A SOFTWARE / HARDWARE CODESIGNED HANDS FREE SYSTEM ON A "RESIZABLE" BLOCK-FLOATING-POINT DSP

Shiro Kobayashi, Isamu Kozuka, Wai Hung Tang, and Diemo Landmann

Information Technology Laboratory, Asahi Kasei Corporation
E-mail: shiro@ljk.ag.asahi-kasei.co.jp

ABSTRACT

A hands free telephone application has been implemented on a low-cost, custom-configured, block-floating-point digital signal processor (DSP). The application consists of an acoustic echo canceller and a spectral subtraction (SS) based noise suppressor. The objective of pursuing a custom configuration was the minimization of hardware cost for the given application. For this objective, implementation has been carried out though a software / hardware codesign design flow on a resizable DSP platform. The intention of exploiting block-floating-point as arithmetic was to remove the burden of time-consuming fixed-point model development, while employing an inexpensive fixed-point DSP. This paper describes an implementation of the application. Signal processing quality evaluation results are also presented for some critical computation modules in the application.

1. INTRODUCTION

A configurable block-floating-point DSP platform has been applied to several embedded audio applications, having shown its potential of achieving high signal processing quality with low-complexity hardware and of completing application program developments without fixed-point models. Towards the first commercial utilization of the platform, a demanding hands free telephone application [1] has been chosen, and has been successfully implemented.The algorithm of the hands free telephone application has been originally developed as a middleware product in the Voice Interface Project at Asahi Kasei Corporation. Information about the middleware product as well as the algorithm itself is available at [4].

2. DSP PLATFORM

The employed DSP platform to implement the hands free telephone application is named as flexibly in-situ tailored DSP (FitDSP), implying that a final DSP can be configured on site during implementation of a given application to meet specific needs of the application. Another important feature of the FitDSP platform can be seen in its arithmetic. These two key features are explained in the following.

2.1. Hierarchical Block-Floating-Point

FitDSP exploits a new variation of block-floating-point (BFP), called hierarchical block-floating-point (H-BFP), as its native arithmetic. BFP is an arithmetic that combines the advantages of floating-point (FP) and fixed-point (FXP). With the concept of BFP, a set of plural data elements is treated as a data block. An exponent to reflect the maximal data value in a data block is referred to as block exponent.

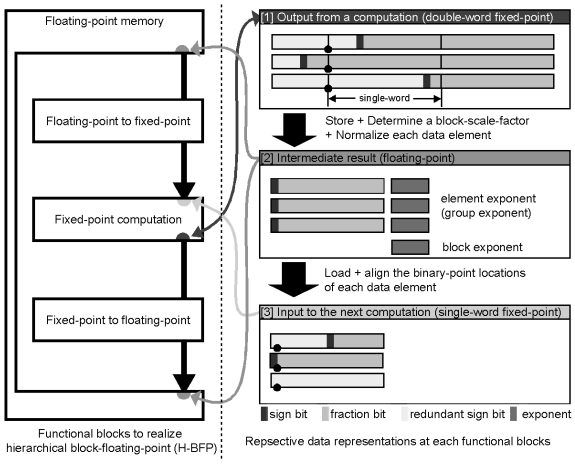

Figure 1: Concept of hierarchical block-floating-point (H-BFP). *Left*: Functional blocks to realize H-BFP, and *Right*: respective data representations in each functional block.

In theory, through block exponent, BFP can perform signal processing tasks with the same manner as FP does. This situation leads to high accuracy and large dynamic range. More importantly, the time-consuming task of a fixed-point model development can be removed from the development flow of an application program. On the other hand, necessary hardware complexity for the realization of BFP stays comparable with that for FXP.

A problem with BFP appears during its implementation. Due to a trade-off between the required number of memory access and achievable signal processing quality, actual implementations are in reality either very costly or with low signal processing quality. Hence, H-BFP has been proposed after challenging this problem [2][3].

