A Versatile Computation Module for Adaptable Multimedia Processors

Yunan Xiang, Ryan Pettibone, Martin Margala
Department of Electrical and Computer Engineering, University of Rochester
Rochester, NY 14627-0231, USA
Email: {xiang,pettibon,margala@ece.rochester.edu}

***Abstract*—This paper describes a low cost, low power, versatile computation module that can be used as a coarse-grain building block in multimedia processors. The module, which has a datapath and a controller integrated with its local data memory, performs various arithmetic operations on different data types, i.e., 8-bit integer, 16-bit integer, 32-bit integer and single precision floating point numbers. Running in parallel, the module provides high data throughput at low hardware cost. Multiple modules will be connected in a multimedia processor operated in mixed SIMD and MIMD modes, providing great flexibility for data parallel, computation intensive multimedia applications.**

I. INTRODUCTION

With the technology advances in integrated circuits (IC) fabrication and the increasing communication channel bit rates, multimedia processors are playing a more significant role in main computer systems as well as in personal mobile devices [1]. Multimedia applications usually involves large amount of data, require high data rate, real-time processing. Although the amount of data and computation throughput varies over a wide range depending on the required quality of the applications, multimedia data processing has the following characteristics: The word lengths of the processing data are 8 bits, 16 bits or 24 bits, which requires frequent use of small integer operations; Arithmetic operations are highly computation-intensive and repetitive with data parallelism; Intensive memory access to a large memory space requires high bandwidth memory interface [2].

In this paper, a low power, low hardware cost computation module is proposed to provide the desirable features for constructing modular array based multimedia processors. The module has a datapath, a controller and a local 16 KB SRAM data cache. The datapath comprises four 8-bit Processing elements (PE). Therefore, the module can operate on four 8-bit or two 16-bit integer numbers in a Single Instruction Multiple Data (SIMD) mode or perform one 32-bit integer arithmetic operation. It also supports addition, subtraction and multiplication of IEEE 754 standard floating point numbers.

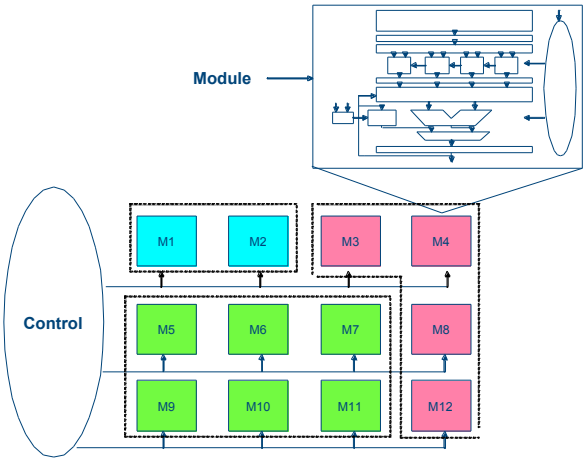

Figure 1. An adaptive multimedia processor using the module

Multimedia signal processing can take advantage of the flexible data types and multiple arithmetic operations available from the module. Local data memory provides the required memory access data bandwidth. As shown in Fig. 1, an adaptive multimedia processor can be constructed using the proposed module. Depending on the applications, an array of modules is divided into several clusters, as indicated by the dash-line frames. Each cluster may have different numbers of modules operated in SIMD fashion for one application, while other clusters performing different applications at the same time in a multiple instruction multiple data (MIMD) mode. For example, one cluster may perform a two-dimensional discrete cosine transform (2-D DCT) while another cluster is performing a discrete wavelet transform (DWT). Instructions for the different modules are provided from the upper level controller using encoded address for the modules.

The paper is organized as the followings: Section II describes the architecture of the proposed module. Section III discusses the supported arithmetic operation modes. Implementation and simulation results are presented in section IV. Section V draws a conclusion.

II. MODULE ARCHITECTURE

The architecture of the proposed module is shown in Fig. 2. The module has a datapath, a controller and a 16 KB SRAM local data memory. When an instruction is executed in the module, the addresses of the input data are provided to the local memory and the 64-bit data are retrieved from the SRAM to the input register. The controller takes the 5-bit operation code (opcode) from the instruction and decodes it into various control signals at different clock cycles to the datapath so that the datapath can perform the specified operations. The Outputs of the operations are available in the output registers and can be either written back to the local data memory, send to other modules through processor level data bus network, or direct to the output of the multimedia processor.

