Send Orders for Reprints to reprints@benthamscience.ae

Method for Ultra-precision FPU Integration based on Fine-Grained Control

Qing-yu Chen and Long-sheng Wu*

Xi'an Microelectronics Technology Institute, Xi'an 710054, China

Abstract: In general, the FPU and processor are decoupled in the method for FPU integration, in which the communication between them requires software intervention and ultra-precision FPU is unsupported. To avoid this problem, a method based on fine-grained control for integration of FPU into the RISC processor is proposed in this paper. In terms of operand width of floating-point instructions, the method divides floating instructions into three categories: S, D and U, and further subdivides the execution status of S, D and U. Then, it regards the execution status as basic granularity to generate the FPU control information and moves the control information needed by destination operands to the next pipeline stage. Finally, segmentation of destination operands is achieved in different pipeline stages and the destination operand is written to register file after segmentation with the pipeline. An 80-bit FPU is embedded into a SPARC V8 processor based on the proposed method. The results of implementation and verification show that the critical path of floating instructions decreases by 37.3%, hardware consumption reduces by 16.9% and the floating-point calculation efficiency increases 1.7 times. The proposed method can be used to apply the ultra-precision FPU embedded into the RISC processor, and to make an efficient collaborative computing between them at low hardware overheads.

Keywords: Execution status, grain, FPU integration, collaboration computing, ultra-precision

1. INTRODUCTION

Various soft computing solutions can be further illustrated based on some particle swarm optimization (PSO) and artificial neural network(ANN) models [1-4] which are computationally time-consuming or may need parameter estimation [5, 6]. In fact, in addition to model simulation, scientific and real-life applications also have also more critical requirements for the floating-point performance and data accuracy of embedded processor [7]. Nowadays, although the vast majority of processors integrate double precision hardware float point unit (FPU) to improve floating-point performance and data accuracy [8-11], which can hardly satisfy the actual application.

Ultra-precision, an industrial standard developed by Intel Corporation, which means floating-data precision exceeding double precision, has the ability to meet the requirements in data accuracy. However, the ultra-precision computing is achieved by software in the contemporary embedded fields [12], which dramatically reduces overall performance of processor [13, 14]. As a result, the ultra-precision FPU integration in Reduced Instruction Set Computer (RISC) processor is an important ongoing research of processor design.

The ultra-precision FPU integration is very complicated as the pipeline state of processor must be taken into consideration. Several published methods for FPU integration cannot be used to ultra-precision FPU integration and have relatively low efficiency because these methods decouple the communication between FPU and processor. Thus, software intervention is needed in floating-point operation. Previous work on FPU integration will be described in detail in section II.

To solve the problem above, a fine-grained integration method for ultra-precision FPU, which based on centralized control and data segmentation, is proposed. The method considers fully pipeline state of processor and makes FPU and processor tightly coupled, which is implemented by appropriative hard modules. Meanwhile, it regards execution status of floating-point instructions as basic granularity to implement the precise control of FPU and to simplify the design complexity. Compared with studies published elsewhere, the main contributions of this paper are as follows:

(1) For the first time, this paper discloses the integration method of ultra-precision FPU into pipeline RISC processors with no need to change the existing microprocessor module. Based on the proposed method, an 80-bit FPU has been integrated into the Scalable Processor Architecture version8 (SPARC V8) processor.

(2) The FPU execution efficiency based on the proposed approach is very high as it is implemented by hardware and there is no need for software intervention in floating-point operation.

This paper is organized as follows. In Section II, related work published is introduced.Moreover, in section III, the ultra-precision FPU integration method based on centralized

*Address correspondence to this author at the Xi'an Microelectronics Technology Institute, Xi'an 710054, China; Tel: +8602988609000-8758; Fax: +8602988609000-8203; E-mail: chenqingyu2006@163.com

1874-4443/15

Table 1. Precision type combination of floating-point operand.

Destination / Source	Integer Type	Single Precision	Double Precision	Ultra-Precision
Integer type		SIDS	SIDD	SIDU
Single precision	SSDI	SSDS	SSDD	SSDU
Double precision	SDDI	SDDS	SDDD	SDDU
Ultra- precision	SUDI	SUDS	SUDD	SUDU

control and data segmentation is proposed, and is used to apply an ultra-precision FPU (80 bits, Meiko interface and compatible with Intel floating-point coprocessor) embedded into the SPARC V8 LEON2 processor of five-stage pipeline[15]. In Section IV, implementation results of proposed method will be contrasted with several published mechanisms. Finally, in Section V, the conclusions will be presented.

