High Performance Hardware Operators for Data Level Parallelism Exploration

Libo Huang, Zhiying Wang, Nong Xiao

***Abstract*—**Many microprocessor vendors have incorporated high performance operators in a single instruction multiple data (SIMD) fashion into their processors to meet the high performance demand of increasing multimedia workloads. This paper presents some recent works on hardware implementation of these operators for data-level parallelism (DLP) exploration. Two general architectural techniques for designing operators with SIMD support are first described including low precision based scheme and high precision based scheme. Then new designs for integer operators as well as floating-point operators are provided to accommodate best tradeoff between cost and performance. To verify the correctness and effectiveness of these methods, a multimedia coprocessor augmented with SIMD operators is designed. The implemented chip successfully demonstrates that the proposed operators get good tradeoff between cost and performance.

***Keywords*—**Operator, SIMD, high performance, data level parallelism

I. Introduction

Arithmetic unit design is a research area that has been of great importance in the development of processor [7]. They are key components in processors which consume most power and attribute most latency. As such, a great deal of research has been devoted to the study of the high efficiency arithmetic units [7, 19]. They have primarily focused on implementing the various basic arithmetic units at smaller areas, lower power, and higher speed. Indeed, novel circuit techniques, and innovation in algorithm and structure have resulted in rapid improvements in arithmetic unit performance. Examples are integer adders, multipliers and floating-point operators. As a standard operator, the basic arithmetic unit design has been already mature.

To further boost the performances of arithmetic units, its design methodology turns to global optimization for program execution instead of single optimal arithmetic unit. Adding Application-specific instruction-set processors (ASIPs) or Single instruction multiple data (SIMD) units to a general purpose processor receives enormous attentions. The ASIP method identifies critical operations and implements them by dedicated arithmetic units based on the performance characterization of the specific application [13]. While ASIPs are only suitable for a limited set of applications, SIMD unit can be beneficial for more general data-intensive applications such as multimedia computing [5, 22]. As multimedia applications contain a lot of inherent parallelism which can easily be exploited by using SIMD unit, these augmented function units can significantly accelerate the multimedia applications. To take advantage of this fact, several multimedia extensions were introduced into microprocessors architectures. Examples are Intel's MMX, SSE1, SSE2 and even SSE3, AMD's 3DNow, Sun's VIS, HP's MAX, MIPS's MDMX, and Motorola's AltiVec [22].

The main feature of these extensions is the exploration of the data level parallelism (DLP) available in multimedia applications by partitioning the processor's execution units into multiple lower precision segments called *subwords*, and performing operations on these subwords simultaneously in a single instruction multiple data (SIMD) fashion [22]. This is called *subword parallelism* (also referred to *microSIMD parallelism* or *packed parallelism*). For example, in MMX technology [3], the execution units can perform one 64-bit, two 32-bit, four 16-bit or eight 8-bit additions simultaneously.

The integer SIMD computation is very popular in the domains of multimedia processing, where data are either 8 or 16 bits wide: this allows for the usage of subword precision in existing 32-bit(or wider) arithmetic functional units. This way, a 32-bit ALU can be used to perform a given arithmetic operation on two 16-bit operands or four 8-bit operands, boosting performance without instantiating additional arithmetic units. However, the floating-point operators with SIMD features appears relatively later such as SSE instructions (in Pentium III) and SSE2 and SSE3 instructions (in Pentium IV). This is due to that the area and power consumption of floating-point operators are relatively larger than integer unit. Another reason is that until a few years ago the applications usually did not require extensive floating-point data parallel computations. Nowadays things change a lot, forcing the develops to address the continued need for SIMD floating-point performance in mainstream scientific and engineering numerical applications, visual processing, recognition, data-mining/synthesis, gaming, physics, cryptography and other areas of applications. Recently, Intel even announced an enhanced set of vector instructions with availability planned for 2010, called AVX which has 256-bit SIMD computation [12]. It enhances existing 128-bit floating-point arithmetic instructions with 256-bit capabilities for floating-point processing.

Manuscript received October 31, 2011: This work was supported in part by the National Natural Science Foundation of China under Grant No. 61103016, No. 60803041, and No.60773024.

Libo Huang is with the School of Computer, National University of Defense Technology, Changsha 410073, China (corresponding author phone: 86-731-84573640; e-mail: libohuang@nudt.edu.cn).

Zhiying Wang and Nong Xiao are with the School of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: {zywang,nongxiao}@nudt.edu.cn).

To effectively implement multimedia extension instructions, the data format to be used in SIMD operations and corresponding hardware support must be provided. The SIMD operators not only need to complete the basic arithmetic operations, but also need to deal with the computations, data conversion and data unpacking of different precision subwords in parallel. This makes the SIMD operator a good capability in data handling, but the hardware to implementing them becomes more complicated, and its power and delay is much greater. Therefore, to design SIMD operators with low power, small area and delay is becoming a new design goal.

To reduce processor area, it is often desirable to have SIMD operations share circuitry with non-SIMD operations. Thus, many optimized hardware structures supporting SIMD operations for fixed-point and floating-point units have been introduced and this paper attempts to survey these existing subword parallel algorithms in a system view. Then new methods for multiplier and floating-point Multiply-add fused (MAF) unit are proposed to achieve a better performance in power, cycle delay and silicon area. To test these methods, a multimedia co-processor with SIMD fixed-point and floating-point units integrating into the LEON-3 host processor is designed.

The main contribution of this paper is proposing and evaluating methods and architectures for SIMD operator design. The remainder of this paper is organized as follows. In Section 2, the general structure of proposed SIMD unit is present. After that, section 3 provides detailed description of the proposed SIMD units. Then Section 4 describes the evaluation results under the context of a multimedia coprocessor. Finally, Section 5 gives the conclusion of the whole work.