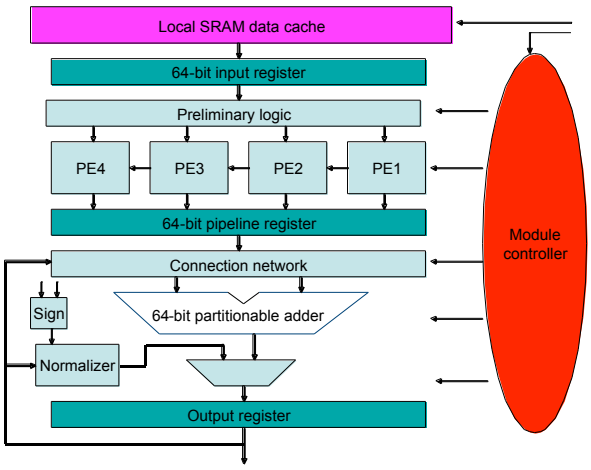

Figure 2. The architecture of the module

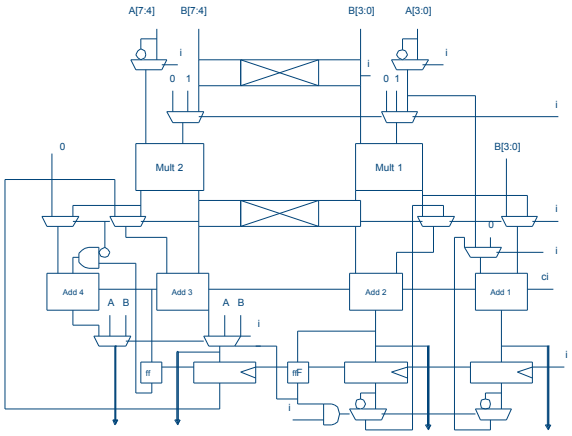

Figure 3. The Processing Element block diagram

A. The Datapath

The datapath of the module consist of two stages separated by the pipeline register, as shown in Fig. 2. The first stage includes a 64-bit input register, the preliminary logic and four processing elements. The 64-bit input register is directly connected to the 64-bit-wide data bus of the local SRAM. For every operation instruction, two 32-bit input data are read into the input register at the same clock cycle. The 32-bit input data could be four 8-bit integer operands, or two 16-bit integer operands, or one 32-bit integer operand, or one IEEE 754 single precision floating point number. Depending on the data types and operations specified in the instruction, the preliminary logic decomposes a 32-bit number into four sets of 8-bit numbers and feeds them into different PEs.

The 8-bit processing elements are the workhorse of the entire datapath. It is designed to be efficiently shared by all of the operation modes with minimum hardware to lower the area cost and power consumption. Fig. 3 shows the block diagram of the PE. The main computation units in it are two 4x4 multipliers and four 4-bit ripple carry adders which can be carry linked. Each PE accepts two 8 bit operands, and outputs a 16 bit vector. It performs the basic operation of partial product generation, as well as addition, subtraction, and absolute value at the 8-bit level. These operations are the building blocks of data manipulations at larger data levels, i.e., 16-bit and 32-bit levels. The PEs can also be carry linked to perform 16 and 32-bit addition and subtraction. Multiplications are handled by decomposing larger data level multiplications into smaller data level multiplications and adding the partial products [3]. For example, an 8-bit multiplication can be computed as in (1):

$$A[7:0]*B[7:0] = (A[7:4]*B[7:4]) \,\&\, (A[3:0]*B[3:0]) + (A[7:4]*B[3:0] + A[3:0]*B[7:4])\ ls4 \quad (1)$$

Note that "&" denotes "concatenation" and "*ls4*" denotes "left shift 4 bit".

The second stage consists of a connect network, a 64 bit partionable adder comprised by two 32-bit Kogge-Stone adders, a normalyzer, and a block that determines the sign of the result. As the preliminary logic in the first stage, the connect network arranges partial products before they entry into the adder, which has the role of accumulating partial products. The output register provides feedback into the connect network as well as the normalyzer, which normalize the format of floating point numbers. A mux before the output register allows a choice between the normalized and un- normalized results, e.g., results for 32-bit integer multiplication and for floating point operations.

B. The Controller and the SRAM

The controller decoded the 5-bit opcode into control signals for the datapath to execute one of the eighteen supported operations. Table 1 summarizes all of the supported arithmetic operations and the thoughput and latency in clock cycles for executing each of these operations, where parallelism means that how many sets of data inputs can be executed in SIMD fashion. As we will see in the next section, different operations may just use one of the two stages, or repeatedly use one stage before using the other stage, or use both of the two stages concurrently. Therefore standard pipeling approach does not apply here and the clock cycles taken by different operations also vary widely. To maximize the efficiency of using the two stages,

the controller has two state variables to keep track of the usage of two stages, allowing two different instructions executing in two stages simultaneously. A next_op signal is issued by the controller when the current operation no longer needs the first stage and the datapath is able to accept a new operation, i.e., pipeling the operations.

