A NEW APPROACH FOR BLOCK-FLOATING-POINT ARITHMETIC

Shiro Kobayashi and Gerhard P. Fettweis

Mobile Communications Systems
Dresden University of Technology
01062 Dresden, Germany
{shiro, fettweis}@ifn.et.tu-dresden.de

ABSTRACT

A new approach for implementing block-floating-point arithmetic is proposed. This approach intends to preserve the least-significant-bits (LSBs) to improve signal processing quality. The preservation of LSBs is automatically and perfectly done by hardware. Several simulation results of the proposed block-floating-point implementation have shown improved SNRs over conventional block-floating-point implementation as expected. For the same number of bits for each data representation in the memory, the SNRs better than floating-point are also observed.

1. INTRODUCTION

Both the complexity and the amount of algorithms assigned to digital signal processors (DSP) are rapidly increasing. Also, in order to adapt to this situation of increase in performance requirements, a number of DSPs are starting to employ multiple arithmetic datapath architectures. These multiple datapath architectures provide programmers with promised high performance only if every datapath is controlled properly. This fact makes the programming itself more complicated and, subsequently, time consuming. On the other hand, the development time of the program code is still a major concern. The traditional approach of assembly coding does not appear to be working any more. Hence, it can be expected that more program code will be developed with high-level languages such as C, C++, Ada, or Java.

The application of high-level languages has not been successful in most commercial DSPs except for 32-bit floating-point DSPs. However, if chip size and power consumption are major concerns, it does not seem realistic to equip a single DSP chip with multiple 32-bits floating-point arithmetic components.

Hence, it is necessary to investigate the appropriate arithmetic system which allows the effective application of high-level languages with acceptable chip cost. Short-word floating-point arithmetic is, of course, one possibility. The fact that the arithmetic of floating-point DSPs matches with the float type of the C language makes the application of C compiler straightforward and thus relatively easy. However, the signal processing quality of short-word floating-point has not been studied well, and there are some questions about its arithmetic performance [1]. Block-floating-point also seems to be a very attractive solution. In view of chip size, a block-floating-point DSP can be as small as a fixed-point DSP while allowing higher signal processing quality. However, again, it does not seem to have been studied well towards its application to commercial DSPs.

This paper is based on a novel idea of block-floating-point implementation, named hierarchical block-floating-point [2]. In this paper, some theoretical background of hierarchical block-floating-point is newly added. Additional simulation results are also presented.

The outline of the following part of this paper is: In Section 2, signal processing qualities of several different implementations of block-floating-point are discussed. The idea of hierarchical block-floating-point is also introduced. The discussion on the performance of arithmetic systems has been confirmed by simulation. The results are given in Section 3. Conclusions based on the discussion and simulations are summarized in Section 4.

2. BLOCK-FLOATING-POINT ARITHMETIC BACKGROUND

The analysis of signal quality can be measured by the signal-to-quantization noise ratio (SNR), where each step of rounding reduces the SNR accordingly. Take an example of the multiply-accumulate operation with each product rounded shown in Figure 1. This operation includes three quantizers: an input signal quantizer, a product quantizer, and an output quantizer with SNRs of SNR_{inp}, SNR_{prd}, and SNR_{out} respectively. The accumulation operation, i.e., addition, does not need to be considered, because it does not require quantization.

The SNR of a (B+1)-bit quantizer is given in [7] as

$$SNR = 6.02B + 10.79 - 20\log_{10}(\frac{1}{A_{rms}}) \qquad (1)$$

where the quantity A_{rms} is the rms value of the signal amplitude written as the ratio to the full-scale value of the quantizer.

Block-floating-point ideally outperforms fixed-point, since its input signals are always block normalized, and thus always the same large rms input signal amplitude can be expected for a random signal sequence. However in reality, depending on how a block-floating-point arithmetic is implemented, each partial SNRs shown in Figure 1 degrades substantially by losing information in least-significant-bits (LSBs) through rounding. In the following two sub-sections, the effect of losing LSBs is discussed.

2.1. Losing LSBs caused by avoiding accumulator overflow

How to deal with possible overflow during accumulation in multiply-accumulate operation is an important issue of fixed-point

This work was supported in part by Asahi Chemical, Tokyo, Japan, Siemens Semiconductor, Munich Germany, and the German Federal Ministry.