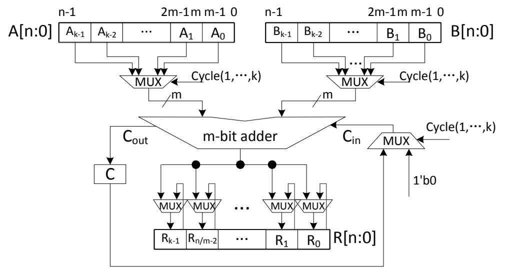
II. Generalized SIMD techniques

All modern microprocessors have now added multimedia instructions to their base instruction set architectures (ISAs). These various ISA extensions are in fact similar with few differences mainly in the types of operands, latencies of the individual instructions, the width of the data path, and in the memory management capabilities. They all need support concurrent operations on the foundation of the original hardware. The simplest realization method is to increase various subword computing hardware resources, and then choose the right result according to various subword modes. But because it consumes hardware resources and the power, this method is considered very little in the actual design. To reduce processor area it is often desirable to have SIMD operations share circuitry with non-SIMD operations. Following text will discuss the general SIMD implementation methods, including integer and floating-point operators for DLP exploration. As the memory and special multimedia instructions are related to specific system architectures, they are not discussed in this paper.

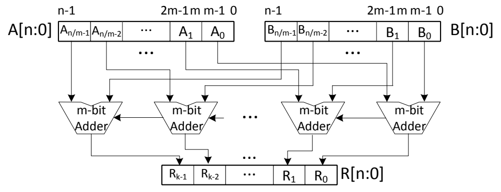
There are generally two schemes to implement SIMD arithmetic unit: low-precision based scheme and high-precision based scheme. The low-precision scheme uses several narrower units to support wider arithmetic operation while the high-precision scheme uses wider arithmetic unit to support multiple narrower parallel operations. The detailed implementation could be explained as follows.

A. Low-precision based scheme

The low-precision based scheme involves building wider SIMD elements out of several of the narrower SIMD elements and then combining the ultiple results together. This can be achieved by iteration or combination. The iteration method is to perform high precision operations by iteratively recalculating the data back through the same low precision unit over more than one cycle while the combination method is to perform high precision operations by consecutively “unrolling the loop” and then combining the results together [1]. For example, the adder implemented in the 2000-MOPS embedded RISC processor uses eight 8-bit adders to build a subword parallel adder [18].



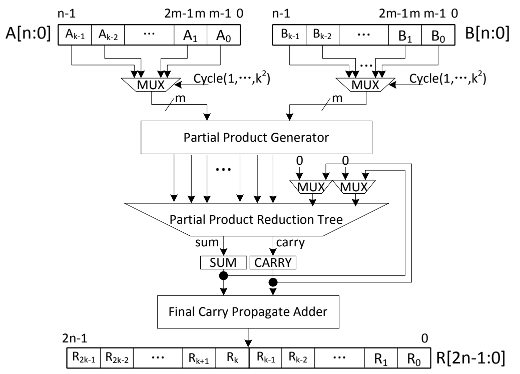
(a) m-bit adder supporting n-bit addition (iteration)



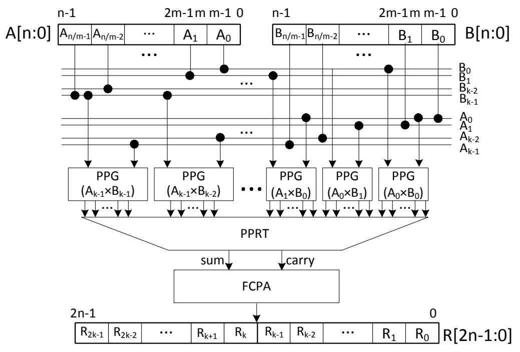
(b) k m-bit adders supporting n-bit addition (combination)

Fig. 1 Subword adder using low-precision based scheme

Figure 1 shows the structures of SIMD adder using low-precision based scheme. In Figure 1(a), a *m*-bit adder is used to perform *n*-bit addition with *n/m* cycles using iteration method, where $n>m$ and *n* can be divided by *m*. Each cycle *m* bits of source operands *A* and *B* are selected to the *m*-bit adder to perform addition operation and then computed *m*-bit result are stored into corresponding location of result operand *R*. To propagate the carry chain to higher part, the carry-out bit of adder should be stored in register and then send back to adder as carry-in bit in the next cycle. The iterative method is useful when SIMD arithmetic units are needed with a minimal amount of hardware. But this method also has its shortcomings that it requires pipeline stalls and it is at the cost of cycle times. Figure 1(b) illustrates the combination method implementation of *n*-bit addition using *n/m* *m*-bit adders. The carry-out bit of lower part adder is sent to the higher part adder to complete the results combination. Though combined adder requires less delay than iterative adder, it needs more area than iterative adder, and more delay than a scalar adder with the same width. This is because the critical path of carry chain in scalar adder can be optimized for performance globally while combined adder is optimized locally and carry chain is propagated in each *m*-bit adder.


A[n:0]
B[n:0]
MUX
Cycle(1,⋯,k²)
Partial Product Generator
Partial Product Reduction Tree
SUM
CARRY
Final Carry Propagate Adder
R[2n-1:0]

(a) m-bit multiplier supporting n-bit multiply using k^2 iteration



(b) k^2 m-bit multiplier supporting n-bit multiply using combination

Fig. 2 Subword multiplier using low-precision based scheme

Figure 2 shows the structure of SIMD multiplier using iteration and combination methods separately. Let n-bit $A=a_{k-1}a_{k-2}...a_1a_0$ and n-bit $B=b_{k-1}b_{k-2}...b_1b_0$, where a_x has m bits and $k=n/m$. Then

$A=a_{k-1}\times 2^{(k-1)m}+a_{k-2}\times 2^{(k-2)m}+...+a_1\times 2^m+a_0$ and $B=b_{k-1}\times b^{(k-1)m}+b_{k-2}\times 2^{(k-2)m}+...+b_1\times 2^m+b_0$.

