International Journal for Innovative Engineering and Management Research

A Peer Revieved Open Access International Journal

www.ijiemr.org

Title: **LOW POWER TECHNIQUE FOR VECTOR PROCESSING -AWARE ADVANCED CLOCK-GATING TECHNIQUES IN FUSED MULTIPLY-ADD**

Volume 09, Issue 06, Pages: 100-107

Paper Authors

SREE VIDYA, T.NAGAVENI

LOW POWER TECHNIQUE FOR VECTOR PROCESSING -AWARE ADVANCED CLOCK-GATING TECHNIQUES IN FUSED MULTIPLY-ADD

SREE VIDYA[1], T.NAGAVENI[2]
[1]PG Scholar, Dept of ECE, SIR C.V. RAMAN Institute of Technology & Science, AP, India
[2] Assistant Professor, Dept of ECE, SIR C.V. RAMAN Institute of Technology & Science, AP, India

ABSTRACT: The need for power efficiency is driving a rethink of design decisions in processor architectures. While vector processors succeeded in the high-performance market in the past, they need a retail ring for the mobile market that they are entering now. Floating-point (FP) fused multiply-add (FMA), being a functional unit with high power consumption, deserves special attention. Although clock gating is a well-known method to reduce switching power in synchronous designs, there are unexplored opportunities for its application to vector processors, especially when considering active operating mode. In this research, we comprehensively identify, propose, and evaluate the most suitable clock-gating techniques for vector FMA units (VFUs). These techniques ensure power savings without jeopardizing the timing. We evaluate the proposed techniques using both synthetic and "real-world" application-based benchmarking. Using vector masking and vector multilane-aware clock gating, we report power reductions of up to 52%, assuming active VFU operating at the peak performance. Among other findings, we observe that vector instruction-based clock-gating techniques achieve power savings for all vector FP instructions. Finally, when evaluating all techniques together, using "real-world" benchmarking, the power reduction. Additionally, in accordance with processor design trends, we perform this research in a fully parameterizable and automated fashion.

1. INTRODUCTION

Although computer arithmetic is sometimes viewed as a specialized part of CPU design, still the discrete component designing is also a very important aspect. A tremendous variety of algorithms have been proposed for use in floating-point systems. Actual implementations are usually based on refinements and variations of the few basic algorithms presented here. In addition to choosing algorithms for addition, subtraction, multiplication, and division, the computer architect must make other choices. Our discussion of floating point will focus almost exclusively on the IEEE floating-point standard (IEEE 754) because of its rapidly increasing acceptance. Although floating-point arithmetic involves manipulating exponents and shifting fractions, the bulk of the time in floating-point operations is spent operating on fractions using integer algorithms. Thus, after our discussion of floating point, we will take a more detailed look at efficient algorithms and architectures.

Many applications require numbers that aren't integers. There are a number of ways that non-integers can be represented. Adding two such numbers can be done with an integer add, whereas multiplication requires some extra shifting. There is various ways to represent the number systems. However, only one non-integer representation has gained widespread use, and that is floating point. Current digital signal processing (DSP) systems are making the transition from fixed-point arithmetic (used initially because of its simplicity) to floating-point arithmetic. The latter has several advantages including the freedom from overflow and underflow and ease of interfacing to the rest of the system. To improve the performance of floating-point arithmetic, several fused floating-point operations have been introduced: Fused Multiply–Add (FMA) used Add–Subtract, and Fused Two-Term Dot-Product. The fused floating-point operations not only improve the performance, but also reduce the area and power consumption compared to discrete floating-point implementations

This design presents improved architecture designs and implementations for a fused floating-point add–subtract unit. Many DSP applications such as fast Fourier Transform (FFT) and Discrete Cosine Transform (DCT) butterfly operations can benefit from the fused floating-point add–subtract unit. Therefore, he improved fused floating-point add–subtract unit will contribute to the next generation floating-point arithmetic and DSP application development. The proposed fused floating-point add–subtract unit takes two normalized floating-point operands and generates their sum and difference simultaneously. It supports all five rounding modes specified in IEEE-754 Standard. Several techniques are applied to achieve low area, low power consumption and high speed:

1) Instead of executing two identical floating-point adders, the fused floating-point add–subtract unit shares the common logic to generate the sum and difference simultaneously. Therefore, it saves much of the area and power consumption compared to a discrete floating-point add–subtract unit.

