A Compact DSP Core with Static Floating-Point Unit & Its Microcode Generation*

Tay-Jyi Lin, Hung-Yueh Lin, Chie-Min Chao, Chih-Wei Liu, and Chein-Wei Jen
Department of Electronics Engineering
National Chiao Tung University, Taiwan
E-Mail: {tjlin, hylin, cmchao, cwliu, cwjen}@twins.ee.nctu.edu.tw

ABSTRACT

The multimedia SoC usually integrates programmable digital signal processors (DSP) to accelerate data-intensive computations. But the DSP and the host processor (e.g. ARM) are both designed for standalone uses, and they must have overlapped functionalities and thus some redundant components. In this paper, we propose a compact DSP core for dual-core multimedia SoC and its complete software development tools. The DSP core contains a dataflow engine that is composed of off-the-shelf memory modules with limited ports, and we have investigated software techniques extensively to reduce the hardware complexity as the principles of VLIW processors. Moreover, the DSP is equipped with novel static floating-point units to emulate expensive floating-point DSP operations at low cost. In our experiments, this core has about thrice the performance (estimated in execution cycles) of Analog Devices ADSP-218x with similar computing resources. Our first prototype in the 0.35μm CMOS technology operates at 100MHz and consumes 122mW power. The core size is 2.8mm^2 including an embedded DMA controller and the AMBA AHB interface.

Categories and Subject Descriptors

B.2 Arithmetic and Logic Structures, C.1 Processor Architectures, C.5 Computer System Implementation

General Terms

Design

Keywords

Digital signal processor, DSP core, Floating-point units

1. INTRODUCTION

A single general-purpose microprocessor (μP) cannot satisfy the computation requirements of today's multimedia/communication systems at acceptable cost or power consumption [1]. The most popular solution is to accompany the μP core with a digital signal processor (DSP) [2][3]. The DSP core executes the data-intensive tasks efficiently with its dataflow engine in the dual-core (or dual-processor) multimedia systems, while the μP handles the control-oriented and interactive tasks by maintaining a huge finite state machine. But both the two processors are designed originally for standalone uses and they must have overlapped functionalities and thus redundant components. Recently, the μP cores have been enhanced for digital signal processing by incorporating some single-cycle multiply-accumulators (MAC), SIMD (MMX-like) datapaths, or some other specific functional units [4]. But their performance is still far behind that of a DSP core with similar computing resources [5], because the DSP tasks are distinct from general-purpose computations. By the way, it is very difficult to optimize the memory subsystem for the two different types of tasks simultaneously in the single-core multimedia SoC [6].

In this paper, we have focused on the compaction of the DSP core, and leave the μP unchanged for software compatibility. DSP-lite is our first implementation with a pure dataflow engine, which is composed of off-the-shelf memory modules with limited access ports. Moreover, we have investigated software techniques to reduce the hardware complexity as the principles of VLIW processors [7]. To shrink the DSP core further, we also propose the static floating-point arithmetic to emulate expensive floating-point (FP) operations, where data are recorded as normalized fractional, similar to the mantissa part of the FP representations. The normalization factors are kept in our analysis software only, in contrast to the exponent part attached to each FP number. The DSP-lite core has similar computing resources to the 16-bit fixed-point Analog Devices ADSP-218x DSP [8], including an adder, a fractional multiplier, and a barrel shifter. Besides, it contains a DMA controller and the standard AMBA AHB interface to reduce the integration efforts. In our experiments, DSP-lite has about thrice the performance of ADSP-218x in the execution cycles, and the 16-bit static FP arithmetic has 38.1165dB signal to round-off noise ratio over IEEE single-precision FP units. The silicon implementation in the 0.35μm 1P4M CMOS technology achieves 100MHz clock rate with 122mW average power dissipation.

The rest of this paper is organized as follows. Section 2 describes the proposed static FP arithmetic. Section 3 and 4 elaborate the DSP-lite core and its software tool respectively. The simulation results and our silicon implementation are available in Section 5. Finally, Section 6 concludes this work and outlines our future research.