So $A\times B$ can be expressed as:

$$A\times B=a_{k-1}b_{k-1}\times 2^{(2k-2)m}+...+a_{k-1}b_0\times 2^{k-1}+a_{k-2}b_{k-1}\times 2^{(2k-3)m}+...+a_{k-2}b_0\times 2^{k-2}$$

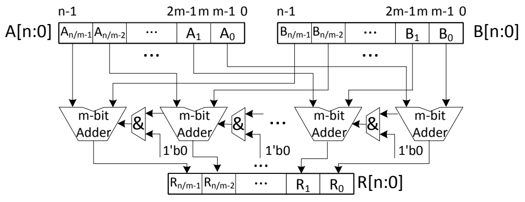
$$......+a_0b_{k-1}\times 2^{(k-1)m}+...+a_0b_0$$

From above equation, we can see that a $n\times n$ multiplier can be built out of k^2 $m\times m$ multipliers by generating k^2 $2m$ products which can then be added to form the $2n$ product. Using iteration method, not all k^2 $m\times m$ multipliers are necessarily required. As shown in Figure 2(a), a n-bit multiplier is built using only one m-bit multiplier with slight hardware modification. The partial product generator (PPG) does not need to change while the partial product reduction tree (PPRT) has two additional rows. To reduce the delay, each cycle the computed multiplication result is in carry-save representation. In the next cycle, the carry and sum vectors are fed back into the multiplier handled by the two additional rows. The final carry propagate adder (CPA) can also utilize iteration method to support $2n$-bit addition.

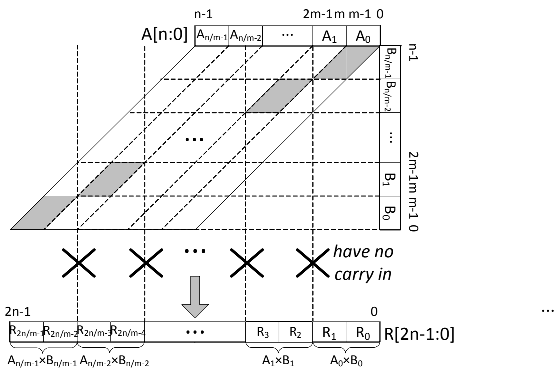
For combination method shown in Figure 2(b), k^2 $m\times m$ multipliers are used to support $n\times n$ multiplication. As mentioned in combined adder, the combination method has no obvious advantages over iteration or scalar implementation. But we can combine the iteration and combination methods to enable various area versus latency trade-offs.

B. High-precision based scheme

The high-precision based scheme uses existing high precision scalar tructure which is segmented based on the size of the smallest SIMD elements, so this method is also called segmentation scheme. It is the main source for implementing SIMD operators which retains existing scalar hardware structures while supporting SIMD operations with little area and delay overheads. This is also the initial design goal of SIMD extension. The key design feature of this method is that the final result comes from the same hardware, as opposed to selecting results from different hardware based on the SIMD mode with a multiplexer.



(a) n-bit adder supporting (n/m) m-bit addition in parallel



(b) n-bit multiplier supporting (n/m) m-bit multiplication in parallel

Fig. 3 High-precision based scheme

Figure 3(a) shows n-bit segmented adder capable of supporting k parallel m-bit addition operations. Each m-bit segment conditionally interacts or remains independent according to the SIMD precision mode. The hinge of this method is to applying certain mechanisms to manipulate the carry in signal's influence on higher position bits. In other words, when performing narrower SIMD addition, the carry in signals from lower element ill be ignored by higher ones to keep the addition operation within element boundary. The simplest way is to control the partition of SIMD element adders by inserting logic gates on the carry chain. But this carry truncation control mechanism will dramatically increase the gate delay since the inserted truncation logic is on the critical path. In contrast, carry elimination mechanism allows the carry signals to propagate to the higher bit position, and clear up the carries' effect by applying control logic on the higher bit position. Because these inserted logics are not all on the critical path, the increased gate delays of carry elimination mechanism is less than carry truncation one [10].

Carry elimination mechanism employs one bit carry select signal SC to perform element partition. In carry look-ahead adder (CLA), carry select adder (CSA) and parallel prefix adder (PPA), the p_i and s_i computation equations for the least significant position of element boundary are modified by combining the *SC* signal.

Figure 3(b) shows the *n*-bit segmented multiplier capable of supporting *k* parallel *m*-bit multiplication operations. To prevent the carry interactions between SIMD element operations, simdization of PPG, PPRT and final CPA should be careful designed. The PPG is aligned that, for a given vector mode, the partial products associated with each SIMD element do not overlap with the partial products associated with the other SIMD elements. This allows the use of the same partial product reduction tree employed in a regular scalar multiplier. Usually, Booth encoding is used to generate the partial products. Though Booth encoding reduces the number of partial products that have to be added, it adds complexity to handle carries across SIMD element boundaries in PPRT and CPA [6]. Another simdization method described in [15] does not use Booth encoding which also does not require detection and suppression of carries across SIMD element boundaries.

III. SIMD OPERATORS DESIGN

In this section, we will discuss the our operator designs for multimedia ISA including adder, multiplier, permutation unit, and floating point unit (FPU). These operator forms the basis for implementing various multimedia ISAs. The area-performance trade-offs are made to get a piratical solution for hardware operator implementation.

A. Adder FU

Study on the design of subword parallel adders is of great significance in the research field of microprocessors with multimedia extension. Subword parallel adder provides a very low cost form of small scale SIMD parallelism in a word-oriented scalar adder. In section 3, we have discussed the general SIMD techniques for adder operator. Figure 4 shows the proposed structure of subword parallel adder based on carry elimination mechanism. The 32-bit adder in the figure could be any type adders such as CLA, CSA and PPA. The CLA structure is used in our implementation. Since the equation (1) is applied only in the least significant bit position in each subword and all other bit positions are still computed by conventional logic equation, the increased gate delays for supporting subword parallelism is small.

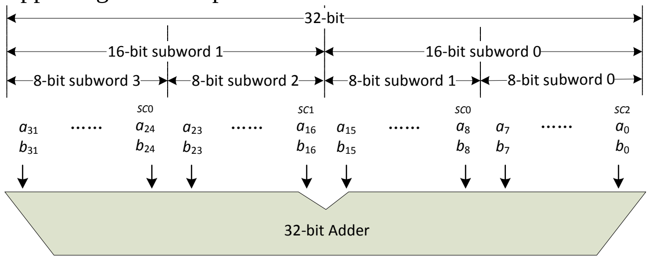


Fig. 4 Subword adder using carry elimination mechanism

Some enhanced parallel multimedia processing instructions can also be derived from subword adder. For example, parallel average ((A+B)/2) is obtained by adding them, then dividing the sum by two. This is equivalent to a right shift of one bit, after the addition. Other instructions can be parallel maximum, parallel compare and parallel shift and add, or conditional instructions like (A=B),(A>B) etc.