2) A dual-path algorithm can be applied to increase speed. The dual-path logic consists of a far path and a close path. In the far path, the addition, subtraction and rounding logic are performed in parallel. By aligning the significands to the minimal number of bits, the addition, subtraction and rounding logic are simplified. There are three cases for the close path depending on the difference of the exponents. For each case, addition, subtraction and leading zero anticipation **(LZA)** are performed in parallel and rounding is not required. Therefore, the dual-path design reduces the latency of the critical path.

3) To increase the throughput, pipelining can be applied. Based on data flow analysis, the proposed dual-path design is split into two pipeline stages. By properly arranging the components, latencies of the two pipeline stages are balanced so that the throughput of the entire design is increased. Also, it reduces the latency by simplifying the control signals.

Clock gating is a common method to reduce switching power in synchronous pipelines.. It is practically a standard in low-power design. The goal is to “gate” the clock of any component whenever it does not perform useful work. In that way, the power spent in the associated clock tree, registers, and the logic between the registers is reduced. It is the most efficient power reduction technique for active operating mode.1 Therefore, the conditions under which clock gating can be applied should be extensively studied and identified. A widely used approach is to clock gate a whole FU when it is idle. A complementary and more challenging approach is clock gating the FU or its subblocks when it is active, i.e., operating at peak performance

Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators by Y. Lee *et al* We present a taxonomy and modular implementation approach for data-parallel accelerators, including the MIMD, vector-SIMD,subword-SIMD, SIMT, and vector-thread (VT) architectural design patterns. We have developed a new VT micro architecture, Maven, based on the traditional vector-SIMD micro architecture that is considerably simpler to implement and easier to program than previous VT designs. Using an extensive design-space exploration of full VLSI implementations of many accelerator design points, we evaluate the varying tradeoffs between programmability and implementation efficiency among the MIMD, vector-SIMD, and VT pat-terns on a workload of micro benchmarks and compiled application kernels. We find the vector cores provide greater efficiency than the MIMD cores, even on fairly irregular kernels. Our results suggest that the Maven VT micro architecture is superior to the traditional vector-SIMD architecture, providing both greater efficiency and easier programmability.

Deterministic clock gating for microprocessor power reduction by H. Li, S. Bhunia, Y. Chen, T. N. Vijaykumar, and K. Roy With the scaling of technology and the need for higher performance and more functionality, power dissipation is becoming a major bottleneck for microprocessor designs. Pipeline balancing (PLB), a previous technique, is essentially a methodology to clock-gate unused components whenever a program's instruction-level parallelism is predicted to be low. However, no nonpredictive methodologies are available in the literature for efficient clock gating. This paper introduces deterministic clock gating (DCG) based on the key observation that for many of the stages in a modern pipeline, a circuit block's usage in a specific cycle in the near future is deterministically known a few cycles ahead of time. Our experiments show an average of 19.9% reduction in processor power with virtually no performance loss for an 8-issue, out-of-order superscalar processor by applying DCG to execution units, pipeline latches, D-Cache wordline decoders, and result bus drivers. In contrast, PLB achieves 9.9% average power savings at 2.9% performance loss.