2. RELATED WORK

The FPU integration methods have been described in a number of literatures. Schwarz and Trong [16, 17] introduce the implementation of high precision FPU, and Yong makes an 80-bit FPU embedded into the X86 processor using the micro-instruction code stored in the ROM [18]. The fetch of micro-instruction code would consume processor execution time, which reduces floating-point efficiency. And due to the difference in processor architecture, it is infeasible for micro-instruction code to apply to pipeline RISC processors.

Joven, Gajjar and Du [14, 19, 20] attenuate the degree of coupling between FPU and processor in which FPU serves as a slave unit of on-chip bus, and the calculation process of FPU is controlled by st/ld instructions. The above schemes need software intervention in floating-point operation and increase the access conflicts of on-chip bus which causes FPU efficiency extremely low. Although some effective measures, FPU dedicated data bus and more effective interactive approach of processor and FPU, are taken to improve the calculation efficiency and the result is still not ideal.

Brunelli [21] integrates a reconfigurable FPU into the main processor core by aa universal I/O interface containing data and control bus. This way is easy to implement, but not to consider the processor pipeline status. IBM developed a dedicated interface for FPU integration [22, 23], namely auxiliary processor unit interface (APU). The APU connects into the processor instruction pipeline and has the ability to negotiate the transfer of particular instructions and data to FPU. The IBM's solution is very efficient but unsuitable for ultra-precision FPU integration.

The proposed ultra-precision FPU integration method considers fully instruction pipeline state of processor and makes FPU and processor tightly coupled, where software intervention is not needed. Thus, communication overheads between FPU and processor can be ignored. At the same time, the design based on fine-grained control can simplify implementation of ultra-precision FPU integration. All that can achieve a significant improvement on floating-point calculation efficiency and play an important role in reducing the hardware overheads.

3. CONTROL ALGORITHM AND ITS IMPLEMENTATION

The implementation of the proposed method, only needs to add control logics of FPU in different pipeline stages with no changing the rest processor modules. The control algorithm and its implementation of the proposed method will be presented by taking a five-stage pipeline RISC processor [24] for example, in which fine-grained centralized control is implemented in instruction decoder stage(ID) whereas data segmentation relates to execution(EX), memory access(MA) and write back stages(WB).

3.1. Principles of Fine-grained Control

Floating-point instructions achieve the conversion and operation of floating-point data. Its classification shown in Table **1**, the precision type of source and destination operand is indicated with S and Q respectively and both include integer (I), single precision (S), double precision(D) and ultra-precision (U). So SDDS refers to the instructions with source and destination operands being double and single precision. Depending on the precision, the 15 types of floating-point instruction are divided into three types: S, D and U (Table **1** indicates with green, red and blue), and the operand width of S, D and U is 32, 64 and 80 bits. The proposed method further subdivides the execution status of D and U. By different status, the wide operand writing to narrow floating-point register file is achieved through pipeline.

The fine-grained control is implemented by state machine analyzing floating-point instructions. As shown in Fig. (**1**), there are four S states corresponding to three kinds of instruction: S, D and U, and X.S0 is a shared state by all three kinds of instructions. In X.S0, three categories of instructions will be finely differentiated. If the S type of instructions or control hazard exists in X.S0 state, state machine remains unchanged. While there are D or U instructions in X.S0, state machine will move D.S1 or U.S1 respectively when pipeline enable is active (*hold=1*). Others than X.S0, PC update is prohibited to prevent new instructions from getting into ID stage. Regarding the status of state machine shown in Fig. (**1**) as fundamental granularity, Control algorithm generates the FPU control information, and transfers the control information needed by destination operand to the

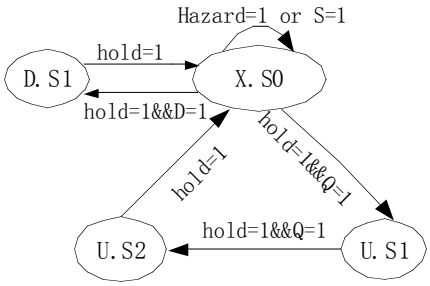


Fig. (1). State machine based on Fine-grained control.

next pipeline stage, which provides principle for segmentation of destination operands subsequently.