To better support multimedia application, many multimedia ISAs have saturation arithmetic instructions. Saturation is used to set a minimum and maximum value for operation. Additional hardware should be added which overflow flags are examined to determine the saturated subwords.

B. Multiplier FU

The subword multiplier also receives extensive research works such as multiplier architecture introduced in [15] and multiple-precision fixed-point vector multiply-accumulator proposed in [6]. We have classified them in previous section. It can be seen that the high precision based scheme is fast, but consumes much hardware resource and the low precision scheme can reduce the cost but can not satisfy the demand of multiple subword patterns. To design fast and area efficient subword multiplier, we can combine the two schemes, namely using low precision subword multipliers instead of usual multipliers to realize the higher precision multiplier. In this way, hardware cost can be reduced and multiple subword modes can be supported, which is a good trade-off between cost and performance. Figure 5 shows the structure of proposed 32-bit combined subword multiplier. It is consisted of two 16-bit subword multipliers also supporting 8-bit subword mode. 2-bit signal SM is introduce to select the subword modes.

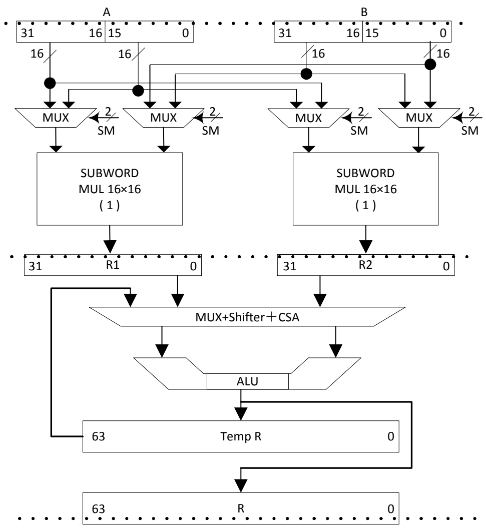


Fig. 5 Subword multiplier using combined scheme

The multiplication operands are input through register A and B. Both the signed and unsigned multiplications are supported. Multiplexers from M1 to M4 choose the corresponding high or low 16-bit operands to the two 16-bit subword multipliers under the control of the *SM* signal. Multiplexer M5 send the value temporarily store in the Temp register sends to the CSA for accumulation in the third cycle when the multiplier is in 32 subword mode.

The 32-bit multiplication is realized as illustrated in Figure 6. We donate A and B are 32-bit signed numbers, A0, B0 and A1, B1 are separately the high and low 16 bits of operands A and B. In the first cycle, A1×B0 (signed×unsigned) and A0×B1 (unsigned×signed) are computed by the two multiplier and stored in register R1 and R2; In the second cycle, A0×B0 (unsigned ×unsigned) and A1×B1 (signed×signed) are also computed and stored in register R1 and R2, at the same time, the results stored in R1 and R2 by the first cycle are accumulated to Register Temp through CPA (Carry Propagation Adder); In the third cycle, the values in register R1, R2 and Temp are accumulated by CSA and CPA so as to get the final 63-bit multiplication result.

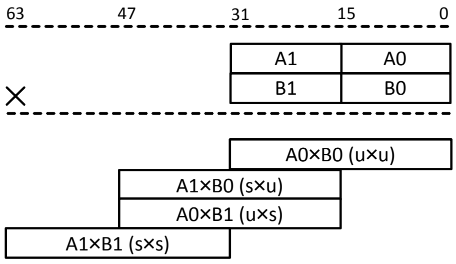


Fig. 6 32-bit signed multiplication

The implementation of 32×16 can be similar to 32×32, but it only needs two cycles. The first cycle calculates A0×B0 and A1×B0 and stores in R1 and R2, and the second cycle get the final result through CPA. The implementations of 16-bit and 8-bit subword multiplication are very easy because the results can be directly computed by the two 16-bit subword multipliers.

Based on the subword multiplier designs, we can get the implementation structure of multiply-adder unit (MAC). The accumulator is fed into the Wallace tree and the carry chain is also prevented on the subword boundaries.

C. Permutation FU

When using subword parallel instructions, a lot of permutation instructions should be used to facilitate the SIMD operations. These permutation instructions reorder the subwords or short vectors from one or two source registers. A good design for permutation instructions is essential for exploiting the full subword parallelism and superword parallelism.

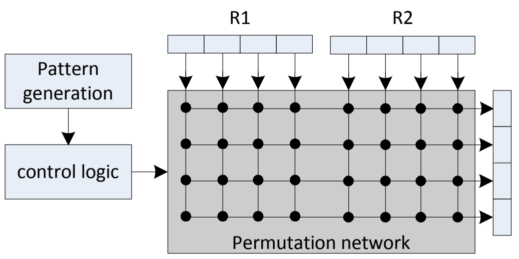


Fig. 7 Block diagram of permutation unit

Figure 7 illustrates the block diagram of proposed permutation unit. It consist of three component: permutation network, pattern generation, and pattern control logic. The permutation network is a crossbar implementation which is the core component of permutation unit. The pattern generation is used to decode the operation codes and generated the inner control signals for various permutation patterns. The pattern control logic is used to drive the permutation network to fulfill the specific permutations. This hardware implementation can support various permutation instructions. It mainly used to realize instruction *vperm*(R1, R2, P) which can supply extremely flexible data permutation functionality. It performs permutation operation which selects an arbitrary set of bytes from two input registers, R1 and R2, as indicated by the permutation pattern P which is also stored in a register. To decrease the mask bit for permutation, there are also some other special data permutation instructions with certain pattern such as pack and mix instructions [3].