2. STATIC FLOATING-POINT ARITHMETIC

Digital signal processing demands high precision for quality and enough dynamic ranges to prevent overflow. Floating-point (FP) arithmetic [9] provides the full precision of ***mantissa*** and a huge dynamic range with ***exponent***, and is very suitable for developing and simulating algorithms. However, the cost of FP arithmetic is prohibitively high in terms of power consumption, speed, and silicon area. Besides, the signal ranges in most well-designed algorithms are modest and do not vary much. Therefore, most embedded DSP systems use integer arithmetic instead and rely on

* This work was supported by the National Science Council, Taiwan under Grant NSC92-2220-E-009-027

GLSVLSI'04, April 26-28, 2004, Boston, Massachusetts, USA

the designers to manually scale the variables to prevent overflow. But the conversion of FP into integer arithmetic is ad-hoc, which requires extensive and time-consuming simulations and usually results in sub-optimal designs. In this paper, we propose the static FP (SFP) arithmetic to make a good compromise between the FP and integer arithmetic, where the exponent is tracked statically and kept in the analysis software only. Fig. 1(a) shows the base-line SFP configuration linear operations. Compared to the FP units, the pre-scalers and normalizers of adders and multipliers are shrunk to 1-bit shifters only. Besides, a barrel shifter with sign extension is integrated to perform scaling or normalization over multiple (>1) bits. The hardware cost of SFP units is similar to that of integer units. But SFP performs fractional multiplications just as FP arithmetic, where the insignificant product bits are rounded off autonomously [10], while integer arithmetic needs to explicitly round the double-wordlength products. Fig. 1(b) shows the example of 8-bit fractional multiplication.

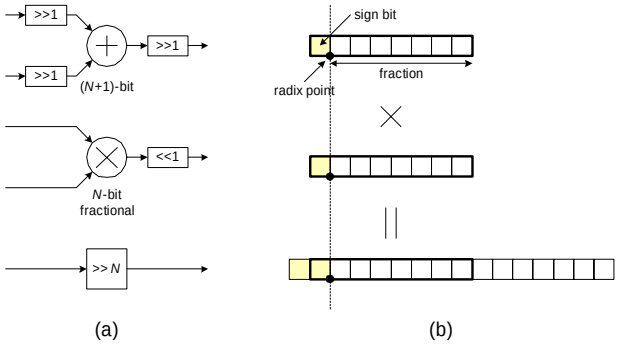

Fig.1 (a) SFPU for linear transforms, and (b) fractional multiply

Assume the DSP kernels have been described in the synchronous dataflow graphs (SDFG) [11][12], and we associate each vertex (including inputs, FP operations, and outputs) a peak estimation vector (PEV) [***M r***] to simulate the FP computations. ***M*** denotes the maximum magnitude that may occur on the output of the vertex and ***r*** is used to track the radix point (i.e. the exponent). ***M*** should be kept between 0.5 and 1 for the fractional arithmetic (with data range between -1 and 1) to maximize the precision while preventing overflow. Assume the inputs are normalized as fractional numbers (i.e. PEV = [1 0]), and the PEV of the remnant variables can be calculated by three rules:

- Keep ***M*** between 0.5 and 1 by carrying out "***M*** divided (multiplied) by 2" and "***r*** minus (plus) 1" simultaneously
- ***r*** (radix point) should be identical before summation or subtraction
- $[\boldsymbol{M_1}\ \boldsymbol{r_1}]\times[\boldsymbol{M_2}\ \boldsymbol{r_2}]=[\boldsymbol{M_1}\times\boldsymbol{M_2}\ \boldsymbol{r_1}+\boldsymbol{r_2}]$