Control algorithm of source operand is shown in Fig. (**2a**), regfile[*rs*] refers to the data in floating-point register file specified by source address(*rs*). In X.S0 state, control algorithm does not pass regfile[*rs*] to the least 32 bits of FPU input(*fpui.rs*) until all hazards disappear. In S1(D.S1 or U.S1), regfile[*rs*+1] will be assigned to the middle of the 32 bits of *fpui.rs*, while in the condition of S2 (U.S2), regfile[*rs*+2] will be connected to the most 16 bits of *fpui.rs*. At the same time, once the source operand being ready according to the source operand precision, FPU operation will be start(*fpui. Start = '1'*).

```
if(state=S0)
 if(any kind of hazard existing )
   wait;
 else
  fpui.start='0';
  when (SI |SS)=>fpui.rs[31:0]=regfile[rs];
        fpui.start='1';
  when (SD |SQ)=>fpui.rs[31:0]=regfile[rs];
 end if;
elseif(state=D.S1)
  fpui.start='0';
  when SD=>fpui.rs[63:32]=regfile[rs+1];
       fpui.start='1';
elseif(state=U.S1)
  fpui.start='0';
  when SD=>fpui.rs[63:32]=regfile[rs+1];
       fpui.start='1';
  when SQ=>fpui.rs[63:32]=regfile[rs+1];
       fpui.start='0';
elseif(state=U.S2)
  fpui.start='0';
  when SQ=>fpui.rs[79:64]=regfile[rs+2];
       fpui.start='1';
  end if;
```

(**a**) Read control method for the source operand

```
if(state=S0)
  pipeline.state="00";
  if(the width of D≥the width of S )
   pipeline.rd=op.rd;
   pipeline.rd_wen='1';
  else --SDDI or SDDS or SQDI or SQDS or SQDD
   pipeline.rd=Zeros;
   pipeline.rd_wen='0';
  end if;
elseif(state=D.S1)
  pipeline.state="01";
  pipeline.rd_wen='1';
  if(the width of D≥the width of S )
    pipeline.rd=op.rd+1;
  else --SDDI or SDDS
   pipeline.rd=op.rd;
  end if;
elseif(state=U.S1)
  pipeline.state="01";
  pipeline.rd=op.rd+1;
  if(the width of D≥the width of S )
   pipeline.rd_wen='1';
  else --SQDI or SQDS or SQDD
   when SQDI | SQDS=>pipeline.rd_wen='0';
            pipeline.rd=Zeros;
   when SQDD =>pipeline.rd=op.rd;
  end if;
elseif(state=U.S2)
  pipeline.state="10";
  pipeline.rd_wen='1';
  if(the width of D≥the width of S )
   pipeline.rd=op.rd+2;
  else --SQDI or SQDS or SQDD
   when SQDI or SQDS=>pipeline.rd=op.rd;
   when SQDD => pipeline.rd=op.rd+1;
 end if;
```

(**b**) Control theory of the destination operand write-back.

Fig. (2). Fine-grained control mechanism based on execution status.

Control theory of destination operand is shown in Fig. (**2b**), *pipeline* means the data structure of pipeline registers. Two rules are defined in this method: (1) the priority of S2, S1 and S0 state decreases in turn; (2) use high priority state to write the least data of FPU output. Compare the width of source and destination operands in X.S0, D.S1, U.S1 and U.S2. Then based on the comparison result generates write enable and write address of destination operands needed by FPU output write-back, and transfers the control