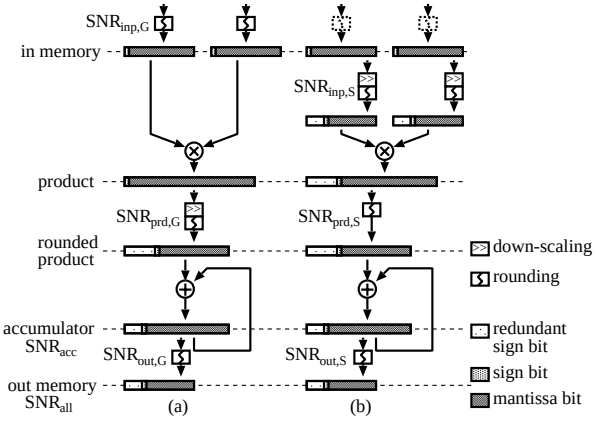

Figure 1: Noise model of multiply-accumulate operation with product rounding: a) Guard-Bits and b) Input down-scaling.

arithmetic. Since block-floating-point realization exploits a fixed-point computational datapath, this accumulator overflow problem needs to be considered. The risk of overflowing the accumulator is reduced by introducing some extra bits of higher significance than the most-significant-bit (MSB) of the multiplier output. These extra bits are designated *guarding-portion* in the following part of this paper. Such guarding-portion is usually realized by either explicitly introducing extra bits into the accumulator, called guard-bits, or scaling down the input of the multiplier, which accordingly moves the MSB of the multiplier output to a less significant location. Previous research of block-floating-point systems seem to have implied scaling-down the multiplier input [3],[4],[5],[6], while most of commercially available DSPs employ guard-bits.

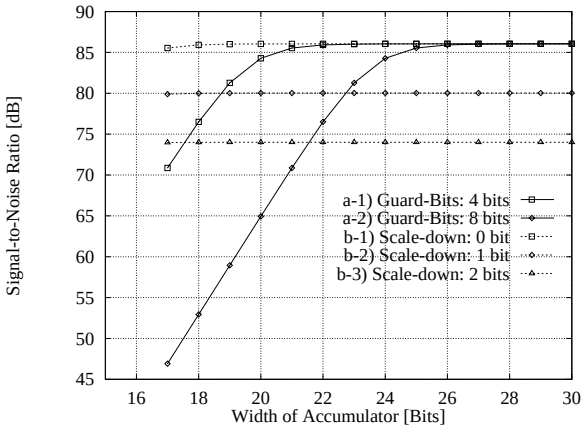

Figure 2: Theoretical SNRs of multiply-accumulate operation (SNR_{acc} in Figure 1: a) Guard-Bits and b) Input down-scaling. The multiplier input width, B in equation (1), is 15 bits.

The SNR of the result in the accumulator after multiply-accumulate operation with each product rounded shown in Figure 1 is given as

$$SNR_{acc} = SNR_{inp} - 20\log_{10}(2 + 10^{\frac{SNR_{inp} - SNR_{prd}}{20}}). \quad (2)$$

Figure 2 plots some example SNR curves from equation (2) for a fixed width of multiplier input and various accumulator widths. Each SNR_{inp} and SNR_{prd} in the equation (2) is calculated with the equation (1). When the multiplier input is scaled down, the width of the accumulator in the practical range of interest does not affect the overall SNR, SNR_{acc}. However, the SNR_{acc} varies based on the width of the guarding-portion, i.e., the amount of the scaling, to be implemented.

On the other hand, it is obvious that in case of the guard-bits approach, the effect of product rounding is negligible if the effective accumulator width is as many as 2 bits wider than the width of multiplier input. The effective accumulator width here is defined as the accumulator width excluding guard-bits. Thus, for the 16 bits multiplier input and 8 bits for guard-bits, which is a very common configuration in many commercial DSPs, an accumulator with total of 26 bits width is enough to keep the precision of the original multiplier input. Note that this is true only for block-floating-point oriented DSP where input signals are always block normalized. The width of the accumulator is in practical fixed in a DSP. Hence, the guard-bits approach accordingly promises a fixed SNR for all algorithms (except recursive algorithms) and wide range of input signal levels.