D. Floating-point unit

In this paper, floating-point operators is implemented for IEEE floating-point format as specified in [4]. Compared with fixed-point arithmetic units, the sharing between different floating-point precision units are not so easy. Most recent processors implement the SIMD floating-point instructions by repeating floating-point units such as the vector FPU in a synergistic processor element of CELL processor [17] and SIMD FPU for BlueGene/L [16]. The work done in [14] extends a double-precision FPU to support single-precision interval addition/subtraction or multiplication. It proposes a method that how double-precision units can be split to compute two bounds of a single precision interval in parallel. [8] introduces a dual mode multiplier is presented, which uses a half-sized (i.e. 27×53) multiplication array to perform either a double-precision or two single-precision multiplication with that the double-precision operation needs two cycles. The multiplier proposed by [2] uses two double-precision multipliers plus additional hardware to perform one quadruple precision multiplication or two parallel double-precision multiplications. The multiple-precision iterative multiplier presented in [23] can perform two SP multiplies every cycle with a latency of two cycles, one DP multiply every two cycles with a latency of four cycles, or one EP multiply every three cycles with a latency of five cycles. It can also support also supports division, square-root, and transcendental functions. The multi-functional QP MAF unit presented in [24] supports a QP MAF, or two DP multiplication, or four SP multiplication, or a SP/DP dot-product operation. The superior performance of the this unit can be obtained in the execution of dot-product of vectors with two elements.

To get a better performance of both single- and double-precision and to save cost by sharing hardware between different precision, this paper proposes a double-precision FPU based on MAF unit to support two single-precision operations. When analyzing the realization structure of FPU in detail, we can discover that the sharing logic to reduce area mostly comes from adder, multiplier, LOP (Leading-One Predictor) etc. And using the techniques in subword adder [10] and multiplier design [6], these units only need more area about 5%. As MAF includes all these units which results in reasonable small area increase, and much more FPUs are implemented using MAF unit [16, 17], It is a wise choice to implement a SIMD FPU based on SIMD MAF Unit.

Using one double-precision unit to support two single-precision parallel operations, we need a modification for each module. The double-precision mantissa units are always two times wider than single-precision mantissa units, so it is possible to share most of the hardware by careful design. Figure 8 shows the hardware architecture of the proposed MAF unit. It is pipelined for a latency of three cycles, confirming to traditional three steps. For simplicity, the exponent computation block, the sign production block, and the exception handling block are not illustrated in the figure.

The first cycle of the MAF is used to compute multiplication of A and B to produce an intermediate product A×B and the alignment of C. The mantissa multiplier is extended to support either one 53-bit multiplication or two 24-bit multiplications using the techniques proposed by D. Tan et al. (2003). The alignment module completes mantissa shifting of operand C according to the exponent difference. Because using 161-bit shifter to do two parallel 75-bit shifting introduces much delay, the alignment module uses two shifters, where the high position single-precision shares with original double-precision shifter, and the low position single-precision shifter is added.

During the second cycle, the aligned C is added to the carry-save representation of A×B and the shift amount for normalization by means of LOP is determined. The 106-bit addition need not to increase additional control logic because two single-precision numbers are separated by one bit which may hold the addition carry, thus two single-precision additions would not affect each other.

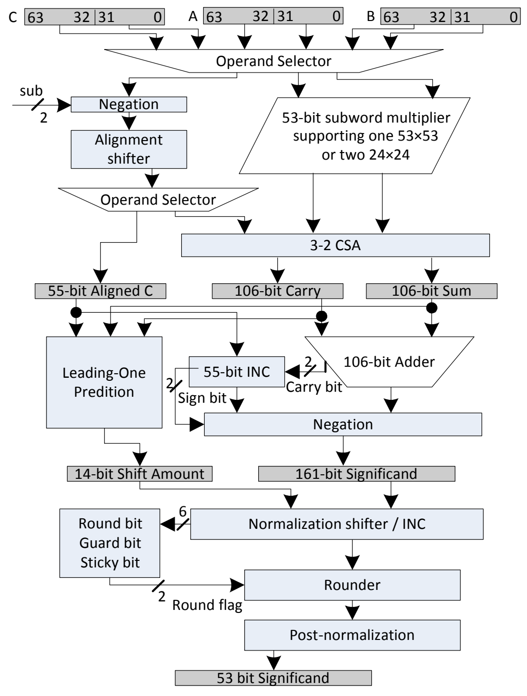


Fig. 8 Structure of SIMD MAF Unit

In the third cycle, the normalization shifting and rounding are performed. Two normalization shifters are also needed for speed and the Round, Guard, Sticky bits are sent to the unification logic to produce round flag whether adding one or not. After rounding, post-normalization is needed to get the final mantissas result.

IV. Evaluation

This section will give a detailed evaluation based on a piratical designed data parallel coprocessor for the proposed operator structures. First the system architecture of the coprocessor is described, and then its specific operators are introduced, finally the hardware cost and runtime performance for multimedia benchmarks are presented.

A. *System overview*

To test the hardware operator design for DLP exploration, we designed a multimedia co-processor based on the TTA (Transport Triggered Architecture) [21] integrating with the LEON-3 host processor. The block diagram of its architecture is shown in Figure 9(a). The co-processor is utilized for accelerating core processing of multimedia applications like data compression and graphics functions.

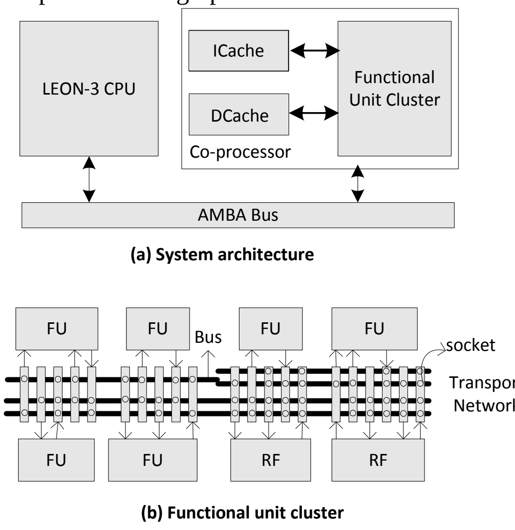


Fig. 9 System Overview

TTA can be viewed as a superset of traditional VLIW architecture. The major difference between TTA and traditional operation triggered architecture is the way that the operation is executed. Instead of triggering data transports, TTA operations occur as a side effect of data transports. The TTA-based architecture makes the coprocessor can integrate abundant arithmetic functional units in a very simple way, and gives flexible and fine-grained exploration of subword parallelism for compiler.