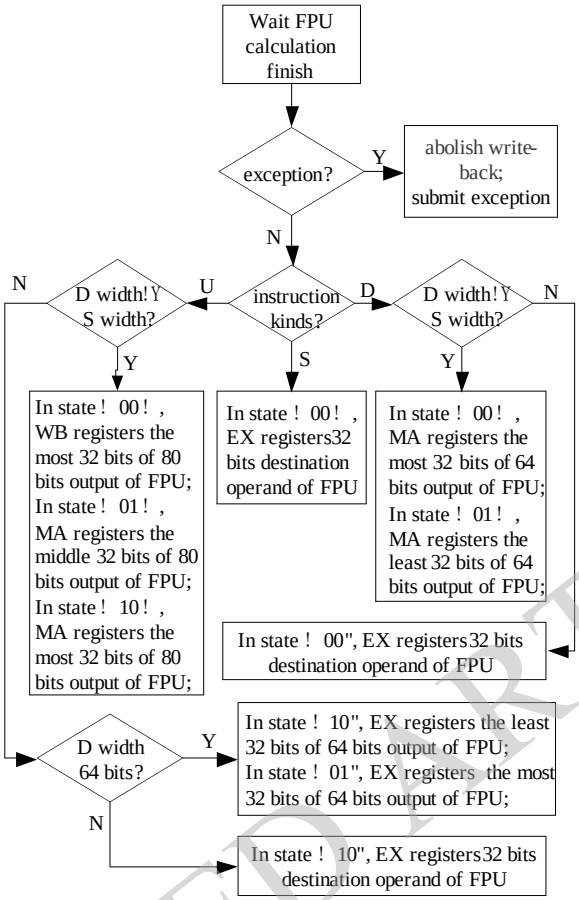


Fig. (3). Flow chart of segment data processing.

information, write enable and write address to next pipeline stage. Taking D.S1 for example, there are five kinds of instructions can enter the state as shown in Table **1** and only SDDI and SDDS whose destination operand width is less than the source operand. According to the principle (2): use high priority state to write back the least data, So in the state of D.S1, we set write enable valid(*rd_wen='1'*) and gives the correct write-back address(*pipeline.rd*).

3.2. Segmentation of Data

As soon as FPU completes calculation, the EX, MA and WB stages of pipeline processor will register FPU output in segmentation and then move to the next stage. Finally, the registered data are written back to floating-point register file through pipeline. The algorithm of processing in segmentation is further described in Fig. (**3**). Firstly, algorithm estimates whether FPU output is normal or not. If there has an abnormal output, write-back of FPU output will be abolished and exception information will be submitted to the exception handling module. Otherwise, FPU output is registered in segmentation on the basis of instruction type and state information(*pipeline.state*), which makes one to one correspondence with the control information, write enable and write address, generated in Fig. (**2b**). Taking U instructions for example, according to the rules (2): use high priority state to write back the least data of FPU output, EX will register 32 bits output of FPU to pipeline register in "10" state for SUDI and SUDS who only have 32 bits destination operand. Yet for SUDD whose valid output is 64 bits, EX will register most 32 bits of FPU output to pipelined register in "10" state, and meanwhile MA register least 32 bits of FPU output in "01" state. However, for the other U kind of instructions owning 80 bits destination operand, EX holds the least 16 bits of 80-bit FPU output in "10" state, MA registers the middle of the 32 bits and WB stores the most 32 bits into pipelined registers in "00" state at the same time.

3.3. Implementation of FPU Integration Method

The SPARC V8 is only architecture that defines the quadruple-precision instruction (ultra-precision) and is fully open and non-proprietary [25]. Other RISC architectures, achieving the ultra-precision floating operation through software, don't have ultra-precision floating-point instructions and need delegation of authority in the process of industrial application. Given all that, the SPARC V8 can make a thorough evaluation for our method with the help of open source implementations and open source simulator, and is chose to further illustrate the implementation of FPU integration method in RISC pipeline processor.