Fig. 2(a) shows two PEV calculation examples. After the PEV analysis, shifts are inserted to normalize the intermediate results. Note that the scaling and normalization of the two examples can be carried out in the embedded 1-bit shifters without invoking the barrel shifter. This simple PEV analysis over-estimates the data ranges because it neglects the correlations among variables, such as the example in Fig. 2(b). Our analysis software improves the range estimation by recording the intermediate variables in the affine form [13]:

$$\sum \alpha_i \cdot x_i$$

where α_i denotes the contribution of each independent variable x_i (i.e. an input node or the output of a non-linear operation). The magnitude ***M*** of a vertex is calculated with α_i instead of its input ***M*** directly. Fig. 2(c) illustrates the PEV analysis based on the affine arithmetic. For non-linear operations such as multiplication of two variables, our analysis software creates new variables to simplify the range estimation.

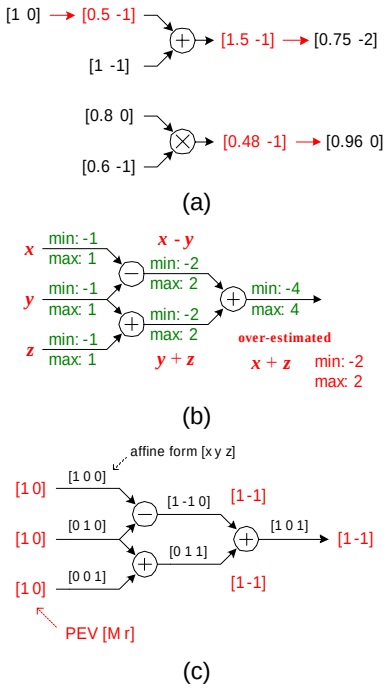

Fig. 2 Examples: (a) PEV analysis; (b) range over-estimation; and (c) affine PEV analysis

3. DSP-LITE CORE

DSP-lite is the first implementation of our proposed compact DSP core, which is composed of off-the-shelf memory modules at the inputs of each functional unit. The memory conflicts are resolved in software via optimal ILP-based operation scheduling described in Section 4. DSP-lite is equipped with the 16-bit baseline static floating-point units, including (a) a 17-bit adder/subtractor with two input scalers and an output normalizer, (b) a 16-bit fractional multiplier with an output normalizer, and (c) a 16-bit barrel shifter with sign-extension. Besides, it contains a DMA controller with ping-pong buffering for efficient data exchanges with the μP core and external I/O devices. The standard AMBA AHB interface is also integrated to reduce the integration efforts. DSP-lite has a micro-instruction memory, which keeps the control data, such as the enable signals for the 1-bit aligners and normalizers, and the addresses needed to access the memory modules. The loading of the microcode during the core initialization is also carried out by the DMA controller. The memory modules all support address remapping to simplify control, which automatically translates the virtual addresses in the microinstructions into different physical addresses for different iterations. An address remapper contains an iteration counter and a stride register, and the virtual addresses are decremented once with the stride number for each iteration. Fig. 3 shows an illustrating example, where the virtual address "3" is mapped to "3", "2", and "1" physical addresses in the first,

the second, and the third iteration respectively, if the address remapper is enabled with the stride number set "1". The decremented addresses are modulo of the size of each physical memory, and thus the physical addresses are rotating. Fig. 3 shows a data dependency across two iterations. The source writes an item to the virtual address "3" and the destination can retrieve the same item from the virtual address "5" after two iterations continuously. Note that additional memory locations should be reserved to prevent overwriting live variables (e.g. the virtual address "4" in this example).

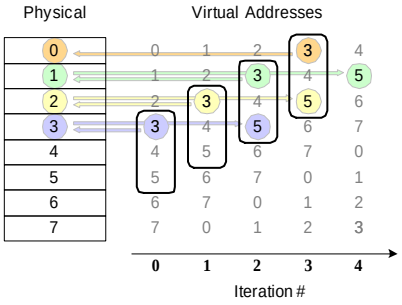

Fig. 3 Examples: memory remapping

Fig. 4 illustrates DSP-lite performs 2-D discrete cosine transform (DCT) [14]. While the dataflow DSP engine performs DCT on the second 8-by-8 image block, the embedded DMA controller is busy storing back the DCT coefficients for the first image block to the main memory and transferring the third image block to the I/O buffer for the next iteration DCT operations.