The structure of the functional unit cluster is organized as shown in Figure 9(b). There are an abundance of functional units (FU) and register files (RF) which are connected to the interconnection network by sockets. Every FU or RF has one or more operator registers, result registers but only one trigger register. Data being transferred to the trigger register will trigger the function unit to work. All operations, including load/store and branch/jump/link occur solely through moves. FUs operate concurrently and may incorporate pipelined execution. Thus, compared with the traditional VLIW architecture, TTA makes parallelism more fine-grained and flexible, which suits to the more efficient schedule in the instruction level as well as the data level.

In each cycle, the specified data transfer in the transport network can trigger the operations of these functional units. This allows for an extremely high utilization of the silicon area and suits for the different signal processing algorithms ranging from mobile communication systems to stream media applications. The computation is done by transferring values to the operand register and starting an operation implicitly via a move targeting the trigger register associated with the command.

Tab. 1 Arithmetic unit configuration

FU	Numbers	Description	Latency
ALU	4	Subword arithmetic, logic	1
MAC	4	Subword MAC	3/2
CMP	1	Integer, FP compare	1
FPU	1	2 way single, 1 way double	5

Table 1 shows the arithmetic unit configuration of the coprocessor, in which the latency of MAC is three cycles in 32-bit subword mode and two in other modes. To save the instruction encoding bits, The ALU and MAC operations are implemented in one functional unit as will be described in following subsection. The CMP Unit is a combined comparator which can perform double-/single-precision floating-point comparison or integer comparison and it does not support SIMD operations.

B. *Operator Implementation*

1) Typical operator

Every functional unit has its own unique internal structure, but their external interfaces are unified with one or more O-type registers, R-type registers and only one T-type register. If an operator supports two or more operations, then opcode should be specified.

The FUs are implemented using a so called virtual time lock (VTL) pipeline technique that the intermediate data in the FUs continues to the next stage on each clock cycle. This means that if the results of last operation are not read in time, they will be modified by the new operation. So whether the results of an operation are available depends on whether the FU is triggered again.

Figure 10 shows a typical Operator. Its pipeline has a latency of 3 clock cycles; this means that reading of the result register can be done 3 cycles after writing to the trigger register. The *bus_select* signal is used to select the right bus data and store it to the O-type or T-type register according to the load signal. Then the actual operation is executed by the FU. Three cycles later, the result of the operation is restored to the R-type register. When this result is needed, the control signal is used to send its value to the corresponding data bus and set other buses all zeros.

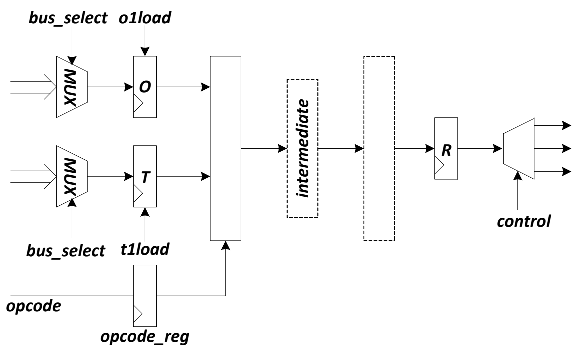


Fig. 10 System Overview

2) ALU Example

Figure 11 shows the structure of the proposed ALU design. It consists of five main components: internal registers, control logic, the adder-related unit, the multiplier-related unit, and permutation unit. The internal registers include configuration register *CONFIG*, operation encoding register *OPCODE*, operand register *X1*, *X2*, *X3*, result register *RESULT*, and status register *STATUS*. The program can specify the computation mode and exception handling via setting *CONFIG* register. The input register, *OPCODE*, is used to specify different operations performed by ALU. The result register *RESULT* is used to store the computation result. The status register, *STATUS*, indicates the whether the addition/substraction operations have overflow exception. The control logic *CTRL* decodes the operations according to *OPCODE* and determines the ways to handle exceptions when happen. The adder-related unit is responsible for various subword arithmetic and logic executions as specified in previous sections. The multiplier-related unit is used to fulfil the subword multiplication and MAC operations. The permutation unit is to perform the subword permutation instructions such as pack/unpack instructions.

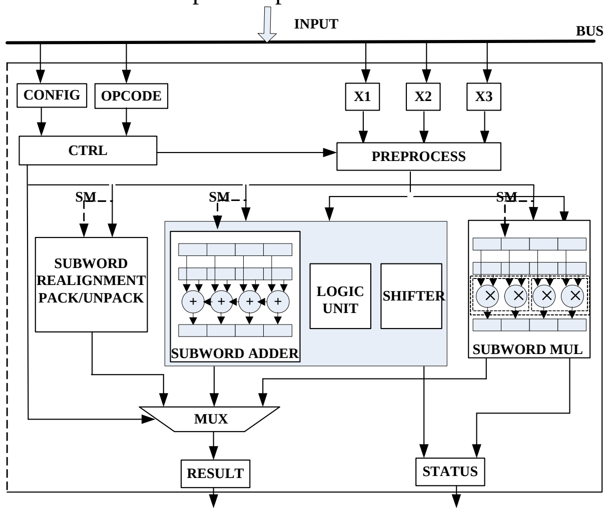


Fig. 11 Structure of porposed ALU

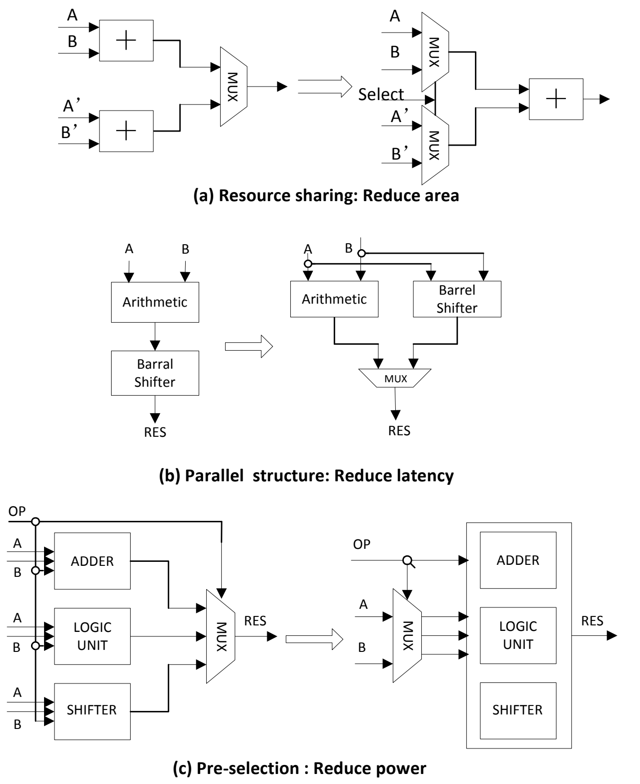

A
B
A'
B'
MUX
Select
(a) Resource sharing: Reduce area
Arithmetic
Barral Shifter
Barrel Shifter
MUX
RES
(b) Parallel structure: Reduce latency
OP
ADDER
LOGIC UNIT
SHIFTER
(c) Pre-selection : Reduce power