2.2. Preserving LSBs of computation result

The necessary width of the guarding-portion, defined in the previous sub-section, is determined through the analysis of the algorithm being implemented. This analysis is done by referring to filter coefficients or the number of accumulation, for example. However, such a simple analysis usually only gives the worst case estimation resulting in a too wide guarding-portion. Some precise analysis is also possible, though the precise analysis has to be specially designed for each algorithm. The analysis itself is also painstaking for achieving such preciseness. Hence, even when considering the higher likelihood of fully utilizing all guarding-portion of block-floating-point over fixed-point due to its normalized input, the probability of having unnecessary guarding-portion is not zero.

Let's think about how to store the computational results with some unnecessary bits in the guarding-portion. These unnecessary bits appear as redundant sign bits. In a block-floating-point system, each computational result in the accumulator is checked during its store operation to the memory in order to find how many redundant sign bits each result contains. However, it is not possible to remove these redundant sign bits from each computational result. The reason is that in a block-floating-point system, all data in a block have to be scaled by the same amount, called block-scale-factor, and hence all of the computational results in a block have to be checked before scaling any of them. Removing the redundant sign bits is done at the input of the next computation.

The computational results in the accumulator are thus stored into the memory with some redundant sign bits, while discarding corresponding amounts of LSBs. This is a very common implementation of block-floating-point on currently available commercial DSPs. The lack of LSBs causes some SNR degradation at the output of current computation. It also leads to similar SNR degradation to that caused by the down-scaling of the multiplier input during the next computation. Figure 3-a illustrates this implementation.

In order to avoid such performance degradation caused by discarding LSBs, another implementation is also widely used (Figure 3-b). With this implementation, each computational result in the accumulator is stored into the memory as double word, keeping

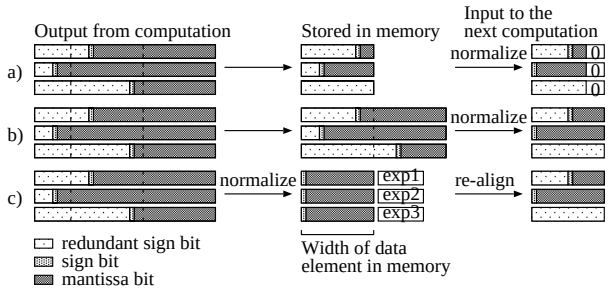

Figure 3: Different Block-Floating-Point Implementations: a) Conventional single-word, b) Conventional double-word, and c) Hierarchical.

the accumulator's double-word precision. The possibility of losing LSBs is drastically reduced. On the other hand, the required number of operation per data is at least doubled.

A novel implementation of block-floating-point, named hierarchical block-floating-point, also preserves the LSBs with much less memory requirement. Figure 3-c illustrates the idea briefly. Each computational result is independently normalized, and only the most significant single-word is stored into memory. A corresponding scale information is also assumed to be available. When these results are used for the next computation, each result is shifted to the right, or scaled down, by the difference of each scale factor from the block-scale-factor. At this point, all data in a block are aligned to the binary point of the largest value in the block. Not a single LSB is discarded. The representation of intermediate data in the memory is in fact the same as floating-point. However, the location of the binary point of each input data is aligned again before the next computation begins, and every computation is done with an inexpensive fixed-point datapath.

2.3. Hierarchical Block-Floating-Point on Multiple Datapath DSP

The advantage of hierarchical block-floating-point is limited if it is applied to a single datapath DSP. Even though the cost for the arithmetic datapath is low, the extra cost for keeping each scale factor in the memory is still expensive compared to a fixed-point implementation.

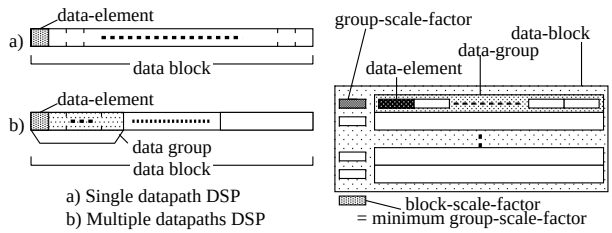

Figure 4: Data-group block-floating-point.

On a multiple datapath DSP, data elements fed into or received from the arithmetic datapath in parallel are likely to be treated as a *group* of several data elements. All data elements in a data-group are accessed together.