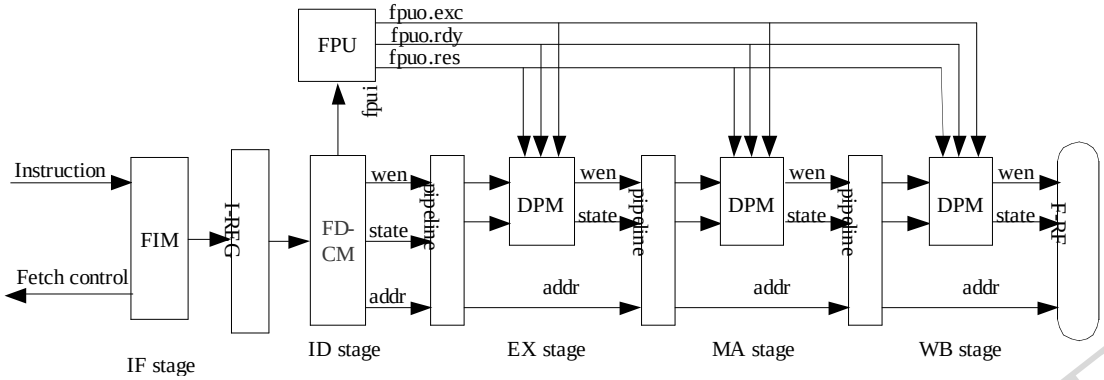
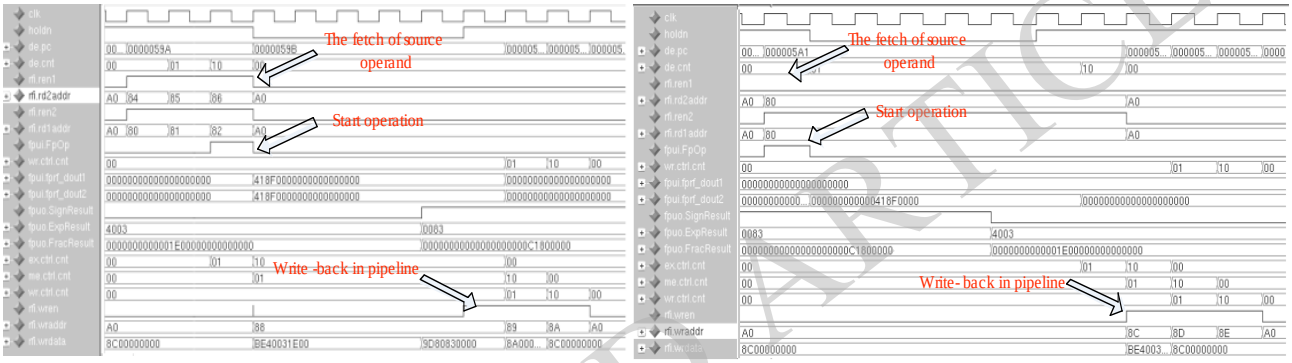


Fig. (4). Coupling schematic diagram between ultra-precise FPU and CPU core.



(**a**) SUDU Timing Diagram (**b**) SSDU Timing Diagram.

Fig. (5). Timing Diagram of the proposed integrated method.

A tight coupling scheme of an ultra-precision FPU(Meiko interface, 80 bits) with typical five-stage pipeline RISC is shown in Fig. (**4**). The fine-grained control module (FDCM) in ID stage achieves control algorithm mentioned in section 2.1 and moves the control information needed by FPU output to the next stage through pipeline register (*pipeline*). The data processing module (DPM) in EX, MA and WB implement the destination operand segmentation and write them to floating-point register file depending on the flow shown in Fig. (**3**).

4. RESULTS OF IMPLEMENTATION AND TEST

Many verification methodologies can be used to verify the proposed method and their most difference for users is the programming language. In this case, the correctness and timing diagram of the proposed method has been verified based on Cadence's e Reuse Methodology(eRM) which is licensed by Cadence. Compared with TestFloat developed by the Stanford, the floating-point results of processor are correct. The typical timing diagram of proposed method is shown in Fig. (**5**) where (a) for SUDU floating-point instructions (both source and destination operands are ultra-precision 80 bits), and (b) for SSDU floating-point instructions(source operand is single precision and destination operand is ultra-precision). In Fig. (**5a**), ID stage takes three states of X.S0, U.S1 and U.S2 to prepare 80 bits source operand, and starts the FPU operation in U.S2. WB writes the destination operand using state of "00", "01" and "10". In Fig. (**5b**), ID stage prepares the 32 bits source operand and starts the FPU operation in X.S0, and subsequently generates the write enable and write address needed by destination operand in corresponding U.S1 and U.S2. When the FPU calculation is finished, the various pipeline stages(EX, MA, WB) will register the 80 bits FPU output in segmentation according to the information stored in pipeline registers and finally WB stage writes the destination operands to floating-point register file.

In order to carefully evaluate this method, some assumptions are made. (1) The design of FPU and timing constraint are the same in other benchmarks as the one which is employed the proposed method. (2) No FPU exception happens in the process of evaluating the floating-point efficiency. The differences between timing constraints may result 20% deviation compared with results at typical corner. Whereas the design and exception of FPU affect significantly results of floating-point efficiency.