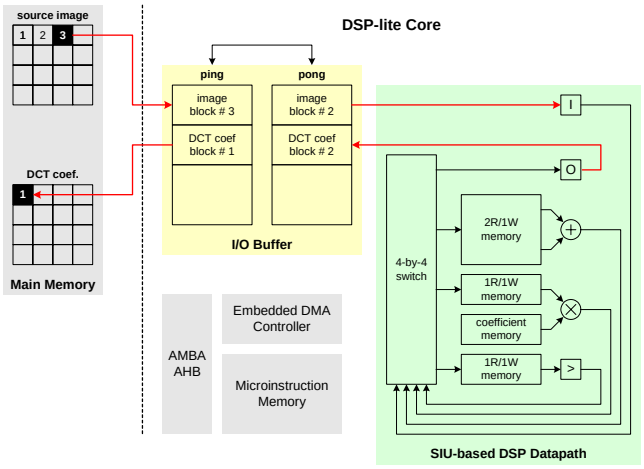

Fig. 4 DSP-lite core and 2-D DCT example

4. MICROCODE GENERATION

We have developed an automation tool to compile the algorithmic descriptions in the synchronous dataflow graphs (SDFG) with floating-point (FP) operations into the DSP-lite microcode with static FP (SFP) arithmetic. The designers first design and verify their DSP algorithms with our SDFG simulator. Note that the SDFG can also be derived from the C/C++ descriptions via the SUIF compiler [15]. After the simulation of the DSP algorithm, the FP operations are emulated with the SFP units by applying the PEV analysis in Section 2 with affine arithmetic. Additional shift operations are inserted in the SDFG for operand alignment and normalization. Then, the operations in the modified SDFG are scheduled with integer linear programming (ILP). ILP is a formal and comprehensive approach to describe and solve the scheduling problems. Here, we use periodic scheduling for simplicity, where only the intra-iteration data dependency is considered and the weighted edges (i.e. inter-iteration dependency) are first removed from SDFG. The scheduling ranges for each operation are first determined using the as-soon-as-possible (ASAP) and as-late-as-possible (ALAP) algorithms [16]. The following illustrates the construction of the ILP model using the example shown in Fig 5.

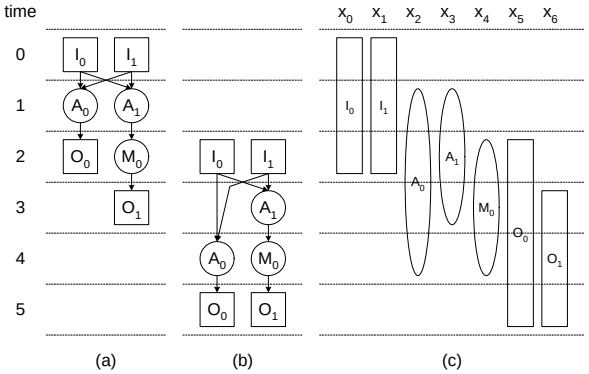

Fig. 5 (a) ASAP; (b) ALAP; (c) scheduling ranges

A Boolean variable $x_{i,j}$ indicates whether a vertex i is scheduled into the time step j, and the following three constraints must first be satisfied [12][16].