The basic concept of H-BFP is explained with Fig. 1, which shows necessary functional blocks for a realization of H-BFP together with respective data representations in each functional block. H-BFP is realized by adding two more functional blocks to a usual fixed-point computation unit. These two functional blocks

are used to convert each data element between floating-point and fixed-point.

At the output of fixed-point computation unit, output data elements from a computation, i.e, computation results, are represented in a double-word fixed-point format. When they are stored into data memory, each of them is normalized individually through a shift operation to the most-significant-bit direction. In other words, each computation result is stored in data memory in a form that includes a normalized mantissa and a exponent, called element exponent. When a computation result is loaded from data memory for the following computation, its normalized mantissa is shifted to the least-significant-bit (LSB) direction. This shift operation to the LSB direction is performed by referring to the difference between each element exponent and block exponent such that binary-point location of all computation results in a data block is arranged to the binary-point location of the computation result with the maximal value.

2.2. Customizability

On the FitDSP platform, a custom DSP configuration can be designed to meet specific requirements of a given application. The requirements are specified in terms of 1) hardware cost such as gate counts, the amount of data memory, and the amount of instruction memory.

The software development tool set for FitDSP platform includes an assembler, a linker, and a debugger. Each tool is capable of reading a DSP configuration file such that any specific DSP configuration in terms of the number of DSP resources can be emulated. Thus, a signal processing engineer can explore a wide design domain for the most preferable solution for a given application. DSP resources which engineers can specify the numbers include computation units and all registers in the DSP platform. Also the number of block-loop unit can be set freely. Not only the numbers but also the word-length of those resources are also left variable.

In addition to consider those DSP core resources, it is also very important to take into account the size of data memory on the design of a custom chip solution. Hence, the tool set allowed that the mantissa word-length of each data element in data memory be set independently of the word-length of computation data path. With this feature, a programmer can evaluate a data memory configuration with a shorter word-length for specific data elements (or a data block) with a little impact on the overall signal processing quality.

Fig. 2 explains the software / hardware codesign design flow that has been taken. At the first stage, with the help of software development tool set, the algorithm is described in the assembly language without any limitation on DSP resources. Also in this stage, a missing functionality of the DSP platform has been identified. Then in the second stage, a short effort has been paid to minimize the number of DSP resources. With assembly language for FitDSP platform, programmers do not need to specify register resources with a specific index. They can, rather, tell assembler to allocate a register that is free at that moment. In order to enable the reuse of allocated registers, programmer need to explicitly declare the life of a register in assembly language syntax.

Finally in the last stage, a horizontal compression scheme is applied to the native VLIW ISA. Only necessary combinations of VLIW fields are identified in a given application program, and encoded into an application-specific horizontally compressed instruction word (HCIW) ISA

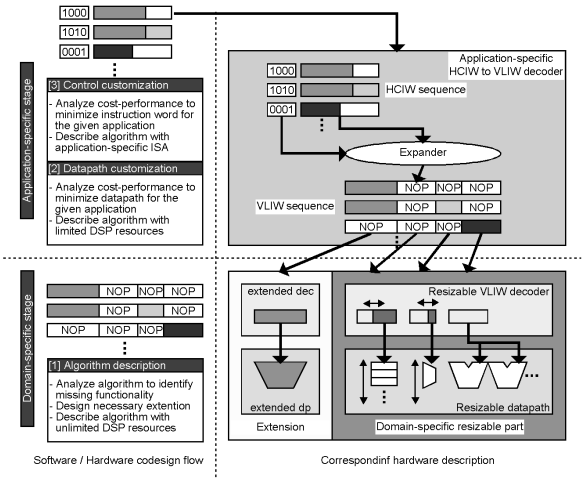

Figure 2: Overview of the exploited software / hardware codesign design flow. A custom DSP configuration is designed through a 3-stage customization step: i.e., [1] Identification of missing functionality and addition of corresponding DSP extensions, [2] Introduction of limitations into the number of DSP resources (resize), and [3] Introduction of limitations into the combinations of each instruction fields in a VLIW ISA.