The comparison in this section can be divided into three types: critical path delay, hardware overhead and floating-point efficiency. However, the evaluation of the integration method involves the design and implementation of float

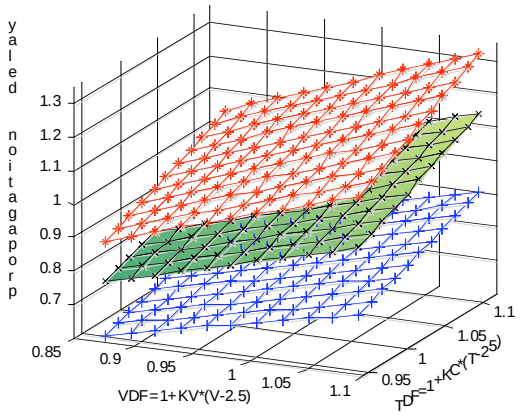


Fig. (6). Propagation delay with derating factors.

Table 2. Comparison of hardware overheads of various methods.

Device	**Xilinx xc5vlx85ff676**		
Design	LEON3 [19]	Cortex-M1 [14]	The proposed method
LUTs	3241	4312	3585
Flip Flops	1450		1594

point unit (FPU), which affects significantly comparison results. So only the same form as the ultra-precision FPU, based on Gaisler research's intellectual property core [26], is adopted as benchmark for fair and thorough comparison [14, 18, 19].

The analysis of critical path delay is firstly presented. In delay model, the critical path propagation delay is calculated based on delay at typical corner(2.5V, typical process, room temperature), derating factors of process, voltage and temperature. However, the propagation delay varies greatly from different derating factors of process, voltage and temperature, as is depicted in Fig. (**6**). The derating factors, defined as important parameters in delay module, illustrate influence on propagation delay of the process, voltage and temperature. The voltage and temperature derating factors(VDF and TDF) are chosen as the x and y coordinate axes and z axis indicates propagation delay. Three parametric surfaces correspond with process derating factors of slow, typical and fast. The derating factor at typical corner(KV, KC) indicated with '1' is constant and adopted as benchmark. With the rise of temperature and reduction in applied voltage, the delay does increase and the fast process derating factor has the minimal delay whereas the slow has maximal. Although all those factors can affect the delay, the most serious type is process derating factor. On the condition of the same typical process factor, the VDF and TDF make almost 18% variation on the propagation delay. In the same VDF and TDF, the propagation delay is 0.804, 1 and 1.15 corresponding to slow, typical and fast process factor.

The typical corner is standard application environment and is adopted as benchmark for synthesis, at which the critical path delay is only 3.7ns based on TSMC 0.25um library. The main reason is that the proposed method regards execution status of instructions as basic granularity to generate the FPU control information, which simplifies significantly the complexity of control logics. Compared with scheme based on micro-instruction code, the delay decreases by 37.3% at typical corner [18]. From Fig. (**6**) we can draw similar conclusion based on other VDF and TDF. The proposed on method can actually lead significant induction in critical path delay.

Then, the evaluation of hardware overhead is done and the synthesis results obtained using the same FPGA device as Cortex-M1 integration is shown in Table **2**. The overheads of LUTs and Flip Flops are 3585 and 1594 respectively, which declines by 16.9% compared with Cortex-M1[14]. The reason why hardware overhead uses less is that destination operand write-back in pipeline is adopted, which reuses many hardware resources. However, LUTs and registers consumption of the proposed method increased by 9.6% and 9% respectively compared with GRFPU LITE [19]. It mainly contributes is that GRFPU LITE only support integration of the single and double precision FPU whereas the proposed method is suitable for single, double and ultra-precision FPU.

Finally, we evaluate the floating-point efficiency of the proposed method and further compare the results with publish mechanisms elsewhere [14, 17, 26]. The results are shown in Fig (**7**). It takes 173 clock cycles for the LEON3 FPU to finish single and double precision floating-point operation as the LEON3 FPU is a slave unit of on chip bus, one way of loose coupling, and software intervention is needed in the operation.