By introducing a data-group scheme and also a new joint scale factor for each data-group, i.e., group-scale-factor (See Figure 4), the memory cost for each scale-factor becomes smaller per data elements along with increased parallelism. In future DSPs with possibly at least 8 to 16 parallel computational datapaths, this cost becomes negligible. A hierarchical block-floating-point realized on such a data-group based DSP, named data-group block-floating-point, becomes cost effective even for the memory.

3. SIMULATION RESULTS

The signal processing quality of three different block-floating-point implementations and a short-word floating-point implementation have been checked by means of simulations.

The signal processing quality is measured by SNR defined as

$$SNR = 10 \log \frac{\sum_n reference(n)^2}{\sum_n (reference(n) - target(n))^2}$$

where reference(n) is a 64 bits double precision floating-point output signal sequence, and target(n) is an output computed with the target data representation. Input sequences are assumed to be Gaussian distributed.

The block-floating-point implementations are summarized in Table 1 following the discussion in Section 2. The discussion in Section 2 suggests that guard-bits outperform input down-scaling due to its fixed high SNR, and the normalization of computation results *before* storing may give higher SNR. The first example of block-floating-point employs guard-bits without the computation result normalization. This is one of the most conventional implementations which can be found on commercial DSPs with guard-bits. With this implementation, the analysis of application and input data properties determines the part of the accumulator to be stored into the memory. If the analysis is inaccurate, the stored result may contain redundant sign-bits (denormalized number). This implementation is shown as Guard-Denormalized Block-floating-point (GD-BFP) in Table 1. Scale-Denormalized Block-floating-point (SD-BFP) is also conventional, which employs input down-scaling instead of guard-bits. The amount of scaling is also determined through the analysis. The Guard-Normalized Block-floating-point (GN-BFP) in Table 1 exploits both ideas. This gives an example of the hierarchical block-floating-point implementation.

	Preserving LSBs of result	
	Yes	No
Guard-Bits	H-BFP(GN-BFP)	GD-BFP
Input Down-Scaling		SD-BFP

Table 1: Block-Floating-Point Implementations.

Figure 5 plots the SNRs of two 11-tap FIR filters whose coefficients require no guard-bits (above figure) and three bits (bottom figure) respectively for avoiding accumulator overflow. In Figure 5, W_{mem} is for the width of each data element in the memory, wg for the width of the guard-bits when a guard-bit approach is taken, sa for the amount of input down-scaling, and we for the width of exponent of floating-point, respectively. The widths of the mantissa in the accumulator are thus defined as $W_{acc} - wg$ for the block-floating-point with guard-bit, $W_{acc} - (2sa - 1)$ for the block-floating-point with input down-scaling, and $W_{acc} - we + 1$

for floating-point, respectively, where W_{acc} is the width of the accumulator. For all implementations except SW-FP, the W_{mem} is fixed to 16 bits, i.e., the double-word block-floating-point shown in Figure 3 is not included.

The SW-FP shown as d) provides almost the same performance for two different sets of coefficients. This fact suggests that the short-word floating-point can achieve some fixed SNR independently from the input signal level or coefficients. The results of two short-word floating-point with wider mantissa, SW-FP 15+4 bits and 13+4, are presented only to show the effect of mantissa length to the SNR. With these two data representations, the W_{mem} are 19 bits and 17 bits, respectively.