Fig. 12 Optimization techniques for ALU operator design

To get an highly optimized ALU targeting at high performance, low power and small area, we use some simple but effective methods to optimize the ALU design. Figure 12 shows the optimization schemes used in the proposed ALU design. The resource sharing method can decrease the hardware cost without performance overhead. Based on the principle of logical equivalence, it modifies the data processing mechanism to increase the utilization frequencies of complex circuits. Figure 12(a) illustrates that the AB datapath and A'B' datapath can sharing the adder under control signal *select*.

To reduce the latency, the conventional serial structure of ALU can be changed to parallel structure as shown in Figure 12(b). The serial structure would increase the execution delay for every instruction and its control description is relatively hard. In the proposed ALU, the parallel structure can not only improve the ALU speed, but also simplify the overall design.

The third technique is to reduce the power consumption of ALU based on that unnecessary circuit activities consume much power. The proposed ALU consists of many sub-modules and each time only one sub-module is used, so power is wasted if other unnecessary sub-modules works. We could perform the data selection before entering those submodules to break down their datapaths. So the states of these sub-module remain unchanged. This leads to reduce unnecessary signal switches resulting in low power. Figure 12(c) illustrates this data pre-selection technique.

C. Results

Utilizing subword implementation techniques described above, we realized all the proposed functional units and integrated them into the multimedia co-processor. They are modeled using structural level Verilog HDL codes. The correctness of various implementations is verified with extensive simulations. Then they are synthesized using Synopsys Design Compiler and standard cell library. It was fabricated in 0.18 μm 6-metal standard CMOS logic process. The chip occupies 4.8 mm^2 and its die photograph is shown in Figure 13. Correct operation has been observed up to 300 MHz and under this frequency, the dynamic power consumption is only 560 mW.

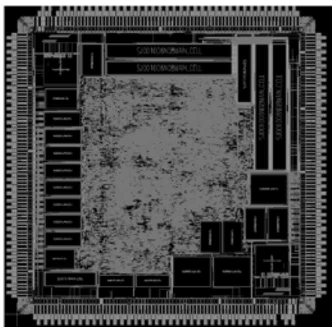


Fig. 13 Die Photograph

Tab. 2 Experimental Results of Arithmetic Units

	Parameters	ALU	MAC	CMP	FPU	Total
delay (ns)	Non-SIMD	2.12	2.60	1.76	2.55	2.60
	SIMD	2.55	2.42	-	2.79	2.79
area (mm^2)	Non-SIMD	63212	140316	21023	530669	1365804
	SIMD	74464	113305	-	615576	1387675

Table 2 gives the hardware experiment results of Arithmetic units. For comparison, the traditional arithmetic units are also designed. As can be seen from the table, the total SIMD arithmetic units only have roughly 2% more area and worst case delay that is 7% longer than the traditional arithmetic units. This is due to the fact that the area of proposed subword multiplier is even smaller than traditional multiplier and the SIMD FPU does not cause much more delay.

For this study, we use a set of common DSP and multimedia algorithms [20], including digital filters, fast Fourier transform (FFT), inverse discrete cosine transform (IDCT), and matrix arithmetic which is listed in Table 4. Matrix arithmetic is extremely important in 3D Graphics because they are used for several point transformations in the 3D world. FIR digital filters are used as general filtering in speech and audio processing, and linear predictive coding. Applications of IIR digital filters include audio equalization, speech compression, linear predictive coding and general filtering. FFT is used in many applications, including MPEG audio compression, radar processing, sonar processing, ADSL modems, and spectral analysis. DCT-based image coding is the basis for all the image and video compression standards, including JPEG image compression, MPEG video compression, and H.263.

Tab. 3 Benchmark Description

Name	Description
fft	128-point radix-4 FFT, 16-bit
mat_mul	4x4 matrix multiplication, 16-bit
maxidx	index of max values in the vector with 64 elements, 16-bit
idct	512x512 IDCT, 8-bit
fir	complex FIR filter, 40 coefficients, 16output samples, 16-bit
iir	IIR filter with 500 output samples, 16-bit

Using the data parallel coprocessor, we test the proposed operators both with SIMD instructions and non-SIMD instructions versions separately. This can shows the performance effectiveness of using data parallel operators. Figure 14 presents the instruction count comparison of SIMD and non-SIMD versions. The instruction code size reduction ranges from 19.6% to 35.8%. This is very beneficial for our coprocessor as the code size is always a big problem for VLIW-like ISA.

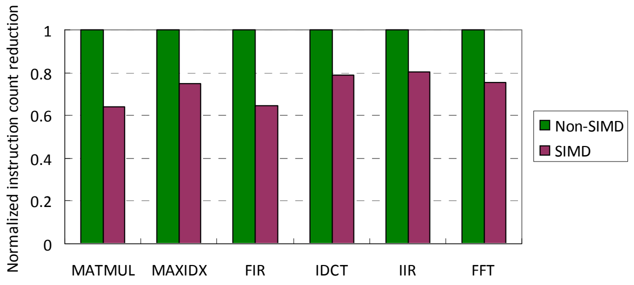


Fig. 14 Instruction count comparison

Figure 15 shows the performance comparison result obtained in this study. The speedup for SIMD instructions over scalar instructions range from 1.02 for IIR and 1.56 for MATMUL. IIR digital filter is the most difficult to vectorize because it involves the feedback path from the previously computed output samples. Hence, it has the least amount of data parallelism, resulting in the least speedup of 1.60. Matrix arithmetic, on the other hand, is straightforward to implement in AltiVec; therefore, we are able to obtain reasonable data parallelism, resulting in reasonable dynamic instruction count reduction and overall speedup. The experimental speedup for all the algorithms is less than its theoretical speedup. This confirms the negative effect of the overhead for data reorganization in packed data types.