The GRFPU and GRFPULITE of Gaisler research spend closely 30 clock cycles to complete the single and double

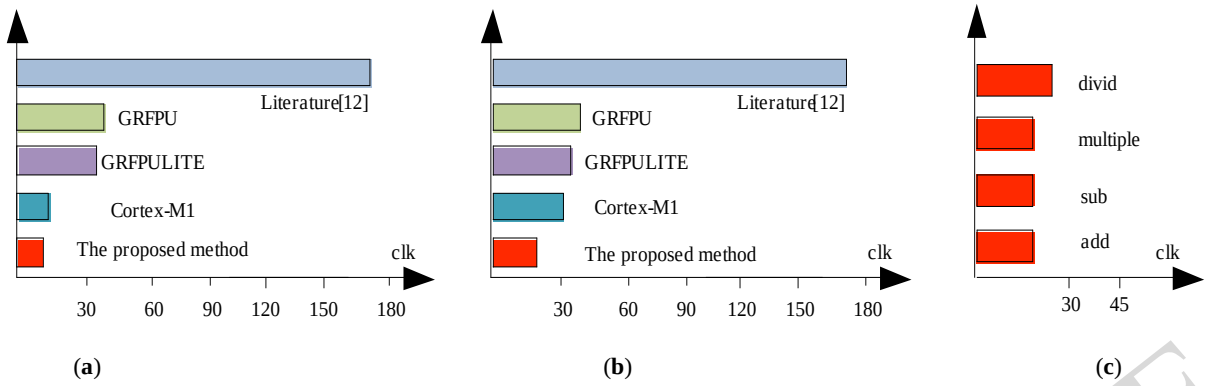


Fig. (7). Efficiency comparison of floating point computation (**a**) the number of clock needed by single-precision (**b**) the number of clock needed by double-precision (**c**) the number of clock needed by ultra-precision in proposed method.

precision operation. The improvement of efficiency is the result of embedding FPU into processor core and implementing tightly coupling between processor and FPU.

The Cortex-M1 needs about 15~30 clocks to finish the execution of floating-point instructions by optimizing the interaction way between FPU and processor. However, the proposed method generates the FPU control information based on execution status, which advances the execution information to next pipeline stage during each clock and embeds FPU into processor cores by hardware. Thus, communication overheads between FPU and processor can be ignored. As the result of all factors, the proposed just need 9~10 clocks finish single and double-precision floating point instructions. The floating-point calculation efficiency increases 1.7 times than Cortex-M1.

The ultra-precision floating point operation is addressed by software imitating floating-point computing in mainstream embedded processors, which spends thousand of clock cycles. Fig. (**7c**) gives the ultra-precision floating point clock overheads of V8 processor based on the proposed method, and the efficiency is 20~100 times higher than software ultra-precision floating-point emulation [14, 19].

CONCLUSION

This paper proposes a fine-grained integration method for ultra-precision FPU, which based on centralized control and data segmentation. The method generates the FPU control information corresponding to execution status and writes destination operands through pipeline, which can integrate 80-bit FPU to pipeline processor. The SPARC V8 processor with 80-bit FPU based on the proposed mechanism has been implemented, verified and analyzed. The results show that the critical path of floating instructions decreases by 37.3%, hardware consumption declines 16.9% and the floating-point calculation efficiency increase 1.7 times. This method can be used to embed ultra-high precision FPU to RISC processors.

Nevertheless, it is recognized that there are limitations in the integration method for ultra-precision FPU. The efficient floating-exception handling, structure of register file and implementation in multi-core processor have not been considered here and there are some limitations in the assumptions used in this study. Therefore, improvement of the FPU integration method based on the proposed mechanism is actually in progress in our study.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

The research work is supported by the National High-Tech Research and Development Program of China(No. 2010ZX01021, No. 2009ZX01023).

REFERENCES

[1] K. W. Chau, “Application of a PSO-based neural network in analysis of outcomes of construction claims”, *Automation in Construction*, vol.16, no.5, pp. 642-646, 2007.

[2] J. Zhang and K. W. Chau, "Multilayer ensemble pruning *via* novel multi-sub-swarm particle swarm optimization", *Journal of Universal Computer Science*, vol.15, no.4, pp.840-858, 2009.

[3] C. T. Cheng, K. W. Chau, Y. G. Sun and J. Y. Lin, "Long-term prediction of discharges in Manwan Reservoir using artificial neural network models", *Advances in Neural Networks–ISNN 2005*, vol. 3498, Springer Berlin Heidelberg, 2005, pp.1040-1045.