Deterministic clock gating to eliminate wasteful activity due to wrong-path instructions in outof-order superscalar processors by N. Mohyuddin, K. Patel, and M. Pedram In this paper we present deterministic clock gating schemes for various micro architectural blocks of a modern out-of-order superscalar processor. We propose to make use of (1) idle stages of the pipelined function units (FUs) and (2) wrong-path instruction execution during branch mis-prediction, in order to clock gate various stages of FUs. The baseline Pipelined Functional unit Clock Gating (PFCG), presented for evaluation purpose only, disables the clock on idle stages and thus results in 13.93% chip-wide energy saving. Wrong-path instruction Clock Gating (WPCG) detects wrong-path instructions in the event of branch mis-prediction and prevents them from being issued to the FUs, and subsequently, disables the clock of these FUs along with reducing the stress on register file and cache. Simulations demonstrate that more than 92% of all wrong-path instructions can be detected and stopped from being executed. The WPCG architecture results in 16.26% chip-wide energy savings which is 2.33% more than that of the baseline PFCG scheme.

What every computer scientist should know about floating-point arithmetic, by D. Goldberg Floating-point arithmetic is considered an esoteric subject by many people. This is rather surprising, because floating-point is ubiquitous in computer systems: Almost every language has a floating-point data type; computers from PCs to supercomputers have floating-point accelerators; most compilers will be called upon to compile floating-point algorithms from time to time; and virtually every operating system must respond to floating-point exceptions such as overflow This paper presents a tutorial on the aspects of floating-point that have a direct impact on designers of computer systems. It begins with background on floating-point representation and rounding error, continues with a discussion of the IEEE floating-point standard, and concludes with examples of how computer system builders can better support floating point,

2. VECTOR PROCESSORS

Vector processors operate on vectors of data within the same instruction.2 Vector instruction set architecture (ISA) provides an efficient organization for controlling a large amount of computation resources. Furthermore, vector ISAs emphasize local communication and provide excellent computation/area ratios. Vector instructions express DLP in a very compact form, thus removing much redundant work (e.g., instruction fetch, decode, and issue). For example, a vector FP FMA instruction (FPFMAV) indicates the operation (FMA), three source vector registers, and one destination vector register. Thus, tuples of three elements, one from each source register,

are the inputs for the VFU, and the result is written to the destination. All tuples can be processed independently, and multiple elements could be accommodated in a vector register. The register file is designed so that a single named register holds a number of elements. The entire architecture is designed to take advantage of the vector style in organizing data. Additionally, the memory system of vector processors allows efficient stride and indexed memory access. The number of elements of a vector register is denoted by the maximum vector length (*MVL*). Occasionally, fewer elements than the *MVL* are used, which reduces the effective vector length (*EVL*). The vector execution model streamlines one vector register element per cycle to a fully pipelined vector FU. As a result, the execution time of a vector instruction is the startup latency (a number of stages) of the vector FU plus the *EVL*.A common technique to reduce this time is to implement multiple vector lanes through replicated lock-stepped vector FU. Each lane accesses its own "slice" of the vector register file, which reduces the need for increasing the number of ports typically associated with a larger number of FUs. Lock stepping the lanes simplifies the control logic and is power efficient. These concepts are shown in Fig. 1. Although lanes were proposed for increasing performance, using multiple lanes can increase the energy efficiency of a vector architecture. Additionally, an interesting feature that vector processors typically offer is a vector mask control. Masked operations are used to vectorize loops that include conditional statements. Masked operation uses an *MVL* -bit vector mask register (VMR) for indicating which operations of the vector instruction are actually performed. In other words, masked vector instructions operate only on the vector elements whose corresponding entries in the VMR are "1."

Conventional vector processors should not be confused with single instruction multiple data (SIMD) multimedia extensions such as AVX-512 that are an alternative way to exploit DLP and indicate operations to perform on multiple elements.3 The main difference of these extensions with a conventional vector processor is that they exploit subword-SIMD parallelism and are typically implemented with multiple vector FUs that operate on all independent elements in parallel. Having a vector FU per element to operate on all them in parallel would be inefficient for vector processors because they operate on much longer vectors. Instead, a vector FU is fully pipelined, and the elements of the vector register are streamlined to the unit, one per cycle, possibly using a small number of vector lanes.