3. DSP REALIZATION

3.1. DSP CONFIGURATION

Through the software / hardware codesign design flow explained in the previous chapter, a DSP configuration has been designed for the given hands free telephone application. The block diagram of resultant configuration, now named as FitDSP-piquant configuration, is presented in Fig. 3.

With the piquant configuration, four data elements are always jointly transferred as a data group between computation unit and data memory. The purpose of exploiting grouped data transfer scheme is twofold: first, to achieve high-rate data transfer, and second, to reduce the bit counts of data memory. A high-rate data transfer scheme is preferred in the echo canceller to implement a sample-wise adaptive filter where the local reuse of filter coefficients is not possible. A grouped data transfer scheme also contributes in an implementation of H-BFP to a reduction in the storage requirement for element exponents. Mantissas and exponents are stored in data memory in a manner shown in data memory in Fig. 3. The exponent field in a data group is also used for storing a block exponents. When a block exponent is stored, the mantissa fields of corresponding data group are not used.

After signal processing quality evaluation, the word-length of a mantissa and an exponent is set to 15 bits and 8 bits, respectively. A 15 bits mantissa from data memory is expanded to a 16 bits mantissa in the alignment shifter by holding the most-significant-bit of shifted-out bits as the least-significant-bit of new mantissa. Internal computations in fixed-point computation units are performed on 16 bits mantissas.

FitDSP-Piquant configuration has been implemented on an FPGA platform. The exploited FPGA is a Cyclone series EP1C20 device of Altera Corporation. In Cyclone series devices, circuits are constructed with configuration blocks called Logic Elements (LEs). Each LE contains a register and a Look-Up-

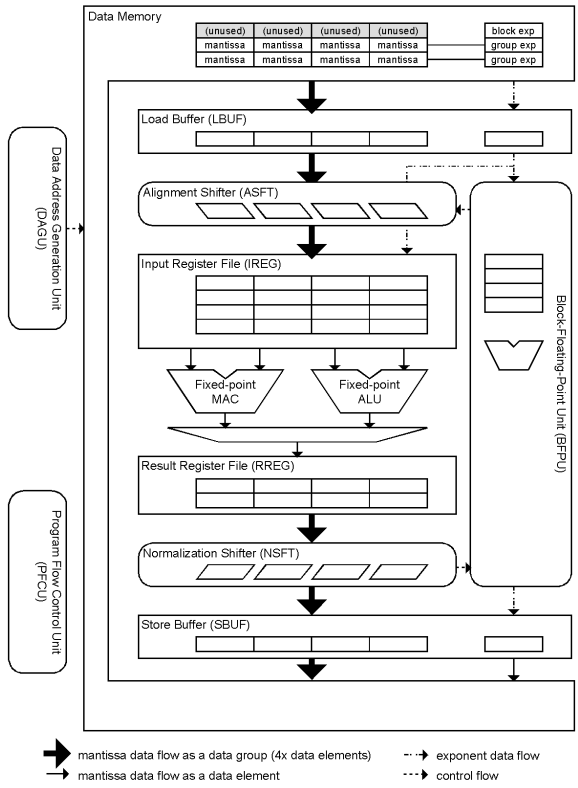

Figure 3: A block diagram of Piquant configuration of FitDSP.

Table (LUT) that can constitute a 4-inputs combinational logic. The number of LEs that are required for implementing FitDSP-Piquant configuration is summarized in Table 1.

Two kinds of numbers are presented in Table 1. They correspond to the different place-and-route strategies of Altera's development tool. With "auto register packed" option set to "normal", the LUT and the register in the same LE can be used to implement a related functionality. For a processor-kind of circuit where registers and combinational logic are tightly coupled, activation of this feature probably makes sense.