Resource constraints (# operations cannot exceed # resources)

$x_{0,0} + x_{1,0} \le 1$; $x_{0,1} + x_{1,1} \le 1$; $x_{0,2} + x_{1,2} \le 1$ (for input) …

Allocation constraints (each operation executes once)

$x_{0,0} + x_{0,1} + x_{0,2} = 1$ …

Dependency constraints

$x_{0,0} + 2\,x_{0,1} + 3\,x_{0,2} - 2\,x_{2,1} - 3\,x_{2,2} - 4\,x_{2,3} - 5\,x_{2,4} \le -1$ …

When multiple functional units simultaneously write their results into the same memory module, the memory accesses will conflict. In our tool, the operations with an identical destination memory module will be scheduled into distinct time slots by incorporating the following port constraints to prevent these conflicts.

Port constraints

$x_{0,0} + x_{1,0} \le 1$; $x_{0,1} + x_{1,1} \le 1$; $x_{0,2} + x_{1,2} \le 1$; (for adder)

$x_{2,2} + x_{4,2} \le 1$; $x_{2,3} + x_{4,3} \le 1$; $x_{2,4} + x_{4,4} \le 1$; (for output)

The objective of the ILP-based scheduler is to minimize execution cycles, and we solve the ILP model using a commercial ILP solver [17]. Lifetime analysis of the variables is performed after scheduling to allocate memory locations with address remapping. Once the memory happens to be exhausted, additional load/store operations should be inserted to spill variables with long lifetimes. At last, microinstructions are synthesized based on the memory addresses and the control signals for the DSP algorithm.

5. EXPERIMENTAL RESULTS

We have several DCT implementations to evaluate the effectiveness of our static floating-point (SFP) arithmetic in Table 1. The first three rows summarize the results of the floating-point (FP), 16-bit and 32-bit integer (i.e. `jfdctflt.c`, `jfdctfst.c`, and

`jfdctint.c`) C codes for 2-D DCT from the independent JPEG group (IJG) [18]. The last two rows are for the 16-bit and 24-bit SFP respectively, both of which are derived directly from the FP `jfdctflt.c`. The second column compares the round-off error using as metric the PSNR over single-precision FP arithmetic, and the results are obtained from simulations on natural images. The 16-bit SFP even outperforms the hand-optimized 32-bit integer code, while the 24-bit SFP 62.1439dB PSNR, which has the same maximum precision as single-precision FP (with 23-bit mantissa). The third column summarizes the performance in cycles, and the four 1-bit shifters for alignment or normalization significantly reduce the cycles from 1,120 to 720. By the way, the functional units in the comparison are all single-cycle with registered I/O (i.e. with 2-cycle latency). This implies that the FP units have much longer cycle time than the integer and SFP units.

Table 1 Comparison of arithmetic units for 2D-DCT

	PSNR (dB)	Cycle count
Single-precision FP unit	-	672
16-bit integer unit	33.2220	848
32-bit integer unit	36.0981	672
16-bit SFP unit	**38.1165**	**720**
24-bit SFP unit	**62.1439**	

Table 2 depicts the performance evaluation with some popular DSP kernels on DSP-lite and Analog Devices ADSP-218x, which has similar computing resources (i.e. a set of ALU, MAC, and barrel shifter). The results for ADSP-218x are excerpted from its application notes [8]. DSP-lite has better performance in most cases because ADSP-218x is constrained by the conventional programming model and limited parallel instructions. DSP-lite needs more cycles for the biquad filter case because its functional units and memory have 2-cycle and 1-cycle latency respectively, instead of zero in ADSP-218x. Note that DSP-lite is still faster in the absolute time. By the way, some transforms (e.g. lookahead [10]) can reduce the iteration bound and thus effectively improve the efficiency of DSP-lite to perform the iterative kernels with feedback loops.

Table 2 Performance evaluation of DSP-lite core (# cycle)

	ADSP-218x (80MHz)	**DSP-lite (100MHz)**
3rd-order lattice filter	32	**13**
2nd-order biquad filter	13	**14**
16-point complex FFT	874	**268**
8-point 1-D DCT	154	**47**
8×8 2-D DCT	2,452	**720**

We have synthesized the DSP-lite core using Synopsys with the 0.35μm cell library and have placed and routed the netlists using Apollo for the 1P4M CMOS technology. The core size is 2.8mm^2 including an embedded DMA controller and the standard AMBA AHB interface. The chip can operate at 100MHz and consumes 122mW average power.