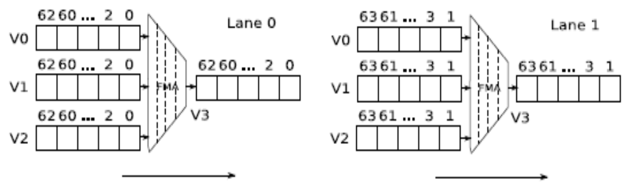

Fig. 1. Two-lane, four-stage VFU (*MVL* = *EVL* = 64) executing FPFMAV V3<-V0,V1,V2.

3. FMA

The FMA unit executes the FMA instruction (FMA R <- A, B, C) that implements $R = A * B + C$. In contrast to a multiplication followed by an addition, the FMA instruction assumes all three operands at the same time. It was introduced for the first time in IBM's RS/600 in 1990. IEEE754-2008 standard defines the FMA instruction to be computed initially with unbounded range and precision, rounding only once to the destination format. For this reason, FMA is faster

and more precise than a multiplication followed by an addition. The FMA unit performs operand alignment in parallel with the multiplication. This leads to shorter latency (*nS*) compared with a multiplication followed by an addition. Additionally, the FMA operation reduces the number of interconnections between FP units and the number of adders and normalizers. The FMA instructions help compilers to produce more efficient code. Potential drawbacks are increased latency of FPADD and FPMUL instructions (if executed on the FMA) and a complex normalizer. A simplified list of steps of the computations the FMA flow is as follows:

1) mantissas multiplication ($MA \times MB$), exponents addition ($EA + EB$), alignment of the addend's mantissa (MC), and calculation of the intermediate result exponent $ER = \max(EA + EB, EC)$;
2) addition of the product ($MA \times MB$) and aligned MC ;
3) normalization of the addition result and exponent update;
4) rounding;
5) determination of the exception flags and special values.

A simplified implementation block diagram of the FMA unit used in our research is shown in Fig. 2. As we assume double precision, we need a 162-bit adder and a 53 × 53 multiplier. For the adder and the multiplier, we choose Brent–Kung and Wallace algorithms, respectively, as it is aligned with our findings. The aligner performs shifting of the addend based on the exponents difference in order to align it with the product ($MA \times MB$). FP addition using the FMA unit is implemented by setting the first operand to 1 (A = 1.0), while FP multiplication is implemented by setting the third operand to 0 ($C = 0.0$).

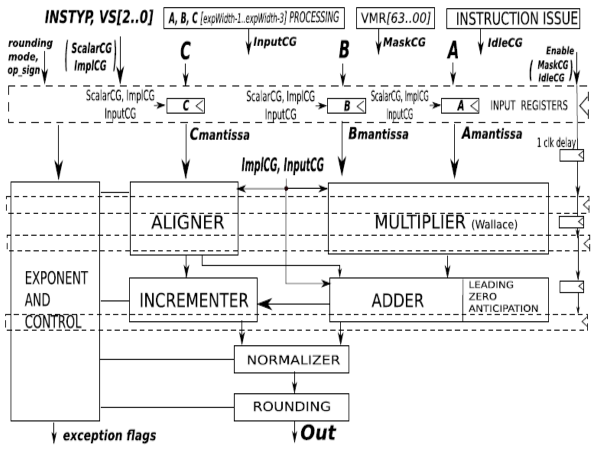

Fig. 2. Simplified block diagram of a one-lane, four-stage VFU with all clock-gating techniques applied (AllCG technique).

4. CONCLUSION:

Traditional clock gating approaches reduce FPU power consumption if no instructions are executed, or at best, re-duce the power consumption for the idle cycles between subsequent instructions. In numerical applications with highly optimized floating-point routines these traditional clock gating schemes are not efficient for the FPU. We have developed new clock gating schemes that address exactly this scenario, i.e., they save power even if the FPU executes an instruction every cycle. The schemes clock gate parts of the FPU based on instruction type, precision, and operand values..