The SRAM local data memory will be either a hard macro created from a third party SRAM compiler or a soft core created from Synopsys. We choose the capacity of 16 KB to meet the storage of a RGB colored 64x64 picture. A larger memory size may be used to buffer more pixels but the access time penalty of the data should be considered.

TABLE I. SUPPORTED ARITHMETIC OPERATIONS

Instruciton	Parallelism	Thoughput/ Latency
add8/sub8	4	1/2
mult8	4	2/3
abs_add8/abs_sub8	4	2/3
add16/sub16	2	1/2
mult16	2	4/5
abs_add16/abs_sub16	2	2/3
add32/sub32	1	2/3
mult32	1	8/9
abs_ad32/abs_sub32	1	4/5
fp_add	1	6/7
fp_sub	1	6/7
fp_mult	1	9/10

III. ARITHMETIC OPERATIONS

The arithmetic operations executed in the datapath can be categorized into three basic types: the integer addition/subtractions and absolute values of the integer addition/subtractions, the integer multiplications, and the floating point operations. We discuss the execution of these three categories in this section.

A. Integer Addition/Subtraction and Absolute Values

8-bit integer addition and subtraction are executed by each PE using the two left most 4-bit adders, i.e., adder1 and adder2, which are carry-linked together. As illustrated in Fig. 3, A[7:4] and B[7:4] have to multiply 0001 through the two multipliers and then are added up in adder2, while A[3:0] and B[3:0] are added up in adder1. The carry-in signal for adder 1 is set to 0. When a subtraction is performed, B[7:0] are simply inverted and the carry in signal for adder1 is set to one to implemented a 2's complement negative integer representation. Note that only the two left most registers keep the valid output data.

When computing the absolute value of the 8-bit addition/subtraction, an extra step (clock cycle) is needed. After getting the results of the addition/subtraction, the controller checks the sign bit of the results. Nothing is done for a positive number. If the result is a negative number, the data are inverted and add 1.

16-bit addition/subtraction and 32-bit addition/subtraction and their absolute values are computed in similar ways. Two PEs are carry-linked for performing 16-bit operations and four PEs are carray-linked for performing 32-bit operations. Note that a flip-flop was inserted between PE2 and PE3 to pipeline the carry out of PE2 for 32-bit addition/subtraction so that the clock cycle can be half of the propagation delay.

B. Integer Multiplications

As mentioned in section II, integer multiplications are handled in the way that larger data level multiplications are decomposed into small data level multiplications and then the partial products are added together to exploit the hardware reuse. Equation (1) illustrated how an 8-bit multiplication is calculated. Since there are only two multipliers in each PE, A[7:4] * B[3:0] and A[3:0] * B[7:4] are computed and added up in adder3 and adder2 in the first clock cycle and 8-bit result are stored in register2 and register3. In the second cycle, A[7:4] * B[7:4] and A[3:0] * B[3:0] are computed and the partial products are accumulated onto the previous results and produce a 16-bit results. Therefore, each PE needs two cycles to finish an 8-bit multiplication.

16 and 32-bit multiplications are performed in a similar way. Two PEs are grouped together to perform a 16-bit multiplication, with each PE calculating an 8-bit partical product. A 16-bit multiplication is executed as in (2). However, the connect network and the SIMD adder in the second stage are used to shift and align the partial products and add them together. Note that the module can perform two 16-bit multiplications in parallel and the 64-bit SIMD adder is split into two 32-bit adders to accumulate the two sets of partial product at the same time.

$$A[15:0]*B[15:0] = (A[15:8]*B[15:8]) \,\&\, (A[7:0]*B[7:0]) + (A[15:8]*B[7:0] + A[7:0]*B[15:8]) \mathit{ls8} \quad (2)$$

C. Floating Point operations

When executing single precision floating point operations, the preliminary logic in the first stage takes the sign bit out for later use and feeds the exponents of the two floating point numbers into PE4. It also extends the 23-bit significands into 24-bit numbers with the leading 1 before send them to PE1, PE2 and PE3. Steps for floating point operations follow the algorithms described in [4].

For floating point addition/subtraction, two exponents are compared to align the significands. One of the significands with different signs is converted into 2's complement number and the two significands may swap positions before adding together. Finally the result is formatted back into a 32-bit floating point number by the Normalizer in the second stage, which decides the right sign for the result, shifts the exponent if necessary. If the results is a negative number, it

has to be converted back to sign magnitude format from 2's complement format in the first stage.