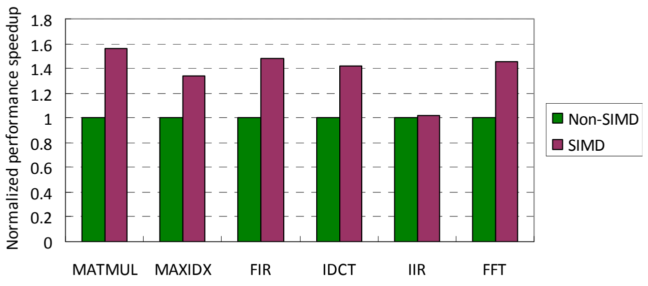


Fig. 15 Performance comparison in terms of execution cycle count

V. CONCLUSION

This paper presents a systematic solution to design high performance hardware operators for DLP exploration. The SIMD techniques are classified into two categories: low precision based scheme and high precision based scheme. To accommodate the best trade-off between hardware cost and performance, combined techniques are used to design integer operators and floating-point operators. Integrating all these operators into a system, a multimedia co-processor was designed and tested in 0.18μm standard CMOS logic process. It shows that the SIMD arithmetic unit cluster can improve the multimedia application by 2%~56% in the cost of a little increase in cycle delay and silicon area.

REFERENCES

[1] A. A. Farooqui and V. G. Oklobdzija, “General Data-Path Organization of a MAC Unit for VLSI Implementation of DSP Processors”, Proc. IEEE Int'l Symp. Circuits and Systems, pp. 260-263. 1998
[2] A. Akkas and M. J. Schulte, “A Quadruple Precision and Dual Double Precision Floating-Point Multiplier”, Proceedings of the 2003 Euromicro Symposium on Digital System Design, pp. 76-81. 2003
[3] Alex Peleg and UriWeiser, “MMX Technology Extension to Intel Architecture”, IEEE Micro, 16(4):42-50. 1996
[4] ANSI/IEEE standard 754-1985, IEEE Standard for Binary Floating-Point Arithmetic 1985.
[5] Corbal J., et al., “DLP+TLP processors for the next generation of media workloads”, Proc. 7th Intl.Symp.on HPCA. 2001.
[6] D. Tan, A. Danysh, and M. Liebelt, “Multiple-Precision Fixed-Point Vector Multiply-Accumulator using Shared Segmentation,” Comp. Arith., Proc. 16th IEEE Symp., pp. 12-19. 2003.
[7] Hwang K., “Computer Arithmetic: Principles, Architecture, and Design”, Wiley, New York 1979.
[8] G. Even, S. Mueller, and P.-M. Seidel, “A Dual Mode IEEE multiplier”, Proc of the 2nd IEEE Int. Conf. on Innovative Systems in Silicon, pp. 282-289. 1997.
[9] J. Bruguera, and T. Lang, “Leading-One Prediction with Concurrent Position Correction”, IEEE Trans. Computers, vol. 48, no. 10, pp. 298-305. 1999.
[10] M. Senthilvelan and M. J.Schulte, “A Flexible Arithmetic and Logic Unit for Multimedia Processing”, Advanced Signal Processing Algorithms, Architectures, and Implementations XIII, 2003.
[11] N. Burgess, “PAPA - Packed Arithmetic on a Prefix Adder for Multimedia Applications,” IEEE International Conference on Application-Speci¯c Systems, Architectures, and Processors (ASAP'02), pp. 197-207. 2002.
[12] N. Firasta, et al., “Intel AVX: New Frontiers In Performance Improvements And Energy Efficiency”, in Intel White paper, 2008
[13] R. E. Gonzalez, “Xtensa: A configurable and extensible processor”, IEEE Micro, 20(2) 2000.
[14] R. Kolla, et al., “The IAX Architecture : Interval Arithmetic Extension”, Technical Report 225, Universitat Wurzburg 1999.
[15] S. Krithivasan and MJ Schulte, “Multiplier Architectures for Media Processing,” Proc. 37th Asilomar Conf. Signals, Systems, and Computers, pp. 2193-2197. 2003.
[16] S. Chatterjee, L. R. Bachega, “Design and exploitation of a high-performance SIMD floating-point unit for Blue Gene/L”, IBM Journal of Research and Development, 49(2/3):377-392. 2005.
[17] S. M. Mueller, C. Jacobi, H. Oh, et al., “The Vector Floating-Point Unit in a Synergistic Processor Element of a CELL Processor”, Proceedings of the 17th IEEE Symposium on Computer Arithmetic, 2005.
[18] Suzuki, K. et al., “A 2000-MOPS embedded RISC processor with a Rambus DRAM controller”, IEEE J. Solid- State Circuits, Vol. 34, pp. 1010-1021. 1999.
[19] Swartzlander, E., “Computer Arithmetic”, IEEE Computer Society Press, Los Alamitos, Calif, 1990.
[20] TMS320C64x DSP Library Programmer's Reference, Texas Instruments Inc, 2002.

[21] V.A.Zivkovic, Ronald J. W. T. Tangelder, Hans G. Kerkhof, “Design and Test Space Exploration of Transport-Triggered Architectures”, pp.146-152, 2000.

[22] V. Lappalainen, P. Liuha, et al., “Current Research E®orts in Media ISA Development”, Technique Report, Nokia, 2000.

[23] D. Tan, C. E. Lemonds, M. J. Schulte, “Low-Power Multiple-Precision Iterative Floating-Point Multiplier with SIMD Support,” IEEE Transactions on Computers, vol. 58, no. 2, pp. 175-187, Feb. 2009

[24] M. Gok, M. M. ozbilen, “Multi-Functional Floating-Point MAF Designs with Dot Product Support,” Microelectronics Journal, vol. 39, pp. 30-43, January, 2008