With this option set to "off", the LUT and the register in the same LE are usually not utilized at the same time. This setting is used to estimate the number of gates (gate count) with standard cell technologies. From the LE counts with "auto register packed" option set to "off", a gate counts with a 0.35 um standard cell library is estimated. For a register, a ratio of 10 gates per bit is assumed. For a combinational logic, gate count is assumed to be 4.55 gates per LE. This conversion ratio is based on the results from a previous development which has performed both FPGA implementation and standard cell implementation. Estimated area from estimated gate counts is about 2 mm^2 for the piquant configuration.

4. IMPLEMENTATION RESULTS

Upon the development of assembly programs for the application, we have not prepared any fixed-point model for finite word-length effect evaluation. This development procedure without a time-consuming preparation of fixed-point model contributes to a great reduction in the total development time of the application program.

	FPGA (actual) with different "auto packed register" settings			# of nand2 gate (estimate)
	Normal	Off		
DSP module	# of LE	# of comb-LE	# of reg-LE	
DAGU	439	299	261	3,970
PFCU	593	394	329	5,083
BFPU	343	332	57	2,081
A/N-SFT	643	728		3,312
L/S-BUF	136		136	1,360
IREG	827	729	355	6,867
RREG	430	352	286	4,462
MAC	749	749	5	3,458
ALU	236	220	36	1,361
Total	4,396	3,803	1,465	31,954
		5,265		

Table 1: Hardware complexity of FitDSP-Piquant configuration on Cyclone EP1C20.
In this table, comb-LE and reg-LE represent a combinational LE (LE in which only the LUT is utilized), and a register LE (LE in which only the register is utilized), respectively.

All computations are realized successfully in H-BFP. Several accuracy-sensitive computation modules are also implemented in software-enabled floating-point. With software-enabled floating-point, no block exponent is defined. However, element exponents are used to align the binary-point locations of two data elements. Division and logarithm computations found in the noise canceller algorithm are classified as accuracy-sensitive modules.

4.1. Noise Suppressor

During the implementation of the noise canceller algorithm, such functionality as division, logarithm, and conditional execution are identified as missing functionality in the stage 2 of the design flow that is explained in the section 2.2. All of these functionality have been put into an ALU extension together with a new dedicated instruction field in very long instruction word (VLIW) type instruction set architecture (ISA).

4.2. Echo Canceller

During the implementation of the echo canceller algorithm, no requirements for special functionality extension has been found. Thus, the implementation effort has been spent in the resizing of DSP resources, and especially in the design of application-specific ISA. A good example can be seen in the different approaches to the computation of signal power.

In echo canceller, estimated echo signals are generated with an adaptive finite impulse response (FIR) filter. With an adaptive filter, the coefficients of the filter is dependent on the previous

filter outputs. Hence, how to keep the accuracy of the coefficients is very important. Computation of the signal power of the far-end input signal is the first step of producing a set of filter coefficients. Two possible approaches to the computation of the signal power of N signal samples are:

• (Approach 1) Straightforward accumulation

$$\text{power}(i) = \sum_{k=0}^{N-1} s^2(i-k) \quad (1)$$

• (Approach 2) Update by subtract-oldest and add-latest

$$\text{power}(i) = \text{power}(i-1) + s^2(i) - s^2(i-N) \quad (2)$$

With these two different approaches to the computation of signal power, the entire echo canceller has been implemented and the accuracies of estimated echo signals have been evaluated. In order to compare the performance of different approaches fairly, a segmental signal-to-noise ratio (SSNR) is used as an objective measure. The SSNR measure is defined as an average of sample-wise signal-to-noise ratio (SNR) that is defined as

$$\text{SNR} = 10\log\frac{\text{reference}(n)^2}{\{\text{reference}(n) - \text{target}(n)\}^2} \quad (3)$$

where an output from a reference C model is taken as a reference(n), while a target(n) is an output from several different target implementations. The reference C model is written in double-precision floating-point ("double" data-type in C). In addition to three implementations on FitDSP-piquant, the target implementations include two single-precision floating-point implementations ("float" data-type in C), and a fixed-point implementation which uses a 32-bit double word format to represent estimated echo signal. An evaluation result is presented in Fig. 4, which presents the SSNR scores of estimated echo signals as well as corresponding actual waveforms of signal power.