The floating point multiplication also uses PE4 to add the exponents and minus the bias number 127. Multiplication of the significands uses the 24-bit integer multiplications in PE1 to PE3 in the same way as the 16 and 32-bit integer multiplication. The result of multiplication is simply truncated with taking rounding mechanisms into consideration to minimize the design complexity.

IV. Implementation Results

The datapath and the controller of the proposed module were coded in VHDL hardware description language. A simplified testbench with all of the extreme cases as well as some random test cases was created to verify the functions of the module. All of the eighteen operation modes are verified using the Cadence NCsim simulator. Table II gives an example of two set of test vectors for abs_add8 and mult16. Note that one test vector for abs_add8 includes four test cases, while one test vector in mult16 includes two test cases.

The VHDL codes were synthesized by Synopsys Design Compiler using the ACI 1.8V CMOS standard cells targeted for TSMC 0.18 _m technology. A maximum clock frequency of 222 MHz can be achieved with an average energy consumption of 0.065 mW/MHz and a silicon area of 132,041 $_m^2$.

A comparison to the datapaths in three previous papers has been illustrated in Table III.

V. Conclusions

A versatile computation module that exploiting hardware reuse has presented in the paper. Eighteen various integer and floating point arithmetic operations are supported. Implemented using standard cells in TSMC 0.18 _m technology, the module consumes 0.065wM/MHz power and has an area of 132,041 $_m^2$. With the great flexibility, it is very suitable as the building block for adaptable multimedia processors.

TABLE II. Testbench Examples

Operations	Input data (test vectors) (Hexadecimal)		Output data (Hexadecimal)
ads_add8	80818200	7F7FFF7F	01007F7F
	85CA009F	39ED8166	42497F05
mult16	00000000	0000FFFF	0000000000000000
	FFFF8000	00018000	0000FFFF40000000
	7FFFFFFF	7FFFFFFF	3FFF0001FFFE0001
	E9DF1489	72A19569	68B8633F0BFC2931

Acknowledgment

The authors would like to thank Marco Lanuzza for his help in clarifying the questions and Quentin Diduck for helping us writing the C code to create the input data for the testbench.

References

[1] K. Diefendorff and P. Dubey, "How multimedia workloads will change processor design," IEEE Computer, 80(9), September, 1997.

[2] J. Kuroda, I.; Nishitani, T., "Multimedia processors," Proceedings of the IEEE Volume 86, Issue 6, pp. 1203-1221, June.

[3] M. Margla, R. Lin, "Highly efficient digital CMOS accelerator for image and graphics processing," 15th Annual IEEE International ASIC/SOC Conference 2002, pp127-132, 25-28 September 2002.

[4] D goldberg, "Appendix A: Computer arithmetic," in "Computer architecture: a quantitative approach," by J. Hennessy and D. Patterson, Morgan Kauffman Publishers, 1996.

[5] M Lanuzza, P. Corsonello, M. Margala, "Cost-effective, low-power processor-in-memory-based reconfigurable datapath for multimedia applications," ISLPED'05, August 8-10, 2005, San Diego, California, USA.

[6] A. Farooqui, V.G. Oklobdzija, "A programmable data-path for MPEG-4 and naural hybrid video coding," 34th Annual Asilomar Conference on Signals, Systems and Computers, Pacific Grove, California, October 29 – November 1, 2000

[7] J.S. Moon, T.J. Kwon, J. Sondeen, J. Draper, " An area-efficient standard-cell floating-point unit design for a processing-in-memory system," Proc. of the Conference on Europen Solid-state Cirucits, 2003, ESSCIRC'03, pp.57-60, 16-18 September 2003.

TABLE III. Comparision of the Proposed Module with Other Works

Architecture	Technology	Gate count	Speed (MHz)	Area ($_m^2$)	Energy (mW/MHz)	SIMD integer operations	Floating point operations	Thoughput/ Latency for mult32	Thoughput/ Latency for fp_add/sub
Proposed module	0.18 _m CMOS	6,224	222	132,041	0.06	YES	YES	8/9	6/7
[5]	0.18 _m CMOS	3,856	285	107,323	0.12	YES	YES	8/9	5/7
[6]	0.25 _m CMOS	N.A.	200	N.A.	N.A.	YES	YES	1/2	1/2
[7]	0.18 _m CMOS	8,446	300	278,210	0.6	NO	YES	N.A.	1/5