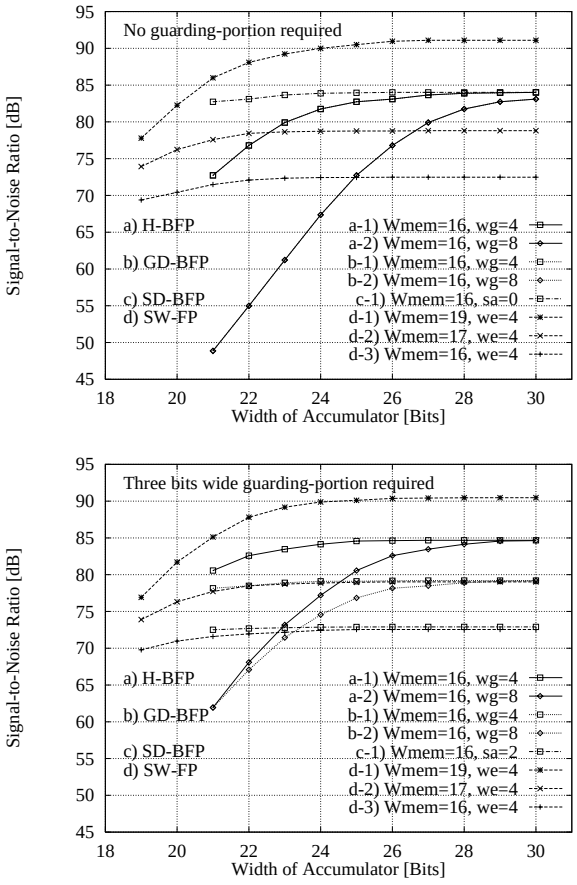

Figure 5: SNRs of two FIR-filters with no requirement for the guarding-portion (above) and three-bits wide guarding-portion required (bottom), respectively: a) Hierarchical Block-floating-point, b) Conventional Guard-Block-floating-point, c) Conventional Scale-Block-floating-point, and d) Short-Word Floating-point.

The coefficients and the signal level of input impacts on the SNR of block-floating-point system. The SNR of block-floating-point with guard-bits theoretically becomes better when the coefficients or input signals require more bits for avoiding accumulator overflow, because then each multiplication result utilizes the higher portion of the accumulator more efficiently, which results in a better SNR. The H-BFP simulation results of the two filters follow this expectation.

The above simple analysis is not enough to explain the observation of the GD-BFP results, which shows worse SNR for the filter with more guard-bits requirement. This is due to the specific of the simulation parameters where only one of the ten input signal sequences used actually required 3 bits of guard-bits and the others just 2 bits. The SNR performance degradation in the filter with potentially better performance is thus introduced by losing the LSBs when storing computation results into the memory.

Input down-scaling, on the other hand, leads to worse SNRs along with increase of the shift amount for avoiding accumulator overflow. This can be seen between two SD-BFP results. The filter which requires no guard-bits, thus no input down-scaling, gives better SNR than the other filter which requires 2 bits down-scaling for avoiding accumulator overflow.

4. CONCLUSION

This paper shows that the SNR performance of block-floating-point implementation can be improved through the preservation of LSBs. Such preservation is done at both the input and output of computation. At the input, employing the accumulator guard-bits saves more LSBs than down-scaling the input. A novel approach for preserving LSBs at the output, called hierarchical block-floating-point, is presented. This approach employs an automatic normalization and re-alignment of data. The block-floating-point which combines both LSB preservation approaches provides a fixed and best SNR over several different block-floating-point implementations for a fixed algorithm, regardless its signal levels and the coefficients. This fixed SNR is even higher than that of short-word floating-point for the same amount of bits for data representation.

The presented novel block-floating-point implementation has two good characteristics towards the application of a compiler. First, the proposed hierarchical block-floating-point always provides a fixed and best SNR performance. Second, the signal processing performance is always provided automatically by the hardware.

5. REFERENCES

[1] Phil Lapsley et al. *DSP Processor Fundamentals: Architectures and Features*. IEEE Press, 1997.

[2] Shiro Kobayashi and Gerhard Fettweis. A block-floating-point system for multiple datapath DSP. In *Proc. 1998 IEEE Workshop on Signal Processing Systems (SiPS'98)*, Boston, USA, October 1998.

[3] K. Kalliojärvi and J. Astola. Roundoff errors in block-floating-point systems. *IEEE Trans. Signal Processing*, 44(4):783–790, April 1996.

[4] S. Sridharan. Implementation of state-space digital filter structures using block floating-point arithmetic. In *Proc. ICASSP-87*, pages 908–911, 1987.

[5] D. Williamson, S. Sridharan, and P. G. McCrea. A new approach for block floating-point arithmetic in recursive filters. *IEEE Trans. Circuits and Systems*, CAS-32(7):719–722, July 1985.

[6] R. Frisch, B. Isenstein, and S. Stein. Arithmetic methods trade off precision for numerical range. *Electronic Product*, 27(21):75–78,80, April 1985.

[7] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Engelwood Cliffs, NJ: Prentice-Hall, 1989.