Two H-BFP implementations with the approach 1 and 2 have been first evaluated. In Fig. 4, corresponding curves are labelled as "Accumulation Assembly" and "Add/Sub Assembly", respectively. A situation of error accumulation in signal power computation with approach 2 can be seen in the top graph. It is also understood that this situation actually degrades overall SSNR score of echo signal. They both uses 15 bits for a mantissa. The required number of computation to compute signal power with approach 2 is 2 MIPS less than with approach 1. Thus, the total required MIPS count is estimated to be either 13 or 15 MIPS for the entire echo canceller of 10 MOPS complexity.

Above situation is also visible even with expensive 32-bit single-precision floating-point implementations, which are presented as "Accumulation C++ Float" and "Add/Sub C++ Float". But there are 2 MIPS less when the addition/subtraction approach is used.

5. CONCLUSIONS

This paper presented a hands free telephone system implementation on a custom-configured, block-floating-point DSP. This implementation consumes 13 or 15 MIPS of DSP performance for an echo canceller task of 10 MOPS complexity, depending of the accuracy. The required MIPS for a SS-based noise canceller is only 2 MIPS. The implementation was carried out on an FPGA platform. An expected core size of the DSP is 2 mm^2 with a 0.35 um standard cell process. A software / hardware codesign design flow was also presented with some actual examples from the implementation. An objective evaluation of signal processing quality has shown that exploited hierarchical block-floating-point can achieve much higher accuracy than fixed-point.

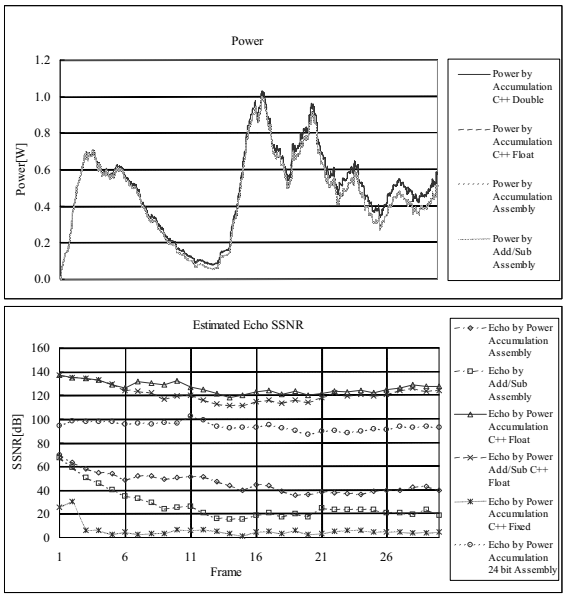

Figure 4: Segmental signal-to-noise ratios of estimated echo signals with different implementations (Bottom graph) with waveforms of corresponding power signals (Top graph). Curves are up to 30 frames (128 samples per frame) from the beginning. Implementations include three FitDSP-piquant implementations with different power computation approaches and different mantissa word-length, two short-precision floating-point implementations with different power computation approaches, and 16-bit fixed-point (in part 32-bit) implementation.

REFERENCES

[1] M. Liem and O. Manck, "Architecture of a Single Chip Acoustic Echo and Noise Canceller using Cross Spectral Estimation," in the processing of ICASSP 2003, II-637-640, 2003.

[2] S. Kobayashi and G. Fettweis, "A Hierarchical Block-Floating-Point Arithmetic," Journal of VLSI Signal Processing, 24(1):19–30, 2000.

[3] S. Kobayashi, "Developing a DSP Core for Embedded Devices which Ease Developing a Fixed Point Program - Capable of Developing an MP3 Decoder in 15 persons/day" (Japanese), Design Wave Magazine, September 2003.

[4] http://www.asahi-kasei.co.jp/vorero/en/nc/index.html