6. CONCLUSION

This paper presents a compact DSP core for dual-core multimedia SoC. Software techniques are extensively investigated to reduce the hardware complexity for the dataflow engine as the principles of VLIW processors. The DSP prevents unnecessary constraints posed by conventional programming models to achieve dataflow-rate computations. The paper also describes our proposed static floating-point arithmetic to emulate expensive floating-point DSP operations, which also helps to further shrink the DSP core. The paper also illustrates a software development tool to automatically generate the microcode from high-level floating-point algorithmic descriptions. In our experiments, the DSP core has about thrice the performance of the Analog Devices ADSP-218x DSP family with similar computing resources. Moreover, the proposed static floating-point arithmetic has 62.1439dB PSNR over the IEEE 754 single-precision floating-point arithmetic with the same maximum precision (i.e. 24 bits versus the 23-bit mantissa).

We are now developing a list-based scheduler to avoid memory conflicts with significantly reduced complexity than the ILP-based one in this paper. We are also improving the round-off error of the static floating-point arithmetic by incorporating the saturated arithmetic [19]. Moreover, we are going to investigate the distributed microinstruction memory to reduce the global routing with the JTAG-like configuration interfaces [20], and study some coding or compression techniques to reduce the configuration bandwidth and the context-switch overheads in the near future.

7. REFERENCES

[1] A. Gatherer, et al, "DSP-based architectures for mobile communications: past, present and future," *IEEE Communications*, Jan. 2000

[2] *Intel PXA800F Cellular Processor – Development Manual*, Intel Corp., Feb. 2003

[3] *OMAP5910 Dual Core Processor – Technical Reference Manual*, Texas Instruments, Jan. 2003

[4] M. Levy, "ARM picks up performance," *Microprocessor Report*, 4/7/03-01

[5] R. A. Quinnell, "Logical combination? Convergence products need both RISC and DSP processors, but merging them may not be the answer," *EDN*, 1/23/2003

[6] *TriCore 2-32-bit Unified Processor Core v.2.0 Architecture – Architecture Manual*, Infineon Technology, June 2003

[7] J. L. Hennessy and D. A. Patterson, *Computer Architecture – A Quantitative Approach*, 3rd Edition, Morgan Kaufmann, 2002

[8] *Digital Signal Processing – Using the ADSP-2100 Family*, Analog Device Inc., 1990

[9] *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE Standard 754, 1985

[10] P. Lapsley, J. Bier, and E. A. Lee, *DSP Processor Fundamentals – Architectures and Features*, IEEE Press, 1996

[11] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, Jan. 1987

[12] K. K. Parhi, *VLSI Digital Signal Processing Systems – Design and Implementation*, John Wiley & Sons, 1999

[13] F. Fang, R. Rutenbar, M. Puschel, and T. Chen, "Toward efficient static analysis of finite-precision effects in DSP applications via affine arithmetic modeling," in *Proc. DAC*, 2003

[14] W. B. Pennebaker, and J. L. Mitchell, *JPEG – Still Image Data Compression Standard*, Van Nostrand Reinhold, 1993

[15] *The SUIF Compiler Infrastructure*, http://suif.stanford.edu/

[16] D. D. Gajski, et al, *High Level Synthesis – Introduction to Chip and System Design*, Kluwer Academic Publisher, 1992

[17] *LINDO API User's Manual*, LINDO System Inc., 2002

[18] *Independent JPEG Group*, http://www.ijg.org

[19] G. A. Constantinides, P. Y. K. Cheung, W. Luk, "Synthesis of saturation arithmetic architectures ," *ACM Trans. Design Automation of Electronic Systems*, July 2003

[20] *IEEE Standard for In-System Configuration of Programmable Devices*, IEEE Standard 1532, 2002