[4] R. Taormina and K. W. Chau, "Neural network river forecasting with multi-objective fully informed particle swarm optimization", *Journal of Hydroinformatics*, vol.17, no.1, pp. 99-113, 2015.

[5] C. L. Wu, K. W. Chau and Y. S. Li. "River stage prediction based on a distributed support vector regression", *Journal of Hydrology*, vol. 358, no.1, pp. 96-111, 2008.

[6] Z. K. Huang and K. W. Chau, "A new image thresholding method based on Gaussian mixture model", *Applied Mathematics and Computation*, vol. 205, no.2, pp.899-907, 2008.

[7] D. H. Bailey, "High-precision floating-point arithmetic in scientific computation", *Computing in science & engineering*, vol.7, no.3, pp. 54-61, 2005.

[8] C. Wang, *Realization of adaptive floating-point multiplication, division and square root unit for single, double and extended Precision*, North China Electric Power University, Beijing, vol. 1, pp. 3-10, 2011.

[9] Aeroflex, *UT699 LEON 3FT/SPARC V8 Microprocessor Functional Manual*, Aeroflex Inc, USA 2012.

[10] G. Kane and J. Heinrich, *MIPS RISC architectures*, Prentice-Hall, Inc, 1992.

[11] M. Boersma, M. Kroner and C. Layer, "The POWER7 binary floating-point unit", In: *20th IEEE Symposium on Computer Arithmetic (ARITH)*, 2011, pp. 87-91.

[12] M. George, P. Elbro and J. Johnson, "Design of high-integration controller", *IEEE Transactions on Consumer Electronics*, vol.43, no. 4, pp. 85-92, 1997.

[13] A. Ramakrishnan and J. M. Conrad, "Analysis of floating point operations in microcontrollers", In: *Proceedings of IEEE on Southeastcon*, 2011, pp. 97-100.

[14] J. Joven, P. Strict and D. R. Castells, "HW-SW implementation of a decoupled FPU for ARM-based Cortex-M1 SoCs in FPGAs", In: *6th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pp.1-8, 2011.

[15] J. Gaisler, *The LEON-2 Processor User's Manual*, Version, 1.0.10, 2003.

[16] E. M. Schwarz, M. Schmookler and S. D. Trong, "FPU implementations with denormalized numbers", *IEEE Transactions on Computers*, vol.54, no.7, pp. 825-836, 2005.

[17] S. D. Trong, M. S. Schmookler and E. M. Schwarz, "P6 Binary Floating-Point Unit", In: *2007 18th IEEE Symposium on Computer Arithmetic(ARITH)*, pp.77-86, 2007.

[18] Y. Zhao, S. Zhang and D. Wang, "The Integration of floating point IP in microprocessor design", *Microelectronics & Computer*, vol.23, no.7, pp.129-133, 2006.

[19] G. Nagendra, N. M. Devahsrayee and K. S. Dasgupta, "Scalable LEON 3 based SoC for multiple floating point operations", In: *IEEE Nirma University International Conference on Engineering (NUiCONE)*, 2011, pp.1-3.

[20] X. Du and X.Jin, "Design and verification of vector floating point coprocessor VFP-A", *Microelectronics*, vol.39, no.5, pp.597-601, 2009.

[21] C. Brunelli, F. Campi and J. Kylliainen, "A reconfigurable FPU as IP component for SoCs", In: *IEEE International Symposium on System-on-Chip*, 2004, pp.103-106.

[22] P. J. Pingree, J. F. L. Blavier and G. C. Toon, "An fpga/soc approach to on-board data processing enabling new mars science with smart payloads", In: *IEEE Aerospace Conference*, 2007, pp. 1-12.

[23] C. D. Wait, "IBM PowerPC 440 FPU with complex-arithmetic extensions", *IBM Journal of Research and Development*, vol. 49, no.2.3, pp. 249-254, 2005.

[24] J. L. Hennessy and A. P. David, *Computer Architecture: a Quantitative Approach*, Elsevier, Amsterdam 2012.

[25] S. Internationa, *The SPARC architecture manual Version 8*, SPARC International Inc, US 1998.

[26] J. Gaisler, *GRLIB IP Core User's Manual*, Gaisler research, Sweden 2007.

Received: March 07, 2015 Revised: March 29, 2015 Accepted: May 20, 2015