception signals. It can execute and complete instructions out of order if some constraints are resolved—data dependency, resource conflict, and exception prediction. It has a simple coprocessor interface, and has exception predictors for the support of precise exception. A standard cell-base design technique is used to curtail design time and cost.

Acknowledgement

This work is partially supported by the National Research Laboratory Project of Ministry of Science and Technology, Republic of Korea.

References

[1] I. Koren, *Computer Arithmetic Algorithms*, John Wiley & Sons, 1993.

[2] R. M. Tomasulo, "A Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM J. Research and Development*, Vol. 11, Jan. 1967, pp. 25 - 33.

[3] W. C. Park, S. W. Lee, O. Y. Kwon and T. D. Han, "Floating point Adder/Subtractor Performing IEEE Rounding and Addition/Subtraction in Parallel," *IEICE Trans. Information & System* Vol. E79-D, No. 4, Apr. 1996, pp. 297-305.

[4] A. R. Omondi, *Computer Arithmetic Systems: Algorithms, Architecture and Implementation*, Prentice Hall, 1994.

[5] D. E. Atkins, "Higher-Radix Division Using Estimates of the Divisor and Partial Remainder," *IEEE Transaction on Computers*, 17, No. 10, Oct. 1968, pp. 925-934.

[6] M. D. Ercegovac and T. Lang, *Division and Square Root: Digit-Recurrence Algorithms and Implementations*, Kluwer Academic Press, 1994.

[7] S. Oberman and M. Flynn, "Division Algorithms and Implementations," *IEEE Transaction on Computers*, Vol. 46, No. 8, Aug. 1997, pp. 833-854.