BIBLIOGRAPHY

[1] I. Ratkovi´c, O. Palomar, M. Stani´c, O. Unsal, A. Cristal, and M. Valero, "A fully parameterizable low power design of vector fused multiply-add using active clock-gating techniques," in Proc. Int. Symp. Low Power Electron. Design (ISLPED), 2016, pp. 362–367.

[2] K. Asanovi´c, "Vector microprocessor," Ph.D. dissertation, Dept. Comput. Sci., Univ. California, Berkeley, Berkeley, CA, USA, 1998.

[3] Y. Lee et al., "Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators," in Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA), 2011, pp. 129–140.

[4] B. Zimmer et al., "A RISC-V vector processor with simultaneousswitching switched-capacitor DC–DC converters in 28 nm FDSOI," IEEE J. Solid-State Circuits, vol. 51, no. 4, pp. 930–942, Apr. 2016.

[5] R. Espasa, M. Valero, and J. E. Smith, "Vector architectures: Past, present and future," in Proc. 12th Int. Conf. Supercomput. (SC), 1998, pp. 425–432.

[6] H. Li, S. Bhunia, Y. Chen, T. N. Vijaykumar, and K. Roy, "Deterministic clock gating for microprocessor power reduction," in Proc. 9th Int. Symp. High-Perform. Comput. Archit. (ISCA), Feb. 2003, pp. 113–122.

[7] N. Mohyuddin, K. Patel, and M. Pedram, "Deterministic clock gating to eliminate wasteful activity due to wrong-path instructions in outof- order superscalar processors," in Proc. IEEE Int. Conf. Comput. Design (ICCD), Oct. 2009, pp. 166–172.

[8] J. Preiss, M. Boersma, and S. M. Mueller, "Advanced clockgating schemes for fused-multiply-add-type floating-point units," in Proc. 19th IEEE Symp. Comput. Arithmetic (ARITH), Jun. 2009, pp. 48–56.

[9] I. Ratkovi´c, O. Palomar, M. Stani´c, O. S. Ünsal, A. Cristal, and M. Valero, "On the selection of adder unit in energy efficient vector processing," in Proc. 14th Int. Symp. Quality Electron. Design (ISQED), Mar. 2013, pp. 143–150.

[10] I. Ratkovi´c et al., "Joint circuit-system design space exploration of multiplier unit structure for energy-efficient vector processors," in Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI), Jul. 2015, pp. 19–26.

[11] M. Stanic, O. Palomar, I. Ratkovi´c, M. Duric, O. Unsal, and A. Cristal, "VALib and SimpleVector: Tools for rapid initial research on vector architectures," in Proc. 11th ACM Conf. Comput. Frontiers (CS), 2014, p. 7.

[12] (2016). ARM Processors. [Online]. Available: http://arm.com/

[13] S. Momose, "NEC Vector supercomputer: Its present and future," in Sustained Simulation Performance. Cham, Switzerland: Springer, 2015, pp. 95–105.

[14] R. Espasa et al., "Tarantula: A vector extension to the alpha architecture," in Proc.

29th Annu. Int. Symp. Comput. Archit. (ISCA), 2002, pp. 281–292.

[15] O. Shacham, O. Azizi, M. Wachs, S. Richardson, and M. Horowitz, “Rethinking digital design: Why design must change,” IEEE Micro, vol. 30, no. 6, pp. 9–24, Nov./Dec. 2010.

[16] B. Nikolic, “Simpler, more efficient design,” in Proc. 41st Eur. Solid-State Circuits Conf. (ESSCIRC), 2015, pp. 20–25.

[17] J. Bachrach et al., “Chisel: Constructing hardware in a scala embedded language,” in Proc. 49th Annu. Design Autom. Conf. (DAC), 2012, pp. 1216–1225.